NXP Semiconductors
Reference Manual

Document identifier: MCXAP144M240F61SRM
Rev. 2, 2025-11-17

MCX A255, A256, A265, and A266 Security Reference Manual

Supports MCX A255VPN, A255VLH, A255VLL, A255VLQ, A256VPN, A256VLH, A256VLL, A256VLQ, A265VPN, A265VLH, A265VLL, A265VLQ, A266VPN, A266VLH, A266VLL, and A266VLQ

Contents

Chapter 1
About This Manual

1.1 Audience

This reference manual (RM) is intended for system software, hardware developers, and applications programmers who need to develop products using this chip. It assumes that its users understand operating systems, microprocessor system design, and basic principles of software and hardware.

1.2 Organization

This manual has two main sets of chapters.

- Chapters in the first set contain information that applies to all components on the chip.
- Chapters in the second set are organized into functional groupings that detail particular areas of functionality.
  - Examples of these groupings are clocking, timers, and communication interfaces.
  - Each grouping includes chapters that provide a technical description of individual modules.

1.2.1 Attachments

This manual includes key information in the files attached to it. For example, memory map and I/O details. Use the content in these attachments in conjunction with this manual's content.

NOTE

Select the paperclip icon on the left side of the PDF window to see the list of attachments.

1.3 Module descriptions

Each module chapter has two main parts:

- The first section, *chip-specific [module name]* information, provides details such as the number of module instances on the chip and connections between that module and the other ones. Read this section *first* because its content is crucial for understanding the information in the other sections of the chapter.
- The subsequent sections provide general information about the module, including its signals, registers, and functional description.

The following figure shows you an example of this demarcation.

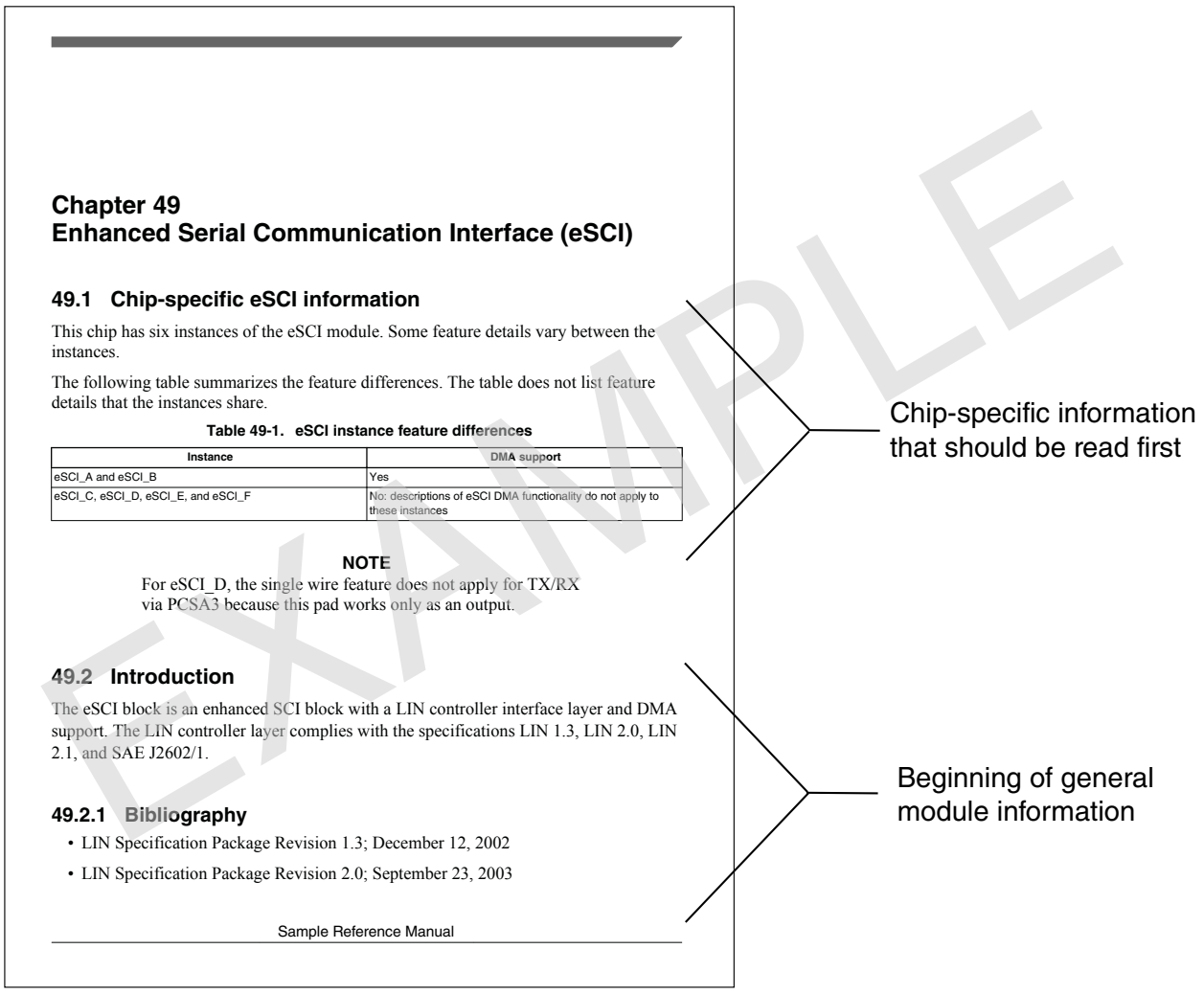


Figure 1. Example of chapter chip-specific information and general module information

1.3.1 Chip-specific information that clarifies content in the same chapter

The following figure shows an example of chip-specific information that clarifies general module information presented later in the chapter. In this case, the chip-specific register reset values supersede the reset values that appear in the register diagram.

Chapter 9
System Integration Unit Lite2 (SIUL2)

9.1 Chip-specific SIUL2 information

9.1.1 Feature configurations

In this device, the SIUL2_0 module instance does not support the following features described in the generic description:

- Interrupts
- DMA channels

9.1.2 Notes for IMCR

Out of reset, PA_00, PA_04, and PA_05 pads have JTAG input functionality selected by default. It should be disabled in the corresponding IMCR registers (IMCR61, IMCR60, and IMCR50 respectively) in order to use other functionality such as GPIO.

9.2 Introduction

9.2.1 Overview

The System Integration Unit Lite2 provides control over all the electrical pin controls and ports with 16 bits of bidirectional, general-purpose input and output signals. One of the most important functions of the SIUL2 is to enable the user to select the functions and electrical characteristics that appear on external device pins. It also controls the multiplexing of internal signals from one module to another and controls chip I/O. It supports as many as 32 external interrupts with trigger event configuration. The following figure is the block diagram of SIUL2 and its interfaces to other system components.

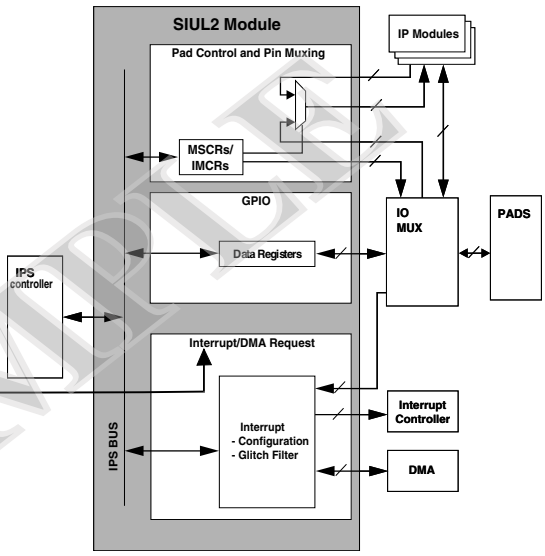


Figure 23. System Integration Unit Lite2 block diagram

This module provides dedicated pad control to general-purpose pads that can be configured as either inputs or outputs. The SIUL2 module provides registers that enable user software to read values from GPIO pads configured as inputs, and write values to GPIO pads configured as outputs:

- When configured as output, you can write to an internal register to control the state driven on the associated output pad.
- When configured as input, you can detect the state of the associated pad by reading the value from an internal register.
- When configured as input and output, the pad value can be read back, which can be used as a method of checking if the written value appeared on the pad.

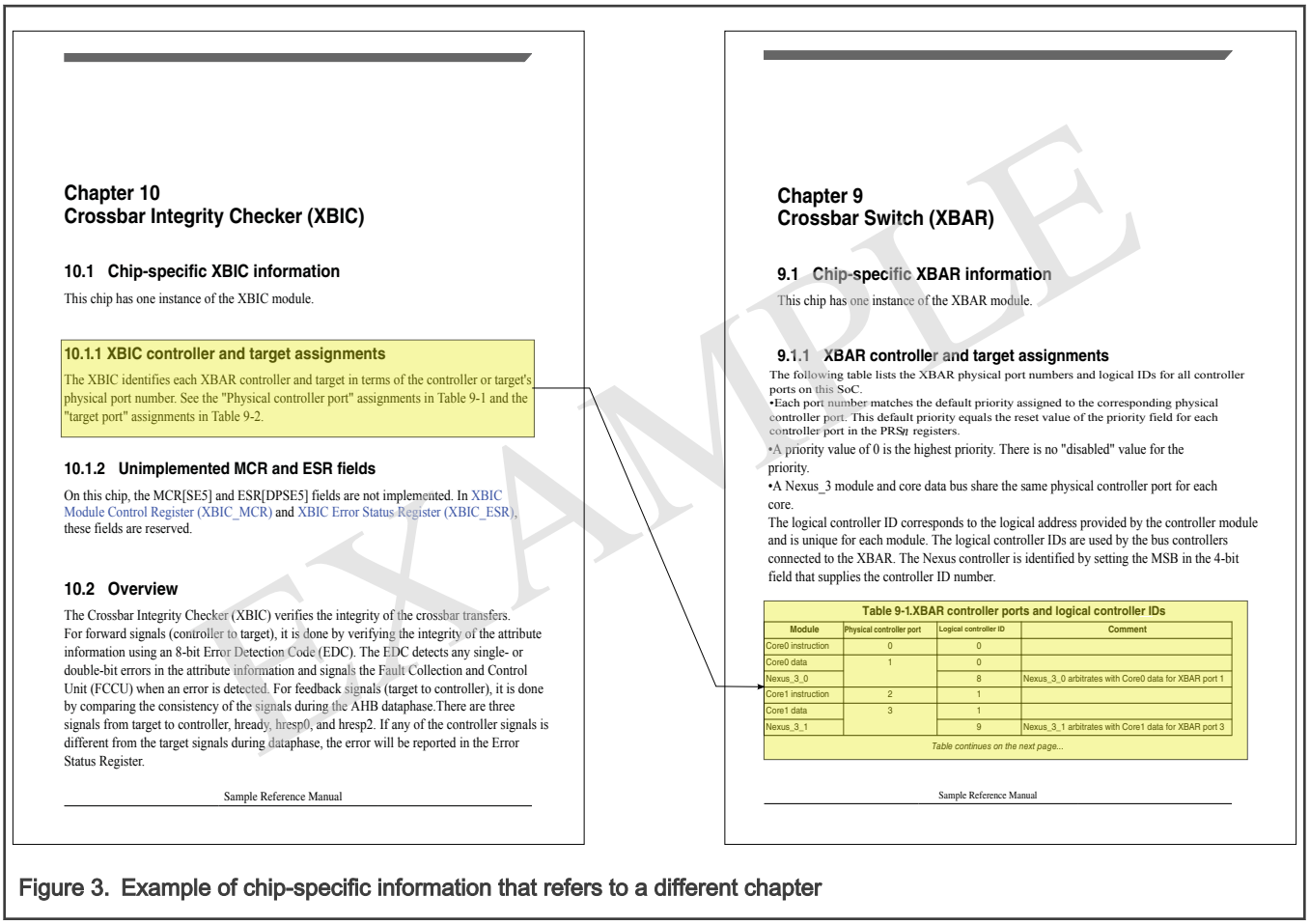
To assist software development, GPIO data registers can be accessed using various mechanisms. These differing mechanisms allow support for port access or for bit manipulation without the need to use read-modify-write operations:

- Access to two 16-bit ports in one access
- Read/write access to a single bit
- A 16-bit port write with a bit mask, using single 32-bit access.

Figure 2. Example of chip-specific information that clarifies content in the same chapter

1.3.2 Chip-specific information that refers to a different chapter

Related chip-specific information may be provided in different chapters of the manual. The following figure shows an example of two such connected pieces of information. In this case, read both before you proceed.



Chapter 10
Crossbar Integrity Checker (XBIC)

10.1 Chip-specific XBIC information

This chip has one instance of the XBIC module.

10.1.1 XBIC controller and target assignments

The XBIC identifies each XBAR controller and target in terms of the controller or target's physical port number. See the "Physical controller port" assignments in Table 9-1 and the "target port" assignments in Table 9-2.

10.1.2 Unimplemented MCR and ESR fields

On this chip, the MCR[SE5] and ESR[DPSE5] fields are not implemented. In XBIC Module Control Register (XBIC_MCR) and XBIC Error Status Register (XBIC_ESR), these fields are reserved.

10.2 Overview

The Crossbar Integrity Checker (XBIC) verifies the integrity of the crossbar transfers. For forward signals (controller to target), it is done by verifying the integrity of the attribute information using an 8-bit Error Detection Code (EDC). The EDC detects any single- or double-bit errors in the attribute information and signals the Fault Collection and Control Unit (FCCU) when an error is detected. For feedback signals (target to controller), it is done by comparing the consistency of the signals during the AHB dataphase.There are three signals from target to controller, hready, hresp0, and hresp2. If any of the controller signals is different from the target signals during dataphase, the error will be reported in the Error Status Register.

Sample Reference Manual

Chapter 9
Crossbar Switch (XBAR)

9.1 Chip-specific XBAR information

This chip has one instance of the XBAR module.

9.1.1 XBAR controller and target assignments

The following table lists the XBAR physical port numbers and logical IDs for all controller ports on this SoC.

- Each port number matches the default priority assigned to the corresponding physical controller port. This default priority equals the reset value of the priority field for each controller port in the PRSn registers.
- A priority value of 0 is the highest priority. There is no "disabled" value for the priority.
- A Nexus_3 module and core data bus share the same physical controller port for each core.

The logical controller ID corresponds to the logical address provided by the controller module and is unique for each module. The logical controller IDs are used by the bus controllers connected to the XBAR. The Nexus controller is identified by setting the MSB in the 4-bit field that supplies the controller ID number.

Table 9-1.XBAR controller ports and logical controller IDs

Module	Physical controller port	Logical controller ID	Comment
Core0 instruction	0	0	
Core0 data	1	0	
Nexus_3_0		8	Nexus_3_0 arbitrates with Core0 data for XBAR port 1
Core1 instruction	2	1	
Core1 data	3	1	
Nexus_3_1		9	Nexus_3_1 arbitrates with Core1 data for XBAR port 3

Table continues on the next page...

Sample Reference Manual

Figure 3. Example of chip-specific information that refers to a different chapter

1.4 Register descriptions

Module chapters present register information in the following:

- Memory maps, which contain:
  - An offset from the module's base address
  - The mnemonic and name of each register
  - The width of each register (in bits)
  - The reset value of each register
- Register figures
- Field-description tables
- Associated text

The following figure shows register figure conventions used throughout the manual.

Access type	Access description	DITA output	Effect of Write on Value	Readback Value
RW	Read/write	R / W: Mnemonic	Changes to Written Value	Current Value
RO	Read-only	R: Mnemonic / W:	No Effect	Current Value
RU	Reserved, unimplemented	R / W: Reserved	No Effect	Value Undefined
ROZ	Reserved, Read-only zero	R: 0 / W:	No Effect	Returns All Zero
ROO	Reserved, Read-only one	R: 1 / W:	No Effect	Returns All Ones
WO	Write-only	R: / W: Mnemonic	Changed to Written Value	Value Undefined
WOZ	Reserved, Write-only zero	R: / W: 0	Write Value Must Be All 0s.	Value Undefined
WOO	Reserved, Write-only one	R: / W: 1	Write Value Must Be All 1s.	Value Undefined
W0C	Write zero to clear	R: Mnemonic / W: w0c	If a Bit in the Written Value is a 0, the Corresponding Bit in the Field is Set to 0. Otherwise, the Field Bit is Not Affected.	Current Value
W1C	Write one to clear	R: Mnemonic / W: w1c	If a Bit in the Written Value is a 1, the Corresponding Bit in the Field is set to 0. Otherwise, the Field Bit is Not Affected.	Current Value
R2C	Read to clear		Value is cleared following the read operation	Current Value
ROWZ	Read-only, writes zero	R: Mnemonic / W: 0	Write Value Must Be All 0s	Current Value
ROWO	Read-only writes one	R: Mnemonic / W: 1	Write Value Must Be All 1s	Current Value
ROWU	Ready-only writes undefined	R: Mnemonic / W: —	Writes Operation Undefine	Current Value
WORZ	Write-only reads zero	R: 0 / W: Mnemonic	Changes to Written Value	Returns All Zero
WORO	Write-only reads one	R: 1 / W: Mnemonic	Changes to Written Value	Returns All Ones
WORU	Write-only reads undefined	R: — / W: Mnemonic	Changes to Written Value	Value Undefined

Figure 4. Register figure conventions

NOTE

Reset values of reserved locations documented in this manual are subject to change and must not be used for diagnostic purposes.

1.5 Conventions

1.5.1 Notes and cautions

Specific information is provided as part of notes and cautions throughout this manual.

NOTE

Emphasizes information that deserves extra attention.

CAUTION

Informs you of situations that could lead to highly undesirable outcomes—such as damage to the chip or irreversible malfunction.

1.5.2 Numbering systems

The following suffixes identify different numbering systems:

Table 1. Numbering systems

This suffix	Identifies a
b	Binary number. For example, the binary equivalent of the number 5 is mentioned as 101b. In some cases, *0b* is prefixed to binary numbers.
d	Decimal number. Decimal numbers are followed by this suffix only when there is a possibility of confusion. In general, decimal numbers are used without a suffix.
h	Hexadecimal number. For example, the hexadecimal equivalent of the number 60 is mentioned as 3Ch. In some cases, *0x* is prefixed to hexadecimal numbers.

1.5.3 Typographic notation

The following typographic notations are used throughout this document:

Table 2. Typographic notation

Example	Description
x and other italicized text	The italicized, lowercase *x* is used as a placeholder for replaceable numbers. In general, italicized text is used for titles of publications and for emphasis. Additionally, italics could be used for metasymbols in syntax descriptions. Plain lowercase letters are used as placeholders for single letters and numbers.
`code font`	Fixed-width font (such as Courier) used for code. It is used for a letter, word, or phrase that you want the user to type. For example, "Type `Read` and press Enter." This type of font is also used for instruction mnemonics, directives, symbols, subcommands, parameters, operators, computer-language elements, code listings, commands that appear in running text, and for sample code. Instruction mnemonics and directives in text and tables are mentioned in all caps; for example, BSR.
SR[SCM]	A mnemonic in square brackets represents the name of a register field. This example refers to the Scaling Mode (SCM) field in the Status Register (SR).
REVNO[6:4], XAD[7:0]	Numbers in brackets that are separated by a colon represent either: • A subset of a register's named field

Table continues on the next page...

Table 2. Typographic notation (continued)

Example	Description
	For example, REVNO[6:4] refers to bits 6-4 that are part of the COREREV field occupying bits 6-0 of the REVNO register. • A continuous range of individual signals of a bus For example, XAD[7:0] refers to signals 7-0 of the XAD bus.
MOD.REG	A period separates the elements of a hierarchy: subsystem.module.register. For example: • SWT.TO means that the TO register is located in the SWT module. • SMU.XRDC.CR means that the CR register is located in the XRDC module within the SMU subsystem.

1.5.4 Special terms

The following terms have special meanings.

Table 3. Special terms

Term	Meaning
Asserted	Refers to the state of a signal as follows: • An active-high signal is asserted when high (1). • An active-low signal is asserted when low (0).
Deasserted	Refers to the state of a signal as follows: • An active-high signal is deasserted when low (0). • An active-low signal is deasserted when high (1). In some cases, deasserted signals are described as *negated*.
Reserved	Refers to memory space, register, field, or programming setting. Writes to a reserved location can result in unpredictable functionality or behavior. You must: • Before writing to a location which contain reserved bits user must make sure the write operation will write the reserved bit with value specified as the reset value in NXP reference manual. • Consider undefined locations in memory to be reserved as reset value 0 shall be assumed. You might get a BERR(transfer error) response on access to undefined locations in memory. • If user reads data from memory area containing reserved bit, the value of reserved bits should be ignored and not used for any functional purposes. **NOTE** BootROM could modify the reserved bit values after reset. Please refer to the BootROM settings attachment.
Write 1 to clear (w1c)	Refers to the access type of a register field that is used to clear the field by writing the value 1 to it.
Undefined (u)	Refers to undefined reset values

1.6 Editorial changes

Each new release of this document includes editorial improvements such as:

- Spelling
- Grammar
- Punctuation
- Voice
- Tense
- Capitalization
- Formatting
- Presentation
- Navigation

Chapter 2
Introduction

2.1 Overview

This group of products expand the MCX Arm® Cortex®-M33 product offerings with multiple high-speed connectivity, operating up to 180 MHz or 240 Mhz, serial peripherals, timers, analog and low power consumption.

2.2 Target applications

This device has following target applications:

Industrial

- Energy Storage and Management System
- Smart Metering
- Factory Automation
- Industrial HMI
- Mobile Robotics Ecosystem
- Motion Control and Robotics
- Motor Drives
- Brushless DC Motor (BLDC) Control
- Permanent Magnet Synchronous Motor (PMSM) Control

Smart Home

- Home Control Panel
- Major Home Appliances
- Robotic Appliance
- Smart Speaker
- Soundbar
- Gaming Accessories
- Smart Lighting
- Smart Power Socket and Light Switch

2.3 Block diagram

The following figure shows a top-level organization of the modules within the chip organized by functional category.

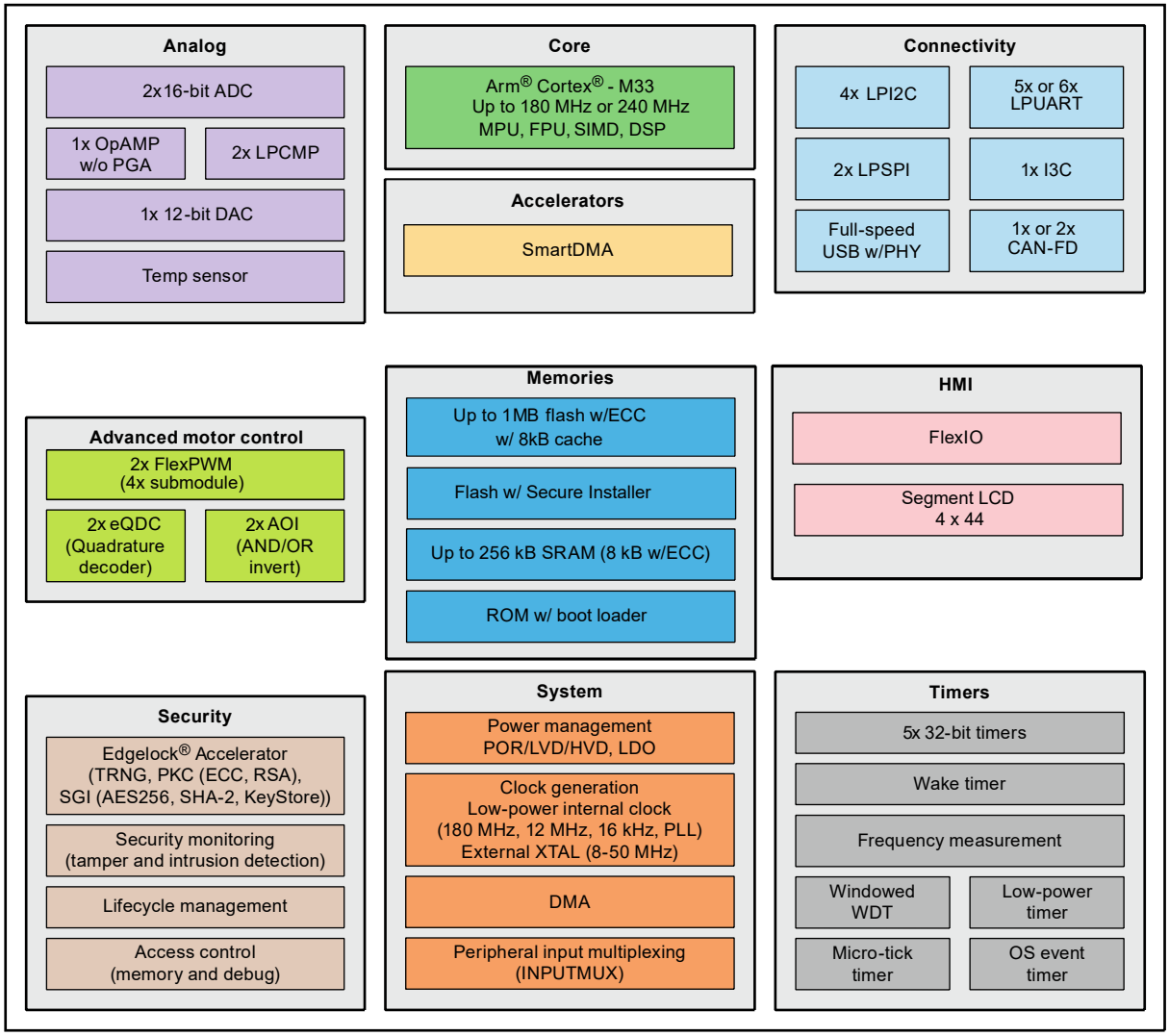


Figure 5. Features block diagram

2.4 Features

This group of products offer the following features:

- Core
  - Arm 32-bit Cortex-M33 CPU, with FPU and DSP extension instruction set and MPU, no Trust Zone. 45 MHz at Mid Drive mode (1.0 V), 180/240 MHz at Over Drive mode (1.2 V)
  - Nested Vectored Interrupt Controller
  - Clock gating for core processor
  - Multilayer AHB Bus Matrix
  - SmartDMA, co-processor for applications such as parallel camera interface and keypad scanning
- Memories
  - 1 MB Flash array implemented as single array. The smallest programming phrase should be 16 bytes.
    - 128-bit + 9-bit ECC, 8 KB sector size

    - 32 KB IFR0 + 8 KB IFR1 for Implicit-protected Flash Region(IFR)
  - Flash Memory Controller
    - Line buffer
    - Prefetch buffer
    - Swap
    - 5 wait states at 240 MHz (OD mode), 4 wait state at 180 MHz (OD mode), 1 wait state at 45 MHz (MD mode)
  - Memory Block Checker (MBC)
    - Protect flash read, write and execute permission.
    - Main array granularity is 16 KB
    - IFR0 granularity is 8 KB.
    - IFR1 granularity is 4 KB
  - RAM
    - 8 KB LPCAC to support on-chip Flash caching. 8 KB LPCAC RAM can be used as Code SRAM when LPCAC is disabled.
    - 16 KB Code SRAM
    - Up to 256 KB System SRAM (8 KB with ECC)
  - System ROM
- Timers
  - 5x 32-bit standard general-purpose asynchronous timers/counters, which support up to four capture inputs and four compare outputs, PWM mode, and external count input. Specific timer events can be selected to generate DMA requests.
  - Low power timer
  - Frequency measurement timer
  - Windowed watchdog timer
  - Wake timer
  - Micro-tick timer (UTICK) running from the watchdog oscillator can be used to wake-up the device from sleep and deep-sleep modes. Includes 4 capture registers with pin inputs.
  - OS event timer
    - 42-bit free running OS Timer as continuous time-base for the system.
- Advanced Motor Control
  - 2x FlexPWM
    - 4 sub-modules per each FlexPWM, providing 16 complementary outputs of PWM (no Nanoedge module)
    - 4 fault inputs are supported
  - 2x Quadrature Decoder (eQDC)
  - 2x AOI (AND/OR/Invert) module
    - Implements four events output. Each event output represents a user-programmed combinational Boolean function based on four event inputs.
    - Four event inputs are selected from 16 input
- Security

    - Security Monitoring
        - Code Watchdog for code flow integrity checking
        - 6x Passive anti tamper pin detect
        - GLIKEY enforces security checks before allowing a write to security-sensitive register.
    - Implicit-protected Flash Region (IFR)
    - 128-bit Universal Unique Identifier (UUID) per device in accordance with IETF's RFC4122 version5 specification
    - Device lifecycle management
    - EdgeLock Accelerator
        - TRNG
        - PKC (Public Key Cryptography), supports ECC and RSA encryption and decryption
        - SGI (Secure Generic Interface), provides AES256, SHA-2 and key generation/derivation
        - Secure key store with key usage policies (protection of platform integrity, manufacturing and applications keys)
- Communication Interfaces for Connectivity
    - 4x LPI2C, 2x LPSPI, up to 6x LPUART
    - Up to 1xI3C
    - USB Full-speed (Device/Host) with on-chip FS PHY
    - Up to 2x FlexCAN with FD
    - FlexIO
- HMI
    - 4 x 44 segment LCD interface
- Analog
    - 1x 12-bit DAC
    - 2x 16-bit ADC
        - up to 3.2Msps in 16-bit mode, and 4Msps in 12-bit mode
        - up to 43 ADC Input channels (depending on the package)
        - One integrated temperature sensor per ADC
    - 2x High-speed Comparators with 8 input pins and 8-bit DAC as internal reference
        - 1x LPCMP is functional down to DPD mode
    - 1x OpAmp without PGA
- Operating characteristics
    - Operating voltage: 1.71 to 3.6 V
    - Temperature range: -40 to 125 °C
- Debug
    - Minimal debug – 4 breakpoint and 2 watch point, no ETM
    - Debug mailbox for allow tools consistency during power up
    - SWD/JTAG
- Packages
    - LQFP144

- LQFP100
- LQFP64
- BGA169

See the device data sheet for details on packages.

2.5 Functional overview

The following table shows the chip modules organized by functional category.

Table 4. Module functional categories

Module category	Description
Core	• The Arm Cortex-M33 is a member of the Cortex-M Series of processors targeting microcontroller cores focused on cost sensitive, deterministic, and interrupt driven environments. The Cortex-M33 processor is based on the Armv8-M Architecture and ThumbR-2 ISA and is upward compatible with the Cortex-M7, M4, M3, M1, and M0/M0+. • Smart DMA Controller (SmartDMA)
System	• Debug Mailbox • AHB Cross-Bar Switch (AXBS) Lite • Enhanced Direct Memory Access (8-channel eDMA) • Wake-Up Unit (WUU) • Peripheral Input Mux (INPUTMUX) • Error Injection Module (EIM) • Error Reporting Module (ERM) • System Power Controller (SPC) • System Controller • RMC (part of CMC)
Memory	• Flash Memory Controller (FMC) • Flash Management Unit (FMU) • ROM-BOOT and ROM-CODE • Static Random Access Memory (SRAM) • AHB Low Power Cache Controller (LPCAC)
Clock	• VBAT Wrapper — Free Running Oscillator - 16 K (FRO16K) • System Clock Generator (SCG) — Crystal Oscillator - System (OSC_SYS) — Free Running Oscillator - 180 M (FRO180M)

Table continues on the next page...

Table 4. Module functional categories (continued)

Module category	Description
	— Free Running Oscillator - 12 M (FRO12M) — PLL
Security	• Cyclic Redundancy Check (CRC) • Code Watchdog • Glikey • Memory Block Checker (MBC) • Secure Generic Interface (SGI) • Public-Key Crypto Coprocessor (PKC) • True Random Generator (TRNG) • Digital Tamper (TDET)
Timer	• Quadrature Decoder (ENC) • Micro-Tick (UTICK) Timer • OS Event Timer (OSTIMER) • Low-power Timer (LPTMR) • Standard Counter/Timer (CTIMER) • Windowed Watchdog Timer (WWDT) • Frequency Measurement (FREQME) • Wake Timer • Real-time counter (RTC)
Communication	• Improved Inter-Integrated Circuit (I3C) • Universal Serial Bus - Full Speed (USBFS) and Transceiver • USB FS Physical Layer Interface (USBFS PHY) • Low-Power UART (LPUART) • Low-Power Serial Peripheral Interface (LPSPI) • Low-Power I2C (LPI2C) • Flexible Data Rate CAN (FlexCAN) • Flexible I/O (FLEXIO)
Human Machine Interface (HMI)	• General Purpose Input/Output (GPIO) • Port Control (PORT)
Analog	• 12-bit Analog-to-Digital Converter (ADC) with 4 Msps conversion rate and 3.2Msps in 16-bit mode • 12-bit DAC

Table continues on the next page...

Table 4. Module functional categories (continued)

Module category	Description
	• Operational Amplifier (OpAmp) • Low Power analog Comparator (LPCMP)

2.5.1 Core

The following core modules are available on this chip.

Table 5. Core modules

Module	Description
CPU	Is an Arm Cortex-M33 processor that runs at a frequency of up to 180 MHz.
Nested Vectored Interrupt Controller (NVIC)	The Armv8M exception model and Nested-Vectored Interrupt Controller (NVIC) implement a relocatable vector table supporting external interrupts, a single non-maskable interrupt (NMI), and priority levels.
System Tick Timer (SysTick)	See the Armv8M Architecture Reference Manual for more information about this system timer.

2.5.2 Clock

The following clock modules are available on this chip.

Table 6. Clock sources

Module	Description
Free Running Oscillator - 16 K (FRO16K)	Is an ultra-low power internal 16.384 kHz clock source. It is functional down to Deep Power Down (DPD) mode. The FRO16K is trimmed to +/- 6% accuracy over the entire voltage and temperature range.
System Clock Generator (SCG)	Provides the user with access to the configuration control registers for the system level clock sources. See *SCG* chapter in *MCX A255, and A256 Reference Manual*.
Crystal Oscillator - System (OSC_SYS)	Generates, in conjunction with an external crystal or resonator, a reference clock for the chip.
Free Running Oscillator - 180 MHz (FRO180M)	The free running oscillator can generate 180 MHz for use by the chip.
Free Running Oscillator - 12 MHz (FRO12M)	Is an internal clock source that generates a 12 MHz frequency for use by the chip.

2.5.3 Communication

The following communication modules are available on this chip.

Table 7. Communication modules

Module	Description
Low Power Inter-Integrated Circuit (LPI2C)	Allows communication between a number of devices. Also supports the System Management Bus (SMBus) Specification, version 2.
Improved Inter-Integrated Circuit (I3C)	Is an extension of the I2C bus protocol supporting higher speeds.
Low Power Serial Peripheral Interface (LPSPI)	Synchronous serial bus for communication to an external device.
Low Power Universal Asynchronous Receive/ Transmit (LPUART)	Asynchronous serial bus communication interface with programmable 8- or 9-bit data format.
Universal Serial Bus Full Speed Device Controller (USBFS)	In Device mode, the USB FS subsystem working together with the SCG module can use the FIRC to generate a 48 MHz USB controller clock tuned using the incoming Host USB data for crystal-less operation.
Flexible Data Rate CAN (FlexCAN)	This module is a communication controller implementing the CAN protocol according to the ISO 11898-1:2015 standard and CAN 2.0 B protocol specifications.
Flexible I/O (FLEXIO)	Flexible I/O (FLEXIO) is a highly configurable module providing a wide range of functionality, including: • Emulation of various serial or parallel communication protocols • Flexible 16-bit timers with support for various trigger, reset, enable, and disable conditions • Programmable logic blocks which allow the implementation of digital logic functions on-chip and configurable interaction of internal and external modules • Programmable state machine for offloading basic system control functions from the CPU
SmartDMA	Supports unique reduced instruction sets and performs event- and IO-driven handling to offload the work from the Arm processor.

2.5.4 Debug

The following debug modules are available on this chip.

Table 8. Debug modules

Module	Description
Data Watchpoint and Trace (DWT)	Is a generic name for modules that allow debug access of the Cortex-M33. The DWT comprises of the Debug Watchpoint and Trace (DWT) module and the Flash Patch and Breakpoint (FPB) unit.
Debug Access Port (DAP)	Enables real-time access to the chip registers from an external debugger without halting the processor cores.
Instruction Trace Macrocell (ITM)	Provides a memory-mapped register interface that applications can use to write logging or event words for profiling software.

Table continues on the next page...

Table 8. Debug modules (continued)

Module	Description
Joint Test Action Group (JTAG)	Implements serial communication protocol for communication with the Test Access Port (TAP).
Serial Wire Debug (SWD)	Is a serial communication interface used for debugging devices with multiple cores while only requiring a single external interface.
Trace Port Interface Unit (TPIU)	Acts as a bridge between the on-chip trace data from the modules such as the ITM, which have separate system IDs, to the external world.

2.5.5 Memory

The following memory modules are available on this chip.

Table 9. Memory modules

Module	Description
Flash Memory Controller (FMC)	Is a programmable flash memory — non-volatile flash memory that can store executable program code or data. See *FMC* chapter in *MCX A255, and A256 Reference Manual*.
Flash Management Unit (FMU)	Manages the interface between the chip and the on-chip flash memory. See *FMU* chapter in *MCX A255, and A256 Reference Manual*.
Static Random Access Memory (SRAM)	Internal system SRAM memory. Each individual block of SRAM can be configured to be retained in low-power modes.
AHB Low Power Cache Controller (LPCAC)	Is a processor-local level 1 (L1) bus cache controller for use with cores using AMBA-AHB input/output buses. See *LPCAC* chapter in *MCX A255, and A256 Reference Manual*.

2.5.6 System

The following system modules are available on this chip.

Table 10. System modules

Module	Description
Peripheral Bridge(PBRIDGE)	Converts an AMBA AHB interface to a peripheral interface that allows the Peripheral Bridge (PBRIDGE) controller to interface to multiple peripherals. This device has two peripheral bridges.
System Controller (SYSCON)	Provides controls and configurations of the system and peripherals for the multiple functions.
Core Mode Controller (CMC)	The CMC provides control and protection on entry and exit to each power mode, control for the System Power Controller (SPC), and reset entry and exit for the complete device.

Table continues on the next page...

Table 10. System modules (continued)

Module	Description
	See *CMC* chapter in *MCX A255, and A256 Reference Manual*.
Enhanced Direct Memory Access (eDMA)	Performs source/destination address calculations and data-movement operations. Capable of performing complex data transfers with minimal intervention from a host processor. See *eDMA* chapter in *MCX A255, and A256 Reference Manual*.
Wake-up Unit (WUU)	Allows selection of external pins and internal modules as interrupt wake-up sources from Deep Sleep and Power Down modes. See *WUU* chapter in *MCX A255, and A256 Reference Manual*.
Peripheral Input Multiplexing (INPUTMUX)	Allows the trigger output of one peripheral to be connected to the trigger input of a second peripheral. See *INPUTMUX* chapter in *MCX A255, and A256 Reference Manual*.
Error Injection Module (EIM)	Provides a method for diagnostic coverage of internal memories (for example, system RAM, cache RAMs, and peripheral memories). See *EIM* chapter in *MCX A255, and A256 Reference Manual*.
Error Reporting Module (ERM)	Provides information and optional interrupt notification on memory error events associated with Error correction code (ECC) and parity. See *ERM* chapter in *MCX A255, and A256 Reference Manual*.
System Power Controller (SPC)	Provides control over the operation and configuration of the system power generation modules to optimize power consumption for the level of functionality needed. See *SPC* chapter in *MCX A255, and A256 Reference Manual*.
Low Drop Out Regulator - Core (LDO_CORE)	A voltage regulator for generating the core voltage for the chip.
RAM_RET_LDO	A voltage regulator for generating the voltage for SRAM retention.

2.5.7 Security

The following security modules are available on this chip.

Table 11. Security modules

Module	Description
Cyclic Redundancy Check (CRC)	Provides error detection for all single, double, and many multi-bit errors. See *CRC* chapter in *MCX A255, and A256 Reference Manual*.
GLIKEY	It provides a mechanism to safely access security-sensitive registers.

Table continues on the next page...

Table 11. Security modules (continued)

Module	Description
Memory Block Checker (MBC)	Provides domain-based access control for all system bus references targeted to on-chip internal memories and slave peripherals. It's a part of TRDC module. See *MBC* chapter in *MCX A255, and A256 Reference Manual*.

2.5.8 Timer

The following timer modules are available on this chip.

Table 12. Timer modules

Module	Description
FlexPWM	Generates various switching patterns, including highly sophisticated waveforms. Controls different Switched Mode Power Supplies (SMPS) topologies. See *FlexPWM* chapter in *MCX A255, and A256 Reference Manual*.
Wake Timer	Provides time keeping and calendaring functions and additionally provides protection against spurious memory/register updates and battery operation. See *Wake Timer* chapter in *MCX A255, and A256 Reference Manual*.
Quadrature Decoder	Interfaces to position/speed sensors that are used in industrial motor control applications. See *Quadrature Decoder* chapter in *MCX A255, and A256 Reference Manual*.
Micro-Tick (UTICK) Timer	Is a 31-bit timer that provides a fixed time interval between interrupts. See *UTICK* chapter in *MCX A255, and A256 Reference Manual*.
OS Event Timer (OSTIMER)	Is a 42-bit Gray code counter. See *OSTIMER* chapter in *MCX A255, and A256 Reference Manual*.
Low-power Timer (LPTMR)	Is a 16-bit timer or pulse counter with compare feature. See *LPTMR* chapter in *MCX A255, and A256 Reference Manual*.
Standard Counter/Timer (CTimer)	Each Counter/timer is designed to count cycles of the CTIMER function clock. See *CTimer* chapter in *MCX A255, and A256 Reference Manual*.
Windowed Watchdog Timer (WWDT)	Helps reset or interrupt an erroneous microcontroller within a programmable time. See *WWDT* chapter in *MCX A255, and A256 Reference Manual*.

Table continues on the next page...

Table 12. Timer modules (continued)

Module	Description
Frequency Measurement	Provides high-accuracy frequency measurement function for on-chip and off-chip clocks. See *Frequency Measurement* chapter in *MCX A255, and A256 Reference Manual*.

2.5.9 Human Machine Interface (HMI)

The following HMI modules are available on this chip.

Table 13. HMI modules

Module	Description
General Purpose Input/Output (GPIO)	All GPIO pins support interrupt and DMA request generation. See *GPIO* chapter in *MCX A255, and A256 Reference Manual*.
Port Control (PORT)	Provides support for pad control functions. See *PORT* chapter in *MCX A255, and A256 Reference Manual*.

2.5.10 Analog

The following analog modules are available on this chip.

Table 14. Analog modules

Module	Description
Analog-to-Digital Converter (ADC)	Successive approximation ADC designed for operation within an integrated microcontroller system-on-chip. See *ADC* chapter in *MCX A255, and A256 Reference Manual*.
Low Power Comparator (LPCMP)	Provides a circuit for comparing two analog input voltages. See *LPCMP* chapter in *MCX A255, and A256 Reference Manual*.
LPDAC	The Low Power Digital-to-Analog Converter(LPDAC) is a low power, 12-bit general-purpose digital-to-analog converter. The output of the LPDAC can be placed on an external pin or set as one of the inputs to the analog comparator or ADC. See *LPDAC* chapter in *MCX A255, and A256 Reference Manual*.
Operational Amplifier (OpAmp)	The OpAmp is a pure voltage amplifier without any built-in programmable gain stage. User needs to implement external feedback network to achieve specific inverting or non-inverting gain amplification. This module is applicable to the signal processing stage before SARADC.

Chapter 3
Core Overview

3.1 Introduction

This section covers the core modules included in this chip.

3.2 Cortex-M33 Code and System buses

This device has one Arm Cortex-M33 processor core, with Floating Point Unit (FPU), Memory Protection Unit (MPU) and without TrustZone-M.

The core implements a modified Harvard memory architecture using two 32-bit bus interfaces: the Code and System buses. The bus interfaces are activated by address range and can include both instruction fetches and operand data references on a given bus port.(A traditional Harvard architecture strictly separates instruction fetches and operand data references onto specific bus ports regardless of access address.) The Code bus is typically used for instruction fetching and data accesses of PC-relative data, while the system bus is typically used for operand data references to the on- and off-chip memories and peripheral accesses. The bus structure fully supports concurrent instruction fetch and data accesses, but the Cortex-M33 implementations can generate both types of references on each bus.

NOTE

It is recommended that performance critical code be located such that it fetches from the Code bus interface as defined by addresses < 0x2000_0000.

3.2.1 Code Bus access

Code Bus accesses are routed to the Code Cache Controller; code bus of the core goes to LPCAC. This controller then processes the cacheable accesses as needed, while bypassing the non-cacheable accesses or forwarding the cache write-through and cache miss accesses to the downstream memories through the master port of this cache controller.

3.2.2 System bus access

All System bus accesses are routed to the target address in destination memories through multilayer AHB matrix slave port.

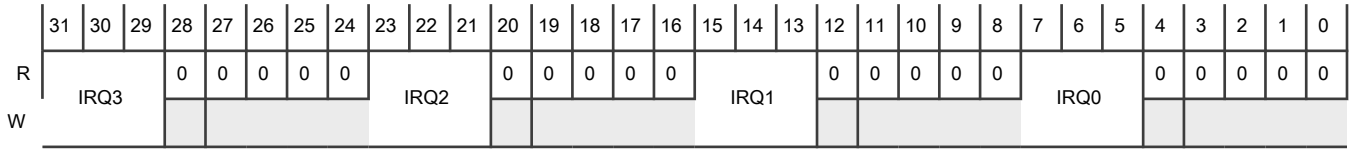
3.2.3 Access control

All core Code and System Bus accesses are checked by the core access control logic. All requests that miss or bypass the cache are checked by downstream secure AHB bus logic. The caches include protection control signals (HPROT[3:0]) and processing domain bits as part of the tags. If a fetch address hits the cache but the protection control and/or domain bits are different, the cache controller forces a miss with the allocate location the same as the address hit location in the cache. This policy allows all the downstream checks to take place, and this new miss is loaded in the cache with the updated protection control and domain bits overwriting the line with the same address. This keeps the cache coherent while always checking accesses that need to see the downstream checks.

3.3 Nested Vectored Interrupt Controller (NVIC)

3.3.1 Interrupt priority levels

This device supports 8 priority levels for interrupts. Therefore, in the NVIC each source in the IPR registers contains 3 bits. For example, IPR0 is shown below:



3.3.2 Non-Maskable Interrupt (NMI) configuration

The Non-Maskable Interrupt (NMI) enable bit and source selection bits are implemented for each core in SYSCON register.

3.3.3 Interrupt channel assignments

The interrupt source assignments are defined in the following table.

- Vector number — the value stored on the stack when an interrupt is serviced.
- IRQ number — non-core interrupt source count, which is the vector number minus 16.

The IRQ number is used within Arm's NVIC documentation.

See the attached NVIC_Configuration spreadsheet for the NVIC interrupt assignments.

3.4 System memory map

See the attached Memory Map spreadsheet (Overview tab) for the chip's high-level memory map.

3.5 Peripheral Bridge

The Peripheral Bridge (PBRG) is the portion of the bus fabric that connects the peripherals to the processor elements. Each peripheral has a base address where the processor elements can access them.

3.5.1 Peripheral Bridge (PBRIDGE0-1) memory maps

See the attached Memory Map spreadsheet for PBRIDGE0, PBRIDGE1, and Fast Peripherals (Peripheral Memory Map tab) peripheral memory mapping on this device.

Chapter 4
Security Overview

4.1 Non-claims

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.

- This SoC has built-in security event detection features. However, NXP does not guarantee against advanced tamper attempts, including operation of the device beyond the defined specification limits. The security event detection features are only effective if the analog glitch detector is enabled in the device configuration. NXP does not guarantee the protection against semi-invasive and invasive attacks.
- This SoC has a built-in feature addressing side channel attacks. 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.
- Several aspects of this SoC's security trust architecture rely on the strength of cryptographic algorithms and digital signatures. 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.

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.

4.2 Overview

This chapter provides an overview of the security features that this chip device implements using the following on-chip components.

- Secure Generic Interface (SGI) that supports secure computation of AES and SHA algorithms.
- Public-Key Cryptography Accelerator (PKC)
- Code Watchdog (CDOG) to ensure software integrity by detecting unexpected changes in the code execution flow.
- Memory Block Checker (MBC) that provides read, write, and execute access control per block of internal flash memory. See the *MBC* chapter in *MCXA 255, and A256 Reference Manual* for information about MBC.
- Digital Tamper (TDET) to support tamper detection.
- Cyclic Redundancy Check (CRC) that provides error detection for multi-bit errors. See the *CRC* chapter in *MCXA 255, and A256 Reference Manual* for information about CRC.
- GLIKEY to ensure safe access to security-sensitive registers.
- True Random Number Generator (TRNG)
- Analog Glitch Detector (AGDET)

The following figure shows the security system diagram of this device.

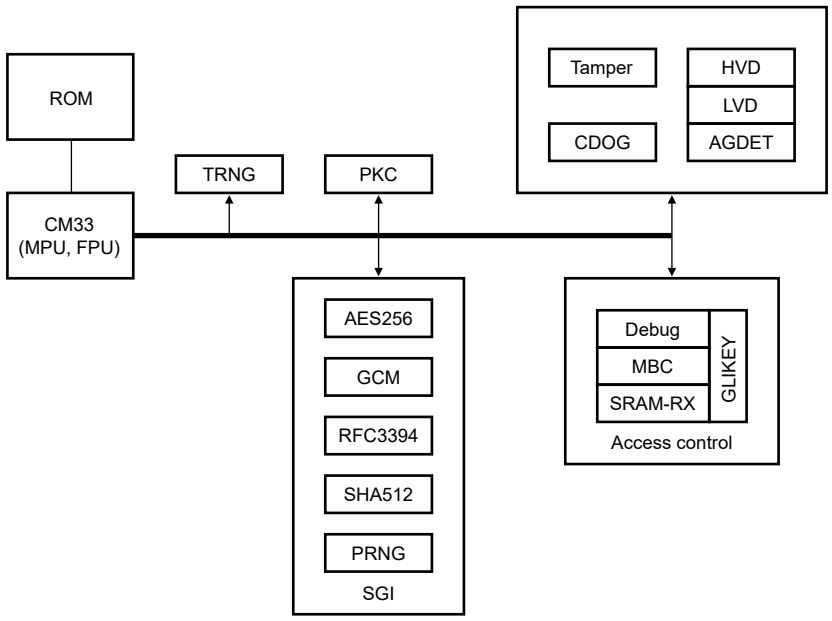


Figure 6. Security system diagram

4.3 Security features

4.3.1 Immutable Root of Trust

As defined by Trusted Computing Group, “an Immutable Root of Trust (RoT) is expected to remain identical across all devices within a set of device models based on a defined threat model. It is also expected not to change across time and, therefore, will behave the same during each device’s lifespan.” It consists of truly immutable hardware logic, including analog and digital logic, read-only memory and one-time programmable memory. Immutable RoT is essential for guaranteeing any security feature, including Life-cycle Management, and a number of others. In this device, Immutable RoT is embedded in the Boot ROM and subsequently in an immutable bootloader.

4.3.2 Life cycle management

During its lifespan, a typical device finds itself in various places around the world. It is manufactured in a semiconductor factory, tested and packaged in silicon manufacturer facilities, sold to various distributors, sold further to the Original Equipment Manufacturer (OEM), assembled, tested and provisioned by their Contract Manufacturers and, finally, delivered to the end-customer. In the case of failure, a device is returned to OEM or even back to the silicon manufacturer for further failure analysis.

Device life-cycle state is used to reflect the actual state of the device, which is further used to instruct the device on how exactly to protect the assets a device hosts during specific time. For example, when a device is being tested at a silicon manufacturer facility and no OEM or end-customer assets have been provisioned on it, then access to the device, in terms of debug or test, is less restrictive than when a device is with the end-customer.

Life-cycle state is monotonic, meaning it can only always be increased. Immutable RoT is in charge of life-cycle management and it enforces device access policies accordingly. See Life Cycle States for details.

4.3.3 Secure key management

Secure key management is a process of securing valuable keys and various key material which are essential in maintaining security of the end-user, OEM and NXP assets. The process strongly relies on the Immutable RoT and hardware logic. A device-unique master key (DUK) is provided by the Immutable RoT and loaded into write-only key slots 6 and 7 of the Secure Generic Interface (SGI). The DUK can be freely used for AES encryption/decryption of sensitive data, key wrapping/unwrapping

or derivation of further key material. When using the RFC3394 key unwrap feature of the SGI, the unwrapped key will be stored in write-only key slots 4 and 5.

NOTE

SGI key registers are not sticky-lockable and can be used freely, so key isolation is not a feature on this device.

4.3.4 Secure installer

To support cases where outsourcing of device software provisioning to an untrusted Contract Manufacturer is desired, this SoC comes with a pre-installed secure installer firmware which can be accessed through In-System Programming (ISP) mode in the default life cycle state. NXP software tools enable the Original Equipment Manufacturer (OEM) to build an encrypted file containing a sequence of provisioning commands. This encrypted file can be transferred to Contract Manufacturers and securely installed on the devices. The devices are able to decrypt the contents directly into persistent memory and advance the life cycle state so that the Contract Manufacturer is unable to inspect the provisioned code and data. SeeSecure Installer for more details.

NOTE

In order to protect against several attacks, it is recommended that an OEM advances the life cycle state as soon as possible in their encrypted image.

4.3.5 Anomaly Detection and Reaction

Anomaly Detection and Reaction describes the processes or algorithms that analyze the device input and output such as sensor data, as well as the software integrity and application operation for abnormal events and, if required, trigger and execute an action. Typically these actions encompass logging the anomaly, issuing a message to the cloud backend, resetting the device, and/or changing a life cycle state.

Chapter 5
Life Cycle States

5.1 Overview

This device supports a life cycle state model to protect code from reading from the device internal flash, which is called code read-out protection (ROP) feature. There are different levels of protections in the system, so that access to the on-chip flash and use of ISP can be restricted. Also the life cycle state of the device determines the debug access and ISP command availability. Please see other sections details. The life cycle state is controlled by ROP_STATE (in the CMPA region).

The Boot ROM will check the life cycle state and then determine what Boot flow to run, and debug port to be enabled/locked and available debug mail access command sets. After ROM exits to extended bootloader, life cycle will be checked again to determine available ISP command sets.

5.2 Life cycle state transitioning

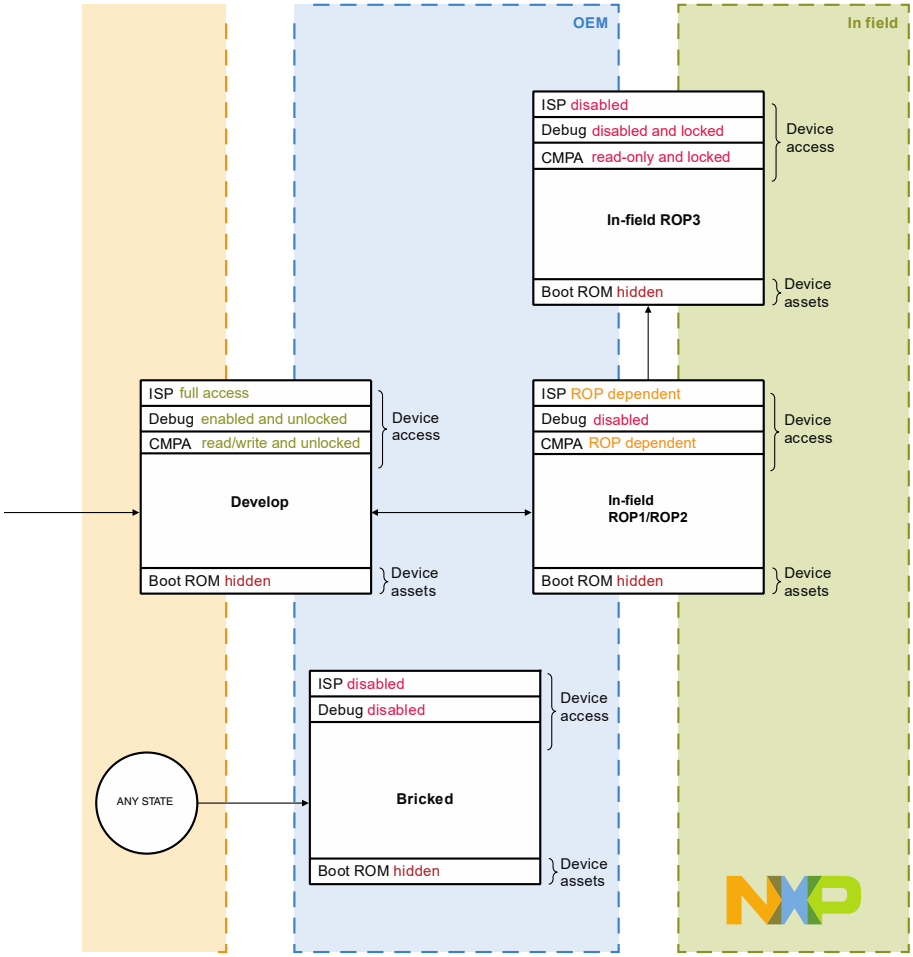


Figure 7. Life cycle state transitioning

5.3 Life cycle states

This device supports the following life cycle states.

Table 16. Life cycle states

Life cycle	Life cycle type	Description
Develop	Customer	Initial customer development state after leaving NXP manufacturing
In-field ROP1/ROP2	Customer	In-field application state, with ROP protection
In-field ROP3	Customer	In-field application state, with ROP protection, prevent use of field return
Bricked	End-of-life	Bricked state to prevent device use

5.4 Identify the life cycle (LC) state

The following chip state is used to identify the life cycle state.

- ROP_STATE - Value can be read out from SYSCON-> ROP_STATE (read/write register)

ROM reads ROP_STATE and ROP_STATE_DP (duplicate of ROP_STATE) from IFR0, and performs the compare. If they don't match, ROM will reset the device; if they match, ROM will program it to SYSCON->ROP_STATE.

Table 17. IFR0 address of ROP_STATE and ROP_STATE_DP

IFR0	IFR0 Address
ROP_STATE	0x0007_E0B0 for MCX A255, and A265 0x000F_E0B0 for MCX A256, and A266
ROP_STATE_DP	0x0007_E0C0 for MCX A255, and A265 0x000F_E0C0 for MCX A256, and A266

5.5 Read-out protection (ROP)

This device does not support secure boot, but it does support four levels of ROP, also refer to ROP_STATE. The ROP mechanism allows user to enable different levels of protection in the system. It is a 32-bit field stored in IFR0. It can be programmed by customer.

- ROP_LEVEL0, ROP_STATE = 0xFFFF_FFFF (erased FLASH value), no ROP. Default for blank state.
- ROP_LEVEL1, ROP_STATE = 0xEEBA_04C3

  Debug is disabled and unlocked, but can be modified by customer. Only limited debug mailbox commands are available.
- ROP_LEVEL2, ROP_STATE = 0x4939_8D8B

  Debug is disabled and locked, and cannot be modified by customer. Only limited debug mailbox commands are available.
- ROP_LEVEL3, ROP_STATE = 0xB0AB_B703

  Debug is disabled and locked, and cannot be modified by customer. No debug mailbox commands are available.
- Anything else = ROP3-like behavior (Debug disabled/Locked, ISP disabled).
- ROP_STATE

  Value can be read out from SYSCON-> ROP_STATE (read/write register).

  ROM reads ROP_STATE from ROP_STATE and ROP_STATE_DP on IFR0, does the compare, and if it does not match, resets the device; if match, then programs it to SYSCON->ROP_STATE.

5.5.1 Available debug mailbox commands per life cycle

Available debug mailbox commands sets depends on life cycle state and read out protection level. If the part is in Bricked state or in-field ROP3, then ROM will lock the part, no debug mail commands are available.

Below table summarizes available debug mail box commands corresponding to read out protection level and life cycle state.

Table 18. Debug mailbox command sets per life cycle state

LC MANAGEMENT	Available commands
Develop	Start DM-AP, Get_ROP_Level, Bulk Erase, (user flash and IFR0 sector 0) Exit DM-AP, Enter_ISP_Mode, Set_FA_Mode, Start Debug Session, Write_Flash_Word, Read_Flash_Word, Erase_One_Sector
In-field ROP1	Start DM-AP, Get_ROP_Level, Bulk_Erase, (user flash and IFR0 sector 0) Enter_ISP_Mode, Set_FA_Mode, Exit DM-AP
In-field ROP2	Start DM-AP, Get_ROP_Level, Bulk_Erase (user flash and IFR0 sector 0), Set_FA_Mode, Exit DM-AP
In-field ROP3	Debug mail box is not available
Bricked	Debug mail box is not available

Chapter 6
ROM API

6.1 Overview

ROM bootloader provides APIs for users. Disabling the interrupts before making any ROM API calls is suggested, since API does not handle any interrupts.

The structure of bootloader_tree is show as below.

```
//! @brief Root of the bootloader API tree.
//!
//! An instance of this struct resides in read-only memory in the bootloader. It provides a user
application access to APIs exported by the bootloader.
//! @note The order of existing fields must not be changed.
//!
//! @ingroup context
typedef struct BootloaderTree
{
    void (*runBootloader)(void *arg)                //!< Function to start the bootloader executing.
    const flash_driver_interface_t *flashDriver;    //!< Internal Flash driver API.
    void (*jump)(uint32_t imageBase);
} bootloader_tree_t;
```

The ROM API table is located at address 0x03005fe0.

```
#define BOOTLOADER_TREE_LOCATION (0x03005fe0)
#define g_bootloaderTree ((bootloader_tree_t *)BOOTLOADER_TREE_LOCATION)
```

6.1.1 runBootloader API

ROM provides an API for the user application to enter ISP mode based on the designated ISP interface mode.

A prototype of the runBootloader API: `Void (*runBootloader) (void *arg).`

Table 19. API prototype fields

Field	Offset	Description
Tag	[31:24]	Fixed value: 0xEB (Enter boot mode)
Boot mode	[23:20]	0 - Enters passive mode
		1 - Enters ISP mode
ISP interface	[19:16]	0 - Auto detection
		1 - USB-HID
		2 - UART
		3 - SPI
		4 - I2C
		5 - CAN
Reserved	[15:4]	
Image index	[3:0]	Must be 0

6.1.2 Flash driver API

Flash driver API provide the flash operation supported by CM33 based on flash technology.

```
//! @brief Interface for the flash driver.
typedef struct FlashDriverInterface
{
    // Flash driver
    status_t (*flash_init)(flash_config_t *config);
    status_t (*flash_erase_sector)(flash_config_t *config, uint_32_t start, uint32_t lengthInBytes,
uint32_t key);
    status_t (*flash_program_phrase)(flash_config_t *config, uint32_t start, uint8_t *src,
uint32_t lengthInBytes);
    status_t (*flash_program_page)(flash_config_t *config, uint32_t start, uint_t *src,
uint_t lengthInBytes);
    status_t (*flash_verify_program)(flash_config_t *config,
                                     uint32_t start,
                                     uint32_t lengthInBytes,
                                     const uint8_t *expectedAddress,
                                     uint32_t *failedAddress,
                                     uint32_t *failedData);
    status_t (*flash_verify_erase_phrase)(flash_config_t *config, uint32_t start,
uint32_t lengthInBytes);
    status_t (*flash_verify_erase_page)(flash_config_t *config, uint32_t start,
uint32_t lengthInBytes);
    status_t (*flash_verify_erase_sector)(flash_config_t *config, uint32_t start,
uint32_t lengthInBytes);
    status_t (*flash_get_property)(flash_config_t *config, flash_property_tag_t whichProperty,
uint32_t *value);
    status_t (*flash_read)(flash_config_t *config, uint32_t start, uint8_t *dest,
uint32_t lengthInBytes);
    // version
    standard_version_t version; //!< flash driver API version number.
} flash_driver_interface_t;
```

And structure of flash_config_t is defined as the following.

```

typedef struct
{
    uint32_t PFlashBlockBase;        /*!< A base address of the first PFlash block */
    uint32_t PFlashTotalSize;        /*!< The size of the combined PFlash block. */
    uint32_t PFlashBlockCount;       /*!< A number of the PFlash blocks. */
    uint32_t PFlashPageSize;         /*!< The size in bytes of a page of PFlash. */
    uint32_t PFlashSectorSize;       /*!< The size in bytes of a sector of PFlash. */
    flash_ffr_config_t ffrConfig;
} flash_config_t;
```

Table 20. Flash driver API function details

Function name	Description
Flash_init	Initializes the global flash properties structure members
Flash_erase_sector	Erases the flash sectors encompassed by parameters passed into function
Flash_program_phrase	Programs flash phrases with data at locations passed in through parameters

Table continues on the next page...

Table 20. Flash driver API function details (continued)

Function name	Description
Flash_program_page	Programs flash with data at locations passed in through parameters
Flash_verify_program	Verifies programming of the desired flash area at a specified margin level.
Flash_verify_erase_phrase	Verify that the flash phrases are erased
Flash_verify_erase_page	Verify that the flash pages are erased
Flash_verify_erase_sector	Verify that the flash sectors are erased
Flash_get_property	Returns the desired flash property
ifr_verify_erase_phrase	Verify that the IFR0 phrases are erased
ifr_verify_erase_page	Verify that the IFR0 pages are erased
ifr_verify_erase_sector	Verify that the IFR0 sectors are erased
flash_read	Reads flash at locations passed in through parameters

Table 21. Flash API return status code

Status	Code	Description
kStatus_FLASH_Success	0	The flash operation is successful
kStatus_FLASH_InvalidArgument	4	Invalid argument during executing API
kStatus_FLASH_SizeError	100	Invalid size during executing API
kStatus_FLASH_AlignmentError	101	Alignment error during executing API
kStatus_FLASH_AddressError	102	Address error during executing API
kStatus_FLASH_AccessError	103	Invalid instruction codes during executing API
kStatus_FLASH_ProtectionViolation	104	Protection violation flag, indicate that program/erase operation is requested to execute on protected areas
kStatus_FLASH_CommandFailure	106	Command execution failure during executing API
kStatus_FLASH_UnknownProperty	106	Unknown property when to call flash_get_property
kStatus_FLASH_EraseKeyError	107	Invalid EraseKey during executing flash_erase API

Chapter 7
Extended Bootloader and In-System Programming (ISP)

7.1 Overview

After a POR or warm reset sequence, CPU will always start ROM code execution in Boot ROM memory. After the operations in Boot ROM, the control of the CPU is transferred to the extended bootloader in flash IFR0 memory. This extended bootloader only supports loading non-secure images (image format specified in Boot ROM section) to on-chip memory, including RAM and flash. The extended bootloader will also perform an integrity check before jumping to the user image if a CRC is present in the image header.

Bootloader features:

- Extended bootloader size can be up to 32 KB (IFR0, sector 0 - 3)
- Booting on USB FS HID interface, LPSPI slave interface, LPI2C interface and LPUART interface with auto-baud detection
- Supporting 3 paths:
  - jumping to user application
  - ISP mode
  - low-power wakeup
- Image update and swap

7.2 Boot peripherals and default pins

Table 22. Bootloader peripheral pinmux

Peripheral	Instance	Alt Mode	Port	GPIO
LPUART	2	3	LPUART0_RXD	P2_2
			LPUART0_TXD	P2_3
LPI2C	2	3	LPI2C2_SDA	P1_8
			LPI2C2_SCL	P1_9
LPSPI	0	2	LPSPI0_SDO	P1_0
			LPSPI0_SCK	P1_1
			LPSPI0_SDI	P1_2
			LPSPI0_PCS	P1_3
USB	0		USB0_DM	USB0_DM
			USB0_DP	USB0_DP

7.3 Functional description

7.3.1 Jumping to user application path

With successful integrity CRC check on user application, before jump to user application, extended bootloader needs:

- Shutdown all boot peripherals.
- Restore the registers to default values, these registers are used by all modules used by ROM or extended bootloader, except MBC and GLIKEY settings.
- Set GLIKEY and MBC settings based on life cycle.

7.3.2 Lower power wake-up Path

During deep-sleep wake-up or sleep wake-up, the core would continue executing application without any ROM- or HW-enforced checks. It would be up to the application to implement its own checks if a customer deems necessary.

The difference between ROM and extended bootloader on lower pawer wakeup path is that extended bootloader run CRC check on user image if image indicates valid image type at offset 0x24 (reserved word of vector table)

The low power wake-up path is focused on reducing the time to start executing customer code when waking from Deep Power-Down mode. This reduction in boot time is primarily achieved by doing simplified integrity CRC check. With successfully pass the integrity CRC check, then jump to wakeup entry specified in IFR0; otherwise enter into ISP path in extended bootloader.

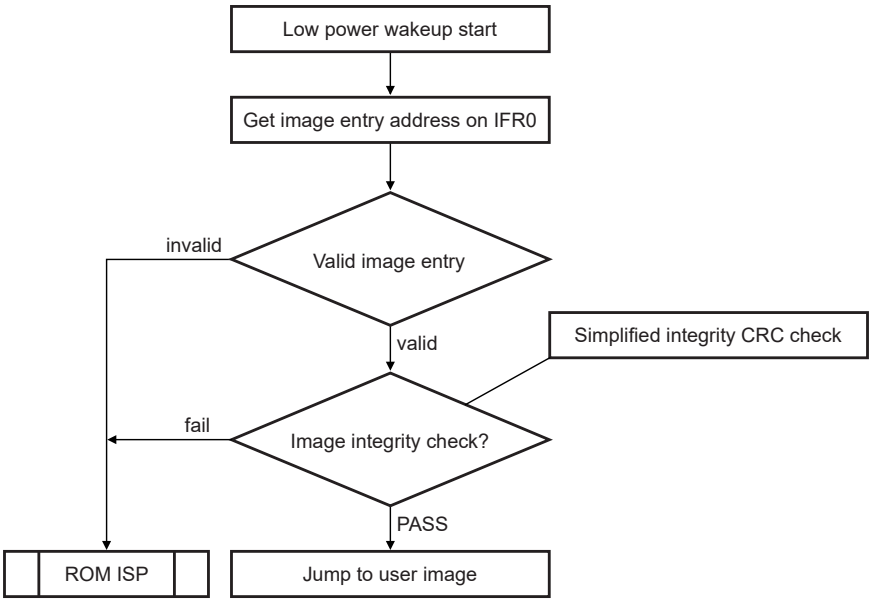


Figure 8. Extended bootloader DPD wake-up flow

7.3.3 ISP calls path

Due to limited IFR0 memory space, extended bootloader support these commands:

- Full ISP command set:
  - ReadMemory
  - FlashEraseAll (user flash & IFR0 sector 0)
  - FlashEraseRegion
  - Reset
  - WriteMemory
  - GetProperty
  - Execute
- Reduced ISP command set:

— FlashEraseAll (user flash & IFR0 sector 0)

— Reset

— GetProperty

See Table 23 for the ISP commands set available for each life cycle state.

Table 23. Available ISP commands set per life cycle state

Life cycle state	Available commands set
NXP_Provisioned	Full ISP command set
OEM_Open	
OEM_Field_Return	
NXP_Filed_Return	
OEM_Closed, ROP1	Reduced command set if entry through: ISP_PIN, INVALID_IMAGE, DEBUG_MAILBOX Full command set if entry through: bootloader_API
OEM_Closed, ROP2	Reduced command set if entry through: INVALID_IMAGE, bootloader_API
OEM_Closed, ROPx	No ISP commands are available
OEM_Closed No return	No ISP commands are available
NXP_Bricked	No ISP commands are available

7.4 Flash swap path

Extended bootloader supports flash swap, which is the function of FMC in order to switch between IMG0 and IMG1. On this device, there is only one bank of flash, this one bank flash is divided to four regions, two swappable regions and two static regions.

- Lower swappable region (swappable region 0): flash physical start address ~ limit
- Lower static region (static region 0): limit + 1 ~ half size of flash – 1
- Upper swappable region (swappable region 1): half size of flash ~ half size of flash + limit
- Upper static region (static region 1): half size of flash + limit + 1 ~ size of flash – 1

To be able to run flash swap, the basic requirements are:

1. Size of swappable region 0 equals to size of swappable region 1
2. Size of swappable region 0 plus size of static region 0 equals to half size of flash

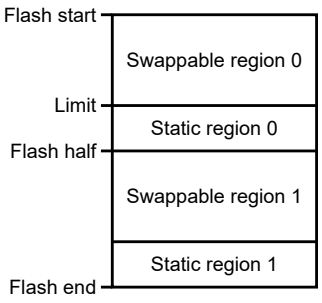


Figure 9. Flash swap path

7.4.1 Flash swap flow chart

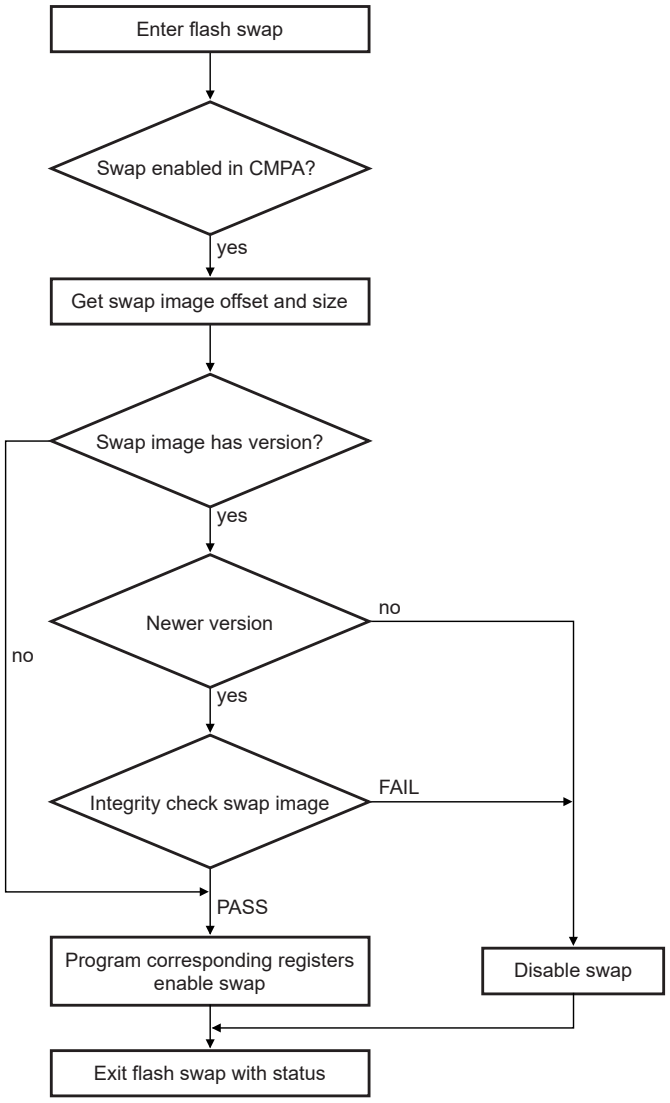


Figure 10. Flash swap flow chart

7.5 ISP protocol

This section explains the general protocol for the packet transfers between the host and the bootloader. The description includes the transfer of packets for different transactions, such as commands with no data phase and commands with incoming or outgoing data phase. The next section describes various packet types used in a transaction.

Each command sent from the host is replied to with a response command.

Commands may include an optional data phase.

- If the data phase is incoming (from the host to the bootloader), it is part of the original command.
- If the data phase is outgoing (from the bootloader to host), it is part of the response command.

7.5.1 Command with no data phase

The protocol for a command with no data phase contains:

- Command packet (from the host)
- Generic response command packet (to host)

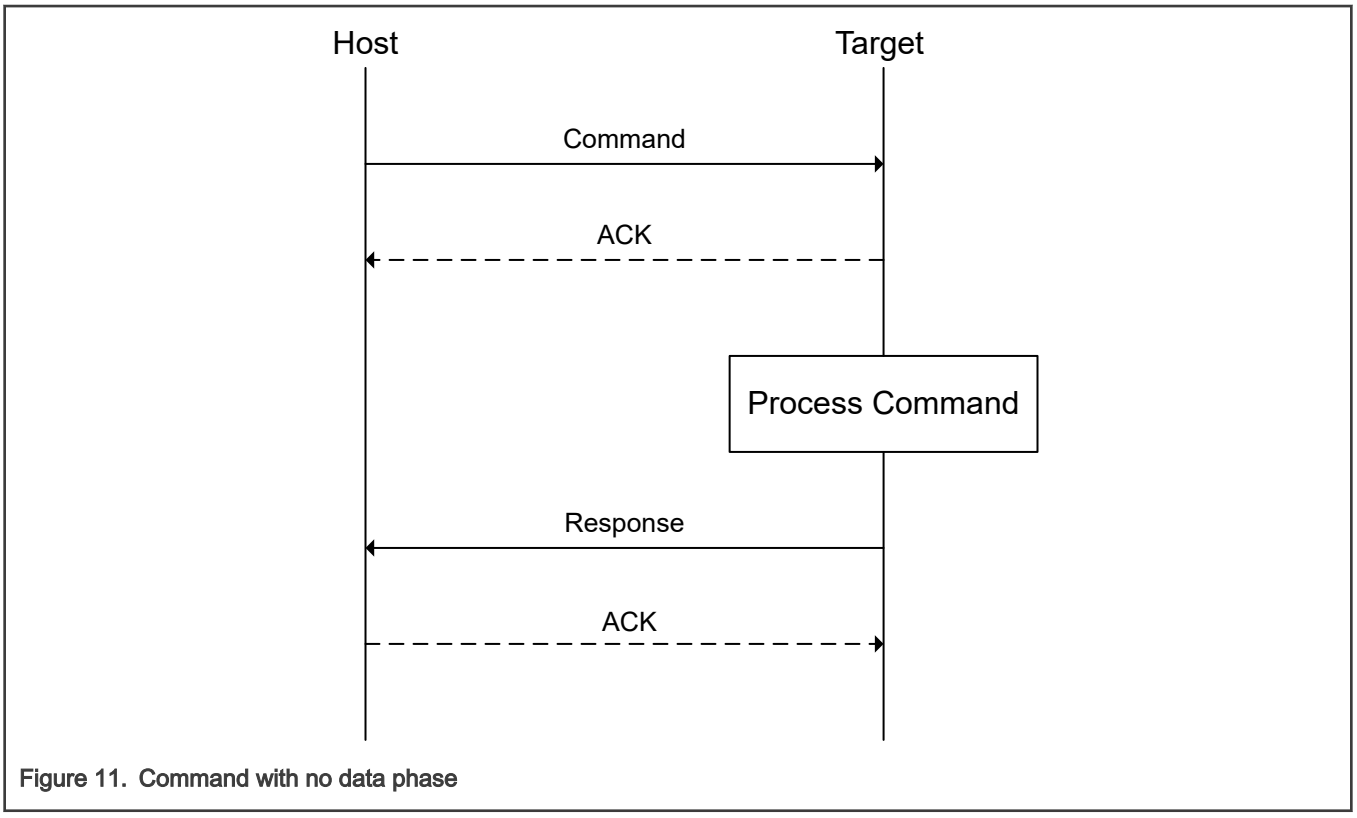


Figure 11. Command with no data phase

Remark: In these diagrams, the ACK sent in response to a command or a data packet can arrive at any time before, during, or after the command or data packet has processed.

7.5.2 Command with incoming data phase

The protocol for a command with incoming data phase contains:

- Command packet (from host) (kCommandFlag_HasDataPhase set).
- Generic response command packet (to host).
- Incoming data packets (from the host).
- Generic response command packet.

NOTE

- The host may not send any further packets while it is waiting for the response to a command.
- The data phase is aborted if the Generic Response packet prior to the start of the Data phase does not have a status of kStatus_Success.
- Data phases may be aborted by the receiving side by sending the final.
- GenericResponse early with a status of kStatus_AbortDataPhase. The host may abort the data phase early by sending a zero-length data packet.
- The final Generic Response packet sent after the data phase includes the status of the entire operation.

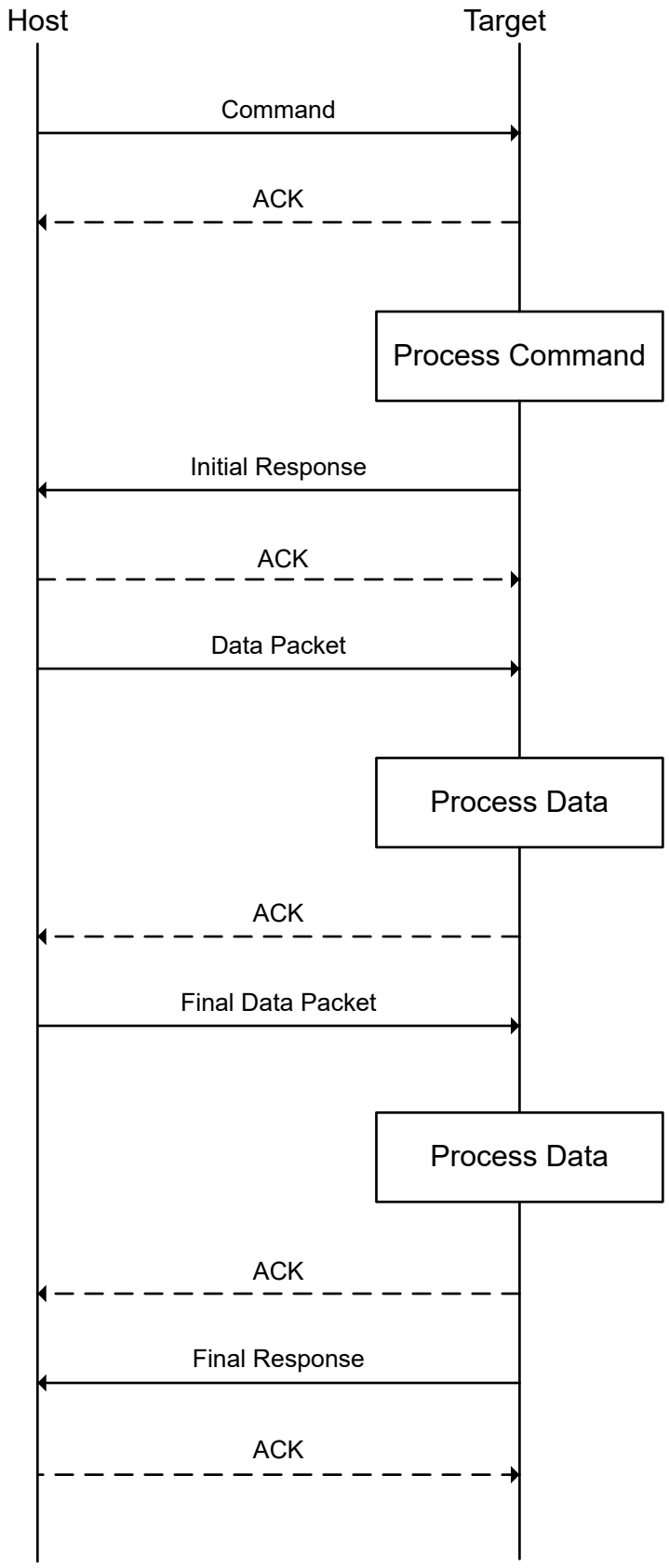


Figure 12. Command with incoming data phase

7.5.3 Command with outgoing data phase

The protocol for a command with an outgoing data phase contains:

- Command packet (from the host).
- ReadMemory Response command packet (to host) (kCommandFlag_HasDataPhase set).
- Outgoing data packets (to host).
- Generic response command packet (to host).

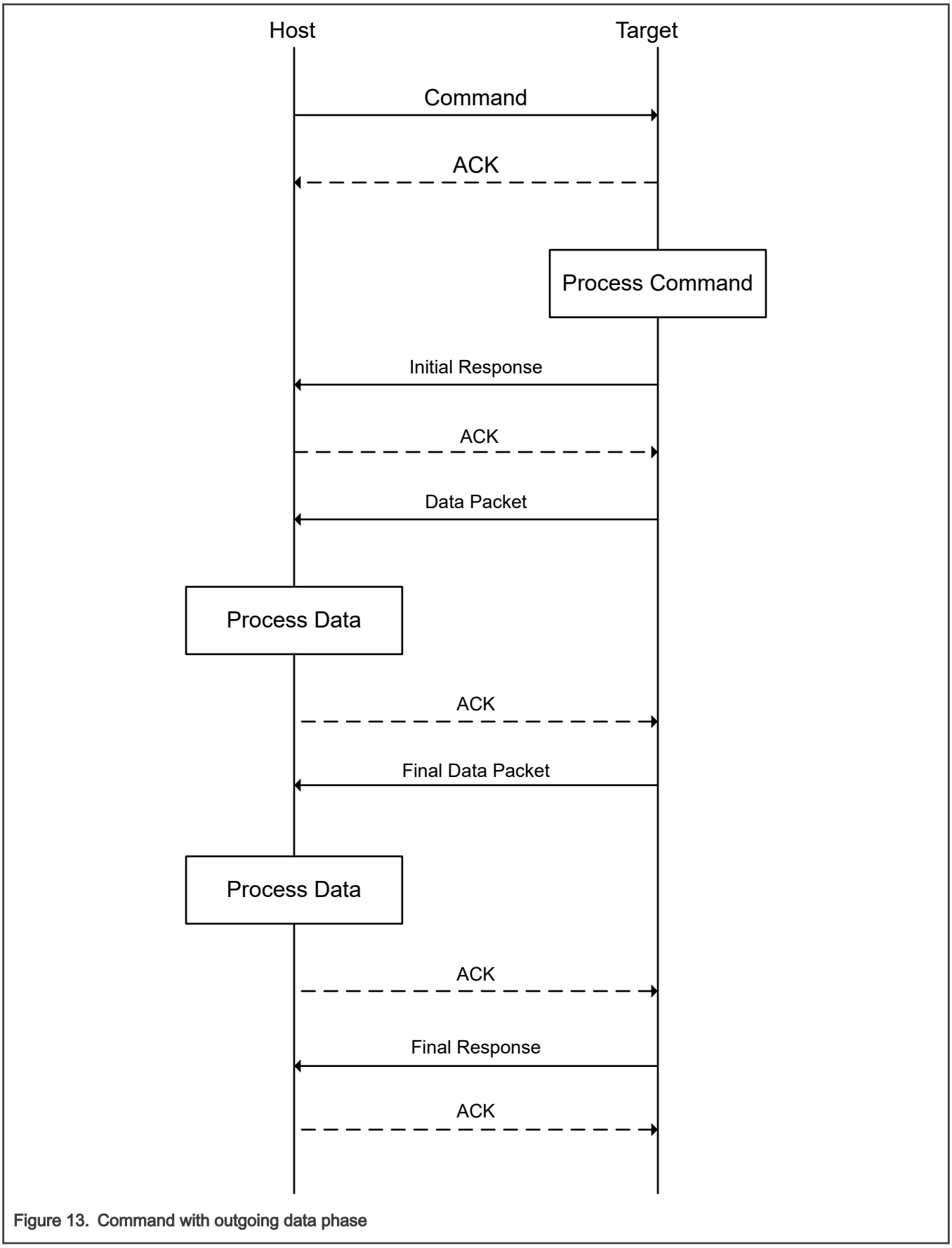


Figure 13. Command with outgoing data phase

7.6 Bootloader packet types

7.6.1 Introduction

The bootloader device works in slave mode. All data communications are initiated by a host, which is either a PC or an embedded host. The bootloader device is the target, which receives a command or data packet. All data communications between host and target are packetized. The following packets are used.

- Ping packet
- Ping response packet
- Framing packet
- Command packet
- Data packet
- Response packet

All fields in the packets are in little-endian byte order.

7.7 Ping packet

The Ping packet is the first packet sent from a host to the target to establish a connection on the selected peripheral in order to run autobaud detection. The Ping packet can be sent from host to target at any time that the target is expecting a command packet. If the selected peripheral is UART, a Ping packet must be sent before any other communications. For other serial peripherals, it is optional.

In response to a Ping packet, the target sends a Ping response packet, discussed in the later sections.

Table 24. Ping packet format

Byte #	Value	Name
0	0x5A	Start byte
1	0xA6	Ping

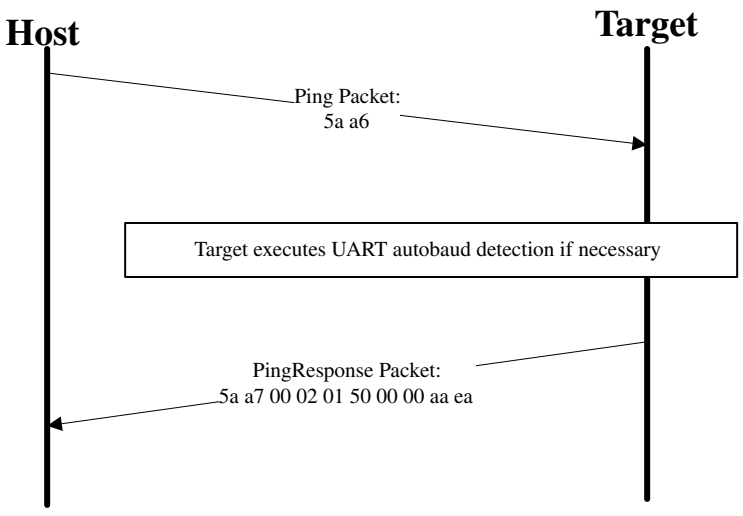


Figure 14. Ping packet protocol sequence

7.7.1 Ping response packet

The target sends a Ping response packet back to the host after receiving a Ping packet. If communication is over a UART peripheral, the target uses the incoming Ping packet to determine the baud rate before replying with the Ping response packet. Once the Ping response packet is received by the host, the connection is established, and the host starts sending commands to the target.

Table 25. Ping response packet format

Byte	Value	Parameter
0	0x5A	Start byte
1	0xA7	Ping response code
2	0x00	Protocol bugfix
3	0x03	Protocol minor
4	0x01	Protocol major
5	0x50	Protocol name = 'P' (0x50)
6	0x00	Options low
7	0x00	Options high
8	0xfb	CRC16 low
9	0x40	CRC16 high

For the UART peripheral, it must be sent by the host when a connection is first established, in order to run outbound. For other serial peripherals, it is optional but recommended to determine the serial protocol version. The version number is in the same format as the bootloader version number returned by the GetProperty command.

7.7.2 Framing packet

The framing packet is used for flow control and error detection for the communications links that do not have such features built in. The framing packet structure sits between the link layer and the command layer. It wraps command and data packets as well.

Every framing packet containing data sent in one direction results in a synchronizing response framing packet in the opposite direction.

The framing packet described in this section is used for serial peripherals including the UART, I^2C, and SPI. The USB HID peripheral does not use framing packets. Instead, the packetization inherent in the USB protocol itself is used.

Table 26. Framing packet format

Byte	Value	Parameter	Description
0	0x5A	Start byte	
1		PacketType	
2		Length_low	Length is a 16-bit field that specifies the entire command or data packet size in bytes.
3		Length_high	
4		Crc16_low	This is a 16-bit field. The CRC16 value covers entire framing packet, including the start byte and command or data packets, but does not include the CRC bytes. See Section 13.5 “CRC16 algorithm”.
5		Crc16_high	
6....n		Command or Data packet payload	

A special framing packet that contains only a start byte and a packet type is used for synchronization between the host and target.

Table 27. Special framing packet format

Byte	Value	Parameter
0	0x5A	Start byte
1	0xA*n*	packetType

The Packet Type field specifies the type of the packet from one of the defined types (below):

Table 28. Packet type field

Packet type	Name	Description
0xA1	kFramingPacketType_Ack	The previous packet was received successfully; the sending of more packets is allowed.
0xA2	kFramingPacketType_Nak	The previous packet was corrupted and must be re-sent.
0xA3	kFramingPacketType_AckAbort	Data phase is being aborted.
0xA4	kFramingPacketType_Command	The framing packet contains a command packet payload.
0xA5	kFramingPacketType_Data	The framing packet contains a data packet payload.
0xA6	kFramingPacketType_Ping	Sent to verify the other side is alive. Also used for UART autobaud.
0xA7	kFramingPacketType_PingResponse	A response to Ping. It contains the framing protocol version number and options.

7.7.3 CRC16 algorithm

The CRC is computed over each byte in the framing packet header, excluding the CRC16 field itself, and all of the payload bytes. The CRC algorithm is the XMODEM variant of CRC16.

The characteristics of the XMODEM variants are:

Table 29. CRC16 algorithm

Width	16
Polynomial	0x1021
Init value	0x0000
Reflect in	False
Reflect out	False
Xor out	0x0000
Check result	0x31c3

The check result is computed by running the ASCII character sequence "123456789" through the algorithm.

```
uint16_t crc16_update(const uint8_t * src, uint32_t lengthInBytes)
{
uint32_t crc = 0;
uint32_t j;
for (j=0; j < lengthInBytes; ++j)
```

```
{
uint32_t i;
uint32_t byte = src[j];
crc ^= byte << 8;
for (i = 0; i < 8; ++i)
{
uint32_t temp = crc << 1;
if (crc & 0x8000)
{
temp ^= 0x1021;
}
crc = temp;
}
}
return crc;
}
```

7.7.4 Command packet

The command packet carries a 32-bit command header and a list of 32-bit parameters.

Table 30. Command packet format

Command packet format (32 bytes)										
Command Header (4 bytes)				28 bytes for Parameters (Max 7 parameters)						
Tag	Flags	Rsvd	Param Count	Param 1 (32-bit)	Param 2 (32-bit)	Param 3 (32-bit)	Param 4 (32-bit)	Param 5 (32-bit)	Param 6 (32-bit)	Param 7 (32-bit)

Table 31. Command header format

Byte #	Command header field	Reset value
0	Command or Response tag	The command header is 4 bytes long with these fields.
1	Flags	
2	Reserved. Should be 0x00.	
3	ParameterCount	

The header is followed by 32-bit parameters up to the value of the ParameterCount field specified in the header. Because a command packet is 32 bytes long, only seven parameters can fit into the command packet.

Command packets are also used by the target to send responses back to the host. As mentioned earlier, command packets and data packets are embedded into framing packets for all of the transfers.

Table 32. Command tags

Command tag	Name	Description
0x01	FlashEraseAll	The command tag specifies one of the commands supported by the bootloader. The valid command tags for the bootloader are listed here.
0x02	FlashEraseRegion	
0x03	ReadMemory	
0x04	WriteMemory	
0x07	GetProperty	
0x09	Execute	
0x0B	Reset	

Table 33. Response tags

Response tag	Name	Description
0xA0	GenericResponse	The response tag specifies one of the responses the bootloader (target) returns to the host. The valid response tags are listed here.
0xA7	GetPropertyResponse (used for sending responses to GetProperty command only)	
0xA3	ReadMemoryResponse (used for sending responses to ReadMemory command only)	

Flags: Each command packet contains a flag byte. Only bit 0 of the flag byte is used. If bit 0 of the flag byte is set to 1, then data packets follow the command sequence. The number of bytes that are transferred in the data phase is determined by a command specific parameter in the parameters array.

ParameterCount: The number of parameters included in the command packet.

Parameters: The parameters are word-length (32 bits). With the default maximum packet size of 32 bytes, a command packet can contain up to seven parameters.

7.7.5 Response packet

The responses are carried using the same command packet format wrapped with framing packet data. Types of responses include:

- GenericResponse.
- GetPropertyResponse.
- ReadMemoryResponse.

GenericResponse: After the bootloader has processed a command, the bootloader sends a generic response with status and command tag information to the host. The generic response is the last packet in the command protocol sequence. The generic response packet contains the framing packet data and the command packet data (with generic response tag = 0xA0) and a list of parameters (defined in the next section). The parameter count field in the header is always set to 2, for status code and command tag parameters.

Table 34. Generic response parameters

Byte #	Parameter	Description
0 - 3	Status code	The Status codes are errors encountered during the execution of a command by the target. If a command succeeds, then a kStatus_Success code is returned.
4 - 7	Command tag	The Command tag parameter identifies the response to the command sent by the host.

GetPropertyResponse: The GetPropertyResponse packet is sent by the target in response to the host query that uses the GetProperty command. The

GetPropertyResponse packet contains the framing packet data and the command packet data, with the command/response tag set to a GetPropertyResponse tag value (0xA7).

The parameter count field in the header is set to greater than 1, to always include the status code and one or many property values.

Table 35. GetPropertyResponse parameters

Byte #	Value	Parameter
0 - 3		Status code
4 - 7		Property value
. . .		. . .
		Can be up to maximum 6 property values, limited to the size of the 32-bit command packet and property type.

ReadMemoryResponse: The ReadMemoryResponse packet is sent by the target in response to the host sending a ReadMemory command. The ReadMemoryResponse packet contains the framing packet data and the command packet data, with the command/response tag set to a ReadMemoryResponse tag value (0xA3), the flags field set to kCommandFlag_HasDataPhase (1).

The parameter count set to two for the status code and the data byte count parameters shown below.

Table 36. ReadMemoryResponse parameters

Byte #	Parameter	Description
0 - 3	Status code	The status of the associated Read Memory command.
4 - 7	Data byte count	The number of bytes sent in the data phase.

FlashReadOnceResponse: The FlashReadOnceResponse packet is sent by the target in response to the host sending a FlashReadOnce command. The FlashReadOnceResponse packet contains the framing packet data and the command packet data, with the command/response tag set to a FlashReadOnceResponse tag value (0xAF), and the flags field set to 0. The parameter count is set to 2 plus *the number of words* requested to be read in the FlashReadOnceCommand.

7.8 ISP command set

Available ISP commands set is limited per life cycle state.

Table 37. ISP command set

Life cycle	Command set
NXP Provisioned	Full commands set:

Table continues on the next page...

Table 37. ISP command set (continued)

Life cycle	Command set
OEM Open OEM Field Return NXP Field Return NXP Closed with Bootloader_API	• ReadMemory • FlashEraseAll • FlashEraseRegion • Reset • WriteMemory • GetProperty • Execute
OEM Closed (ROp1 or ROP2)	Partial commands set • FlashEraseAll • Reset • GetProperty
OEM Closed No Return Bricked	NA

7.8.1 GetProperty command

The GetProperty command is used to query the bootloader about various properties and settings. Each supported property has a unique 32-bit tag associated with it. The tag occupies the first parameter of the command packet. The target returns a GetPropertyResponse packet with the property values for the property identified with the tag in the GetProperty command.

Properties are the defined units of data that can be accessed with the GetProperty or SetProperty commands. Properties may be read-only or read-write. All read-write properties are 32-bit integers, so they can easily be carried in a command parameter.

The 32-bit property tag is the only parameter required for GetProperty command.

Table 38. Parameters for GetProperty Command

Byte #	Parameter
0 - 3	Property tag (which can be received from the GetProperty 0 command)
4 - 7	External Memory Identifier (only applies to GetProperty for external memory, or status identifier if the property tag is equal to 8).

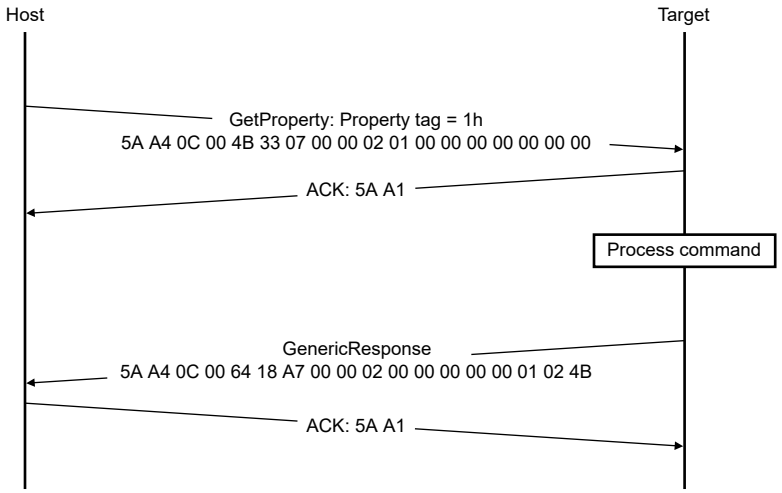


Figure 15. Protocol sequence for GetProperty command

Table 39. GetProperty command packet format (example)

GetProperty	Parameter	Value
Framing packet	Start byte	5Ah
	PacketType	A4h, kFramingPacketType_Command
	Length	0Ch 00h
	Crc16	4Bh 33h

Table 40. GetProperty command packet format (example)

GetProperty	Parameter	Value
Command packet	CommandTag	07h—GetProperty
	Flags	00h
	Reserved	00h
	ParameterCount	02h
	PropertyTag	0000_0001h—CurrentVersion
	Memory ID	0000_0000h—Internal Flash

The GetProperty command has no data phase.

Response: In response to a GetProperty command, the target sends a GetPropertyResponse packet with the response tag set to A7h. The parameter count indicates the number of parameters sent for the property values, with the first parameter showing status code 0, followed by the property value(s). Table 41 shows an example of a GetProperty response packet.

Table 41. GetProperty response packet format (example)

GetPropertyResponse	Parameter	Value
Framing packet	Start byte	5Ah
	PacketType	A4h, kFramingPacketType_Command
	Length	0Ch 00h (12 bytes)
	Crc16	07h 7Ah
Command packet	ResponseTag	A7h
	Flags	00h
	Reserved	00h
	ParameterCount	02h
	Status	0000_0000h
	PropertyValue	0000_014Bh—CurrentVersion

7.8.2 FlashEraseAll command

The FlashEraseAll command performs an erase of the entire flash memory (excluding IFR region). If any flash regions are protected, then the FlashEraseAll command fails and returns an error status code. The command tag for FlashEraseAll command is 01h set in the commandTag field of the command packet.

The FlashEraseAll command requires memory ID. If memory ID is not specified, the internal flash (memory ID = 0) is selected as default.

Table 42. Parameter for FlashEraseAll command

Byte#	Parameter	Value	Description
0-3	Memory ID	000h	Internal Flash
		09h	FlexSPI NOR
		110h	Serial NOR / EEPROM through SPI

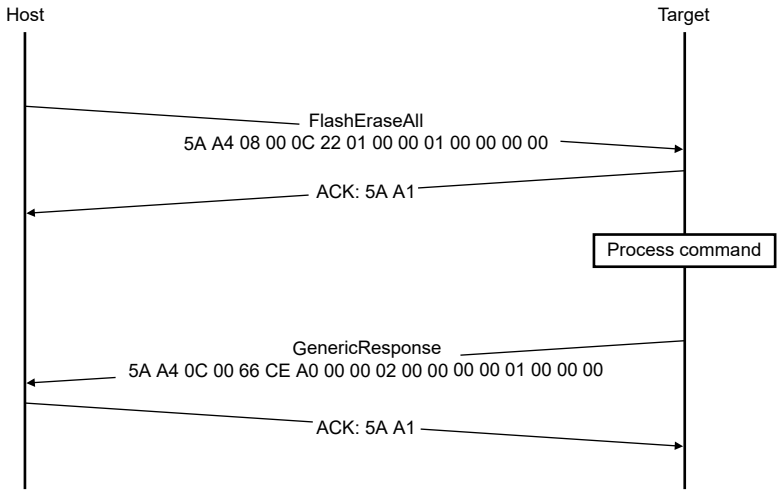


Figure 16. FlashEraseAll command

Table 43. FlashEraseAll command packet format (example)

FlashEraseAll	Parameter	Value
Framing packet	Start byte	5Ah
	PacketType	A4h, kFramingPacketType_Command
	Length	08h 00h
	Crc16	0Ch 22h
Command packet	CommandTag	01h—FlashEraseAll
	Flags	00h
	Reserved	00h
	ParameterCount	01h
	Memory ID	See the above table.

The FlashEraseAll command has no data phase.

Response: The target returns a GenericResponse packet with status code either set to kStatus_Success for successful execution of the command or set to an appropriate error status code.

7.8.3 FlashEraseRegion command

The FlashEraseRegion command performs an erase of one or more sectors of the flash memory.

The start address and number of bytes are the two parameters required for the FlashEraseRegion command. The start address and byte count parameters must be 4-byte aligned ([1:0] = 00), or the FlashEraseRegion command fails and returns kStatus_FlashAlignmentError (101). If the region specified does not fit in the flash memory space, the FlashEraseRegion command fails and returns kStatus_FlashAddressError (102). If any part of the region specified is protected, the FlashEraseRegion command fails and returns kStatus_MemoryRangeInvalid (10200).

Table 44. Parameter for FlashEraseRegion command

Byte #	Parameter
0–3	Start address
4–7	Byte count
8–11	Memory ID

The FlashEraseRegion command has no data phase.

Response: The target returns a GenericResponse packet with one of the following error status codes.

Table 45. FlashEraseRegion response status codes

Status code
kStatus_Success (0).
kStatus_MemoryRangeInvalid (10200).
kStatus_FlashAlignmentError (101).
kStatus_IFlashAddressError (102).
kStatus_FlashAccessError (103).
kStatus_FlashProtectionViolation (104).
kStatus_FlashCommandFailure (105).

7.8.4 ReadMemory command

The ReadMemory command returns the contents of memory at the given address, for a specified number of bytes. This command can read any region of memory that is accessible by the CPU and not protected by security.

The start address and the number of bytes are the two parameters required for the ReadMemory command. The memory ID is optional. Internal memory is selected as the default if memory ID is not specified.

Table 46. Parameter for read memory command

Byte #	Parameter	Description
0–3	Start address	Start address of memory to read from
4–7	Byte count	Number of bytes to read and return to the caller
8–11	Memory ID	Internal or external memory identifier

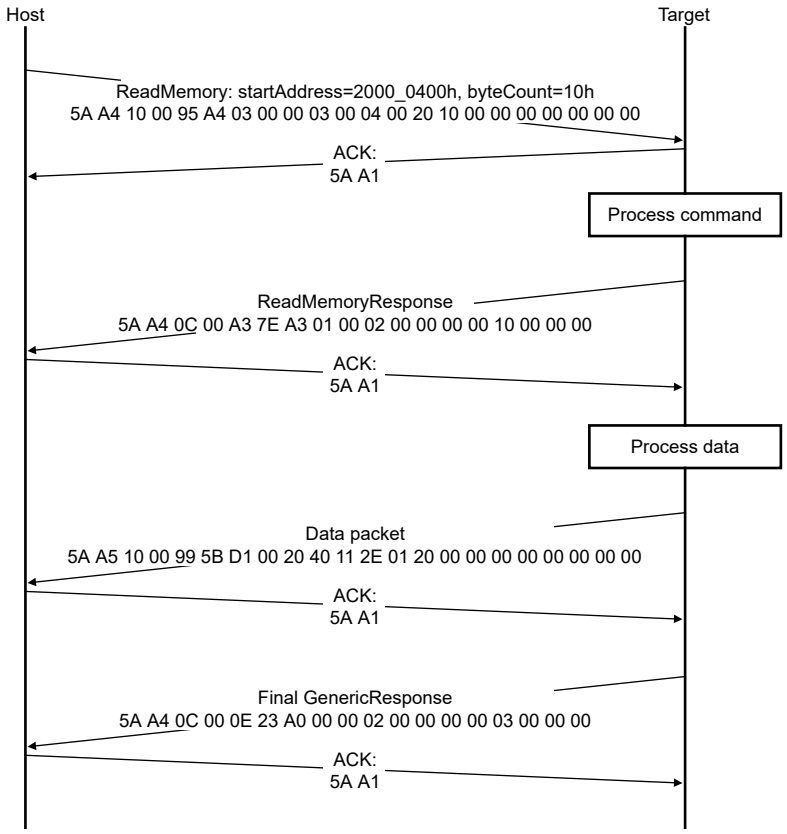


Figure 17. Command sequence for ReadMemory

Table 47. ReadMemory command packet format (example)

ReadMemory	Parameter	Value
Framing packet	Start byte	5Ah
	PacketType	A4h, kFramingPacketType_Command
	Length	10h 00h
	Crc16	F4h 1Bh
Command packet	CommandTag	03h—ReadMemory
	Flags	00h
	Reserved	00h
	ParameterCount	03h
	StartAddress	2000_0400h
	ByteCount	0000_0064h
	Memory ID	0h

Data phase: The ReadMemory command has a data phase. Because the target works in slave mode, the host needs to pull data packets until it has received the number of bytes of data specified in the byteCount parameter of the ReadMemory command.

Response: The target returns a GenericResponse packet with a status code either set to kStatus_Success (upon successful execution of the command) or set to an appropriate error status code.

7.8.5 WriteMemory command

The WriteMemory command writes data that is provided in the data phase to a specified range of bytes in memory (flash memory or RAM). However, if flash memory protection is enabled, then writes to protected sectors fail.

Consider the following information when you write to flash memory:

- First, any flash sector you write to must have been previously erased with a FlashEraseAll or FlashEraseRegion command.
- Writing to flash memory requires the start address to be page-aligned.
- The byte count is rounded up to a page size, and trailing bytes are filled with the flash erase pattern (FFh).
- If the VerifyWrites property is set to true, then writes to flash memory also perform a flash memory verify program operation.

When writing to RAM, the start address does not need to be aligned, and the data is not padded.

The two parameters required for the WriteMemory command are the start address and the number of bytes. The memory ID is optional. Internal memory is selected as the default if memory ID is not specified.

Table 48. Parameters for WriteMemory command

Byte #	Command
0–3	Start address
4–7	Byte count
8–11	Memory ID

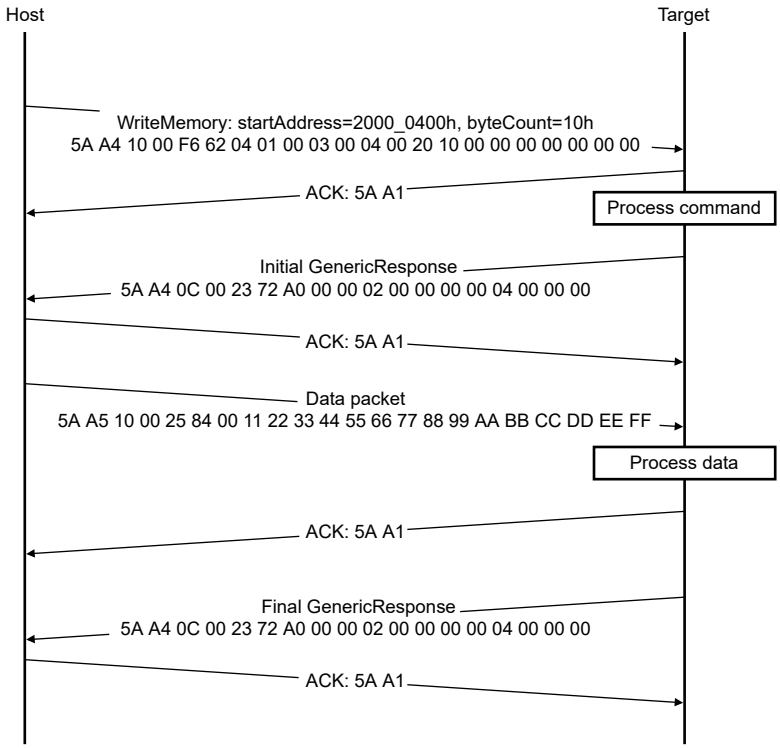


Figure 18. Protocol sequence for WriteMemory command

Table 49. WriteMemory command packet format (example)

WriteMemory	Parameter	Value
Framing packet	Start byte	5Ah
	PacketType	A4h, kFramingPacketType_Command
	Length	10h 00h
	Crc16	97h DDh
Command packet	CommandTag	04—WriteMemory
	Flags	01h
	Reserved	00h
	ParameterCount	03h
	StartAddress	2000_0400h
	ByteCount	0000_0064h
	Memory ID	0h

Data phase: The WriteMemory command has a data phase: the host sends data packets until the target has received the number of bytes of data specified in the byteCount parameter of the WriteMemory command.

Response: The target returns a GenericResponse packet with a status code set to kStatus_Success (upon successful execution of the command), or to an appropriate error status code.

7.8.6 Execute command

The Execute command results in the bootloader setting of each of these items:

- The program counter to the code at the provided jump address
- R0 to the provided argument
- A stack pointer to the provided stack pointer address

Prior to the jump, the system returns to the Reset state.

The Execute command requires these parameters:

- Jump address
- Function argument pointer
- Stack pointer

If the stack pointer is set to zero, the called code is responsible for setting the processor stack pointer before using the stack.

Table 50. Parameters for Execute command

Byte #	Command
0–3	Jump address
4–7	Argument word
8–11	Stack pointer address

The Execute command has no data phase.

Response: Before executing the Execute command, the target validates the parameters and returns a GenericResponse packet with a status code, set to either kStatus_Success or to an appropriate error status code.

7.8.7 Reset command

The Reset command results in the bootloader resetting the chip.

The Reset command requires no parameters.

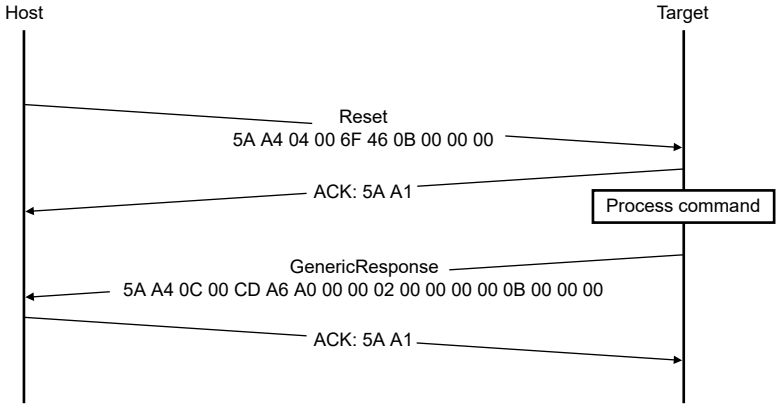


Figure 19. Protocol sequence for Reset command

Table 51. Reset command packet format (example)

Reset	Parameter	Value
Framing packet	Start byte	5Ah
	PacketType	A4h, kFramingPacketType_Command
	Length	04h 00h
	Crc16	6Fh 46h
Command packet	CommandTag	0Bh—reset
	Flags	00h
	Reserved	00h
	ParameterCount	03h

The Reset command has no data phase.

Response: Before resetting the chip, the target returns a GenericResponse packet with status code set to kStatus_Success.

You can also use the Reset command to switch boot from flash memory after successful flash image provisioning via the ROM bootloader. After issuing the Reset command, allow five seconds for the user application to start running from flash memory.

7.9 Get/SetProperty command properties

This section lists the properties of the GetProperty and SetProperty commands.

Get/Set property definitions are provided in this section.

Table 52. Properties used by Get/SetProperty commands

Property	Writable	Tag Value	Size	Description
CurrentVersion	No	01h	4	Current bootloader version.
AvailablePeripherals	No	02h	4	The set of peripherals supported on this chip.
AvailableCommands	No	07h	4	The set of commands supported by the bootloader
Check Status	No	08h	4	Return the status based on specified status identifier 0 – CRC status 32-bit return value for CRC Check 1. – Application CRC check failed 2. – Application CRC check is inactive 3. – Application CRC check is invalid a. – Last Error See the details of last error in later section

Table continues on the next page...

Table 52. Properties used by Get/SetProperty commands (continued)

Property	Writable	Tag Value	Size	Description
MaxPacketSize	No	0Bh	4	Maximum supported packet size for the currently active peripheral interface.
LifeCycleState	No	11h	4	The life cycle of the device. 0x5aa55aa5 – Device is in development life cycle. 0xc33cc33c – Device is in deployment life cycle.
UniqueDevice/UUID	No	12h	16	Unique device identification
Target ROM Version	No	18h	4	The target device identification

7.9.1 CurrentVersion property

The value of this property is a 4-byte structure containing the current version of the bootloader.

Table 53. CurrentVersion property fields

Bit	[31:24]	[23:16]	[15:8]	[7:0]
Field	Name = 'K' (4Bh)	Major version	Major version	Bugfix version

7.9.2 AvailablePeripherals property

The value of this property is the peripherals supported by the bootloader, and the hardware on which it runs.

Table 54. Peripheral bits

Bit	[31:7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Field	Reserved	Reserved	Reserved	USB HID	CAN	SPI slave	I2C slave	LPUART

If the peripheral is available, then the corresponding bit is set in the property value. All reserved bits must = 0.

7.9.3 AvailableCommands property

This property value is a bit field that indicates the commands enabled in the bootloader. Only commands that can be sent from the host to the target are listed in the bit field. Response commands such as GenericResponse are excluded.

The bit number that identifies whether a command is present is the command's tag value minus 1. The value 1 is subtracted from the command tag because the lowest command tag value is 01h. To determine the bit mask for a given command, use this expression: mask = 1 << (tag – 1).

Table 55. Commands bit

Bit	Command
[0]	FlashEraseAll
[1]	FlashEraseRegion
[2]	ReadMemory
[3]	WriteMemory

Table continues on the next page...

Table 55. Commands bit (continued)

Bit	Command
[5:4]	Reserved
[6]	GetProperty
[7]	ReceiveSbFile
[8]	Execute
[9]	Reserved
[10]	Reset
[20:11]	Reserved
[21]	TrustProvisioning
[Others]	Reserved

7.10 Serial Boot on USB path

The bootloader supports In-System Programming using the USB peripheral. The target is implemented as USB-HID device classes.

When transfer data through USB-HID device class, USB-HID does not use framing packets. Instead, the packetization, inherent in the USB protocol itself is used. The ability for the device to NAK Out transfers (until they can be received) provides the required flow control. The built-in CRC of each USB packet provides the required error detection

7.10.1 Device descriptor

The bootloader configures the default USB VID, PID, and strings as follows:

Default VID and PID:

- VID = 1FC9h
- PID = 014fh

Default strings:

- Manufacturer [1] = "NXP SEMICONDUCTOR INC"
- Product [2] = "USB COMPOSITE DEVICE"

You can customize the USB VID, PID, and strings using the CMPA of the flash memory. For example, you can customize the USB VID and PID by writing:

- The new VID to the CMPA location 0100_4034h in flash memory
- The new PID to the usbPid field of the CMPA in flash memory

7.10.2 Endpoints

The HID peripheral uses three endpoints:

- Control (0)
- Interrupt IN (1)
- Interrupt OUT (2)

The interrupt OUT endpoint is optional for HID class devices. The chip bootloader uses the interrupt OUT endpoint as a pipe, where the firmware can NAK send requests from the USB host.

7.10.3 HID Reports

There are four HID reports defined and used by the bootloader USB HID peripheral. The report ID determines the direction and type of packet sent in the report; otherwise, the contents of all reports are the same.

Table 56. HID reports assigned for the bootloader

Report ID	Packet Type	Direction
1	Command	OUT
2	Data	OUT
3	Command	IN
4	Data	IN

Each report has a maximum size of 60 bytes. The maximum payload size is 56 bytes. In addition, there is a 4-byte report header that indicates the length (in bytes) of the payload and report id sent the packet.

NOTE

In the future, the maximum report size may be increased, to support transfers of larger packets. Alternatively, additional reports may be added with larger maximum sizes.

The actual data sent in all of the reports looks like:

Table 57. Data format sent in USB HID packet

Report ID	Data
0	Report ID
1	Padding
2	Packet Length LSB
3	Packet Length MSB
4	Packet[0]
5	Packet[1]
6	Packet[2]
...	…
N+4-1	Packet[N-1]

This data includes the Report ID, which is required if more than one report is defined in the HID report descriptor. The actual data sent and received has a maximum length of 35 bytes. The Packet Length header is written in little-endian format, and it is set to the size (in bytes) of the packet sent in the report. This size does not include the Report ID or the Packet Length header itself. During a data phase, a packet size of 0 indicates a data phase abort request from the receiver.

7.11 Serial boot on LPUART path

The bootloader integrates an autobaud detection algorithm for the UART peripheral, thereby providing flexible baud rate choices.

Autobaud feature: If UART*n* is used to connect to the bootloader, then the UART*n*_RX pin must be kept high and not left floating during the detection phase in order to comply with the autobaud detection algorithm. After the bootloader detects the Ping packet (0x5A 0xA6) on UART*n*_RX, the bootloader firmware executes the autobaud sequence.

If the baudrate is successfully detected, then the bootloader sends a Ping packet response [(0x5A 0xA7), protocol version (4 bytes), protocol version options (2 bytes) and crc16 (2 bytes)] at the detected baudrate. The bootloader then enters a loop, waiting for bootloader commands via the UART peripheral.

NOTE

The data bytes of the ping packet must be sent continuously (with no more than 80 ms between bytes) in a fixed UART transmission mode (8-bit data, no parity bit and 1 stop bit). If the bytes of the ping packet are sent one-by-one with more than 80 ms delay between them, then the autobaud detection algorithm may calculate an incorrect baud rate. In this instance, the autobaud detection state machine should be reset.

Supported baud rates: The baud rate is closely related to the MCU core and system clock frequencies. Typical baud rates supported are 9600, 19200, 38400, 57600, 115200 ,230400 and 460800.

Packet transfer: After autobaud detection succeeds, bootloader communications can take place over the UART peripheral. The following flow charts show:

- How the host detects an ACK from the target.
- How the host detects a ping response from the target.
- How the host detects a command response from the target.

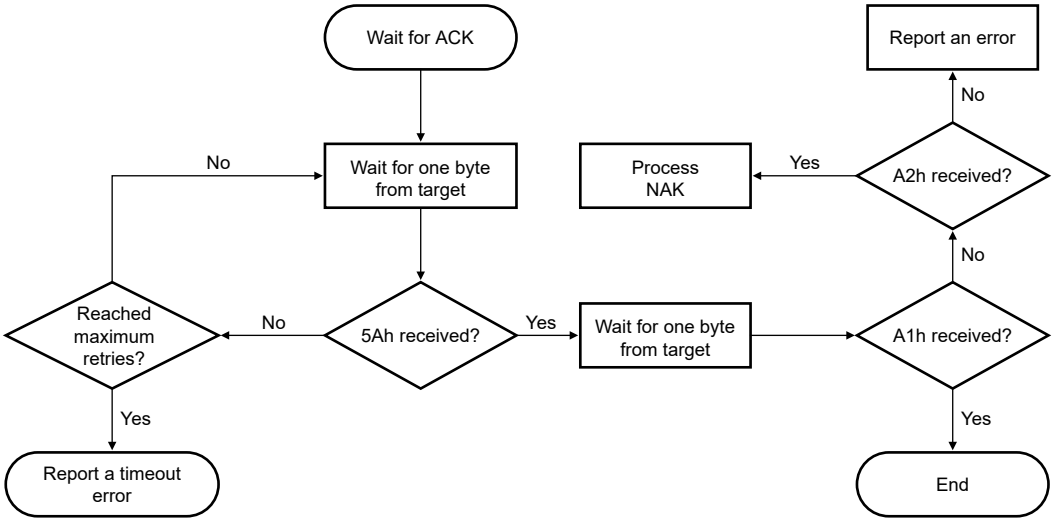


Figure 20. Host reads an ACK from target via LPUART

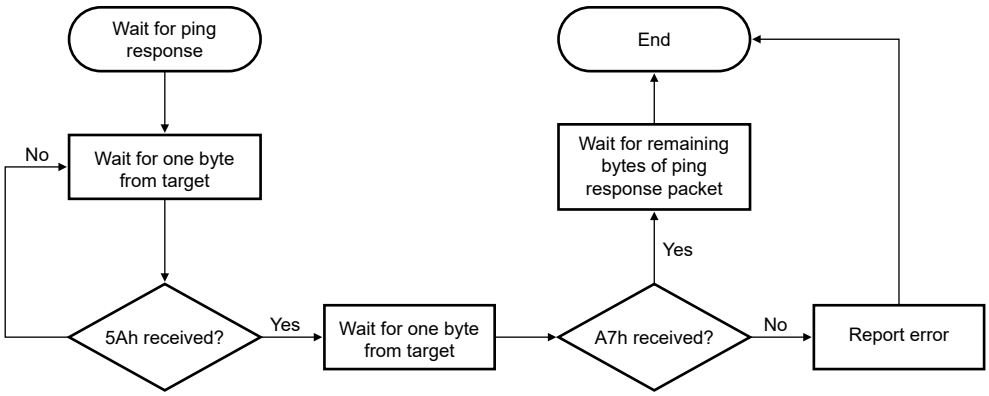


Figure 21. Host reads a ping response from target via LPUART

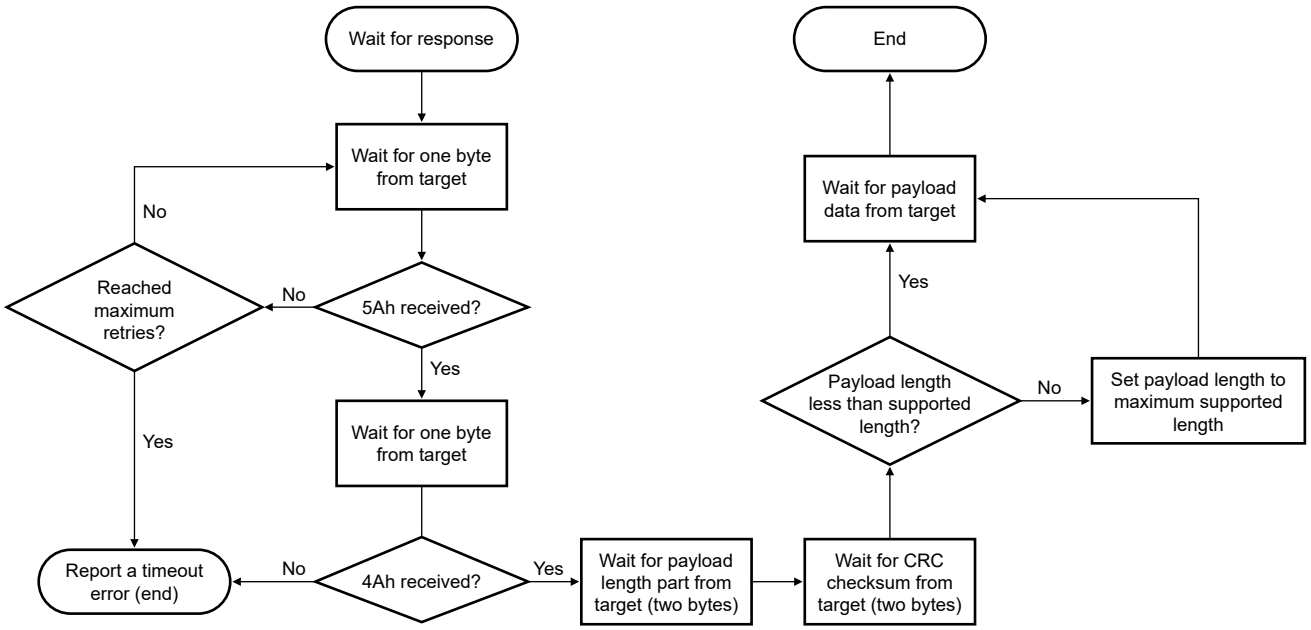


Figure 22. Host reads a command response from target via LPUART

7.12 Serial boot on LPI2C path

The bootloader supports loading data into flash via the I^2C peripheral, where the I^2C peripheral serves as the I^2C slave. A 7-bit slave address is used during the transfer. The bootloader uses 0x10 as the I^2C slave address and supports up to 400 kbit/s as the I^2C baud rate.

The maximum supported I^2C baud rate depends on the core clock frequency when the bootloader is running. The typical baud rate is 400 kbit/s with factory settings.

Because the I^2C peripheral serves as an I^2C slave device, each transfer should be started by the host, and each outgoing packet should be fetched by the host.

- An incoming packet is sent by the host with a selected I^2C slave address and the direction bit is set to write.
- An outgoing packet is read by the host with a selected I^2C slave address and the direction bit is set as read.
- 0x00 is sent as the response to host if the target is busy with processing or preparing data.

The following charts show the communication flow of the host reading the ping and ACK packets, and the corresponding responses from the target.

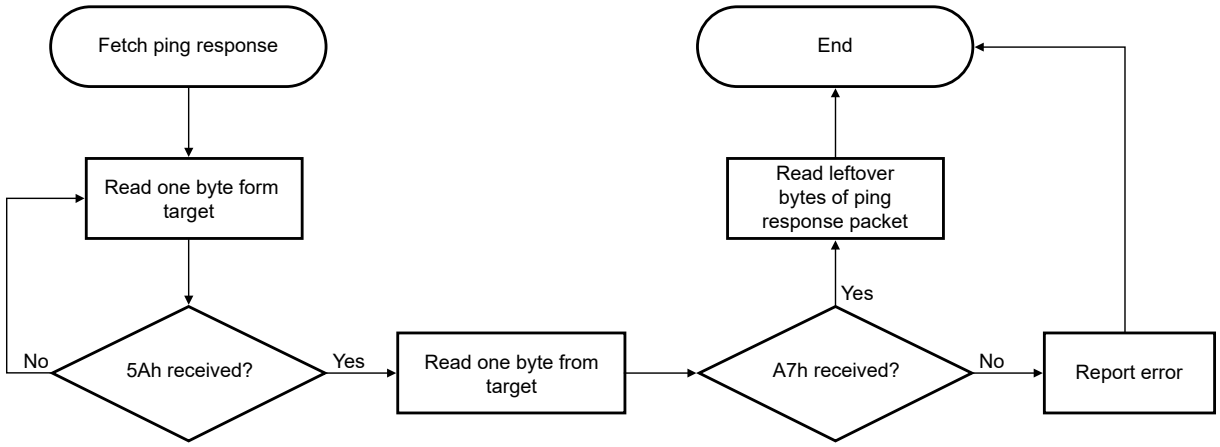


Figure 23. Host reads ping response from target via LPI2C

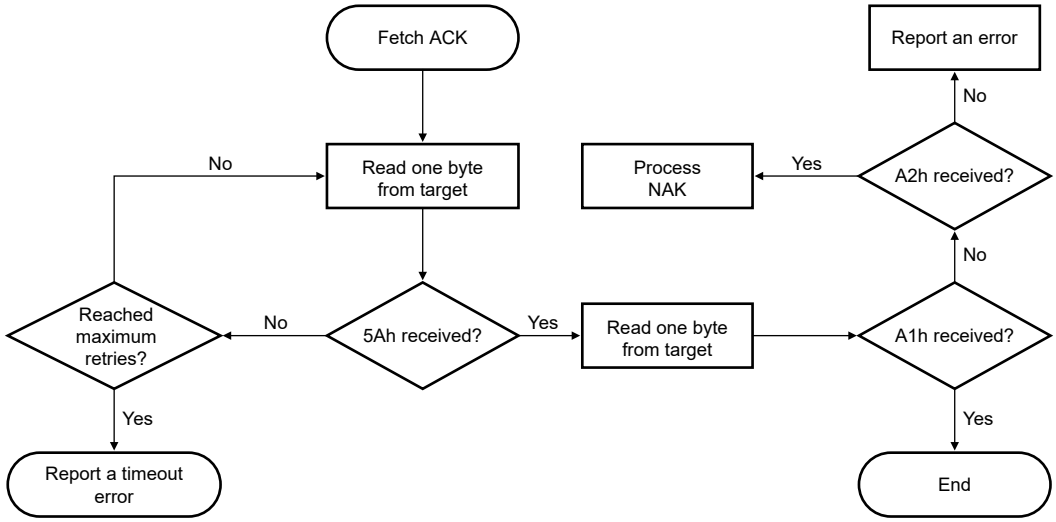


Figure 24. Host reads ACK packet from target via LPI2C

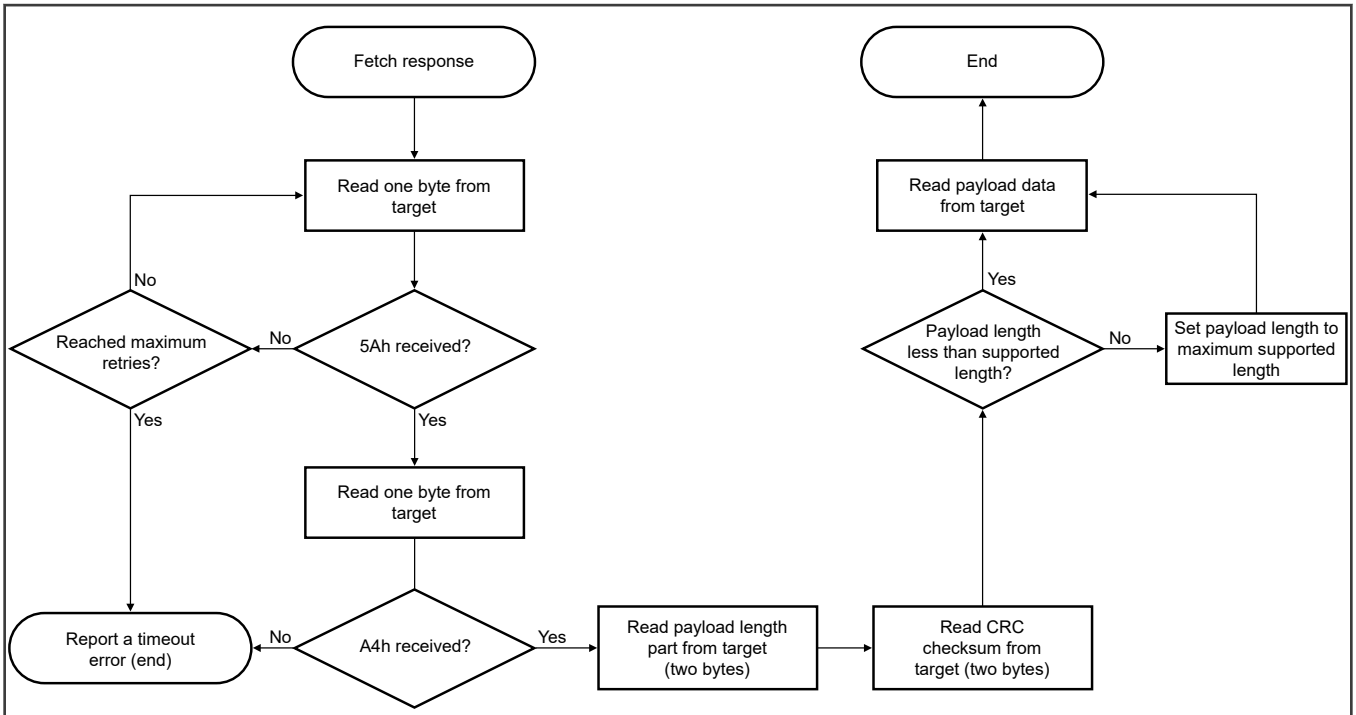


Figure 25. Host reads response from target via LPI2C

7.13 Serial boot on LPSPI path

The bootloader supports loading data into flash via the SPI peripheral, where the SPI peripheral serves as an SPI slave. The SPI transfer should be SPI Mode 3 with 8 data bits.

The maximum supported baud rate of the SPI depends on the core clock frequency when the bootloader is running. The typical baud rate is 2000 kbit/s with the factory settings. The actual baud rate is lower or higher than 2000 kbit/s, depending on the actual value of the core clock.

Because the SPI peripheral serves as an SPI slave device, each transfer should be started by the host, and each outgoing packet should be fetched by the host.

- The transfer on SPI is slightly different from I^2C:
- The host receives 1 byte after it sends out any byte.
- Received bytes should be ignored when the host is sending out bytes to the target
- The host starts reading bytes by sending 0x00s to target

The byte 0x00 is sent as a response to host if the target is under the following conditions:

- Processing incoming packet.
- Preparing outgoing data.
- Received invalid data.

The bootloader also supports the active notification pin (nIRQ pin) to notify the host processor it is busy or ready for new commands/data. See below figure for the typical physical connection between the host and the bootloader device.

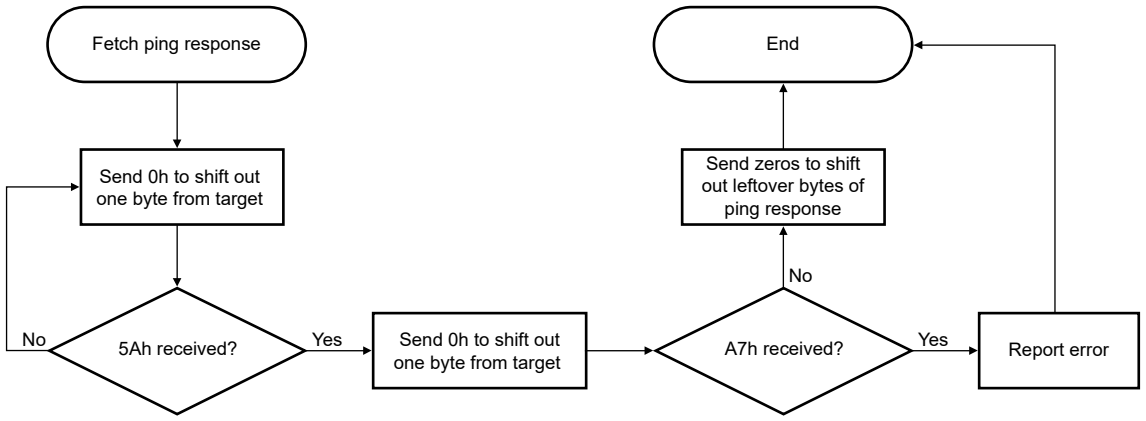


Figure 26. Host reads ping packet form target via LPSPI

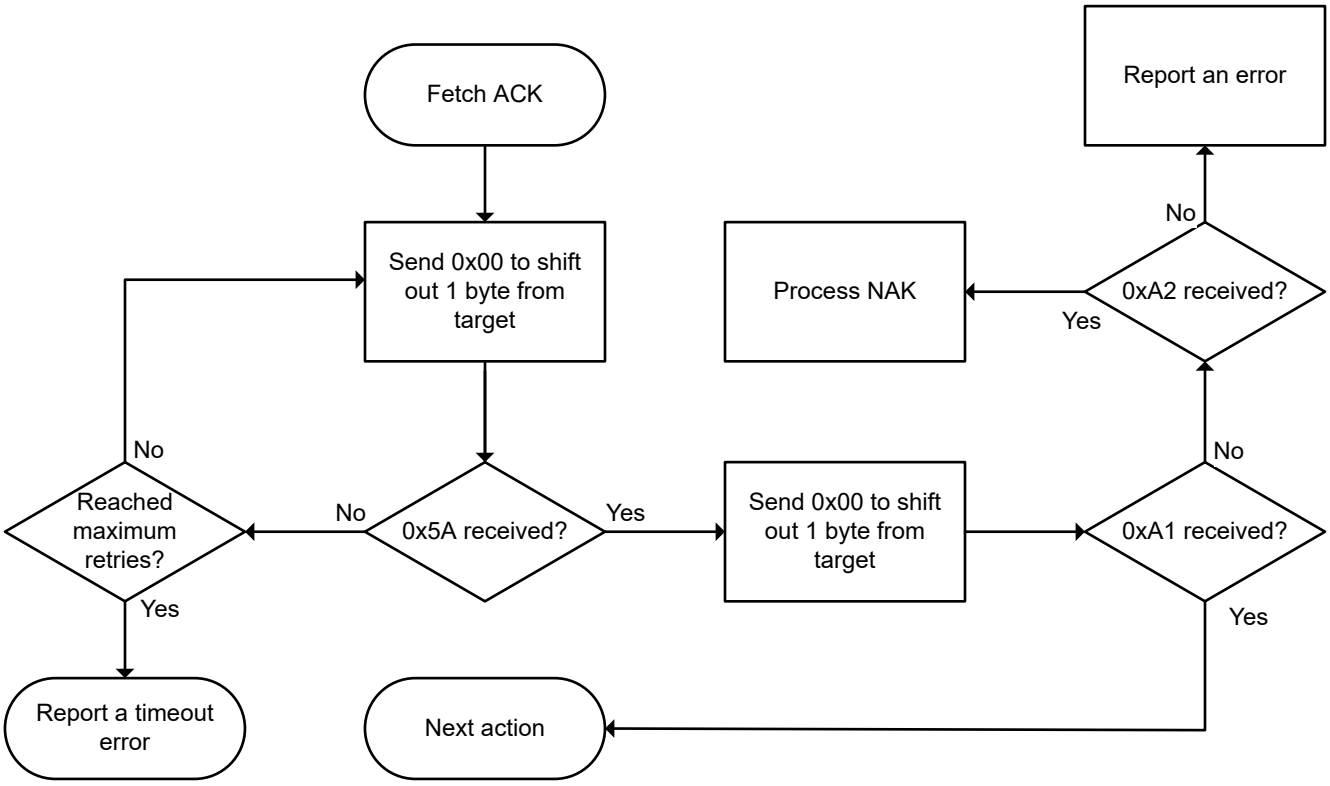


Figure 27. Host reads ACK from target via LPSPI

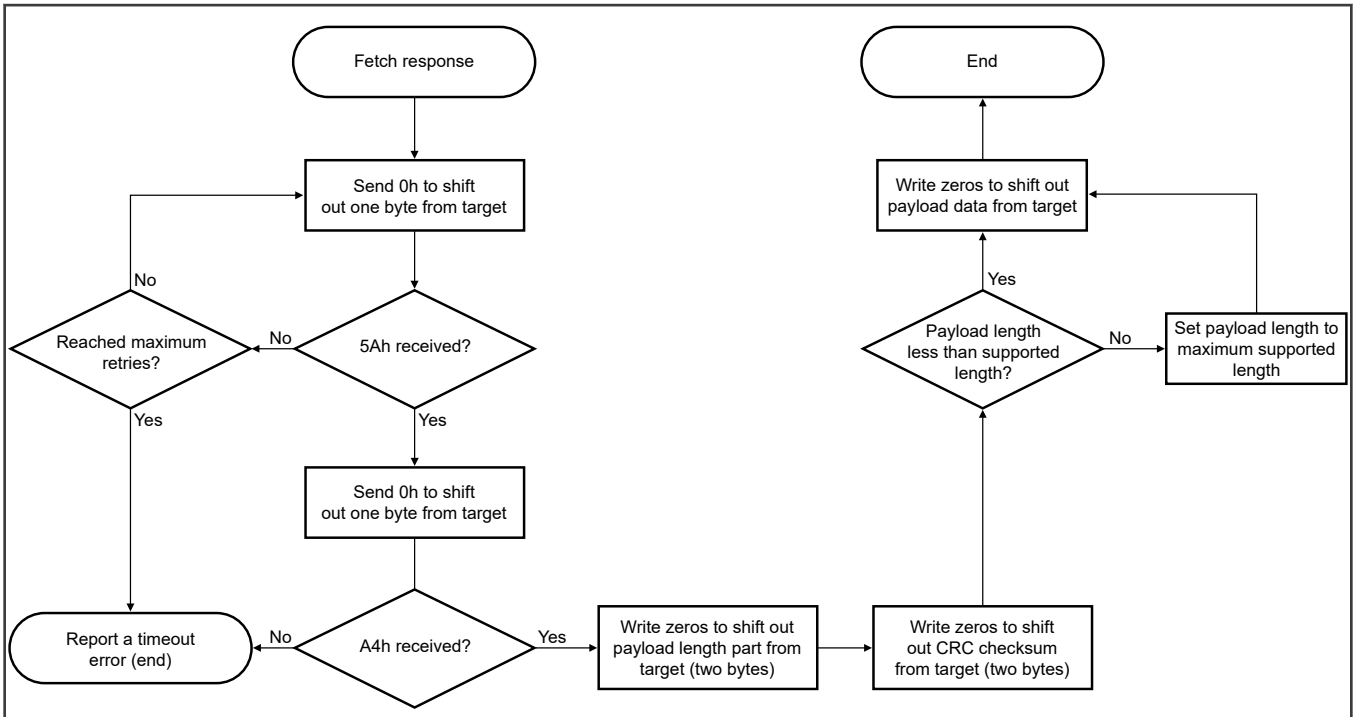


Figure 28. Host reads response from target via LPSPI

7.14 Bootloader status error codes

Table 58 describes the status error codes that the bootloader returns to the host.

Table 58. Bootloader status error codes

Error code	Value	Description
kStatus_Success	0	The operation succeeded without error.
kStatus_Fail	1	The operation failed with a generic error.
kStatus_ReadOnly	2	Requested value is read-only and cannot be changed.
kStatus_OutOfRange	3	Requested value is out of range.
kStatus_InvalidArgument	4	The requested command's argument is undefined.
kStatus_Timeout	5	A timeout occurred.
kStatus_NoTransferInProgress	6	No send in progress
kStatus_FLASH_Success	0	API is executed successfully.
kStatus_FLASH_InvalidArgument	4	An invalid argument was provided.
kStatus_FlashSizeError	100	Not used

Table continues on the next page...

Table 58. Bootloader status error codes (continued)

Error code	Value	Description
kStatus_FlashAlignmentError	101	Address or length does not meet required alignment.
kStatus_FlashAddressError	102	Address or length is outside addressable memory.
kStatus_FlashAccessError	103	The FTFA_FSTAT[ACCERR] bit is set.
kStatus_FlashProtectionViolation	104	The FTFA_FSTAT[FPVIOL] bit is set.
kStatus_FLASH_CommandFailure	105	The FTFA_FSTAT[MGSTAT0] bit is set.
kStatus_FlashUnknownProperty	106	Unknown flash property
kStatus_FlashEraseKeyError	107	The key provided does not match the programmed flash key.
kStatus_FlashRegionExecuteOnly	108	The area of flash is protected as execute-only.
kStatus_FLASH_ExecuteInRamFunctionNotReady	109	Execute-in-RAM function is not available.
kStatus_FLASH_CommandNotSupported	111	Flash API is not supported.
kStatus_FLASH_ReadOnlyProperty	112	The flash property is read-only.
kStatus_FLASH_InvalidPropertyValue	113	The flash property value is out of range.
kStatus_FLASH_InvalidSpeculationOption	114	The option of flash prefetch speculation is invalid.
kStatus_FLASH_EccError	116	A correctable or uncorrectable error occurred during command execution.
kStatus_FLASH_CompareError	117	Destination and source memory contents do not match.
kStatus_FLASH_RegulationLoss	118	A loss of regulation during read
kStatus_FLASH_InvalidWaitStateCycles	119	The wait state cycle set to read/write mode is invalid.
kStatus_FLASH_OutOfDateCfpaPage	132	CFPA page version is out of date.
kStatus_FLASH_BlankIfrPageData	133	Blank page cannot be read.
kStatus_FLASH_EncryptedRegionsEraseNotDoneAtOnce	134	Encrypted flash subregions are not erased at once.
kStatus_FLASH_ProgramVerificationNotAllowed	135	Program verification is not allowed when encryption is enabled.

Table continues on the next page...

Table 58. Bootloader status error codes (continued)

Error code	Value	Description
kStatus_FLASH_HashCheckError	136	Hash check of page data failed.
kStatus_FLASH_SealedFfrRegion	137	The FFR region is sealed.
kStatus_FLASH_FfrRegionWriteBroken	138	The FFR spec region is not allowed to be written discontinuously.
kStatus_FLASH_NmpaAccessNotAllowed	139	The NMPA region is not allowed to be read, written, or erased.
kStatus_FLASH_CmpaCfgDirectEraseNotAllowed	140	The CMPA configuration region is not allowed to be erased directly.
kStatus_FLASH_FfrBankIsLocked	141	The FFR bank region is locked.
kStatus_FLASH_CfpaScratchPageInvalid	148	CFPA scratch page is invalid.
kStatus_FLASH_CfpaVersionRollbackDisallowed	149	CFPA version rollback is not allowed.
kStatus_FLASH_ReadHidingAreaDisallowed	150	Flash hiding read is not allowed.
kStatus_FLASH_ModifyProtectedAreaDisallowed	151	Flash firewall page is locked, and erase and program are not allowed.
kStatus_FLASH_CommandOperationInProgress	152	The flash state is busy, indicating that a flash command is in progress.
kStatus_UnknownCommand	10000	The command is not recognized.
kStatus_SecurityViolation	10001	Security violation occurred when receiving disallowed commands.
kStatus_AbortDataPhase	10002	Sender requested data phase abort
kStatus_Ping	10003	Ping command received from host
kStatus_NoResponse	10004	No response from host
kStatus_NoResponseExpected	10005	Expected no response from the host
kStatus_CommandUnsupported	10006	Unsupported command was received.
kStatusRomLdrSectionOverrun	10100	Reached the end of SB file processing.
kStatusRomLdrSignature	10101	The signature or version is incorrect.
kStatusRomLdrSectionLength	10102	The bootOffset or new section count is out of range.
kStatusRomLdrUnencryptedOnly	10103	The unencrypted image is disabled.

Table continues on the next page...

Table 58. Bootloader status error codes (continued)

Error code	Value	Description
kStatusRomLdrEOFReached	10104	The end of the image file is reached.
kStatusRomLdrChecksum	10105	The checksum of command tag block is invalid.
kStatusRomLdrCrc32Error	10106	The CRC-32 of the data for a load command is incorrect.
kStatusRomLdrUnknownCommand	10107	An unknown command was found in the SB file.
kStatusRomLdrIdNotFound	10108	There was no bootable section found in the SB file.
kStatusRomLdrDataUnderrun	10109	The SB state machine is waiting for more data.
kStatusRomLdrJumpReturned	10110	The function that was jumped to by the SB file has returned.
kStatusRomLdrCallFailed	10111	The call command in SB file failed.
kStatusRomLdrKeyNotFound	10112	A matching key was not found in SB file's key dictionary to unencrypt the section.
kStatusRomLdrSecureOnly	10113	The SB file sent is unencrypted, and security on the target is enabled.
kStatusRomLdrResetReturned	10114	The SB file reset operation has unexpectedly returned.
kStatusRomLdrRollbackBlocked	10115	An image version rollback event is detected.
kStatusRomLdrInvalidSectionMacCount	10116	Invalid section MAC count is detected in SB file.
kStatusRomLdrUnexpectedCommand	10117	The command tag in SB file is unexpected.
kStatusRomLdrBadSBKEK	10118	Bad SBKEK is detected.
kStatusRomLdrPendingJumpCommand	10119	Jump command is pending, and actual jump is implemented in sbloader_finalize().
kStatusMemoryRangeInvalid	10200	The requested address range does not match an entry, or the length extends past the matching entry's end address.
kStatusMemoryReadFailed	10201	Memory read failed.
kStatusMemoryWriteFailed	10202	Memory write failed.

Table continues on the next page...

Table 58. Bootloader status error codes (continued)

Error code	Value	Description
kStatusMemoryCumulativeWrite	10203	Cumulative write occurred due to a write to an unerased flash memory region.
kStatusMemoryNotConfigured	10205	Memory is not configured before access.
kStatusMemoryAlignmentError	10206	Alignment error occurred during access to memory.
kStatusMemoryVerifyFailed	10207	Verifying operation failed after erasing or programming flash memory.
kStatusMemoryWriteProtected	10208	The memory to be written is protected.
kStatusMemoryAddressError	10209	The memory address is invalid or wrong.
kStatusMemoryBlankCheckFailed	10210	Check of the blank memory failed.
kStatusMemoryBlankPageReadDisallowed	10211	The memory is blank, and read command is disallowed.
kStatusMemoryProtectedPageReadDisallowed	10212	The memory is protected, and read command is disallowed.
kStatusMemoryFfrSpecRegionWriteBroken	10213	The write operation to FFR region was broken.
kStatusMemoryUnsupportedCommand	10214	The memory command is not supported
kStatus_UnknownProperty	10300	The requested property value is undefined.
kStatus_ReadOnlyProperty	10301	The requested property value cannot be written.
kStatus_InvalidPropertyValue	10302	The specified property value is invalid.
kStatus_AppCrcCheckPassed	10400	CRC check is valid and passed.
kStatus_AppCrcCheckFailed	10401	CRC check is valid but failed.
kStatus_AppCrcCheckInactive	10402	CRC check is inactive.
kStatus_AppCrcCheckInvalid	10403	CRC check is invalid because the BCA is invalid, or the CRC parameters are unset (all 0xFF bytes).
kStatus_AppCrcCheckOutOfRange	10404	CRC check is valid, but addresses are out of range.
kStatus_RomApiExecuteCompleted	0	ROM successfully processed the whole SB file or boot image.

Table continues on the next page...

Table 58. Bootloader status error codes (continued)

Error code	Value	Description
kStatus_RomApiNeedMoreData	10801	ROM needs more data to continue processing the boot image.
kStatus_RomApiBufferSizeNotEnough	10802	The user buffer is not large enough for use by Kboot during execution.
kStatus_RomApiInvalidBuffer	10803	The user buffer is not acceptable for sbloader or authentication.
kStatus_FLEXSPI_SequenceExecutionTimeout	6000	FLEXSPI sequence execution timeout
kStatus_FLEXSPI_InvalidSequence	6001	Invalid FLEXSPI LUT sequence
kStatus_FLEXSPI_DeviceTimeout	6002	FLEXSPI device timeout
kStatus_FLEXSPINOR_ProgramFail	20100	Page programming failure
kStatus_FLEXSPINOR_EraseSectorFail	20101	Sector erase failure
kStatus_FLEXSPINOR_EraseAllFail	20102	Chip erase failure
kStatus_FLEXSPINOR_WaitTimeout	20103	Execution timeout
kStatus_FlexSPINOR_NotSupported	20104	Page size overflow
kStatus_FlexSPINOR_WriteAlignmentError	20105	Address alignment error
kStatus_FlexSPINOR_CommandFailure	20106	Erase or Program verify error
kStatus_FlexSPINOR_SFDP_NotFound	20107	Timeout occurred during the API call.
kStatus_FLEXSPINOR_Unsupported_SFDP_Version	20108	Unrecognized SFDP version
kStatus_FLEXSPINOR_FLASH_NotFound	20109	Flash memory detection failed.
kStatus_FLEXSPINOR_DTRRead_DummyProbeFailed	20110	DDR failed to read dummy probe.
kStatus_IAP_Success	0	IAP API execution succeeded.
kStatus_IAP_Fail	1	IAP API execution failed.
kStatus_IAP_InvalidArgument	100001	Invalid argument is detected during API execution.
kStatus_IAP_OutOfMemory	100002	The heap size not large enough during API execution.
kStatus_IAP_ReadDisallowed	100003	The read memory operation disallowed during API execution.

Table continues on the next page...

Table 58. Bootloader status error codes (continued)

Error code	Value	Description
kStatus_IAP_CumulativeWrite	100004	The Flash memory region to be programmed is not empty.
kStatus_IAP_EraseFailuire	100005	Erase operation failed.
kStatus_IAP_CommandNotSupported	100006	The specific command is not supported.
kStatus_IAP_MemoryAccessDisabled	100007	Memory access is disabled. **NOTE** Typically occurs on FLEXSPI NOR if it is not configured properly.

NOTE

In UART, I2C, CAN, and SPI ISP modes, the ROM expects posted responses to be read by the host within (20 ms x number of bytes), otherwise the ROM reports the terminate command. Because an ACK from the ROM is only two bytes, the host must read those bytes within 40 ms of the ROM posting.

Chapter 8
Boot ROM

8.1 Overview

ROM on this chip only supports on-chip FLASH image boot. It doesn't support external memory boot.

There are two software on this chip: boot ROM, which is located on ROM memory, and extended bootloader, which is located on IFR0, sector 0 - 3.

Table 59. Boot ROM and bootloader definition

Term	Definition
Boot ROM (software)	In the context of this document, “Boot ROM” is the piece of software present in ROM memory implementing features explained throughout this document.
Extended Bootloader (Software)	In the context of this document, “Extended Bootloader” is the piece of software present in IFR0 memory implementing features explained throughout this document. IFR0 sector 0 - 3 (32 KB).

ROM working with extended bootloader takes responsibility for boot flow.

Boot related parameters are in Customer Manufacturing/Factory Programming Area (CMPA), and ROM will use some settings on this field to update the booting options.

8.2 Features

The main features of the Boot ROM (32 KB) and extended bootloader (32 KB) include:

- Check life cycle, set Read Out Protection (ROP).
- Enable/disable debug port per life cycle.
- Set ROM hidden per life cycle.
- Handling debug mailbox commands, but available commands sets are per life cycle.
- Handling ISP (In-System Programming) commands, but available commands sets are per life cycle.
- Provide Flash API
- Support Flash swap
- Only support ISP boot, not support master boot

8.3 Functional description

8.3.1 Software block diagram

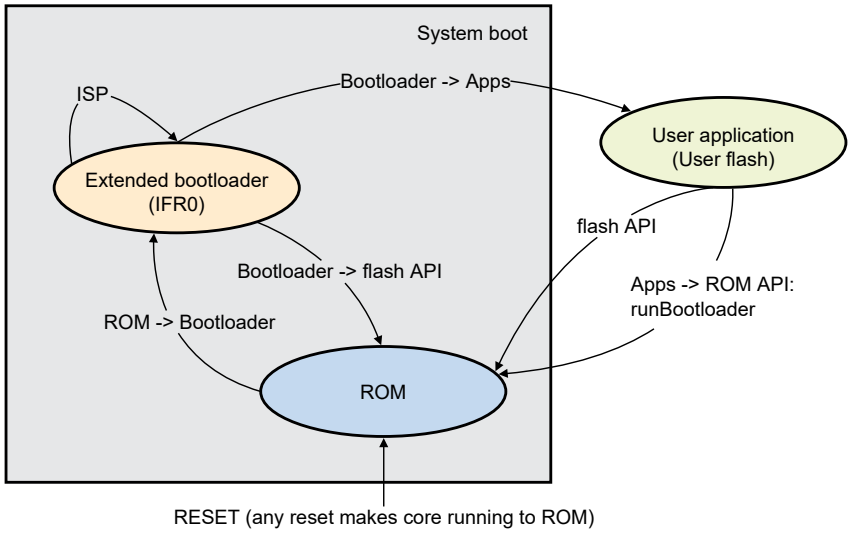


Figure 29. Software block diagram

8.3.2 ROM component

After any reset, including POR reset and warm reset sequence, CPU always runs to boot ROM. ROM does the following:

- Checks life cycle
- Processes mailbox request through debug access port
- Configures the GLIKEY and MBC
- Runs image integrity check based on wakeup source
- Before exit to extended bootloader (on IFR0), hides critical sections
- Provides flash driver API to user

ROM memory starts at 0x1300_0000, with a total size of 32 KB.

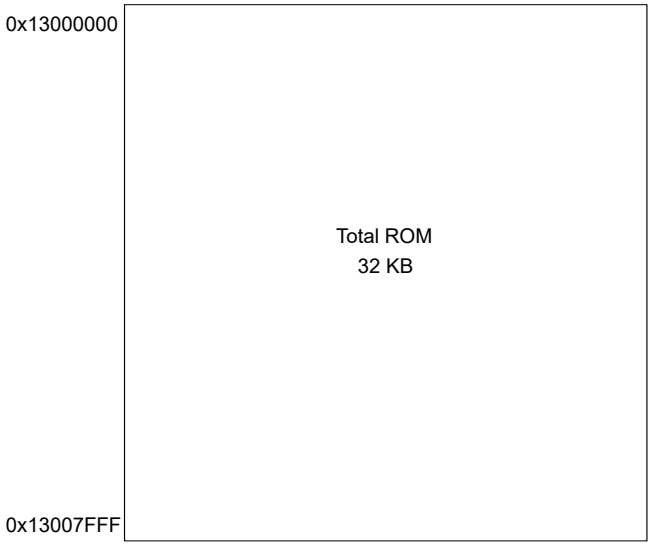


Figure 30. ROM memory

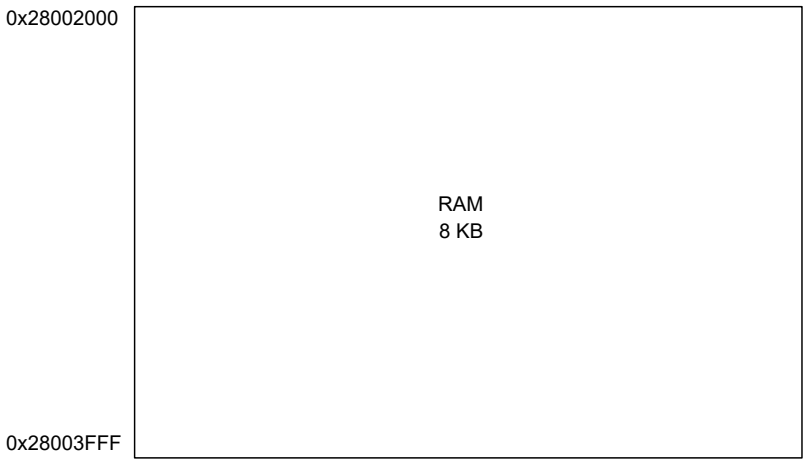


Figure 31. RAM memory used by ROM and extended bootloader

8.3.3 Extended bootloader

The extended bootloader is located in IFR0, with a total size of 32 KB. It checks configuration, enables booting interfaces, and performs ISP commands, among other tasks.

8.3.4 User Application

The user application is located in the main flash region. It can call the flash driver API provided by ROM. Secure boot is not supported by this device, so only a plain image or a CRC32 image is supported by the bootloader. Verification of the CRC32 checksum is performed only if the image indicates a valid image type at offset 0x24 (reserved words of the vector table).

Figure 32 shows the structure of user image.

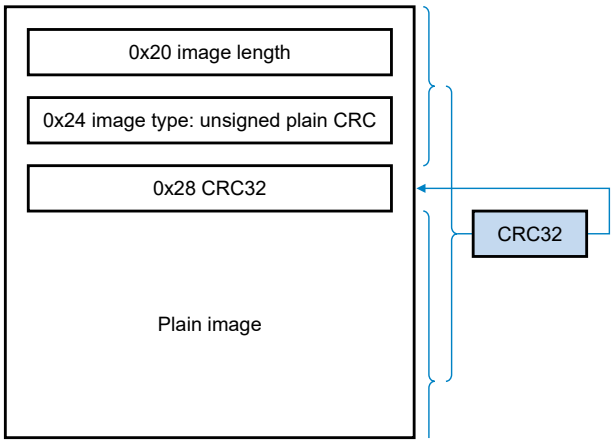


Figure 32. User application component

8.3.5 User image header

Table 60. User image header details

Offset	Size in bytes	Symbol	Description
0x00	4	Initial SP	Stack pointer
0x04	4	Initial PC	The application first execution instruction
0x08	24	Vector table	Cortex-M33 Vector table entries
0x20	4	Image length	The length of the current image
0x24	4	Image type	Image Type: Bit [7:0]: - 0x0, plain image - 0x2, plain image with CRC - 0x5, plain image with CRC Load_to_ram image: image has the non-zero image_load_address and image_length in the image header Other values are reserved Bit[10] – Image version included in Image Type [31:16]: 0 – Image version is not in the Image Type 1 – Image version is included in the Image Type Bit[31:16] – Image version (for on chip flash) when bit[10] is set
0x28	4	offsetToExtendedHeader	Offset to extended header For a CRC image (Image Type = 0x2 or 0x5), this is the CRC checksum value

Table continues on the next page...

Table 60. User image header details (continued)

Offset	Size in bytes	Symbol	Description
0x2C	8	Vector table	Cortex-M33 Vector table entries
0x34	4	ImageExecutionAddress	The execution address of the image Set 0 if image type is XIP Set to actual image execution address if the image type is load to RAM
0x38		Vector table	Cortex-M33 Vector table entries

8.4 Boot pins

Table 61 shows the ISP pin assignment and is the default assignment used by the extended bootloader. The ISP pin assignment can be changed in the CMPA settings.

Table 61. ROM pin assignment

ISP pin	Port pin assignment
ISPMODE_N	P3_29
	P0_6
LPUART ISP pin	
LPUART_RX	P2_3
LPUART_TX	P2_2
LPI2C ISP pin	
LPI2C_SDA	P1_8
LPI2C_SCL	P1_9
LPSPI ISP pin	
LPSPI_SDO	P1_0
LPSPI_SCK	P1_1
LPSPI_SDI	P1_2
LPSPI_PCS	P1_3
USB0_DM	
USB0_DP	
USB0_VBUS_DET	P2_12

ISP_mode pin selection depends on IFR1 configuration field ispmode_pin_sel

- 0: P3_2 is used as ISPMODE_N

8.5 Top-level boot flow

The below chart show the top-level boot flow. The chip always starts ROM after reset.

- By default, ROM runs at 45 MHz.
- Boot ROM checks if there is valid extended bootloader on IFR0 (sector 0 - 3). If the extended bootloader is valid, ROM jumps to the extended bootloader.
- The extended bootloader will determine whether to execute a normal boot or ISP boot based on the ISP_Mode pin or CMPA settings
- The extended bootloader will run a user image validity check.

8.6 CMPA configuration options

The Costumer Manufacturing Area (CMPA) is a region in main flash used to configure the features for boot ROM and the extended bootloader. The CMPA is located in the last 8KB of main flash. The address will change depending on the size of main flash on the device.

```
#define FLASH_SIZE_32KB 0x8000U
#define FLASH_SIZE_8KB 0x2000U
uint32_t cmpa_address = SYSCON->FLASHSIZECFG * FLASH_SIZE_32KB - FLASH_SIZE_8KB;
```

Table 62. CMPA configuration supported by ROM and the extended bootloader

CMPA field	Offset	Bit	Description
Header	0	[31:15]	CMPA header marker, 16'h5963 If the CMPA header marker is cleared and the device is in the Develop life cycle state, MBC Global Access Control 4 will be changed from R/X to R/W/X. Global Access Control 4 is the default permission of the main flash. When the CMPA header marker is written, the permissions of the sectors of the main flash assigned to Global Access Control 4 will be returned to R/W after reset.
BOOT_SPEED	0	[13:12]	Core clock, default 48MHz 00 - 45MHz FRO @1v0 01 - 90MHz FRO @1v1 10 - 180MHz FRO @1v2
ISP_BOOT_IF	0	[6:4]	ISP boot interface. See Table 63
ISP_DM_ENTRY	4	[31:12]	Disable ISP mode entry through debug mailbox command. 01 - ISP entry disabled. 00,10, 11 - ISP entry allowed.
ISP_PIN_ENTRY	4	[11:10]	Disable ISP mode entry through pin assertion. 01 - ISP entry disabled. 00,10, 11 - ISP entry allowed.

Table continues on the next page...

Table 62. CMPA configuration supported by ROM and the extended bootloader (continued)

CMPA field	Offset	Bit	Description
FLASH_REMAP_SIZE	4	[4:0]	Flash remap size. This field should be written to remap field in flash.
BOOT_FAIL_LED	8	[23:16]	Assert on fatal errors during boot. ROM toggles the GPIO pin identified by this field whenever primary boot fails due to fatal errors before locking-up/reset. **NOTE** Use QUICK_SET_/CLR_GPIOx field to set the default level of pin. [4:0] GPIO Pin number [7:5] GPIO port number
POWERDOWN_TIMEOUT_SECS	0x0c	[15:0]	Power down timeout: Timeout value in seconds. When a non-zero value is programmed in this field ROM uses it as idle timeout value to enter power-down state to conserve power.
ISP_UART_CFG: UART_BAUD_RATE	0x10	[31:28]	Baud rate configured during UART ISP mode. 0000 - Auto baud detection. default: Auto baud detection.
ISP_UART_CFG: UART_TX_FUNC_SLOT	0x10	[27:24]	Identifies the pin mux function slot.
ISP_UART_CFG: UART_TX_PIN	0x10	[23:16]	Override default UART TX ISP pin. Identifies the pin to be used as UART_TX pin. [4:0] GPIO Pin number [7:5] GPIO port number
ISP_UART_CFG: UART_ISP_INSTANCE	0x10	[15:12]	Identifies the LPUART instance used for UART ISP mode.
ISP_UART_CFG: UART_RX_FUNC_SLOT	0x10	[11:8]	Identifies the pin mux function slot.
ISP_UART_CFG: UART_RX_PIN	0x10	[7:0]	Override default UART RX ISP pin. Identifies the pin to be used as UART_RX pin. [4:0] GPIO Pin number [7:5] GPIO port number

Table continues on the next page...

Table 62. CMPA configuration supported by ROM and the extended bootloader (continued)

CMPA field	Offset	Bit	Description
ISP_I2C_CFG: I2C_SDA_FUNC_SLOT	0x14	[27:24]	Identifies the pin mux function slot.
ISP_I2C_CFG: I2C_SDA_PIN	0x14	[23:16]	Override default UART TX ISP pin. Identifies the pin to be used as UART_TX pin. [4:0] GPIO Pin number [7:5] GPIO port number
ISP_I2C_CFG: I2C_ISP_INSTANCE	0X14	[15:12]	Identifies the LPI2C instance used for LPI2C ISP mode.
ISP_I2C_CFG: I2C_SCL_FUNC_SLOT	0x14	[11:8]	Identifies the pin mux function slot.
ISP_I2C_CFG: I2C_SCL_PIN	0x14	[7:0]	Override default UART RX ISP pin. Identifies the pin to be used as UART_RX pin. [4:0] GPIO Pin number [7:5] GPIO port number
ISP_SPI_CFG0: SPI_MOSI_FUNC_SLOT	0x1C	[27:24]	Identifies the pin mux function slot.
ISP_SPI_CFG0: SPI_MOSI_PIN	0x1C	[23:16]	Override default SPI_MOSI ISP pin. Identifies the pin to be used as SPI_MOSI pin. [4:0] GPIO Pin number [7:5] GPIO port number
ISP_SPI_CFG0: SPI_ISP_INSTANCE	0x1C	[15:12]	Identifies the LPSPI instance used for LPSPI ISP mode.
ISP_SPI_CFG0: SPI_SCK_FUNC_SLOT	0x1C	[11:8]	Identifies the pin mux function slot.
ISP_SPI_CFG0: SPI_SCK_PIN	0x1C	[7:0]	Override default SPI_SCK ISP pin. Identifies the pin to be used as SPI_SCK pin. [4:0] GPIO Pin number [7:5] GPIO port number
ISP_SPI_CFG1: ISP_SPI_SSEL_X	0x20	[29:28]	SPI chip select number. LPSPI interfaces supports up to four chip selects.

Table continues on the next page...

Table 62. CMPA configuration supported by ROM and the extended bootloader (continued)

CMPA field	Offset	Bit	Description
ISP_SPI_CFG1: SPI_SSEL_FUNC_SLOT	0x20	[27:24]	Identifies the pin mux function slot.
ISP_SPI_CFG1: SPI_SSEL_PIN	0x20	[23:16]	Override default SPI_SSEL ISP pin. Identifies the pin to be used as SPI_SSEL pin. [4:0] GPIO Pin number [7:5] GPIO port number
ISP_SPI_CFG1: SPI_MISO_FUNC_SLOT	0x20	[11:8]	Identifies the pin mux function slot.
ISP_SPI_CFG1: SPI_MISO_PIN	0x20	[7:0]	Override default SPI_MISO ISP pin. Identifies the pin to be used as SPI_MISO pin. [4:0] GPIO Pin number [7:5] GPIO port number
USB Product ID	0x24	[31:16]	USB Product ID
USB Vendor ID	0x24	[15:0]	USB Vendor ID
USBx_VBUS_FUNC_SLOT	0x28	[11:8]	Identifies the pin mux function slot.
USBx_VBUS_PIN	0x28	[7:0]	Override default USB0_VBUS_DETECT ISP pin. Identifies the pin to be used as USB0_VBUS detect pin. [4:0] GPIO Pin number [7:5] GPIO port number
ISP_MISC_CFG: ISP_UART_CUST	0x2C	[31:30]	Use customer defined UART ISP pins. 01: Customer defined. 00, 10, 11: Default ROM defined pins.
ISP_MISC_CFG: ISP_I2C_CUST	0x2C	[29:28]	Use customer defined I2C ISP pins. 01: Customer defined. 00, 10, 11: Default ROM defined pins.
ISP_MISC_CFG: ISP_SPI_CUST	0x2C	[27:26]	Use customer defined SPI ISP pins. 01: Customer defined. 00, 10, 11: Default ROM defined pins.
ISP_MISC_CFG: ISP_USB_CUST	0c2C	[23:22]	Use customer defined GPIO for USB VBUS detect function during ISP mode. 00: Use dedicated VBUS pins.

Table continues on the next page...

Table 62. CMPA configuration supported by ROM and the extended bootloader (continued)

CMPA field	Offset	Bit	Description
			01: Customer defined GPIO for USB0_VBUS detect. 10: Reserved 11: Use VDD_USB for VBUS presence. On board regulator should generate VDD_USB voltage using 5V input for VBUS pin on connector.
I2C_SLAVE_ADDR	0x2C	[7:0]	I2C slave addres A 7-bit Address used for selecting our device on shared I2C bus system. By default ROM uses 0x10 as slave address. If this address conflicts with another slave on board customer could use this field to override the address.
ACL_SEC_7	0x40 ~ 0x5C	[30:28]	ACL_SEC_7, 15, 23, 31, 39, 47, 55, 63, 71, 79, 87, 95, 103, 111, 119, 127
ACL_SEC_6		[26:24]	ACL_SEC_6, 14, 22, 30, 38, 46, 54, 62, 70, 78, 86, 94, 102, 110, 118, 126
ACL_SEC_5		[22:20]	ACL_SEC_5, 13, 21, 29, 37, 45, 53, 61, 69, 77, 85, 93, 101, 109, 117, 125
ACL_SEC_4		[18:16]	ACL_SEC_4, 12, 20, 28, 36, 44, 52, 60, 68, 76, 84, 92, 100, 108, 116, 124
ACL_SEC_3		[14:12]	ACL_SEC_3, 11, 19, 27, 35, 43, 51, 59, 67, 75, 83, 91, 99, 107, 115, 123
ACL_SEC_2		[10:8]	ACL_SEC_2, 10, 18, 26, 34, 42, 50, 58, 66, 74, 82, 90, 98, 106, 114, 122
ACL_SEC_1		[6:4]	ACL_SEC_1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105, 113, 121
ACL_SEC_0		[2:0]	ACL_SEC_0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120
QUICK_SET_GPIO_0	0x80	[31:0]	Drive GPIO 0 port pins high after reset. Each bit corresponds to the pin in GPIO port 0. When set ROM drives the corresponding pin high as soon as possible. By default most pins come-up as tri-stated inputs. This feature allows customer to specify active drive pins soon after reset instead of waiting till complete boot.
QUICK_CLR_GPIO_0	0x84	[31:0]	Drive GPIO 0 port pins low. See description of QUICK_SET_GPIO_0 field.

Table continues on the next page...

Table 62. CMPA configuration supported by ROM and the extended bootloader (continued)

CMPA field	Offset	Bit	Description
QUICK_SET_GPIO_1	0x88	[31:0]	Drive GPIO 1 port pins high after reset. Each bit corresponds to the pin in GPIO port 1. When set ROM drives the corresponding pin high as soon as possible. By default most pins come-up as tri-stated inputs. This feature allows customer to specify active drive pins soon after reset instead of waiting till complete boot.
QUICK_CLR_GPIO_1	0x8C	[31:0]	Drive GPIO 1 port pins low. See description of QUICK_SET_GPIO_1 field.
QUICK_SET_GPIO_2	0x90	[31:0]	Drive GPIO 2 port pins high after reset. Each bit corresponds to the pin in GPIO port 2. When set ROM drives the corresponding pin high as soon as possible. By default most pins come-up as tri-stated inputs. This feature allows customer to specify active drive pins soon after reset instead of waiting till complete boot.
QUICK_CLR_GPIO_2	0x94	[31:0]	Drive GPIO 2 port pins low. See description of QUICK_SET_GPIO_2 field.
QUICK_SET_GPIO_3	0x98	[31:0]	Drive GPIO 3 port pins high after reset. Each bit corresponds to the pin in GPIO port 3. When set ROM drives the corresponding pin high as soon as possible. By default most pins come-up as tri-stated inputs. This feature allows customer to specify active drive pins soon after reset instead of waiting till complete boot.
QUICK_CLR_GPIO_3	0x9C	[31:0]	rive GPIO 3 port pins low. See description of QUICK_SET_GPIO_3 field.
QUICK_SET_GPIO_4	0xA0	[31:0]	Drive GPIO 4 port pins high after reset. Each bit corresponds to the pin in GPIO port 4. When set ROM drives the corresponding pin high as soon as possible. By default most pins come-up as tri-stated inputs. This feature allows customer to specify active drive pins soon after reset instead of waiting till complete boot.
QUICK_CLR_GPIO_4	0xA4	[31:0]	Drive GPIO 4 port pins low. See description of QUICK_SET_GPIO_4 field.
ROP_STATE	0xB0	[31:0]	This field allows the device life cycle to be advanced from the Develop life cycle.
SBL_PROT	0XB4	[31:0]	If this field contains a value other than 0x0000_0000 or 0xFFFF_FFFF, Debug Mailbox commands and ISP

Table continues on the next page...

Table 62. CMPA configuration supported by ROM and the extended bootloader (continued)

CMPA field	Offset	Bit	Description
			commands will not be allowed at Develop 2 and In-Field life cycles.
SBL_START_ADDR	0XB8	[31:0]	At the end of boot ROM execution, if the user does not wish boot ROM to jump at address 0x0000_0000 (the default jump address) of the main flash, this field needs to be programmed with a valid flash jump address.
ROP_STATE_DP	0XC0	[31:0]	Duplicate of ROP_STATE_DP
SBL_PROT_DP	0xC4	[31:0]	Duplicate of SBL_PROT_DP
SBL_START_ADDR_DP	0xC8	[31:0]	Duplicate of SBL_START_ADDR_DP

8.6.1 ISP boot mode

Depending on the values of ISP mode in the CMPA and ISP pin, the bootloader decides whether to enter normal boot flow or entering ISP mode.

Table 63. ISP mode based on ISP_BOOT_IF CMPA bit field (word0[6:4])

ISP boot interface	CMPA field value	Description
Auto ISP	3’b000	Bootloader probes the active peripheral form ton the below serial interfaces to download image: LPUART, USB0, LPI2C and LPSPI (however it depends on extended bootloader choice)
UART ISP	3’b001	LPUART is used to download the image
SPI Slave ISP	3’b010	LPSPI is used to download the image
I2C Slave ISP	3’011	LPI2C is used to download the image
USB0 HID	3’100	USB0 is used to download the image
Default	3’111	

8.6.2 OEM secondary bootloader

OEMs have the option to flash their own secondary bootloader. Enabling this feature is done by writing the address of the secondary bootloader to both the SBL_START_ADDR and SBL_START_ADDR_DP fields within the CMPA. The address of the secondary bootloader must be 8KB aligned. The extended bootloader will boot the secondary bootloader by default after this feature is enabled.

Secondary bootloader protection can be enabled by setting both the SBL_PROT and SBL_PROT_DP fields to an equivalent non-zero and non-cleared (0xFFFFFFFF) value. Enabling protection will disable the Debug Mailbox and ISP within Boot ROM and the extended bootloader while in In-field life cycle states. This protects from both writing to the secondary bootloader, and from a mass erase, which would erase the secondary bootloader from flash.

Figure 33 illustrates the flow of OEM secondary bootloadr.

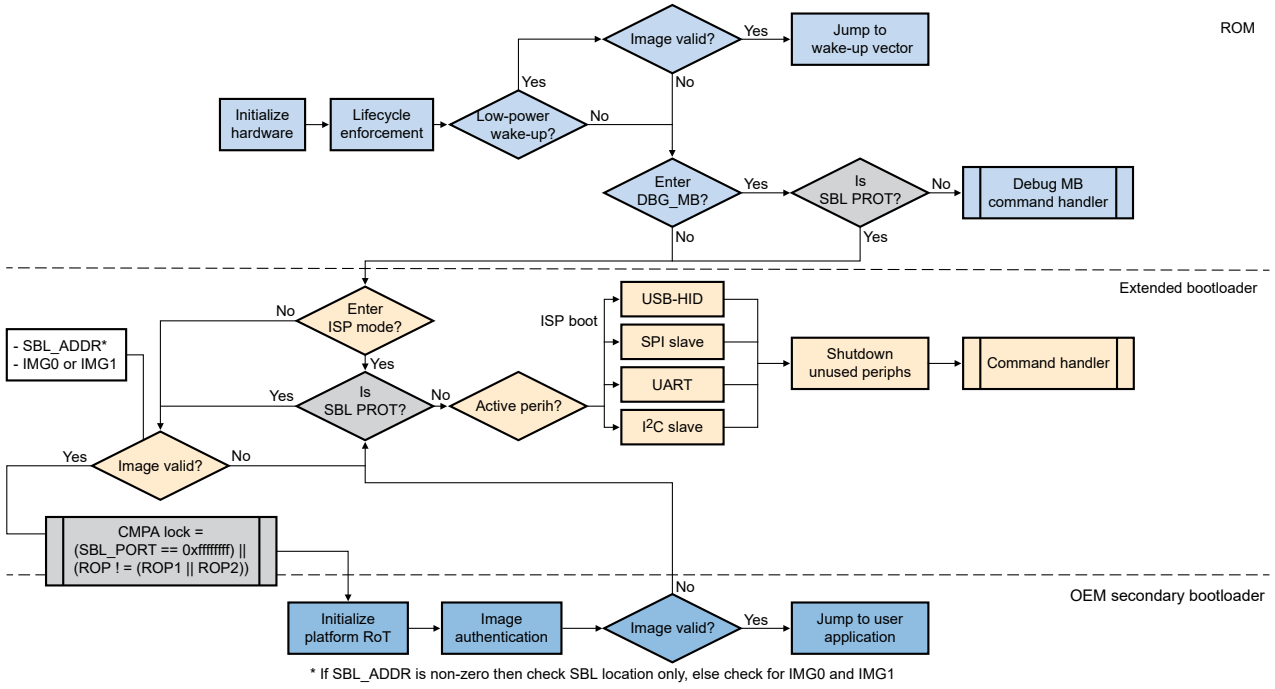


Figure 33. OEM secondary bootloader

The boot process involves multiple integrity checks and hand-offs:

1. Boot ROM:
   - Performs integrity check of the extended boot loader (in NXP-locked flash).
   - Transfers control to the extended boot loader.
2. Extended bootloader:
   - Validates the OEM secondary bootloader using the CMPA configuration.
   - Transfers control to the OEM secondary bootloader.
3. OEM secondary bootloader installed on the device using SI:
   - Authenticates the field-updatable application image using cryptographic keys.
   - May implement custom debug authentication using the Debug Mailbox module.

NOTE

Both the extended bootloader and SI regions are write-locked during manufacturing, ensuring immutability in the field.

8.7 All DM-AP commands

Table 64 lists all DM-AP commands supported by this device.

Table 64. DM-AP commands

Command	ID	Parameter/Response	Description
Start DM-AP (Legacy command)	0x01	Parameters: None Response: 32-bit status	Cause the device to enter DM-AP command mode.

Table continues on the next page...

Table 64. DM-AP commands (continued)

Command	ID	Parameter/Response	Description
			This must be done prior to sending other commands.
Get ROP Level	0x02	Parameters: None Response: 32-bit status	Return the ROP level value.
Bulk Erase (Legacy command)	0x03	Parameters: None Response: 32-bit status	Erase the entire on-chip flash memory and IFR0, sector 0.
Exit DM-AP (Legacy command)	0x04	Parameters: None Response: 32-bit status	Cause the device to exit DM-AP command mode. The device returns to normal mode.
Enter ISP Mode (For Extended bootloader)	0x05	Parameters: dataWordCount: 0x1 data[0]: ISP mode enum. Same as IAP API. Response: 32-bit status	Enter specified ISP mode.
Set FA Mode	0x06	Parameters: dataWordCount: 0xFE data[]: FA Request Response: 32-bit status	Set the part permanently in "Fault Analysis" mode to return to NXP factory.
Start Debug Session	0x07	Parameters: None Response: 32-bit status	This command is used to indicate ROM the intention of connecting debugger. ROM enables debug access and enters while(1) loop.
Write words to flash (New)	0x09	Parameters: dataWordCount: 0x5 data[0]: FlashAddress data[4]: data to be written Response: 32-bit status	Write four words (128 bits) to flash. Flash address must be 16-bit aligned.
Read words from flash (New)	0x0A	Parameters: dataWordCount: 0x5 data[0]: FlashAddress Response: 32-bit status data[4]: Data out	Read four words from flash. Flash address must be 16-bit aligned.
Erase one Sector (New)	0x0B	Parameters: dataWordCount: 0x1 data[0]: flash page index Response: 32-bit status	Erase one flash sector.

Chapter 9
Secure Installer

9.1 Overview

The Secure Installer (SI) is a library which provides an interface for SB3.1 file handling and trust provisioning services (SB3.1 creation, key store/generation/rewrapping).

The SI has the following features:

- It's pre-installed in the user flash during manufacturing at NXP.
- It's not readable, writable, or executable when user code is executing.
- It takes up 56 kB of the user flash.
- It can be erased upon request so the 56 KB of the user flash can be reclaimed for customer use.
- It supports 2 flash sizes:
  - 512 KB (SI is placed at [0x70000 – 0x7Efff].)
  - 1 MB (SI is placed at [0xF0000 – 0xFEFFF].)

9.2 How to erase the SI

Below is a table of fields in the CMPA that are related to the SI.

Table 65. CMPA fields related to SI

Name	Size (bytes)	Offset in CMPA (bytes)	Description
CMPA_DELETE_INSTALLER	4	0xE0	This field is used for the customer to signal the bootloader to erase the SI.
CMPA_ERASE_TOKEN	16	0xD0	This field is used to let the bootloader know that the SI is erased. It is programmed by the bootloader after the bootloader erases the SI. This token is device unique.

Below are the steps to delete the SI.

1. Erase the CMPA_ERASE_TOKEN. This step is important, see Hazards.
2. Program the CMPA_DELETE_INSTALLER field with the DELETE_INSTALLER_MAGIC value (0x4027FF65).
3. Reset the device. This will cause the SI to be erased, and the CMPA_ERASE_TOKEN will be programmed.

9.3 SB file handling

9.3.1 Overview

SB file handling is done through the receive-sb-file command. This device only supports SB3.1. There are four supported SB commands: Erase, Load, Execute, and LoadKeyBlob.

9.3.2 SB file creation

Below are the steps to create the SB3.1 file:

1. A secure session should be established with the secure installer by the secure provisioning SDK (SPSDK) tool using OemGenMasterShare command.
   - This creates the seed for the key generation. The user should choose a randomly generated input share (`oemShareInputAddr` parameter) to diversify the seed. The encrypted image generated using two different OEM Share Inputs will be different even if the contents and randomized parameters are otherwise identical.
   - The command provides a session handle also referred as OEM blob stored at address location specified by `oemEncShareOutputAddr` parameter.
   - All sub-sequent command should use this OEM blob for proper file generation.
2. For encrypting and CMAC signing the SB3 image the root key called customer master key (CUSTMKSK), is generated using HsmGenKey command.
   - From CUSTMKSK, block encryption keys, key wrapping key and MAC signing keys are derived.
   - The parameter keyProp defines the security levels of keys generated for encryption of the file.
     - 0 for 128-bit security: When this option is selected following algorithms are used to create the SB3.1 file.
       - AES with 128-bit key used or encryption
       - SHA-256 for hashing
       - CMAC with AES-128 key for signing
     - 1 for 256-bit security:
       - AES with 256-bit key used or encryption
       - SHA-384 for hashing
       - CMAC with AES-256 key for signing
3. Using SPSDK tool, assemble the SB command script payload file. The payload file contains a sequence of SB command and their parameters described in SB commands.
4. Create the SB3.1 header using the OEM blob created in step 1.
5. Segment the payload file into multiple blocks which are then encrypted using HsmEncBlk command.
   - The command derives a separate key per block using CMAC based key derivation function.
6. Daisy chain the blocks by adding hash digest of the next encrypted block.
7. Update the SB3.1 header with hash digest of first and sign it using HsmEncSign command.

9.3.3 SB file format

Figure 34 illustrates the format of the SB3.1 file.

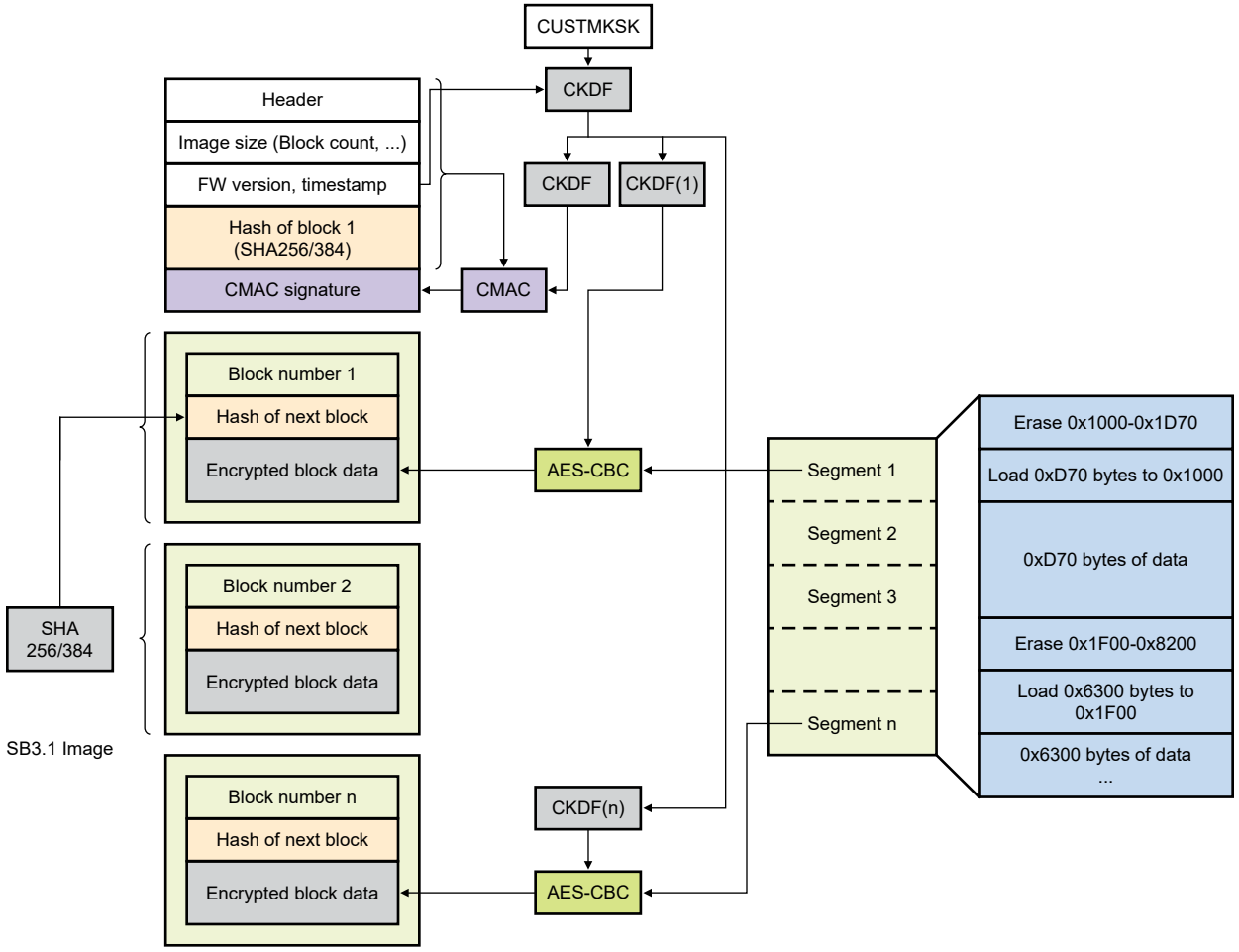


Figure 34. SB3.1 file format

9.3.4 SB file contents

The SB3.1 file must include:

- SI image
- Optional application image
- Optional Key blobs containing product common application AES keys
  - These key blobs are created using 5.4.3 HsmStoreKey commands.
  - Secure installer unwraps these key blobs and wraps them back with device-unique master key (DUK). The DUK is loaded into write-only key slots 6 and 7 of the Secure Generic Interface (SGI).
- CMPA configuration data, including:
  - ACL_SEC_0 to ACL_SEC_7: Set to 0x2 (MBC0_MEMN_GLBAC2) to permanently write-lock flash sectors used by the SBL.
  - ROP_STATE & ROP_STATE_DP:
    - 0xEEBA_04C3 (ROP_LEVEL1): Debug registers unlocked before SBL execution
    - 0x4939_8D8B (ROP_LEVEL2): Debug registers locked before SBL execution
  - SBL_PROT & SBL_PROT_DP: Set to a non-default value (e.g., 0xA5A5_A5A5) to protect the SBL.

— SBL_START_ADDR & SBL_START_ADDR_DP: Set to the flash address where the SBL is programmed.

9.3.5 SB commands

9.3.5.1 Erase

This command erases a memory region. This command only allows erasing main flash and RAM memory. This excludes the SI, RAM X0, and RAM X1.

Parameters:

Range Header Items	Description
u32 tag	Header tag for synchronization 0x55AAAA55. This tag is valid for all Range Headers.
u32 startAddress	The start of the region being erased.
u32 length	The size of the region.
u32 cmd	0x01 denotes the Erase command.

Extended Range Header Items	Description
u32 MemoryId	0x0
u32 reserved0	0x0
u32 reserved1	0x0
u32 reserved2	0x0

9.3.5.2 Load

This command loads data to the specified memory region. This command only allows loading to main flash and RAM. This excludes the SI, RAM X0, and RAM X1.

Parameters:

Range Header Items	Description
u32 tag	0x55aaaa55
u32 startAddress	The start of the region where data is to be loaded.
u32 length	The size of the data being loaded.
u32 cmd	0x02 denotes the Load command.

Extended Range Header Items	Description
u32 memoryId	0x0
u32 reserved0	0x0
u32 reserved1	0x0
u32 reserved2	0x0

This block represents the data being loaded, and it is placed immediately after the extended range header.

9.3.5.3 Execute

This command executes from the address specified as an argument. Any commands following this will be ignored because the bootloader will jump to the image and will no longer be in ISP.

Parameters:

Range Header Items	Description
u32 tag	0x55aaaa55
u32 startAddress	The start of where the image is located.
u32 reserved0	0x0
u32 reserved1	0x0

9.3.5.4 LoadKeyBlob

This command unwraps the key blob placed in the data section of this command, and rewraps it and stores it in the CMPA specified by the offset argument.

Parameters:

Range Header Items	Description
U32 tag	0x55aaaa55
U16 offset	This offset is with respect to the start of the key-store region in the CMPA, which is located at offset 0x100 with respect to the base of the CMPA. For example, if you set this argument to 0x0, the key blob would be stored at CMPA_BASE[0x100]. The size of this region is 0x1f00.
U16 keyWrapId	The only key that is supported on this device is customer key encryption key: NXP_CUST_KEK_EXT_SK (17).
U32 length	The length of the keyblob.
U32 cmd	0x0A denotes the LoadKeyBlob command.

This is the key blob data, and it should be place immediately following the range header.

9.4 Trust provisioning

The SI supports trust provisioning via SB3.1 files using commands defined below. These commands enable secure encryption and signing of the SI image.

9.4.1 OemGenMasterShare

This command is used by OEM to provide its share to create initial trust provisioning keys. The output is placed inside of the header of the SB3.1 file in the OEM blob field.

Syntax:

```
blhost -p <COM> -- trust-provisioning oem_gen_master_share
<oemShareInputAddr>
<oemShareInputSize>
```

```
<oemEncShareOutputAddr>
<oemEncShareOutputSize>
<oemEncMasterShareOutputAddr>
<oemEncMasterShareOutputSize>
<oemCustCertPukOutputAddr>
<oemCustCustCertPukOutputSize>
```

Parameters:

Name	Description
oemShareInputAddr	The address of where the random seed is on the device.
oemShareInputSize	The size of the random seed.
oemEncShareOutputAddr	The address of where the OEM blob will be written. The OEM blob is stored in the SB header, and is used for block encryption.
oemEncShareOutputSize	The size of the OEM blob.
oemEncMasterShareOutputAddr	This parameter is not used on this device.
oemEncMasterShareOutputSize	This parameter is not used on this device.
oemCustCertPukOutputsAddr	This parameter is not used on this device.
oemCustCertPukOutputSize	This parameter is not used on this device.

9.4.2 HsmGenKey

This command generates the key used for encrypting and signing.

Syntax:

```
blhost -p <COM> -- trust-provisioning hsm_gen_key
<keyType>
<keyProp>
<keyBlobOutputAddr>
<keyBlobOutputSize>
<ecdsaPukOutputAddr>
<ecdsaPukOutputSize>
```

Parameters:

Name	Description
keyType	There is only one supported key. • CUSTMKSK (customer master key)
keyProp	This chip only supports the following size property. • 0 for 128-bit key • 1 for 256-bit key
keyBlobOutputAddr	The address of the key blob. This blob is wrapped using RFC3394 command.

Table continues on the next page...

Table continued from the previous page...

Name	Description
ecdsaPukOutputAddr	Ignored
ecdsaPukOutputSize	Ignored

9.4.3 HsmStoreKey

This command doesn't store the key. Instead, it wraps the key into RFC3394 blobs that can be later stored on the device using the LoadKeyBlob SB command.

Syntax:

```
blhost -p <COM> -- trust-provisioning HsmStoreKey
<keyType>
<keyProp>
<keyInputAddr>
<keyInputSize>
<keyBlobOutputAddr>
<keyBlobOutputSize>
```

Parameters:

Name	Description
keyType	Pass one of the following constant values to specify the key to be generated by the function. • CUST_AESK_128 (128-bit AES key) • CUST_AESK_256 (256-bit AES key)
keyProp	This parameter is not used on this device.
keyInputAddr	The address where the keyInput is stored.
keyInputSize	The size of the keyInput.
keyBlobOutputAddr	The address where the key blob is written. The blob is an RFC3394 key blob wrapped with the NXP_CUST_KEK_EXT_SK key.
keyBlobOutputSize	The max size of the key blob.

9.4.4 HsmEncBlk

This command encrypts blocks, using CUSTMKSK key generated from HsmGenKey command and information from the SB3.1 manifest.

Syntax:

```
blhost -p <COM> -- trust-provisioning HsmEncBlk
<sbxHeaderInputAddr>
<sbxHeaderInputSize>
<blockNum>
<blockDataAddr>
<blockDataSize>
```

Parameters:

Name	Description
sbHeaderInputAddr	The address where the SB3.1 header is stored. The header is needed because the oemBlob and timestamp are used for encryption.
sbHeaderInputSize	The size of the SB header.
blockNum	The current block being encrypted.
blockDataAddr	The address where the block is stored.
blockDataSize	The size of the block.

9.4.5 HsmEncSign

This command signs a block of data with the key provided in the keyBlob parameter.

Syntax:

```
blhost -p <COM> -- trust-provisioning HsmEncSign
<blockDataInputAddr>
<blockDataInputSize>
<signatureOutputAddr>
<signatureOutputSize>
```

Parameters:

Name	Description
blockDataInputAddr	The address where the block is being stored.
blockDataInputSize	The size of the block.
signatureOutputAddr	The address where the signature will be written to.
signatureOutputSize	The size of the signature.

9.5 Secure provisioning using SI

illustrates the secure provisioning programming flow.

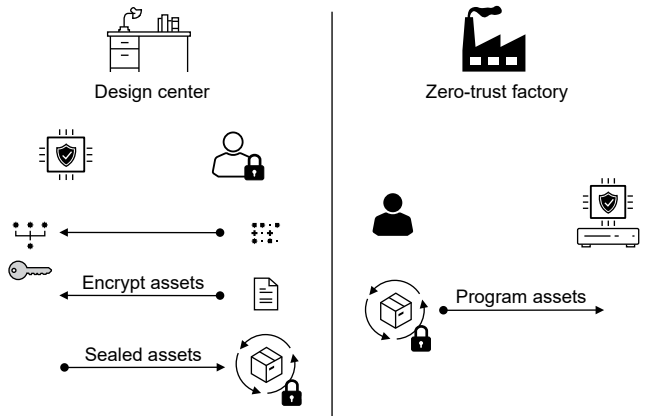


Figure 35. Secure provisioning using SI

The SB3.1 file is provisioned using the following flow:

- OEM provisioning
  - Use an EVK board with a genuine MCU and the SPSDK tool.
  - Create an SB3.1 file containing the SI and optional application image.
  - See SB file creation for details about SB3.1 file creation.
- Factory programming
  - The SB3.1 file is securely transferred to the factory floor.
  - The manufacturing tool (using SPSDK) programs the device with the encrypted and signed SB3.1 file.
  - The SB3.1 file is installed confidentially and securely.

9.6 Hazards

When erasing the SI, make sure the erase-token is erased before programming the CMPA_DELETE_INSTALLER_MAGIC into the CMPA_DELETE_INSTALLER field. Otherwise, the device will be bricked on the next reset.

When moving to OEM Closed lifecycle, make sure the SI has been erased. Otherwise, the bootloader will be unable to revert to OEM Open.

Chapter 10
Debug Mailbox (DBGMB)

10.1 Chip-specific Debug Mailbox information

Table 66. Reference links to related information[1]

Topic	Related module	References
Full description	DBGMB	DBGMB
System memory map		System memory map
Clocking		See the chapter "Clocking"
Signal multiplexing	Port control	See the chapter "Signal multiplexing"
System debug		See the chapter "Debug"

1. For all the reference sections and chapters mentioned in this table, refer the Reference Manual.

10.1.1 Module instance

This device has one instance of the DBGMB module.

10.1.2 Security considerations

For secure applications, NXP recommends that this module is configured for secure and privileged access only to prevent unintentional or malicious modification of the system operation by nonsecure software.

10.2 Overview

DBGMB is a small register-based mailbox accessible by both the AHB and the debug port of the chip.

The purpose of DBGMB is to allow the debugger to communicate with the onboard ROM code through the AHB bus, even when the main CPU AP is disabled.

Using this interface, ROM implements a command-response protocol.

10.2.1 Block diagram

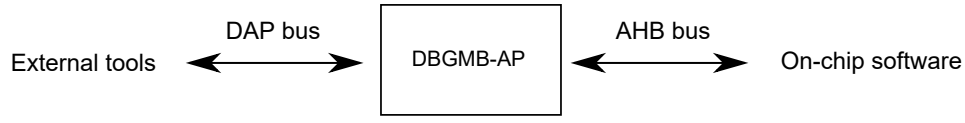


Figure 36. Debug Mailbox -Access Port (DBGMB-AP)

10.2.2 Features

- Support for Arm SWD mode
- Direct debug access to all memories, registers, and peripherals
- No target resource requirements for a debugging session
- Support for setting instruction breakpoints
- Support for setting data watchpoints, which you can use as triggers
- Optional additional software-controlled trace for the CPU via the instrumentation trace macrocell (ITM)

10.3 Functional description

10.3.1 Debugger mailbox access port (DM-AP)

The DM-AP provides a register-based mailbox accessible by a chip's core and the device debug port (DP). This port is always enabled. An external host communicating through the SWD interface can exchange messages and data with the boot code executing from the ROM. This port implements the NXP debug authentication protocol.

The boot ROM implements a debug mailbox protocol to interact with tools via the SWD interface.

The debug mailbox protocol has the following features:

- Request-and-response based, where the requests and responses use the same basic structure
- Support for relatively large command and response data
- 32-bit word alignment for all commands and responses
- Support for data above 32 bits via an ACK_TOKEN that moderates the transfer in 32-bit value chunks

10.3.2 Mailbox commands

This section describes the request and response message formats and available mailbox commands.

10.3.2.1 Request packet layout

The first word transmitted in a request is a header word containing the command ID and the number of following data words. The command packet is sent to the device by writing 32 bits at a time to the REQUEST register. When sending command packets greater than 32 bits, the debugger must read an ACK_TOKEN in the RETURN register before writing the next 32 bits.

The 32-bit words quantified by the header follow the header itself.

Table 67. Request register byte description

Word	Byte 0	Byte 1	Byte 2	Byte 3
0	commandID[7:0]	commandID[15:8]	dataWordCount[7:0]	dataWordCount[15:8]
1	*data*	—	—	—

The C structure definition for a request is:

```
struct dm_request {
        uint16_t commandID;
        uint16_t dataWordCount;
        uint32_t data[ ];
        };
```

10.3.2.1.1 DM-AP commands

DM-AP commands are written to the REQUEST register. An Exit DM_AP command follows one or more of the DM-AP commands in the table below to resume the normal device boot flow. This sequence does not occur after the Enter ISP Mode and Start Debug Session commands.

NOTE

DM-AP commands can vary among chips. See the full DM-AP commands list in the chip RM.

Table 68. DM-AP commands

Command ID	Name
01h	Start DM-AP
02h	Get Lifecycle status
03h	Mass Erase
04h	Exit DM-AP
05h	Enter ISP Mode
06h	Set FA Mode
07h	Start DBG Session
10h	Debug Auth. Start
11h	Debug Auth. Response
12h	NXP Insert Certificate
13h	Execute SB3

10.3.2.2 Response packet layout

The first word transmitted in a response is a header word containing the command status and the number of following data words. The command response can be read 32 bits at a time through Return Value (RETURN). The initial 32 bits contains the response header, as shown in the table below. When reading a response longer than 32 bits, the debugger writes the ACK_TOKEN to REQUEST after every read until the full response packet is received.

NOTE

To support legacy LPC command and response values, Bit_31 in the header indicates that the response follows the new protocol structure (see Table 70). This bit is 1 when using the new protocol.

Table 69. Response register byte description

Word	Byte 0	Byte 1	Byte 2	Byte 3
0	bits[7:0]:commandStatus[7:0]	bits[7:0]:commandStatus[15:8]	bits[7:0]:dataWordCount[7:0]	bits[6:0]:dataWordCount[14:8] bits[7]:new_protocol[15]
1	*data*	—	—	—

The C structure definition for a response is:

```
struct dm_response {
        uint16_t commandStatus;
        uint16_t dataWordCount;
        uint32_t data[];
        };
```

10.3.2.2.1 Response packet

You can read the command response 32 bits at a time through Return Value (RETURN). The initial 32 bits contain the response header, as shown in the table below. When reading a response greater than 32 bits, the debugger writes the ACK_TOKEN to Request Value (REQUEST) after every read of RETURN until the full response packet is received.

Table 70. Response Packet

Bits	Field	Description
31	LONG_RESP	Long response packet indicator; also called the new protocol indicator
[30:16]	REMAIN_TRANS	Number of times the debugger must read RETURN to receive remaining response data. The debugger writes ACK_TOKEN to REQUEST after every read of RETURN until the full response packet is received. **NOTE** REMAIN_TRANS is valid only when ERROR_SRESP is 0 and LONG_RESP is 1.
[15:0]	ERROR_SRESP	Short response data or error code If bit_20 is not 1, this field is interpreted as short response data. For example, this field returns the CRP level for a GET_CRP_LEVEL command. Error codes • 0000h: Command succeeded. • 0001h: Debug mode not entered. This value is returned when other commands are sent prior to the Enter DM-AP command. • 0002h: Command is not supported. • 0003h: Communication failure. ACK_TOKEN is missing during data transactions.

10.3.2.3 ACK_TOKEN

The ACK_TOKEN provides an acknowledgment to the sender during debug mailbox data transactions. DBGMB uses the acknowledgment in the following ways:

- When the debugger issues a command with parameter data, it waits for the ACK_TOKEN (read through Return Value (RETURN)) before writing the next 32-bit value to Request Value (REQUEST).
- When the debugger receives a long response packet, it writes the ACK_TOKEN to REQUEST before reading the next 32-bit value RETURN.

Table 71. ACK_TOKEN

Bits	Field	Description
[31:16]	REMAIN_TRANS	Number of remaining data transactions
[15:0]	ACK_MARKER	Acknowledgment marker; must always be A5A5h

The C structure definition for the ACK_TOKEN is:

```
struct dm_ack_token {
        uint16_t token; /* always set to A5A5h */
        uint16_t remainCount; /* count of remaining word */
        };
```

10.3.2.4 Error handling

When an overrun occurs from either side of the communication, DBGMB sets the appropriate error flag in Command and Status Word (CSW). The state machine hardware prevents further communication in either direction. The debugger must start with a new resynchronization request to clear the error flag.

10.4 External signals

Table 72 shows the signals related to the debug process. A trace using the serial wire output has limited bandwidth.

Table 72. Serial wire debug signals

Signal	I/O	Description
SWCLK	I	Serial wire clock. Provides the clock for the SWD debug logic in Serial Wire Debug (SWD) mode. SWCLK is the default signal for its pin. At the release of reset, the pin is pulled down internally.
SWDIO	I/O	Serial wire debug data input and output. Used by an external debug tool to communicate with and control the part. SWDIO is the default signal of its pin. At the release of reset, the pin is pulled up internally.
SWO	O	Serial wire output. Optionally provides data from the Instrumentation Trace Macrocell (ITM) for an external debug tool to evaluate. See the chip-specific DBGMB information for the clocking required to enable SWO.

10.5 Memory map and register definition

10.5.1 DBGMB register descriptions

10.5.1.1 DBGMB memory map

DebugMailbox0 base address: 4010_1000h

Offset	Register	Width (In bits)	Access	Reset value
0h	Command and Status Word (CSW)	32	RW	0000_0000h
4h	Request Value (REQUEST)	32	RW	0000_0000h
8h	Return Value (RETURN)	32	RW	0000_0000h
FCh	Identification (ID)	32	R	002A_0000h

10.5.1.2 Command and Status Word (CSW)

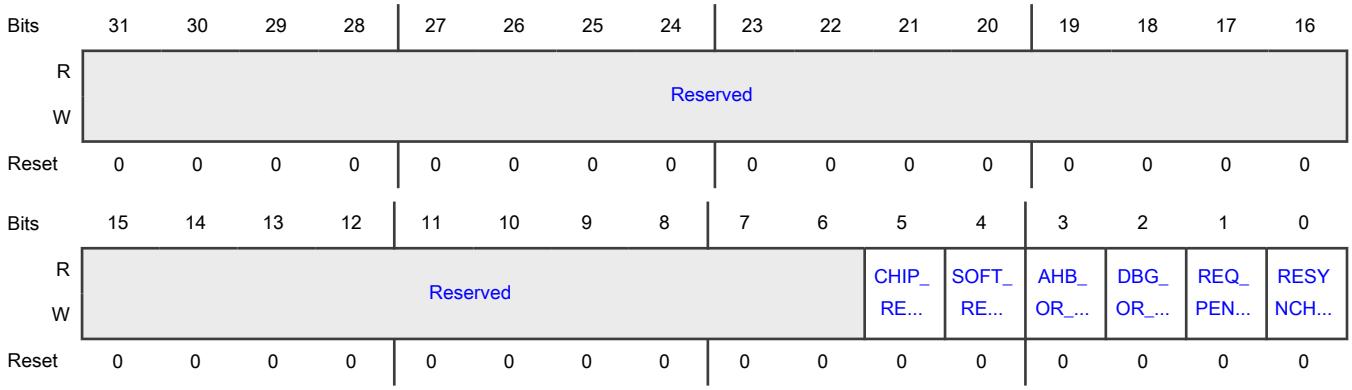
Offset

Register	Offset
CSW	0h

Function

Contains command and status bits to facilitate communication between DBGMB and the chip.

Diagram



Fields

Field	Function
31-6 —	Reserved
5 CHIP_RESET_REQ	Chip Reset Request Causes the chip (but not the DM-AP) to be reset by generating SYSRESET_REQ. This field is write only. 0b - No effect 1b - Reset
4 SOFT_RESET	Soft Reset Resets the DM-AP. This field is write only by the chip. 0b - No effect 1b - Reset
3 AHB_OR_ERR	AHB Overrun Error Indicates whether an AHB overrun has occurred: the chip has overwritten a RETURN value before DBGMB read the RETURN value. 0b - No overrun 1b - Overrun occurred
2 DBG_OR_ERR	DBGMB Overrun Error Indicates whether a DBGMB overrun has occurred: DBGMB has overwritten a REQUEST value before the chip read the REQUEST value. 0b - No overrun 1b - Overrun occurred
1 REQ_PENDING	Request Pending Indicates a pending request for DBGMB: a value is waiting to be read from Request Value (REQUEST).

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - No request pending 1b - Request for resynchronization pending
0 RESYNCH_REQ	Resynchronization Request Requests a resynchronization. 0b - No request 1b - Request for resynchronization

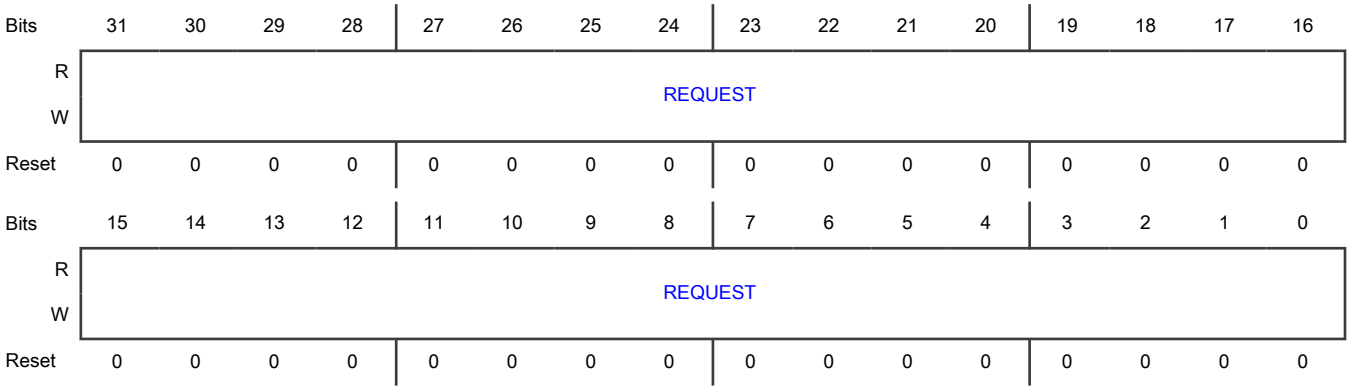
10.5.1.3 Request Value (REQUEST)

Offset

Register	Offset
REQUEST	4h

Function
Used by DBGMB to send action requests to the chip.

Diagram



Fields

Field	Function
31-0 REQUEST	Request Value Indicates the request value. This field reads 0 when no new request is present. The chip clears this field. DBGMB can read back this field to confirm communication.

10.5.1.4 Return Value (RETURN)

Offset

Register	Offset
RETURN	8h

Function
Provides the responses from the chip to DBGMB.

Diagram

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

Fields

Field	Function
31-0 RET	Return Value Indicates the return value, which is a response from the chip to DBGMB. If no new data is present, DBGMB reading is stalled until new data is available.

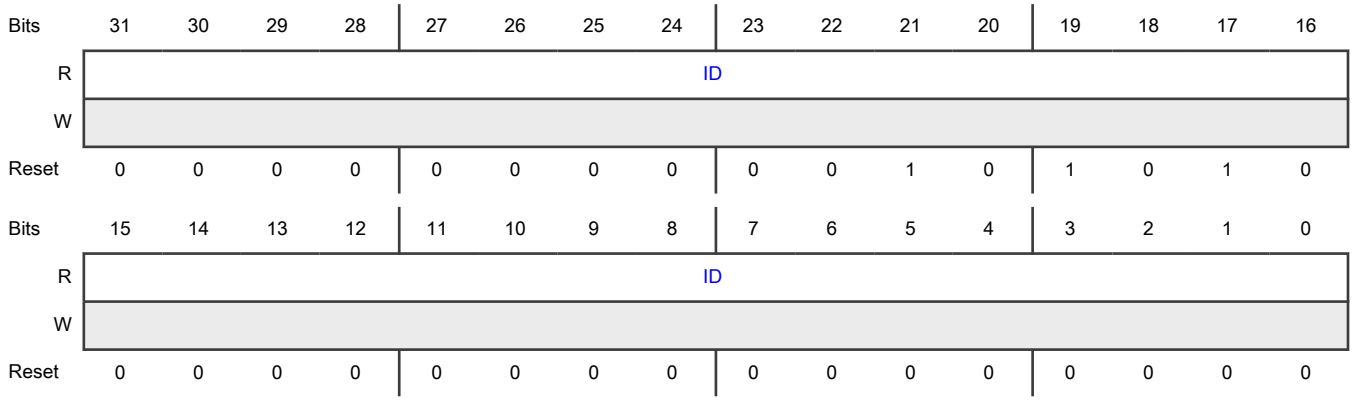
10.5.1.5 Identification (ID)

Offset

Register	Offset
ID	FCh

Function
Provides an identification of the DM-AP interface.

Diagram



Fields

Field	Function
31-0 ID	Identification Value Provides an identification of the DM-AP interface. The Arm Debug Interface Specification version 5.0 (ADIv5) requires every AP to implement an AP identification register, at offset FCh. This register is the last register in the AP register space. This ID register is only readable by external tools (a debug dongle) when identifying the Access Port (AP). For the debug mailbox AP, the identification value is 002A_0000h. This register is not readable from on-chip software. If you attempt a read, you receive a value of 0000_0000h.

Chapter 11
System Controller (SYSCON)

11.1 Chip-specific SYSCON information

Table 73. Reference links to related information[1]

Topic	Related module	References
Full description	SYSCON	SYSCON
System memory map		System memory map
Clocking		See the chapter "Clocking"
Signal multiplexing	Port control	See the chapter "Signal multiplexing"
System debug		See the chapter "Debug"

1. For all the reference sections and chapters mentioned in this table, refer the Reference Manual.

11.1.1 Module instance

This device has one instance of the SYSCON module.

11.1.2 Security considerations

For secure applications, NXP recommends that this module is configured for secure and privileged access only to prevent unintentional or malicious modification of the system operation by nonsecure software.

11.1.3 Configuration

Configure the SYSCON block as follows:

- No clock configuration is needed. The clock to the SYSCON block is always enabled.
- The SYSCON block controls use of the CLKOUT pin, which must also be configured through Port control module.

11.2 Overview

The SYSCON module provides controls and configurations on the system and peripherals for the multiple functions.

11.2.1 Features

The SYSCON module supports the following features and configurations:

- System and bus configuration
  - AHB matrix priority
  - CPU control and status
  - CPU debug access
  - CPU LPCAC control
  - NMI source select
  - Calibrate system tick timer
  - Boot control
- System/Peripherals - clock select and control

- Allows enabling and selection of clocks to individual peripherals and memories
- Allows configuration of clock dividers

- Memory configuration
  - ROM access
  - RAM ECC
  - FLASH cache, bus error
- Gray-2-Binary converter
  - Allows decoding gray value coming from OS Event Timer
- Peripherals configuration
  - PWM control
  - CTIMER global start
- Security configuration
  - Security control
  - Security attestation
- Reset control
  - Monitors and release resets to individual peripherals
- Device ID register

11.3 Functional description

The MRCC submodule provides on-chip modules their own dedicated MRCC bits for clock gating, reset control and configuration options. Generally, MRCC register contains a clock configuration (CC = {MRCC_GLB_ACC*[peripherals name]*, MRCC_GLB_CC*[peripherals name]* }) field for the module's clocks.

NOTE

Before a module can be used, its clocks must be enabled (CC != 00) and it must be released from reset (MRCC_GLB_RST*[peripherals name]*= 1). If a module is *not* released from reset (MRCC_GLB_RST*[peripherals name]*= 0), an attempt to access a register within that module is terminated with a bus error.

If a module has a functional clock, its MRCC register may provide options for the clock source, selected by programming the mux select (MUX) field. The selected functional clock source can then be divided by programming the divider (DIV) field.

NOTE

Before configuring a functional clock, the module's clocks must be disabled (CC = 0b00).

11.4 Signals

Table 74. SYSCON pin description

Function	Type	Pin	Description
CLKOUT	O	P0_6, P3_6, P3_8, P4_2	The CLKOUT function provides a clock output through these pins if they are available in the package.

11.5 Memory map and register definition

This section includes the SYSCON module memory map and detailed descriptions of all registers.

11.5.1 SYSCON register descriptions

11.5.1.1 SYSCON memory map

SYSCON.syscon base address: 4009_1000h

Offset	Register	Width (In bits)	Access	Reset value
200h	AHB Matrix Remap Control (REMAP)	32	RW	0000_0000h
210h	AHB Matrix Priority Control (AHBMATPRIO)	32	RW	0000_0000h
23Ch	Non-Secure CPU0 System Tick Calibration (CPU0NSTCKCAL)	32	RW	0000_0000h
248h	NMI Source Select (NMISRC)	32	RW	0000_0000h
24Ch	Protect Level Control (PROTLVL)	32	RW	0000_0000h
378h	SLOW_CLK Clock Divider (SLOWCLKDIV)	32	RW	0000_0005h
37Ch	BUS_CLK Clock Divider (BUSCLKDIV)	32	RW	0000_0001h
380h	System Clock Divider (AHBCLKDIV)	32	RW	0000_0000h
388h	FRO_HF_DIV Clock Divider (FROHFDIV)	32	RW	4000_0000h
38Ch	FRO_LF_DIV Clock Divider (FROLFDIV)	32	RW	4000_0000h
3E4h	PLL1_CLK_DIV Clock Divider (PLL1CLKDIV)	32	RW	4000_0000h
3FCh	Clock Configuration Unlock (CLKUNLOCK)	32	RW	0000_0000h
400h	NVM Control (NVM_CTRL)	32	RW	0002_0400h
414h	SmartDMA Interrupt Hijack (SmartDMAINT)	32	RW	See section
470h	Controls RAM Interleave Integration (RAM_INTERLEAVE)	32	RW	0000_0000h
80Ch	CPU Status (CPUSTAT)	32	R	0000_0000h
824h	LPCAC Control (LPCAC_CTRL)	32	RW	0000_0031h
938h	PWM0 Submodule Control (PWM0SUBCTL)	32	RW	0000_0000h
93Ch	PWM1 Submodule Control (PWM1SUBCTL)	32	RW	0000_0000h
940h	CTIMER Global Start Enable (CTIMERGLOBALSTARTEN)	32	RW	0000_0000h
944h	RAM Control (RAM_CTRL)	32	RW	0000_0001h
B60h	Gray to Binary Converter Gray Code [31:0] (GRAY_CODE_LSB)	32	RW	0000_0000h
B64h	Gray to Binary Converter Gray Code [41:32] (GRAY_CODE_MSB)	32	RW	0000_0000h
B68h	Gray to Binary Converter Binary Code [31:0] (BINARY_CODE_LSB)	32	R	0000_0000h
B6Ch	Gray to Binary Converter Binary Code [41:32] (BINARY_CODE_MSB)	32	R	0000_0000h
E1Ch	MSF Configuration (MSFCFG)	32	RW	0000_0000h
E3Ch	ROP State Register (ROP_STATE)	32	R	See section

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
E58h	RAM XEN Control (SRAM_XEN)	32	RW	0000_0000h
E5Ch	RAM XEN Control (Duplicate) (SRAM_XEN_DP)	32	RW	0000_0000h
E80h	Life Cycle State Register (ELS_OTP_LC_STATE)	32	R	See section
E84h	Life Cycle State Register (Duplicate) (ELS_OTP_LC_STATE_DP)	32	R	See section
FA0h	Control Write Access to Security (DEBUG_LOCK_EN)	32	RW	0000_000Ah
FA4h	Cortex Debug Features Control (DEBUG_FEATURES)	32	RW	See section
FA8h	Cortex Debug Features Control (Duplicate) (DEBUG_FEATURES_DP)	32	RW	See section
FB4h	CPU0 Software Debug Access (SWD_ACCESS_CPU0)	32	RW	0000_0000h
FC0h	Debug Authentication BEACON (DEBUG_AUTH_BEACON)	32	RW	0000_0000h
FF0h	JTAG Chip ID (JTAG_ID)	32	R	0726_802Bh
FF4h	Device Type (DEVICE_TYPE)	32	R	See section
FF8h	Device ID (DEVICE_ID0)	32	R	See section

11.5.1.2 AHB Matrix Remap Control (REMAP)

Offset

Register	Offset
REMAP	200h

Function

The Multilayer AHB Matrix remap for all masters, when they attempt to access the matrix slave port.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	LOCK	Reserved					USB0		Reserved							
W	1															
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		PKC		Reserved		DMA0		SmartDMA_D		SmartDMA_I		CPU0_SBUS		Reserved	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31 LOCK	This 1-bit field provides a mechanism to limit writes to this register to protect its contents. Once set, this bit remains asserted until a system reset. 0b - This register is not locked and can be altered. 1b - This register is locked and cannot be altered until a system reset.
30-26 —	Reserved Read value is undefined, only zero should be written.
25-24 USB0	RAMX0 address remap for USB0 00b - RAMX0: alias space is disabled. 01b - RAMX0: same alias space as CPU0_SBUS
23-14 —	Reserved Read value is undefined, only zero should be written.
13-12 PKC	RAMX0 address remap for PKC 00b - RAMX0: alias space is disabled. 01b - RAMX0: same alias space as CPU0_SBUS
11-10 —	Reserved Read value is undefined, only zero should be written.
9-8 DMA0	RAMX0 address remap for DMA0 00b - RAMX0: alias space is disabled. 01b - RAMX0: same alias space as CPU0_SBUS
7-6 SmartDMA_D	RAMX0 address remap for SmartDMA D-BUS 00b - RAMX0: alias space is disabled. 01b - RAMX0: same alias space as CPU0_SBUS
5-4 SmartDMA_I	RAMX0 address remap for SmartDMA I-BUS 00b - RAMX0: alias space is disabled. 01b - RAMX0: same alias space as CPU0_SBUS
3-2 CPU0_SBUS	RAMX0 address remap for CPU System bus 00b - RAMX0: alias space is disabled. 01b - RAMX0: alias space is enabled. It's linear address space from bottom of system ram. The start address is 0x20000000 + (system ram size - RAMX size)*1024.
1-0 —	Reserved Read value is undefined, only zero should be written.

11.5.1.3 AHB Matrix Priority Control (AHBMATPRIO)

Offset

Register	Offset
AHBMATPRIO	210h

Function

The Multilayer AHB Matrix arbitrates between masters, when they attempt to access the same matrix slave port at the same time. The priority values are 3 = highest, 0 = lowest. When the priority is the same, the master with the lower master number is given priority.

NOTE

Be careful when modifying this register as improper settings can seriously degrade the performance.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved		Reserved		Reserved		USB_FS_ENET		Reserved		Reserved		Reserved		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		PKC_ELS		Reserved		DMA0		CPU1_SBUS_Smart...		CPU1_CBUS_Smart...		CPU0_SBUS		CPU0_CBUS	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-30 —	Reserved
29-28 —	Reserved
27-26 —	Reserved
25-24 USB_FS_ENET	USB-FS bus master priority level 00b - level 0 01b - level 1 10b - level 2

Table continues on the next page...

Table continued from the previous page...

Field	Function
	11b - level 3
23-22 —	Reserved
21-20 —	Reserved
19-18 —	Reserved
17-16 —	Reserved
15-14 —	Reserved
13-12 PKC_ELS	PKC and ELS bus master priority level 00b - level 0 01b - level 1 10b - level 2 11b - level 3
11-10 —	Reserved
9-8 DMA0	DMA0 controller bus master priority level 00b - level 0 01b - level 1 10b - level 2 11b - level 3
7-6 CPU1_SBUS_SmartDMA_D	SmartDMA-D bus master priority level 00b - level 0 01b - level 1 10b - level 2 11b - level 3
5-4 CPU1_CBUS_SmartDMA_I	SmartDMA-I bus master priority level 00b - level 0

Table continues on the next page...

Table continued from the previous page...

Field	Function
	01b - level 1 10b - level 2 11b - level 3
3-2 CPU0_SBUS	CPU0 S-AHB bus master priority level 00b - level 0 01b - level 1 10b - level 2 11b - level 3
1-0 CPU0_CBUS	CPU0 C-AHB bus master priority level 00b - level 0 01b - level 1 10b - level 2 11b - level 3

11.5.1.4 Non-Secure CPU0 System Tick Calibration (CPU0NSTCKCAL)

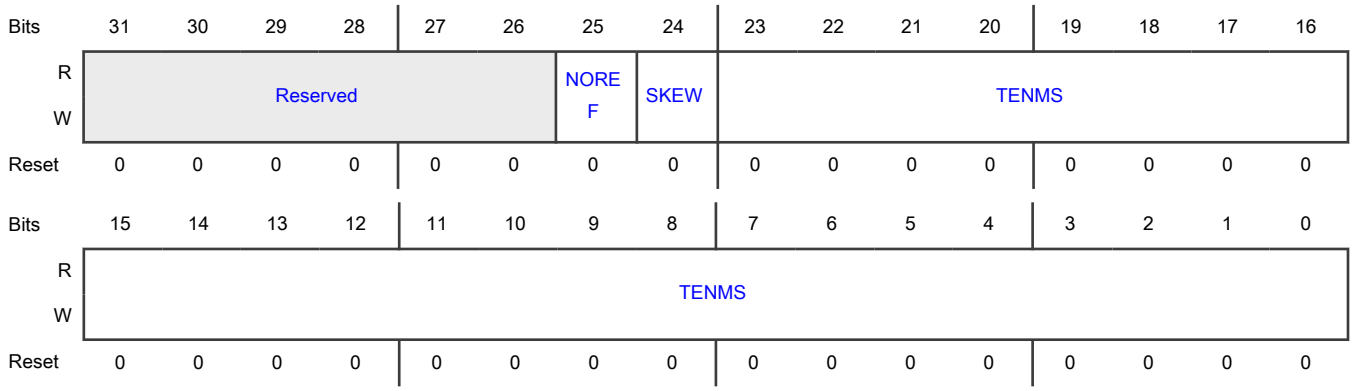
Offset

Register	Offset
CPU0NSTCKCAL	23Ch

Function

The CPU0NSTCKCAL register allows software to set up a default value for the SYST_CALIB register in the System Tick Timer of non-secure part of the CPU0.

Diagram



Fields

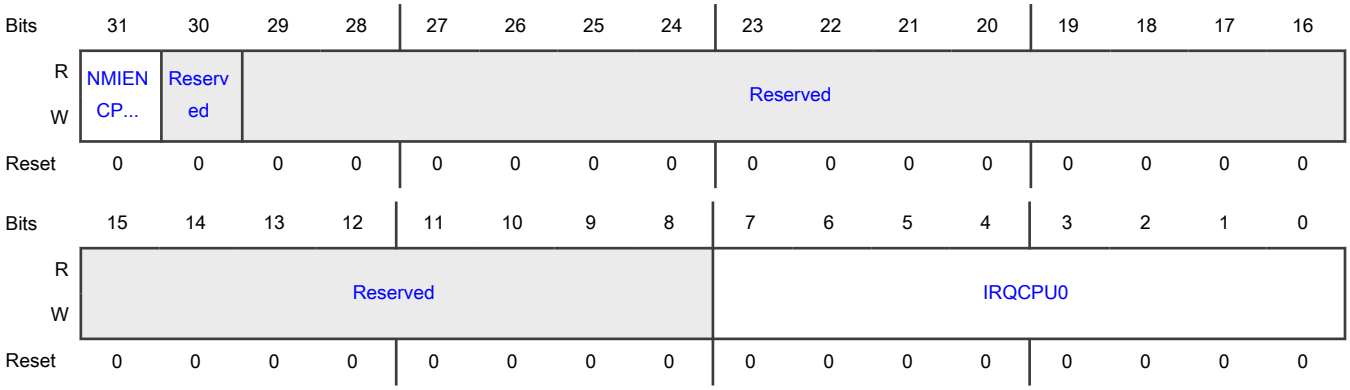
Field	Function
31-26 —	Reserved
25 NOREF	Indicates whether the device provides a reference clock to the processor. 0b - Reference clock is provided 1b - No reference clock is provided
24 SKEW	Indicates whether the TENMS value is exact. 0b - TENMS value is exact 1b - TENMS value is not exact or not given
23-0 TENMS	Reload value for 10 ms (100 Hz) timing, subject to system clock skew errors. If the value reads as zero, the calibration value is not known.

11.5.1.5 NMI Source Select (NMISRC)

Offset

Register	Offset
NMISRC	248h

Diagram



Fields

Field	Function
31 NMIENCPU0	Enables the Non-Maskable Interrupt (NMI) source selected by IRQCPU0. 0b - Disable. 1b - Enable.

Table continues on the next page...

Table continued from the previous page...

Field	Function
30 —	Reserved
29-16 —	Reserved
15-8 —	Reserved
7-0 IRQCPU0	The IRQ number of the interrupt that acts as the Non-Maskable Interrupt (NMI) for CPU0, if enabled by NMIENCPU0.

11.5.1.6 Protect Level Control (PROTLVL)

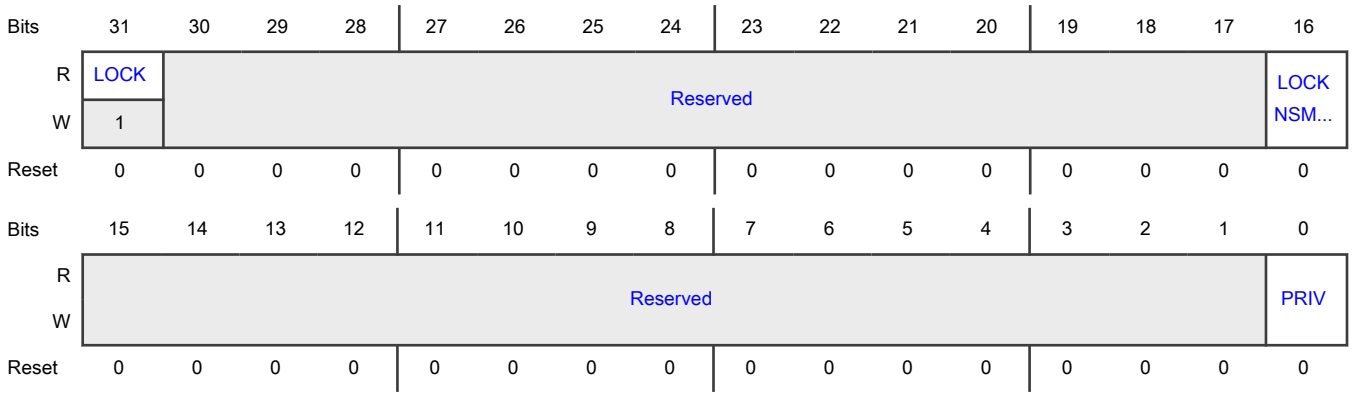
Offset

Register	Offset
PROTLVL	24Ch

Function

This register is privileged access only.

Diagram



Fields

Field	Function
31 LOCK	This 1-bit field provides a mechanism to limit writes to this register to protect its contents. Once set, this bit remains asserted until a system reset.

Table continues on the next page...

Table continued from the previous page...

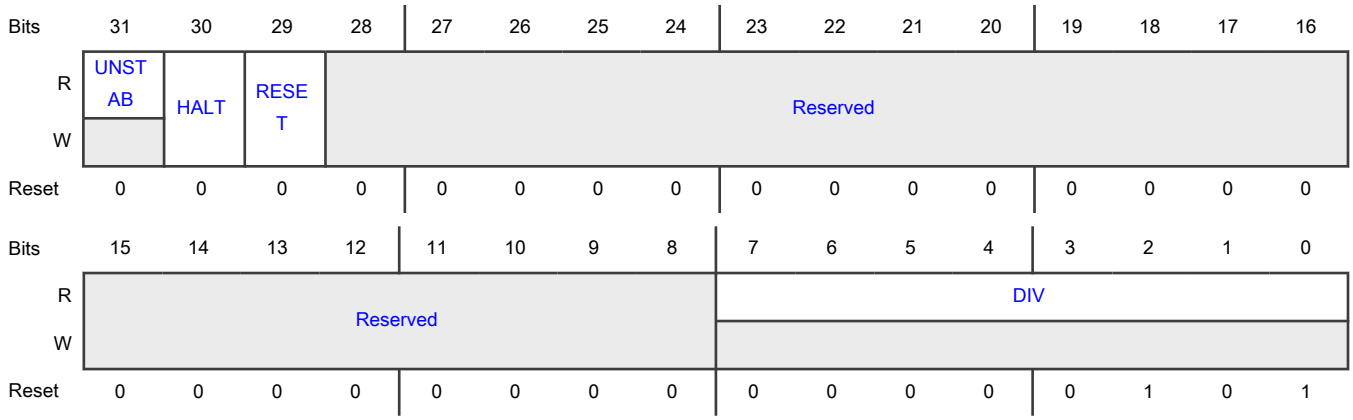
Field	Function
	0b - This register is not locked and can be altered. 1b - This register is locked and cannot be altered until a system reset.
30-17 —	Reserved
16 LOCKNSMPU	Control write access to Nonsecure MPU memory regions. 0b - Unlock these registers. privileged access to Nonsecure MPU memory regions is allowed. 1b - Disable writes to the MPU_CTRL_NS, MPU_RNR_NS, MPU_RBAR_NS, MPU_RLAR_NS, MPU_RBAR_A_NSn and MPU_RLAR_A_NSn. All writes to the registers are ignored.
15-1 —	Reserved
0 PRIV	Control privileged access of EIM, ERM, Flexcan, MBC, SCG. 0b - privileged access is disabled. the peripherals could be access in user mode. 1b - privileged access is enabled. the peripherals could be access in privilege mode.

11.5.1.7 SLOW_CLK Clock Divider (SLOWCLKDIV)

Offset

Register	Offset
SLOWCLKDIV	378h

Diagram



Fields

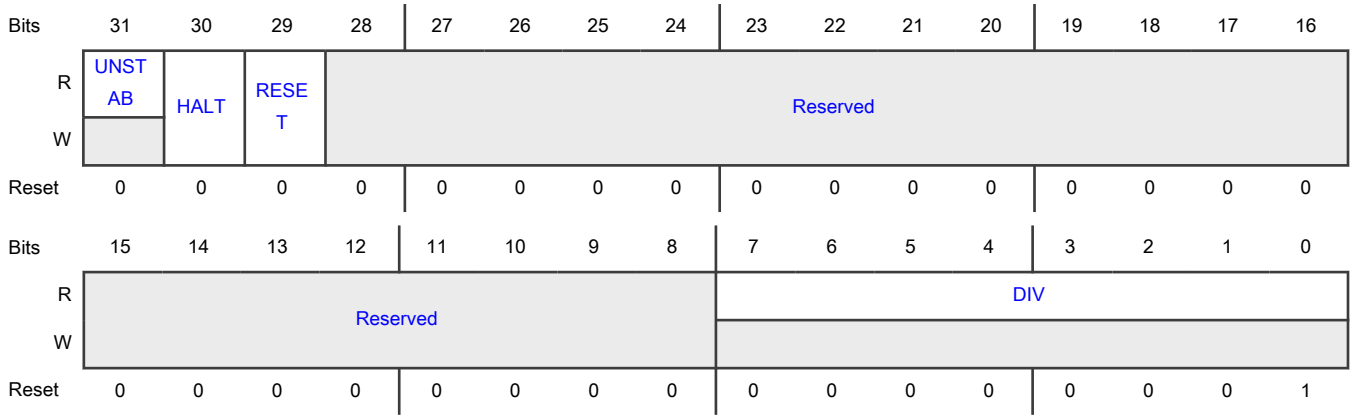
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter **NOTE** Limited to configure RESET bit when HALT bit is set to 1 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.5.1.8 BUS_CLK Clock Divider (BUSCLKDIV)

Offset

Register	Offset
BUSCLKDIV	37Ch

Diagram



Fields

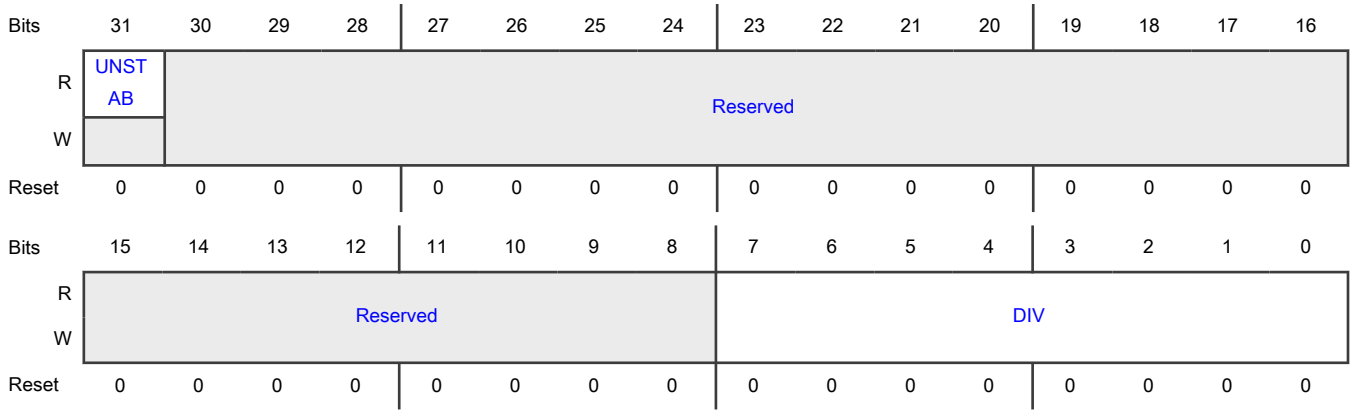
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter **NOTE** Limited to configure RESET bit when HALT bit is set to 1 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.5.1.9 System Clock Divider (AHBCLKDIV)

Offset

Register	Offset
AHBCLKDIV	380h

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

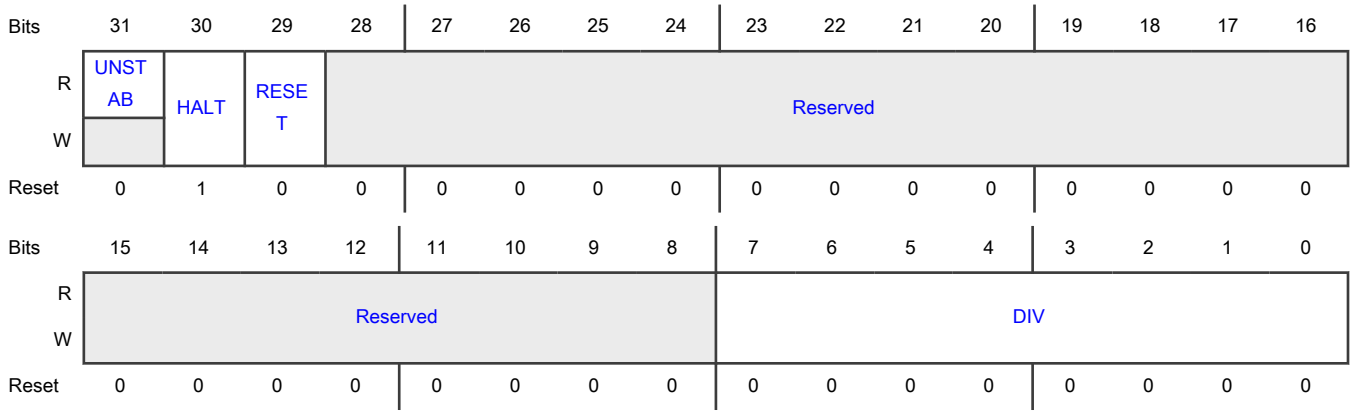
11.5.1.10 FRO_HF_DIV Clock Divider (FROHFDIV)

Offset

Register	Offset
FROHFDIV	388h

Function
This register is used to generate fro_hf_div clock from fro_hf clock.

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable

Table continues on the next page...

Table continued from the previous page...

Field	Function
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.5.1.11 FRO_LF_DIV Clock Divider (FROLFDIV)

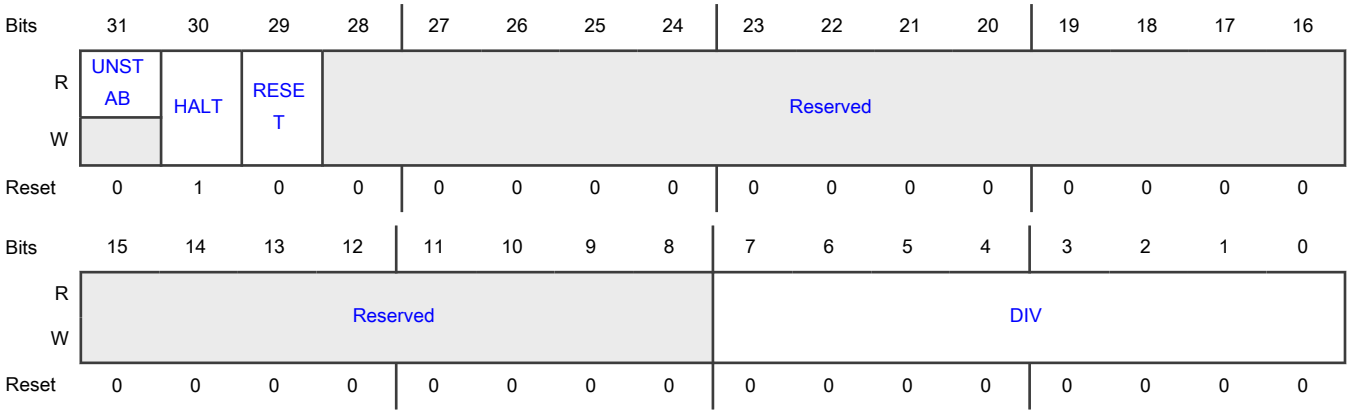
Offset

Register	Offset
FROLFDIV	38Ch

Function

This register is used to generate fro_lf_div clock from fro_lf clock.

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.5.1.12 PLL1_CLK_DIV Clock Divider (PLL1CLKDIV)

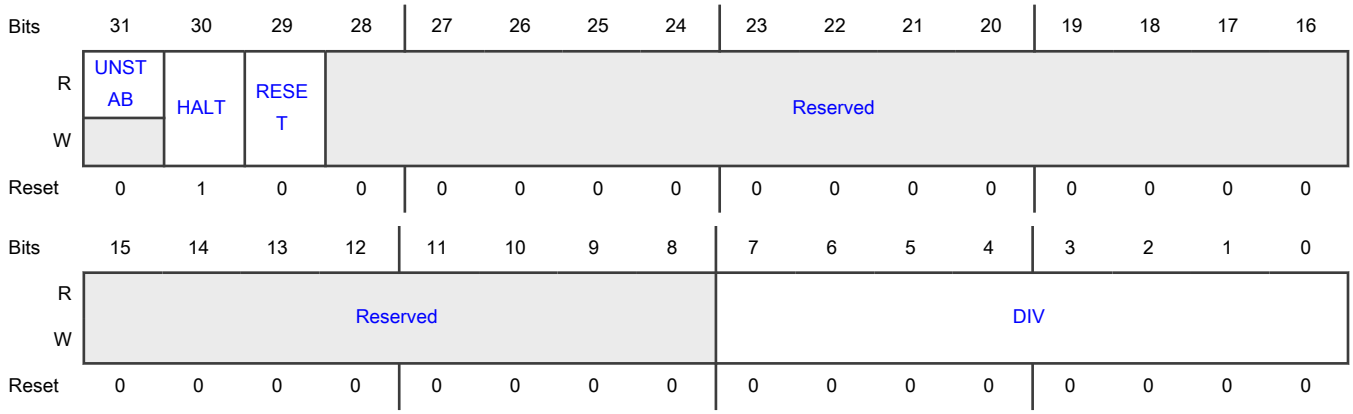
Offset

Register	Offset
PLL1CLKDIV	3E4h

Function

This register is used to generate pll1_clk_div clock from pll1_clk clock.

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.5.1.13 Clock Configuration Unlock (CLKUNLOCK)

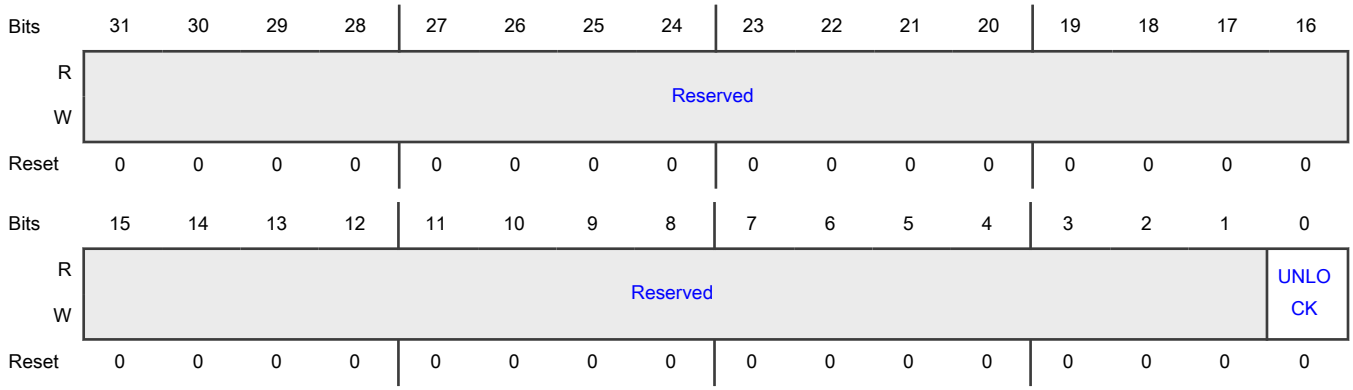
Offset

Register	Offset
CLKUNLOCK	3FCh

Function

This register controls access to the clock select and divider configuration registers.

Diagram



Fields

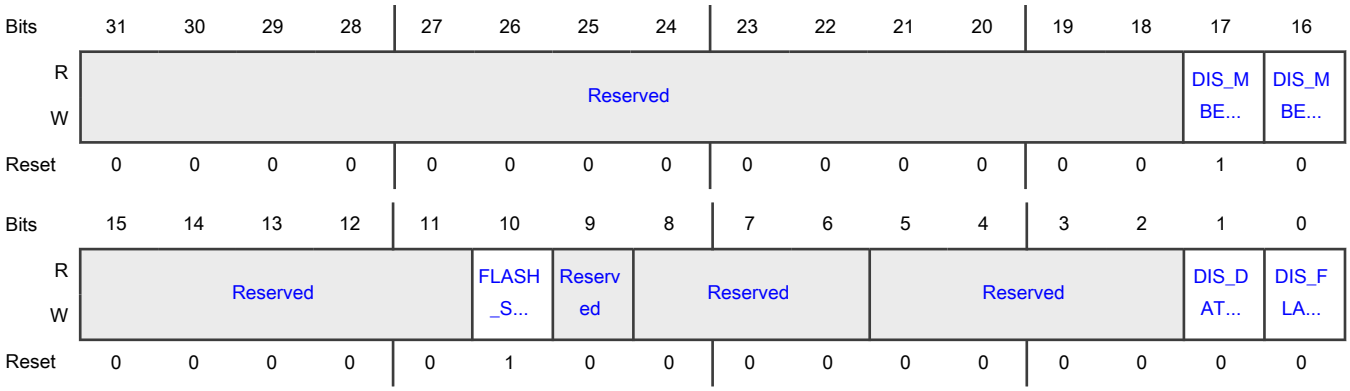
Field	Function
31-1 —	Reserved
0 UNLOCK	Controls clock configuration registers access (for example, SLOWCLKDIV, BUSCLKDIV, AHBCLKDIV, FROHFDIV, FROLFDIV, PLLxCLKDIV, MRCC_xxx_CLKDIV, MRCC_xxx_CLKSEL, MRCC_GLB_xxx) 0b - Updates are allowed to all clock configuration registers 1b - Freezes all clock configuration registers update.

11.5.1.14 NVM Control (NVM_CTRL)

Offset

Register	Offset
NVM_CTRL	400h

Diagram



Fields

Field	Function
31-18 —	Reserved
17 DIS_MBECC_ERR_DATA	Bus error on data multi-bit ECC error control **NOTE** Set this field to 0 if you want to enable flash speculative 0b - Enables bus error on multi-bit ECC error for data 1b - Disables bus error on multi-bit ECC error for data

Table continues on the next page...

Table continued from the previous page...

Field	Function
16 DIS_MBECC_ERR_INST	Bus error on data multi-bit ECC error control **NOTE** Set this field to 0 if you want to enable flash speculative 0b - Enables bus error on multi-bit ECC error for instruction 1b - Disables bus error on multi-bit ECC error for instruction
15-11 —	Reserved
10 FLASH_STALL_EN	FLASH stall on busy control 0b - No stall on FLASH busy 1b - Stall on FLASH busy
9 —	Reserved Keep the default value.
8-6 —	Reserved
5-2 —	Reserved
1 DIS_DATA_SPEC	Flash data speculation control **NOTE** If DIS_MBECC_ERR_DATA and/or DIS_MBECC_ERR_INST are set, then speculation will not be enabled, even if this bit is cleared. 0b - Enables data speculation 1b - Disables data speculation
0 DIS_FLASH_SPEC	Flash speculation control **NOTE** If DIS_MBECC_ERR_DATA and/or DIS_MBECC_ERR_INST are set, then speculation will not be enabled, even if this bit is cleared. 0b - Enables flash speculation 1b - Disables flash speculation

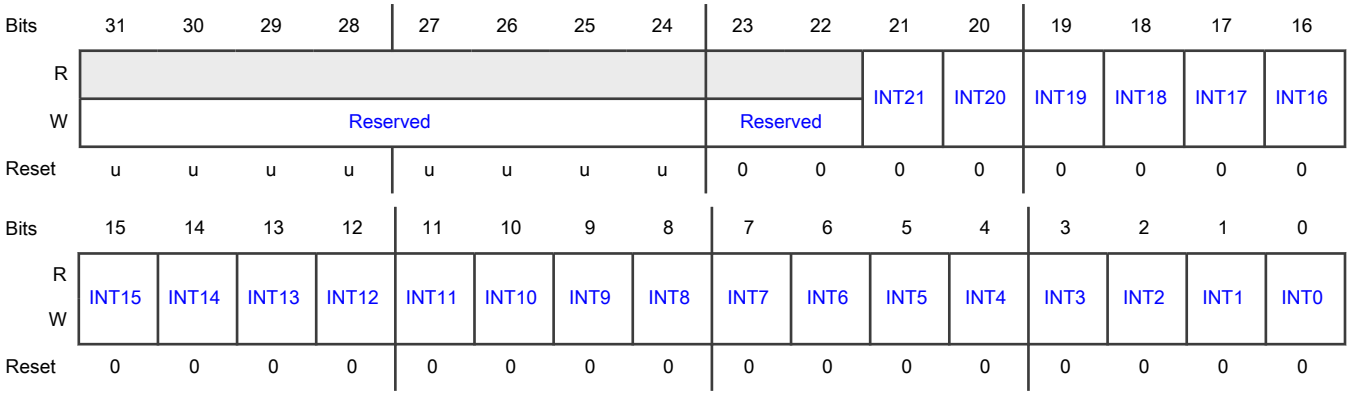
11.5.1.15 SmartDMA Interrupt Hijack (SmartDMAINT)

Offset

Register	Offset
SmartDMAINT	414h

Function
Bits directly control the SmartDMA hijacking the system interrupts.

Diagram



Fields

Field	Function
31-24 —	Reserved Read value is undefined, only zero should be written.
23-22 —	Reserved Read value is undefined, only zero should be written.
21 INT21	SmartDMA hijack NVIC IRQ75 0b - Disable 1b - Enable
20 INT20	SmartDMA hijack NVIC IRQ74 0b - Disable 1b - Enable
19 INT19	SmartDMA hijack NVIC IRQ73 0b - Disable 1b - Enable
18	SmartDMA hijack NVIC IRQ72

Table continues on the next page...

Table continued from the previous page...

Field	Function
INT18	0b - Disable 1b - Enable
17 INT17	SmartDMA hijack NVIC IRQ71 0b - Disable 1b - Enable
16 INT16	SmartDMA hijack NVIC IRQ64 0b - Disable 1b - Enable
15 INT15	SmartDMA hijack NVIC IRQ62 0b - Disable 1b - Enable
14 INT14	SmartDMA hijack NVIC IRQ59 0b - Disable 1b - Enable
13 INT13	SmartDMA hijack NVIC IRQ41 0b - Disable 1b - Enable
12 INT12	SmartDMA hijack NVIC IRQ40 0b - Disable 1b - Enable
11 INT11	SmartDMA hijack NVIC IRQ39 0b - Disable 1b - Enable
10 INT10	SmartDMA hijack NVIC IRQ36 0b - Disable 1b - Enable
9 INT9	SmartDMA hijack NVIC IRQ34 0b - Disable 1b - Enable
8 INT8	SmartDMA hijack NVIC IRQ33 0b - Disable 1b - Enable

Table continues on the next page...

Table continued from the previous page...

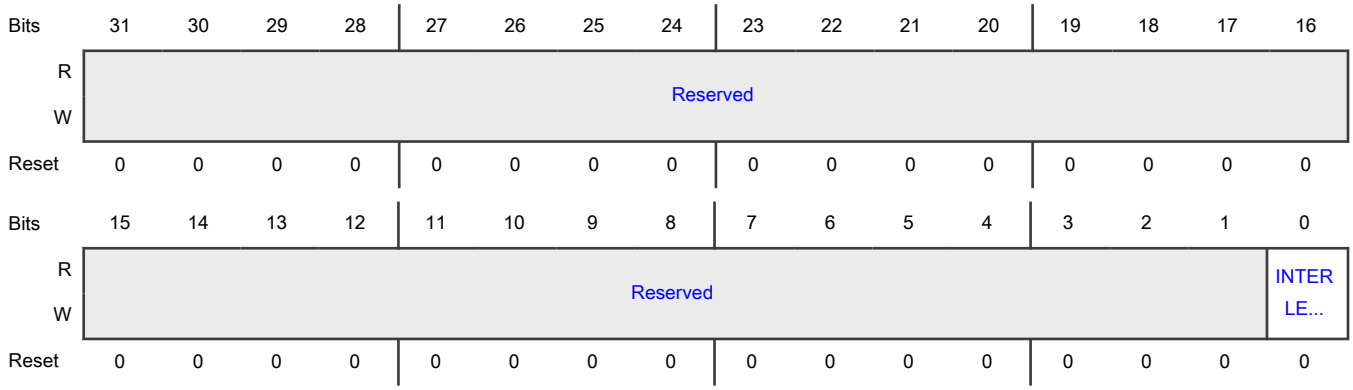
Field	Function
7 INT7	SmartDMA hijack NVIC IRQ32 0b - Disable 1b - Enable
6 INT6	SmartDMA hijack NVIC IRQ31 0b - Disable 1b - Enable
5 INT5	SmartDMA hijack NVIC IRQ29 0b - Disable 1b - Enable
4 INT4	SmartDMA hijack NVIC IRQ28 0b - Disable 1b - Enable
3 INT3	SmartDMA hijack NVIC IRQ27 0b - Disable 1b - Enable
2 INT2	SmartDMA hijack NVIC IRQ26 0b - Disable 1b - Enable
1 INT1	SmartDMA hijack NVIC IRQ23 0b - Disable 1b - Enable
0 INT0	SmartDMA hijack NVIC IRQ2 0b - Disable 1b - Enable

11.5.1.16 Controls RAM Interleave Integration (RAM_INTERLEAVE)

Offset

Register	Offset
RAM_INTERLEAVE	470h

Diagram



Fields

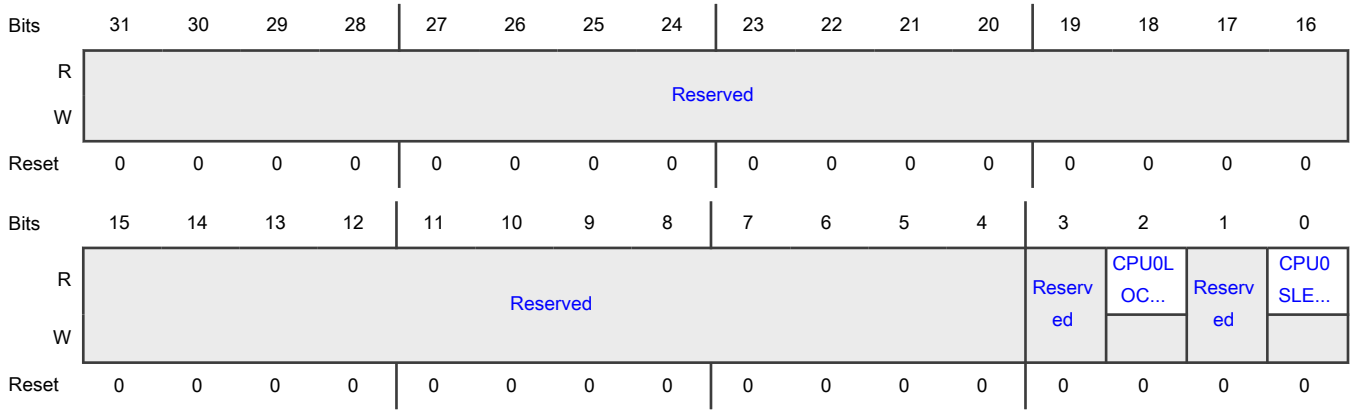
Field	Function
31-1 —	Reserved
0 INTERLEAVE	Controls RAM access for RAMA1 and RAMA2 0b - RAM access is consecutive. 1b - RAM access is interleaved. This setting is need for PKC L0 memory access.

11.5.1.17 CPU Status (CPUSTAT)

Offset

Register	Offset
CPUSTAT	80Ch

Diagram



Fields

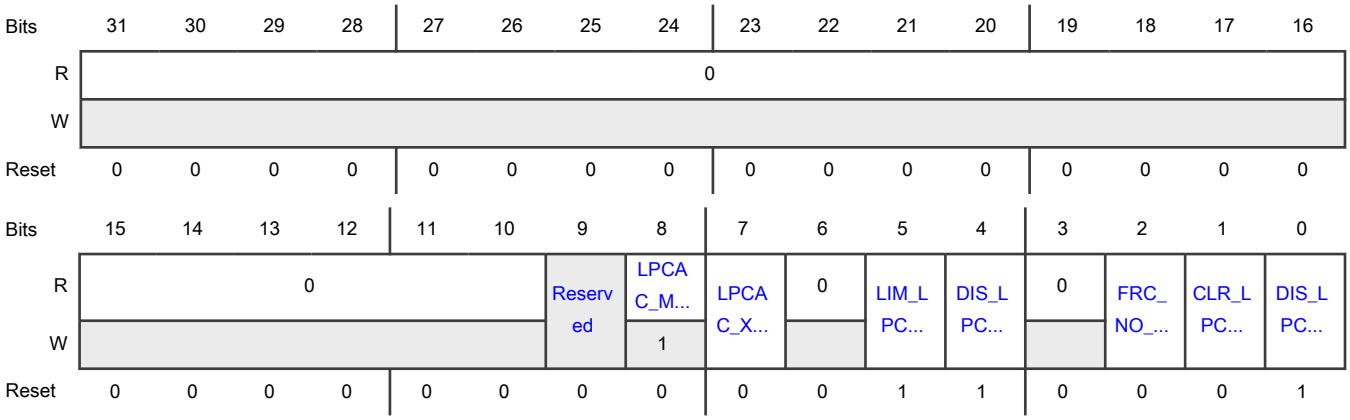
Field	Function
31-4 —	Reserved
3 —	Reserved
2 CPU0LOCKUP	CPU0 lockup state 0b - CPU is not in lockup 1b - CPU is in lockup
1 —	Reserved
0 CPU0SLEEPING	CPU0 sleeping state 0b - CPU is not sleeping 1b - CPU is sleeping

11.5.1.18 LPCAC Control (LPCAC_CTRL)

Offset

Register	Offset
LPCAC_CTRL	824h

Diagram



Fields

Field	Function
31-10 —	Reserved
9 —	Reserved
8 LPCAC_MEM_REQ	Request LPCAC memories. 0b - Configure shared memories RAMX1 as general memories. 1b - Configure shared memories RAMX1 as LPCAC memories, write one lock until a system reset.
7 LPCAC_XOM	LPCAC XOM(eXecute-Only-Memory) attribute control Controls if the instruction fetch attribute is used as part of the address input to the LPCAC. When XOM regions in the internal flash are not configured at the MBC, then this option should be disabled so that instructions and data can be stored within the same cache line. This provides the best cache efficiency for non-XOM applications. When XOM areas in the internal flash are configured at the MBC, then this bit must be set so that instructions and data are cached using separate lines within the LPCAC. 0b - Disabled. 1b - Enabled.
6 —	Reserved
5 LIM_LPCAC_WTBF	Limit LPCAC Write Through Buffer. 0b - Write buffer enabled when transaction is bufferable. 1b - Write buffer enabled when transaction is cacheable and bufferable
4 DIS_LPCAC_WTBF	Disable LPCAC Write Through Buffer. 0b - Enables write through buffer 1b - Disables write through buffer
3 —	Reserved
2 FRC_NO_ALLOC	Forces no allocation. 0b - Forces allocation 1b - Forces no allocation
1 CLR_LPCAC	Clears the cache function. 0b - Unclears the cache 1b - Clears the cache

Table continues on the next page...

Table continued from the previous page...

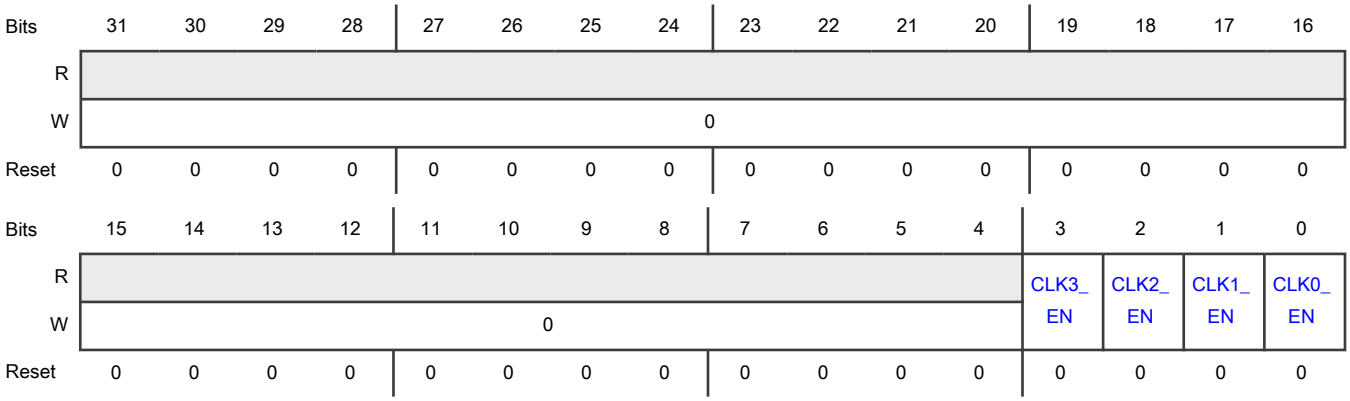
Field	Function
0 DIS_LPCAC	Disables/enables the cache function. 0b - Enabled 1b - Disabled

11.5.1.19 PWM0 Submodule Control (PWM0SUBCTL)

Offset

Register	Offset
PWM0SUBCTL	938h

Diagram



Fields

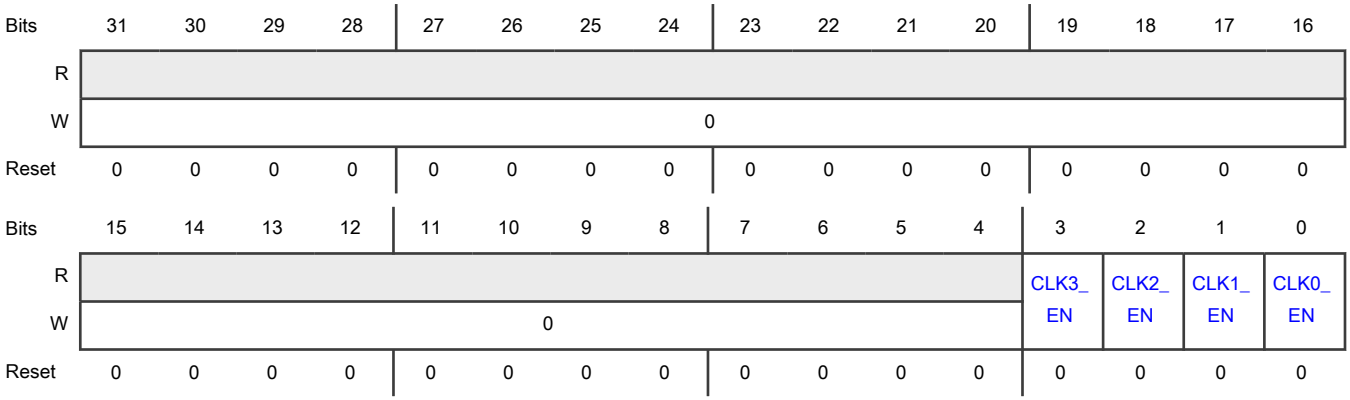
Field	Function
31-16 —	Reserved
15-4 —	Reserved
3-0 CLKn_EN	Enables PWM0 SUB Clockn 0b - Disable 1b - Enable

11.5.1.20 PWM1 Submodule Control (PWM1SUBCTL)

Offset

Register	Offset
PWM1SUBCTL	93Ch

Diagram



Fields

Field	Function
31-16 —	Reserved
15-4 —	Reserved
3-0 CLKn_EN	Enables PWM1 SUB Clockn 0b - Disable 1b - Enable

11.5.1.21 CTIMER Global Start Enable (CTIMERGLOBALSTARTEN)

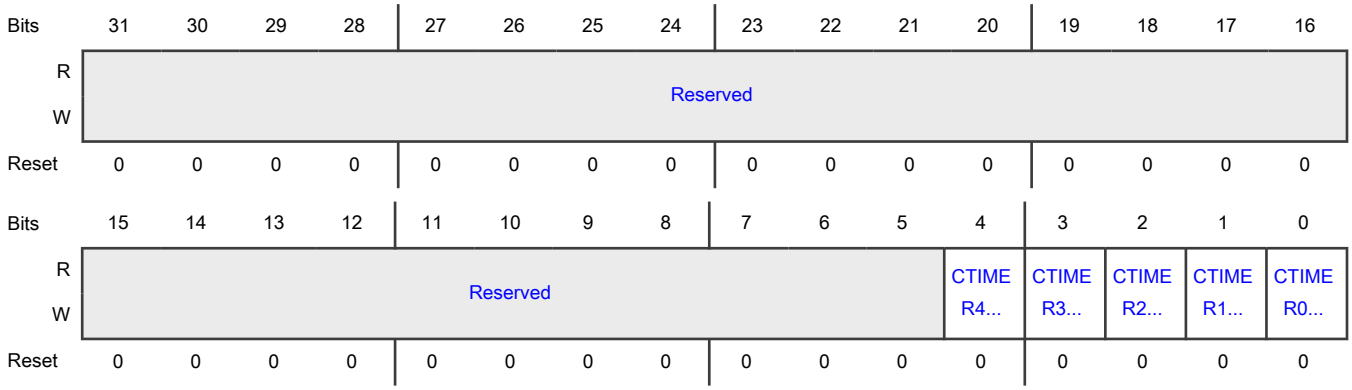
Offset

Register	Offset
CTIMERGLOBALSTARTEN	940h

Function

Refer to CTIMERn TCR.AGCEN

Diagram



Fields

Field	Function
31-5 —	Reserved
4 CTIMER4_CLK_EN	Enables the CTIMER4 function clock 0b - Disable 1b - Enable
3 CTIMER3_CLK_EN	Enables the CTIMER3 function clock 0b - Disable 1b - Enable
2 CTIMER2_CLK_EN	Enables the CTIMER2 function clock 0b - Disable 1b - Enable
1 CTIMER1_CLK_EN	Enables the CTIMER1 function clock 0b - Disable 1b - Enable
0 CTIMER0_CLK_EN	Enables the CTIMER0 function clock 0b - Disable 1b - Enable

11.5.1.22 RAM Control (RAM_CTRL)

Offset

Register	Offset
RAM_CTRL	944h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved								Reserved	Reserved	Reserved	Reserved	RAMC_CG...	RAMB_CG...	RAMX_CG...	RAMA_CG...
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								Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	RAMA_EC...
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Fields

Field	Function
31-24 —	Reserved
23 —	Reserved
22 —	Reserved
21 —	Reserved
20 —	Reserved
19 RAMC_CG_OVERRIDE	RAMC bank clock gating control 0b - Memory bank clock is gated automatically if no access more than 16 clock cycles 1b - Auto clock gating feature is disabled
18 RAMB_CG_OVERRIDE	RAMB bank clock gating control 0b - Memory bank clock is gated automatically if no access more than 16 clock cycles 1b - Auto clock gating feature is disabled
17 RAMX_CG_OVERRIDE	RAMX bank clock gating control 0b - Memory bank clock is gated automatically if no access more than 16 clock cycles 1b - Auto clock gating feature is disabled
16	RAMA bank clock gating control, only avaiable when RAMA_ECC_ENABLE = 0.

Table continues on the next page...

Table continued from the previous page...

Field	Function
RAMA_CG_OVERRIDE	0b - Memory bank clock is gated automatically if no access more than 16 clock cycles. Auto-clkgating using bank_adapter gives a timeout counter for each bank. Once it expires upon no access per specified width (16 cycles), then clock to the bank is stopped high and one cycle penalty will be incurred if a read is the first access after the break). A write will incur no penalty. 1b - Auto clock gating feature is disabled
15-8 —	Reserved
7 —	Reserved
6 —	Reserved
5 —	Reserved
4 —	Reserved
3 —	Reserved
2 —	Reserved
1 —	Reserved
0 RAMA_ECC_ENABLE	RAMA ECC enable 0b - ECC is disabled 1b - ECC is enabled

11.5.1.23 Gray to Binary Converter Gray Code [31:0] (GRAY_CODE_LSB)

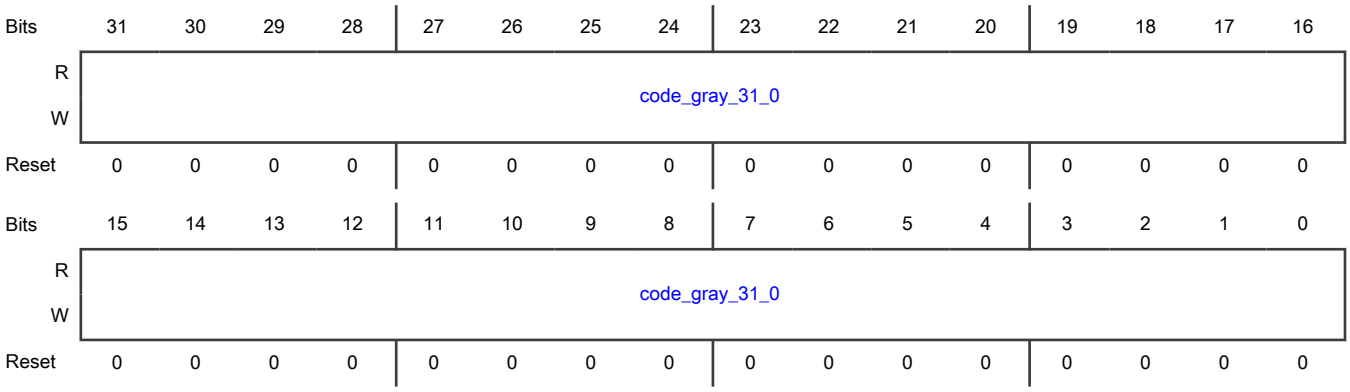
Offset

Register	Offset
GRAY_CODE_LSB	B60h

Function

The Gray Code LSB Input register (CODE_GRAY_LSB) contains the least-significant portion of the Gray code to be converted back to binary.

Diagram



Fields

Field	Function
31-0 code_gray_31_0	Gray code [31:0] CODE_GRAY_LSB is the least-significant 32 bits of the 42-bit Gray code to be converted.

11.5.1.24 Gray to Binary Converter Gray Code [41:32] (GRAY_CODE_MSB)

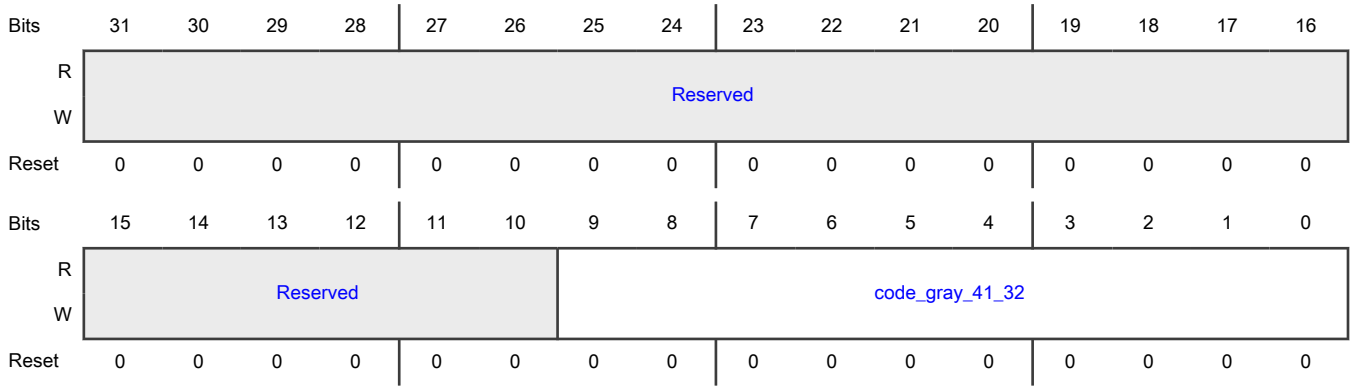
Offset

Register	Offset
GRAY_CODE_MSB	B64h

Function

The Gray Code MSB Input register (CODE_GRAY_MSB) contains the most-significant portion of the Gray code to be converted back to binary.

Diagram



Fields

Field	Function
31-10 —	Reserved
9-0 code_gray_41_32	Gray code [41:32] CODE_GRAY_MSB is the most-significant 10 bits of the 42-bit Gray code to be converted.

11.5.1.25 Gray to Binary Converter Binary Code [31:0] (BINARY_CODE_LSB)

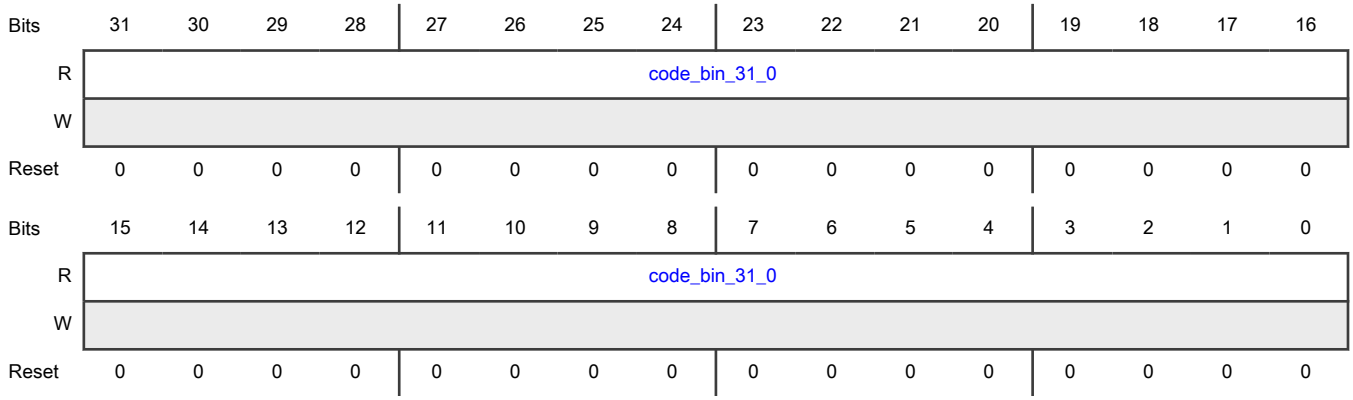
Offset

Register	Offset
BINARY_CODE_LSB	B68h

Function

The Binary Code LSB register (BINARY_CODE_LSB) contains the least-significant portion of the code converted from Gray to binary coding.

Diagram



Fields

Field	Function
31-0 code_bin_31_0	Binary code [31:0] code_bin_31_0 is the least-significant 32 bits of the 42-bit converted code.

11.5.1.26 Gray to Binary Converter Binary Code [41:32] (BINARY_CODE_MSB)

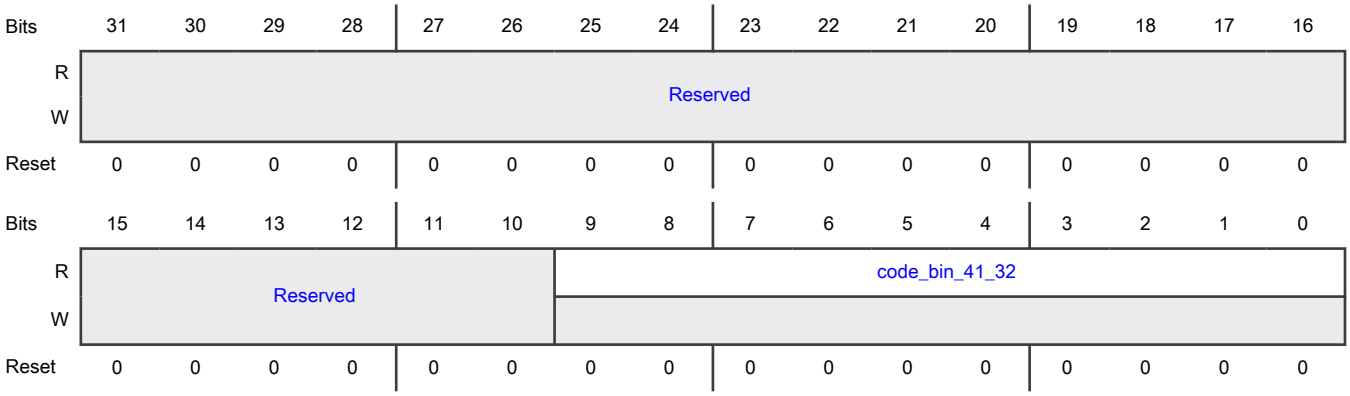
Offset

Register	Offset
BINARY_CODE_MSB	B6Ch

Function

The Binary Code MSB register (BINARY_CODE_MSB) contains the most-significant portion of the code converted from Gray to binary coding.

Diagram



Fields

Field	Function
31-10 —	Reserved
9-0 code_bin_41_32	Binary code [41:32] code_bin_41_32 is the most-significant 10 bits of the 42-bit converted code.

11.5.1.27 MSF Configuration (MSFCFG)

Offset

Register	Offset
MSFCFG	E1Ch

Function

Protected by GLIKEY CRTL_0.WRITE_INDEX[7:0]=3

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	Reserved							MASS_ER...	Reserved				IFR_ERA...	IFR_ERA...	IFR_ERA...	IFR_ERA...
W								1					1	1	1	1
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-9 —	Reserved
8 MASS_ERASE_DIS	Mass erase control 0b - Enables mass erase 1b - Disables mass erase, write one lock until a system reset.
7-4 —	Reserved
3-0 IFR_ERASE_DISn	user IFR sector n erase control 0b - Enable IFR sector erase. 1b - Disable IFR sector erase, write one lock until a system reset.

11.5.1.28 ROP State Register (ROP_STATE)

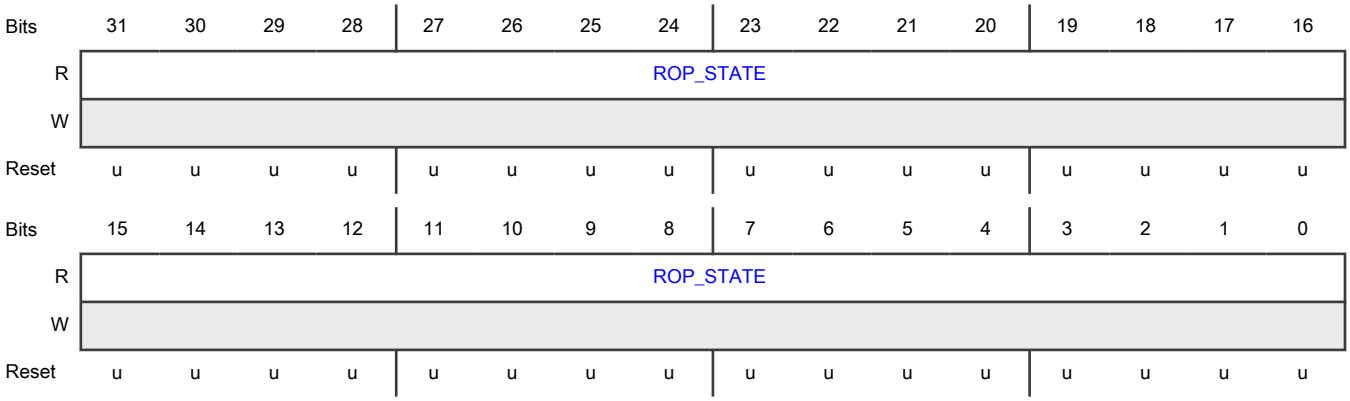
Offset

Register	Offset
ROP_STATE	E3Ch

Function

This register is written by Boot-ROM code and is only one-time write and locked down until a system reset.

Diagram



Fields

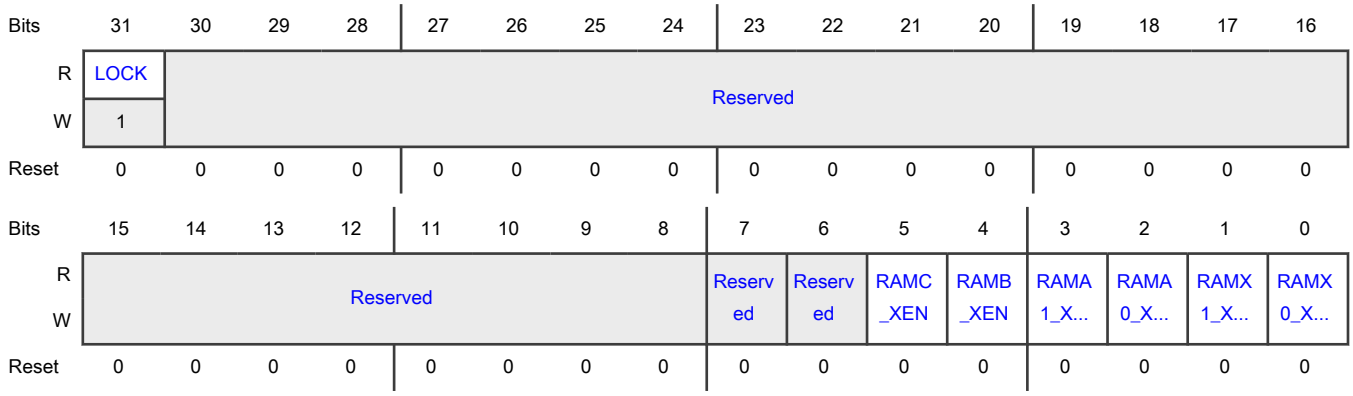
Field	Function
31-0 ROP_STATE	ROP state

11.5.1.29 RAM XEN Control (SRAM_XEN)

Offset

Register	Offset
SRAM_XEN	E58h

Diagram



Fields

Field	Function
31 LOCK	This 1-bit field provides a mechanism to limit writes to this register (and SRAM_XEN_DP) to protect its contents. Once set, this bit remains asserted until a system reset. 0b - This register is not locked and can be altered. 1b - This register is locked and cannot be altered.
30-8 —	Reserved
7 —	Reserved
6 —	Reserved
5 RAMC_XEN	RAMCx Execute permission control. 0b - Execute permission is disabled, R/W are enabled. 1b - Execute permission is enabled, R/W/X are enabled.
4 RAMB_XEN	RAMBx Execute permission control. 0b - Execute permission is disabled, R/W are enabled. 1b - Execute permission is enabled, R/W/X are enabled.
3 RAMA1_XEN	RAMAx (excepts RAMA0) Execute permission control. 0b - Execute permission is disabled, R/W are enabled. 1b - Execute permission is enabled, R/W/X are enabled.
2 RAMA0_XEN	RAMA0 Execute permission control. 0b - Execute permission is disabled, R/W are enabled. 1b - Execute permission is enabled, R/W/X are enabled.

Table continues on the next page...

Table continued from the previous page...

Field	Function
1 RAMX1_XEN	RAMX1 Execute permission control. 0b - Execute permission is disabled, R/W are enabled. 1b - Execute permission is enabled, R/W/X are enabled.
0 RAMX0_XEN	RAMX0 Execute permission control. 0b - Execute permission is disabled, R/W are enabled. 1b - Execute permission is enabled, R/W/X are enabled.

11.5.1.30 RAM XEN Control (Duplicate) (SRAM_XEN_DP)

Offset

Register	Offset
SRAM_XEN_DP	E5Ch

Function

Protected by GLIKEY CRTL_0.WRITE_INDEX[7:0]=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								Reserved	Reserved	RAMC_XEN	RAMB_XEN	RAMA1_X...	RAMA0_X...	RAMX1_X...	RAMX0_X...
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-8 —	Reserved
7 —	Reserved

Table continues on the next page...

Table continued from the previous page...

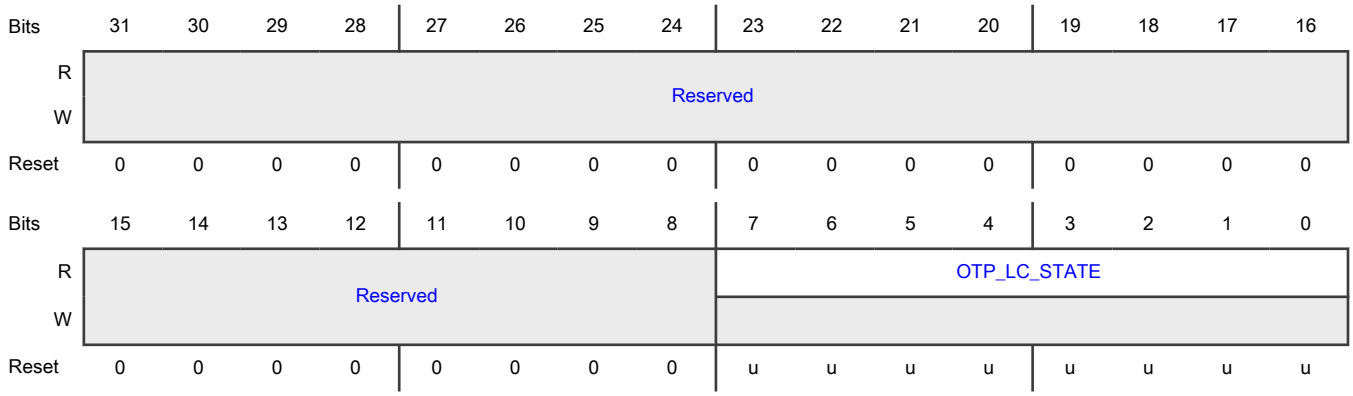
Field	Function
6 —	Reserved
5 RAMC_XEN	Refer to SRAM_XEN for more details.
4 RAMB_XEN	Refer to SRAM_XEN for more details.
3 RAMA1_XEN	Refer to SRAM_XEN for more details.
2 RAMA0_XEN	Refer to SRAM_XEN for more details.
1 RAMX1_XEN	Refer to SRAM_XEN for more details.
0 RAMX0_XEN	Refer to SRAM_XEN for more details.

11.5.1.31 Life Cycle State Register (ELS_OTP_LC_STATE)

Offset

Register	Offset
ELS_OTP_LC_STATE	E80h

Diagram



Fields

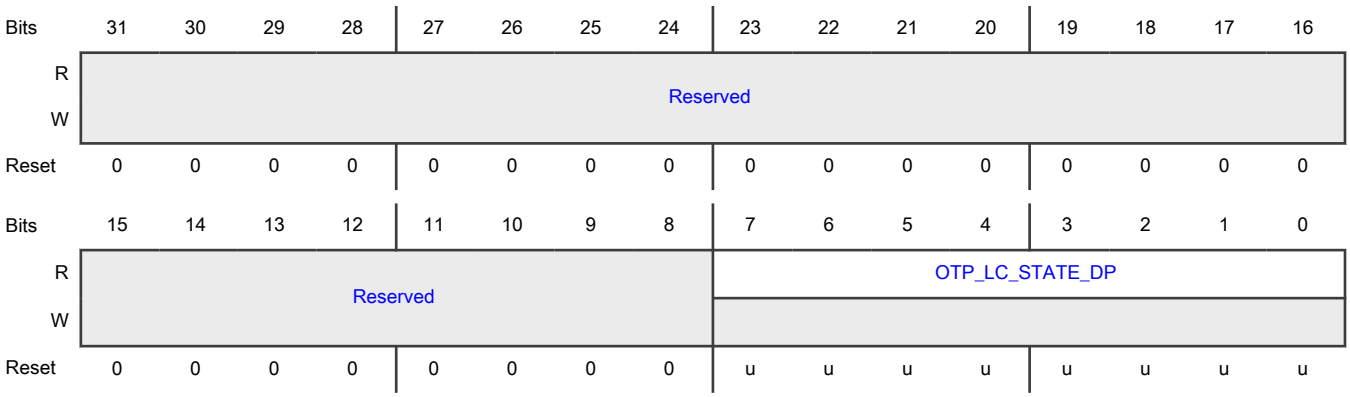
Field	Function
31-8 —	Reserved
7-0 OTP_LC_STATE	OTP life cycle state

11.5.1.32 Life Cycle State Register (Duplicate) (ELS_OTP_LC_STATE_DP)

Offset

Register	Offset
ELS_OTP_LC_STATE_DP	E84h

Diagram



Fields

Field	Function
31-8 —	Reserved
7-0 OTP_LC_STATE_DP	OTP life cycle state

11.5.1.33 Control Write Access to Security (DEBUG_LOCK_EN)

Offset

Register	Offset
DEBUG_LOCK_EN	FA0h

Function

Controls write access to the SWD_ACCESS_CPU0, DEBUG_FEATURES, DEBUG_FEATURES_DP and DEBUG_AUTH_BEACON registers

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												LOCK_ALL			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

Fields

Field	Function
31-4 —	Reserved
3-0 LOCK_ALL	Controls write access to the security registers Controls write access to the SWD_ACCESS_CPU0, DEBUG_FEATURES, DEBUG_FEATURES_DP and DEBUG_AUTH_BEACON registers 0000b - Any other value than b1010: disables write access to all registers 1010b - Enables write access to all registers

11.5.1.34 Cortex Debug Features Control (DEBUG_FEATURES)

Offset

Register	Offset
DEBUG_FEATURES	FA4h

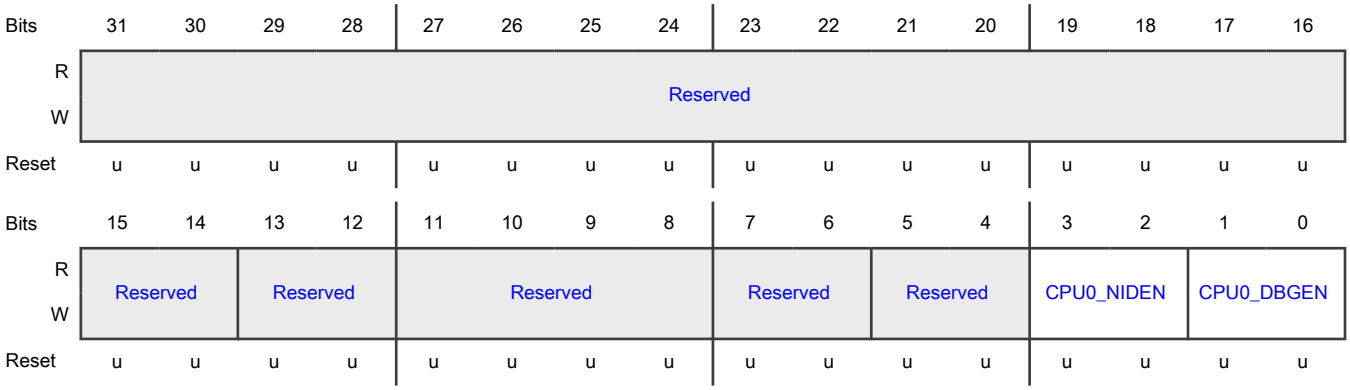
Function

Cortex M33 (CPU0) debug features control.

NOTE

Reset on PoR only

Diagram



Fields

Field	Function
31-14 —	Reserved
13-12 —	Reserved
11-8 —	Reserved
7-6 —	Reserved
5-4 —	Reserved
3-2 CPU0_NIDEN	CPU0 non-invasive debug control Enables non-invasive debug for CPU0. Values not listed are reserved. 01b - Disables debug 10b - Enables debug
1-0 CPU0_DBGEN	CPU0 invasive debug control Enables invasive debug for CPU0. Values not listed are reserved. 01b - Disables debug 10b - Enables debug

11.5.1.35 Cortex Debug Features Control (Duplicate) (DEBUG_FEATURES_DP)

Offset

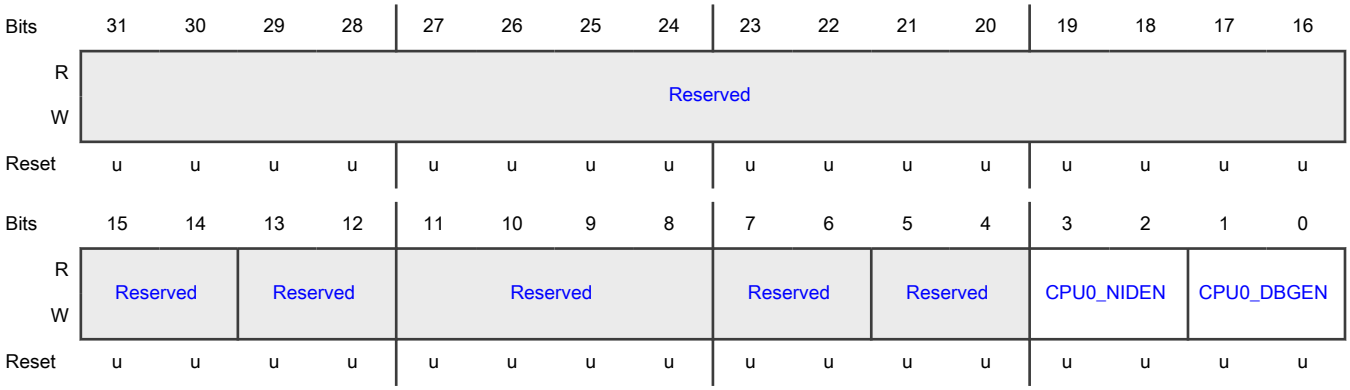
Register	Offset
DEBUG_FEATURES_DP	FA8h

Function

Cortex M33 (CPU0) debug features control. Protected by GLIKEY CRTL_0.WRITE_INDEX[7:0]=1

with GLIKEY reset function

Diagram



Fields

Field	Function
31-14 —	Reserved
13-12 —	Reserved
11-8 —	Reserved
7-6 —	Reserved
5-4 —	Reserved
3-2 CPU0_NIDEN	CPU0 non-invasive debug control Enables non-invasive debug for CPU0. Values not listed are reserved.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	01b - Disables debug 10b - Enables debug
1-0 CPU0_DBGEN	CPU0 invasive debug control Enables invasive debug for CPU0. Values not listed are reserved. 01b - Disables debug 10b - Enables debug

11.5.1.36 CPU0 Software Debug Access (SWD_ACCESS_CPU0)

Offset

Register	Offset
SWD_ACCESS_CPU0	FB4h

Function

This register is used by ROM during DEBUG authentication mechanism to enable debug access port for CPU0. write once only.

NOTE

Reset on PoR only

Diagram

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

Fields

Field	Function
31-0 SEC_CODE	CPU0 SWD-AP: 0x12345678

Table continues on the next page...

Field	Function
	0000_0000_0000_0000_0000_0000_0000_0000b - CPU0 DAP is not allowed. Reading back register is read as 0x5. 0001_0010_0011_0100_0101_0110_0111_1000b - Value to write to enable CPU0 SWD access. Reading back register is read as 0xA.

11.5.1.37 Debug Authentication BEACON (DEBUG_AUTH_BEACON)

Offset

Register	Offset
DEBUG_AUTH_BEACON	FC0h

Function

This register protected by security. ROM sets register (read-only) with value received in debug credentials before passing control to the user code. This can be used to extend debug authentication control for the customer application.

NOTE

Reset on PoR only

Diagram

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

Fields

Field	Function
31-0 BEACON	Sets by the debug authentication code in ROM to pass the debug beacons (Credential Beacon and Authentication Beacon) to the application code.

11.5.1.38 JTAG Chip ID (JTAG_ID)

Offset

Register	Offset
JTAG_ID	FF0h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	JTAG_ID															
W																
Reset	0	0	0	0	0	1	1	1	0	0	1	0	0	1	1	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	JTAG_ID															
W																
Reset	1	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1

Fields

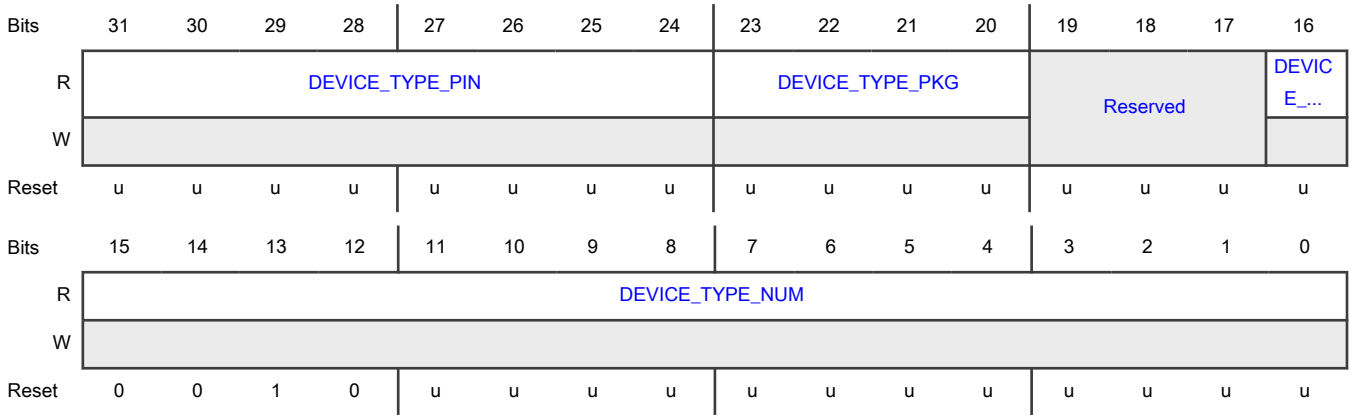
Field	Function
31-0 JTAG_ID	Indicates the device ID

11.5.1.39 Device Type (DEVICE_TYPE)

Offset

Register	Offset
DEVICE_TYPE	FF4h

Diagram



Fields

Field	Function
31-24 DEVICE_TYPE_PIN	Indicates the device's pin number
23-20 DEVICE_TYPE_PKG	Indicates the device's package type 0000b - HLQFP 0001b - HTQFP 0010b - BGA 0011b - HDQFP 0100b - QFN 0101b - CSP 0110b - LQFP
19-17 —	Reserved
16 DEVICE_TYPE_SEC	Indicates the device type 0b - Non Secure 1b - Secure
15-0 DEVICE_TYPE_NUM	Indicates the device part number [15:12]: ... 0010b: Axxx ... [11:0]: 3 digital of the part unmber.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	For example: A156: 0x2156 A176: 0x2176 A256: 0x2256 A346: 0x2346

11.5.1.40 Device ID (DEVICE_ID0)

Offset

Register	Offset
DEVICE_ID0	FF8h

Function

This register contains the device ID.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved				SECURITY				ROM_REV_MINOR				Reserved			
W																
Reset	0	0	0	0	u	u	u	u	u	u	u	u	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved								FLASH_SIZE				RAM_SIZE			
W																
Reset	0	0	0	0	0	0	0	0	u	u	u	u	u	u	u	u

Fields

Field	Function
31-28 —	Reserved
27-24 SECURITY	0101b - Secure version. (All values other than 1010b represent the secure version.) 1010b - Non secure version.
23-20	Indicates the device's ROM revision

Table continues on the next page...

Table continued from the previous page...

Field	Function
ROM_REV_MINOR	
19-8 —	Reserved
7-4 FLASH_SIZE	Indicates the device's flash size 0000b - 32KB. 0001b - 64KB. 0010b - 128KB. 0011b - 256KB. 0100b - 512KB. 0101b - 768KB. 0110b - 1MB. 0111b - 1.5MB. 1000b - 2MB.
3-0 RAM_SIZE	Indicates the device's ram size 0000b - 8KB. 0001b - 16KB. 0010b - 32KB. 0011b - 64KB. 0100b - 96KB. 0101b - 128KB. 0110b - 160KB. 0111b - 192KB. 1000b - 256KB. 1001b - 288KB. 1010b - 352KB. 1011b - 512KB.

11.5.2 MRCC register descriptions

11.5.2.1 MRCC memory map

SYSCON.MRCC base address: 4009_1000h

Offset	Register	Width (In bits)	Access	Reset value
0h	Peripheral Reset Control 0 (MRCC_GLB_RST0)	32	RW	0000_0000h
4h	Peripheral Reset Control Set 0 (MRCC_GLB_RST0_SET)	32	W	0000_0000h
8h	Peripheral Reset Control Clear 0 (MRCC_GLB_RST0_CLR)	32	W	0000_0000h
10h	Peripheral Reset Control 1 (MRCC_GLB_RST1)	32	RW	0000_0000h
14h	Peripheral Reset Control Set 1 (MRCC_GLB_RST1_SET)	32	W	0000_0000h
18h	Peripheral Reset Control Clear 1 (MRCC_GLB_RST1_CLR)	32	W	0000_0000h
20h	Peripheral Reset Control 2 (MRCC_GLB_RST2)	32	RW	0000_0000h
24h	Peripheral Reset Control Set 2 (MRCC_GLB_RST2_SET)	32	W	0000_0000h
28h	Peripheral Reset Control Clear 2 (MRCC_GLB_RST2_CLR)	32	W	0000_0000h
40h	AHB Clock Control 0 (MRCC_GLB_CC0)	32	RW	0001_0000h
44h	AHB Clock Control Set 0 (MRCC_GLB_CC0_SET)	32	W	0001_0000h
48h	AHB Clock Control Clear 0 (MRCC_GLB_CC0_CLR)	32	W	0001_0000h
50h	AHB Clock Control 1 (MRCC_GLB_CC1)	32	RW	0000_0000h
54h	AHB Clock Control Set 1 (MRCC_GLB_CC1_SET)	32	W	0000_0000h
58h	AHB Clock Control Clear 1 (MRCC_GLB_CC1_CLR)	32	W	0000_0000h
60h	AHB Clock Control 2 (MRCC_GLB_CC2)	32	RW	0000_0000h
64h	AHB Clock Control Set 2 (MRCC_GLB_CC2_SET)	32	W	0000_0000h
68h	AHB Clock Control Clear 2 (MRCC_GLB_CC2_CLR)	32	W	0000_0000h
80h	Control Automatic Clock Gating 0 (MRCC_GLB_ACC0)	32	RW	0001_0200h
84h	Control Automatic Clock Gating 1 (MRCC_GLB_ACC1)	32	RW	0000_0000h
88h	Control Automatic Clock Gating 2 (MRCC_GLB_ACC2)	32	RW	0000_040Eh
A0h	I3C0_FCLK clock selection control (MRCC_I3C0_FCLK_CLKSEL)	32	RW	0000_0007h
A4h	I3C0_FCLK clock divider control (MRCC_I3C0_FCLK_CLKDIV)	32	RW	4000_0000h
A8h	CTIMER0 clock selection control (MRCC_CTIMER0_CLKSEL)	32	RW	0000_0007h
ACh	CTIMER0 clock divider control (MRCC_CTIMER0_CLKDIV)	32	RW	4000_0000h
B0h	CTIMER1 clock selection control (MRCC_CTIMER1_CLKSEL)	32	RW	0000_0007h
B4h	CTIMER1 clock divider control (MRCC_CTIMER1_CLKDIV)	32	RW	4000_0000h
B8h	CTIMER2 clock selection control (MRCC_CTIMER2_CLKSEL)	32	RW	0000_0007h
BCh	CTIMER2 clock divider control (MRCC_CTIMER2_CLKDIV)	32	RW	4000_0000h
C0h	CTIMER3 clock selection control (MRCC_CTIMER3_CLKSEL)	32	RW	0000_0007h
C4h	CTIMER3 clock divider control (MRCC_CTIMER3_CLKDIV)	32	RW	4000_0000h
C8h	CTIMER4 clock selection control (MRCC_CTIMER4_CLKSEL)	32	RW	0000_0007h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
CCh	CTIMER4 clock divider control (MRCC_CTIMER4_CLKDIV)	32	RW	4000_0000h
D4h	WWDT0 clock divider control (MRCC_WWDT0_CLKDIV)	32	RW	0000_0000h
D8h	FLEXIO0 clock selection control (MRCC_FLEXIO0_CLKSEL)	32	RW	0000_0007h
DCh	FLEXIO0 clock divider control (MRCC_FLEXIO0_CLKDIV)	32	RW	4000_0000h
E0h	LPI2C0 clock selection control (MRCC_LPI2C0_CLKSEL)	32	RW	0000_0007h
E4h	LPI2C0 clock divider control (MRCC_LPI2C0_CLKDIV)	32	RW	4000_0000h
E8h	LPI2C1 clock selection control (MRCC_LPI2C1_CLKSEL)	32	RW	0000_0007h
ECh	LPI2C1 clock divider control (MRCC_LPI2C1_CLKDIV)	32	RW	4000_0000h
F0h	LPSPI0 clock selection control (MRCC_LPSPI0_CLKSEL)	32	RW	0000_0007h
F4h	LPSPI0 clock divider control (MRCC_LPSPI0_CLKDIV)	32	RW	4000_0000h
F8h	LPSPI1 clock selection control (MRCC_LPSPI1_CLKSEL)	32	RW	0000_0007h
FCh	LPSPI1 clock divider control (MRCC_LPSPI1_CLKDIV)	32	RW	4000_0000h
100h	LPUART0 clock selection control (MRCC_LPUART0_CLKSEL)	32	RW	0000_0007h
104h	LPUART0 clock divider control (MRCC_LPUART0_CLKDIV)	32	RW	4000_0000h
108h	LPUART1 clock selection control (MRCC_LPUART1_CLKSEL)	32	RW	0000_0007h
10Ch	LPUART1 clock divider control (MRCC_LPUART1_CLKDIV)	32	RW	4000_0000h
110h	LPUART2 clock selection control (MRCC_LPUART2_CLKSEL)	32	RW	0000_0007h
114h	LPUART2 clock divider control (MRCC_LPUART2_CLKDIV)	32	RW	4000_0000h
118h	LPUART3 clock selection control (MRCC_LPUART3_CLKSEL)	32	RW	0000_0007h
11Ch	LPUART3 clock divider control (MRCC_LPUART3_CLKDIV)	32	RW	4000_0000h
120h	LPUART4 clock selection control (MRCC_LPUART4_CLKSEL)	32	RW	0000_0007h
124h	LPUART4 clock divider control (MRCC_LPUART4_CLKDIV)	32	RW	4000_0000h
128h	USB0 clock selection control (MRCC_USB0_CLKSEL)	32	RW	0000_0003h
12Ch	USB0 clock divider control (MRCC_USB0_CLKDIV)	32	RW	4000_0000h
130h	LPTMR0 clock selection control (MRCC_LPTMR0_CLKSEL)	32	RW	0000_0007h
134h	LPTMR0 clock divider control (MRCC_LPTMR0_CLKDIV)	32	RW	4000_0000h
138h	OSTIMER0 clock selection control (MRCC_OSTIMER0_CLKSEL)	32	RW	0000_0003h
140h	ADCx clock selection control (MRCC_ADC_CLKSEL)	32	RW	0000_0007h
144h	ADCx clock divider control (MRCC_ADC_CLKDIV)	32	RW	4000_0000h
14Ch	CMP0_FUNC clock divider control (MRCC_CMP0_FUNC_CLKDIV)	32	RW	4000_0000h
150h	CMP0_RR clock selection control (MRCC_CMP0_RR_CLKSEL)	32	RW	0000_0007h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
154h	CMP0_RR clock divider control (MRCC_CMP0_RR_CLKDIV)	32	RW	4000_0000h
15Ch	CMP1_FUNC clock divider control (MRCC_CMP1_FUNC_CLKDIV)	32	RW	4000_0000h
160h	CMP1_RR clock selection control (MRCC_CMP1_RR_CLKSEL)	32	RW	0000_0007h
164h	CMP1_RR clock divider control (MRCC_CMP1_RR_CLKDIV)	32	RW	4000_0000h
16Ch	CMP2_FUNC clock divider control (MRCC_CMP2_FUNC_CLKDIV)	32	RW	4000_0000h
170h	CMP2_RR clock selection control (MRCC_CMP2_RR_CLKSEL)	32	RW	0000_0007h
174h	CMP2_RR clock divider control (MRCC_CMP2_RR_CLKDIV)	32	RW	4000_0000h
178h	DAC0 clock selection control (MRCC_DAC0_CLKSEL)	32	RW	0000_0007h
17Ch	DAC0 clock divider control (MRCC_DAC0_CLKDIV)	32	RW	4000_0000h
180h	FLEXCAN0 clock selection control (MRCC_FLEXCAN0_CLKSEL)	32	RW	0000_0007h
184h	FLEXCAN0 clock divider control (MRCC_FLEXCAN0_CLKDIV)	32	RW	4000_0000h
188h	FLEXCAN1 clock selection control (MRCC_FLEXCAN1_CLKSEL)	32	RW	0000_0007h
18Ch	FLEXCAN1 clock divider control (MRCC_FLEXCAN1_CLKDIV)	32	RW	4000_0000h
190h	LPI2C2 clock selection control (MRCC_LPI2C2_CLKSEL)	32	RW	0000_0007h
194h	LPI2C2 clock divider control (MRCC_LPI2C2_CLKDIV)	32	RW	4000_0000h
198h	LPI2C3 clock selection control (MRCC_LPI2C3_CLKSEL)	32	RW	0000_0007h
19Ch	LPI2C3 clock divider control (MRCC_LPI2C3_CLKDIV)	32	RW	4000_0000h
1A0h	LPUART5 clock selection control (MRCC_LPUART5_CLKSEL)	32	RW	0000_0007h
1A4h	LPUART5 clock divider control (MRCC_LPUART5_CLKDIV)	32	RW	4000_0000h
1A8h	DBG_TRACE clock selection control (MRCC_DBG_TRACE_CLKSEL)	32	RW	0000_0000h
1ACh	DBG_TRACE clock divider control (MRCC_DBG_TRACE_CLKDIV)	32	RW	0000_0000h
1B0h	CLKOUT clock selection control (MRCC_CLKOUT_CLKSEL)	32	RW	0000_0007h
1B4h	CLKOUT clock divider control (MRCC_CLKOUT_CLKDIV)	32	RW	4000_0000h

11.5.2.2 Peripheral Reset Control 0 (MRCC_GLB_RST0)

Offset

Register	Offset
MRCC_GLB_RST0	0h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	FLEXPWM0	QDC1	QDC0	USB0	LPUART4	LPUART3	LPUART2	LPUART1	LPUART0	LPSPI1	LPSPI0	LPI2C1	LPI2C0	FLEXIO0	AOI1	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	ERM0	EIM0	CRC0	AOI0	DMA0	SMARTDM...	Reserved	UTICK0	FREQME	CTIMER4	CTIMER3	CTIMER2	CTIMER1	CTIMER0	I3C0	INPUTMU...
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31 FLEXPWM0	FLEXPWM0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
30 QDC1	QDC1 0b - Peripheral is held in reset 1b - Peripheral is released from reset
29 QDC0	QDC0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
28 USB0	USB0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
27 LPUART4	LPUART4 0b - Peripheral is held in reset 1b - Peripheral is released from reset
26 LPUART3	LPUART3 0b - Peripheral is held in reset 1b - Peripheral is released from reset
25 LPUART2	LPUART2 0b - Peripheral is held in reset 1b - Peripheral is released from reset
24	LPUART1

Table continues on the next page...

Table continued from the previous page...

Field	Function
LPUART1	0b - Peripheral is held in reset 1b - Peripheral is released from reset
23 LPUART0	LPUART0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
22 LPSPI1	LPSPI1 0b - Peripheral is held in reset 1b - Peripheral is released from reset
21 LPSPI0	LPSPI0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
20 LPI2C1	LPI2C1 0b - Peripheral is held in reset 1b - Peripheral is released from reset
19 LPI2C0	LPI2C0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
18 FLEXIO0	FLEXIO0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
17 AOI1	AOI1 0b - Peripheral is held in reset 1b - Peripheral is released from reset
16 —	reserved reserved
15 ERM0	ERM0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
14 EIM0	EIM0 0b - Peripheral is held in reset 1b - Peripheral is released from reset

Table continues on the next page...

Table continued from the previous page...

Field	Function
13 CRC0	CRC0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
12 AOI0	AOI0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
11 DMA0	DMA0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
10 SMARTDMA0	SMARTDMA0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
9 —	reserved reserved
8 UTICK0	UTICK0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
7 FREQME	FREQME 0b - Peripheral is held in reset 1b - Peripheral is released from reset
6 CTIMER4	CTIMER4 0b - Peripheral is held in reset 1b - Peripheral is released from reset
5 CTIMER3	CTIMER3 0b - Peripheral is held in reset 1b - Peripheral is released from reset
4 CTIMER2	CTIMER2 0b - Peripheral is held in reset 1b - Peripheral is released from reset
3 CTIMER1	CTIMER1 0b - Peripheral is held in reset 1b - Peripheral is released from reset

Table continues on the next page...

Table continued from the previous page...

Field	Function
2 CTIMER0	CTIMER0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
1 I3C0	I3C0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
0 INPUTMUX0	INPUTMUX0 0b - Peripheral is held in reset 1b - Peripheral is released from reset

11.5.2.3 Peripheral Reset Control Set n (MRCC_GLB_RST0_SET - MRCC_GLB_RST2_SET)

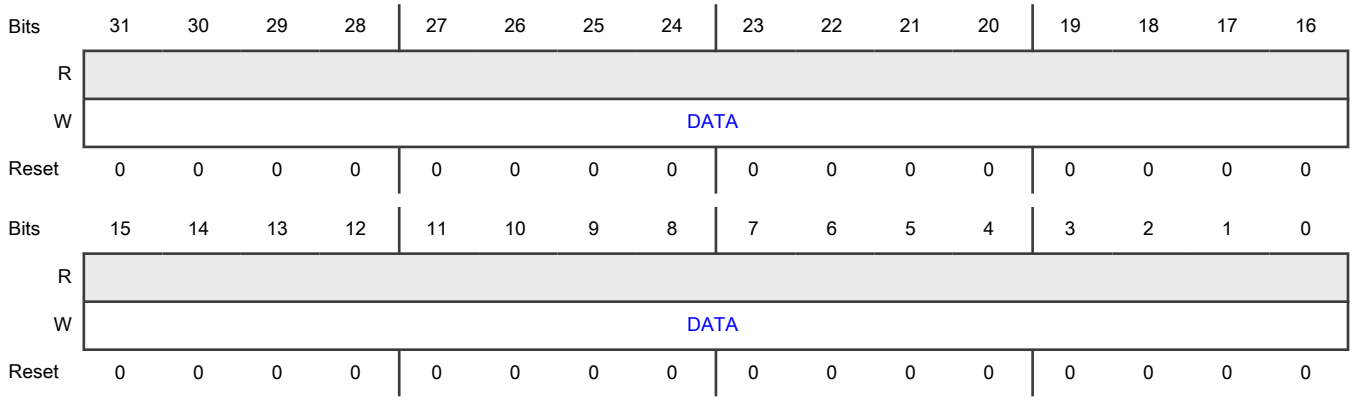
Offset

Register	Offset
MRCC_GLB_RST0_SET	4h
MRCC_GLB_RST1_SET	14h
MRCC_GLB_RST2_SET	24h

Function

Writing a 1 to a bit position in a write-only MRCC_GLB_RSTn_SET register sets the corresponding position in MRCC_GLB_RSTn.

Diagram



Fields

Field	Function
31-0 DATA	Data array value, refer to corresponding position in MRCC_GLB_RSTn.

11.5.2.4 Peripheral Reset Control Clear n (MRCC_GLB_RST0_CLR - MRCC_GLB_RST2_CLR)

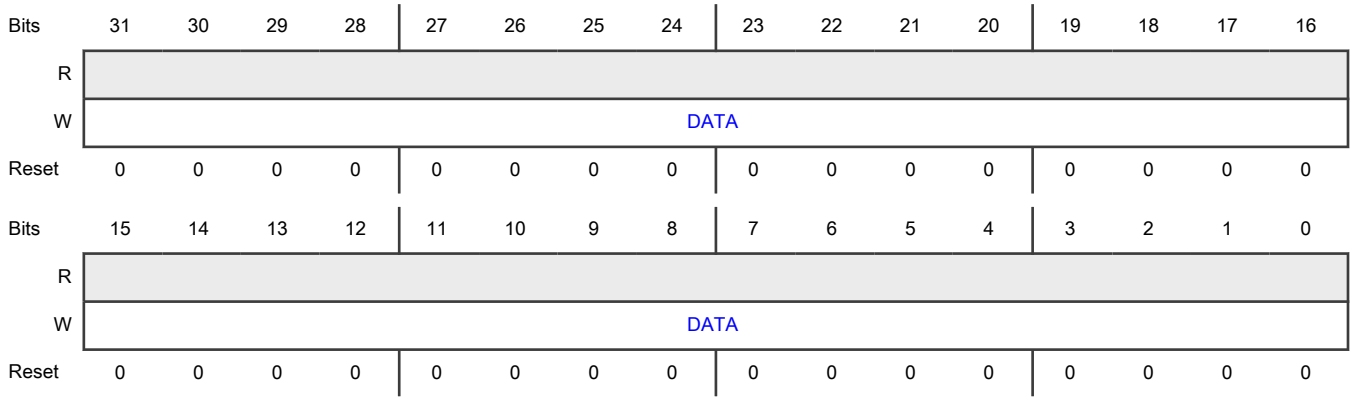
Offset

Register	Offset
MRCC_GLB_RST0_CLR	8h
MRCC_GLB_RST1_CLR	18h
MRCC_GLB_RST2_CLR	28h

Function

Writing a 1 to a bit position in a write-only MRCC_GLB_RSTn_CLR register clears the corresponding position in MRCC_GLB_RSTn.

Diagram



Fields

Field	Function
31-0 DATA	Data array value, refer to corresponding position in MRCC_GLB_RSTn.

11.5.2.5 Peripheral Reset Control 1 (MRCC_GLB_RST1)

Offset

Register	Offset
MRCC_GLB_RST1	10h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved	Reserved	ADC3	ADC2	Reserved	TRNG0	Reserved	PKC0	Reserved	LPUART5	LPI2C3	LPI2C2	FLEXCAN1	FLEXCAN0	SLCD0	PORT4
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	PORT3	PORT2	PORT1	PORT0	OPAMP3	OPAMP2	OPAMP1	OPAMP0	DAC0	CMP2	CMP1	Reserved	ADC1	ADC0	OSTIMER0	FLEXPWM1
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31 —	reserved reserved
30 —	Reserved
29 ADC3	ADC3 0b - Peripheral is held in reset 1b - Peripheral is released from reset
28 ADC2	ADC2 0b - Peripheral is held in reset 1b - Peripheral is released from reset
27 —	reserved reserved
26 TRNG0	TRNG0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
25	reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	reserved
24 PKC0	PKC0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
23 —	reserved reserved
22 LPUART5	LPUART5 0b - Peripheral is held in reset 1b - Peripheral is released from reset
21 LPI2C3	LPI2C3 0b - Peripheral is held in reset 1b - Peripheral is released from reset
20 LPI2C2	LPI2C2 0b - Peripheral is held in reset 1b - Peripheral is released from reset
19 FLEXCAN1	FLEXCAN1 0b - Peripheral is held in reset 1b - Peripheral is released from reset
18 FLEXCAN0	FLEXCAN0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
17 SLCD0	SLCD0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
16 PORT4	PORT4 0b - Peripheral is held in reset 1b - Peripheral is released from reset
15 PORT3	PORT3 0b - Peripheral is held in reset 1b - Peripheral is released from reset
14	PORT2

Table continues on the next page...

Table continued from the previous page...

Field	Function
PORT2	0b - Peripheral is held in reset 1b - Peripheral is released from reset
13 PORT1	PORT1 0b - Peripheral is held in reset 1b - Peripheral is released from reset
12 PORT0	PORT0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
11 OPAMP3	OPAMP3 0b - Peripheral is held in reset 1b - Peripheral is released from reset
10 OPAMP2	OPAMP2 0b - Peripheral is held in reset 1b - Peripheral is released from reset
9 OPAMP1	OPAMP1 0b - Peripheral is held in reset 1b - Peripheral is released from reset
8 OPAMP0	OPAMP0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
7 DAC0	DAC0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
6 CMP2	CMP2 0b - Peripheral is held in reset 1b - Peripheral is released from reset
5 CMP1	CMP1 0b - Peripheral is held in reset 1b - Peripheral is released from reset
4 —	reserved reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
3 ADC1	ADC1 0b - Peripheral is held in reset 1b - Peripheral is released from reset
2 ADC0	ADC0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
1 OSTIMER0	OSTIMER0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
0 FLEXPWM1	FLEXPWM1 0b - Peripheral is held in reset 1b - Peripheral is released from reset

11.5.2.6 Peripheral Reset Control 2 (MRCC_GLB_RST2)

Offset

Register	Offset
MRCC_GLB_RST2	20h

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						MAU0	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0	Reserved			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-10	reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	reserved
9 MAU0	MAU0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
8 GPIO4	GPIO4 0b - Peripheral is held in reset 1b - Peripheral is released from reset
7 GPIO3	GPIO3 0b - Peripheral is held in reset 1b - Peripheral is released from reset
6 GPIO2	GPIO2 0b - Peripheral is held in reset 1b - Peripheral is released from reset
5 GPIO1	GPIO1 0b - Peripheral is held in reset 1b - Peripheral is released from reset
4 GPIO0	GPIO0 0b - Peripheral is held in reset 1b - Peripheral is released from reset
3-0 —	reserved reserved

11.5.2.7 AHB Clock Control 0 (MRCC_GLB_CC0)

Offset

Register	Offset
MRCC_GLB_CC0	40h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	FLEXPWM0	QDC1	QDC0	USB0	LPUART4	LPUART3	LPUART2	LPUART1	LPUART0	LPSPI1	LPSPI0	LPI2C1	LPI2C0	FLEXIO0	AOI1	Reserved
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	ERM0	EIM0	CRC0	AOI0	DMA0	SMARTDM...	WWDT0	UTICK0	FREQME	CTIMER4	CTIMER3	CTIMER2	CTIMER1	CTIMER0	I3C0	INPUTMU...
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31 FLEXPWM0	FLEXPWM0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
30 QDC1	QDC1 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
29 QDC0	QDC0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
28 USB0	USB0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
27 LPUART4	LPUART4 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
26 LPUART3	LPUART3 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
25 LPUART2	LPUART2 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
24	LPUART1

Table continues on the next page...

Table continued from the previous page...

Field	Function
LPUART1	0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
23 LPUART0	LPUART0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
22 LPSPI1	LPSPI1 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
21 LPSPI0	LPSPI0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
20 LPI2C1	LPI2C1 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
19 LPI2C0	LPI2C0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
18 FLEXIO0	FLEXIO0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
17 AOI1	AOI1 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
16 —	reserved reserved
15 ERM0	ERM0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
14 EIM0	EIM0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled

Table continues on the next page...

Table continued from the previous page...

Field	Function
13 CRC0	CRC0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
12 AOI0	AOI0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
11 DMA0	DMA0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
10 SMARTDMA0	SMARTDMA0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
9 WWDT0	WWDT0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
8 UTICK0	UTICK0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
7 FREQME	FREQME 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
6 CTIMER4	CTIMER4 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
5 CTIMER3	CTIMER3 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
4 CTIMER2	CTIMER2 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
3 CTIMER1	CTIMER1

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
2 CTIMER0	CTIMER0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
1 I3C0	I3C0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
0 INPUTMUX0	INPUTMUX0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled

11.5.2.8 AHB Clock Control Set 0 (MRCC_GLB_CC0_SET)

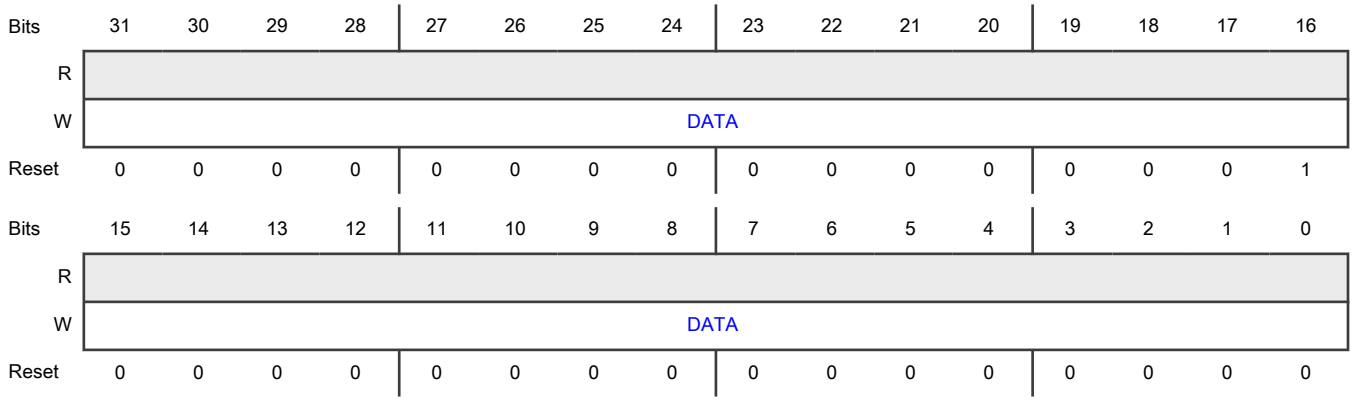
Offset

Register	Offset
MRCC_GLB_CC0_SET	44h

Function

Writing a 1 to a bit position in a write-only MRCC_GLB_CCn_SET register sets the corresponding position in MRCC_GLB_CCn.

Diagram



Fields

Field	Function
31-0 DATA	Data array value, refer to corresponding position in MRCC_GLB_CCn.

11.5.2.9 AHB Clock Control Clear 0 (MRCC_GLB_CC0_CLR)

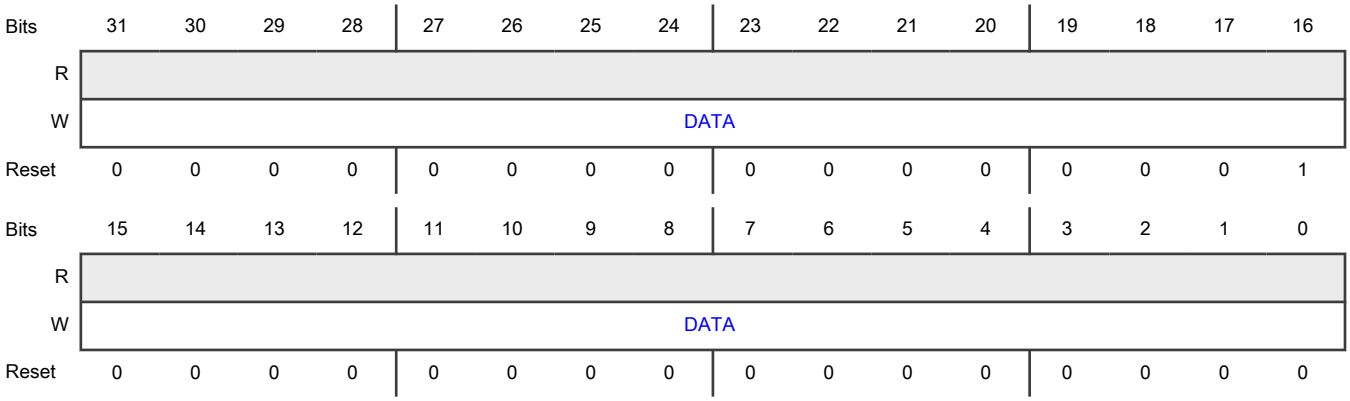
Offset

Register	Offset
MRCC_GLB_CC0_CLR	48h

Function

Writing a 1 to a bit position in a write-only MRCC_GLB_CCn_CLR register clears the corresponding position in MRCC_GLB_CCn.

Diagram



Fields

Field	Function
31-0 DATA	Data array value, refer to corresponding position in MRCC_GLB_CCn.

11.5.2.10 AHB Clock Control 1 (MRCC_GLB_CC1)

Offset

Register	Offset
MRCC_GLB_CC1	50h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved	Reserved	ADC3	ADC2	UDF0	TRNG0	SGI0	PKC0	TDET0	LPUART5	LPI2C3	LPI2C2	FLEXCAN1	FLEXCAN0	SLCD0	PORT4
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	PORT3	PORT2	PORT1	PORT0	OPAMP3	OPAMP2	OPAMP1	OPAMP0	DAC0	CMP2	CMP1	CMP0	ADC1	ADC0	OSTIMER0	FLEXPWM1
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31 —	Reserved
30 —	Reserved
29 ADC3	ADC3 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
28 ADC2	ADC2 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
27 UDF0	UDF0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
26 TRNG0	TRNG0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
25 SGI0	SGI0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
24 PKC0	PKC0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled

Table continues on the next page...

Table continued from the previous page...

Field	Function
23 TDET0	TDET0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
22 LPUART5	LPUART5 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
21 LPI2C3	LPI2C3 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
20 LPI2C2	LPI2C2 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
19 FLEXCAN1	FLEXCAN1 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
18 FLEXCAN0	FLEXCAN0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
17 SLCD0	SLCD0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
16 PORT4	PORT4 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
15 PORT3	PORT3 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
14 PORT2	PORT2 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
13 PORT1	PORT1

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
12 PORT0	PORT0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
11 OPAMP3	OPAMP3 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
10 OPAMP2	OPAMP2 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
9 OPAMP1	OPAMP1 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
8 OPAMP0	OPAMP0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
7 DAC0	DAC0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
6 CMP2	CMP2 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
5 CMP1	CMP1 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
4 CMP0	CMP0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
3 ADC1	ADC1 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled

Table continues on the next page...

Table continued from the previous page...

Field	Function
2 ADC0	ADC0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
1 OSTIMER0	OSTIMER0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
0 FLEXPWM1	FLEXPWM1 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled

11.5.2.11 AHB Clock Control Set n (MRCC_GLB_CC1_SET - MRCC_GLB_CC2_SET)

Offset

Register	Offset
MRCC_GLB_CC1_SET	54h
MRCC_GLB_CC2_SET	64h

Function

Writing a 1 to a bit position in a write-only MRCC_GLB_CCn_SET register sets the corresponding position in MRCC_GLB_CCn.

Diagram

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

Fields

Field	Function
31-0 DATA	Data array value, refer to corresponding position in MRCC_GLB_CCn.

11.5.2.12 AHB Clock Control Clear n (MRCC_GLB_CC1_CLR - MRCC_GLB_CC2_CLR)

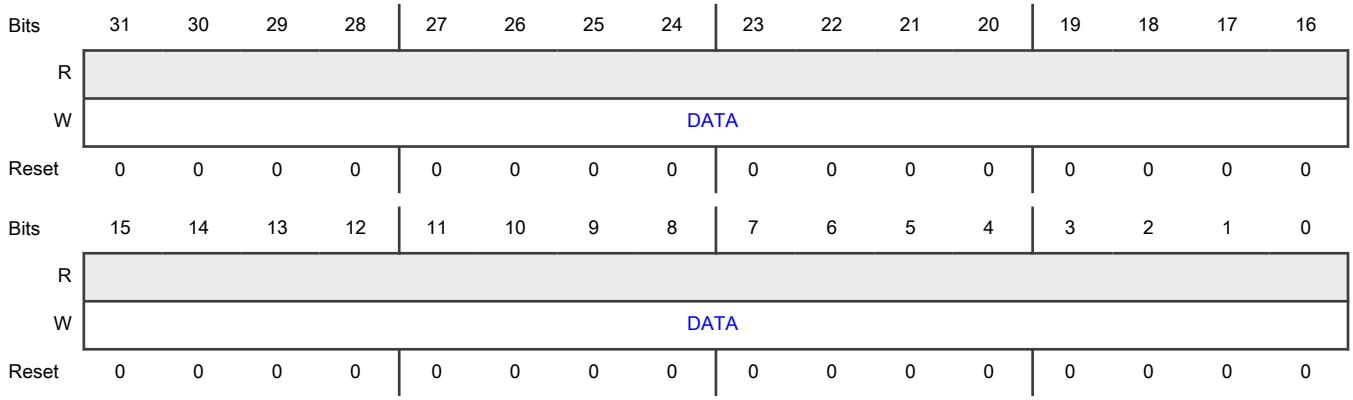
Offset

Register	Offset
MRCC_GLB_CC1_CLR	58h
MRCC_GLB_CC2_CLR	68h

Function

Writing a 1 to a bit position in a write-only MRCC_GLB_CCn_CLR register clears the corresponding position in MRCC_GLB_CCn.

Diagram



Fields

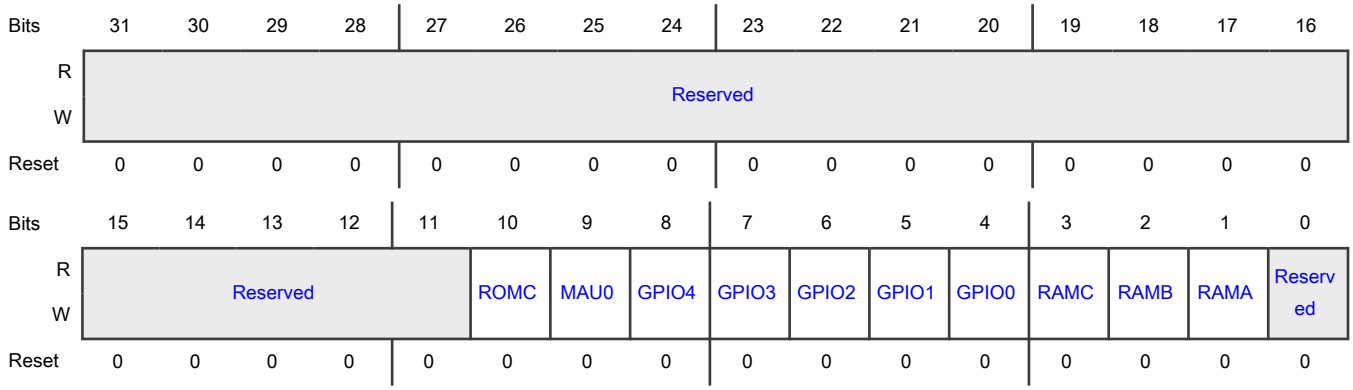
Field	Function
31-0 DATA	Data array value, refer to corresponding position in MRCC_GLB_CCn.

11.5.2.13 AHB Clock Control 2 (MRCC_GLB_CC2)

Offset

Register	Offset
MRCC_GLB_CC2	60h

Diagram



Fields

Field	Function
31-11 —	reserved reserved
10 ROMC	ROMC 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
9 MAU0	MAU0 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
8 GPIO4	GPIO4 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
7 GPIO3	GPIO3 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
6 GPIO2	GPIO2 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
5 GPIO1	GPIO1 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
4 GPIO0	GPIO0 0b - Peripheral clock is disabled

Table continues on the next page...

Table continued from the previous page...

Field	Function
	1b - Peripheral clock is enabled
3 RAMC	RAMC 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
2 RAMB	RAMB 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
1 RAMA	RAMA 0b - Peripheral clock is disabled 1b - Peripheral clock is enabled
0 —	Reserved

11.5.2.14 Control Automatic Clock Gating 0 (MRCC_GLB_ACC0)

Offset

Register	Offset
MRCC_GLB_ACC0	80h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	FLEXPWM0	QDC1	QDC0	USB0	LPUART4	LPUART3	LPUART2	LPUART1	LPUART0	LPSPI1	LPSPI0	LPI2C1	LPI2C0	FLEXIO0	AOI1	Reserved
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	ERM0	EIM0	CRC0	AOI0	DMA0	SMARTDM...	WWDT0	UTICK0	FREQME	CTIMER4	CTIMER3	CTIMER2	CTIMER1	CTIMER0	I3C0	INPUTMU...
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31 FLEXPWM0	FLEXPWM0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
30 QDC1	QDC1 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
29 QDC0	QDC0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
28 USB0	USB0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
27 LPUART4	LPUART4 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
26 LPUART3	LPUART3 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
25 LPUART2	LPUART2 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
24 LPUART1	LPUART1 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
23 LPUART0	LPUART0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
22 LPSPI1	LPSPI1 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
21 LPSPI0	LPSPI0

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
20 LPI2C1	LPI2C1 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
19 LPI2C0	LPI2C0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
18 FLEXIO0	FLEXIO0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
17 AOI1	AOI1 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
16 —	reserved reserved
15 ERM0	ERM0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
14 EIM0	EIM0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
13 CRC0	CRC0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
12 AOI0	AOI0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
11 DMA0	DMA0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled

Table continues on the next page...

Table continued from the previous page...

Field	Function
10 SMARTDMA0	SMARTDMA0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
9 WWDT0	WWDT0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
8 UTICK0	UTICK0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
7 FREQME	FREQME 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
6 CTIMER4	CTIMER4 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
5 CTIMER3	CTIMER3 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
4 CTIMER2	CTIMER2 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
3 CTIMER1	CTIMER1 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
2 CTIMER0	CTIMER0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
1 I3C0	I3C0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
0	INPUTMUX0

Table continues on the next page...

Table continued from the previous page...

Field	Function
INPUTMUX0	0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled

11.5.2.15 Control Automatic Clock Gating 1 (MRCC_GLB_ACC1)

Offset

Register	Offset
MRCC_GLB_ACC1	84h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved	Reserved	ADC3	ADC2	UDF0	TRNG0	SGI0	PKC0	Reserved	LPUART5	LPI2C3	LPI2C2	FLEXCAN1	FLEXCAN0	SLCD0	PORT4
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	PORT3	PORT2	PORT1	PORT0	OPAMP3	OPAMP2	OPAMP1	OPAMP0	DAC0	CMP2	CMP1	CMP0	ADC1	ADC0	OSTIMER0	FLEXPWM1
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31 —	reserved reserved
30 —	Reserved
29 ADC3	ADC3 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
28 ADC2	ADC2 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled

Table continues on the next page...

Table continued from the previous page...

Field	Function
27 UDF0	UDF0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
26 TRNG0	TRNG0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
25 SGI0	SGI0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
24 PKC0	PKC0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
23 —	reserved reserved
22 LPUART5	LPUART5 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
21 LPI2C3	LPI2C3 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
20 LPI2C2	LPI2C2 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
19 FLEXCAN1	FLEXCAN1 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
18 FLEXCAN0	FLEXCAN0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
17 SLCD0	SLCD0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled

Table continues on the next page...

Table continued from the previous page...

Field	Function
16 PORT4	PORT4 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
15 PORT3	PORT3 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
14 PORT2	PORT2 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
13 PORT1	PORT1 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
12 PORT0	PORT0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
11 OPAMP3	OPAMP3 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
10 OPAMP2	OPAMP2 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
9 OPAMP1	OPAMP1 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
8 OPAMP0	OPAMP0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
7 DAC0	DAC0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
6 CMP2	CMP2

Table continues on the next page...

Table continued from the previous page...

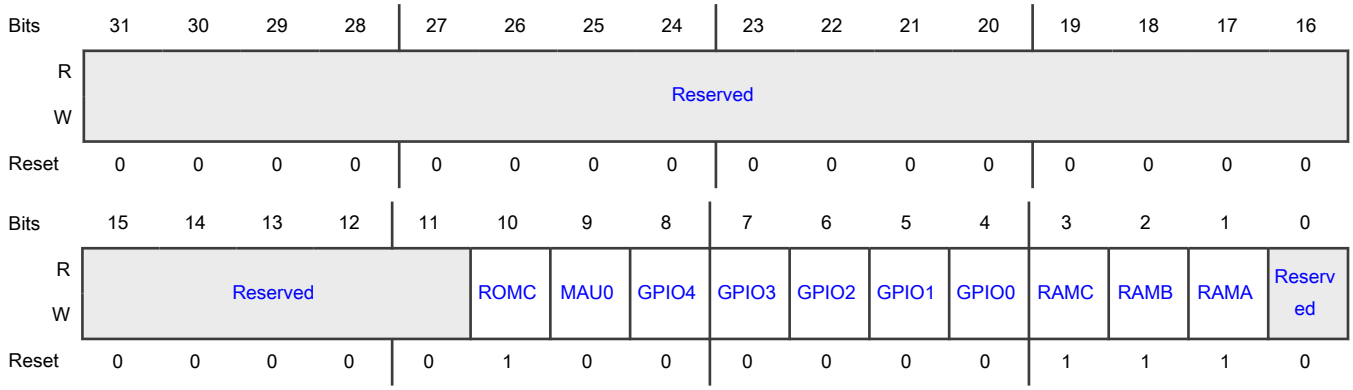
Field	Function
	0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
5 CMP1	CMP1 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
4 CMP0	CMP0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
3 ADC1	ADC1 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
2 ADC0	ADC0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
1 OSTIMER0	OSTIMER0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
0 FLEXPWM1	FLEXPWM1 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled

11.5.2.16 Control Automatic Clock Gating 2 (MRCC_GLB_ACC2)

Offset

Register	Offset
MRCC_GLB_ACC2	88h

Diagram



Fields

Field	Function
31-11 —	reserved reserved
10 ROMC	ROMC 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
9 MAU0	MAU0 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
8 GPIO4	GPIO4 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
7 GPIO3	GPIO3 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
6 GPIO2	GPIO2 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
5 GPIO1	GPIO1 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
4 GPIO0	GPIO0 0b - Automatic clock gating is disabled

Table continues on the next page...

Table continued from the previous page...

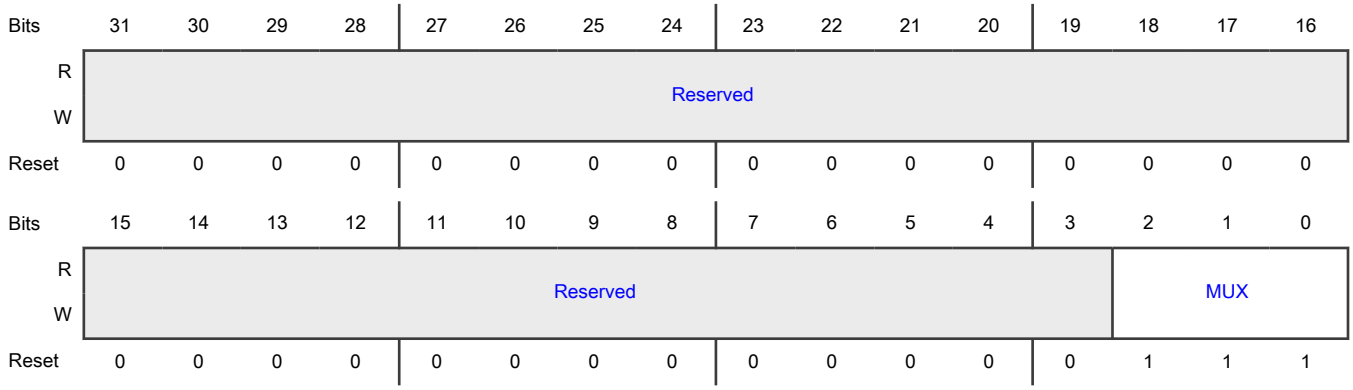
Field	Function
	1b - Automatic clock gating is enabled
3 RAMC	RAMC 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
2 RAMB	RAMB 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
1 RAMA	RAMA 0b - Automatic clock gating is disabled 1b - Automatic clock gating is enabled
0 —	reserved reserved

11.5.2.17 I3C0_FCLK clock selection control (MRCC_I3C0_FCLK_CLKSEL)

Offset

Register	Offset
MRCC_I3C0_FCLK_CLKSEL	A0h

Diagram



Fields

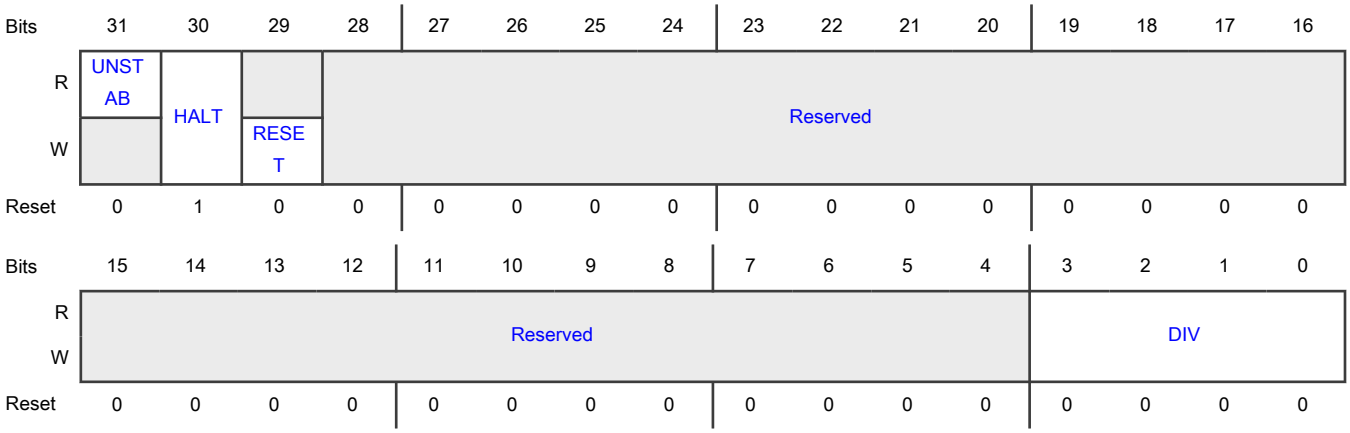
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 010b - FRO_HF_DIV 011b - CLK_IN 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.18 I3C0_FCLK clock divider control (MRCC_I3C0_FCLK_CLKDIV)

Offset

Register	Offset
MRCC_I3C0_FCLK_CLKDIV	A4h

Diagram



Fields

Field	Function
31	Divider status flag

Table continues on the next page...

Table continued from the previous page...

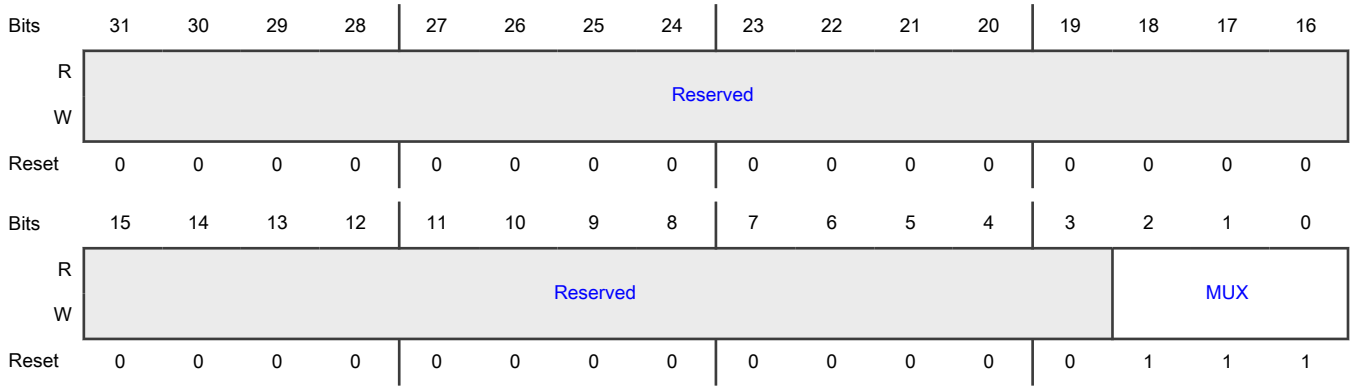
Field	Function
UNSTAB	Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.19 CTIMER0 clock selection control (MRCC_CTIMER0_CLKSEL)

Offset

Register	Offset
MRCC_CTIMER0_CLKSEL	A8h

Diagram



Fields

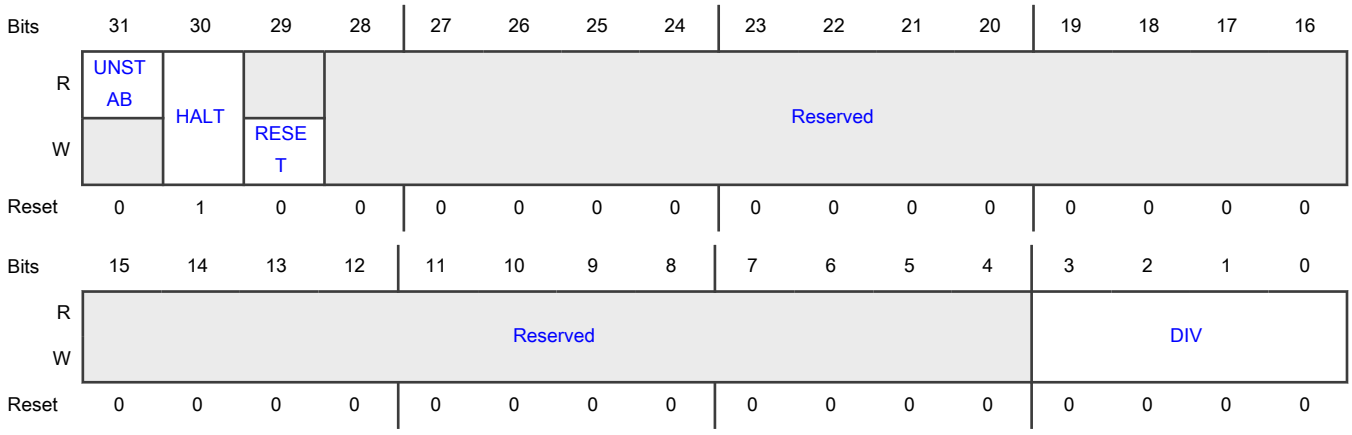
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 001b - FRO_HF_GATED 011b - CLK_IN 100b - CLK_16K 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.20 CTIMER0 clock divider control (MRCC_CTIMER0_CLKDIV)

Offset

Register	Offset
MRCC_CTIMER0_CLKDIV	ACh

Diagram



Fields

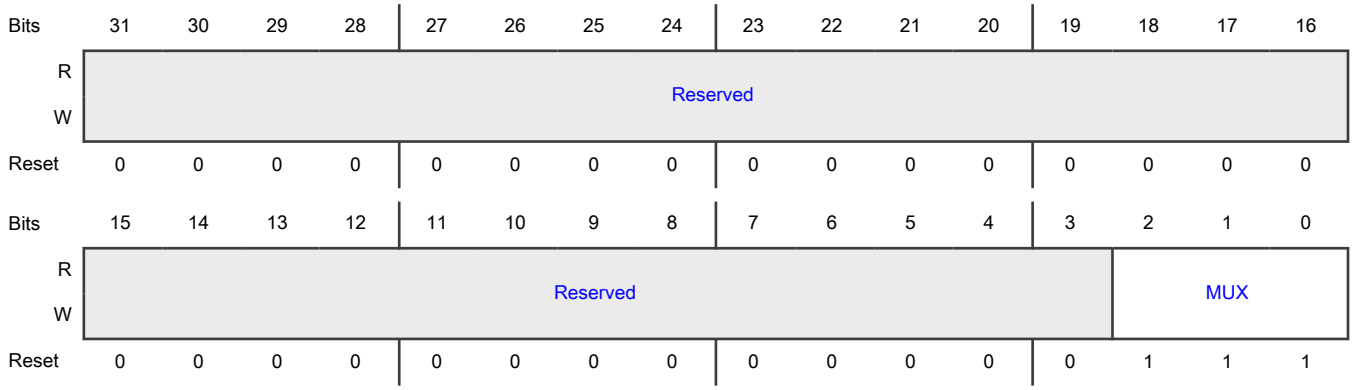
Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.21 CTIMER1 clock selection control (MRCC_CTIMER1_CLKSEL)

Offset

Register	Offset
MRCC_CTIMER1_CLKSEL	B0h

Diagram



Fields

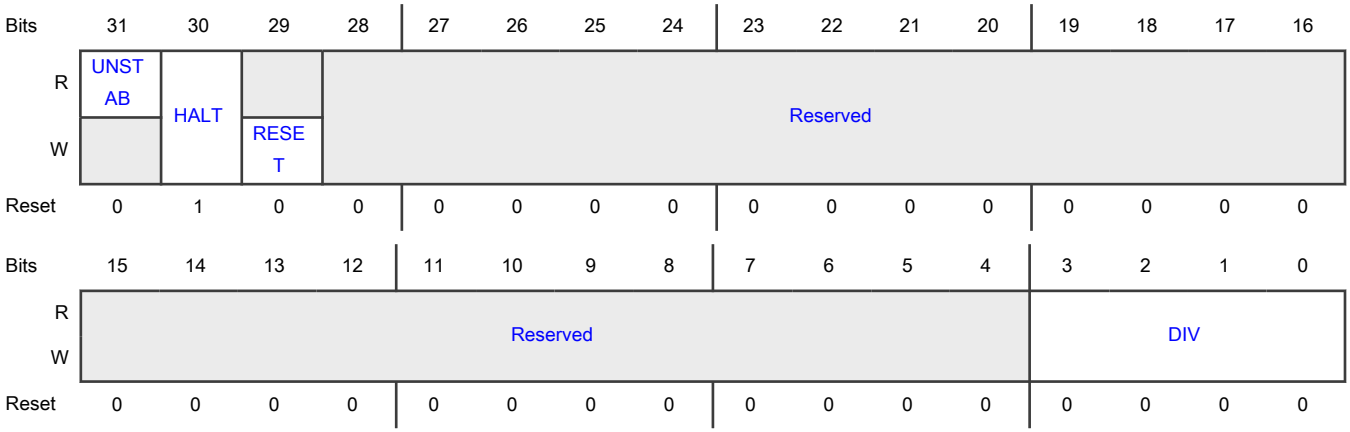
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 001b - FRO_HF_GATED 011b - CLK_IN 100b - CLK_16K 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.22 CTIMER1 clock divider control (MRCC_CTIMER1_CLKDIV)

Offset

Register	Offset
MRCC_CTIMER1_CLKDIV	B4h

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.23 CTIMER2 clock selection control (MRCC_CTIMER2_CLKSEL)

Offset

Register	Offset
MRCC_CTIMER2_CLKSEL	B8h

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													MUX		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

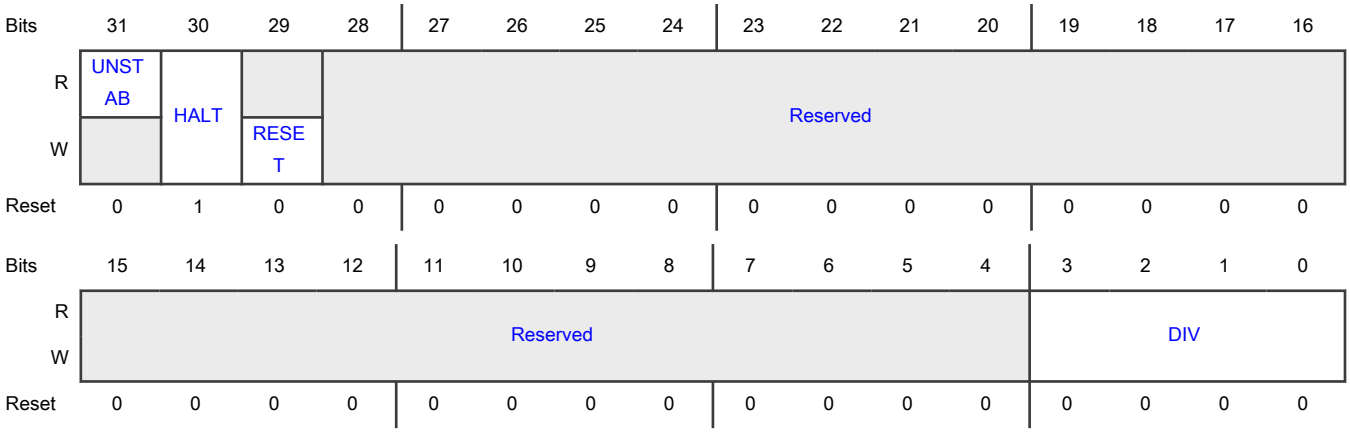
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 001b - FRO_HF_GATED 011b - CLK_IN 100b - CLK_16K 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.24 CTIMER2 clock divider control (MRCC_CTIMER2_CLKDIV)

Offset

Register	Offset
MRCC_CTIMER2_CLKDIV	BCh

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.25 CTIMER3 clock selection control (MRCC_CTIMER3_CLKSEL)

Offset

Register	Offset
MRCC_CTIMER3_CLKSEL	C0h

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												MUX			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 001b - FRO_HF_GATED 011b - CLK_IN 100b - CLK_16K

Table continues on the next page...

Table continued from the previous page...

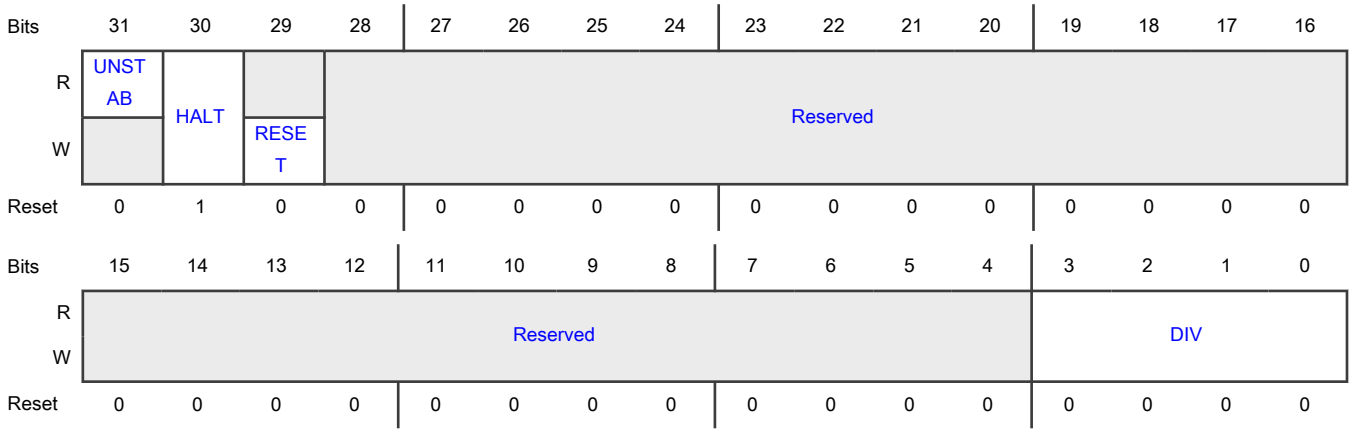
Field	Function
	101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.26 CTIMER3 clock divider control (MRCC_CTIMER3_CLKDIV)

Offset

Register	Offset
MRCC_CTIMER3_CLKDIV	C4h

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped

Table continues on the next page...

Table continued from the previous page...

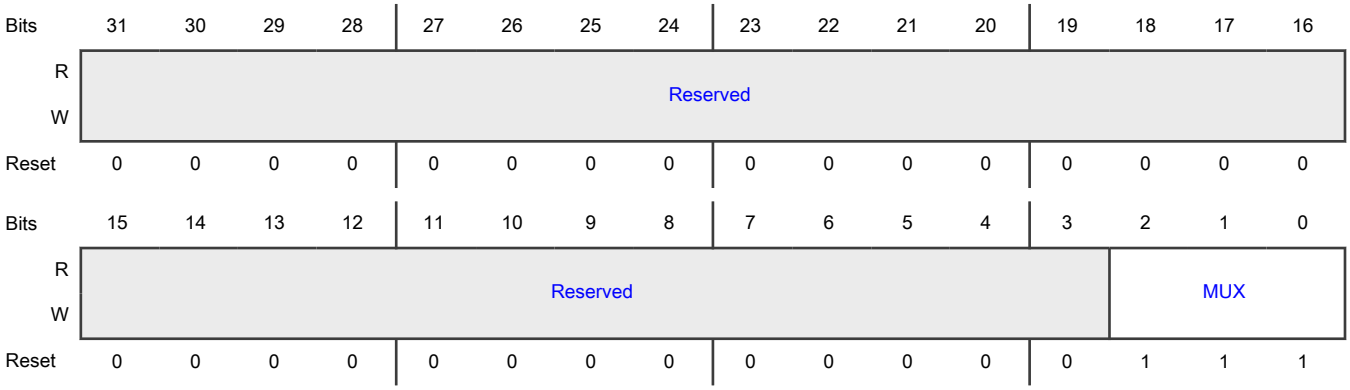
Field	Function
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.27 CTIMER4 clock selection control (MRCC_CTIMER4_CLKSEL)

Offset

Register	Offset
MRCC_CTIMER4_CLKSEL	C8h

Diagram



Fields

Field	Function
31-3 —	reserved reserved
2-0	Functional Clock Mux Select

Table continues on the next page...

Table continued from the previous page...

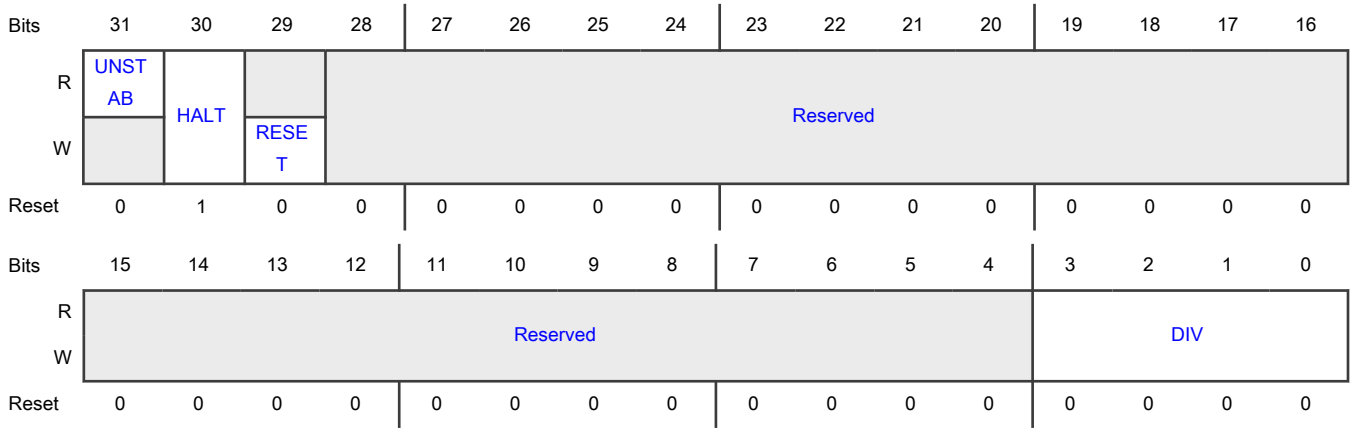
Field	Function
MUX	Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 001b - FRO_HF_GATED 011b - CLK_IN 100b - CLK_16K 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.28 CTIMER4 clock divider control (MRCC_CTIMER4_CLKDIV)

Offset

Register	Offset
MRCC_CTIMER4_CLKDIV	CCh

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable

Table continues on the next page...

Table continued from the previous page...

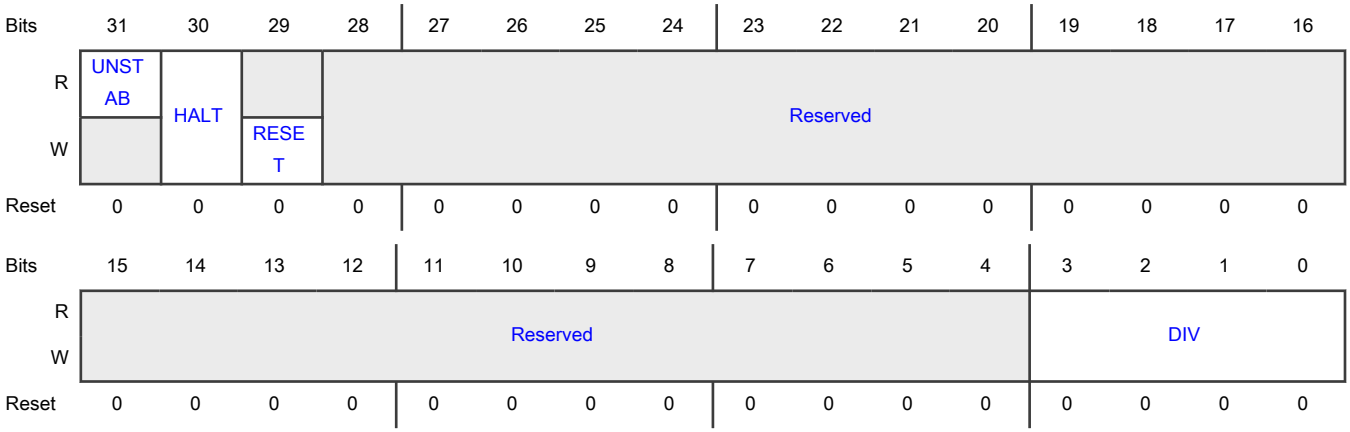
Field	Function
	1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.29 WWDT0 clock divider control (MRCC_WWDT0_CLKDIV)

Offset

Register	Offset
MRCC_WWDT0_CLKDIV	D4h

Diagram



Fields

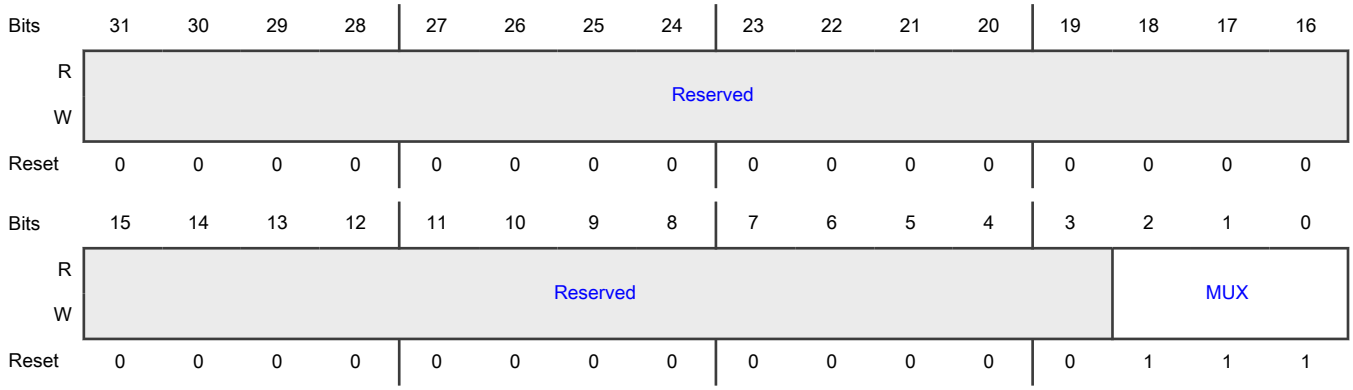
Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.30 FLEXIO0 clock selection control (MRCC_FLEXIO0_CLKSEL)

Offset

Register	Offset
MRCC_FLEXIO0_CLKSEL	D8h

Diagram



Fields

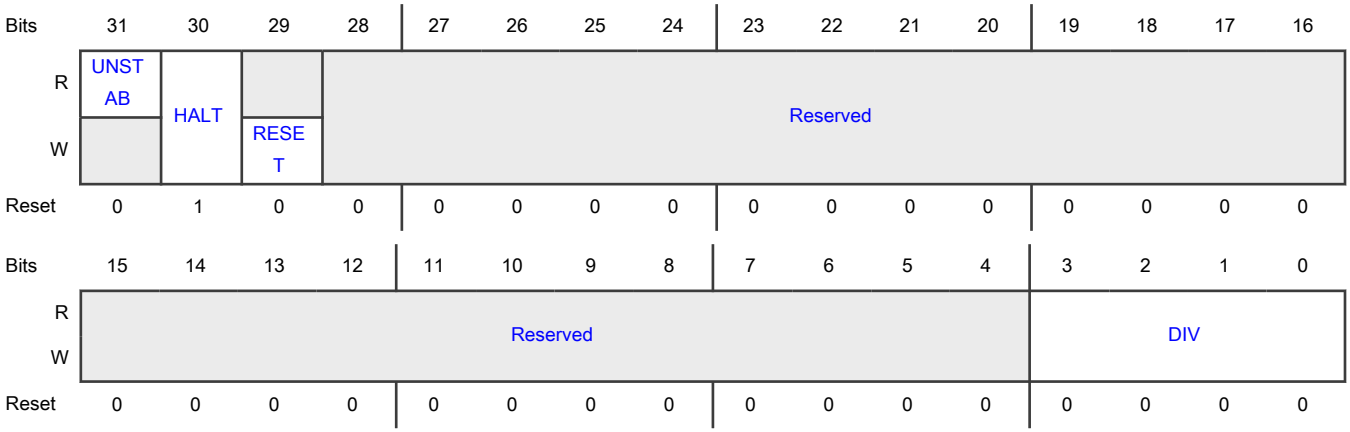
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 001b - FRO_HF_GATED 011b - CLK_IN 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.31 FLEXIO0 clock divider control (MRCC_FLEXIO0_CLKDIV)

Offset

Register	Offset
MRCC_FLEXIO0_CLKDIV	DCh

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.32 LPI2C0 clock selection control (MRCC_LPI2C0_CLKSEL)

Offset

Register	Offset
MRCC_LPI2C0_CLKSEL	E0h

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	Reserved												MUX			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

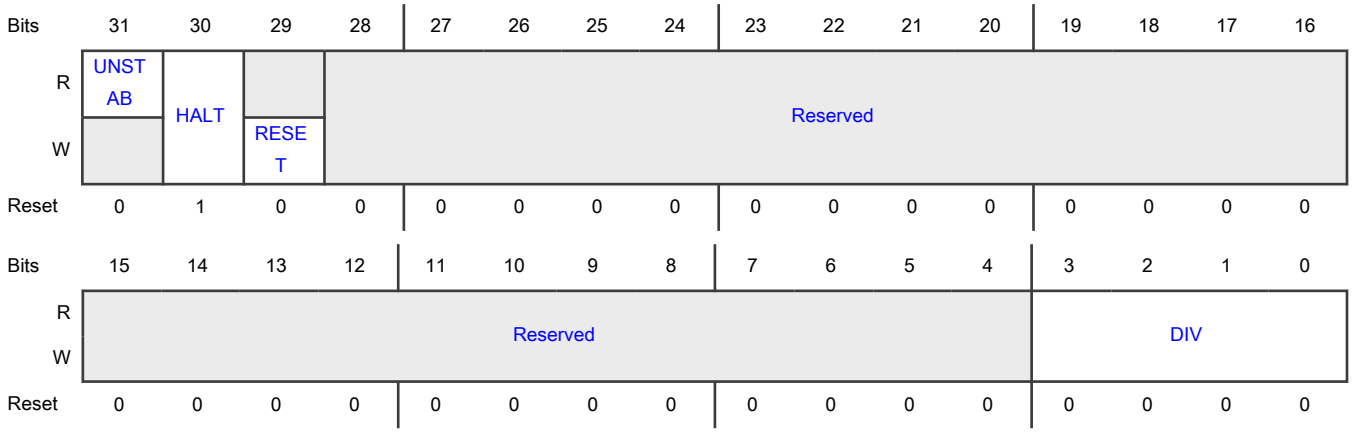
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 010b - FRO_HF_DIV 011b - CLK_IN 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.33 LPI2C0 clock divider control (MRCC_LPI2C0_CLKDIV)

Offset

Register	Offset
MRCC_LPI2C0_CLKDIV	E4h

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.34 LPI2C1 clock selection control (MRCC_LPI2C1_CLKSEL)

Offset

Register	Offset
MRCC_LPI2C1_CLKSEL	E8h

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	Reserved												MUX			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

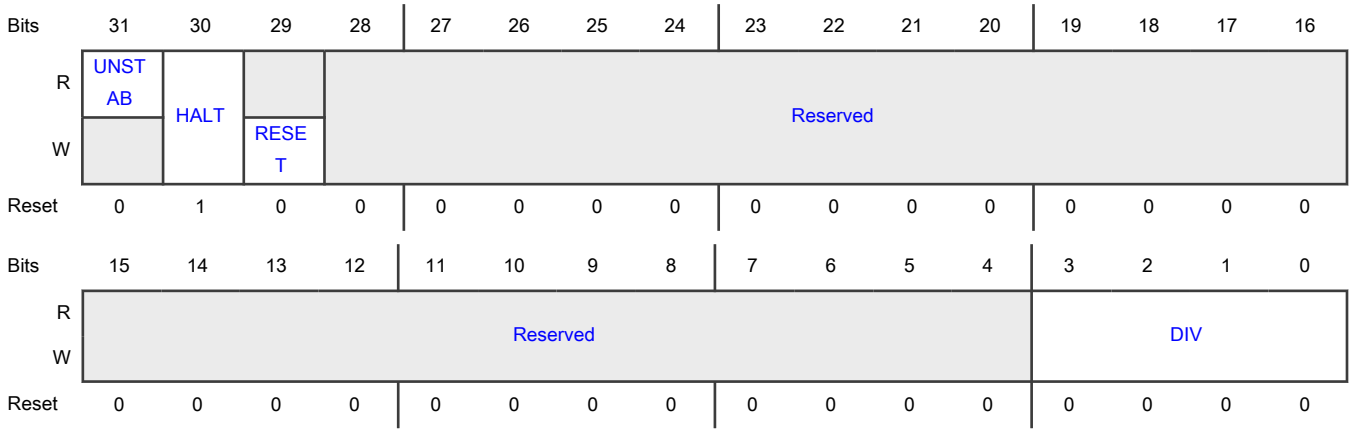
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 010b - FRO_HF_DIV 011b - CLK_IN 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.35 LPI2C1 clock divider control (MRCC_LPI2C1_CLKDIV)

Offset

Register	Offset
MRCC_LPI2C1_CLKDIV	ECh

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.36 LPSPI0 clock selection control (MRCC_LPSPI0_CLKSEL)

Offset

Register	Offset
MRCC_LPSPI0_CLKSEL	F0h

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	Reserved												MUX			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

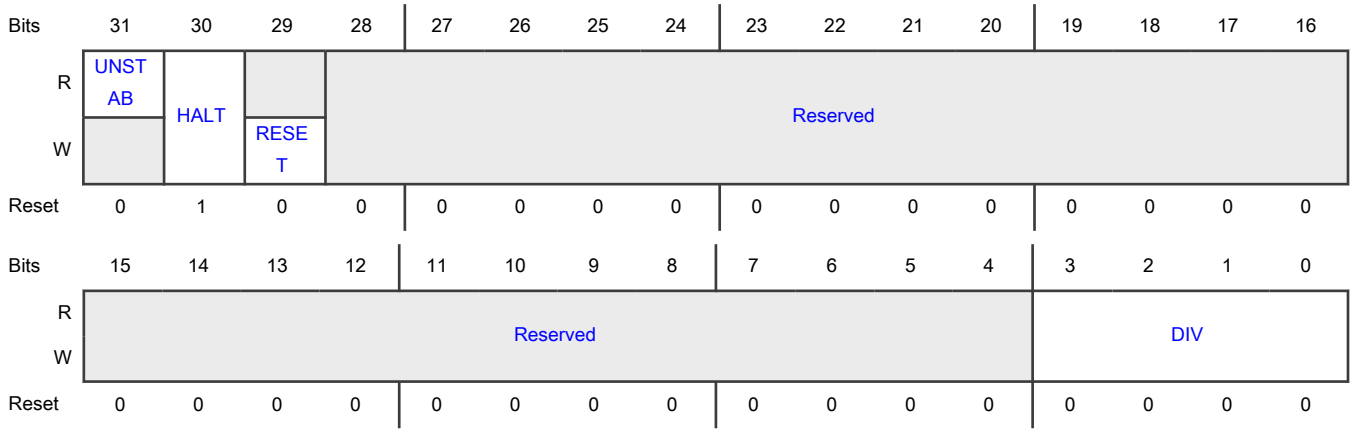
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 010b - FRO_HF_DIV 011b - CLK_IN 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.37 LPSPI0 clock divider control (MRCC_LPSPI0_CLKDIV)

Offset

Register	Offset
MRCC_LPSPI0_CLKDIV	F4h

Diagram



Fields

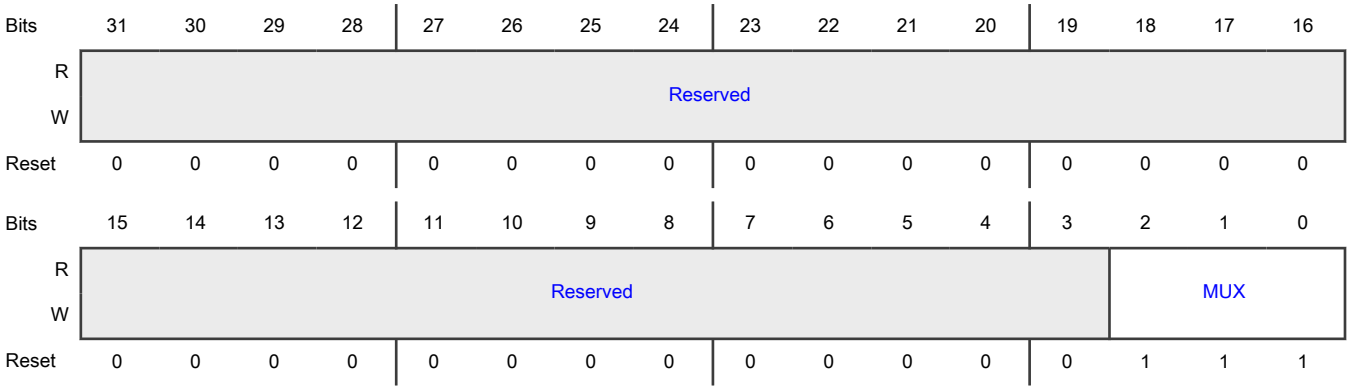
Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.38 LPSPI1 clock selection control (MRCC_LPSPI1_CLKSEL)

Offset

Register	Offset
MRCC_LPSPI1_CLKSEL	F8h

Diagram



Fields

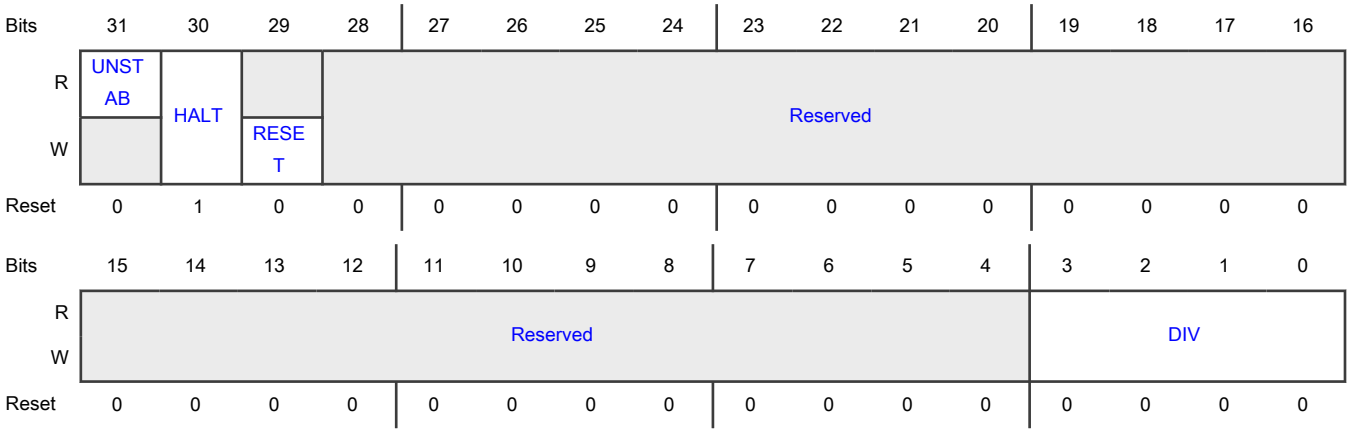
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 010b - FRO_HF_DIV 011b - CLK_IN 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.39 LPSPI1 clock divider control (MRCC_LPSPI1_CLKDIV)

Offset

Register	Offset
MRCC_LPSPI1_CLKDIV	FCh

Diagram



Fields

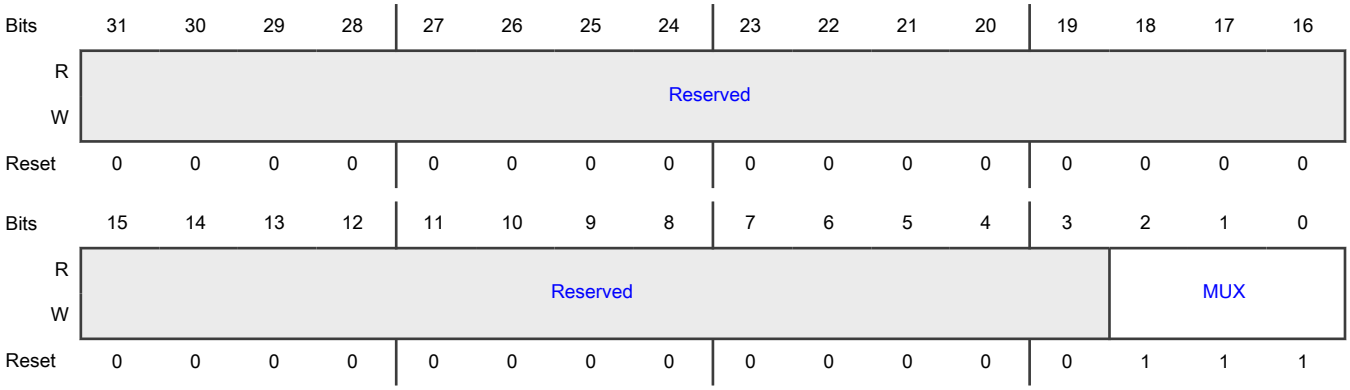
Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.40 LPUART0 clock selection control (MRCC_LPUART0_CLKSEL)

Offset

Register	Offset
MRCC_LPUART0_CLKSEL	100h

Diagram



Fields

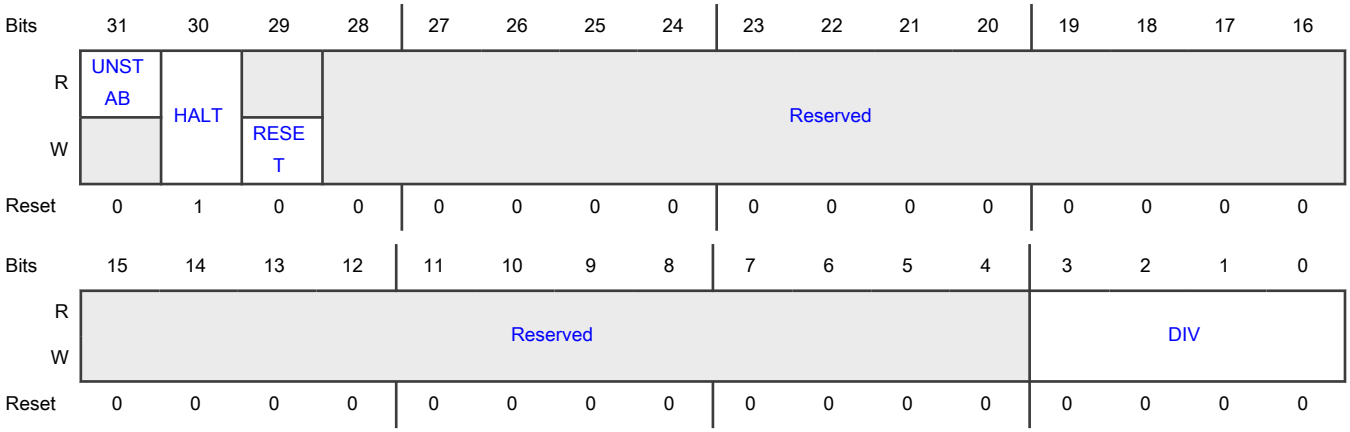
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 010b - FRO_HF_DIV 011b - CLK_IN 100b - CLK_16K 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.41 LPUART0 clock divider control (MRCC_LPUART0_CLKDIV)

Offset

Register	Offset
MRCC_LPUART0_CLKDIV	104h

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.42 LPUART1 clock selection control (MRCC_LPUART1_CLKSEL)

Offset

Register	Offset
MRCC_LPUART1_CLKSEL	108h

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													MUX		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 010b - FRO_HF_DIV 011b - CLK_IN 100b - CLK_16K

Table continues on the next page...

Table continued from the previous page...

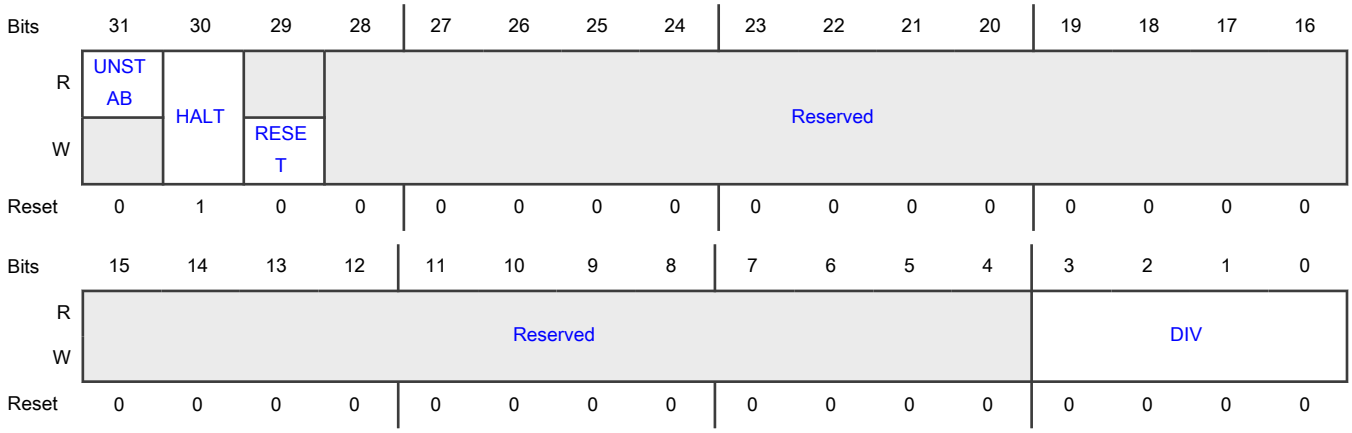
Field	Function
	101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.43 LPUART1 clock divider control (MRCC_LPUART1_CLKDIV)

Offset

Register	Offset
MRCC_LPUART1_CLKDIV	10Ch

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped

Table continues on the next page...

Table continued from the previous page...

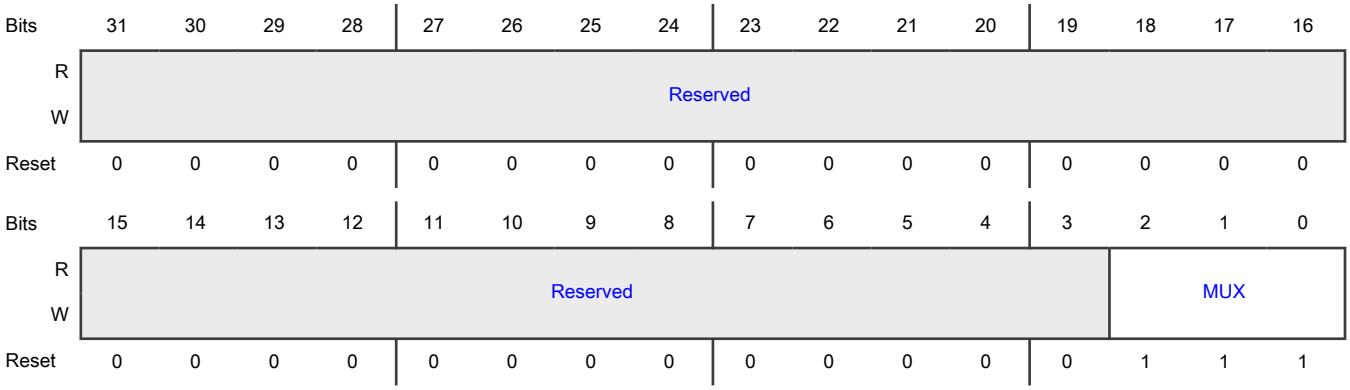
Field	Function
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.44 LPUART2 clock selection control (MRCC_LPUART2_CLKSEL)

Offset

Register	Offset
MRCC_LPUART2_CLKSEL	110h

Diagram



Fields

Field	Function
31-3 —	reserved reserved
2-0	Functional Clock Mux Select

Table continues on the next page...

Table continued from the previous page...

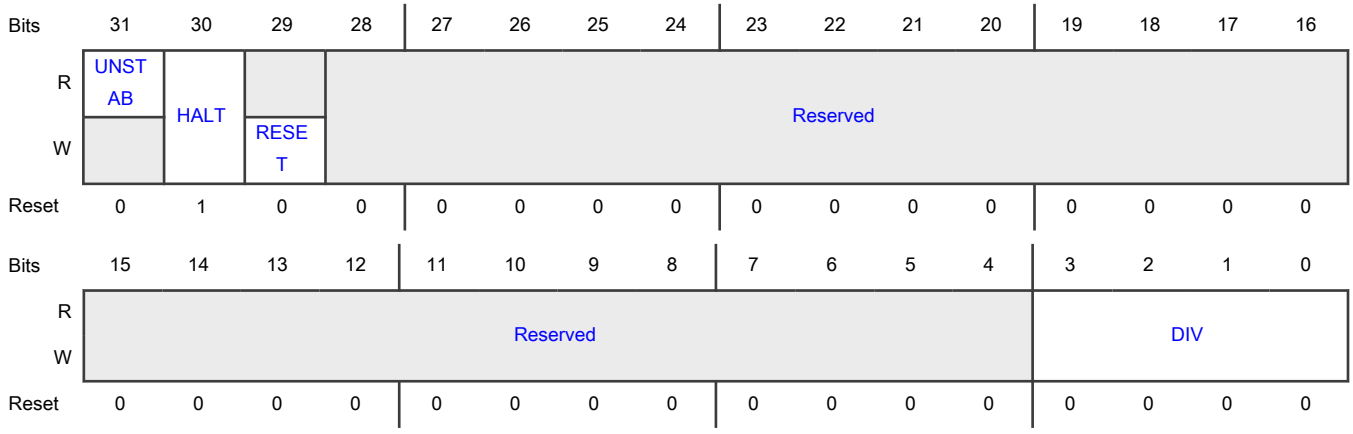
Field	Function
MUX	Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 010b - FRO_HF_DIV 011b - CLK_IN 100b - CLK_16K 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.45 LPUART2 clock divider control (MRCC_LPUART2_CLKDIV)

Offset

Register	Offset
MRCC_LPUART2_CLKDIV	114h

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable

Table continues on the next page...

Table continued from the previous page...

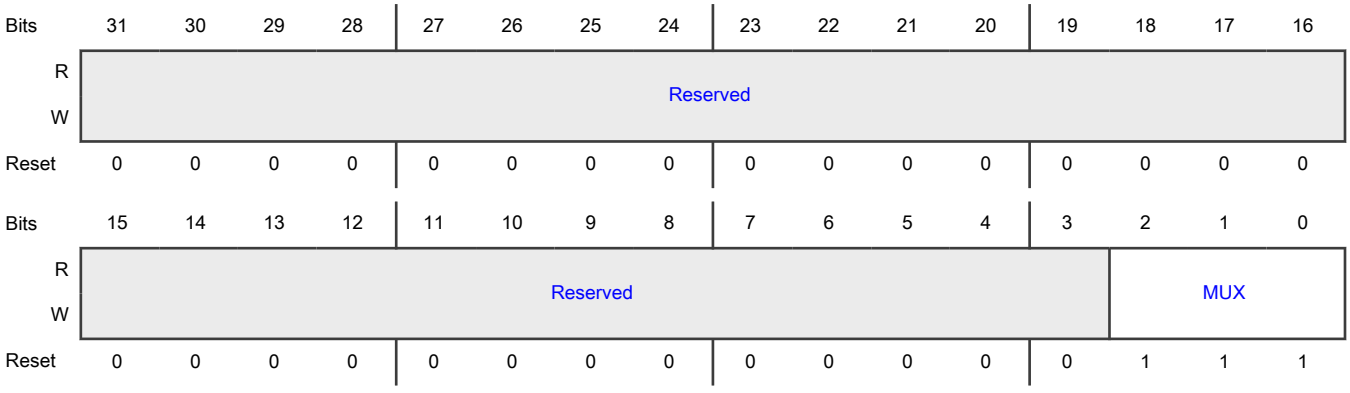
Field	Function
	1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.46 LPUART3 clock selection control (MRCC_LPUART3_CLKSEL)

Offset

Register	Offset
MRCC_LPUART3_CLKSEL	118h

Diagram



Fields

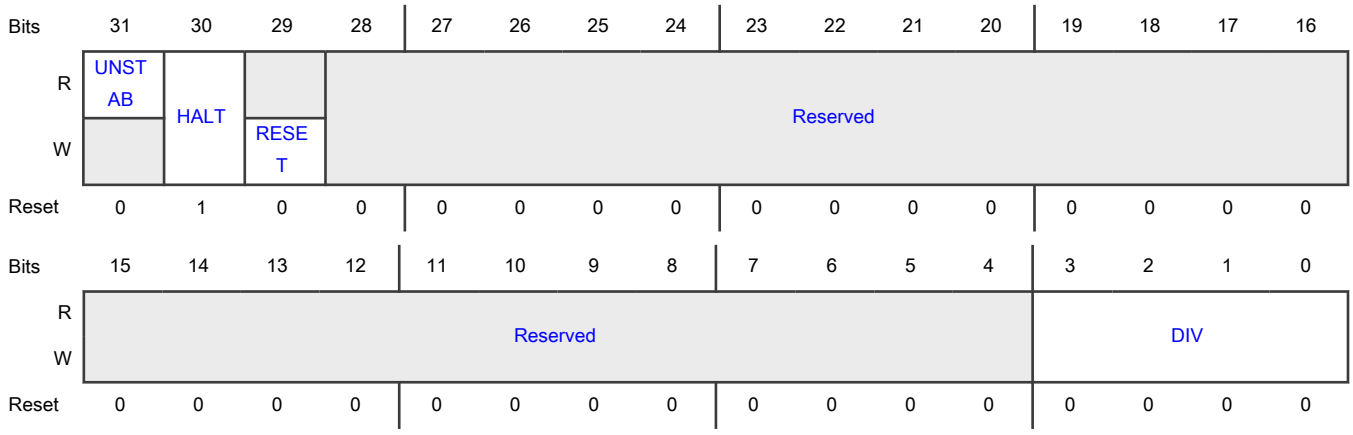
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 010b - FRO_HF_DIV 011b - CLK_IN 100b - CLK_16K 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.47 LPUART3 clock divider control (MRCC_LPUART3_CLKDIV)

Offset

Register	Offset
MRCC_LPUART3_CLKDIV	11Ch

Diagram



Fields

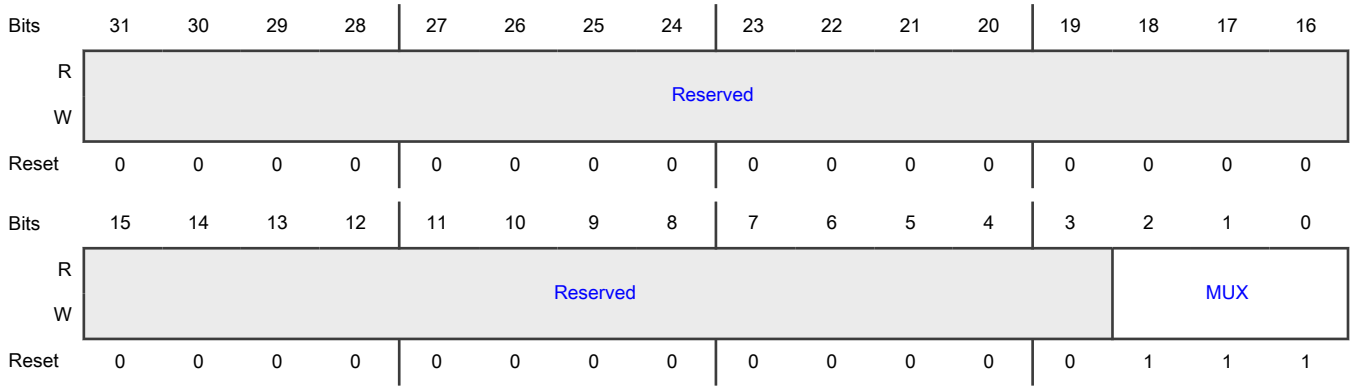
Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.48 LPUART4 clock selection control (MRCC_LPUART4_CLKSEL)

Offset

Register	Offset
MRCC_LPUART4_CLKSEL	120h

Diagram



Fields

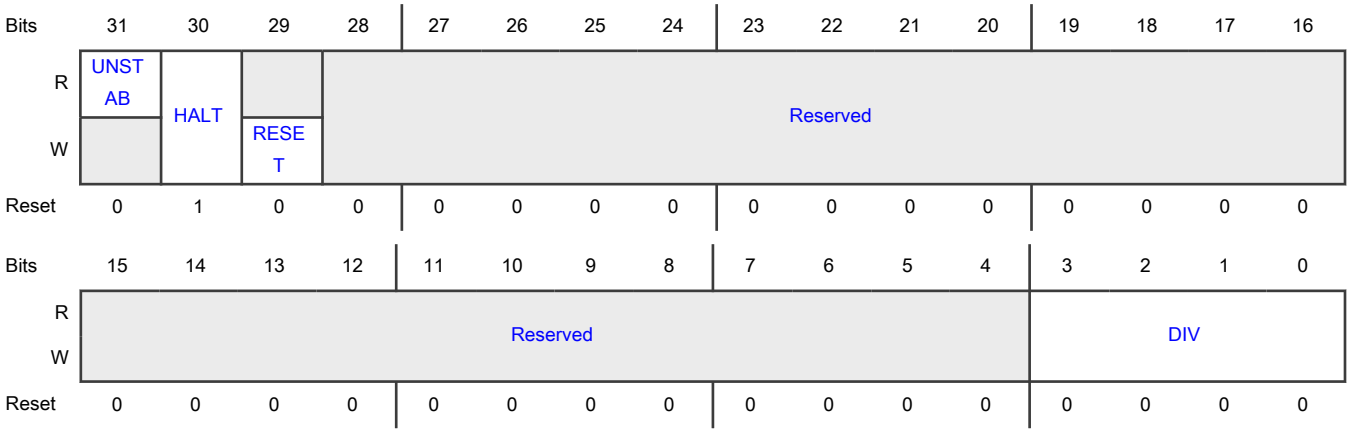
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 010b - FRO_HF_DIV 011b - CLK_IN 100b - CLK_16K 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.49 LPUART4 clock divider control (MRCC_LPUART4_CLKDIV)

Offset

Register	Offset
MRCC_LPUART4_CLKDIV	124h

Diagram



Fields

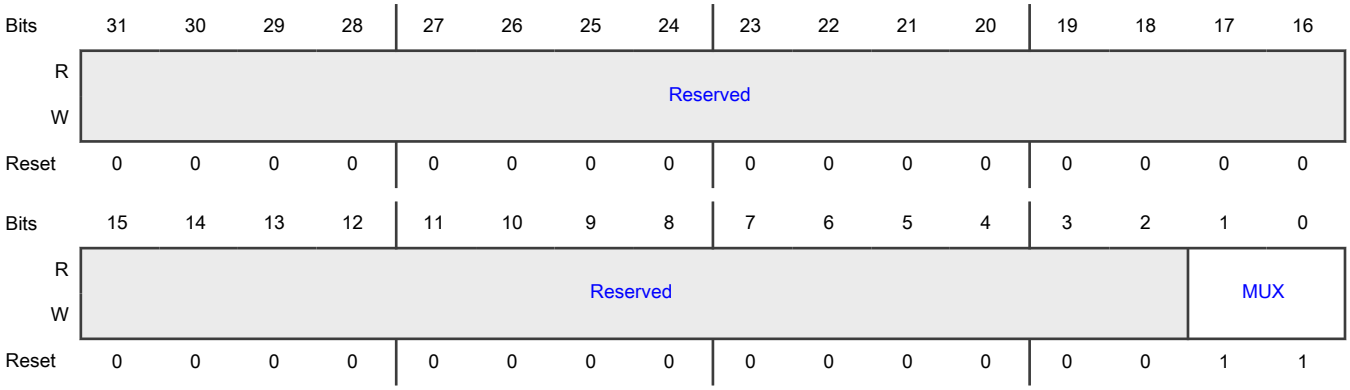
Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.50 USB0 clock selection control (MRCC_USB0_CLKSEL)

Offset

Register	Offset
MRCC_USB0_CLKSEL	128h

Diagram



Fields

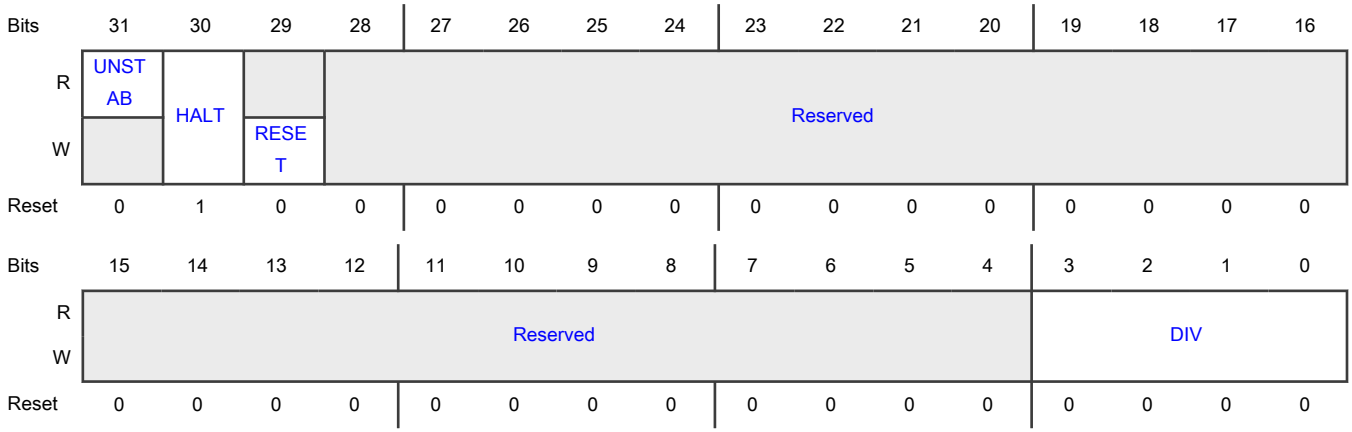
Field	Function
31-2 —	reserved reserved
1-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 00b - PLL1_CLK 01b - CLK_48M 10b - CLK_IN 11b - Reserved(NO Clock)

11.5.2.51 USB0 clock divider control (MRCC_USB0_CLKDIV)

Offset

Register	Offset
MRCC_USB0_CLKDIV	12Ch

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.52 LPTMR0 clock selection control (MRCC_LPTMR0_CLKSEL)

Offset

Register	Offset
MRCC_LPTMR0_CLKSEL	130h

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													MUX		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

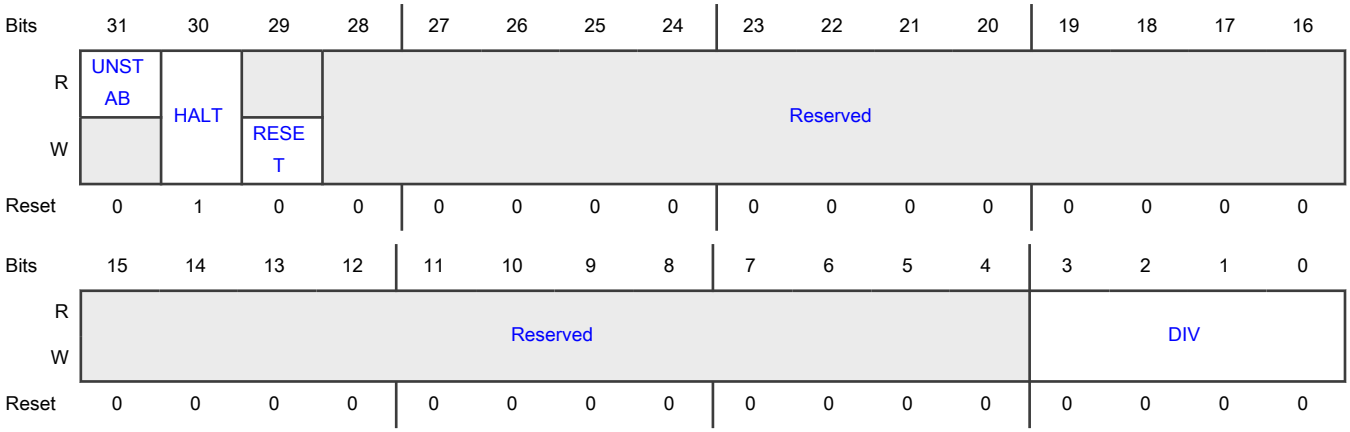
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 010b - FRO_HF_DIV 011b - CLK_IN 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.53 LPTMR0 clock divider control (MRCC_LPTMR0_CLKDIV)

Offset

Register	Offset
MRCC_LPTMR0_CLKDIV	134h

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.54 OSTIMER0 clock selection control (MRCC_OSTIMER0_CLKSEL)

Offset

Register	Offset
MRCC_OSTIMER0_CLKSEL	138h

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														MUX	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Fields

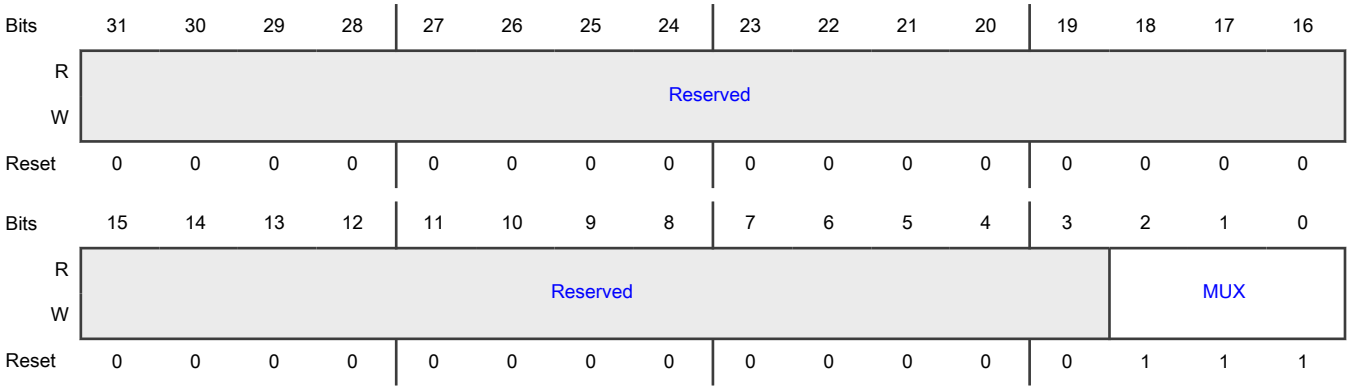
Field	Function
31-2 —	reserved reserved
1-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 00b - CLK_16K 10b - CLK_1M 11b - Reserved2(NO Clock)

11.5.2.55 ADCx clock selection control (MRCC_ADC_CLKSEL)

Offset

Register	Offset
MRCC_ADC_CLKSEL	140h

Diagram



Fields

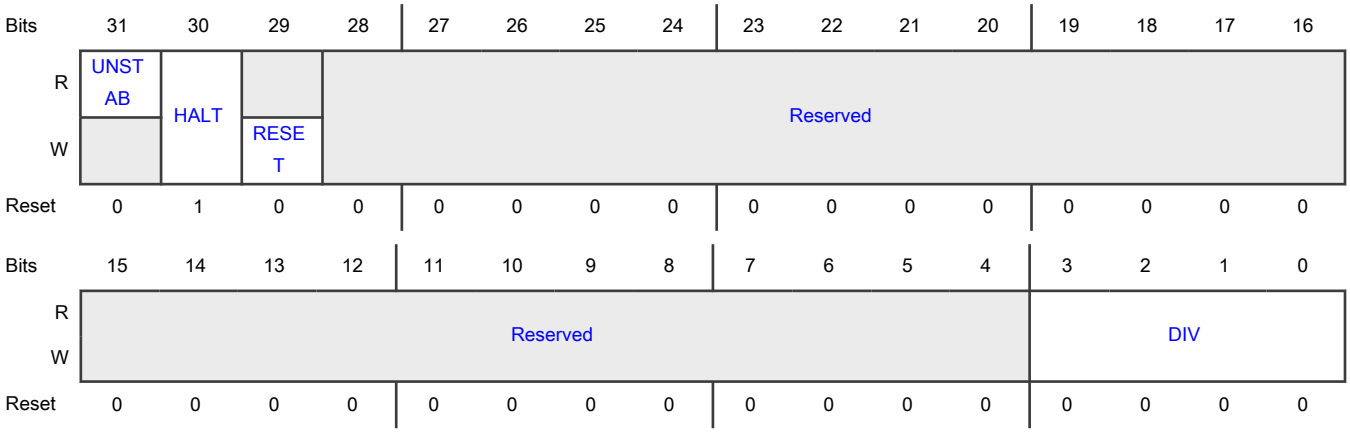
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 001b - FRO_HF_GATED 011b - CLK_IN 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.56 ADCx clock divider control (MRCC_ADC_CLKDIV)

Offset

Register	Offset
MRCC_ADC_CLKDIV	144h

Diagram



Fields

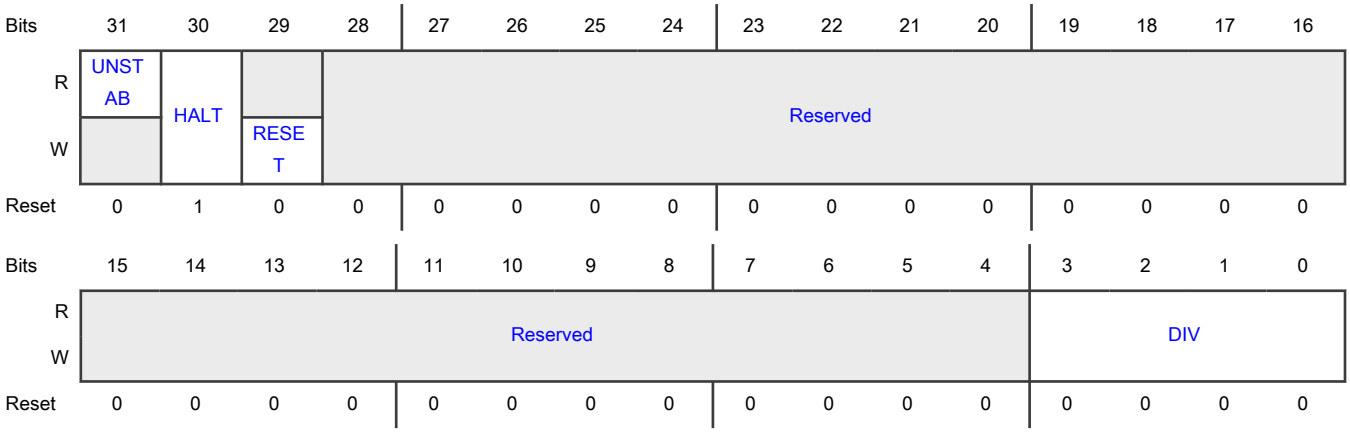
Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.57 CMP0_FUNC clock divider control (MRCC_CMP0_FUNC_CLKDIV)

Offset

Register	Offset
MRCC_CMP0_FUNC_CLKDIV	14Ch

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.58 CMP0_RR clock selection control (MRCC_CMP0_RR_CLKSEL)

Offset

Register	Offset
MRCC_CMP0_RR_CLKSEL	150h

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												MUX			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

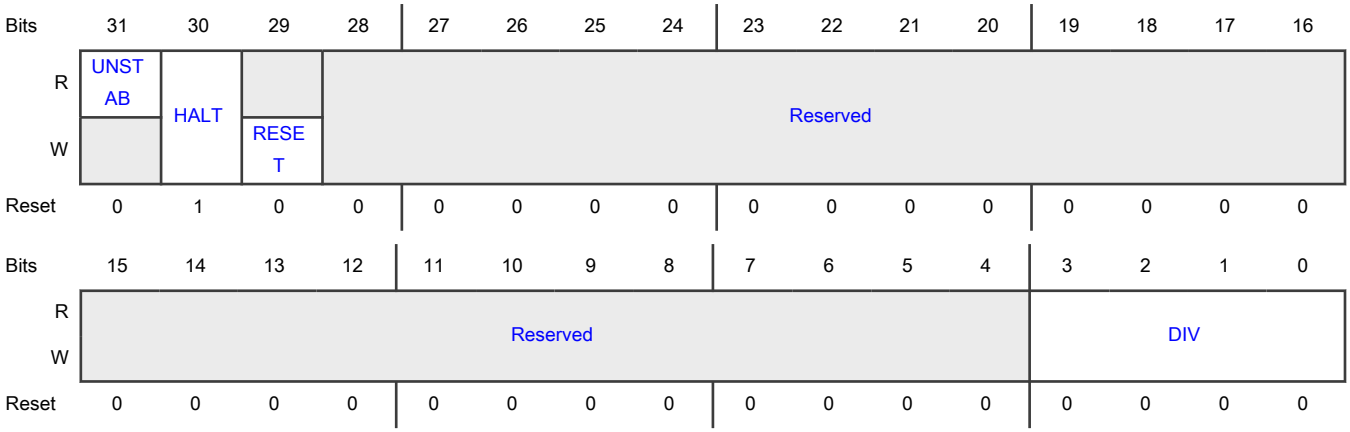
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 010b - FRO_HF_DIV 011b - CLK_IN 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.59 CMP0_RR clock divider control (MRCC_CMP0_RR_CLKDIV)

Offset

Register	Offset
MRCC_CMP0_RR_CLKDIV	154h

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved

Table continues on the next page...

Table continued from the previous page...

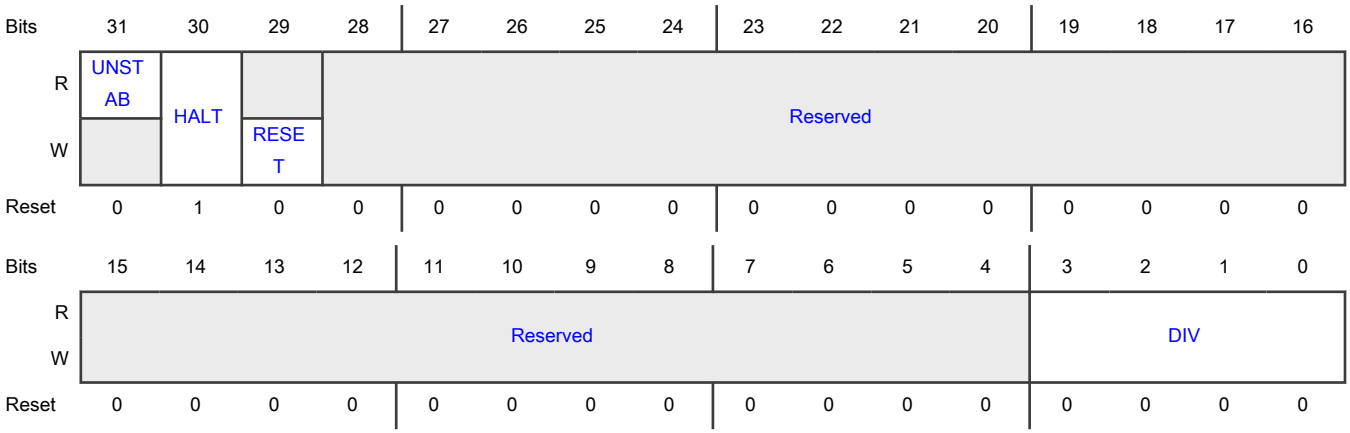
Field	Function
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.60 CMP1_FUNC clock divider control (MRCC_CMP1_FUNC_CLKDIV)

Offset

Register	Offset
MRCC_CMP1_FUNC_CLKDIV	15Ch

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped

Table continues on the next page...

Table continued from the previous page...

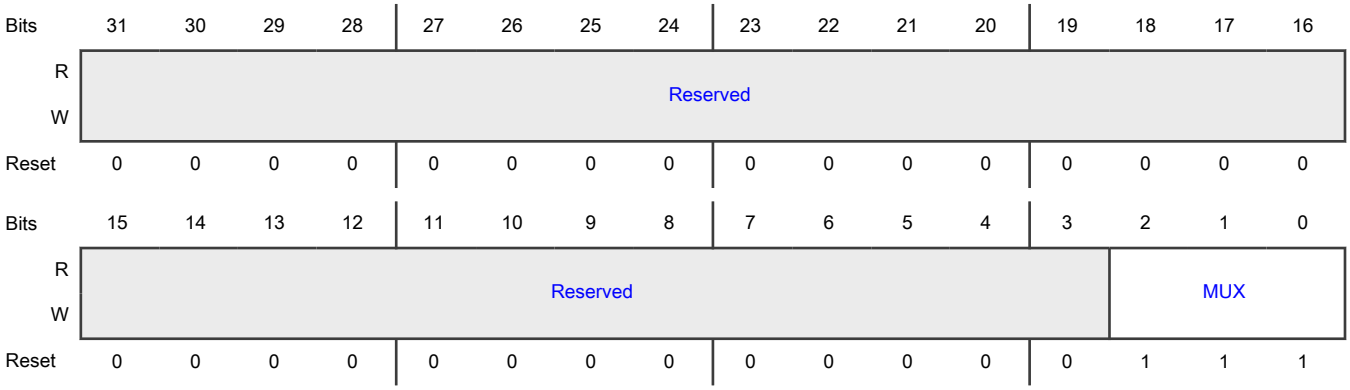
Field	Function
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.61 CMP1_RR clock selection control (MRCC_CMP1_RR_CLKSEL)

Offset

Register	Offset
MRCC_CMP1_RR_CLKSEL	160h

Diagram



Fields

Field	Function
31-3 —	reserved reserved
2-0	Functional Clock Mux Select

Table continues on the next page...

Table continued from the previous page...

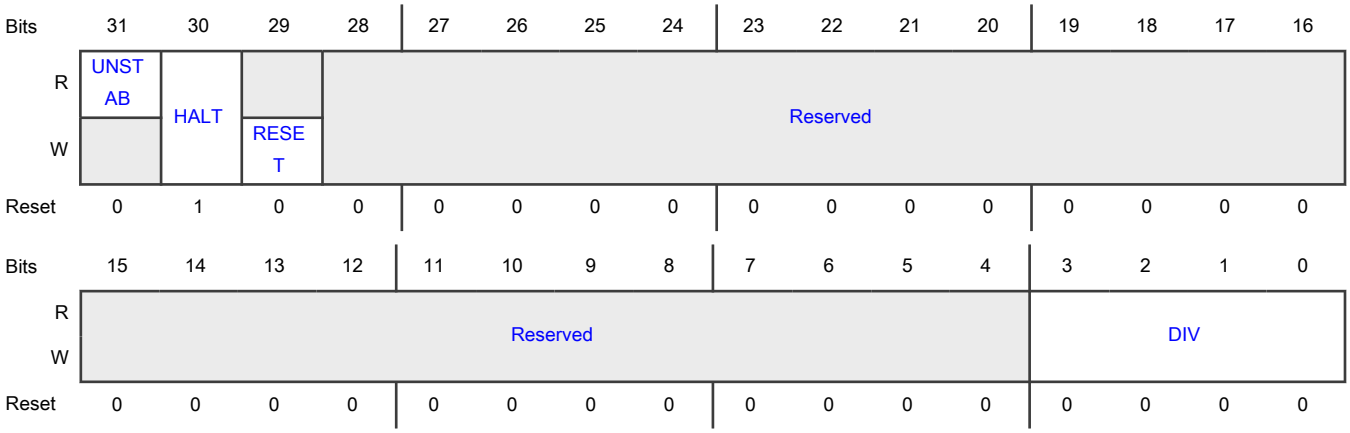
Field	Function
MUX	Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 010b - FRO_HF_DIV 011b - CLK_IN 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.62 CMP1_RR clock divider control (MRCC_CMP1_RR_CLKDIV)

Offset

Register	Offset
MRCC_CMP1_RR_CLKDIV	164h

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable

Table continues on the next page...

Table continued from the previous page...

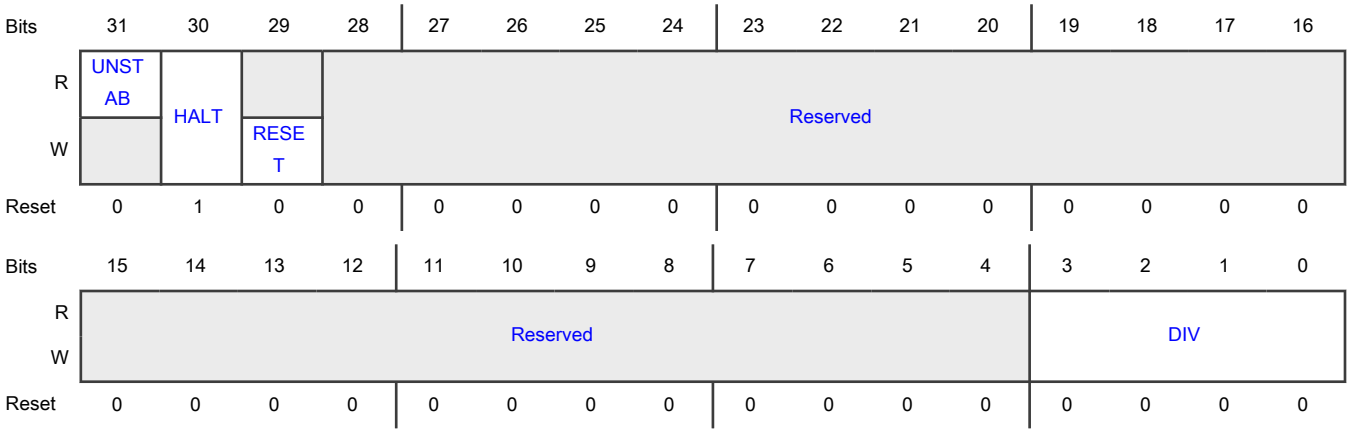
Field	Function
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.63 CMP2_FUNC clock divider control (MRCC_CMP2_FUNC_CLKDIV)

Offset

Register	Offset
MRCC_CMP2_FUNC_CLKDIV	16Ch

Diagram



Fields

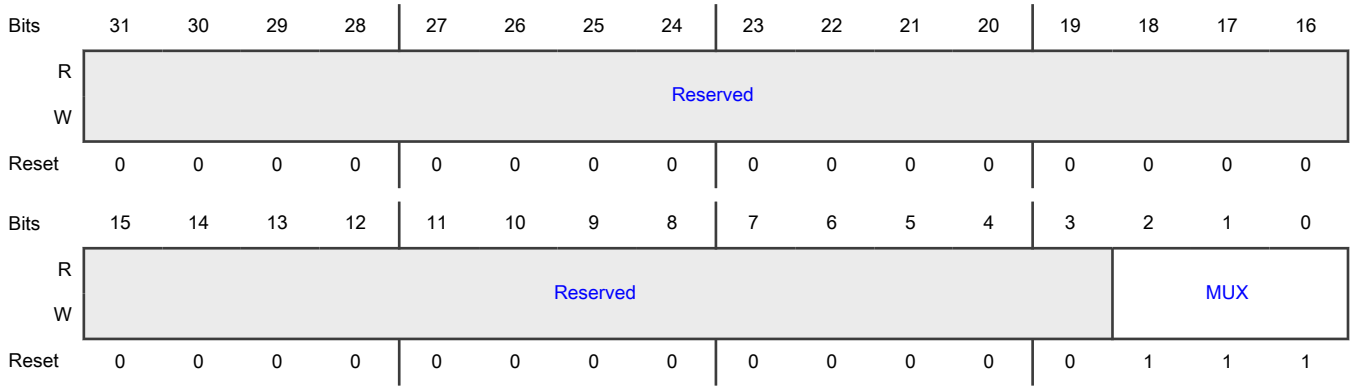
Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.64 CMP2_RR clock selection control (MRCC_CMP2_RR_CLKSEL)

Offset

Register	Offset
MRCC_CMP2_RR_CLKSEL	170h

Diagram



Fields

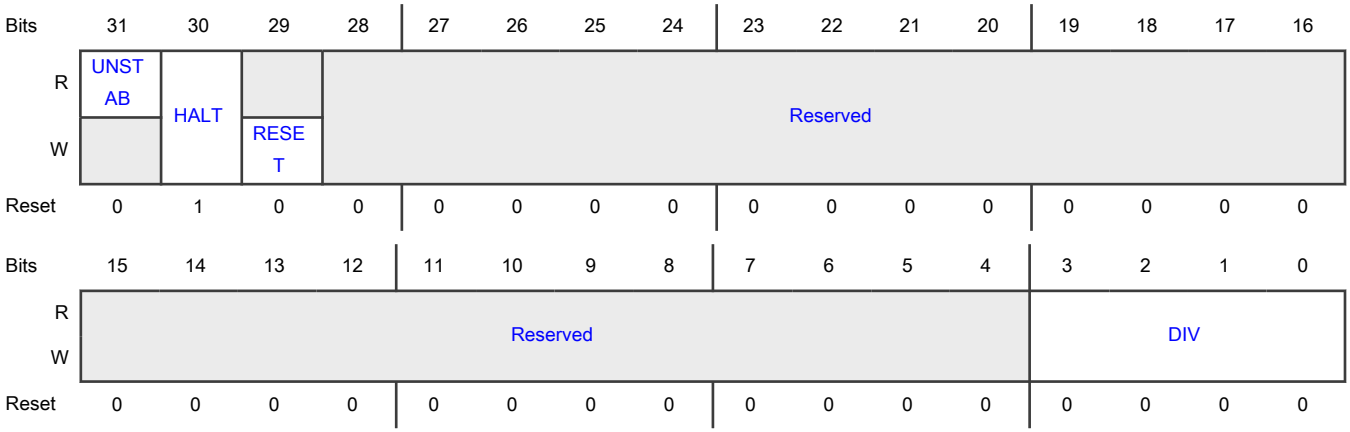
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 010b - FRO_HF_DIV 011b - CLK_IN 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.65 CMP2_RR clock divider control (MRCC_CMP2_RR_CLKDIV)

Offset

Register	Offset
MRCC_CMP2_RR_CLKDIV	174h

Diagram



Fields

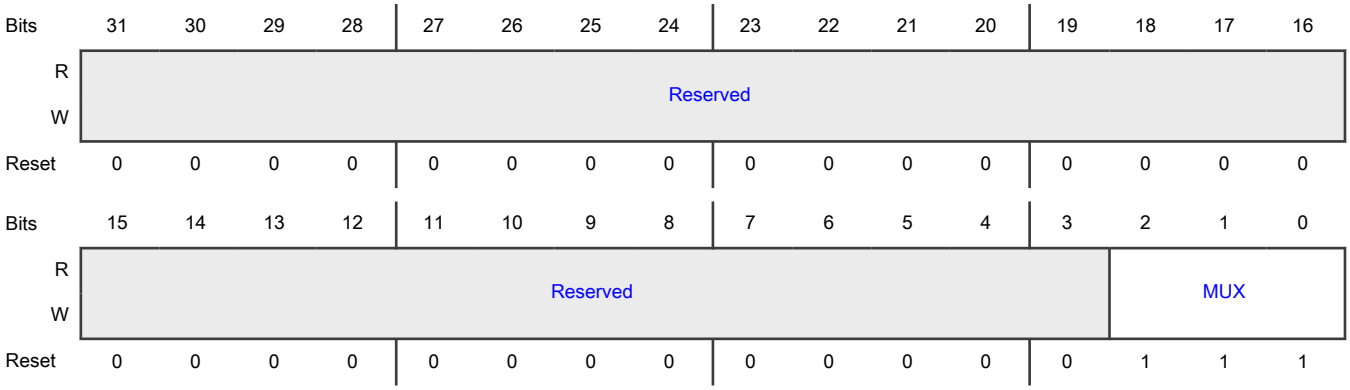
Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.66 DAC0 clock selection control (MRCC_DAC0_CLKSEL)

Offset

Register	Offset
MRCC_DAC0_CLKSEL	178h

Diagram



Fields

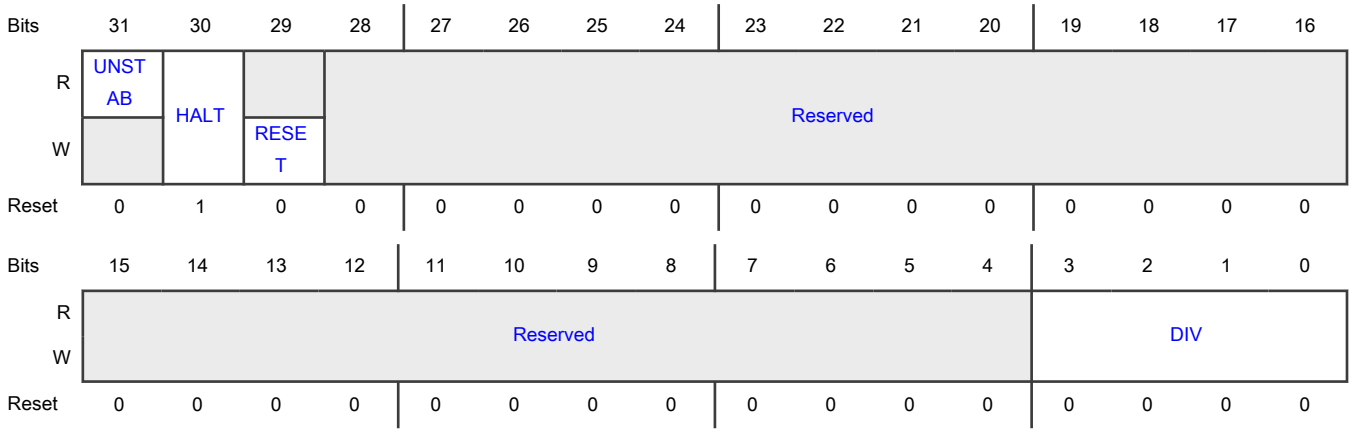
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 010b - FRO_HF_DIV 011b - CLK_IN 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.67 DAC0 clock divider control (MRCC_DAC0_CLKDIV)

Offset

Register	Offset
MRCC_DAC0_CLKDIV	17Ch

Diagram



Fields

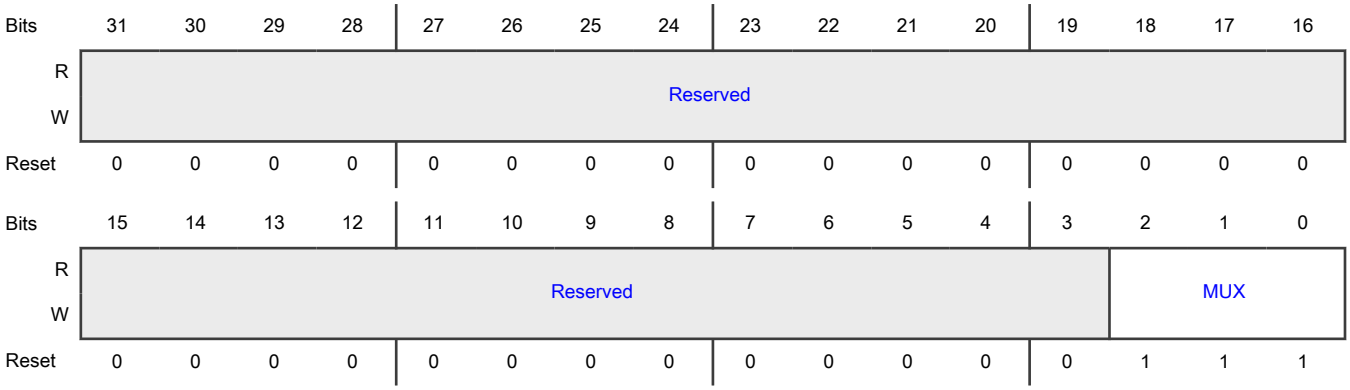
Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.68 FLEXCAN0 clock selection control (MRCC_FLEXCAN0_CLKSEL)

Offset

Register	Offset
MRCC_FLEXCAN0_CLKSEL	180h

Diagram



Fields

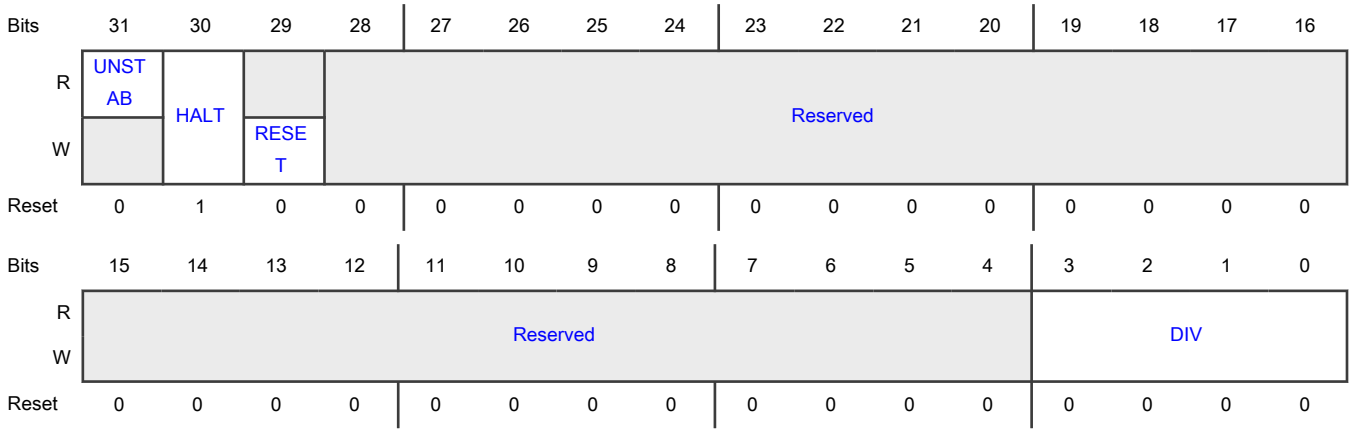
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 001b - FRO_HF_GATED 010b - FRO_HF_DIV 011b - CLK_IN 110b - PLL1_CLK 111b - Reserved(NO Clock)

11.5.2.69 FLEXCAN0 clock divider control (MRCC_FLEXCAN0_CLKDIV)

Offset

Register	Offset
MRCC_FLEXCAN0_CLKDIV	184h

Diagram



Fields

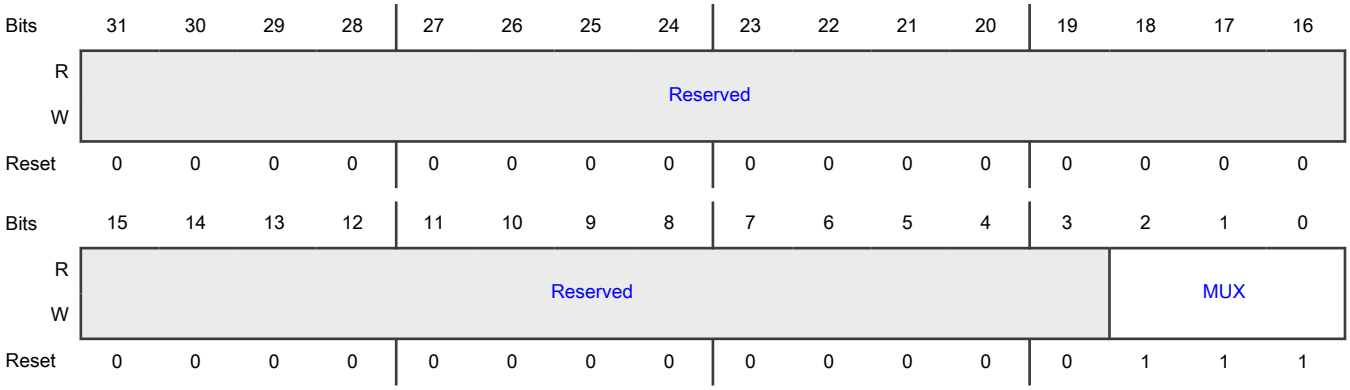
Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.70 FLEXCAN1 clock selection control (MRCC_FLEXCAN1_CLKSEL)

Offset

Register	Offset
MRCC_FLEXCAN1_CLKSEL	188h

Diagram



Fields

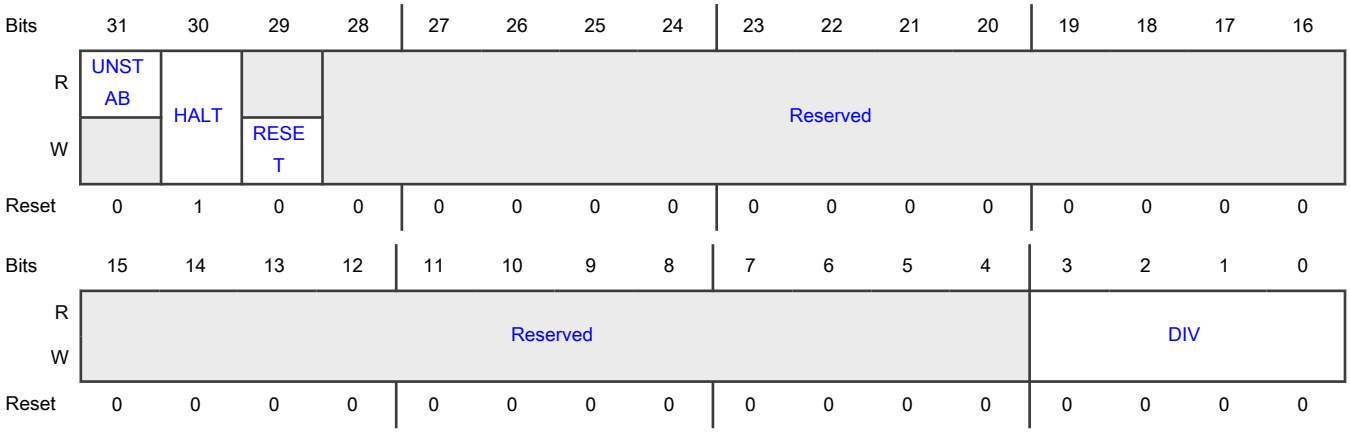
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 001b - FRO_HF_GATED 010b - FRO_HF_DIV 011b - CLK_IN 110b - PLL1_CLK 111b - Reserved(NO Clock)

11.5.2.71 FLEXCAN1 clock divider control (MRCC_FLEXCAN1_CLKDIV)

Offset

Register	Offset
MRCC_FLEXCAN1_CLKDIV	18Ch

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.72 LPI2C2 clock selection control (MRCC_LPI2C2_CLKSEL)

Offset

Register	Offset
MRCC_LPI2C2_CLKSEL	190h

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												MUX			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

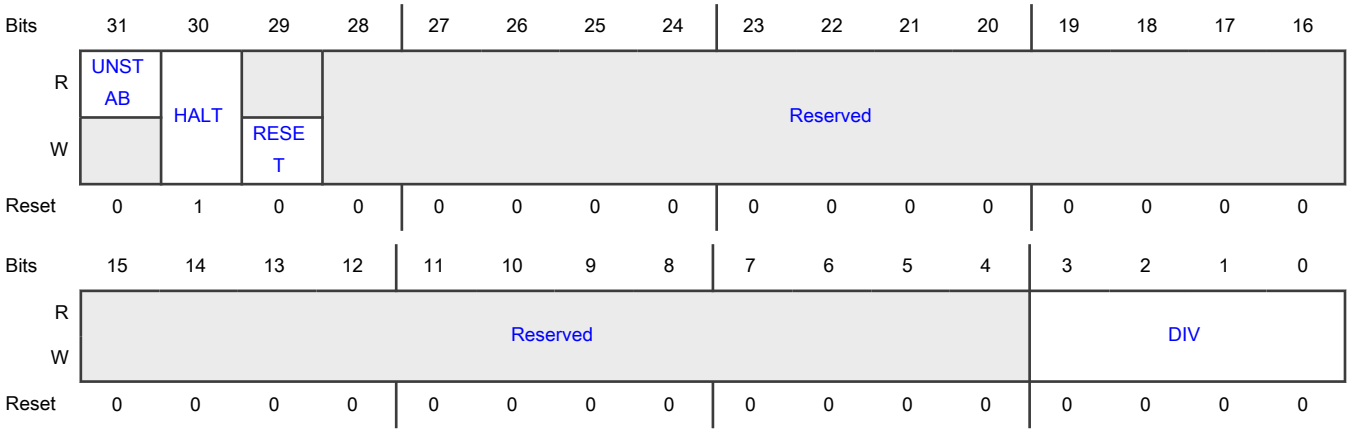
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 010b - FRO_HF_DIV 011b - CLK_IN 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.73 LPI2C2 clock divider control (MRCC_LPI2C2_CLKDIV)

Offset

Register	Offset
MRCC_LPI2C2_CLKDIV	194h

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.74 LPI2C3 clock selection control (MRCC_LPI2C3_CLKSEL)

Offset

Register	Offset
MRCC_LPI2C3_CLKSEL	198h

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	Reserved												MUX			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

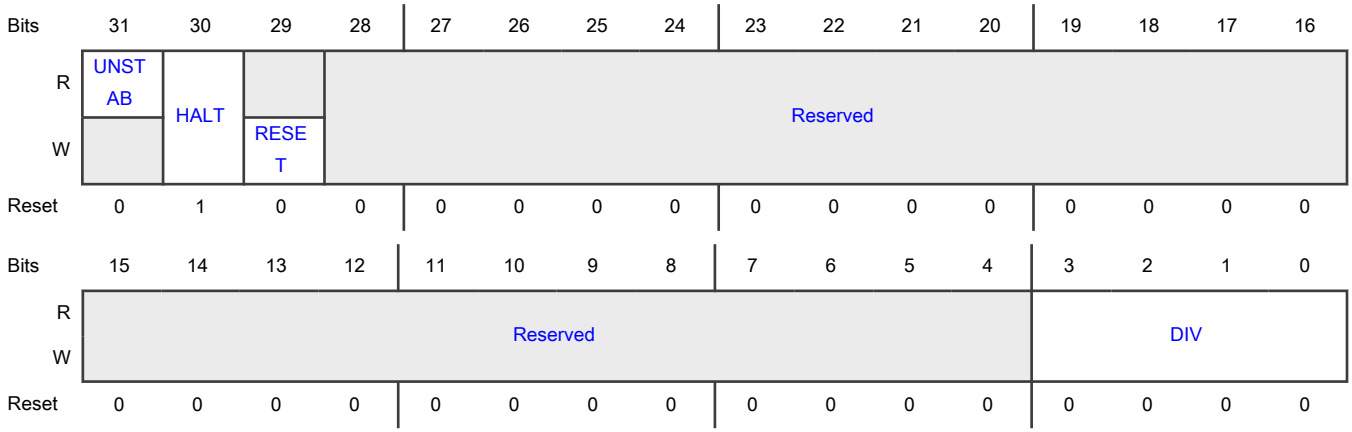
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 010b - FRO_HF_DIV 011b - CLK_IN 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.75 LPI2C3 clock divider control (MRCC_LPI2C3_CLKDIV)

Offset

Register	Offset
MRCC_LPI2C3_CLKDIV	19Ch

Diagram



Fields

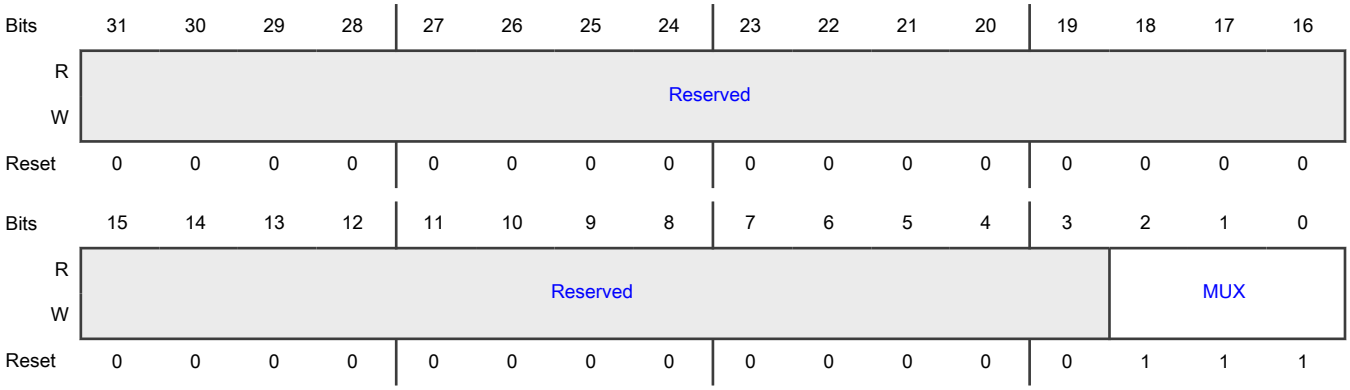
Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.76 LPUART5 clock selection control (MRCC_LPUART5_CLKSEL)

Offset

Register	Offset
MRCC_LPUART5_CLKSEL	1A0h

Diagram



Fields

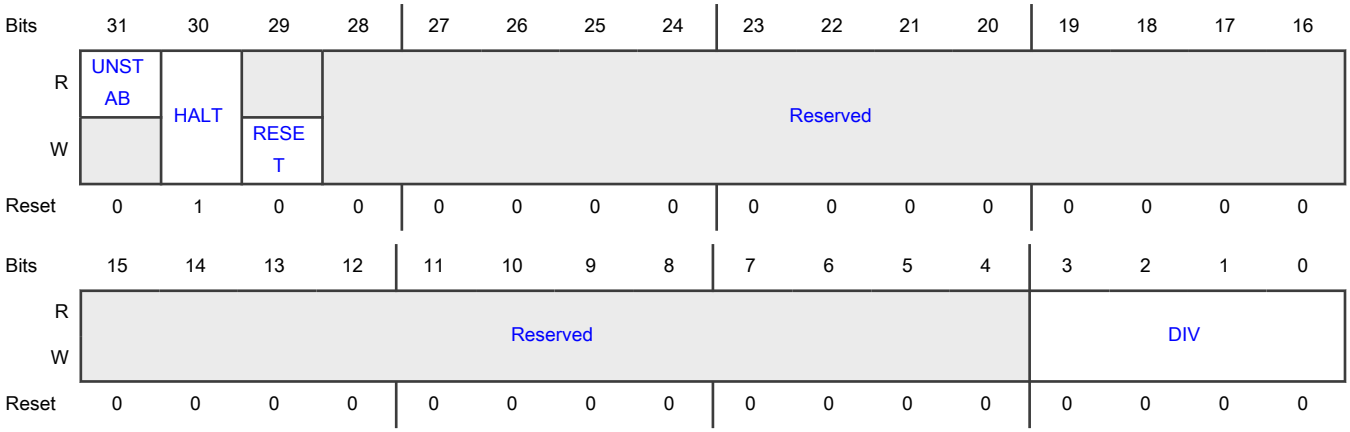
Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_LF_DIV 010b - FRO_HF_DIV 011b - CLK_IN 100b - CLK_16K 101b - CLK_1M 110b - PLL1_CLK_DIV 111b - Reserved(NO Clock)

11.5.2.77 LPUART5 clock divider control (MRCC_LPUART5_CLKDIV)

Offset

Register	Offset
MRCC_LPUART5_CLKDIV	1A4h

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.78 DBG_TRACE clock selection control (MRCC_DBG_TRACE_CLKSEL)

Offset

Register	Offset
MRCC_DBG_TRACE_CLKSEL	1A8h

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														MUX	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

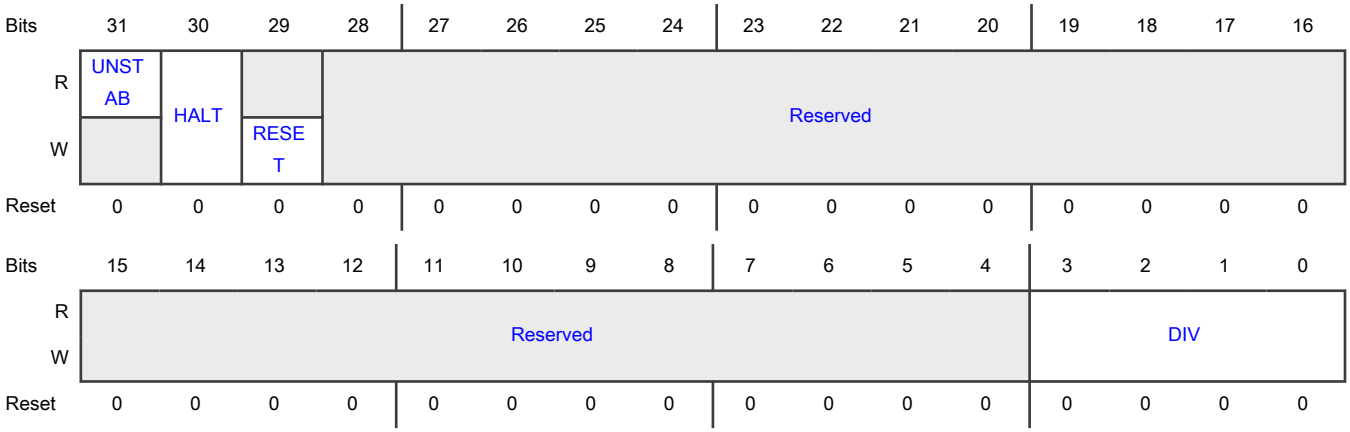
Field	Function
31-2 —	reserved reserved
1-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 00b - CPU_CLK 01b - CLK_1M 10b - CLK_16K 11b - Reserved1(NO Clock)

11.5.2.79 DBG_TRACE clock divider control (MRCC_DBG_TRACE_CLKDIV)

Offset

Register	Offset
MRCC_DBG_TRACE_CLKDIV	1ACh

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.5.2.80 CLKOUT clock selection control (MRCC_CLKOUT_CLKSEL)

Offset

Register	Offset
MRCC_CLKOUT_CLKSEL	1B0h

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													MUX		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

Field	Function
31-3 —	reserved reserved
2-0 MUX	Functional Clock Mux Select Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. 000b - FRO_12M 001b - FRO_HF_DIV 010b - CLK_IN 011b - CLK_16K

Table continues on the next page...

Table continued from the previous page...

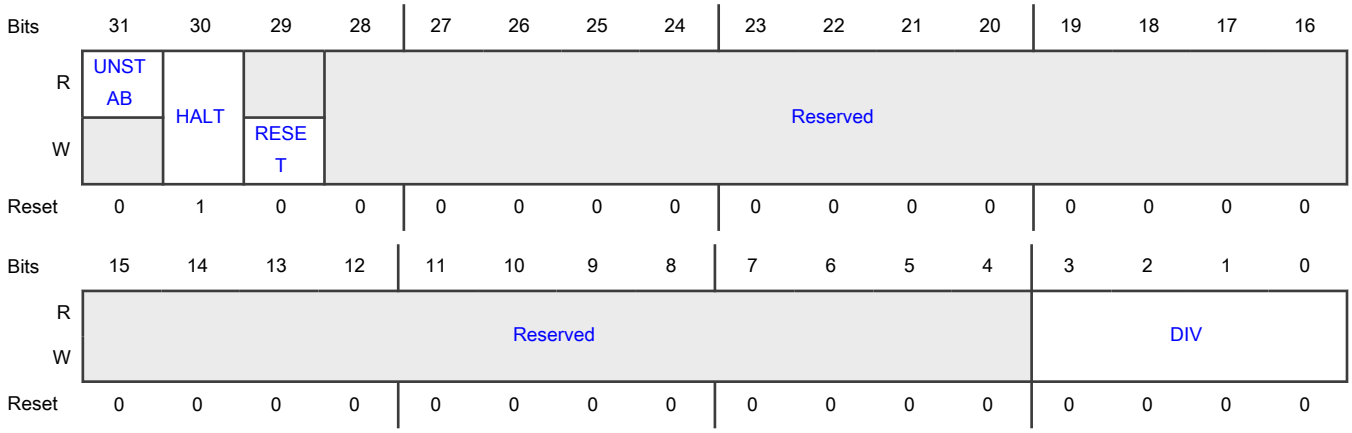
Field	Function
	101b - PLL1_CLK 110b - SLOW_CLK 111b - Reserved(NO Clock)

11.5.2.81 CLKOUT clock divider control (MRCC_CLKOUT_CLKDIV)

Offset

Register	Offset
MRCC_CLKOUT_CLKDIV	1B4h

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag Determines the stability of the divider clock. 0b - Divider clock is stable 1b - Clock frequency isn't stable
30 HALT	Halt divider counter Halt divider counter 0b - Divider clock is running 1b - Divider clock is stopped

Table continues on the next page...

Table continued from the previous page...

Field	Function
29 RESET	Reset divider counter Reset divider counter 0b - Divider isn't reset 1b - Divider is reset
28-4 —	reserved reserved
3-0 DIV	Functional Clock Divider Before change peripheral's funcitional clock source, should configure the module's MRCC_GLB_CC and MRCC_GLB_ACC bits to 0. divider value = (DIV+1)

11.6 Application information

Peripheral clock is gated by using CC (CC = {MRCC_GLB_ACC.*[peripherals name]*, MRCC_GLB_CC.*[peripherals name]* })

1. CC = 0b00 : Peripheral clocks are disabled; module does not stall low power mode entry. Clock is always gated, peripheral access causes hard fault.
2. CC = 0b01 : Peripheral clocks are enabled; module does not stall low power mode entry. Clock is never gated unless root clock source is gated in DEEPSLEEP mode.
3. CC = 0b10 : Peripheral clocks are enabled unless module is idle; low power mode entry stalls until module is idle. Only LPUART, LPSPI, LPI2C, FLEXIO, GPIO, FMU support CC=0b10. for other modules, it's same as CC=0b11.
4. CC = 0b11 : Peripheral clocks are enabled unless in DEEPSLEEP (or lower) mode; low power mode entry stalls until module is idle.

NOTE

FMC's CC bits are only available during low power mode, FMC is accessible and no hardfault occurs if CC=0b00 in RUN mode.

Chapter 12
Code Watchdog Timer (CDOG)

12.1 Chip-specific CDOG information

Table 75. Reference links to related information[1]

Topic	Related module	References
Full description	CDOG	CDOG
System memory map		System memory map
Clocking		See the chapter "Clocking"
Signal multiplexing	Port control	See the chapter "Signal multiplexing"
System debug		See the chapter "Debug"

1. For all the reference sections and chapters mentioned in this table, refer the Reference Manual.

12.1.1 Module instance

This device has two instances of the CDOG module, CDOG0 and CDOG1.

12.1.2 Security considerations

For secure applications, NXP recommends that this module is configured for secure and privileged access only to prevent unintentional or malicious modification of the system operation by nonsecure software.

12.2 Overview

CDOG helps protect the integrity of software by detecting unexpected changes (faults) in code execution flow. This module can be configured to reset or interrupt the processor core when it detects a fault.

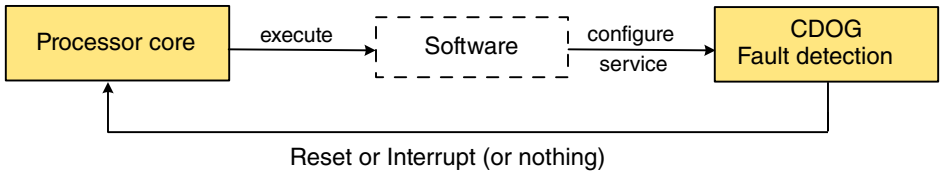


Figure 37. CDOG overview

12.2.1 Block diagram

The following block diagram shows the components of CDOG.

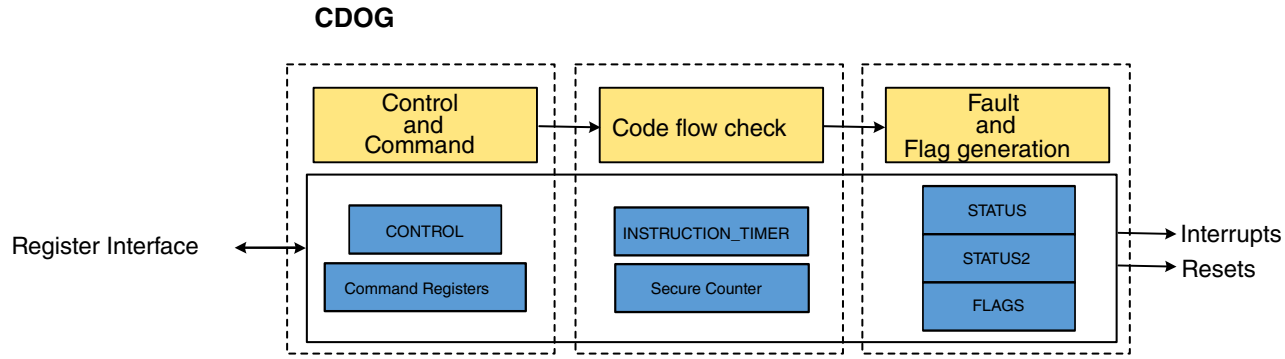


Figure 38. Block diagram of CDOG

12.2.2 Features

CDOG includes the following code flow and data integrity checking:

- Faults and flags configurable to generate a system reset, interrupts, or nothing
- Counters for statistics on code behavior patterns for fault types

12.3 Functional description

The following sections describe functional details of this module.

12.3.1 Code flow checking

CDOG provides two primary mechanisms for detecting low-cost fault attacks and the execution of unexpected instruction sequences:

- Secure Counter
- Instruction Timer (INSTRUCTION_TIMER)

12.3.1.1 Secure counter

The Secure Counter is a 32-bit accumulator that holds a dynamically changing value that can periodically be evaluated to determine if a program is executing as expected. If a mismatch is detected, a fault is generated.

- Secure Counter is an accumulator that when loaded with an initial value lets runtime software issue ADD & SUB commands to increment/decrement the counter.
- Periodically, a Secure Counter value check is initiated by passing the expected value to STOP, RESTART, or ASSERT16 command register. The value passed to STOP, RESTART, or ASSERT16 is compared with the current value of the Secure Counter.
- If a mismatch is detected between the Secure Counter and the value written to STOP, RESTART, or ASSERT16 command register, the execution flow has potentially been altered by a fault attack or some other suspicious activity.

12.3.1.2 Instruction timer

The Instruction Timer is a 32-bit count-down timer that an application uses to set the number of instructions that software expects to execute. The Instruction Timer counts off the instructions as they are executed (clocks) and if the number is exhausted before the CDOG is serviced, a fault is generated.

The application programmer pre-loads the Instruction Timer with slightly more than the number of instructions in the next execution sequence. The CDOG detects cases where the counter is reset to 0 before all instructions execute, which may indicate that unauthorized instructions are being executed.

- Instruction Timer places a hard upper-limit on the interval between checks of the Secure Counter.

- The Instruction Timer can be programmed to either pause, or keep running while the interrupt service routines are executing.
- The START command loads the internal decremental counter. Before the counter generates an underflow (reaches 0), a STOP or RESTART command must be executed to force a Secure Counter check.

12.3.2 Modes of operation (STATE)

The CDOG has two legal states: IDLE and ACTIVE.

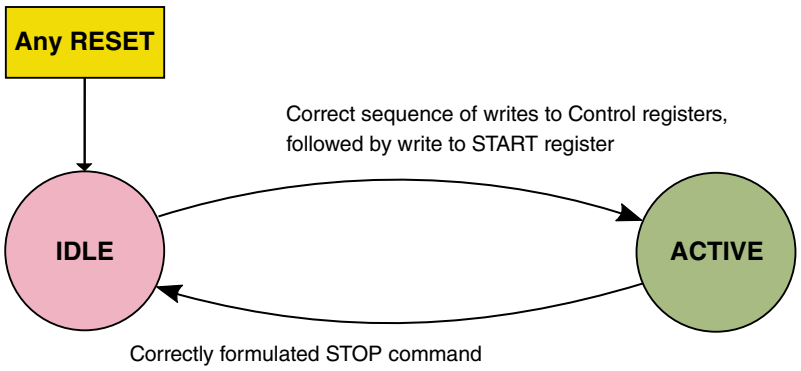


Figure 39. CDOG state diagram

- The two states are encoded in the read-only STATUS[CURST] field.
- After any reset (including a reset generated by the CDOG itself), the module will be in IDLE state.
- To change the module state to ACTIVE, software must execute a correct sequence of writes to the CONTROL group, followed by a START command. See Example use cases.
- Once ACTIVE, a correctly formulated STOP command will change the state back to IDLE.

12.3.3 Faults, flags, and counters

12.3.3.1 Fault types

All fault types (except CONTROL) can be individually controlled to generate either a system reset, an interrupt, or nothing.

Table 76. Fault types

This fault...	Occurs when...
TIMEOUT	The Instruction Timer reaches '0'.
MISCOMPARE	Either a STOP or a RESTART command is issued, and the value passed in the instruction does not match the content of the Secure Counter. ASSERT16 command is issued, and the lower 16-bit value passed in the instruction does not match the lower 16-bit value of the Secure Counter.
SEQUENCE	The prescribed sequence of interactions between software and CDOG is violated.
CONTROL	Any of the CONTROL register's fields contain an illegal value.
STATE	The internal state machine contains any value other than 5h or Ah.
ADDRESS	When an undefined address in the module's register space is accessed.

Table continues on the next page...

Table 76. Fault types (continued)

This fault...	Occurs when...
	NOTE Address 0xC is reserved, but no ADDRESS fault will be generated when 0xC is accessed.

For CONTROL fault, a system reset is requested when FLAGS[CNT_FLAG] = 1 and CONTROL[LOCK_CTRL] != 10b.

Lock the CONTROL register by writing CONTROL[LOCK_CTRL] == 01b at the same time that the desired, valid configuration is written. This blocks additional writes, and prevents CONTROL faults.

Interrupts are available as an alternative to system reset generation, for all fault types (except CONTROL), primarily to facilitate code development and debug operations. In the final application, interrupts can be enabled for some faults, and reset can be enabled for other faults (or disabled completely), at the risk of reduced security.

12.3.3.2 Fault counters

Each fault type has an associated counter that increments when the fault is detected. The counters are cleared by POR, but not by a system reset. Thus, statistics can be built up over many code watchdog resets to reveal the behavior patterns of a specific type of attack.

12.3.3.3 Flags

Each fault type has an associated flag accessible through the FLAGS register. A flag sets whenever its associated fault is detected, and can be cleared by software. The flags themselves (when enabled) generate the module's system reset or interrupt outputs.

The flags retain their value through a system reset (including one generated by the CDOG), but are reset by a POR. Using this mechanism, software can easily answer the 'How did I get here?' question by reading the FLAGS register after any reset.

To facilitate testing and code development, write to the flags directly when the module is not locked (CONTROL[LOCK_CTRL] = 10b). When the module is locked (CONTROL[LOCK_CTRL] = 01b), the flags can be cleared by writing '1' to their bit positions.

Table 77. FLAGS summary

Name	POR	ADDR	STATE	CONTROL	SEQUENCE	MISCOMPARE	TIMEOUT
Bit	16	5	4	3	2	1	0
Reset value after POR?	1	0	0	0	0	0	0
Writeable when unlocked?	Y	Y	Y	Y	Y	Y	Y
W1C when locked?	Y	Y	Y	Y	Y	Y	Y

12.3.3.4 Fault generation

Some faults are related to when and how registers can be accessed. See Figure 40.

The CDOG supports the following faults:

Table 78. Fault generation

This fault...	Is generated when...
SEQUENCE	• Software does not write to RELOAD before the START command is issued. • Software writes to RELOAD in ACTIVE state. Only write to RELOAD in IDLE state. • Software attempts to write to START in ACTIVE state. Only write to START in IDLE state. • In IDLE state, software attempts to write to a COMMAND group register other than START. • In ACTIVE state, software attempts to write to a CONTROL group register.
ADDRESS	• A read access (within the CDOG address space) to any address outside the CONTROL register group. • A write access (within the CDOG address space) to any address outside both the CONTROL and COMMAND register groups. **NOTE** Address 0xC is reserved in the address space of the CONTROL register group.
TIMEOUT	• The countdown is one clock before the Instruction Timer reaches '0'.
MISCOMPARE	• The STOP register is written and the current Secure Counter value does not match the write data. • The RESTART register is written and the current Secure Counter value does not match the write data. • The ASSERT16 register is written and the current Secure Counter lower 16-bit value does not match the 16-bit write data.
CONTROL	• The CONTROL[LOCK_CTRL] field is any value other than 01b or 10b. • The CONTROL bit fields: TIMEOUT_CTRL, MISCOMPARE_CTRL, SEQUENCE_CTRL, STATE_CTRL, or ADDRESS_CTRL contain any value other than 001b, 010b, or 100b. • The CONTROL[DEBUG_HALT_CTRL] field contains any value other than 01b or 10b. • The CONTROL[IRQ_PAUSE] field contains any value other than 01b or 10b.
STATE	• The STATE machine contains any value other than 5h or Ah. See STATUS[CURST].

12.3.4 Clocking

CDOG has only one clock input. See the chip-specific CDOG section for information on available clock inputs and the default option for a specific device.

12.3.5 Reset

CDOG has two hardware reset input sources: power-on reset (POR) and system reset.

CDOG can generate a system reset using the CONTROL register. See Faults, flags, and counters for details on how CDOG generates a system reset.

12.3.5.1 Power on reset (POR)

The logic and all registers of this module are reset to their default states on a POR.

12.3.5.2 System reset

The logic and most of the registers of this module are reset to their default states on a system reset.

The following registers cannot be reset by a system reset:

- Persistent Data Storage (PERSISTENT)
- Flags (FLAGS)
- Status 1 (STATUS), except the STATUS[CURST] field
- Status 2 (STATUS2)

12.3.6 Interrupts

CDOG has only one interrupt output. The interrupt signal output by CDOG can be connected to the system's interrupt controller. The following table lists CDOG's interrupt sources:

Table 79. Interrupt Summary

Interrupt flag	Interrupt enable	Description
FLAGS[TO_FLAG]	CONTROL[TIMEOUT_CTRL] = 3'b010	TIMEOUT fault interrupt
FLAGS[MISCOM_FLAG]	CONTROL[MISCOMPARE_CTRL] = 3'b010	MISCOMPARE fault interrupt
FLAGS[SEQ_FLAG]	CONTROL[SEQUENCE_CTRL] = 3'b010	SEQUENCE fault interrupt
FLAGS[STATE_FLAG]	CONTROL[STATE_CTRL] = 3'b010	STATE fault interrupt
FLAGS[ADDR_FLAG]	CONTROL[ADDRESS_CTRL] = 3'b010	ADDRESS fault interrupt

12.4 Application information

This section gives examples on how to program CDOG to monitor code execution flow.

12.4.1 Example use cases

There is generally a strict sequence to configure the CDOG, activated and serviced as follows:

1. Write an Instruction Timer reload value, corresponding to the current code section, to the Instruction Timer Reload Register (RELOAD) register.
2. Write a control word to the Control Register (CONTROL) register, where each fault type is configured to generate a reset, interrupt, or neither. Then lock the Control register using the CONTROL[LOCK_CTRL] field. Instruction Timer Register (INSTRUCTION_TIMER) may be always kept running, or run only during non-IRQ execution (paused during interrupt processing) based on the value in the CONTROL[IRQ_PAUSE] field.
3. Activate the module by writing to the START Command Register (START) register. The initial value for the Secure Counter is the value written. Instruction Timer Register (INSTRUCTION_TIMER) immediately starts decrementing from the RELOAD value on every clock cycle.
4. At strategically chosen way points in the code flow of the application, update the Secure Counter by issuing ADD or SUB commands, or assert the lower 16-bit value of the Secure Counter by the ASSERT16 command.

5. When the way point that corresponds to the number of executed instructions represented by the RELOAD value (the end of the current code section) is reached, write the expected value of the Secure Counter to the STOP command register. Assuming the written value compares exactly to the contents of the Secure Counter, the instruction timer stops. Then the process can begin again for the next code section, by repeating steps 1 through 5.

Another example of a configuration and start procedure is as follows:

1. Write the Instruction Timer Register (INSTRUCTION_TIMER) reload value to the Instruction Timer Reload Register (RELOAD) register. Write a very large number to provide a buffer for large values.
2. Write a control word to the Control Register (CONTROL) register, where each fault type is configured to generate a reset, interrupt, or neither. Then lock the Control register using the CONTROL[LOCK_CTRL] field.
3. Activate the module by writing to the START Command Register (START) register. The initial value for the Secure Counter is the value written. INSTRUCTION_TIMER immediately starts decrementing from the RELOAD value on every clock.
4. At strategically chosen way points in the code flow of the application, update the Secure Counter by issuing ADD or SUB commands, or assert the lower 16-bit value of the Secure Counter by the ASSERT16 command.
5. During the execution flow, the application must always be aware of the INSTRUCTION_TIMER register value approach toward 0.
    a. Read the INSTRUCTION_TIMER value regularly to determine the expended time.
    b. Before the INSTRUCTION_TIMER value reaches 0, software must service the CDOG by writing the Secure Counter expected value to the RESTART command register.
    c. Assuming the value written compares exactly to the contents of the Secure Counter, INSTRUCTION_TIMER reloads with the value in the RELOAD register as it decrements toward 0.

12.4.2 Using CDOG during code development debug

Following are suggestions for facilitating the code development and debug cycle:

- Maintain control while adjusting the timing and fine-tuning of the counting values by using interrupt-on-fault (instead of reset).
- Trigger faults by direct-writing the bits in the FLAGS register with the same result as hardware-triggered faults.
- Pause the instruction timer during a pause of a debug session by using the CONTROL[DEBUG_HALT_CTRL] field.

12.5 Memory map and register definition

This section includes the memory map and detailed descriptions of all registers of this module.

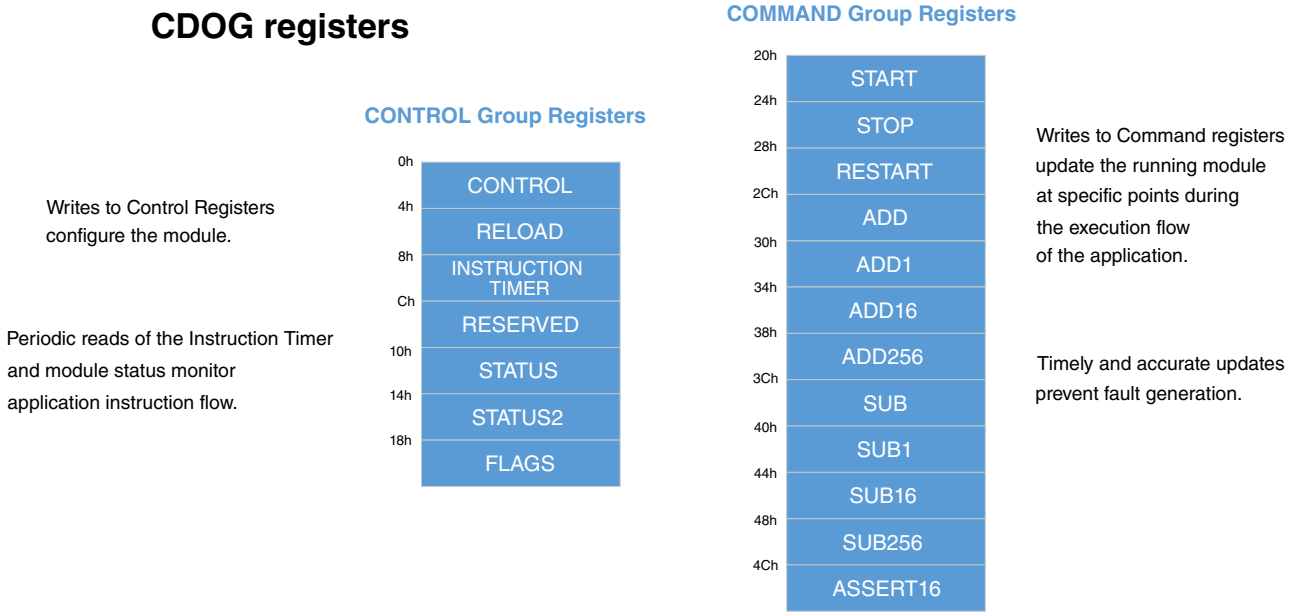


Figure 40. CDOG register groups

12.5.1 CDOG register descriptions

12.5.1.1 CDOG memory map

CDOG0 base address: 4010_0000h

CDOG1 base address: 4010_7000h

Offset	Register	Width (In bits)	Access	Reset value
0h	Control Register (CONTROL)	32	RW	5009_2492h
4h	Instruction Timer Reload Register (RELOAD)	32	RW	FFFF_FFFFh
8h	Instruction Timer Register (INSTRUCTION_TIMER)	32	R	FFFF_FFFFh
10h	Status 1 Register (STATUS)	32	R	5000_0000h
14h	Status 2 Register (STATUS2)	32	R	0000_0000h
18h	Flags Register (FLAGS)	32	RW	0001_0000h
1Ch	Persistent Data Storage Register (PERSISTENT)	32	RW	0000_0000h
20h	START Command Register (START)	32	W	0000_0000h
24h	STOP Command Register (STOP)	32	W	0000_0000h
28h	RESTART Command Register (RESTART)	32	W	0000_0000h
2Ch	ADD Command Register (ADD)	32	W	0000_0000h
30h	ADD1 Command Register (ADD1)	32	W	0000_0000h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
34h	ADD16 Command Register (ADD16)	32	W	0000_0000h
38h	ADD256 Command Register (ADD256)	32	W	0000_0000h
3Ch	SUB Command Register (SUB)	32	W	0000_0000h
40h	SUB1 Command Register (SUB1)	32	W	0000_0000h
44h	SUB16 Command Register (SUB16)	32	W	0000_0000h
48h	SUB256 Command Register (SUB256)	32	W	0000_0000h
4Ch	ASSERT16 Command Register (ASSERT16)	32	W	0000_0000h

12.5.1.2 Control Register (CONTROL)

Offset

Register	Offset
CONTROL	0h

Function

The Control register (CONTROL) contains all the controllable attributes of the module, such as how the CDOG module responds when detecting a fault.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DEBUG_HALT_CTRL		IRQ_PAUSE		0								ADDRESS_CTRL			STATE_C...
W																
Reset	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	STATE_CTRL		Reserved		SEQUENCE_CTRL				MISCOMPARE_CTRL				TIMEOUT_CTRL			LOCK_CTRL
W																
Reset	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0

Fields

Field	Function
31-30	DEBUG_HALT control Controls whether the Instruction Timer runs or stops when the CPU asserts DEBUG_HALT.

Table continues on the next page...

Table continued from the previous page...

Field	Function
DEBUG_HALT_CTRL	See chip-specific CDOG section for information on DEBUG_HALT connections for a specific device. 01b - Keep the timer running 10b - Stop the timer
29-28 IRQ_PAUSE	IRQ pause control Controls whether the Instruction Timer runs or stops when the CPU asserts IRQ_PAUSE. See chip-specific CDOG section for information on IRQ_PAUSE connections for a specific device. 01b - Keep the timer running 10b - Stop the timer
27-20 —	Reserved **NOTE** This field is not testable by an automated register test.
19-17 ADDRESS_CTRL	ADDRESS fault control Controls how the CDOG module responds when detecting an ADDRESS fault (FLAGS[ADDR_FLAG]). 001b - Enable reset 010b - Enable interrupt 100b - Disable both reset and interrupt
16-14 STATE_CTRL	STATE fault control Controls how the CDOG module responds when detecting a STATE fault (FLAGS[STATE_FLAG]). 001b - Enable reset 010b - Enable interrupt 100b - Disable both reset and interrupt
13-11 —	Reserved Do not change its value.
10-8 SEQUENCE_CTRL	SEQUENCE fault control Controls how the CDOG module responds when detecting a SEQUENCE fault (FLAGS[SEQUENCE_FLAG]). 001b - Enable reset 010b - Enable interrupt 100b - Disable both reset and interrupt
7-5 MISCOMPARE_CTRL	MISCOMPARE fault control Controls how the CDOG module responds when detecting a MISCOMPARE fault (FLAGS[MISCOMPARE_FLAG]).

Table continues on the next page...

Table continued from the previous page...

Field	Function
	001b - Enable reset 010b - Enable interrupt 100b - Disable both reset and interrupt
4-2 TIMEOUT_CTRL	TIMEOUT fault control Controls how the CDOG module responds when detecting a TIMEOUT fault (FLAGS[TIMEOUT_FLAG]). 001b - Enable reset 010b - Enable interrupt 100b - Disable both reset and interrupt
1-0 LOCK_CTRL	Lock control Resets to unlocked. Once locked, LOCK_CTRL remains locked until the next system reset. The Control register is writable and readable only when unlocked, LOCK_CTRL = 10. Once locked, the written value to the Control register is ignored, and reading the Control register returns 0. 01b - Locked 10b - Unlocked

12.5.1.3 Instruction Timer Reload Register (RELOAD)

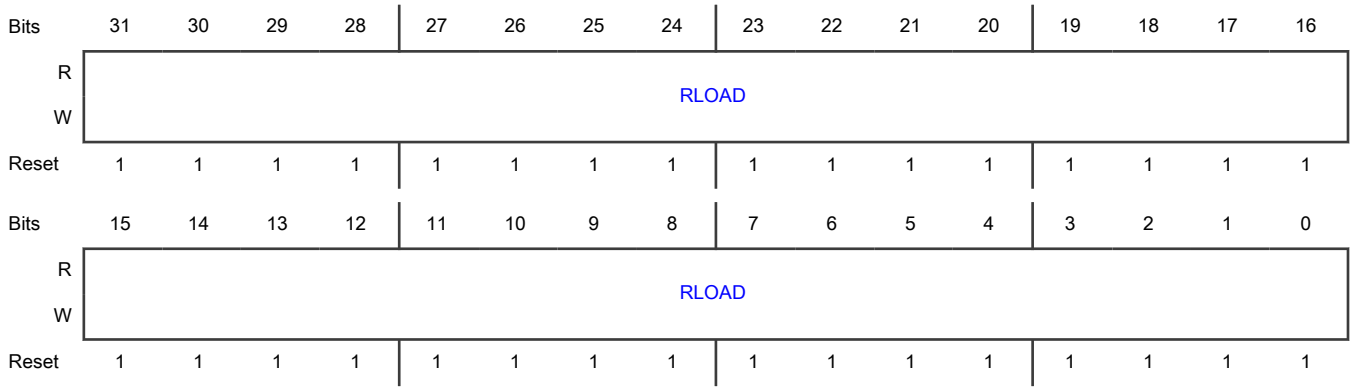
Offset

Register	Offset
RELOAD	4h

Function

The Instruction Timer RELOAD register contains the value from which the Instruction Timer counts down when a valid START or RESTART command is executed. Write to RELOAD only when the module is in the IDLE state. A write to RELOAD must occur before issuing a START command.

Diagram



Fields

Field	Function
31-0 RLOAD	Instruction Timer reload value Contains the starting countdown value used by the Instruction Timer whenever a START or RESTART command is executed.

12.5.1.4 Instruction Timer Register (INSTRUCTION_TIMER)

Offset

Register	Offset
INSTRUCTION_TIMER	8h

Function

The Instruction Timer register (INSTRUCTION_TIMER) contains the current count value of the Instruction Timer.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	INSTIM															
W																
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	INSTIM															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Fields

Field	Function
31-0 INSTIM	Current value of the Instruction Timer The current value of the timer can be read from this address. The Software cannot write to INSTRUCTION_TIMER.

12.5.1.5 Status 1 Register (STATUS)

Offset

Register	Offset
STATUS	10h

Function

The Status 1 register (STATUS) holds the current module status and fault counts. The fields (except CURST) in STATUS are reset only by POR.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CURST				0				NUMILSEQF							
W																
Reset	0	1	0	1	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	NUMMISCOMPF								NUMTOF							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-28 CURST	Current State Indicates the current state of the module. The only legal states are: • 0x5 IDLE • 0xA ACTIVE Any other value generates a STATE fault (FLAGS[STATE_FLAG]). Resets to 0x5 = IDLE on any reset (unlike other bits in this register that only reset on POR).
27-24 —	Reserved **NOTE** This field is not testable by an automated register test.
23-16 NUMILSEQF	Number of SEQUENCE faults (FLAGS[SEQUENCE_FLAG]) since the last POR Resets to 0. Will not increment past 0xFF.
15-8 NUMMISCOMPF	Number of MISCOMPARE faults (FLAGS[MISCOMPARE_FLAG]) since the last POR Resets to 0. Will not increment past 0xFF.
7-0 NUMTOF	Number of TIMEOUT faults (FLAGS[TIMEOUT_FLAG]) since the last POR Resets to 0. Will not increment past 0xFF.

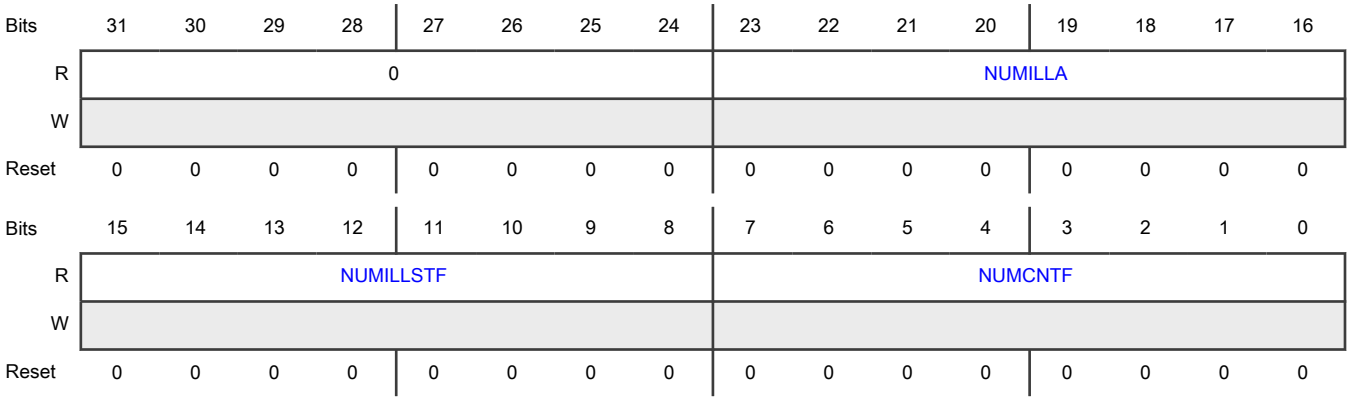
12.5.1.6 Status 2 Register (STATUS2)

Offset

Register	Offset
STATUS2	14h

Function
Status 2 register (STATUS2) holds additional fault counts. The fields in STATUS2 are reset only by POR.

Diagram



Fields

Field	Function
31-24 —	Reserved
23-16 NUMILLA	Number of ADDRESS faults (FLAGS[ADDR_FLAG]) since the last POR Resets to 0. Will not increment past 0xFF.
15-8 NUMILLSTF	Number of STATE faults (FLAGS[STATE_FLAG]) since the last POR Resets to 0. Will not increment past 0xFF.
7-0 NUMCNTF	Number of CONTROL faults (FLAGS[CONTROL_FLAG]) since the last POR Resets to 0. Will not increment past 0xFF.

12.5.1.7 Flags Register (FLAGS)

Offset

Register	Offset
FLAGS	18h

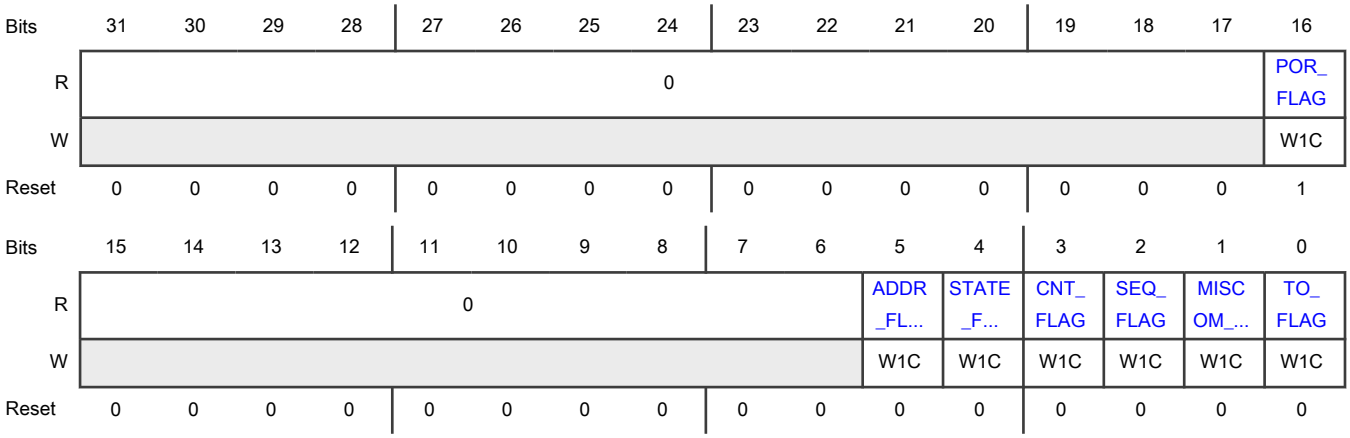
Function

The Flags register (FLAGS) contains the fault detection flags, as well as a POR event flag. The fault detection flags may generate a reset or an interrupt as configured in the CONTROL register.

The FLAGS fields retain their value through a system reset, including one generated by the CDOG. However, FLAGS is reset by a POR.

Writability of the fields depends on the module state. See Flags.

Diagram



Fields

Field	Function
31-17 —	Reserved
16 POR_FLAG	Power-on reset flag POR_FLAG is set by a Power-on reset event. 0b - A Power-on reset event has not occurred 1b - A Power-on reset event has occurred
15-6 —	Reserved
5 ADDR_FLAG	ADDRESS fault flag Hardware sets ADDR_FLAG when CDOG detects an ADDRESS fault. 0b - An ADDRESS fault has not occurred 1b - An ADDRESS fault has occurred
4 STATE_FLAG	STATE fault flag Hardware sets STATE_FLAG when CDOG detects a STATE fault. 0b - A STATE fault has not occurred

Table continues on the next page...

Table continued from the previous page...

Field	Function
	1b - A STATE fault has occurred
3 CNT_FLAG	CONTROL fault flag Hardware sets CNT_FLAG when CDOG detects a CONTROL fault. 0b - A CONTROL fault has not occurred 1b - A CONTROL fault has occurred
2 SEQ_FLAG	SEQUENCE fault flag Hardware sets the SEQ_FLAG when CDOG detects a SEQUENCE fault. 0b - A SEQUENCE fault has not occurred 1b - A SEQUENCE fault has occurred
1 MISCOM_FLAG	MISCOMPARE fault flag Hardware sets MISCOM_FLAG when CDOG detects a MISCOMPARE fault. 0b - A MISCOMPARE fault has not occurred 1b - A MISCOMPARE fault has occurred
0 TO_FLAG	TIMEOUT fault flag Hardware sets TO_FLAG when CDOG detects a TIMEOUT fault. 0b - A TIMEOUT fault has not occurred 1b - A TIMEOUT fault has occurred

12.5.1.8 Persistent Data Storage Register (PERSISTENT)

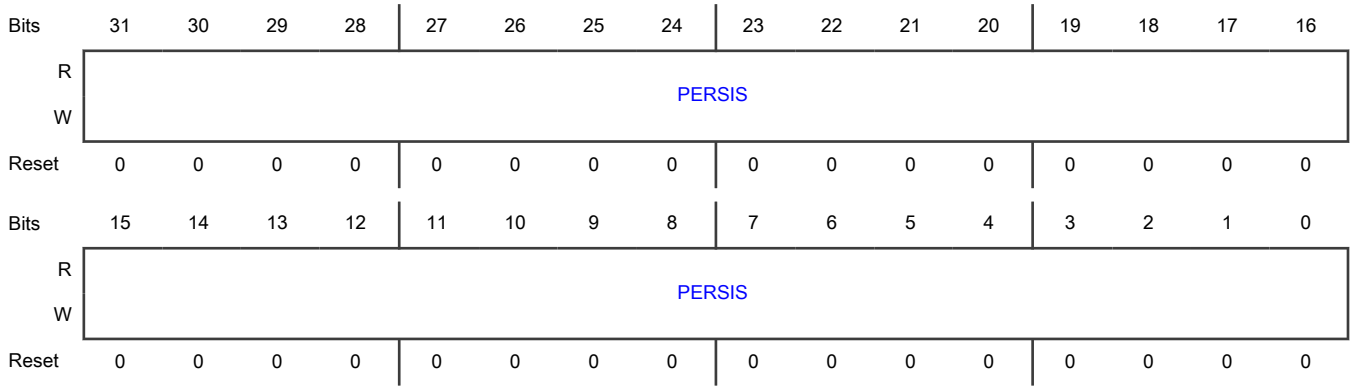
Offset

Register	Offset
PERSISTENT	1Ch

Function

The Persistent Data Storage register (PERSISTENT) provides an application with 32 bits of scratchpad storage for information that needs to persist through resets other than a Power-On Reset (POR).

Diagram



Fields

Field	Function
31-0 PERSIS	Persistent Storage A write value to PERSIS remains unchanged after any reset other than a POR.

12.5.1.9 START Command Register (START)

Offset

Register	Offset
START	20h

Function

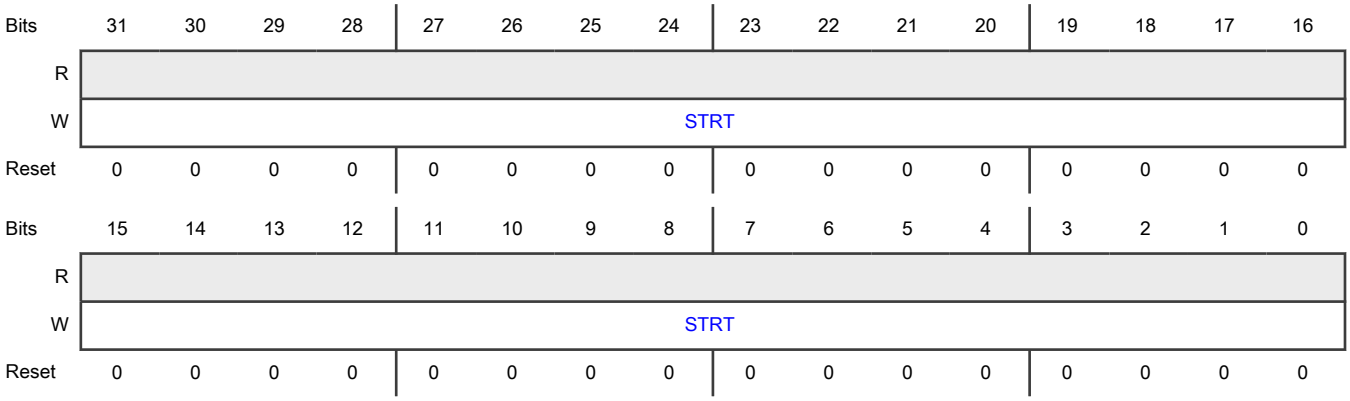
Writing to the Start Command register (START) issues a Start command:

- The value written to STRT is loaded into the Secure Counter as a start value.
- The RELOAD value is loaded into the Instruction Timer.
- The module enters the ACTIVE state in which the Instruction Timer counts down on every clock.

NOTE

Write to START only during the IDLE state, and only after writing to the RELOAD register. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 STRT	Start command The value written to STRT is loaded into the Secure Counter as a start value.

12.5.1.10 STOP Command Register (STOP)

Offset

Register	Offset
STOP	24h

Function

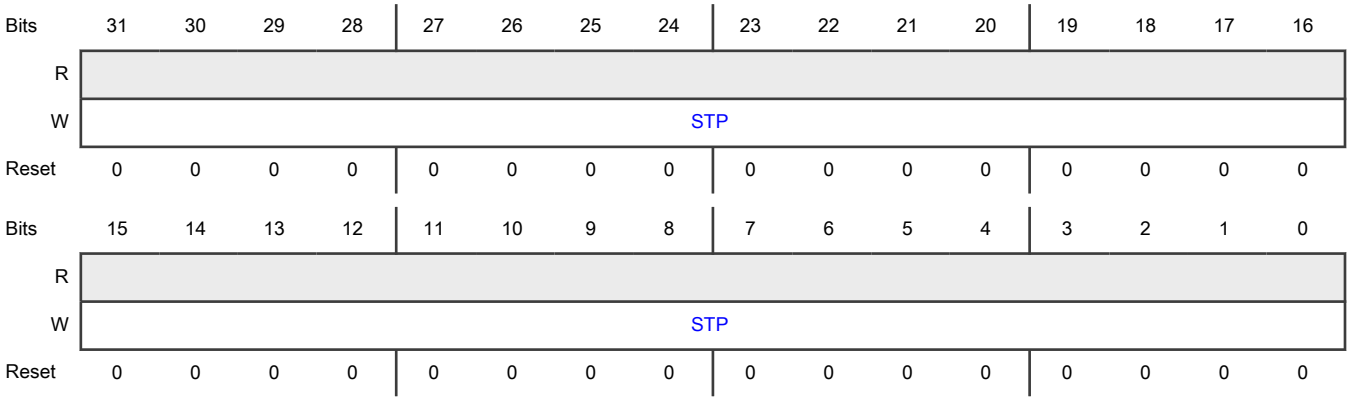
Writing to the Stop Command register (STOP) issues a Stop command:

- The Instruction Timer is stopped.
- The value written to STP is compared with the current value of the Secure Counter.

NOTE

Write to STOP only during the ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 STP	Stop command The value written to STP is compared with the current value of the Secure Counter.

12.5.1.11 RESTART Command Register (RESTART)

Offset

Register	Offset
RESTART	28h

Function

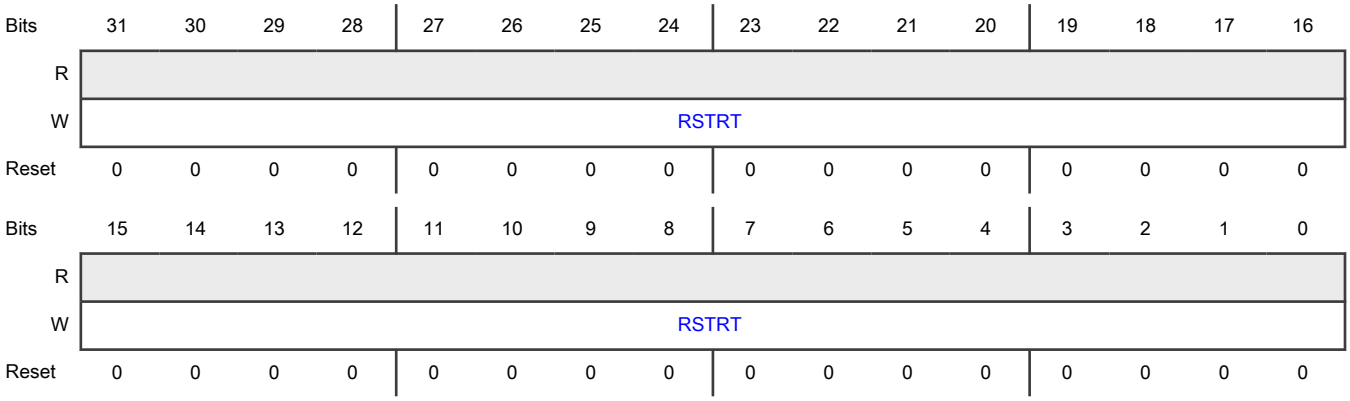
Writing to the Restart Command register (RESTART) issues a Restart command:

- The value written to RSTRT is compared with the current value of the Secure Counter.
- If the values match, the Instruction Timer is reloaded (with the RELOAD value) and starts counting down again.

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 RSTRT	Restart command The value written to RSTRT is compared with the current value of the Secure Counter.

12.5.1.12 ADD Command Register (ADD)

Offset

Register	Offset
ADD	2Ch

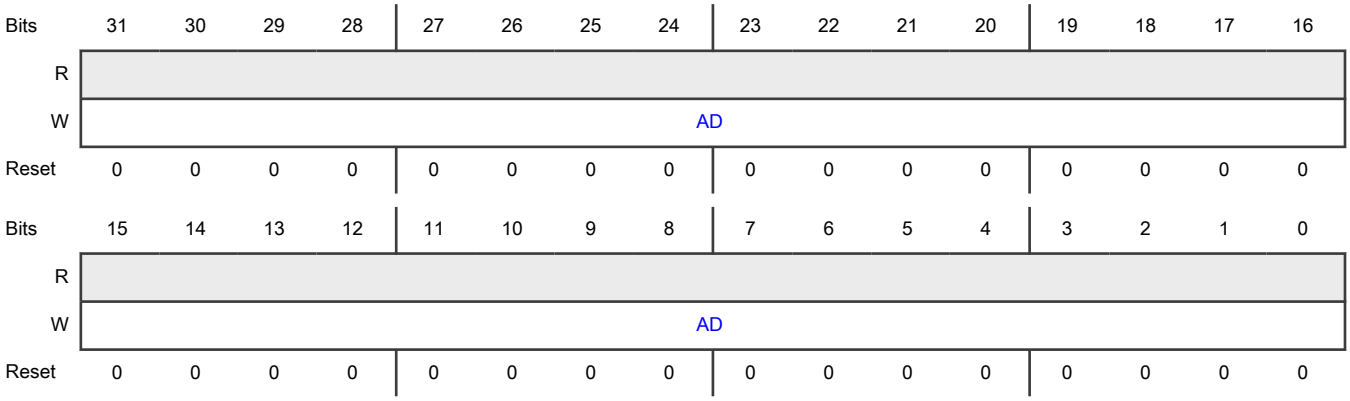
Function

Writing to the Add Command register (ADD) issues an Add command: The value written to AD is added to the current value of the Secure Counter

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 AD	ADD Write Value The value written to AD is added to the current value of the Secure Counter.

12.5.1.13 ADD1 Command Register (ADD1)

Offset

Register	Offset
ADD1	30h

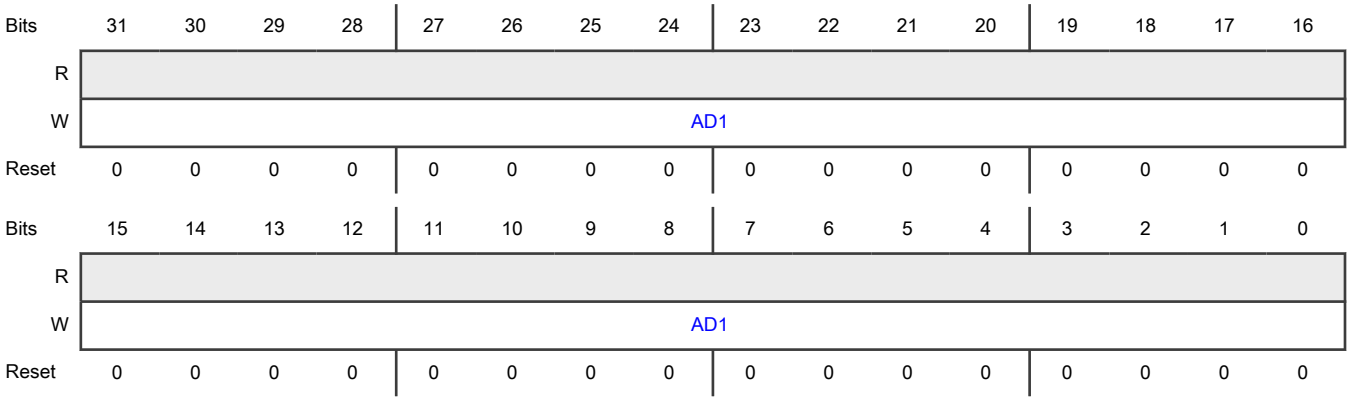
Function

Writing to the ADD1 Command register (ADD1) issues an ADD1 command: The value 1 is added to the current value of the Secure Counter.

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 AD1	ADD 1 Secure Counter is always incremented by 1. The value written to AD1 is ignored.

12.5.1.14 ADD16 Command Register (ADD16)

Offset

Register	Offset
ADD16	34h

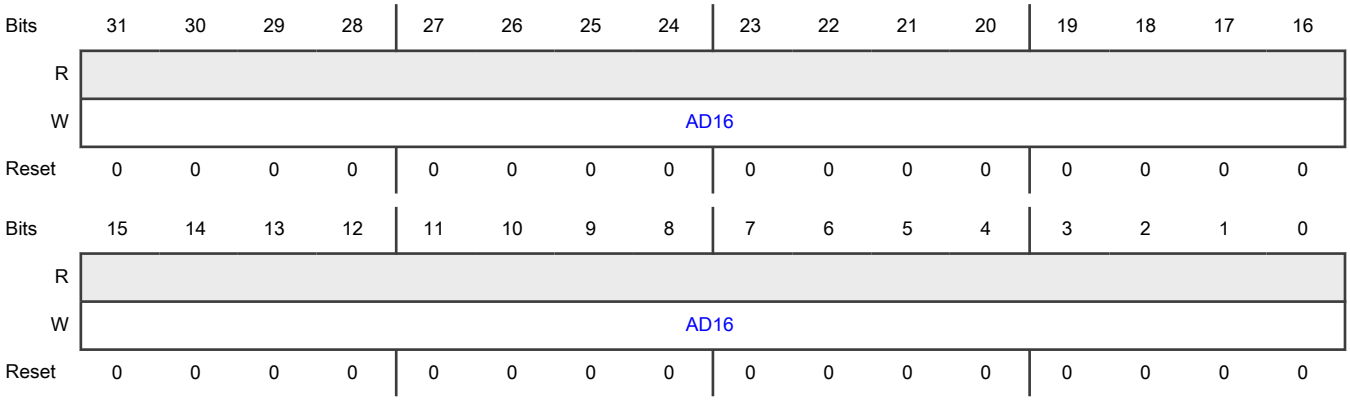
Function

Writing to the ADD16 Command register (ADD16) issues an ADD16 command: The value 16 is added to the current value of the Secure Counter.

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 AD16	ADD 16 Secure Counter is always incremented by 16. The value written to AD16 is ignored.

12.5.1.15 ADD256 Command Register (ADD256)

Offset

Register	Offset
ADD256	38h

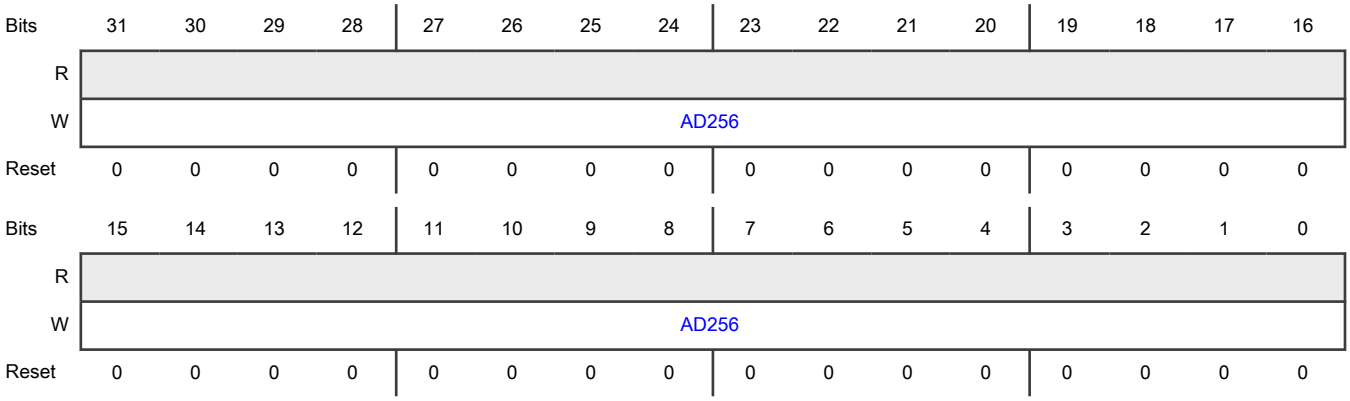
Function

Writing to the ADD256 Command register (ADD256) issues an ADD256 command: The value 256 is added to the current value of the Secure Counter.

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 AD256	ADD 256 Secure Counter is always incremented by 256. The value written to AD256 is ignored.

12.5.1.16 SUB Command Register (SUB)

Offset

Register	Offset
SUB	3Ch

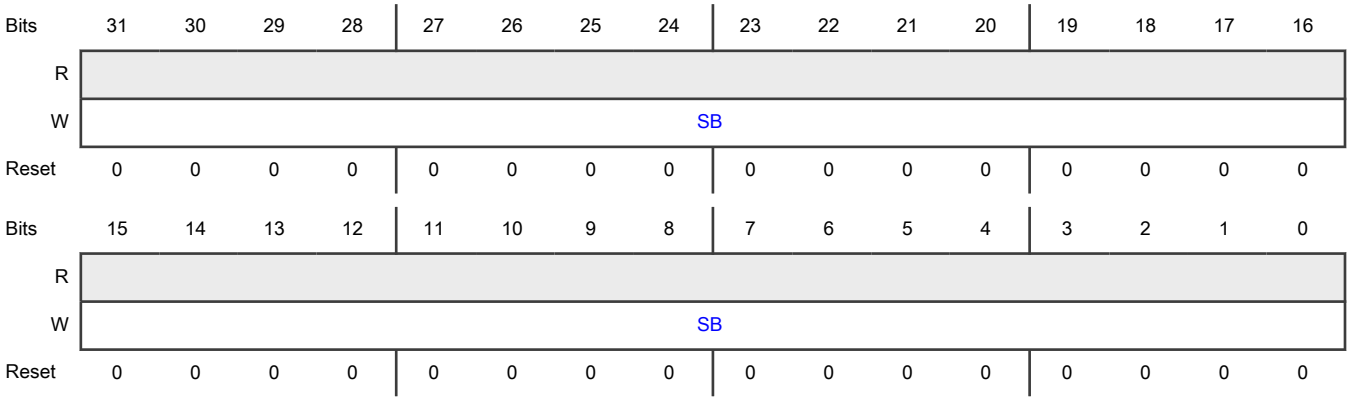
Function

Writing to the SUB Command register (SUB) issues a SUB command: The value written to SB is subtracted from the current value of the Secure Counter.

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 SB	Subtract Write Value The value written to SB is subtracted from the current value of the Secure Counter.

12.5.1.17 SUB1 Command Register (SUB1)

Offset

Register	Offset
SUB1	40h

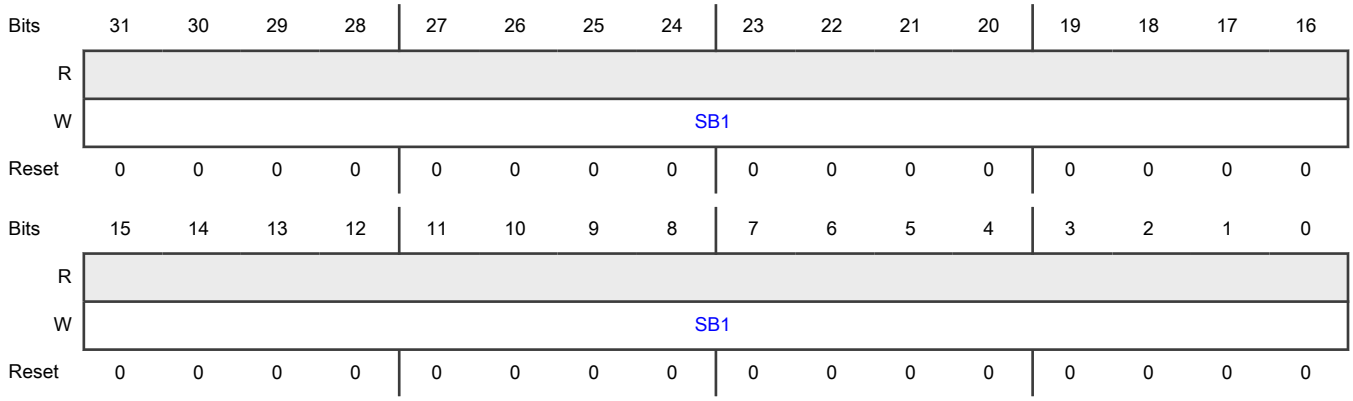
Function

Writing to the SUB1 Command register (SUB1) issues an SUB1 command: The value 1 is subtracted from the current value of the Secure Counter.

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 SB1	Subtract 1 Secure Counter is always decremented by 1. The value written to SB1 is ignored.

12.5.1.18 SUB16 Command Register (SUB16)

Offset

Register	Offset
SUB16	44h

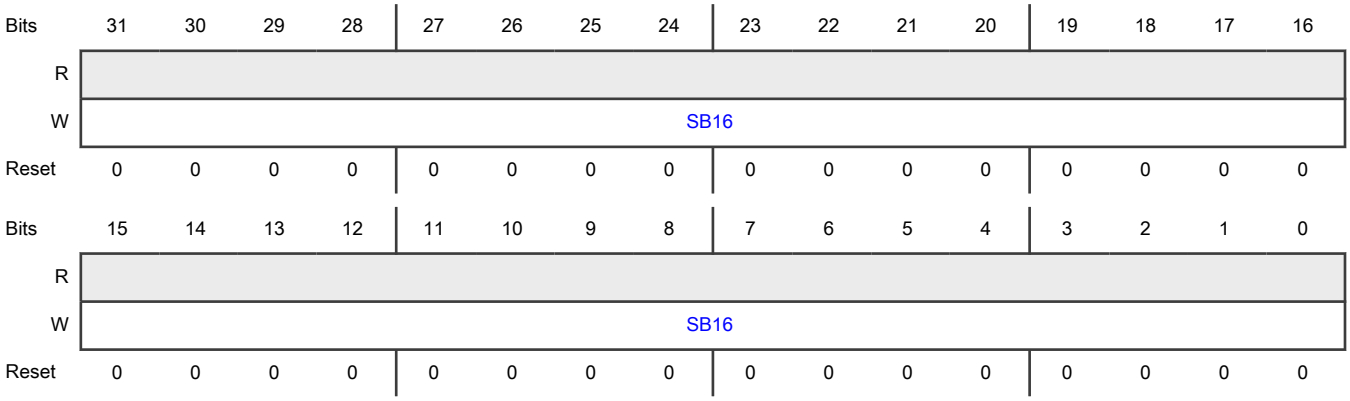
Function

Writing to the SUB16 Command register (SUB16) issues an SUB16 command: The value 16 is subtracted from the current value of the Secure Counter.

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 SB16	Subtract 16 Secure Counter is always decremented by 16. The value written to SB16 is ignored.

12.5.1.19 SUB256 Command Register (SUB256)

Offset

Register	Offset
SUB256	48h

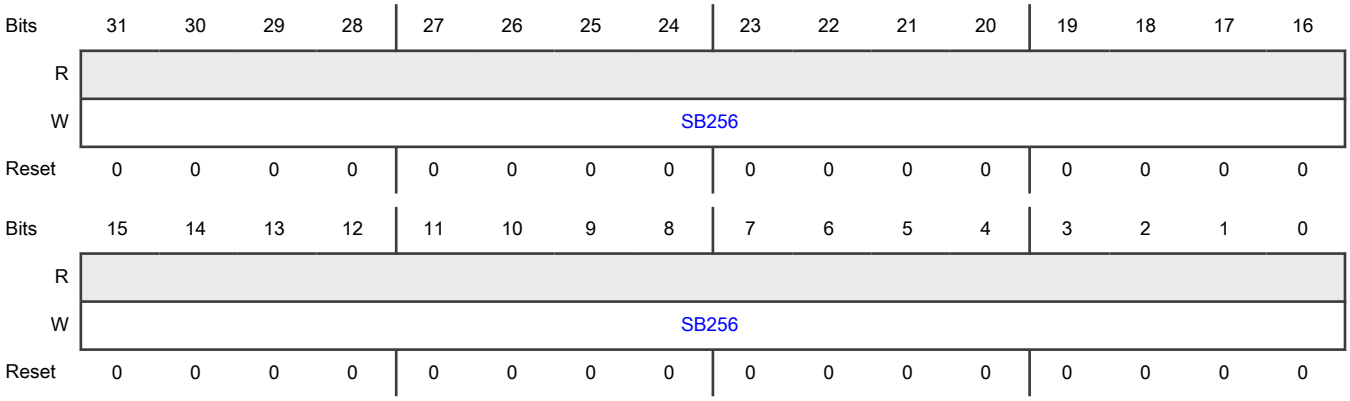
Function

Writing to the SUB256 Command register (SUB256) issues an SUB256 command: The value 256 is subtracted from the current value of the Secure Counter.

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 SB256	Subtract 256 Secure Counter is always decremented by 256. The value written to SB256 is ignored.

12.5.1.20 ASSERT16 Command Register (ASSERT16)

Offset

Register	Offset
ASSERT16	4Ch

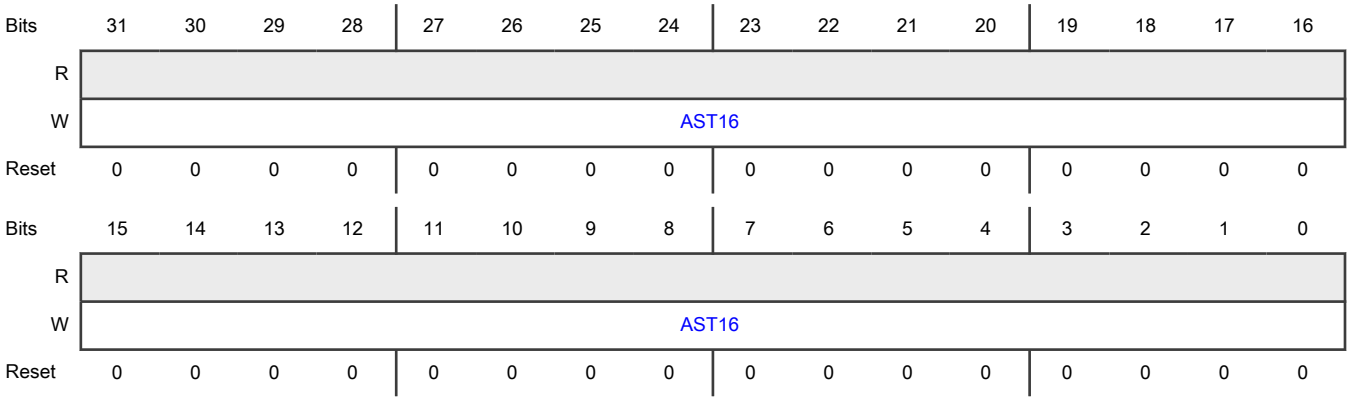
Function

Writing to the ASSERT16 Command register (ASSERT16) issues an ASSERT16 command: The lower 16-bit value written to AST16 is compared with the current lower 16-bit value of the Secure Counter. This command does not affect the Instruction Timer.

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 AST16	ASSERT16 Command The lower 16-bit value written to AST16 is compared with the current lower 16-bit value of the Secure Counter.

Chapter 13
Digital Tamper (TDET)

13.1 Chip-specific TDET information

Table 80. Reference links to related information[1]

Topic	Related module	References
Full description	TDET	TDET
System memory map		System memory map
Clocking		See the chapter "Clocking"
Signal multiplexing	Port control	See the chapter "Signal multiplexing"
System debug		See the chapter "Debug"

1. For all the reference sections and chapters mentioned in this table, refer the Reference Manual.

13.1.1 Module instance

This device has one instance of the TDET module.

13.1.2 Security considerations

For secure applications, NXP recommends that this module is configured for secure and privileged access only to prevent unintentional or malicious modification of the system operation by nonsecure software.

When a tamper event is enabled in Tamper Enable register, and the tamper event is detected by TDET, TDET generates SRAM zeroize request. Hardware zeroizes the PKC SRAM and SRAM A0 during the following reset. SGI key register can be zeroized immediately. When a tamper event is enabled in Tamper Enable register, TFSR is set in Control register, and the tamper event is detected by TDET, TDET can generate warm reset.

13.1.3 On-chip tamper inputs

In addition to detecting tampers for the digital tamper pins, a number of on-chip tamper sources from other modules are routed to the TDET. The table below shows the SR register fields associated with the on-chip tamper sources, the on-chip tamper source, and the flag associated with the tamper source in the module where the tamper originates:

Table 81. On-chip tamper sources

SR register fields	Description	Module	Tamper flag (tamper originating module)
SR[TIF0]	Reserved	-	-
SR[TIF1]	Reserved	-	-
SR[TIF2]	Reserved	-	-
SR[TIF3]	Reserved	-	-
SR[TIF4]	AGDET	SPC	VDD_CORE_GLITCH_DETECT_SC[GLITCH_DETECT_FLAG] or

Table continues on the next page...

Table 81. On-chip tamper sources (continued)

SR register fields	Description	Module	Tamper flag (tamper originating module)
			four bits together
SR[TIF5]	LVD	SPC	VD_STAT[SYSVDD_LVDF] or VD_STAT[COREVDD_LV DF]
SR[TIF6]	HVD	SPC	VD_STAT[SYSVDD_HVDF]
SR[TIF7]	CDOG	CDOG0/1	CDOG0 or CDOG1 reset request
SR[TIF8]	Reserved	-	-
SR[TIF9]	MBC Secure Violation	MBC	MBC access violation
SR[TIF10]	RAM Zeroize Fail	-	SRAM is not zeroized successfully during reset by hardware

13.2 Overview

TDET detects potential on-chip and external device tampering, triggering an interrupt or chip reset.

13.2.1 Block diagram

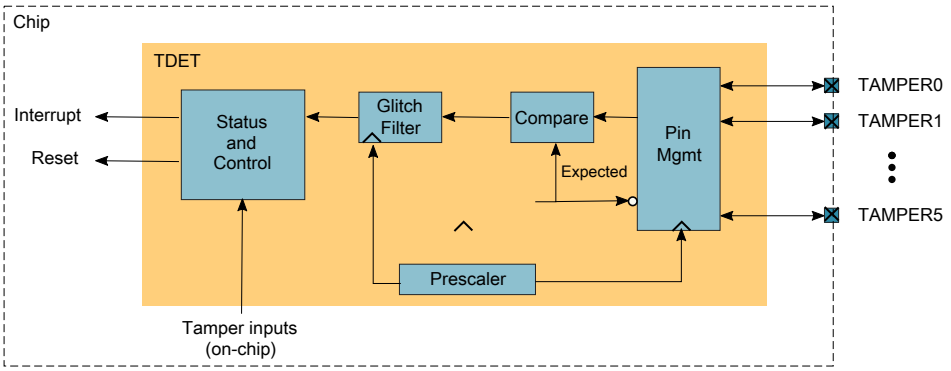


Figure 41. TDET block diagram

13.2.2 Features

TDET features include:

- 6 external tamper pins capable of generating interrupt or tamper event:
  - Configurable polarity and digital glitch filter with optional prescaler
  - Supports software-initiated tamper pin assertion
  - Supports periodic sampling of the tamper pin
- 11 internal tamper sources plus software-initiated tamper capable of generating interrupt or tamper event

- Tamper seconds register uses the time provided by RTC module to record the time of tamper event
- Register protection
  - Lock register requires VBAT POR or software reset to enable write access

13.3 Functional description

13.3.1 External tamper pins

Each pin has configurable polarity.

13.3.1.1 Pull-resistors for input pins

When configured as a tamper input , a pull-resistor can be enabled (using PGFRn[TPE]) on the tamper pin to cause the tamper pin to assert or negate (based on PGFRn[TPS]) if the pin is floating. Disable the pull-resistor if the external tamper circuit is unable to overdrive the pull-resistor. In this case, the external circuit must ensure a floating tamper pin generates a tamper event.

13.3.1.2 Glitch filters

Each tamper pin input has an independently configured glitch filter that can filter out voltage transient widths as small as 30us or as large as 248ms.

Clocking describes the clock source options for the glitch filters.

13.3.1.2.1 Glitch filter operation

The following pseudo-code describes glitch filter operation:

```
If (tamper_pin != expected)
  {
    If (counter == { filter_width, 1'b1 } )
    {
    Filtered_tamper = tamper_pin;
    Counter = 0;
    } else {
    Counter++;
    }
  }
Else if (counter > 0)
{
Counter --;
}
```

The following figures provide examples of glitch filter operation:

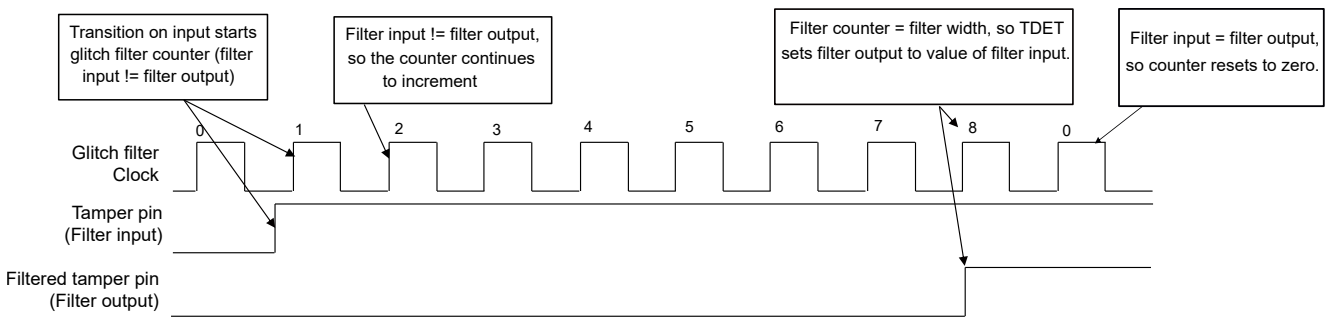


Figure 42. Tamper pin glitch filtering example: no glitch

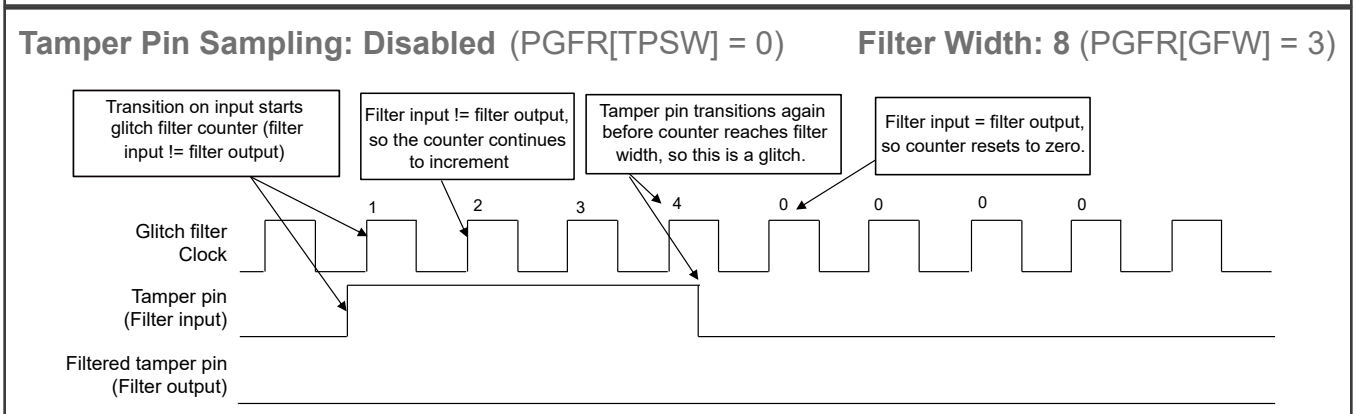


Figure 43. Tamper pin glitch filtering example: glitch

13.3.1.3 Tamper pin sampling

By default, a passive tamper input will continuously monitor the input to detect assertion of the tamper input. Tamper pin sampling can be enabled by configuring the tamper pin sample frequency and the tamper pin sample width for individual tamper pins. When pin sampling is enabled, the tamper pin internal pull resistor and input buffer will only be enabled for some of the time. This can be useful in reducing power consumption of the tamper pin, especially if the pull resistor is configured to assert the tamper pin while external logic is driving the tamper input to the opposite state.

When tamper pin sampling is enabled, the pull resistor is enabled for twice as long as the input buffer. This provides time for the pull resistor to affect the state of the pin before the input buffer is enabled. While the input buffer is disabled, the tamper pin state is internally driven as if the tamper pin is not asserted. If the glitch filter is enabled, the input buffer and pull resistor will remain enabled after the sample window if the glitch filter is still actively resolving the input state (for example, if the glitch filter is configured for a longer width than the sampling width, if the glitch filter detects an asserted tamper then it will keep both the input buffer and pull resistor enabled until the assertion propagates through the glitch filter or the input negates).

The tamper pin sampling can be configured to sample the tamper pin inputs every 8, 32, 128 or 512 cycles. The clock used in determining the frequency is the same clock used by the tamper pin glitch filter (software configurable to either 16.384 kHz or 256 Hz). The width of the tamper pin sampling can be configured for 1 cycle, 2 cycles or 4 cycles (with the pull resistor enabled for twice that length). Configuring the sample width to 4 cycles with a sample frequency of 8 cycles is not recommended as the pull resistor will remain enabled all the time.

13.3.1.4 Passive tamper

Use a passive tamper input pin to monitor an external tamper circuit with a static expected value. Configure the optional pull resistor and glitch filter as needed.

13.3.1.4.1 Configuring a passive tamper

To configure a tamper pin as a passive tamper:

1. If the tamper pin expected value needs to be 1, program its polarity to be inverted (PPR[TPPn] = 1).
2. As needed, select the pull-resistor (PGFRn[TPS]) and enable it (PGFRn[TPE] = 1).
3. As needed, in the corresponding PGFRn:
   - Select the pin-sampling options (TPSF, TPSW).
   - Select the filter-clocking and filter-width options (GFP, GFW).
   - Enable the glitch filter (GFE).
4. Enable tamper detection (TER[TPEn] = 1) and/or interrupts (IER[TPIEn] = 1) for the pin as needed.

13.3.2 On-chip tamper inputs

Each on-chip tamper input source can be configured to generate an interrupt, set the Digital Tamper flag (SR[DTF]), or both. Each on-chip tamper source includes a status flag (SR[TIF*n*]), tamper enable (TER[TIE*n*]), and interrupt enable (IER[TIIE*n*]). The status flag can update whether the tamper or interrupt are enabled or not. Clearing a status flag requires the on-chip tamper source to negate and software to write 1 to the flag.

An enabled tamper source causes SR[DTF] to set if its individual tamper status flag is set. To clear SR[DTF], first clear any enabled tamper input status flags and then write 1 to SR[DTF].

NOTE

The Tamper Acknowledge flag (SR[TAF]) must be set to clear SR[DTF]. This requires a chip reset to occur between SR[DTF] being set due to a tamper event and when it is cleared by software.

13.3.3 Locking registers

If needed for additional security, use the Lock register (LR) to block write accesses to individual registers until the next VBAT POR or TDET software reset (CR[SWR]). Attempted write accesses to a locked register are ignored and do not generate a bus error.

Locking the Control register (CR) disables the software reset. Locking LR itself disables future updates to the Lock register.

13.3.4 Operation in chip power modes

TDET is always powered by the chip's backup power supply (VBAT) and remains operational regardless of the chip's power mode (power-up or power-down).

13.3.5 Clocking

The Digital Tamper clock and prescaler must be enabled (CR[DEN] = 1) for the pin glitch filters to operate. Each glitch filter can be clocked by a 16.384 kHz clock or 256 Hz prescaler clock (PGFRn[GFP]). The Tamper Seconds register (TSR) is clocked by the 0.5 Hz prescaler clock.

13.3.6 Reset

The VBAT supply includes its own analog POR block that generates a power-on-reset signal whenever the VBAT domain is powered-up and initializes all registers to their default state for modules in the VBAT domain.

13.3.6.1 Software reset

Writing 1 to CR[SWR] forces the equivalent of a VBAT POR to the rest of TDET. This software-controlled reset does not affect the CR[SWR] bit. Write 0 to CR[SWR] to release TDET from software reset. The writes can occur back-to-back.

13.3.7 Interrupts

The Digital Tamper interrupt is asserted whenever a status flag and its corresponding interrupt enable bit are both set. It is always asserted on VBAT POR, TDET software reset and when the VBAT power supply is powered down.

13.4 External signals

Table 82. TDET signals

Signal	Description	I/O
TAMPER[5:0]	External tamper input	I/O

13.5 Initialization

As long as the system provides the required clocking in chip power-up and power-down modes, TDET is operational and ready to be configured for specific applications.

After a VBAT POR resets TDET, SR[DTF] is set. Clear SR[DTF] before configuring other TDET registers.

NOTE

Before attempting to clear DTF, confirm that a system reset has occurred (indicated by SR[TAF] = 1) to ensure a secure environment. If DTF = 1 and TAF = 0, first initiate a chip reset before clearing DTF. A chip reset must occur before DTF can be cleared.

13.6 Register definitions

All registers must be accessed using 32-bit writes. Writing to a register protected by the Lock register does not generate a bus error, but the write is ignored.

An attempt to access a TDET register results in a bus error under any of the following power-related conditions:

- VBAT domain is powered down.
- VBAT domain is electrically isolated.
- VBAT POR is asserted.

13.6.1 TDET register descriptions

13.6.1.1 TDET memory map

TDET0 base address: 400E_9000h

Offset	Register	Width (In bits)	Access	Reset value
10h	Control (CR)	32	RW	0000_0000h
14h	Status (SR)	32	RW	0000_0000h
18h	Lock (LR)	32	RW	003F_0BF0h
1Ch	Interrupt Enable (IER)	32	RW	0000_0000h
20h	Tamper Seconds (TSR)	32	RW	0000_0000h
24h	Tamper Enable (TER)	32	RW	0000_0000h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
2Ch	Pin Polarity (PPR)	32	RW	0000_0000h
40h - 54h	Pin Glitch Filter (PGFR0 - PGFR5)	32	RW	0000_0000h

13.6.1.2 Control (CR)

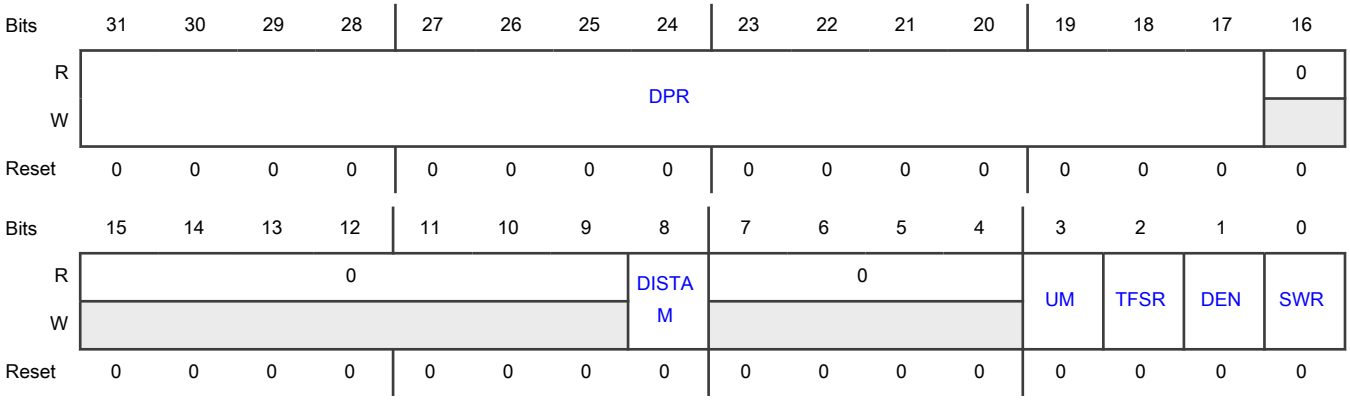
Offset

Register	Offset
CR	10h

Function

Contains the TDET prescaler and configuration options, as well as the software reset control bit.

Diagram



Fields

Field	Function
31-17 DPR	Digital Tamper Prescaler Contains the prescaler. When writing, configures the initial value of the prescaler. When reading, returns the current value of the prescaler. The glitch filters can be clocked by the 16.384 kHz clock or the 256 Hz prescaler clock output (when bit 22 transitions). **NOTE** DPR can be written only when the clocks are disabled (CR[DEN] = 0).
16-9	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	
8 DISTAM	Disable Prescaler On Tamper Allows the 16 KHz clock and prescaler to be automatically disabled after tamper detection (SR[DTF]=1) and until the system acknowledges the tamper (SR[TAF]=1). Disabling the prescaler after detecting a tamper event conserves power and freezes the state of the glitch filters. To ensure a clean transition, the prescaler is disabled at the end of a 0.5 Hz period. 0b - No effect 1b - Automatically disables the prescaler after tamper detection
7-4 —	Reserved
3 UM	Update Mode When no tampering has been detected (SR[DTF]=0), enables software to clear tamper pin flags and on-chip tamper input flags in the Status register (SR[TPFn] and SR[TIFn]) even when SR is locked in the Lock register (LR[SRL]=0). UM applies to TPFn and TIFn that are enabled for interrupts only (IER[TPIEn] and IER[TIIEn] that equal 1 and their corresponding TER enable bits equal 0). 0b - No effect 1b - Allows the clearing of interrupts
2 TFSR	Tamper Force System Reset Enables generating a chip reset when tampering is detected (SR[DTF] = 1 and SR[TAF] = 0). 0b - Do not force chip reset 1b - Force chip reset
1 DEN	Digital Tamper Enable Enables the 16.384 kHz clock and the prescaler that generates the 256 Hz, 32 Hz and 0.5 Hz prescaler clocks. Must be enabled before enabling a glitch filter. Must be disabled only after disabling all glitch filters. 0b - Disables TDET clock and prescaler 1b - Enables TDET clock and prescaler
0 SWR	Software Reset Resets all registers. Exceptions: • CR[SWR] itself is not affected; it is reset by VBAT POR only. 0b - No effect 1b - Perform a software reset

13.6.1.3 Status (SR)

Offset

Register	Offset
SR	14h

Function
Provides the status of all tamper flags, including the global Digital Tamper flag.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0										TPF5	TPF4	TPF3	TPF2	TPF1	TPF0
W											W1C	W1C	W1C	W1C	W1C	W1C
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	0			TIF10	TIF9	TIF8	TIF7	TIF6	TIF5	TIF4	TIF3	TIF2	TIF1	TIF0	TAF	DTF
W				W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-22 —	Reserved
21-16 TPFn	Tamper Pin n Flag Indicates that tamper pin *n* does not equal its expected value and was not filtered by the glitch filter (if enabled). To clear, write 1 to the flag after its tamper pin equals its expected value and the glitch filter output has updated. 0b - Pin tamper not detected 1b - Pin tamper detected
15-13 —	Reserved
12-2 TIFn	Tamper Input n Flag Indicates that on-chip tamper input *n* has asserted. To clear, write 1 to a flag after the on-chip source negates. 0b - On-chip tamper not detected 1b - On-chip tamper detected

Table continues on the next page...

Table continued from the previous page...

Field	Function
1 TAF	Tamper Acknowledge Flag Indicates a chip reset has occurred after the Digital Tamper Flag (SR[DTF]) was set. The chip acknowledges the tamper by resetting. TAF is cleared on VBAT POR, software reset or by software writing 1 to this flag whenever SR[DTF] is clear. 0b - Digital Tamper Flag (SR[DTF]) is clear or chip reset has not occurred after Digital Tamper Flag (SR[DTF]) was set. 1b - Chip reset has occurred after Digital Tamper Flag (SR[DTF]) was set.
0 DTF	Digital Tamper Flag Indicates tampering has been detected. Sets on VBAT POR, software reset, a write to the tamper seconds register or whenever an enabled tamper flag is set. Can be cleared by software by writing 1 to the flag provided the Tamper Acknowledge flag is set. 0b - TDET tampering not detected 1b - TDET tampering detected

13.6.1.4 Lock (LR)

Offset

Register	Offset
LR	18h

Function

Clearing individual fields locks the corresponding register. Once cleared, fields can be set again (registers unlocked) only by VBAT POR or software reset.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0										GFL5	GFL4	GFL3	GFL2	GFL1	GFL0
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0				PPL	0	TEL	TSL	IEL	LRL	SRL	CRL	Reserved			Reserved
W																
Reset	0	0	0	0	1	0	1	1	1	1	1	1	0	0	0	0

Fields

Field	Function
31-22 —	Reserved
21-16 GFLn	Glitch Filter Lock Locks the corresponding Pin Glitch Filter *n* register (PGFRn). 0b - Locked and writes are ignored 1b - Not locked and writes complete as normal
15-12 —	Reserved
11 PPL	Pin Polarity Lock Locks the Pin Polarity register (PPR). 0b - Locked and writes are ignored 1b - Not locked and writes complete as normal
10 —	Reserved
9 TEL	Tamper Enable Lock Locks the Tamper Enable register (TER). 0b - Locked and writes are ignored 1b - Not locked and writes complete as normal
8 TSL	Tamper Seconds Lock Locks the Tamper Seconds register (TSR). 0b - Locked and writes are ignored 1b - Not locked and writes complete as normal
7 IEL	Interrupt Enable Lock Locks the Interrupt Enable register (IER). 0b - Locked and writes are ignored 1b - Not locked and writes complete as normal
6 LRL	Lock Register Lock Locks the Lock register (LR). 0b - Locked and writes are ignored 1b - Not locked and writes complete as normal
5	Status Register Lock

Table continues on the next page...

Table continued from the previous page...

Field	Function
SRL	Locks the Status register (SR). 0b - Locked and writes are ignored 1b - Not locked and writes complete as normal
4 CRL	Control Register Lock Locks the Control register (CR). 0b - Locked and writes are ignored 1b - Not locked and writes complete as normal
3-1 —	Reserved
0 —	Reserved

13.6.1.5 Interrupt Enable (IER)

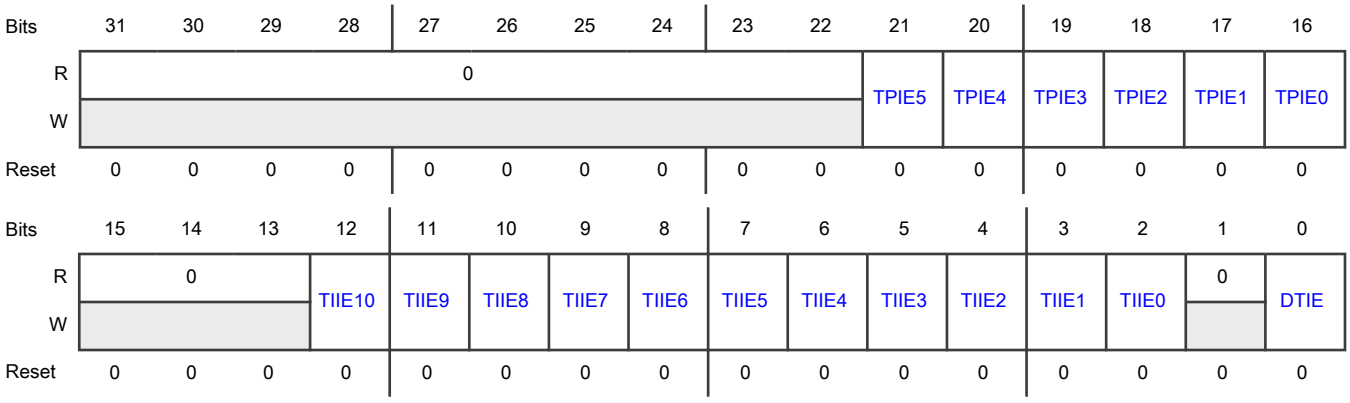
Offset

Register	Offset
IER	1Ch

Function

Enables interrupts for the Digital Tamper flag and the individual tamper pins or on-chip tamper input flags.

Diagram



Fields

Field	Function
31-22 —	Reserved
21-16 TPIEn	Tamper Pin n Interrupt Enable Enables interrupts when the corresponding tamper pin *n* flag (SR[TPFn]) is set. 0b - Disables 1b - Enables
15-13 —	Reserved
12-2 TIIEn	Tamper Input n Interrupt Enable Enables interrupts when the corresponding on-chip tamper input *n* flag (SR[TIFn]) is set. 0b - Disables 1b - Enables
1 —	Reserved
0 DTIE	Digital Tamper Interrupt Enable Enables interrupts when the Digital Tamper flag (SR[DTF]) is set. 0b - Disables 1b - Enables

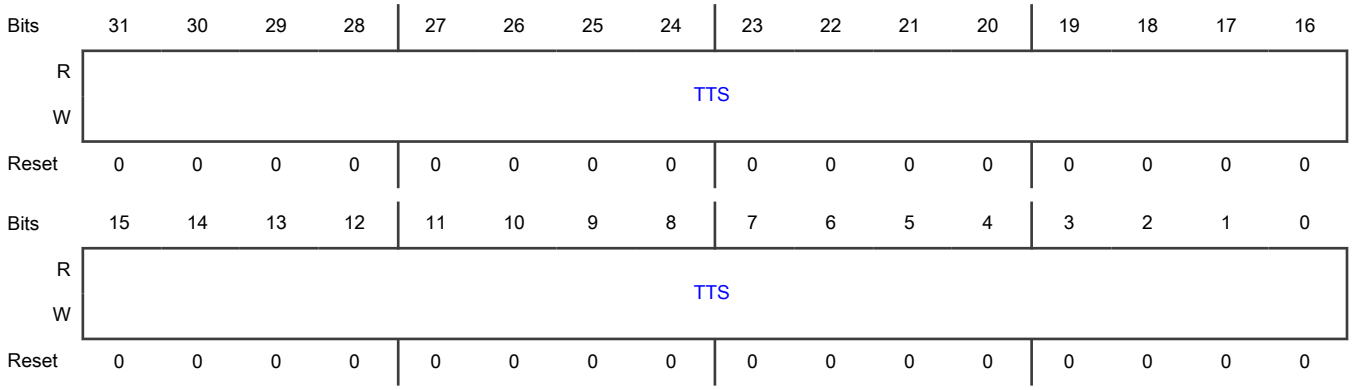
13.6.1.6 Tamper Seconds (TSR)

Offset

Register	Offset
TSR	20h

Function
Contains the time in seconds corresponding to when the Digital Tamper flag (SR[DTF]) was set.

Diagram



Fields

Field	Function
31-0 TTS	Tamper Time Seconds Reading this register returns the time in seconds at which SR[DTF] was set. Returns zero when SR[DTF] is clear. Writing to TSR[TTS] is considered tampering and sets SR[DTF].

13.6.1.7 Tamper Enable (TER)

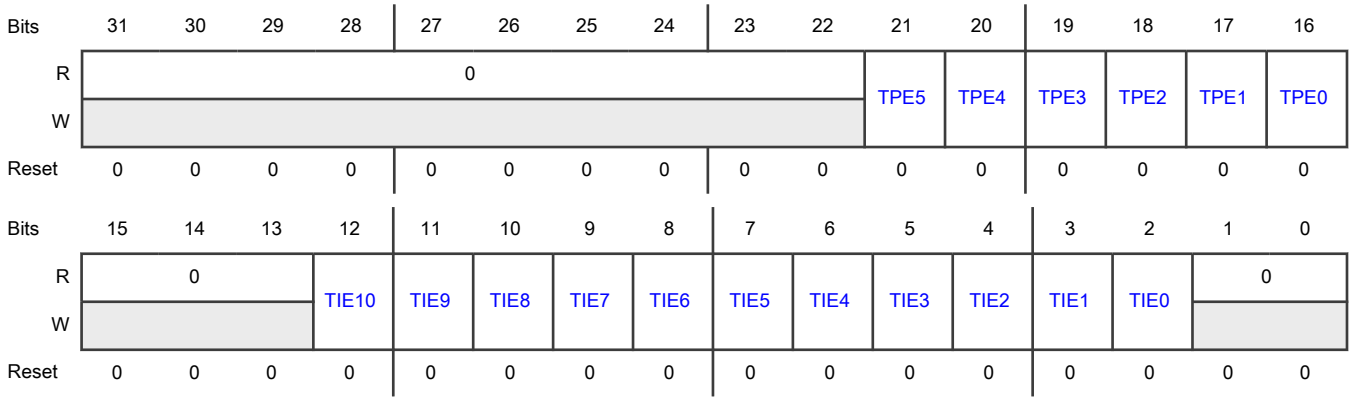
Offset

Register	Offset
TER	24h

Function

Enables tamper detection for tamper pins and on-chip tamper inputs.

Diagram



Fields

Field	Function
31-22 —	Reserved
21-16 TPEn	Tamper Pin Enable Enables tamper detection when the corresponding tamper pin *n* flag (SR[TPFn]) is set. 0b - Disables 1b - Enables
15-13 —	Reserved
12-2 TIEn	Tamper Input Enable Enables tamper detection when the corresponding on-chip tamper input *n* flag (SR[TIFn]) is set. 0b - Disables 1b - Enables
1-0 —	Reserved

13.6.1.8 Pin Polarity (PPR)

Offset

Register	Offset
PPR	2Ch

Function

Configures the pin polarity of tamper pins and provides read access to the current values of tamper pin input data.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0										TPID5	TPID4	TPID3	TPID2	TPID1	TPID0
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	0										TPP5	TPP4	TPP3	TPP2	TPP1	TPP0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-22 —	Reserved
21-16 TPIDn	Tamper Pin n Input Data Contains the current value of the tamper pin *n* input data before the glitch filter. 0b - Zero 1b - One
15-6 —	Reserved
5-0 TPPn	Tamper Pin n Polarity Configures the polarity of the tamper pin *n* expected value. 0b - Not inverted 1b - Inverted

13.6.1.9 Pin Glitch Filter (PGFR0 - PGFR5)

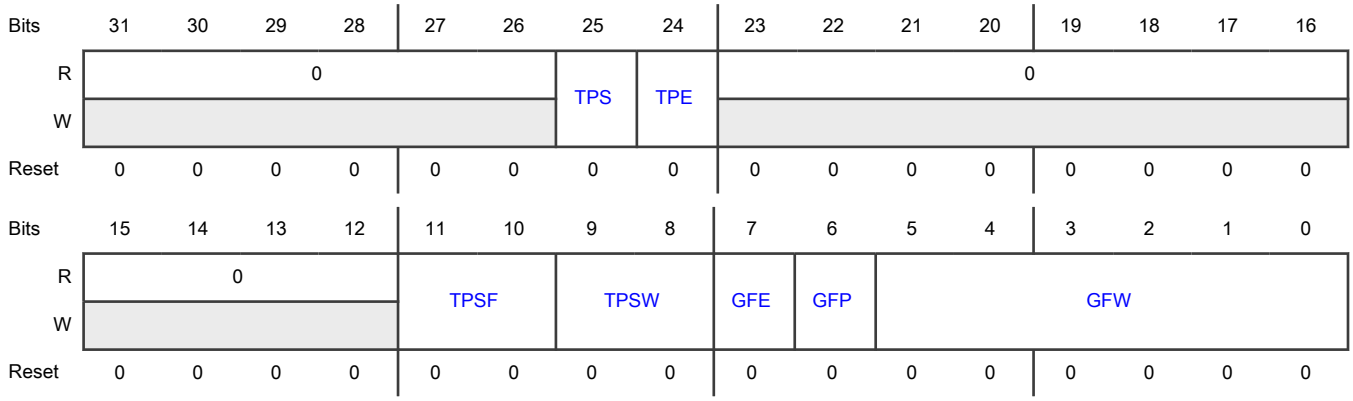
Offset

Register	Offset
PGFR0	40h
PGFR1	44h
PGFR2	48h
PGFR3	4Ch
PGFR4	50h
PGFR5	54h

Function

Configures the glitch filter, pin pull and sampling operation, as well as the expected value, for the corresponding tamper pin.

Diagram



Fields

Field	Function
31-26 —	Reserved
25 TPS	Tamper Pull Select Configures the direction of the tamper pin internal pull resistor such that the tamper pull resistor always asserts or negates the tamper pin. The tamper pull direction is therefore also dependent on the Tamper Pin Polarity (PPR[TPPn]) configuration. 0b - Asserts 1b - Negates
24 TPE	Tamper Pull Enable Enables the pull resistor on the tamper pin. 0b - Disables 1b - Enables
23-16 —	Reserved
15-12 —	Reserved
11-10 TPSF	Tamper Pin Sample Frequency Configures the tamper pin sampling frequency (measured in number of glitch filter clock cycles). When tamper pin sampling is enabled, the tamper pin pull enable and input buffer enable periodically assert at a frequency determined by TPSF. The tamper pin pull enable and input buffer enable are always asserted whenever the glitch filter is actively filtering the input. The number of cycles refers to the number of cycles of the selected glitch filter clock (either 16.384 kHz or 256 Hz).

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Every 8 cycles 01b - Every 32 cycles 10b - Every 128 cycles 11b - Every 512 cycles
9-8 TPSW	Tamper Pin Sample Width Configures the tamper pin sampling width (measured in number of glitch filter clock cycles) during which the pin pull enable and input buffer enable are activated, or disables pin sampling. When tamper pin sampling is disabled, the tamper pin monitors the input continuously. When tamper pin sampling is enabled, the tamper pin pull enable and input buffer enable periodically assert for time widths determined by TPSW. The tamper pin pull enable and input buffer enable are always asserted whenever the glitch filter is actively filtering the input. The number of cycles refers to the number of cycles of the selected glitch filter clock (either 16.384 kHz or 256 Hz). 00b - Continuous monitoring, pin sampling disabled 01b - 2 cycles for pull enable and 1 cycle for input buffer enable 10b - 4 cycles for pull enable and 2 cycles for input buffer enable 11b - 8 cycles for pull enable and 4 cycles for input buffer enable
7 GFE	Glitch Filter Enable Enables the glitch filter. **NOTE** Do not change the glitch filter enable when the corresponding tamper pin is enabled. 0b - Bypasses 1b - Enables
6 GFP	Glitch Filter Prescaler Selects the clock source for the glitch filter. **NOTE** Do not change the prescaler when the glitch filter is enabled. 0b - 256 Hz prescaler clock 1b - 16.384 kHz clock
5-0 GFW	Glitch Filter Width Configures the number of clock edges during which the input must remain stable in order to pass through the glitch filter. The number of clock edges is (GFW + 1) * 2, supporting a configuration of between 2 and 128 clock edges.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	NOTE Do not change the glitch filter width when the glitch filter is enabled.

Chapter 14
GLIKEY

14.1 Chip-specific GLIKEY information

Table 83. Reference links to related information[1]

Topic	Related module	References
Full description	GLIKEY	GLIKEY
System memory map		System memory map
Clocking		See the chapter "Clocking"
Signal multiplexing	Port control	See the chapter "Signal multiplexing"
System debug		See the chapter "Debug"

1. For all the reference sections and chapters mentioned in this table, refer the Reference Manual.

14.1.1 Module instance

This device has one instance of the GLIKEY module.

14.1.2 Security considerations

For secure applications, NXP recommends that this module is configured for secure and privileged access only to prevent unintentional or malicious modification of the system operation by nonsecure software.

14.1.3 GLIKEY control list

Below registers are controlled by GLIKEY:

Table 84. GLIKEY control list

GLIKEY write enable	GLIKEY reset	Module	Register
wr_en_out[1]	rst[1]	SYSCON	DEBUG_FEATURES_DP
wr_en_out[2]	N/A	SYSCON	SRAM_XEN_DP
wr_en_out[3]	N/A	SYSCON	MSFCFG
wr_en_out[15]	N/A	MBC	All MBC register[1]

1. See the *MBC* chapter in *MCX A255, and A256 Reference Manual*.

14.2 Terms and definitions

Acronym	Explanation
FSM	Finite State Machine
SSR	Security Sensitive Register
SW	Software

14.3 Overview

Glikey IP provides a mechanism to safely access security-sensitive registers. The write-enable and reset's of these registers are controlled via GLIKEY.

14.3.1 Block diagram

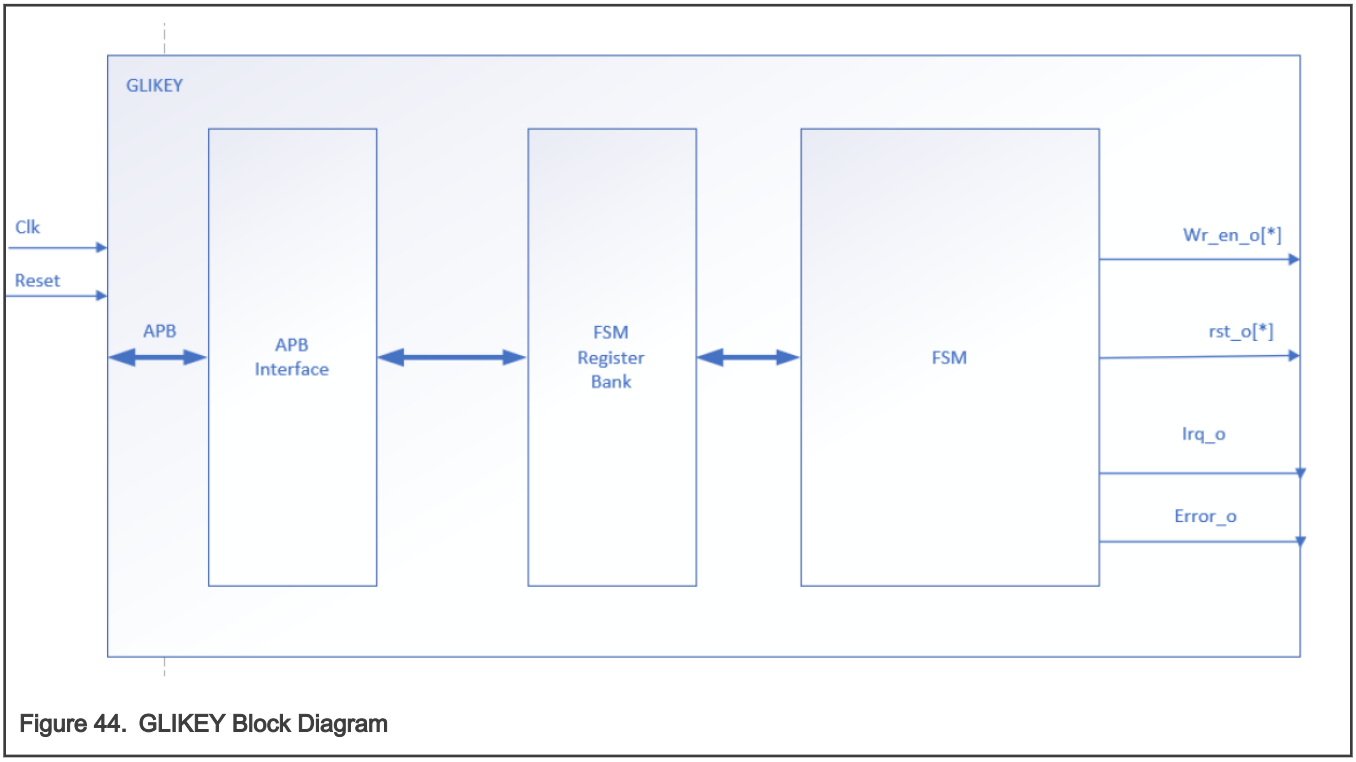


Figure 44. GLIKEY Block Diagram

14.3.2 Features

- Integrity protection of security-sensitive registers during write and at rest against power glitch attacks
- Integrity protection of security-sensitive registers at rest against privilege escalation
- Error Management
  - Dedicated Interrupt
  - Dedicated integrity protection

14.4 Configuration

GLIKEY can be configured in following ways:

- FSM to include 4 steps as mentioned in Figure 45

14.5 Architecture description

14.5.1 Functional description

GLIKEY controls the write-enables and resets for the SSR's through FSM. Software has to follow strict procedure to enable the write-enables and resets for SSR.

GLIKEY safe guards against attacks by constantly monitoring the state which the FSM is in and the expected SW procedure for the current state. If fault is found, then an interrupt is raised and the FSM goes to secure state. Procedure cannot continue until SW resets the GLIKEY.

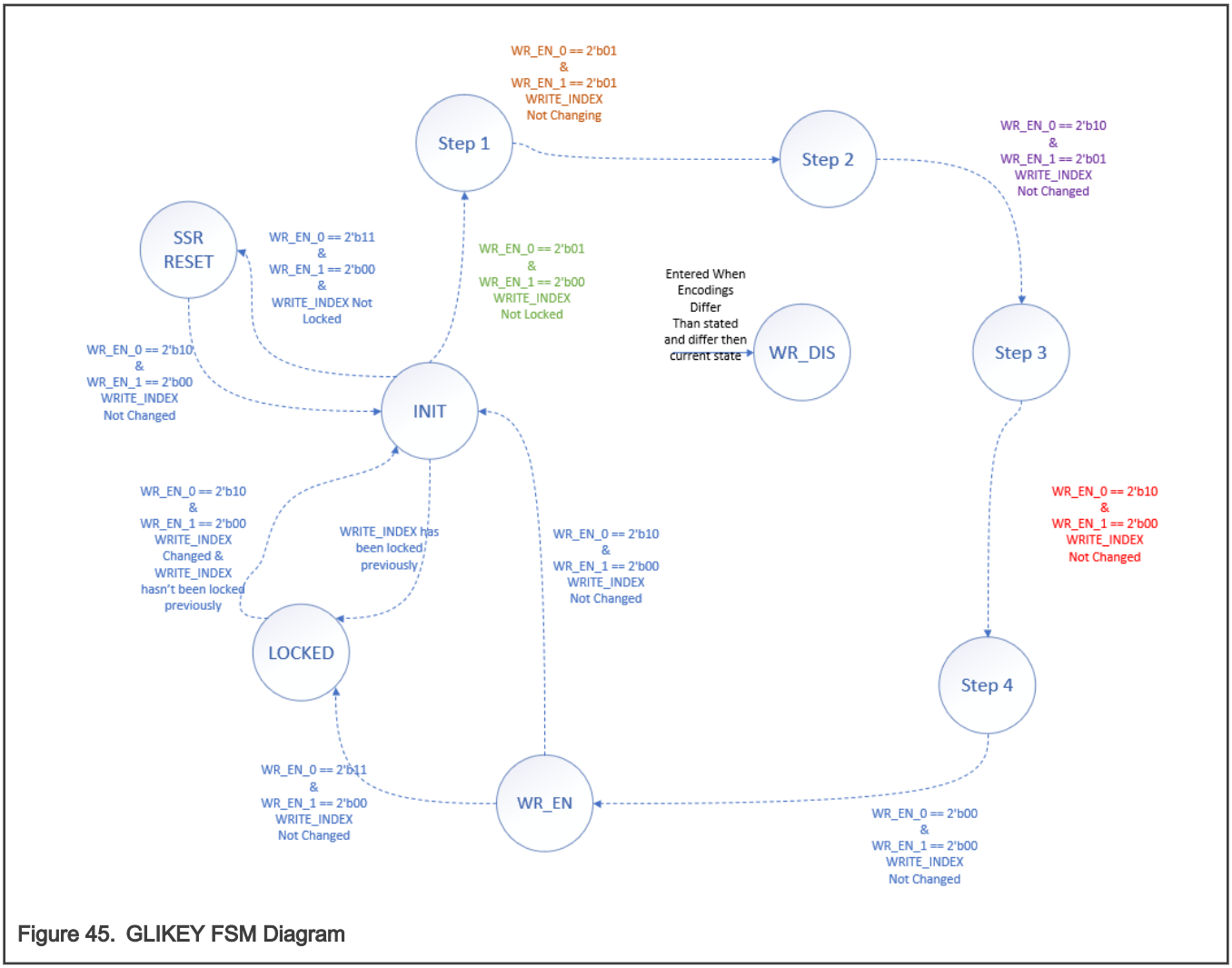


Figure 45. GLIKEY FSM Diagram

14.5.1.1 Progressing Through the FSM

Progressing through the FSM is done through writing to SFR registers: CTRL_0[WR_EN_0] and CTRL_1[WR_EN_1].

Any misconfiguration will lead to the FSM entering the WR_DISABLED state that would require a reset, either a soft reset or a hard reset.

The current state of FSM is read out by reading STATUS[FSM_STATE].

The sequence of progressing through the FSM can be seen in STATUS[FSM_STATE]. If the integrity of FSM encodings is compromised, then the error_o will be triggered.

14.5.1.2 Initial State

Upon reset (either triggered through hardware or software) FSM will enter the INIT state.

The reset value for the CTRL_0[WRITE_INDEX] is 0x0. If software reset is triggered and CTRL_0[WRITE_INDEX] was previously locked, then the FSM enters into the LOCKED state. To progress, the software needs to update the CTRL_0[WRITE_INDEX] to an index that is currently not locked.

In the INIT state if CTRL_0[WRITE_INDEX] is written that is currently LOCKED, then the FSM will enter the LOCKED state irrespective of CTRL_0[WR_EN_0] and CTRL_1[WR_EN_1].

For example, when in the INIT state, one write to the CTRL_0 happens that updates the CTRL_0[WR_EN_0] = 2'b01 and the CTRL_0[WRITE_INDEX] to a locked index, the FSM will enter the LOCKED state. In this case, software must update the CTRL_0[WR_EN_0] = 2'b10 and the CTRL_0[WRITE_INDEX] to an unlocked value to move back into the INIT state.

For normal operation while in the INIT state and CTRL_0[WRITE_INDEX] is written that is not locked and the CTRL_0[WR_EN_0] is set to 2'b01 or 2'b11 the FSM will either enter STEP1 or SSR_RESET and continue normal operation.

14.5.1.3 WR_DIS State

WR_DISABLED state is considered a secured state.

Any misconfiguration of the FSM will lead to this state. Incorrect addressing of the index through CTRL_0[WRITE_INDEX] will also lead to this state.

The FSM monitors the current state of which it is in, if the control signals change and do not match the expected next state encodings the error state is entered.

To leave the WR_DISABLED state, software must perform a soft (Writing one to CTRL_0[SFT_RST]) or hard reset of GLIKEY.

The NON-Secure state is considered a state in which the SSR can be accessed (this is the WR_EN state). This access is allowed via the write-enable outputs of GLIKEY. GLIKEY can enter only the NON-Secure state when the full software procedure has occurred.

Trigger a write-enable out

The write-enable output for any given index is only asserted when the FSM is in WR_ENABLED state.

When the FSM moves out of this state then it is de-asserted. The output wr_en_o[*] outputs are not registered and are driven directly by FSM.

Upon entering the WR_DIS state, all bits of the output rst_o will assert to high causing reset of all SSRs

14.5.1.4 Trigger a Reset output

The reset output for any given index is asserted while the FSM is in SSR_RESET state.

When the FSM moves out of this state, then it is de-asserted.

The output rst_o[*] outputs are not registered and are driven directly by the FSM.

14.5.1.5 Lock SSR access

After accessing the WR_ENABLE state for any given index, further access to this index can be locked. This causes the GLIKEY to lock access for this. If addressed again, FSM jumps to the LOCKED state and cannot trigger a write-enable or reset for this index.

The locked indexes can only be reset by a hardware reset. The lock registers are protected against integrity errors with replication. In case the lock and the inverse lock registers are not matching the error_o port is triggered.

Example:

Assuming software is in the WR_EN state with WRITE_INDEX =1, software writes 2'b11 to CTRL_0[WR_EN_0] and enters the LOCKED State. This Index is now considered locked. To enter back into INIT state, software writes 2'b10 to CTRL_0[WR_EN_0] and the WRITE_INDEX is updated to a WRITE_INDEX not previously locked. If software updates WRITE_INDEX that is previously locked, then the software remains in LOCKED STATE.

This means when all the indexes are locked, software remains in the LOCKED state upon entering the last index that is locked.

To read the status of WRITE_INDEX that are locked, software needs to write the corresponding WRITE_INDEX number to the CTRL_0[WRITE_INDEX] to know the lock status. Software then reads STATUS[LOCK_STATUS] to know the status. 1 = The index is locked, 0 = The index is not locked.

14.5.1.6 Reset FSM

Software has the ability to reset the FSM. Writing one to CTRL_0[SFT_RST] brings the FSM into INIT state. This can be triggered while in any state.

Resetting the FSM will cause the current procedure to be restarted. Resetting the FSM will not reset the indexes that have been locked previously. When all WRITE_INDEX's are locked then the FSM will stay in the LOCKED State and cannot be moved from.

Triggering the CTRL_0[SFT_RST] will also reset the following registers to their reset values:

- CTRL_0[WR_EN_0]
- CTRL_1[WR_EN_1]
- CTRL_0[WRITE_INDEX]

CTRL_0[SFT_RST] takes priority if CTRL_0[WRITE_INDEX] and CTRL_0[WR_EN_0] are written in the same bus access, resulting in these being reset.

14.5.1.7 Disable GLIKEY

GLIKEY can be disabled by writing the value 4'h5 to CTRL_1[SFR_LOCK]. Only hardware reset can re-enable GLIKEY.

Once the GLIKEY is disabled, software has no write permissions to the SFR's only read permissions.

GLIKEY cannot perform any accesses to the security sensitive registers.

14.5.1.8 Interrupt Description

There are 3 interrupt sources, as listed below.

Table 85. Interrupt Sources

Interrupt Source	Description	Register
FSM Interrupt	Upon entering the WR_DIS state this interrupt is raised	STATUS[ERROR_STATUS][0]
WRITE_INDEX out of bounds	If a WRITE_INDEX is addressed that is greater than the number of addressable indexes configured	STATUS[ERROR_STATUS][1]
READ_INDEX out of bounds	If a READ_INDEX is addressed that is greater than the number of addressable indexes configured	STATUS[ERROR_STATUS][2]

Interrupts can be enabled or disabled through INTR_CTRL[INT_EN].

Interrupts can be cleared by writing 1 to INTR_CTRL[INT_CLR].

Interrupts can be set by writing 1 to INTR_CTRL[INT_SET].

Software can monitor the status of the interrupt sources through STATUS[ERROR_STATUS]. Clearing the interrupt will not clear the STATUS[ERROR_STATUS] registers.

Table 86. Interrupt Clearing Methods

Error Status Register	Description

Table continues on the next page...

Table 86. Interrupt Clearing Methods (continued)

STATUS[ERROR_STATUS][0]	Soft or hard reset
STATUS[ERROR_STATUS][1]	Updating WRITE_INDEX to less than number of addressable indexes
STATUS[ERROR_STATUS][2]	Updating READ_INDEX to less than number of addressable indexes

14.5.1.9 Error description

When error_o is high this inidicates that an internal fault has occured.

NOTE

This signal is not driven by any of the interrupt sources

Sources of internal faults

- Duplication Registers

  The internal registers of the locked reigsters are duplicated. These are constantly monitor and when not equal, raises error.
- Entering into the WR_DIS State

  When the FSM enters the WR_DIS state, this will raise the error.
- Attack on the FSM State register

  When the FSM state is not equal to a defined state, this will raise the error.

14.5.2 Registers description (SFRs)

The block has only SFR space that can be accessed directly from the <APB/AHB> slave

14.5.2.1 GLIKEY register descriptions

14.5.2.1.1 GLIKEY memory map

GLIKEY0.GLIKEY base address: 4009_1D00h

Offset	Register	Width (In bits)	Access	Reset value
0h	Control Register 0 SFR (CTRL_0)	32	RW	See section
4h	Control Register 1 SFR (CTRL_1)	32	RW	See section
8h	Interrupt Control (INTR_CTRL)	32	RW	See section
Ch	Status (STATUS)	32	R	See section
FCh	IP Version (VERSION)	32	R	See section

14.5.2.1.2 Control Register 0 SFR (CTRL_0)

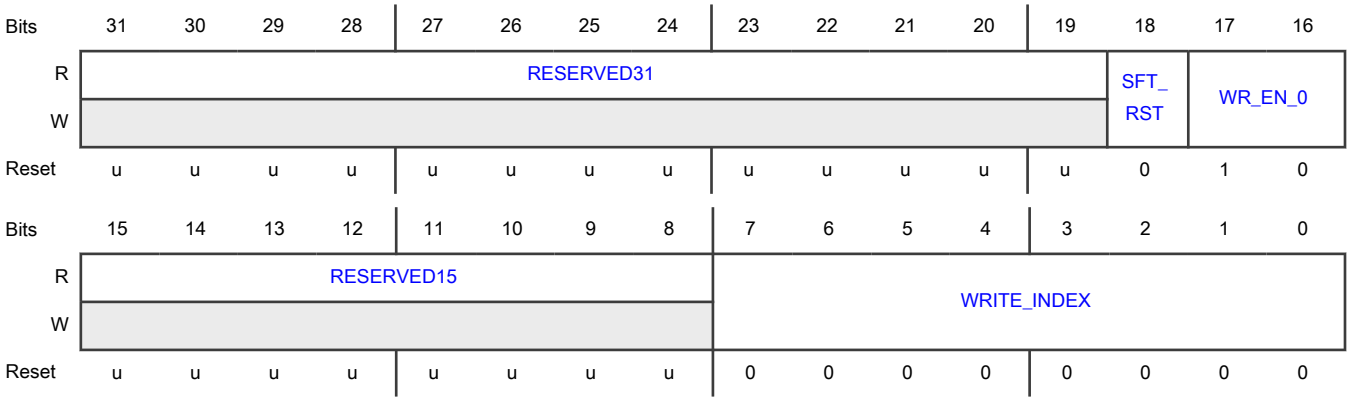
Offset

Register	Offset
CTRL_0	0h

Function

Control Register 0 SFR controls the flow of FSM

Diagram



Fields

Field	Function
31-19 RESERVED31	Reserved for Future Use
18 SFT_RST	Soft reset for the core reset (SFR configuration will be preseved).This register reads as 0 Use this field to bring the logic into a known state. **NOTE** This field is volatile. 0b - No effect 1b - Triggers the soft reset
17-16 WR_EN_0	Write Enable 0 Controls the state of FSM. Use these bits with Write Enable 1. **NOTE** This field is volatile.
15-8	Reserved for Future Use

Table continues on the next page...

Table continued from the previous page...

Field	Function
RESERVED15	
7-0 WRITE_INDEX	Write Index Targets given index for SSR. **NOTE** This field is volatile.

14.5.2.1.3 Control Register 1 SFR (CTRL_1)

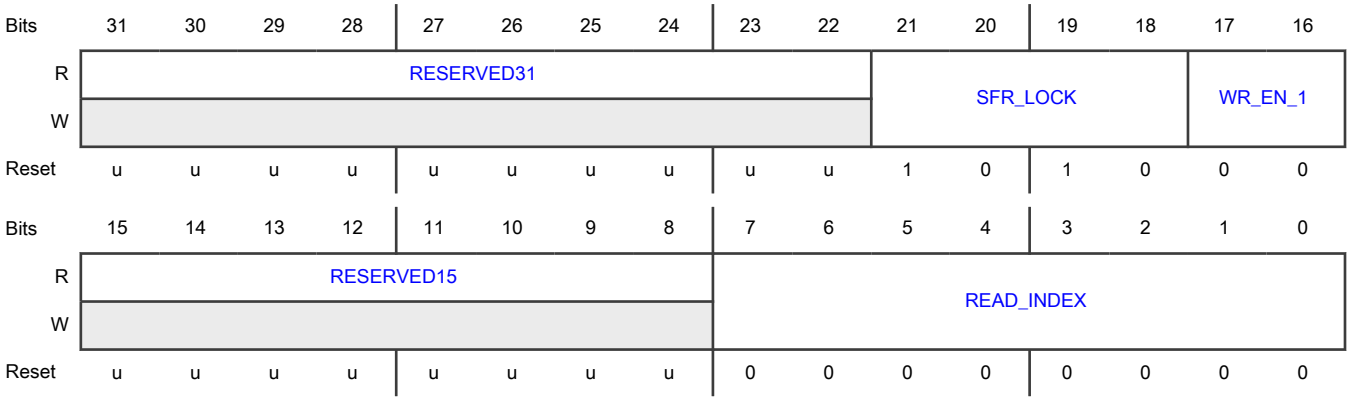
Offset

Register	Offset
CTRL_1	4h

Function

Control Register 1 SFR controls the flow of FSM.

Diagram



Fields

Field	Function
31-22 RESERVED31	Reserved for Future Use
21-18 SFR_LOCK	LOCK register for GLIKEY LOCK register for GLIKEY 4'h5 - Locked

Table continues on the next page...

Table continued from the previous page...

Field	Function
	4'hA - Unlocked **NOTE** When IP is locked, Write access to SFR is invalid **NOTE** This field is volatile.
17-16 WR_EN_1	Write Enable One **NOTE** This field is volatile.
15-8 RESERVED15	Reserved for Future Use
7-0 READ_INDEX	Index status, Writing an index value to this register will request the block to return the lock status of this index. Use this field to read the status of a target index. **NOTE** This field is volatile.

14.5.2.1.4 Interrupt Control (INTR_CTRL)

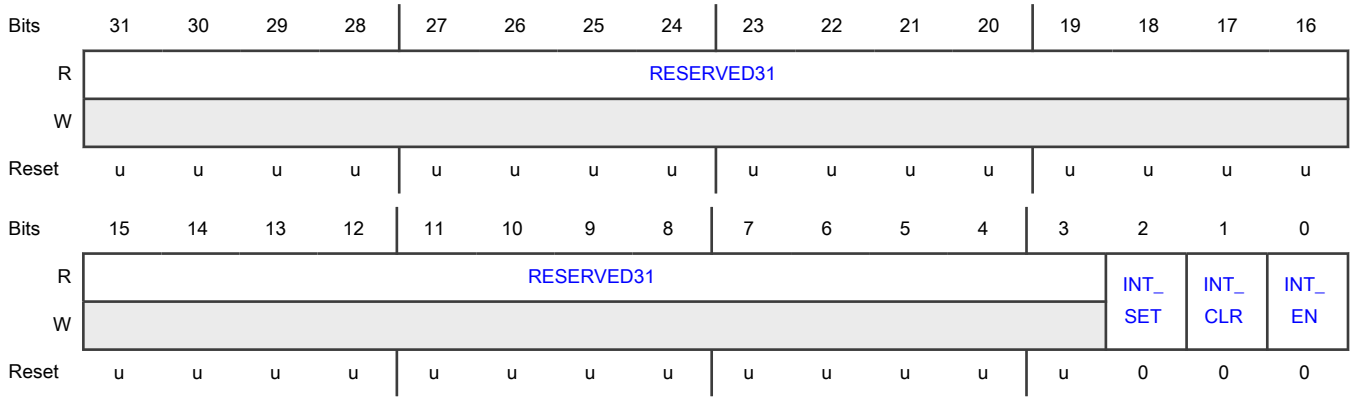
Offset

Register	Offset
INTR_CTRL	8h

Function

Interrupt Control Register

Diagram



Fields

Field	Function
31-3 RESERVED31	Reserved for Future Use
2 INT_SET	Interrupt Set. Writing a 1 to this register asserts the interrupt. This register reads as 0 Use this field to set interrupt. **NOTE** This field is volatile. 0b - No effect 1b - Triggers interrupt
1 INT_CLR	Interrupt Clear. Writing a 1 to this register creates a single interrupt clear pulse. This register reads as 0 When the interrupt is set, use this field to clear this interrupt by writing 1 to this register. **NOTE** This field is volatile.
0 INT_EN	Interrupt Enable. Writing a 1, Interrupt asserts on Interrupt output port Use this field to control whether to trigger an interrupt. **NOTE** This field is volatile.

14.5.2.1.5 Status (STATUS)

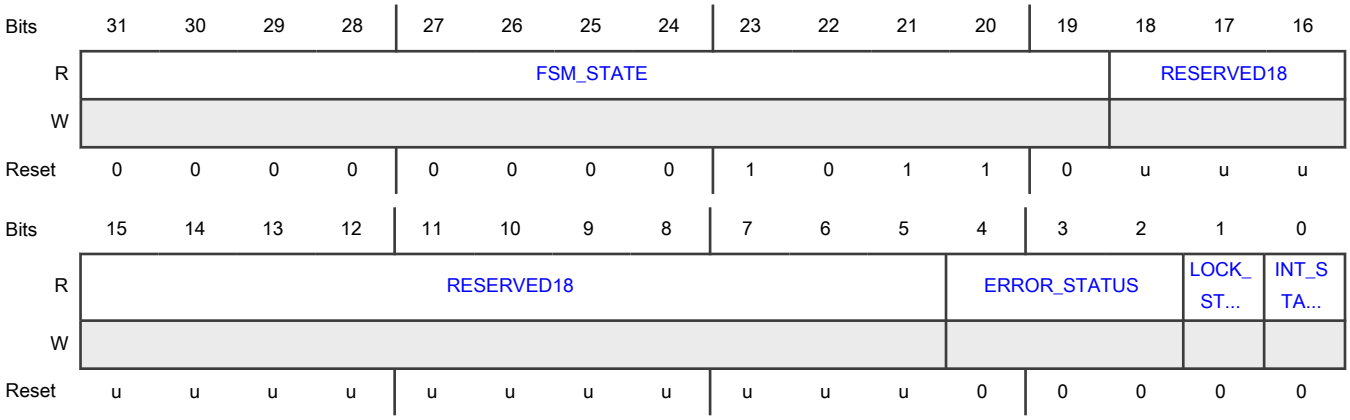
Offset

Register	Offset
STATUS	Ch

Function

Status Register provides the status of GLIKEY

Diagram



Fields

Field	Function
31-19 FSM_STATE	Status of FSM Status of FSM WR_DISABLED 13'b0_0000_0000_1011 INIT 13'b0_0000_0001_0110 STEP1 13'b0_0000_0010_1100 STEP2 13'b0_0000_0101_1000 STEP3 13'b0_0000_1011_0000 STEP4 13'b0_0001_0110_0000 Reserved 13'b0_0010_1100_0000 Reserved 13'b0_0101_1000_0000 Reserved 13'b0_1011_0000_0000 Reserved 13'b1_0110_0000_0000 LOCKED 13'b0_1100_0000_0001 WR_ENABLED 13'b1_1000_0000_0010 SSR_RESET 13'b1_0000_0000_0101 **NOTE** This field is volatile.
18-5 RESERVED18	Reserved for Future Use
4-2	Status of the Error

Table continues on the next page...

Table continued from the previous page...

Field	Function
ERROR_STATUS	Reports the status of error in GLIKEY, where Bit 0 (1 at position 0) indicates that an FSM error has occurred. Bit 1 (1 at position 1) indicates that software tried to address write index greater than the available value. Bit 2 (1 at position 2) indicates that software tried to address read index greater than the available value. **NOTE** This field is volatile. 000b - No error 001b - FSM error has occurred 010b - Write index out of the bound (OOB) error 011b - Write index OOB and FSM error 100b - Read index OOB error 110b - Write index and read index OOB error 111b - Read index OOB, write index OOB, and FSM error
1 LOCK_STATUS	Provides the current lock status of indexes. **NOTE** This field is volatile. 0b - Current read index is not locked 1b - Current read index is locked
0 INT_STATUS	Interrupt Status. Reflects the current status of interrupt. **NOTE** This field is volatile. 0b - No effect 1b - Triggers interrupt

14.5.2.1.6 IP Version (VERSION)

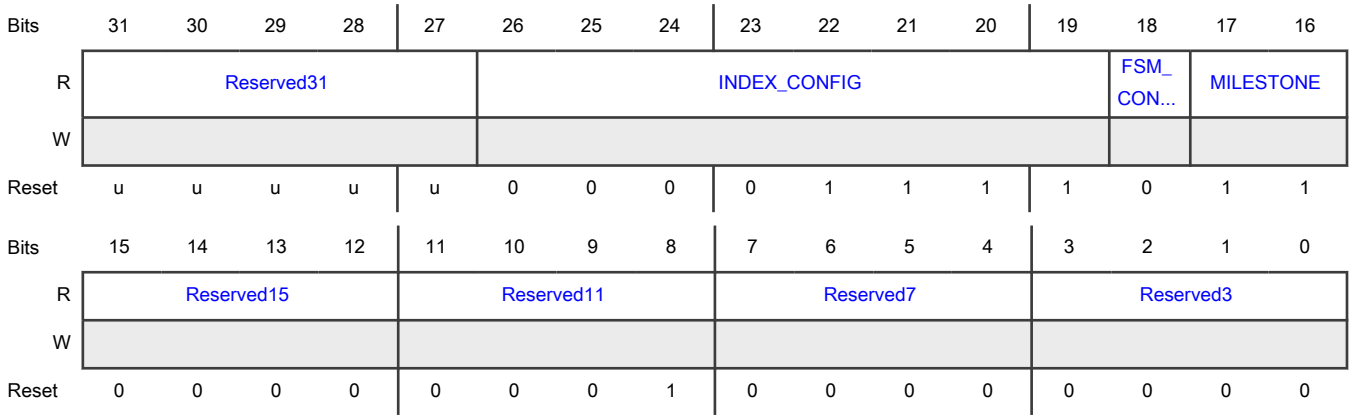
Offset

Register	Offset
VERSION	FCh

Function

IP Version register

Diagram



Fields

Field	Function
31-27 Reserved31	Reserved for Future Use
26-19 INDEX_CONFIG	Configured number of addressable indexes **NOTE** This field is volatile.
18 FSM_CONFIG	0:4 step, 1:8 step **NOTE** This field is volatile.
17-16 MILESTONE	Release milestone. 00-PREL, 01-BR, 10-SI, 11-GO. **NOTE** This field is volatile.
15-12 Reserved15	Reserved
11-8 Reserved11	Reserved
7-4 Reserved7	Reserved
3-0 Reserved3	Reserved

Chapter 15
Secure Generic Interface (SGI)

15.1 Chip-specific SGI information

Table 87. Reference links to related information[1]

Topic	Related module	References
Full description	SGI	SGI
System memory map		System memory map
Clocking		See the chapter "Clocking"
Signal multiplexing	Port control	See the chapter "Signal multiplexing"
System debug		See the chapter "Debug"

1. For all the reference sections and chapters mentioned in this table, refer the Reference Manual.

15.1.1 Module instance

This device has one instance of the SGI module.

15.1.2 Security considerations

For secure applications, NXP recommends that this module is configured for secure and privileged access only to prevent unintentional or malicious modification of the system operation by nonsecure software.

15.2 Overview

The Secure Generic Interface (SGI) is a generic, symmetric cryptographic accelerator which supports secure computation of AES and DES algorithms.

15.2.1 Block diagram

The figure below shows the block diagram for the SGI.

The internal data paths are 32-bit wide.

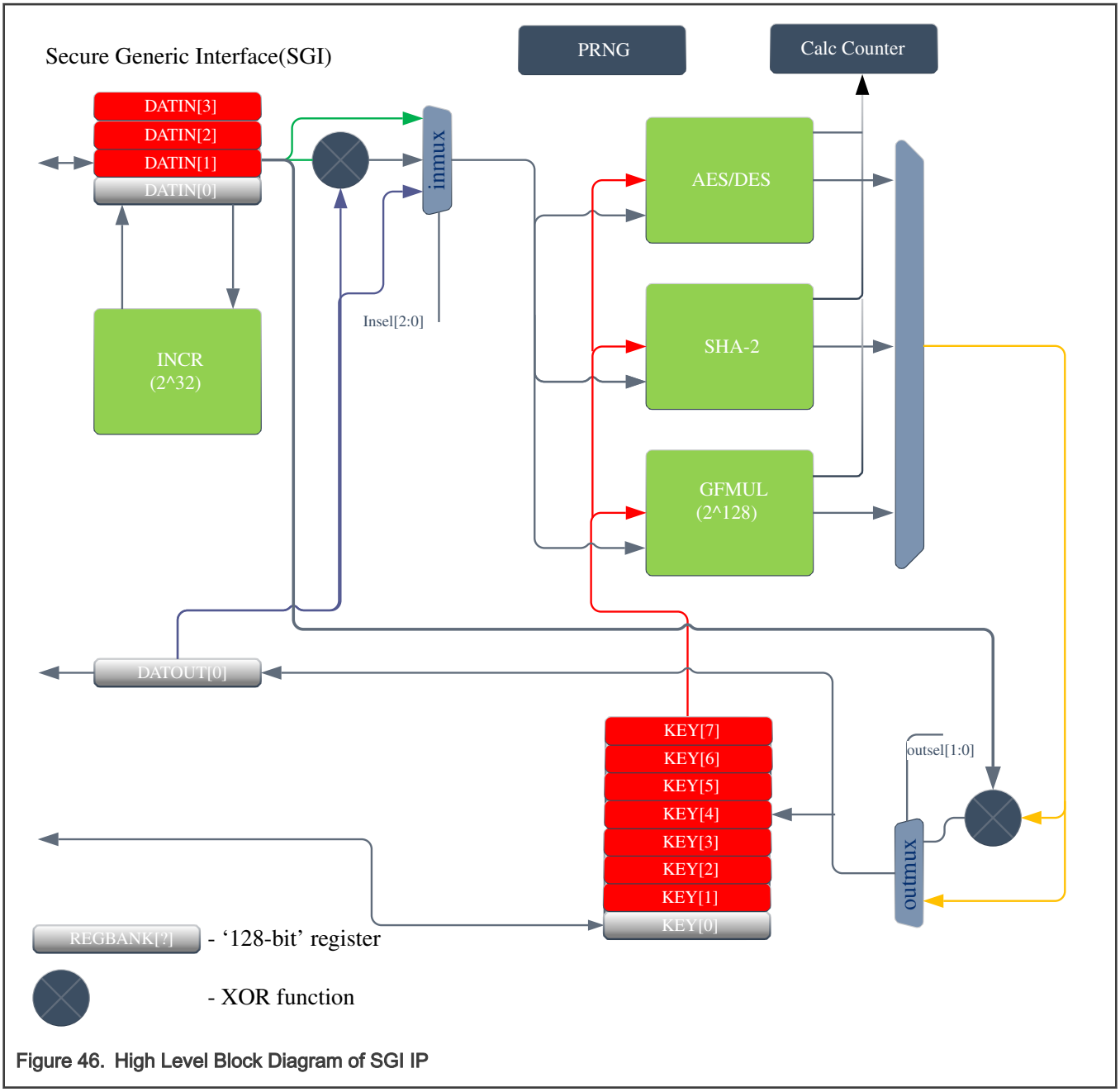


Figure 46. High Level Block Diagram of SGI IP

NOTE

- All internal data buses shown in the diagram are 32-bit.
- All banks have a 32-bit read/write interface.
- The INCR operation can be performed on any DATIN bank, and is invoked via a SFR write to the DATIN register.
- The DES data input can only be driven by the Lower Blocks of the DATIN register banks. The Upper Blocks are not connected to the DES kernel.

15.2.2 Features

The following list gives an overview of the supported features for this configuration of the SGI.

- AES encryption and decryption
- Galois Field Multiplier (AES-GCM)
- 32-bit Increment (INCR)
- SHA-2 (224/256,384/512)
- Chaining mode support (CBC, CFB, OFB, CTR)
- AUTOMODE support including DMA Handshake
  - Key wraping/unwraping support
- Export Fuses Input
- Run time support for making KEYs write-only

15.3 Functional description

The architecture of the SGI IP is based on a number of dedicated register banks for handling data/keys and dedicated kernels for performing crypto and hashing computations. The SGI also includes a Galois Field multiplier and a 32-bit increment function in order to support chaining modes and AES-GCM.

The register banks are used to store data/keys that are input to the kernels. A dedicated data-out register bank is used to store the result of the kernels. The register banks allow concurrent loading/reading of data to increase the overall throughput of the SGI, especially when performing chaining operations such as CBC etc.

For SHA operations, the DATIN and KEY register banks can be used as a FIFO to continuously load a multi-block message that is to be hashed by the SGI.

The SGI provide three main features to support chaining modes, all three are a means to allow xoring of data as described below:

1. XORWR - Optional XOR during writing of a register via the SGI input ports (i.e. a CPU write). Enabled via SGI_CTRL2[XORWR]. This is detailed in Write-XOR (XORWR).
2. INSEL - Optional XOR of DATIN with DATOUT on input to kernels, controlled via SGI_CTRL[INSEL]. This is detailed in INSEL - (kernel input configuration).
3. OUTSEL - Optional XOR of kernel result with DATIN before loading into DATOUT, controlled via SGI_CTRL[OUTSEL]. This is detailed in OUTSEL - (DATOUT input configuration).

15.3.1 Operations

The Generic SGI supports the following operations: AES/DES symmetric crypto operations, SHA-2 cryptographic hash function, a mod(2^32) increment, a GF multiply(GFMUL) and CMAC subkey generation.

The AES/DES/GFMUL and SHA operations will be referred to as crypto operations throughout this document.

NOTE

Please refer to the Features section for the actual operations supported by the SGI configured for your SoC.

The crypto operations are started by selecting the desired operation via the i_crypto_op input and asserting the i_start input.

The INCR is started by writing to the DATIN register to be incremented while i_incr is asserted.

The SGI also supports flushing of register banks.

Operation Selection

SGI operations are selected via the i_crypto_op input. Please refer to Registers description (SFRs), SGI_CTRL[CRYPTO_OP] field for the encoding of this input.

NOTE

If an operation corresponds to a coprocessor which is not enabled in the configuration, the SGI will instead select the available (i.e. configured) copro which corresponds to the lowest encoding value (i.e. AES if SGI_AES=1, else DES if SGI_DES=1, else GFMUL if SGI_GCM=1 and so on...). The same applies for selecting an RFU encoding.

Export Fuses

The SGI supports a 6-bit i_export_fuses input to allow disabling certain operations via SoC fuses. The following table shows the bit mapping of this input.

NOTE

A setting of 1 in the i_export_control input disables operations.

Table 88. Fuse control mapping

i_export_fuses Bit Settings	Operation Disabled
i_export_fuses[0]==1	AES
i_export_fuses[1]==1	DES
i_export_fuses[2]==1	TDES
i_export_fuses[3]==1	GFMUL
i_export_fuses[4]==1	SHA
i_export_fuses[5]==1	CMAC

Attempting to start an operation which has been disabled results in a security alert (see Security alerts). Attempting to disable via fuse an operation that is already disabled via compile time configuration has no effect. For example, if CMAC is not available because it is configured out, and i_export_uses[5]==1, then attempting to start a CMAC will actually default to the 1st available operation (see Note above). In this case a security-alert would only trigger if an AES was configured to be in, but was disable via fuse.

INSEL - (kernel input configuration)

Please refer to SGI_CTRL[INSEL] field for different options for configuring kernel input.

NOTE

INSEL feature does not apply to SHA operations. In the case of a SHA operation the SHA input data comes from the SHA FIFO and the INSEL field is ignored.

Concurrent operations

Only one crypto operation can be started at a time. Any attempt to start a subsequent operation will be ignored. An INCR operation however can be performed concurrently to an ongoing operation (that is, while SGI_STATUS[BUSY] flag is 1) by simply writing to the DATIN register while SGI_CTRL2[INCR] is set. A flush operation can be triggered at any time. The flush operation is asserted by asserting SGI_CTRL2[FLUSH]. To flush only the KEY register bank, the SGI_CTRL2[KEY_FLUSH] input can be used instead.

15.3.1.1 AES

The AES coprocessor supports 128 and 256-bit AES encryption and decryption operations. The AES kernel always uses the AES encryption key as input key regardless of whether an encryption or decryption operation is supported. The SGI does not support AES decryption keys.

The AES operation is selected by setting i_crypto_op to 3'b000.

The data used for the AES operation is selected based on the configuration of the SGI_CTRL[INSEL], see INSEL - (kernel input configuration).

The key used for the AES operation is selected based on the configuration of the SGI_CTRL[INKEYSEL], see KEY register bank and SGI_CTRL[AES_NO_KL] (Only avaliable on Aegis_hs, Athenium and Athenium_hs only with the ID_CFG_AES_KEEP_KEY option). If SGI_CTRL[AES_NO_KL] is asserted no new key will be loaded and the previously loaded key will be used. For decryption this means a speed up in cycle count (to be the same cycle count as encryption) as the key schedule will not need to be ran.

The AES keysize is configured via the SGI_CTRL[AESKEYSZ] input.

The result of the AES operation is stored in the DATOUT or KEY register banks, see DATOUT register bank and Result to Key.

Table 89. AES (AEGIS) computation time

Number of S-boxes	i_keysize	Clock cycles (total ENC (and DEC if ID_CFG_KEEP_KEY=1))
1	AES-128 AES-256	272+7 380+7

The following table shows the selection of the different AES key sizes.

Table 90. Mapping of SGI_CTRL[AESKEYSZ] to AES key options

SGI_CTRL[AESKEYSZ]	AES Key Size
2'b00	128-bit
2'b10	256-bit
2'b11	128-bit[1]

1. unless ID_CFG_AES_K128_ONLY is defined in which case it is 128-bits

15.3.1.2 Increment (INCR)

The INCR operation is used to increment words within the selected DATIN register. The result of the increment overwrites the value to be incremented. The increment operation can be used to support the CTR chaining mode and AES-GCM.

The increment operation increments by 1 mod 2^32. Any of the 32-bits words can be incremented. In addition, the carry (indicated via SGI_STATUS[OFLOW]) from the previous increment can be used as the carry input for the current increment operation.

The increment operation is triggered by writing any value to the target DATIN word while SGI_CTRL2[INCR] is set. The SFR write does not need to write the full 32-bits of the word to trigger the 32-bit increment, even a byte write (i.e. by asserting only a single write-strobe bit.

```
i_wr_byte_* = 16'h000F  -> Full 32-bit WORD write
i_wr_byte_* = 16'h000C  -> Only 16-bit write(upper 2 bytes)
i_wr_byte_* = 16'h0003  -> Only 16-bit write(lower 2 bytes)
i_wr_byte_* = 16'h0001  -> Only  8-bit write(lowest byte)
```

As described in DATIN register bank, wait states will be inserted (by asserting o_wait) in the case a used DATIN register is incremented immediately after starting a crypto operation, i.e. while the contents of the used DATIN are being transferred to the kernel. At all other times the INCR will not incur any wait states.

In the case where an overflow occurs, the SGI_STATUS[OFLOW] signal will be asserted. This signal is registered within the SGI. It is updated at the end of the write to DATIN (when INCR is enabled, taking into account the priority described in Special write priority.

```
1) Increment word w0
No Overflow(o_overflow=0):
             <------UB-------> <-------LB------>         <------UB-------> <-------LB------>
             b7b6b5b4 b3b2b1b0 b7b6b5b4 b3b2b1b0         b7b6b5b4 b3b2b1b0 b7b6b5b4 b3b2b1b0
```

```
DATIN = 128'h00000000_00000000_00000000_12345678 -> 128'h00000000_00000000_00000000_12345679
                                                 -                                           -
With Overflow(o_overflow=1):
             <------UB-------> <-------LB------>         <------UB-------> <-------LB------>
             b7b6b5b4 b3b2b1b0 b7b6b5b4 b3b2b1b0         b7b6b5b4 b3b2b1b0 b7b6b5b4 b3b2b1b0
DATIN = 128'h00000000_00000000_00000000_FFFFFFFF -> 128'h00000000_00000000_00000000_00000000
                                        --------                                    --------

1) Increment word w0
No Overflow(o_overflow=0):
             <------UB-------> <-------LB------>         <------UB-------> <-------LB------>
             b7b6b5b4 b3b2b1b0 b7b6b5b4 b3b2b1b0         b7b6b5b4 b3b2b1b0 b7b6b5b4 b3b2b1b0
DATIN = 128'h00000000_00000000_12345678_00000000 -> 128'h00000000_00000000_12345679_00000000
```

The overflow flag is cleared (set to 0) on the following conditions:

- IP reset (that is, assertion of i_rst)
- On a increment operation which does not overflow
- On a full flush (SGI_CTRL2[FLUSH])
- On a DATIN flush (SGI_CTRL2[DATIN_FLUSH])
- On a SGI_CTRL2[FLUSHWR] to DATIN

The SGI INCR function supports optionally using the carry out (that is, SGI_STATUS[OFLOW]) from the previous increment as the carry input for the current increment operation. This is selected via the SGI_CTRL2[INCR_CIN] input. When this input if 0, the INCR function always performs a standard increment by 1. When this input is 1, the INCR function checks the overflow from the previous increment operation. If there was no overflow, the INCR operation basically has no effect on the output data, however if the previous overflow was 1 the INCR will work as normal and increment the DATIN word by 1.

The table below summarises what was described above.

Table 91. SGI increment bahaviour

SGI_CTRL2[INCR_CIN]	SGI_STATUS[OFLOW]	DATIN
0	0	+1
0	1	+1
1	0	NO CHANGE
1	1	+1

15.3.1.3 GF Multiplier(GFMUL)

The GFMUL coprocessor supports GF (2^128) multiplication of two 128-bit numbers as specified in the NIST SP-800-38D standard. The GFMUL coprocessor, together with the AES or DES coprocessors and the SGI's INCR operation can be used to implement the Galois/Counter Mode (GCM) of operation and to generate Message Authentication codes (MAC) to support authenticated encryption.

The GFMUL operation is selected by setting i_crypto_op to 3'b011.

The first data input (i.e. operand 1) used for the GFMUL operation is selected based on the configuration of SGI_CTRL[INSEL], see INSEL - (kernel input configuration).

The second data input (i.e. operand 2) used for the GFMUL operation is always taken from key register based on the configuration of the SGI_CTRL[INKEYSEL], see KEY register bank.

The result of the GFMUL operation is stored in the DATOUT or KEY register bank, see DATOUT register bank and Result to Key.

The GFMUL INXOR mode allows the user to xor the data input of the GF multiplier with the current contents of the coprocessors working register (i.e. previous GFMUL result). This mode allows the GHASH result to remain in the coprocessor, only having to

be written to the SGI DATOUT register bank at the very end of a AES-GCM operation. This mode can be enabled by setting the SGI_CTRL2[GCM_INXOR] input to 1.

The following table shows the number of sgi_clk cycles needed to perform an GFMUL operation. The numbers indicate the number of cycles SGI_STATUS[BUSY] will be asserted.

Table 92. GCM Computation Time

Number of STEPS COMPILE TIME OPTION	Clock Cycles (SC+FA)	Clock Cycles SC Only	Clock Cycles No-SC, No-FA
0	580 + 8	516 + 8	132 + 8
1	324 + 8	260 + 8	68 + 8
2	196 + 8	132 + 8	36 + 8
3	132 + 8	68 + 8	20 + 8
4	100 + 8	36 + 8	12 + 8
5	84 + 8	20 + 8	8 + 8
6	76 + 8	12 + 8	6 + 8

15.3.1.4 SHA-2

The SHA-2 coprocessor supports the following HASH sizes: 224/256/384/512.

The SHA-2 operation is selected by setting i_crypto_op to 3'b100.

The input message for the SHA-2 operation is taken from the SHA FIFO (see SHA FIFO control).

The result of the SHA operation is stored in the DATOUT or KEY register bank (see SHA result read-out).

The SHA must be enabled by asserting SGI_SHA2_CTRL2[SHA2_EN] prior to using the SHA kernel.

In Automatic mode the SHA performance is dependent on other factors such as bus throughput, FIFO size etc.

The SHA size can be selected via the SGI_SHA2_CTRL[SHA2_SIZE] input according to the following table:

Table 93. SHA Size selection

SGI_SHA2_CTRL[SHA2_SIZE] value	SHA size
2'b00	SHA-224
2'b01	SHA-256
2'b10	SHA-384
2'b11	SHA-512

15.3.1.4.1 Normal (NORM) SHA mode

In this mode, the SHA operation is started by asserting the SGI_CTRL[START] input while SGI_SHA2_CTRL[SHA2_MODE] = 0. In this mode the number of blocks to be processed is determined prior to starting the SHA operation via the SHA FIFO high and low limits (see SHA FIFO control). Once the SHA operation completes the user can read the result as described in SHA result read-out.

15.3.1.4.2 Automatic (AUTO) SHA mode

In this mode, the SHA operation is started by asserting the SGI_CTRL[START] input while SGI_SHA2_CTRL[SHA2_MODE] = 1. In this mode the number of blocks to be processed is determined during the operation based on the amount of data written into the SHA FIFO after starting the AUTO mode SHA operation.

In AUTO mode, when 16 words (or 32 in the case of SHA 384/512) have been written into the FIFO the SGI will automatically start processing the input data. Further blocks can be loaded into the FIFO whilst the SHA coprocessor is busy processing the current block. In the case where the FIFO becomes full the SGI_STATUS[SHA_FIFO_FULL] signal will be asserted. When a write attempt is made to the FIFO whilst the FIFO is full the o_wait output will be asserted and the write to the FIFO will complete when a free space becomes available in the FIFO.

To end the SHA operation when in AUTO mode the SGI_SHA2_CTRL[SHA2_STOP] input simply needs to be asserted (1-cycle pulse). The SGI will finish off processing the last block and will then de-assert SGI_STATUS[BUSY] once the last block has been processed. An exception to this is if the AUTO SHA operation is stopped prematurely, see SHA errors for further details.

15.3.1.4.3 SHA FIFO control

The SHA FIFO is composed of the DATIN and KEY register banks as depicted below and is used to store the input block for both SHA modes (AUTO and NORM).

```
<----------128-bits----------->
+-----------------------------+
+           KEY[N]            +
+-----------------------------+
+              .              +
+-----------------------------+
+              .              +
+-----------------------------+
+              .              +
+-----------------------------+
+           KEY[0]            +  -- HIGH_LIMIT --
+-----------------------------+                  |
+          DATIN[N]           +                  |
+-----------------------------+                  |
+              .              +                  |
+-----------------------------+                  |
+              .              +                  |
+-----------------------------+                  |
+          DATIN[0]           +  -- LOW_LIMIT ----- Limits are inclusive, so in this example
+-----------------------------+                     all of DATIN register bank plus KEY[0]
+   W3   +  W2  +  W1  +  W0  +                     make up the SHA FIFO
+-----------------------------+

'MAX FIFO SIZE'       = SGI_NUM_KEY + SGI_NUM_DATIN
'ACTUAL FIFO SIZE'    = HIGH_LIMIT - LOW_LIMIT + 1

The LOW_LIMIT and HIGH_LIMIT control fields allow the user to restrict the available DATIN and KEY
register banks for SHA FIFO use.

Note 1: The 'LOW_LIMIT' and 'HIGH_LIMIT' can be set to the same value indicating that the SHA FIFO
consists of a single 128-bit bank

Note 2: When setting the 'LOW_LIMIT' to be greater than the 'HIGH_LIMIT' (LOW_LIMIT>HIGH_LIMIT), the
SGI will internally force the LOW_LIMIT to be the same as the HIGH_LIMIT effectively configuring the
SHA FIFO for a single 128-bit bank.

Note 3: If the HIGH/LOW limits exceed the maximum configured FIFO size, then internally force
HIGH/LOW limit to 'MAX_FIFO_SIZE'
        Example: SGI_NUM_KEY=4, SGI_NUM_DATIN=3 -> MAX_FIFO_SIZE=7, so if HIGH_LIMIT is 10, force it
to 6(since limit numbering starts at 0)
```

In AUTO mode, the SHA FIFO must be loaded via the SGI_SHA_FIFO SFR after having started the SHA operation. This allows the user to load a full SHA message by simply writing the same SFR and not having to perform any address management. The SGI will automatically handle where in the FIFO the data is written to and the SGI will wait the bus in case the FIFO becomes full

and a new write is attempted. The SGI will start to empty the FIFO as soon as it can (that is, whenever the SHA coprocessor is not busy). In AUTO mode the FIFO does **not** need to be large enough to hold a full SHA block as it only acts as a message buffer.

In NORM mode, the message must be loaded into the SHA-FIFO by directly accessing the DATIN and or KEY register banks and placing the SHA block into the FIFO accordingly. The SHA operation can be started once the message has been fully loaded into the FIFO. The SGI will only start to empty the FIFO when i_start is asserted so the full message needs to be pre-loaded into the FIFO prior to starting the operation. In NORM mode the FIFO must be configured to be large enough (via the low/high limit settings) to store an integer multiple of SHA blocks (depending on SGI_SHA2_CTRL[SHA2_SIZE] input a block is either 512-bits or 1024-bits) as the full message must be pre-loaded into the SGI prior to starting the operation.

NOTE

In NORM mode the SHA message size is limited to the ACTUAL FIFO SIZE where as in AUTO mode the message can be any size.

NOTE

Writes to SGI_SHA_FIFO are only effective in SHA AUTO mode. Writes to SGI_SHA_FIFO at any other time (that is, during SHA NORM mode or out with a SHA operation) will be ignored.

Internally within the SGI the SGI maintains a FIFO write pointer and a FIFO read pointer. The two pointers are reset whenever AUTO mode is entered and point to the location Low Limit-WO. Every write to the FIFO in AUTO mode via the SGI_SHA_FIFO SFR causes the write pointer to increment by 1.

15.3.1.4.4 SHA errors

In certain cases the SGI can inform the user of incorrect use of the SHA. The SHA error is intended as a functional error detector rather than as a security countermeasure and therefore the SHA error notification is done via the SGI_STATUS[SHA_ERROR] output rather than the SGI_STATUS[ERROR] output of the SGI. The SHA error can occur in the following two cases:

- Disabling AUTO mode whilst a block has been partially loaded into the FIFO but not yet processed. See Note below.
- Starting NORM mode SHA operation when the ACTUAL FIFO SIZE is configured to not be a multiple of the block size (or hash size if SGI_SHA2_CTRL[HASH_RELOAD] is high). See Note below.

NOTE

In this case the SGI_STATUS[BUSY] signal will de-assert and o_done will assert to indicate the incorrect ending of the AUTO SHA operation.

NOTE

In this case because the invalid use is detected at the time of starting the computation both SGI_STATUS[BUSY] and o_done will not be asserted.

The SHA error is cleared by any one of the following methods:

- Assertion of i_reset (or i_sync_reset if used)
- Assertion of SGI_CTRL2[FLUSH]
- Completion of a successful SHA operation

NOTE

When a SHA-error occurs the last crypto operation state is not updated so performing a TRIGGER_UP operation after an errored SHA will return the results of the operation carried out before the SHA. Please refer to DATOUT register bank for further details on last crypto op state.

15.3.1.4.5 Busy and done indicators

The SGI supports a SGI_STATUS[BUSY] and SGI_STATUS[SHA2_BUSY] output. When performing SHA operations, the SGI_STATUS[SHA2_BUSY] output only asserts when the SHA-2 coprocessor is busy processing a block. The SGI_STATUS[BUSY] output asserts as soon as the SGI_CTRL[START] input is asserted, and de-asserts when: 1) In NORM

mode when the last block of the message has been processed, and 2) In AUTO mode when the user ends the operation by asserting SGI_SHA2_CTRL[SHA2_STOP].

The o_done output pulses each time the o_busy signal de-asserts at the end of the SHA operation.

15.3.1.4.6 SHA operation counter

The SHA operations can increment the Calculation Counter(see Calculation counter) whenever a block is processed. This behaviour can be enabled by setting the SGI_SHA2_CTRL[SHA2_COUNT_EN] input to 1. When SGI_SHA2_CTRL[SHA2_COUNT_EN] is 0 the SHA does not influence the Calculation Counter.

15.3.1.4.7 SHA result read-out

The SGI SHA supports both Result to DATOUT (SGI_CTRL2[RKEY] = 0) and Result to KEY (SGI_CTRL2[RKEY] = 1) for outputting the HASH value at the end of a SHA operation(for both AUTO or NORM modes).

Only 128-bit of the HASH are output at the end of the SHA operation. The remaining parts of the HASH need to be output using the TRIGGER_UP feature.

NOTE

Starting a new SHA operation always resets the chunk selector to point to the lowest 128-bits of the HASH result. Only by performing a TRIGGER_UP after a SHA operation (i.e. SHA was the last crypto operation executed) can the chunk selector be incremented.

15.3.1.4.8 SHA auto initialization

When starting a new SHA operation, whether in NORMAL or AUTO mode, the SHA copro always re-initialises the working register with the relevant SHA initialisation vector so a fresh HASH operation is assumed. However, this may not be desired as the user may want to HASH a message by performing multiple SGI SHA operations (NORMAL or AUTO). To support this the user may disable the auto initialisation of the SHA working register by setting SGI_SHA2_CTRL[NO_AUTO_INIT] to 1 at the time of starting the SHA operation. This feature can also be used to facilitate SGI partial-HASH mode (see section below).

15.3.1.4.9 SHA partial hash mode

The SGI supports partial processing of a SHA message. Partial processing is supported in the following two forms:

- Start without intermediate HASH value (SGI starts a partial HASH)

  In this case, the SGI is used to process M out of N blocks (where N blocks is the size of the total message) and the remaining N-M blocks are to be processed by some other means (possibly using the SGI at a later moment in time or by FW). For this case no special features are needed and the user must simply load into the SGI the M blocks and read out the partial HASH value when the computation is finished.

- With intermediate HASH value (SGI completes a partial HASH)

  In this case, the SGI is used to complete a HASH computation that has been already partially computed elsewhere (possibly by the SGI at some earlier moment in time or by FW). For this case the SGI offers the SGI_SHA2_CTRL[HASH_RELOAD] and SGI_SHA2_CTRL[NO_AUTO_INIT] features. To perform a partial HASH in this case the user must first load the partial HASH value into the SGI while the SGI_SHA2_CTRL[HASH_RELOAD] is set to 1. After loading in the partial HASH value into the SGI, the user can then complete the HASH computation with SGI_SHA2_CTRL[NO_AUTO_INIT] set to 1 so that the partial HASH value is used as the starting point for the new HASH computation.

  NOTE

  To load the partial HASH value the user may use AUTO or NORM mode. In AUTO mode SGI_CTRL[START] needs to be set to 1 first and then the partial HASH value is loaded into the SGI (via SHA_FIFO SFR). In NORM mode it is the opposite, first the partial HASH value is loaded into the SGI and thenSGI_CTRL[START] is set to 1 to trigger the transfer of the HASH value into the SHA copro.

NOTE

When using Partial HASH mode with NORM mode, the size of reload partial hash value should be 256 bits for SHA-224/256 and 512 bits for SHA-384/512.

NOTE

HASH reload (using SGI_SHA2_CTRL[HASH_RELOAD]) does not cause DATOUT to update. HASH reload simply updates the SHA kernels working register.

15.3.2 Register banks

The DATIN register bank can be composed of up to 4 128-bit banks. The number of banks is compile time configurable. The DATIN register bank is used to store the kernel data input and can be loaded whilst a crypto operation is in progress (SGI_STATUS[BUSY] = 1). The DATIN registers can be optionally xored with the DATOUT register before using the DATIN value as input to a kernel (enabled via SGI_CTRL[INSEL]).

NOTE

There is no option to XOR the different DATIN registers with each other prior to feeding the kernel data input. Instead the user should make use of the XORWR feature (see Write-XOR (XORWR)).

The DATOUT register bank is composed of a single 128-bit bank. It stores the result from the kernels including the GFMUL multiplier and SHA kernel. The result from the kernels can optionally be xored with one of the DATIN registers before loading into the DATOUT register(enabled via SGI_CTRL[OUTSEL]).

NOTE

The result of the INCR function is NOT stored in DATOUT but instead it overwrites the DATIN register that is to be incremented.

The KEY register bank can be composed of up to 8 128-bit banks. The number of banks is compile time configurable. The KEY register bank is used to store the kernel key input, and is also used to store one of the operands for the GF multiplier (GFMUL). The KEY register bank may be used to store the result of a crypto operation to handle encrypted keys within the SGI.

15.3.2.1 General register bank

The SGI is composed of a number of register banks. All register banks are based on the general scheme depicted below:

Each register bank is composed of N 128-bit registers. Depending on the actual register bank (i.e. DATIN, DATOUT, KEY). The actual register banks are detailed later in this section.

Write ccess

Each 128-bit register bank supports a 32-bit write port. However the write strobes are based on a 8-bit granularity. The kernels will always write to the register banks with a 32-bit write operation (i.e. asserting 4 bits of the write strobe). The wrapper may write 32-bits or 16-bits by asserting the correct number of bits in the write strobe.

The write signals are declared as follows:

```
input [SGI_BUS_WIDTH:0] i_wr_data                       - 16/32-bit write-data
input [            1:0] i_addr_<regbankName>            - Selects one of the 'N' 'banks' for write
input [           15:0] i_wr_byte_<regbankName>         - Selects any number of 16 'bytes' in a
'bank' for write

for example:
input [ 1:0] i_addr_datin
input [15:0] i_wr_byte_datin


When SGI_BUS_WIDTH=16, the 16-bit 'i_wr_data' input bus is duplicated to make a single 32-bit
internal bus:
  -> internal_bus = {i_wr_data[15:0], i_wr_data[15:0]}
```

```
When SGI_BUS_WIDTH=32, the 32-bit 'i_wr_data' input bus directly drives the 32-bit internal bus.
  -> internal_bus = {i_wr_data[31:0]}

The 32-bit internal bus maps to the bytes of the register bank(see Figure titled 'Generic Register
Bank' above) as detailed below:
Bytes b4/b0 will map to 'internal_bus[ 7: 0]'.
Bytes b5/b1          to 'internal_bus[15: 8]'.
Bytes b6/b2          to 'internal_bus[23:16]'.
Bytes b7/b3          to 'internal_bus[31:23]'.

Example:
So setting:
    i_wr_byte_datin=16'h0FF0, and
    i_addr_datin   =1
will set DATIN[1].UB[b3:b0] and DATIN[1].LB[b7:b4] to the value of 'internal_bus[31:0]'.
In other words 8 bytes in total will be written in parallel.  Note that this example
is just to illustrate the interface between the wrapper and the SGI, it is unlikely that
the wrapper will perform such a write.

Note: The above reflects the behaviour when i_bytes_order=0.  When i_bytes_order=1 the byte order is
swapped, see further details in the sections below.
```

NOTE

Where the i_addr_* input points to a register that does not exist (i.e. due to configuration of the SGI), then the REGBANK[0] register will be written instead.

Read access

Each Register Bank can be accessed for read 32-bits at a time. The data is presented on the output o_rd_data_<regbank_name> bus. The read strobes basically address the 32-bit words within a certain bank of the register bank. The wrapper must perform further multiplexing to handle a 16-bit interface.

```
input        i_rd_sel_<regbankName>
input [ 1:0] i_addr_<regbankName>
input [ 1:0] i_rd_word_<regbankName>
output[31:0] o_rd_data_<regbankName>

for example:
input         i_rd_sel_datin
input [1:0]   i_addr_datin
input [1:0]   i_rd_word_datin
input [31:0]  o_rd_data_datin

The 'i_addr_<regbankName>' input selects on of the 'N' 'banks' for read.
The 'i_rd_word_<regbankName>' input selects one of the four 32-bit words within a 'bank' as follows:
The read address map as follows:

i_rd_word_<regbankName>        |           Sub-block
-------------------------------------------------------------------
2'b00                          |   <regbankName>[N].LB[b3:b0]
2'b01                          |   <regbankName>[N].LB[b7:b4]
2'b10                          |   <regbankName>[N].UB[b3:b0]
2'b11                          |   <regbankName>[N].UB[b7:b4]


Example:
So setting 'i_addr_datain=1' and'i_rd_word_datin=2' will present the 32-bits of DATIN[1].UB[b3:b0]
on 'o_rd_data_datin[31:0]> for example.
```

```
Byte b0/b4 will map to 'o_rd_data_<regbank_name>[ 7: 0]' and
     b3/b7          to 'o_rd_data_<regbank_name>[31:24]'
```

NOTE

Where the i_addr_* input points to a register that does not exist (i.e. due to configuration of the SGI), then the REGBANK[0] register will be returned instead.

In order to handle concurrent accesses, each register bank has a corresponding read-strobe, i_rd_sel_<regbankName> which indicates that a read to the corresponding register bank is in progress. This signal can be used in the SGI to handle concurrent read accesses (i.e. read while SGI_STATUS[BUSY] is 1).

Byte order (Endian) control

The SGI allows swapping the byte order of the writes and read to and from the regbanks. This is controlled via the SGI_CTRL2[BYTES_ORDER] input. This only affects regbank (DATIN, DATOUT, KEY) accesses and the KEY DIGEST. It does not influence access to any other register (COUNT, SFRSEED etc).

The byte order is swapped as follows:

Normal - i_bytes_order = 0

{b3, b2, b1, b0} = {bus[31:24] , bus[23:16], bus[15: 8], bus[7: 0] }

Normal - i_bytes_order = 1

{b3, b2, b1, b0} = {bus[7 : 0] , bus[15: 8], bus[23:16], bus[31:24] }

Write-XOR (XORWR)

The register bank can be configured to support the XORWR feature. Register bank matrix provides details on which actual register bank uses the XORWR feature.

For register banks that support this feature, it is possible to write to the register bank such that the value actually loaded into the register is the written value xored with the current contents of the register. This is enabled by asserting the SGI_CTRL2[XORWR] input of the SGI. The following snippet shows the XORWR feature using pseudo-code.

```
if (i_xorwr==0)
  register = write_value
else
  regsiter = write_value ^ register

^ - Xor operator
```

Register bank reset

The register banks do not have a reset input and therefore cannot be reset. However, the register banks may be flushed as described in Flush. This means that at power-up the register banks will uninitialised but due to the auto flush after reset they will be loaded with random data.

Special write priority

The SGI register banks can be written in three special modes, i.e. INCR (Increment (INCR)), XORWR (Write-XOR (XORWR)) and FLUSHWR (Flush Write (FLUSHWR)). Each of these are enabled via their corresponding input, namely SGI_CTRL2[INCR], SGI_CTRL2[XORWR] and SGI_CTRL2[FLUSHWR]. These inputs may be asserted simultaneously, but only one special mode will be selected based on the following priority: FLUSHWR-1, XORWR-2, INCR-3.

NOTE

1 indicates highest priority and 3 the lowest.

DATIN/KEY access during SHA operation

Please be aware that DATIN/KEY SFRs are not accessible (writes are ignored, reads return random data) during a SHA operation (i.e. from assertion of SGI_CTRL[START] to de-assertion of SGI_STATUS[BUSY] in both AUTO and NORM SHA modes)

15.3.2.2 DATIN register bank

The DATIN register bank is composed of a configurable number of banks. They are used to store the data input for the AES and DES kernels, as well as one of the inputs for the GFMUL multiplier.

DATIN access

The DATIN banks can be accessed (read or write) at any time. If a used DATIN bank is accessed immediately after starting a AES/DES or GCM operation, the o_wait signal will be asserted while the contents of the used DATIN register are transferred to the kernel.

INCR

The DATIN register banks can be incremented as described in Increment (INCR). The increment is performed by writing to the register while the SGI_CTRL2[INCR] input is set.

Write XOR

The DATIN register bank supports the XORWR feature.

Concurrent access

The DATIN register bank may be accessed as normal during the SGI idle period, i.e. at the time that no operation is currently being executed. This idle period is indicated with the SGI_STATUS[BUSY] input being at 0.

During busy, the DATIN register bank can be accessed as normal. However, when writing DATIN banks during busy a number of wait cycles may be inserted depending on how soon after the start of the computation the used DATIN register is accessed. The wait cycles are inserted so that the DATIN register value can be transferred to the kernel internal working register before the register is written/read since there is only a single write/read port on the internal register banks.

Also note that when using the START_UP update mode for DATOUT (see DATOUT register bank for details about updating of DATOUT), any attempt to write DATIN during the period where DATOUT is being updated (and therefore DATIN contents still need to be transferred to the kernel) will result in the o_wait signal asserting as otherwise this would cause the overwriting of the input data.

The wait states are indicated by the o_wait signal.

The following timing diagram shows an example of a write to DATIN while the DATIN is being transferred to the kernel (indicated by datin transfer-in-progress).

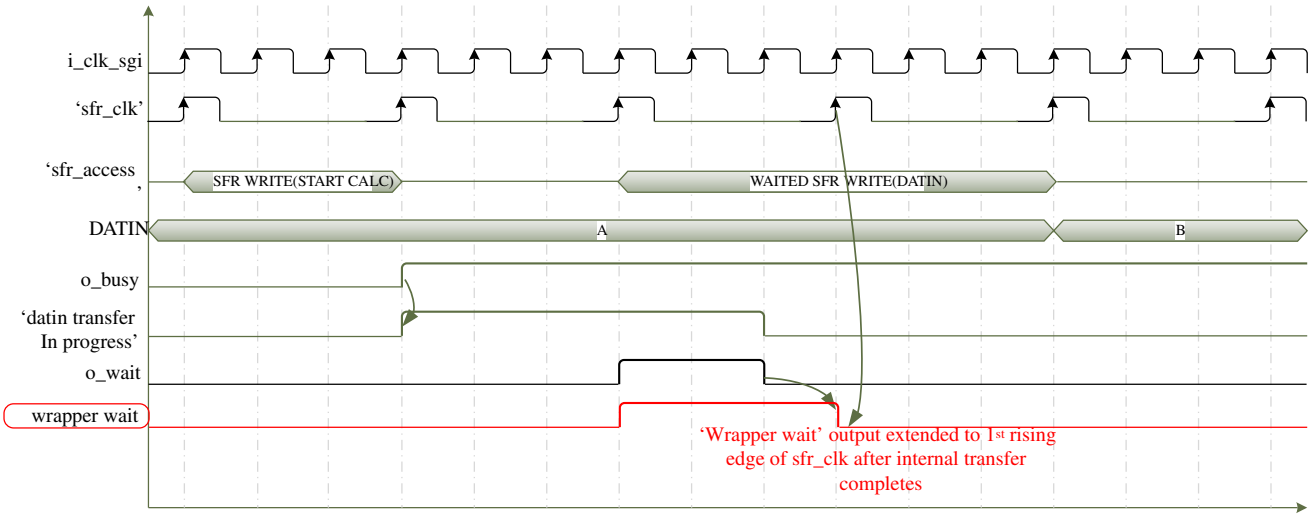


Figure 47. Timing diagram showing wait states insertion due to DATIN write

The write access to DATIN begins while the SGI is busy transferring the DATIN contents to the kernel (AES/DES or GCM). This transfer activity is triggered by the starting of the crypto operation (START CALC). Since the write to DATIN occurs while the transfer is in progress, the o_wait output is asserted to wait the system bus until the SGI is ready. The o_wait signal is asserted

as soon as a write is detected to a used DATIN register (while it is being internally transferred to the kernel), and is de-asserted on the first sfr_clk edge after the de-assertion of the internal datin transfer-in-progress signal.

DATIN flush

The DATIN register bank may be flushed (filled with random data) by asserting the SGI_CTRL2[FLUSH] input at any time (see Flush).

Access to non-existent DATIN

When attempting an access to a non-existent DATIN bank, that is, when i_addr_datin is greater than SGI_NUM_DATIN, the DATIN[0] bank will be accessed. No security alert will be triggered in this case as illegal SFR accesses should be handled in the platform-specific wrapper.

15.3.2.3 DATOUT register bank

The DATOUT register bank is composed of a single bank. The DATOUT register bank is a RW register that is updated by the SGI during an AES/DES or GFMUL operation. The update of DATOUT with the kernel result can occur either at the start of the operation (i.e. the result of a previous operation) or the end of the operation (i.e. the result of the current operation). Alternatively the kernel result may not be loaded into DATOUT at all during the computation but instead the transfer of the kernel result to DATOUT can be triggered directly by using the SGI_CTRL[DATOUT_RES] input. Further details of this can be found in the sections below.

When using START_UP mode(see details below), the DATOUT register can be read at any time. However, reading may cause the SGI to insert wait states depending on when the read is attempted. When using END_UP mode, reading the DATOUT while SGI_STATUS[BUSY] = 1 will trigger a security alert, see Table 98.

Clock

The DATOUT register is clocked off the sgi_clk, see Power Clock Reset description.

OUTSEL - (DATOUT input configuration)

The DATOUT register bank can be configured to store the result directly from the kernels or alternatively perform an optional XOR between the kernel output and the DATIN registers. Please refer to SGI_CTRL[OUTSEL] field for different options for configuring DATOUT input.

Updating of DATOUT

The DATOUT register can be configured to be updated at either the start of a crypto operation (START_UP) or at the end of a crypto operation (END_UP). Alternatively the DATOUT register can be updated without executing a crypto operation (TRIGGER_UP). A crypto operation can also be executed without updating the DATOUT register (NO_UP). These four modes are described in more detail below.

NOTE

The Key Checksum register is updated at the same time as DATOUT.

The reason why the four options have been provided is to allow DATOUT to be read by the firmware while a crypto operation is currently in progress. This essentially allows the previous crypto operation result to be read while the next operation is running. However, if only option-2 (END_UP) below was supported, there is a risk that the kernel may overwrite the DATOUT register before the previous result has been read out by the FW. This may be the case if the FW performance can not be guaranteed, for example due to a slow CPU clock. To get around this problem, options (1) and (3) allow the FW to control exactly when the DATOUT register is updated with the result.

In a chaining sequence that consists of M crypto operations, option-1 (START_UP) will be used when launching operation 2 to M (where the first operation is numbered as 1), and option-3 (TRIGGER_UP) will be used after the last crypto operation has ended to obtain the result of operation M. The first operation in such a sequence will use option-4 (NO_UP) since the result of the first operation is actually read during the second operation of the chain.

1) START_UP - Update DATOUT at the start of the operation:

This mode is selected when the SGI_CTRL[DATOUT_RES] input is set to 2'b01 when the operation is started (i.e. SGI_CTRL[START] is asserted). In this mode the DATOUT register will be loaded with the current contents of the kernel's internal

working register. The kernel will be selected based on the last crypto operation that had been run (after reset there is no last operation so SGI will use first available kernel based on IP configuration instead). The loading of DATOUT will take place BEFORE the starting of the requested operation so that the updated DATOUT value can be used as input data to the operation.

Accessing the DATOUT register immediately after the starting of the computation, while the DATOUT is still being updated will cause the o_wait signal to assert. This should be used within the wrapper to wait state the SFR bus until the DATOUT register has been updated.

2) END_UP - Update DATOUT at the end of the operation

This mode is selected when the SGI_CTRL[DATOUT_RES] input is set to 2'b00 when the operation is started (i.e. SGI_CTRL[START] is asserted). In this mode the DATOUT register will be loaded with the kernel result at the end of the current operation. Accessing DATOUT during busy (i.e. SGI_STATUS[BUSY] is high) whilst using END_UP mode will result in an error (see Table 98). In other words, in this mode the DATOUT register cannot be read during an operation.

3) TRIGGER_UP - Update DATOUT on request

This update to DATOUT is triggered by asserting the SGI_CTRL[START] trigger input whilst SGI_CTRL[DATOUT_RES] is set to 2'b10. This will trigger the transfer of the contents of the kernel's internal working register to DATOUT. The SGI_STATUS[BUSY] signal will be asserted for the duration of the DATOUT update.

NOTE

The TRIGGER_UP option does not allow the intermediate contents of the kernels to be output to DATOUT since SGI_CTRL[START] assertion while the kernels are busy has no effect.

NOTE

The TRIGGER_UP option does NOT start an actual crypto operation. It simply transfers the contents of the kernel's working register to DATOUT.

NOTE

During TRIGGER_UP, the i_crypto_op input is completely ignored and instead the last crypto operation is determined from an internal variable.

4) NO_UP - Crypto operation without an update of DATOUT

This mode is selected when the SGI_CTRL[DATOUT_RES] input is set to 2'b11 when the operation is started (i.e. SGI_CTRL[START] is asserted). In this mode the DATOUT register will be **not** be updated when the requested crypto operation completes.

The table below summarises the DATOUT update modes.

Table 94. Configuration setting for controlling DATOUT update modes

SGI_CTRL[DATOUT_RES]	Update mode
2'b00	END_UP
2'b01	START_UP
2'b10	TRIGGER_UP
2'b11	NO_UP

NOTE

The contents of DATOUT after the very first use of TRIGGER_OP or START_OP will be indeterminate.

DATOUT flush

The DATOUT register bank may be flushed (filled with random data) by asserting the SGI_CTRL2[FLUSH] input at any time (see Flush).

15.3.2.4 KEY register bank

The KEY register bank is composed of a configurable number of banks. They are used to store the key input for the AES and DES kernels. The KEY register banks can also be configured to store the result of a crypto operation for example to support encrypted keys without having to unnecessarily load/offload keys from the SGI.

NOTE

The result of a GFMUL/SHA operation can also be stored into the KEY register bank.

One of the operands for the GFMUL operation is also taken from one of the key registers.

KEY register bank - Kernel KEY input

The KEY register bank can be composed of 2,3 or 4 banks. The key used for an AES/DES operation, and the second operand for the GFMUL are selected from with the KEY register bank with the 4-bit SGI_CTRL[INKEYSEL] signal. The SGI_CTRL[INKEYSEL] signal basically specifies the base address of the key/operand within the register bank. This effectively allows the key/operand to be 32-bit word aligned within the register bank.

The 3 MSBs of SGI_CTRL[INKEYSEL] select the bank and the 2 LSBs select the word. So for example:

SGI_CTRL[INKEYSEL] == 5'b01001 selects the KEY[2].w1 as the base word for the key

SGI_CTRL[INKEYSEL] == 5'b01111 selects the KEY[3].w3 as the base word for the key

The number of 32-bit words that are read from the KEY register bank starting from the base word are specified in the table below:

NOTE

- For the case where SGI_KEY_NUM=2, SGI_CTRL[INKEYSEL]/SGI_CTRL2[KEYRES] values 8 to 15 will map to 0 since there is only two bank's and hence only 8 words.
- For the case where SGI_KEY_NUM=3, SGI_CTRL[INKEYSEL]/SGI_CTRL2[KEYRES]values 12 to 15 will map to 0 since there is only three bank's and hence only 12 words.

Table 95. Number of 32-bit words read by kernel from KEY register bank for different operations

Operation	Number of KEY words
Single DES	2
TDES (2-key)	4
TDES (3-key)	6
AES-128	4
AES-192	6
AES-256	8
GFMUL	4

NOTE

The keys may be stored in such a way that they wrap round the KEY register bank. Note the wrap around point will be influenced by the setting of the SGI_NUM_KEY parameter setting. For example, setting SGI_CTRL[INKEYSEL] to 7 while SGI_NUM_KEY=2 would result in the key for an AES-128 operations being read from DATIN[1].W3 -> DATIN[0].W0 -> DATIN[0].W1 -> DATIN[0].W2.

OUTSEL

The KEY register bank can be configured to store the result directly from the kernels or alternatively perform an optional XOR between the kernel output and the DATIN registers. Please refer to SGI_CTRL[OUTSEL], field for different options for configuring KEY input.

Result to Key

The KEY register bank can be used to store a DES result (64 bits) or an AES result (128 bits) at the end of a crypto operation instead of storing the result in the DATOUT register. This feature can be enabled by setting the SGI_CTRL2[RKEY] input to 1. The SGI_CTRL2[KEYRES] signal specifies the base address (exactly the same as SGI_CTRL[INKEYSEL]) within the register bank where the result is to be stored. This effectively allows the generated/derived key to be 32-bit word aligned within the register bank. The wrapper may decide to force 64-bit alignment or 128-bit alignment by setting the lower bit or lower 2 bits of SGI_CTRL2[KEYRES] to 0.

NOTE

Again, similar to the SGI_CTRL[INKEYSEL] case described above, the result can be placed in the KEY register bank with wrap around. Note the wrap around point will be influenced by the setting of the SGI_NUM_KEY parameter setting.

Write XOR

The KEY register bank supports the XORWR feature.

Key flush

The KEY register bank may be flushed (filled with random data) by asserting the SGI_CTRL2[KEY_FLUSH] input at any time. This will flush the key and also update the physical shuffling mapping.

Concurrent access

The KEY register bank cannot be accessed while SGI_STATUS[BUSY] is high. Any write attempt is simply ignored. Reading the key register bank during busy (SGI_STATUS[BUSY] is 1) returns a random value (see Security alerts).

Access to non-existent KEY

When attempting an access to a non-existent KEY bank, that is, when i_addr_key is greater than SGI_NUM_KEY, the KEY[0] bank will be accessed. No security alert will be triggered in this case as illegal SFR accesses should be handled in the platform-specific wrapper.

15.3.2.5 Register bank matrix

The following table shows some of the main characteristics of the three register banks (DATIN, DATOUT and KEY).

Table 96. Register bank matrix

Register Bank	FLUSHWR	XORWR	INCR	Concurrent Access	Physical Shuffling	Shares	EDC	Configurable Size	Access
DATIN	Y	Y	Y	Y	Y	2	Y	Y	RW
DATOUT	N/A	N/A	N/A	Y	Y	2	Y	N	RO
KEY	Y	Y	N	N	Y	2	Y	Y	RW

EDC faults are detected when the faulted register is used (written in case on incr/xorwr, or read, or used as input for computation). In other words, there is no continuous detection of faults on all register banks, but in some cases the detection my happen immediately depending on the state of internal address signals used by the register banks.

15.3.3 AUTO mode

AUTO mode is basically a mode which automates the execution of the crypto algorithms supported by the SGI. Without auto-mode, the SGI requires FW (or crypto-lib software) to configure and start each block of a crypto operation such as a CBC chaining mode operation for example. With auto-mode, the idea is that after initial setup by FW, the SGI simply needs to be fed with the stream of data (i.e plaintext) while at the same time the result (i.e. ciphertext) can be read from the SGI without the FW having to do anything else. In fact, the data movement to/from SGI can then be offloaded to a system DMA to improve throughput and to free-up the CPU.

The SGI AUTO mode is controlled via the SGI_AUTO_MODE SFR. Please refer to the SFR sections for information on the options.

15.3.3.1 Data transfer order

The SGI auto-mode expects the input data to be written first before any output access is attempted. If an output access (i.e. read to DATOUT) is attempted before the writing of any data that the SGI would lock the CPU bus.

15.3.3.2 Partial mode

SGI must support partial mode of operation.

Partial processing is the technique of splitting the input data into "chunks", and processing each chunk with a separate call of cipher commands. With partial mode, an host can prioritize which data to be processed first.

Partial processing avoids that by allowing a single command that processes a lot of data to be split into a series of calls to the same command with each call processing a subset of the data.

During partial mode, the host can store the status information externally but it should be possible to keep in the SGI such state.

ECB doesn't have a concept of state and therefore execution can be started and stopped on different chunks without the need to load anything except the data to be processed.

CBC encryption/decryption has a concept of status that is intrinsic to the data to be/that has been processed itself (being the previous block processed or the result itself). Since this information is commonly stored in memory itself, there is no need to load anything specific.

CTR mode is the only mode where the status information (the counter value) is not linked to the data and therefore the only one that can be saved in memory.

15.3.3.3 Operation sequencing

The sequence of operations that a CPU or a DMA should follow to use the automode is described in the following sections.

15.3.3.3.1 ECB mode

To start a sequence of ECB automode for both encryption and decryption:

1. Set the SGI_AUTO_MODE[AUTO_MODE_EN] bit and the SGI_AUTO_MODE[CMD] to 0x00
2. Set the SGI_CTRL[START] bit (as well as all the other settings (SGI_CTRL[DECRYPT], SGI_CTRL[AES_EN], etc.).

> **NOTE**
>
> SGI_CTRL[DATOUT_RES] must be set to NO_UP for the first operation.

3. DMA transfer to DATINx (1st data to be processed)
4. Loop DMA
    a. DMA write access to DATINx
    b. DMA read access to DATOUT

> **NOTE**
>
> Considering that ECB doesn't have any dependence between one block and the other meaning that any DATINx register bank can be used and changed during the same automode session.

To finish a sequence of ECB automode:

1. Reset the SGI_AUTO_MODE[AUTO_MODE_EN] bit
2. Set the SGI_AUTO_MODE[AUTO_MODE_STOP] bit
3. DMA read access to DATOUT to collect the last data that has been processed

15.3.3.3.2 CBC encryption

To start a sequence of CBC encryption in automode:

1. Set the SGI_AUTO_MODE[AUTO_MODE_EN] bit and the SGI_AUTO_MODE[CMD] to 0x02
2. Set the SGI_AUTO_MODE[DECRYPT] bit to 0 (must be done before loading IV)
3. DMA transfer to DATOUT of the IV value
4. Set the SGI_CTRL[START] bit (as well as all the other settings (SGI_CTRL[AES_EN], etc.).

NOTE

SGI_CTRL[DATOUT_RES] must be set to NO_UP for the first operation.

5. DMA transfer to DATIN0 (1st data to be processed)
6. Loop DMA
    a. DMA write access to DATIN0 of the [i] data to be processed
    b. DMA read access to DATOUT to get the result of the [i-1] operation

To finish a sequence of CBC encryption automode:

1. Reset the SGI_AUTO_MODE[AUTO_MODE_EN] bit
2. Set the SGI_AUTO_MODE[AUTO_MODE_STOP] bit
3. DMA read access to DATOUT to collect the last data that has been processed

NOTE

SGI internally uses DATIN[0] and DATIN[1] for CBC encryption so any data in these registers will be lost after this operation.

15.3.3.3.3 CBC decryption

To start a sequence of CBC decryption in automode:

1. Set the SGI_AUTO_MODE[AUTO_MODE_EN] bit and the SGI_AUTO_MODE[CMD] to 0x02
2. Set the SGI_AUTO_MODE[DECRYPT] bit to 1 (must be done before loading IV)
3. DMA write access to DATIN0 of the IV value
4. Set the SGI_CTRL[START] bit (as well as all the other settings (SGI_CTRL[AES_EN], etc.).

NOTE

SGI_CTRL[DATOUT_RES] must be set to NO_UP for the first operation.

5. DMA transfer to DATIN0 (1st data to be processed)
6. Loop DMA
    a. DMA write access to DATIN0 of the [i] data to be processed
    b. DMA read access to DATOUT to get the result of the [i-1] opeation

To finish a sequence of CBC encryption automode:

1. Reset the SGI_AUTO_MODE[AUTO_MODE_EN] bit
2. Set the SGI_AUTO_MODE[AUTO_MODE_STOP] bit
3. DMA read access to DATOUT to collect the last data that has been processed

NOTE

SGI internally uses DATIN[0], DATIN[1] and DATIN[2] for CBC decryption so any data in these registers will be lost after this operation.

15.3.3.3.4 CBCMAC mode

To start a sequence of CBCMAC in automode:

1. Set the SGI_AUTO_MODE[AUTO_MODE_EN] bit and the SGI_AUTO_MODE[CMD] to 0x03
2. DMA transfer to DATOUT of the IV value
3. Set the SGI_CTRL[START] bit (as well as all the other settings (SGI_CTRL[DECRYPT], SGI_CTRL[AES_EN], etc.).

NOTE

SGI_CTRL[DATOUT_RES] must be set to NO_UP for the first operation.

4. Loop of DMA write access to DATIN0 of the [i] data to be processed

To finish a sequence of CBCMAC automode:

1. Reset the SGI_AUTO_MODE[AUTO_MODE_EN] bit
2. Set the SGI_AUTO_MODE[AUTO_MODE_STOP] bit
3. DMA read access to DATOUT to collect CBCMAC result

NOTE

The difference between CBCMAC and CBC is that you don't need to read the intermediate result of each block for CBCMAC.

NOTE

SGI internally uses DATIN[0] and DATIN[1] for CBCMAC so any data in these registers will be lost after this operation.

15.3.3.3.5 CTR single chunk mode

To start a sequence of CTR automode for both encryption and decryption:

1. Set the SGI_AUTO_MODE[AUTO_MODE_EN] bit and the SGI_AUTO_MODE[CMD] to 0x01.
2. Set the SGI_AUTO_MODE[INCR_MODE] field to the desired increment mode.
3. Set the SGI_CTRL[START] bit (as well as all the other settings (SGI_CTRL[AES_EN], etc.).

NOTE

SGI_CTRL[DATOUT_RES] must be set to NO_UP for the first operation and that decrypt should be left to 0 in CTR mode.

4. DMA transfer to DATIN0 to transfer the counter and nonce value
5. Loop DMA
    a. DMA write access to DATIN1 of data[i] to be processed
    b. DMA read access to DATOUT of the result of the [i] iteraction

To finish a sequence of CTR automode:

1. Reset the SGI_AUTO_MODE[AUTO_MODE_EN] bit
2. Set the SGI_AUTO_MODE[AUTO_MODE_STOP] bit

NOTE

Compared to ECB or CBC modes there is no need to trigger a last read operation for the last data to be processed.

On the other hand the counter crypto operation is performed anyway after the last data load of DATIN1 and the AES kernel would have processed this last data. This is inevitable as a copro operation is started every time DATIN is been loaded.

The SGI doesn't issue any TRIG_UP operation for CTR mode.

15.3.3.3.6 CTR partial mode

To restart a sequence of CTR automode for both encryption and decryption:

1. Set the SGI_AUTO_MODE[AUTO_MODE_EN] bit and the SGI_AUTO_MODE[CMD] to 0x01.
2. Set the SGI_AUTO_MODE[INCR_MODE] field to the desired increment mode
    a. Using 2**32 mode, only DATIN0A will be incremented.
    b. Using 2*64 mode, {DATIN0A, DATIN0B} will be incremented.
    c. Using 2*96 mode, {DATIN0A, DATIN0B, DATIN0C} will be incremented.
    d. Using 2*128 mode, {DATIN0A, DATIN0B, DATIN0C, DATIN0D} will be incremented.
3. Set the SGI_CTRL[START] bit (as well as all the other settings (SGI_CTRL[AES_EN], etc.).

NOTE

SGI_CTRL[DECRYPT] should be left to 0 in CTR mode.

4. Loop DMA
    a. DMA write access to DATIN1 of data[i] to be processed
    b. DMA read access to DATOUT of the result of the [i] iteraction

To finish a sequence of CTR automode:

1. Reset the SGI_AUTO_MODE[AUTO_MODE_EN] bit
2. Set the SGI_AUTO_MODE[AUTO_MODE_STOP] bit

15.3.3.4 AUTO mode and keep key

The SGI auto-mode is compatible with the AES KEEP KEY feature. When using KEEP KEY (via the SGI_CTRL[AES_NO_KL] bit) the SGI will automatically run the first operation in auto-mode without asserting the AES kernels SGI_CTRL[AES_NO_KL] input. For subsequent AES operations the SGI will drive this input to 1.

This basically means that decryption operations in auto-mode will be speeded up when SGI_CTRL[AES_NO_KL] is set, but unlike SGI normal-mode the setting of SGI_CTRL[AES_NO_KL] will have no functional impact (in normal mode misusing this bit will give incorrect results, in auto-mode this is not the case as the 1st operation will always load the actual key from SGI).

15.3.3.5 DMA handshake

DMA Handshake

The SGI auto-mode supports a DMA handshake interface. The SGI outputs a request signal per channel (input and output channel) when the SGI is ready to accept new INPUT data to DATIN or ready to provide new OUTPUT data from DATOUT.

The SGI will keep the request signal asserted until the SGI receives the peripheral bus access to complete the requested write/read.

NOTE

The SGI does not use the DMA done signal but instead relies on the existing mechanism to stop the auto-mode operation (see SGI_AUTO_MODE[AUTO_MODE_STOP]).

The SGI Auto-mode handshake feature can be enabled via the SGI_AUTO_DMA_CTRL SFR.

15.3.3.6 Known limitations of AUTO mode

Known auto-mode limitations

1) SGI must have 2 blocks (2x128) loaded in before attempting the first block read. So for example, a 4-block ECB sequence would be as follows:

→ IN[0]->IN[1]->OUT[0] → IN[2] → OUT[1] → IN[3] → OUT[2] → OUT[3]

NOTE

The above limitation means that the SGI automode DOES NOT work for a single block.

2) After feeding SGI with all input data, SGI must be notified by de-asserting SGI_AUTO_MODE[AUTO_MODE_EN] bit. SGI will continue to process the final blocks and will keep busy asserted. **However there is no way to inform SW that the final result is ready**. Once the final result is read the SGI busy bit will de-assert. While there is no IRQ or BUSY indicator to inform the user of when the final block is ready, the SW can attempt the read at any time and if done too early the SGI will halt the bus until the data is available

3) When disabling auto-mode (i.e. setting SGI_AUTO_MODE[AUTO_MODE_EN] to 0 the user must not change the SGI_AUTO_MODE[CMD] field. The SGI_AUTO_MODE[CMD] field must only be changed when SGI busy is low. Further, please clear the SGI_AUTO_MODE[CMD] field back to 0x00 before using SGI for normal operations (for example a normal AES operation) otherwise the SGI_AUTO_MODE[CMD] setting will interfere with the operation.

15.3.3.7 Key wraping and unwraping

The SGI supports RFC3394 AES Key Wrap Algorithm. The key wrapping and unwrapping can be enabled via the SGI_AUTO_MODE[CMD] field.

NOTE

Key wrapping/unwrapping does not support DMA handshake feature.

NOTE

Key wrapping/unwrapping currently only support 128/256 bits keys.

15.3.3.7.1 Key wraping

To start a sequence of key wrapping

1. Set the SGI_AUTO_MODE[AUTO_MODE_EN] bit and the SGI_AUTO_MODE[CMD] to 0x10(wrap 128-bit key) or 0x11(wrap 256-bit key).
2. Set SGI_CTRL[DECRYPT] to 0.
3. Set the SGI_CTRL[START] bit (as well as the AES key size).
4. CPU/DMA transfer to DATIN0 (wrapping 128-bit key) or DATIN0/DATIN1(wrapping 256-bit key) the key to be wrapped.
5. Poll for Busy to deassert, or use IRQ to detect end of operation.
6. CPU/DMA read the SGI_KEY_WRAP SFR to extract the wrapped key.
7. Reset the SGI_AUTO_MODE[AUTO_MODE_EN] bit.

NOTE

- INSEL is not supported for Key Wrapping and key to be wrapped must be placed starting at DATIN0.
- The contents of DATOUT will be overwritten during the key wrap operation so any data in DATOUT will be lost.
- SGI_CTRL2[RKEY] is ignored for key wrap operation. The result of a key wrap operation cannot be loaded to the key register bank.
- SFRMASK is currently NOT SUPPORTED for reading wrapped key (via SGI_KEY_WRAP).

15.3.3.7.2 Key unwraping

To start a sequence of key wrapping

1. Set the SGI_AUTO_MODE[AUTO_MODE_EN] bit and the SGI_AUTO_MODE[CMD] to 0x10 (wrap 128-bit key) or 0x11 (wrap 256-bit key).
2. Set SGI_CTRL[DECRYPT] to 1.
3. Set the SGI_CTRL[START] bit (as well as the AES key size).
4. CPU/DMA transfer to DATIN0/DATIN1 (unwrapping 128-bit key) or DATIN0/DATIN1/DATIN2 (wrapping 256-bit key) the key to be wrapped.
5. Poll for Busy to deassert, or use IRQ to detect end of operation.
6. Reset the SGI_AUTO_MODE[AUTO_MODE_EN] bit.
7. The wrapped KEY will be available in the key register bank that has been configured at compile time.

NOTE

- INSEL is not supported for Key Unwrapping and Key to be unwrapped must be placed starting at DATIN0.
- A mismatch in the expected IV will cause a SGI to set the SGI_STATUS[KEY_UNWRAP_ERR] bit. The invalid unwrapped KEY will still be loaded into the KEY regbank.
- The contents of DATOUT will be flushed after the completion of a key unwrapping operation.
- SGI_CTRL2[RKEY] and SGI_CTRL2[KEYRES] are ignored for key unwrap operation. The result of a key unwrap operation cannot be loaded to the DATOUT register bank. It will always be loaded into the KEY bank configured at compile time.
- The KEK (Key Encryption Key) used for unwrapping can be placed in any of the available KEY slots. However, it can only be placed in the same KEY location as the unwrapped key(i.e. destination of unwrapped key) if SGI_CTRL[AES_NO_KL] is set to 1. Otherwise, using the same KEY for KEK and unwrapping destination will result in a corrupted result(wrong unwrapped key).
- The SGI_CTRL[AES_NO_KL] bit can be used with the key unwrapping command to speed up key unwrapping.

15.3.4 Security description

15.3.4.1 Security Features

15.3.4.1.1 Calculation counter

The SGI provides a 32-bit counter to allow the user to determine the number of crypto operations that have been performed. This is reported via the SGI_COUNT[COUNT] output. The counter value can be initialised by writing to the internal count register via the i_wr_count strobe and the i_wr_data[31:0] data bus.

The counter cannot be initialised while SGI_STATUS[BUSY] is 1. The write will be ignored and the counter will remain unchanged.

Reading the counter (SGI_COUNT[COUNT]) while SGI_STATUS[BUSY] is 1 will trigger a security alert (see Table 98)

Table 97 shows the increment value of the counter for the different operations.

Table 97. Increment value of the counter for the different operations

Operation	Increment
AES	1
DES	1
GCM	1

Table continues on the next page...

Table 97. Increment value of the counter for the different operations (continued)

Operation	Increment
TDES	3
SHA (increment for each processed block)	1
CMAC	1

The counter register is reset to 0 every time there is a full flush.

15.3.4.1.2 Security alerts

The SGI and the kernels support fault detection countermeasures but the SGI also has the capability to detect invalid use of the SGI, especially in cases where this invalid use may reduce the security of the SGI or give an attacker access to the secret key/plaintext. In both cases (that is, faults or invalid use) the SGI triggers a security alert.

Any use of the SGI that produces unpredictable SGI behaviour is considered as an invalid operation. Some invalid operations directly reduce the security of the SGI, while others may indicate problems with the environment that is driving SGI. The SGI response to all of these cases is the same: A security alert is triggered.

All detected errors are indicated by the SGI_STATUS[ERROR] output signal. These should be used to set corresponding bits in the wrapper status register. The following table covers all such cases of invalid use which will trigger an error. The SGI_STATUS[ERROR] output should be set to 3'b010 if any of the invalid operations in the table below occur, otherwise the output should remain at value 3'b101.

Table 98. Invalid SGI operations

Security alert ID	Invalid operation
SECALERT-1	Asserting any of {SGI_CTRL[START], SGI_CTRL2[FLUSH], SGI_CTRL2[KEY_FLUSH], SGI_CTRL2[DATIN_FLUSH]} during any of {start-up phase, full flush}
SECALERT-2	Accessing any register bank during start-up phase or a full flush
SECALERT-3	Reading SGI count while SGI_STATUS[BUSY] = 1
SECALERT-4	Reading DATOUT or CHECKSUM while SGI_STATUS[BUSY] = 1 when using END_UP mode
SECALERT-4.1	Writing of DATOUT or CHECKSUM while SGI_STATUS[BUSY] = 1 (if writing DATOUT/CHECKSUM is supported)
SECALERT-5	Writing a DATIN register that will be used for OUTSEL while the SGI is performing a crypto operation (i.e. SGI_STATUS[BUSY] = 1 and i_datout_res==END_UP and OUTSEL!=0). Please note that the used DATIN is based on the SGI_NUM_DATIN setting and not only on OUTSEL as explained in OUTSEL - (DATOUT input configuration)
SECALERT-6	Accessing DATIN register bank or asserting i_start during a DATIN-only flush (i.e. i_datin_flush)

Table continues on the next page...

Table 98. Invalid SGI operations (continued)

Security alert ID	Invalid operation
SECALERT-7	Accessing KEY register bank or asserting i_start during a KEY-only flush (i.e. i_key_flush)
SECALERT-8	Accessing KEY or DATIN register banks or asserting i_start during a combined KEY/DATIN flush (triggered by asserting both i_key_flush and i_datin_flush together)
SECALERT-9	Triggering a DATIN-only, KEY-only or combined DATIN/KEY flush while SGI_STATUS[BUSY] = 1
SECALERT-10	Starting an operation which has been fused(but is compile time enabled in the configuration)
SECALERT-11	Using a write-only key(see key locking section) as input to GCM operation

NOTE

- The default, non-error state of o_err is 3'b101.
- During an invalid SFR read the SGI should return random values on the read buses, see below.
- During an invalid SFR write, the SFR being written MUST not be updated.
- Accessing DATOUT or DATIN during a KEY-only flush is possible. Similarly, accessing DATOUT during DATIN-only flush is possible. Note that KEY is not accessible during busy so cannot be accessed during a DATIN-only flush.

Security alert response

When a security alert is triggered, the SGI immediately asserts the o_err output and proceeds with an Error Flush (see Terms and definitions). To clear the security alert the system must either reset the SGI or assert the SGI_CTRL2[FLUSH] input.

Attempting to read any of the SGI registers during a security alert will cause the SGI to return random values: DATIN, KEY, DATOUT, or CHECKSUM.

Additionally, if there is a read attempt of DATIN, KEY, DATOUT, or CHECKSUM, the SGI will return zero on the first share and the second share will be fed from a dedicated output of the PRNG (hence, resulting in a random logical output). If COUNT or SFRSEED are accessed for read, the SGI will return a random value from a dedicated output of the PRNG.

Finally, any action to SGI_CTRL2[KEY_FLUSH], SGI_CTRL2[DATIN_FLUSH], i_sfrmask_ctrl (SMASKSTEP, SMASKEN, and SMASKSW) and SGI_CTRL[START] during a security alert will be ignored.

15.3.4.1.3 Flush

The SGI supports flushing of the register banks. During the flush operation one of the shares of the register bank will be loaded with random data (or data on wr_data input bus for UNSECURE version of SGI). The full flush operation is invoked by asserting one of the following inputs SGI_CTRL2[FLUSH] or i_prng_ready. In addition, upon any error or fault detection event (SGI_STATUS[ERROR] != 3'b101), the full flush event is triggered as well. Next to the full flush event, a partial flush can also be triggered by asserting SGI_CTRL2[KEY_FLUSH] in which case only the KEY register bank is flushed or by asserting SGI_CTRL2[DATIN_FLUSH] in which case only the DATIN register bank is flushed. The full flush can be triggered at any time, even during an ongoing crypto operation. In this case the current operation will be terminated.

The flush operation only flushes one share of the register banks.

The number of cycles taken to flush the SGI depends on the configuration of the SGI.

NOTE

The flush operation flushes 32-bits at a time to minimise the peak power consumption and to reduce the flush from being identifiable from a power trace.

NOTE

The order of the flush is implementation specific and not neccarilly in the order shown above (i.e. AES first and Checksum register last)

The SGI_STATUS[BUSY] output will be set to 1 for the duration of the flush.

Reset invoked flush

Following an SGI reset (i.e. assertion of i_rst or i_sync_rst) the SGI will flush all register banks including the AES and DES kernel's internal working registers. The flush however will be done only after the i_prng_rdy input has been asserted. That way we make sure that all the registers are flushed only after the internal PRNG has finished its boot-up phase and is ready for normal usage. This flush operation will be indicated by the SGI_STATUS[BUSY] output.

DATIN only flush

The DATIN register bank may be flushed using the SGI_CTRL2[DATIN_FLUSH] input. This is useful to load the DATIN register bank with dummy data prior to starting a dummy operation, for example in the case of generating the reference key checksum. Refer to Table 98 for other rules regarding DATIN only flush.

KEY only flush

The KEY register bank is flushed as part of the flush operation. In order to flush ONLY the key register, the SGI_CTRL2[DATIN_FLUSH] input (at SGI-core level) is used. Refer to Table 98 for other rules regarding KEY only flush.

The KEY flush can also be triggered by the SGI_CTRL2[DATIN_FLUSH] input (SGI-top level). This input should be asserted for a single SGI clock (i_sgi_clk).

Flush matrix

The table below shows the different register banks that are flushed with the different flush options.

Table 99. Different flush options

Flush input	Flushed register banks
SGI_CTRL2[FLUSH] (full flush)	DATIN DATOUT KEY Key checksum register
SGI_CTRL2[KEY_FLUSH]	KEY
SGI_CTRL2[DATIN_FLUSH]	DATIN
SGI_CTRL2[KEY_FLUSH] + SGI_CTRL2[DATIN_FLUSH]	KEY + DATIN

Flush priority

When asserting multiple flush inputs (i.e. SGI_CTRL2[FLUSH], SGI_CTRL2[KEY_FLUSH], SGI_CTRL2[DATIN_FLUSH]) simultaneously the priority will be as follows:

i_flush-1, i_key_flush-2, i_datin_flush-2

NOTE

1 indicates highest priority and 2 the lowest

This means that when all 3 are asserted simultaneously, a full flush will be performed. If both SGI_CTRL2[KEY_FLUSH] and SGI_CTRL2[DATIN_FLUSH] are asserted simultaneously (without SGI_CTRL2[FLUSH] being asserted), both KEY and DATIN will be flushed.

The above sentence clarifies the priority between the 3 flush signals. The following note clarifies the behaviour when a SGI_CTRL2[FLUSHWR] is triggered during an ongoing flush.

NOTE

Triggering a flush write (i.e writing to a register bank while SGI_CTRL2[FLUSHWR] is 1) will trigger an error if that particular register bank is being flushed as covered in Table 98.

Further, if SGI_CTRL[START] is asserted simultaneously with any of the 3 flush inputs (i.e. SGI_CTRL2[FLUSH], SGI_CTRL2[KEY_FLUSH], SGI_CTRL2[DATIN_FLUSH]), the SGI_CTRL[START] will simply be ignored.

Flush Write (FLUSHWR)

The SGI supports loading of random data (or write data if UNSECURE version of SGI) into selected register banks by simply writing (i.e. via the primary inputs to SGI) to the register while the SGI_CTRL2[FLUSHWR] input is 1. This allows the written register, be it a 8/16/32 bit write to load the written register with random data.

15.3.4.1.4 SFRMASK feature

SFRMASK

The SFRMASK feature allows the user to write masked data into the SGI but have the SGI unmask the data before it is loaded into the regbanks. Likewise, the SGI can apply a mask on the read path of the register banks so that the data in the regbanks is masked before being sent out of the SGI. The details of the SFRMASK as well as the masking algorithm is described in Appendix A - SFRMASK.

The masking applies to all three register banks, namely the DATIN, KEY and DATOUT regbanks as well as the KEYCHK register (regardless of whether the KEYCHK is avaliable: i.e. (ID_CFG_SGI_NO_KEYCHECKSUM=1). The masking **IS NOT** applied to the COUNT or SFRSEED registers.

NOTE

When SGI_BUS_WIDTH=16 (i.e. SGI_CONFIG[BUS_WIDTH]=0), the 16-bit SFRMASK is duplicated AND byte swapped before being applied to the 32-bit read buses of the register banks as illustrated by the code below:

```
if (SGI_BUS_WIDTH=16)
  rd_data = rd_regbank ^ {2 {sfrmask[7:0], sfrmask[15:8]} };
```

The SFRMASK feature allows secure data transfer between the CPU and the SGI register bank by utilising a deterministic random number (mask) generator within the SGI module. The mask generator can be seeded in order to re-generate the same sequence of mask values. The figure below illustrates the SFRMASK feature.

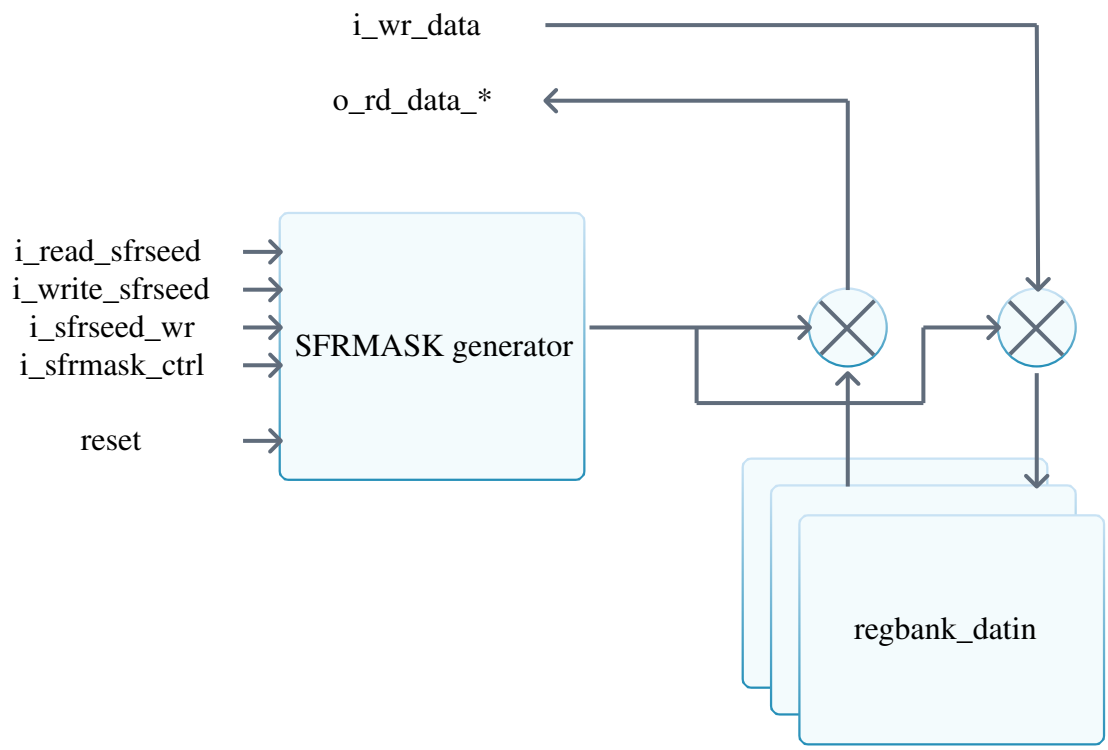


Figure 48. SFRMASK

Table 100. SFRMASK control bits

i_sfrmask_ctrl	Name
i_sfrmask_ctrl[1]	SMASKSTEP
i_sfrmask_ctrl[0]	SMASKEN
i_sfrmask_ctrl[2]	SMASKSW

The SFRMASK generator generates a stream of random values that are used to mask the data from SGI to CPU or unmask the data from CPU to SGI(within the SGI). The generator can be seeded by SW via the SGI_SFRSEED register. When SMASKSW = 1, the SW will manually update the generated mask by directly controlling the seed. When SMASKSW = 0, the generator will provide a fresh mask for every SFR access to the register bank. However, by setting SMASKSTEP to 1, the SFRSEED can also be updated by reading SGI_SFRSEED.

Refer to Appendix A - SFRMASK for further details on the algorithm for computing the SFRMASK mask output.

SFRSEED update

The SGI_SFRSEED SFR is updated whenever the following any of the following registers is read:

{DATIN , DATOUT , KEY , CHECKSUM}

The SGI_SFRSEED SFR is updated whenever the following any of the following registers is written:

{DATIN , DATOUT , KEY , CHECKSUM}

The SGI_SFRSEED SFR is updated regardless of whether the SFRMASK feature is enabled (i.e. SMASKEN= 0). SMASKEN is only used to enable the MASK output from the generator, when SMASKEN is 0 the mask is forced to 0.

The SFRSEED is also updated when the SGI_SFRSEED SFR is read, this is only the case when SMASKSTEP is 1.

NOTE

For SoCs that use Secure Peripheral Bus pmode functionality, SFRSEED will be updated regardless of the access rights of the registers that trigger an SFRSEED access. For example accessing DATIN in SU mode will update SFRSEED even if DATIN itslef may not be accessible in SU mode.

SFRSEED read

The read back of the SGI_SFRSEED SFR is controlled as follows:

When SMASKSW=1, the SGI_SFRSEED SFR reads back as zero;

When SMASKSW=0, If SMASKEN=0, a read of SFRSEED returns the random mask output of the SFRMASK generator, otherwise it returns the value in SFRSEED.

Generated mask

The generated mask used to mask/unmask the data being read/written is always generated based on the current SFRSEED value, not the updated value of SFRSEED that is caused by the write/read action.

15.3.4.1.5 IDLE dummy operations

The SGI can be used to act as a noise source by enabling the AES or DES kernels via the i_crypto_en input. This activates dummy operations within the kernels, the dummy operation continues until a crypto operation is started or the kernel is disabled. Although the SGI does not allow both the AES and DES kernels to be used simultaneously to perform crypto operations, the two kernels can however be enabled simultaneously. One of the kernels (for example AES kernel) can actually be enabled at the same time as the other kernel (for example DES kernel) is performing a computation.

15.3.4.1.6 Key locking (write-only)

The SGI provides a 32-bit SFR (SGI_KEY_CTRL) to allow locking of KEY SFRs to be write-only. The feature works as described below:

- SGI_KEY_CTRL SFR has 32-bit SGI_KEY_CTRL[KEY_WO] field
  - Each bit corresponds to one of the 32 KEY SFRs (SGI_KEY0A -> SGI_KEY7D): bit[0] maps to KEY0A and bit[31] maps to KEY7D
  - Setting of 1'b0 makes KEY SFRs read/writable (this is the reset condition)
  - Setting of 1'b1 makes KEY SFRs write-only
- The SGI_KEY_CTRL SFR has following behaviour
  - Clears to 0 whenever the following occurs:
    1. Reset
    2. User requested flush (SGI_CTRL[FLUSH] is set)
    3. User requested key flush (SGI_CTRL2[KEY_FLUSH] is set
  - SGI_KEY_CTRL SFR is read/writable
  - SGI_KEY_CTRL[KEY_WO] field can only be written to 1 (SW cannot write to 0, i.e. cannot make key readable after its made write-only)
- When attempting a read of a write-only KEY SFR, the following occurs:
  1. The SFR returns random data on bus (or fixed constant in NON-SCA configs)
  2. The SGI sets the SGI_STATUS[KEY_READ_ERR] but this bit is cleared on user flush (SGI_CTRL2[FLUSH] is set)

A20 SGI has KEY4 to KEY7 as permanently write-only.

NOTE

In cases where SGI supports a GCM, the locked keys cannot be used as input for the GCM operation.

15.3.4.2 Appendix A - SFRMASK

The SFRMASK generator consists of a SEED SFR that can be initialised/updated by SW. The SEED SFR is either 16-bit or 32-bit depending on the setting of the SGI_BUS_WIDTH parameter.

The SEED can also be updated by HW whenever there is an SFR access to the SGI 'register bank' or the KEYCHK register (HW can only update the SEED when SMASKSW = 0, also see 'HW SEED UPDATE' section below).

Note that KEYCHK accesses cause SFRSEED to increment even if the KEYCHK feature is not available (ID_CFG_SGI_NO_KEYCHECKSUM=1)

When SMASKSW = 1, The SEED is used to directly set the mask output of the SFRMASK generator.

When SMASKSW = 0, The SEED is used as input to a 'function' that computes the mask output of the SFRMASK generator. The 'function' is described in the 'MASK FUNCTION' section below.

15.3.5 Power Clock Reset description

Clocks

The SGI is designed to work in a system that supports multiple clocks that can be divided independently using clock pulse stealing. The SGI takes as input a single clock, called sgi clock (i_clk_sgi) and a sfr clock enable (i_sfr_cken) as shown in the figure below. This allows the SGI to use a single clock throughout yet support a sfr clock that is operating at a different frequency.

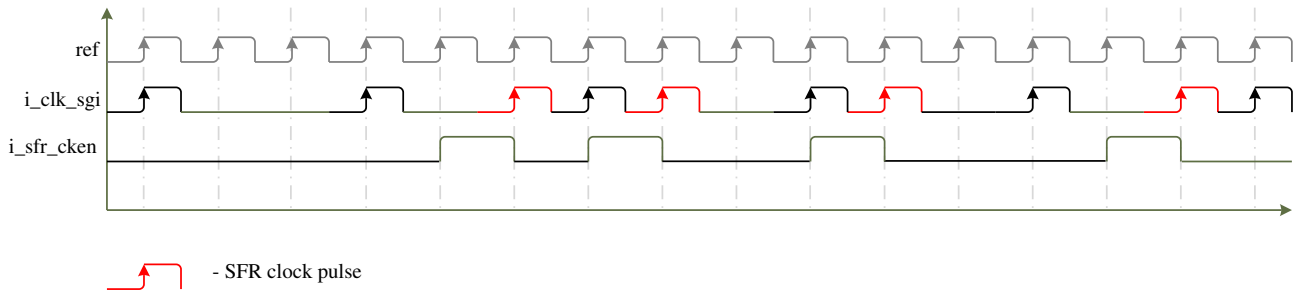


Figure 49. SGI clock and SFR clock enable timing diagram

NOTE

The sgi clock is used to clock the internal logic of the SGI as well as the kernels, so SFR accesses during a crypto operation where the sgi clock is slower than the sfr clock will result in increasing the frequency of the sgi clock as each SFR access forces a clock pulse on the sgi clock

Resets

The SGI supports an asynchronous reset (i_rst). This is duplicated to protect against fault attacks. The signal is active-high. The SGI also contains non-reset registers.

The reset initialises the SGI internal registers, however the register banks are not reset and neither are the AES and DES kernel's data and key working registers. The following table provides the reset value of the SGI register banks as well as some of the SGI output ports (those that may be used to drive a STATUS SFR).

Table 101. Table showing reset values for internal SGI registers

Register	Reset Value
DATIN register bank	Not Reset (but is flushed)
DATOUT register bank	Not Reset (but is flushed)

Table continues on the next page...

Table 101. Table showing reset values for internal SGI registers (continued)

Register	Reset Value
KEY register bank	Not Reset (but is flushed)
Key checksum register	Not Reset (but is flushed)
Calculation counter[15:0] (source for SGI_COUNT[COUNT])	16'h0000
SFRSEED (source for SGI_SFRSEED[SFRSEED])	16'h0000/32'h00000000
busy (source for SGI_STATUS[BUSY])	1'b1
sha_busy (source for SGI_STATUS[SHA2_BUSY])	1'b0
error (source for SGI_STATUS[ERROR])	3'b101

15.4 Application information

15.4.1 CBC mode

CBC chaining mode is described in "NIST Special Publication 800-38A".

CBC encryption

```
- N="NUMBER OF CBC SUB OPS"
- OUTSEL=0x0
- load(KEY  , key        )
- load(DATIN, iv      , 0)
- load(DATIN, plain(1), 1)                          // use xorwr
- fori(1,N+1)
  - (i=1)   ? UP=NO_UP : START_UP
  - (i<N+1) ? strt(ENC, UP, (i==1) ? 0x0 : 0x1)  // insel=0x0 for first iteration
  - (i<N)   ? load(DATIN[0], plain(i+1), 0)        // for all but last operation
  - (i>1)   ? read(DATOUT)                         // from second operation onwards
  - (i<N+1) ? wait()
- getd(TRIGGER_UP)                                          //Triggers the loading of DATOUT with
kernel result
- read(DATOUT)                                     //read final data
```

The CBC encryption mode uses the XORWR function to the DATIN (IV xor Plaintext 1).

Starting with second iteration the DATIN xor DATOUT (Ciphertext1 xor Plaintext2) is used by setting INSEL=0x1.

The load of the next Plaintext can be done concurrent to a running crypto-operation.

Table 102. CBC encryption - SGI xor use

	XORWR	INSEL	OUTSEL
iteration(i)	0	1	-

CBC decryption (based on a SGI_NUM_DATIN=2 configuration)

```
- N="NUMBER OF CBC SUB OPS"
- OUTSEL=0x1
- load(KEY  , key
- load(DATIN, cipher(1), 0)
- fori(1,N+1)
  - (i=1)   ? UP=NO_UP : START_UP
  - (i<N+1) ? strt(DEC, UP, 0)
  - (i<N)   ? load(DATIN[0], cipher(i+1), 0)              //For all but last operation and trigger
```

```
DATOUT
  - (i>1)   ? read(DATOUT)                                  //from second operation onwards
  -           load(DATIN[1] , (i==1) ? iv : cipher(i-1) ,  0)
  - (i<N+1) ? wait())
- getd()                                                  //Triggers the loading of DATOUT with kernel
result
- read(DATOUT)                                           //read DATOUT
```

The CBC decryption mode uses the OUTSEL function selected to **Kernel Res ^ DATIN[1]**, this results in a xor between PLAINTEXT(i) and OUTPUT BLOCK(i).

The load of the next Ciphertext can be done concurrent to a running crypto-operation.

NOTE

A SGI_NUM_DATIN>2 configuration would allow the ciphertext to be loaded only once

Table 103. CBC decryption - SGI xor use

	XORWR	INSEL	OUTSEL
iteration(i)	-	-	all

15.4.2 CFB mode

CFB chaining mode is described in "NIST Special Publication 800-38A".

CFB encryption

The CFB encryption mode uses the OUTSEL function selected to **Kernel Res ^ DATIN[0]**, this results in a xor between PLAINTEXT(i) and OUTPUT BLOCK(i).

Starting with second iteration the input to the kernels is selected to DATOUT (result of previous iteration), INSEL=0x8.

The load of the Plaintext can be done concurrent to a running crypto-operation.

Table 104. CFB encryption - SGI xor use

	XORWR	INSEL	OUTSEL
iteration(i)	-	>1	all

CFB decryption

The CFB decryption mode uses the OUTSEL function selected to **Kernel Res ^ DATIN[0]**, this results in a xor between PLAINTEXT(i) and OUTPUT BLOCK(i).

The load of the Ciphertext can be done concurrent to a running crypto-operation.

Table 105. CFB decryption - SGI xor use

	XORWR	INSEL	OUTSEL
iteration(i)	-	-	all

15.4.3 OFB mode

OFB chaining mode is described in "NIST Special Publication 800-38A".

OFB encryption

The OFB encryption mode uses the OUTSEL function selected to **Kernel Res ^ DATIN[1]**, this results in a xor between PLAINTEXT(i) and OUTPUT BLOCK(i)

Starting with second iteration the input to the kernels is selected to DATOUT (result of previous iteration), INSEL=0x2.

The load of the Plaintext can be done concurrent to a running crypto-operation.

NOTE

Starting with the second iteration, the input of the kernel is kernels-out(0) xor Plaintext(1). This xor is done twice, once for generating Ciphertext(1) and second to generate the input for second kernel operation.

Table 106. OFB encryption - SGI xor use

	XORWR	INSEL	OUTSEL
iteration(i)	-	>1	all

OFB decryption

The OFB decryption mode uses the OUTSEL function selected to **Kernel Res ^ DATIN[1]**, this results in a xor between PLAINTEXT(i) and OUTPUT BLOCK(i).

The load of the Ciphertext can be done concurrent to a running crypto-operation.

Table 107. OFB decryption - SGI xor use

	XORWR	INSEL	OUTSEL
iteration(i)	-	>1	all

15.4.4 CTR (Counter) mode

CTR chaining mode is described in "NIST Special Publication 800-38A".

CTR encryption

The CTR encryption mode uses the OUTSEL function selected to **Kernel Res ^ DATIN[1]**, this results in a xor between PLAINTEXT(i) and OUTPUT BLOCK(i).

The load of the Plaintext and next counter value can be done concurrent to a running crypto-operation.

Table 108. CFB encryption - SGI xor use

	XORWR	INSEL	OUTSEL
iteration(i)	-	-	all

CTR decryption

The CTR decryption mode uses the OUTSEL function selected to **Kernel Res ^ DATIN[1]**, this results in a xor between PLAINTEXT(i) and OUTPUT BLOCK(i).

The load of the Ciphertext can be done concurrent to a running crypto-operation.

Table 109. CFB decryption - SGI xor use

	XORWR	INSEL	OUTSEL
iteration(i)	-	-	all

15.4.5 CMAC/GCM and INCR

The two AES modes CMAC ("AES-CMAC rfc 4493") and GCM can be easily implemented using the xor-rw, outsel and insel features of the SGI. This will minimize the xor-operations needed by sw.

INCR

When INCR is used for the CTR-DRBG mode, we need to prevent the counter overflow since, if visible in a single-trace Simple Power Analysis (SPA), the overflow might reveal information about the counter value which is supposed to be secret in this mode

of operation. In order to prevent that, the increment will always be executed by SW by calling the SGI INCR function N times, depending on the size of the operand.

For the first increment, FW will set SGI_CTRL2[INCR_CIN] to 1'b0 to increment the LS-Word of the operand. For subsequent increments, the FW will set SGI_CTRL2[INCR_CIN] to 1'b1 and based on the output carry (indicated by SGI_STATUS[OFLOW]) the SGI will either increment the remaining words or not.

NOTE

During the increment operation, if the operand is smaller than 128 bits, the SW needs to fill the unused bits with random dummy data. This will make sure that the dummy increment is not distinguishable from the real one by observing a single power trace.

15.5 Registers description (SFRs)

Please ignore the **Reset Value** field for tables detailing the SGI_STATUS SFR. This SFR is connected to outputs of the SGI core and the value of this SFR will vary after reset. The value of this SFR after reset will progress through the following values: {0x00000029 -> 0x0000002D -> 0x0000002C}. This is due to BUSY de-assertion after reset and PRNG_READY asserting.

SGI_STATUS -> Please note that SGI_STATUS[ERROR] cannot be cleared, the user must still use RESET or FLUSH for clearing error state. Also, please note that when writing SGI_STATUS[ERROR] to an ERROR values (val!=d'5) you cannot set the SGI_STATUS[OFLOW] bit to 1 at the same time. This should not be an issue since there should be no need to restore SGI_STATUS[OFLOW] for a context where SGI is in ERROR.

15.5.1 register descriptions

15.5.1.1 id_ips_sgi memory map

SGI0 base address: 400E_B000h

Offset	Register	Width (In bits)	Access	Reset value
200h	Input Data register 0 - Word-3 (SGI_DATIN0A)	32	RW	See section
204h	Input Data register 0 - Word-2 (SGI_DATIN0B)	32	RW	See section
208h	Input Data register 0 - Word-1 (SGI_DATIN0C)	32	RW	See section
20Ch	Input Data register 0 - Word-0 (SGI_DATIN0D)	32	RW	See section
210h	Input Data register 1 - Word-3 (SGI_DATIN1A)	32	RW	See section
214h	Input Data register 1 - Word-2 (SGI_DATIN1B)	32	RW	See section
218h	Input Data register 1 - Word-1 (SGI_DATIN1C)	32	RW	See section
21Ch	Input Data register 1 - Word-0 (SGI_DATIN1D)	32	RW	See section
220h	Input Data register 2 - Word-3 (SGI_DATIN2A)	32	RW	See section
224h	Input Data register 2 - Word-2 (SGI_DATIN2B)	32	RW	See section
228h	Input Data register 2 - Word-1 (SGI_DATIN2C)	32	RW	See section
22Ch	Input Data register 2 - Word-0 (SGI_DATIN2D)	32	RW	See section
230h	Input Data register 3 - Word-3 (SGI_DATIN3A)	32	RW	See section
234h	Input Data register 3 - Word-2 (SGI_DATIN3B)	32	RW	See section

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
238h	Input Data register 3 - Word-1 (SGI_DATIN3C)	32	RW	See section
23Ch	Input Data register 3 - Word-0 (SGI_DATIN3D)	32	RW	See section
240h	Input Key register 0 - Word-3 (SGI_KEY0A)	32	RW	See section
244h	Input Key register 0 - Word-2 (SGI_KEY0B)	32	RW	See section
248h	Input Key register 0 - Word-1 (SGI_KEY0C)	32	RW	See section
24Ch	Input Key register 0 - Word-0 (SGI_KEY0D)	32	RW	See section
250h	Input Key register 1 - Word-3 (SGI_KEY1A)	32	RW	See section
254h	Input Key register 1 - Word-2 (SGI_KEY1B)	32	RW	See section
258h	Input Key register 1 - Word-1 (SGI_KEY1C)	32	RW	See section
25Ch	Input Key register 1 - Word-0 (SGI_KEY1D)	32	RW	See section
260h	Input Key register 2 - Word-3 (SGI_KEY2A)	32	RW	See section
264h	Input Key register 2 - Word-2 (SGI_KEY2B)	32	RW	See section
268h	Input Key register 2 - Word-1 (SGI_KEY2C)	32	RW	See section
26Ch	Input Key register 2 - Word-0 (SGI_KEY2D)	32	RW	See section
270h	Input Key register 3 - Word-3 (SGI_KEY3A)	32	RW	See section
274h	Input Key register 3 - Word-2 (SGI_KEY3B)	32	RW	See section
278h	Input Key register 3 - Word-1 (SGI_KEY3C)	32	RW	See section
27Ch	Input Key register 3 - Word-0 (SGI_KEY3D)	32	RW	See section
280h	Input Key register 4 - Word-3 (SGI_KEY4A)	32	RW	See section
284h	Input Key register 4 - Word-2 (SGI_KEY4B)	32	RW	See section
288h	Input Key register 4 - Word-1 (SGI_KEY4C)	32	RW	See section
28Ch	Input Key register 4 - Word-0 (SGI_KEY4D)	32	RW	See section
290h	Input Key register 5 - Word-3 (SGI_KEY5A)	32	RW	See section
294h	Input Key register 5 - Word-2 (SGI_KEY5B)	32	RW	See section
298h	Input Key register 5 - Word-1 (SGI_KEY5C)	32	RW	See section
29Ch	Input Key register 5 - Word-0 (SGI_KEY5D)	32	RW	See section
2A0h	Input Key register 6 - Word-3 (SGI_KEY6A)	32	RW	See section
2A4h	Input Key register 6 - Word-2 (SGI_KEY6B)	32	RW	See section
2A8h	Input Key register 6 - Word-1 (SGI_KEY6C)	32	RW	See section
2ACh	Input Key register 6 - Word-0 (SGI_KEY6D)	32	RW	See section
2B0h	Input Key register 7 - Word-3 (SGI_KEY7A)	32	RW	See section

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
2B4h	Input Key register 7 - Word-2 (SGI_KEY7B)	32	RW	See section
2B8h	Input Key register 7 - Word-1 (SGI_KEY7C)	32	RW	See section
2BCh	Input Key register 7 - Word-0 (SGI_KEY7D)	32	RW	See section
2C0h	Output Data register - Word-3 (SGI_DATOUTA)	32	RW	See section
2C4h	Output Data register - Word-2 (SGI_DATOUTB)	32	RW	See section
2C8h	Output Data register - Word-1 (SGI_DATOUTC)	32	RW	See section
2CCh	Output Data register - Word-0 (SGI_DATOUTD)	32	RW	See section
C00h	Status register (SGI_STATUS)	32	RW	0000_0000h
C04h	Calculation counter (SGI_COUNT)	32	RW	0000_0000h
C08h	Key checksum register (SGI_KEYCHK)	32	RW	0000_0000h
D00h	SGI Control register (SGI_CTRL)	32	RW	0000_0000h
D04h	SGI Control register 2 (SGI_CTRL2)	32	RW	0000_0000h
D08h	Configuration of dummy controls (SGI_DUMMY_CTRL)	32	RW	0000_0000h
D0Ch	Sofware Assisted Masking register (SGI_SFR_SW_MASK)	32	RW	0000_0000h
D10h	SFRSEED register for SFRMASK feature (SGI_SFRSEED)	32	RW	0000_0000h
D14h	SHA Control Register (SGI_SHA2_CTRL)	32	RW	0000_0F00h
D18h	SHA FIFO lower-bank low (SGI_SHA_FIFO)	32	RW	See section
D1Ch	SHA Configuration Reg (SGI_CONFIG)	32	R	See section
D20h	SHA Configuration 2 Reg (SGI_CONFIG2)	32	R	See section
D24h	SGI Auto Mode Control register (SGI_AUTO_MODE)	32	RW	0000_0000h
D28h	SGI Auto Mode Control register (SGI_AUTO_DMA_CTRL)	32	RW	0000_0000h
D30h	SGI internal PRNG SW seeding register (SGI_PRNG_SW_SEED)	32	RW	0000_0000h
D40h	SGI Key Control SFR (SGI_KEY_CTRL)	32	RW	0000_0000h
D50h	Wrapped key read SFR (SGI_KEY_WRAP)	32	R	0000_0000h
F08h	SGI Version (SGI_VERSION)	32	R	0000_0000h
FC0h	Access Error (SGI_ACCESS_ERR)	32	RW	0000_0000h
FC4h	Clear Access Error (SGI_ACCESS_ERR_CLR)	32	RW	0000_0000h
FE0h	Interrupt status (SGI_INT_STATUS)	32	R	0000_0000h
FE4h	Interrupt enable (SGI_INT_ENABLE)	32	RW	0000_0000h
FE8h	Interrupt status clear (SGI_INT_STATUS_CLR)	32	RW	0000_0000h
FECh	Interrupt status set (SGI_INT_STATUS_SET)	32	RW	0000_0000h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
FFCh	Module ID (SGI_MODULE_ID)	32	R	0000_0000h

15.5.1.2 Input Data register 0 - Word-3 (SGI_DATIN0A)

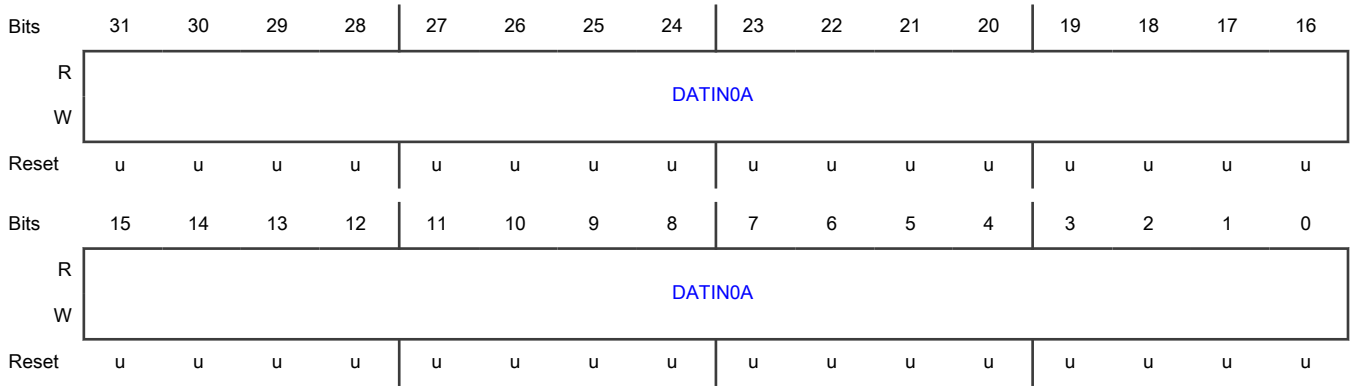
Offset

Register	Offset
SGI_DATIN0A	200h

Function

Input Data register 0 - Word-3

Diagram



Fields

Field	Function
31-0 DATIN0A	Input Data register **NOTE** This field is volatile.

15.5.1.3 Input Data register 0 - Word-2 (SGI_DATIN0B)

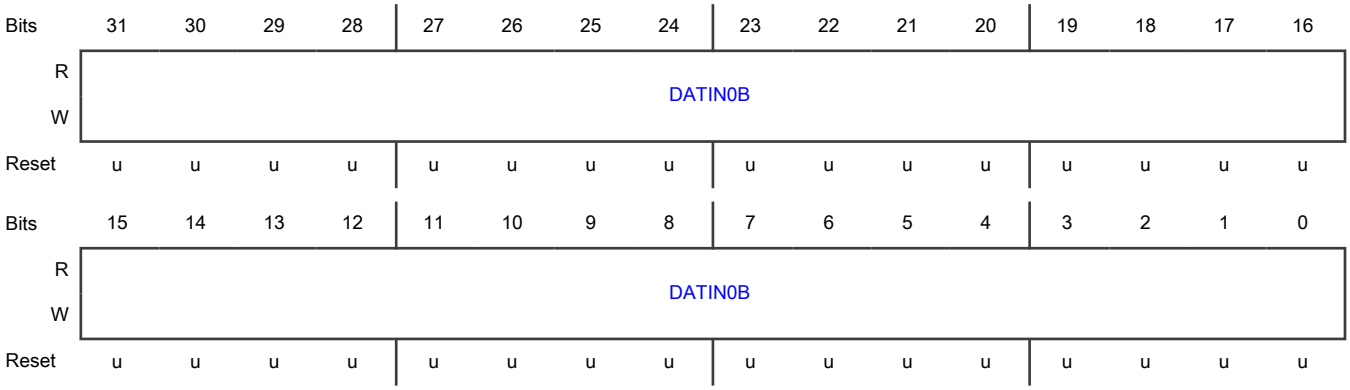
Offset

Register	Offset
SGI_DATIN0B	204h

Function

Input Data register 0 - Word-2

Diagram



Fields

Field	Function
31-0 DATIN0B	Input Data register **NOTE** This field is volatile.

15.5.1.4 Input Data register 0 - Word-1 (SGI_DATIN0C)

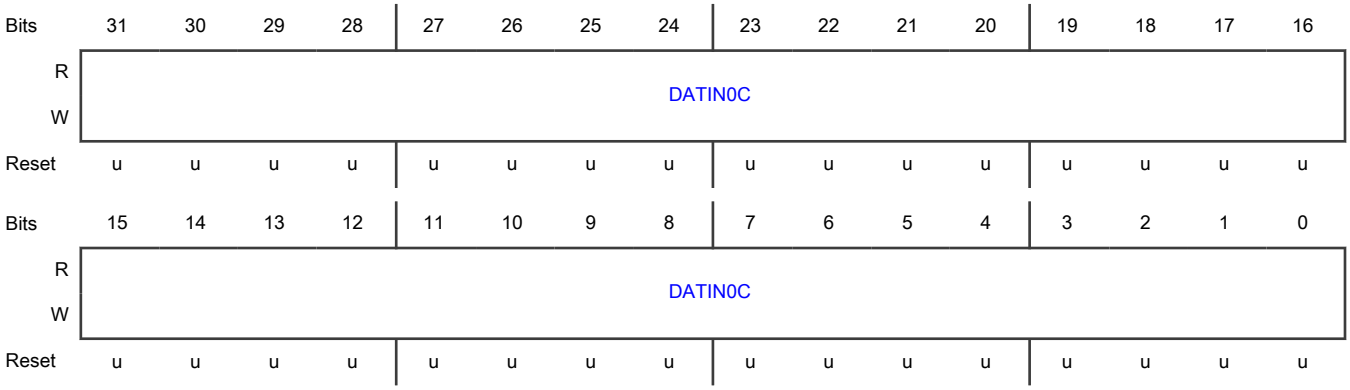
Offset

Register	Offset
SGI_DATIN0C	208h

Function

Input Data register 0 - Word-1

Diagram



Fields

Field	Function
31-0 DATIN0C	Input Data register **NOTE** This field is volatile.

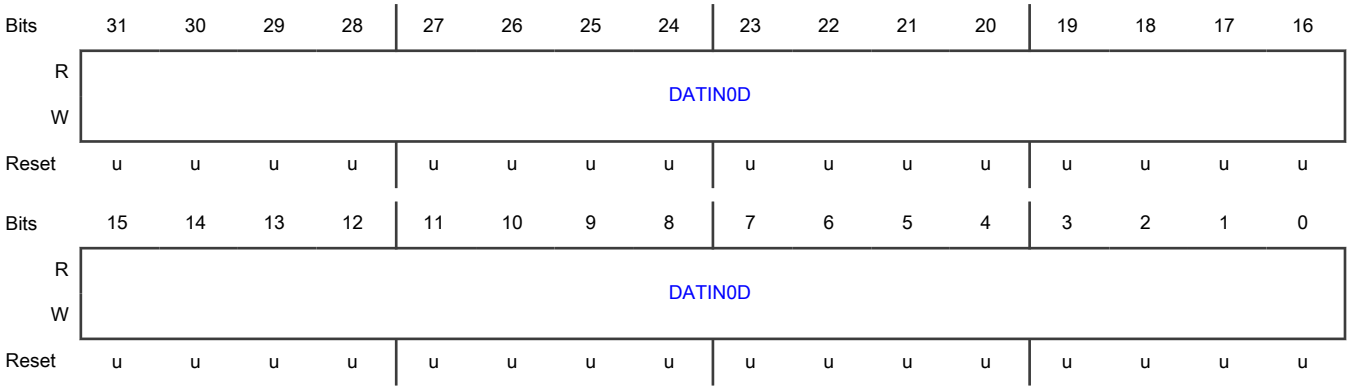
15.5.1.5 Input Data register 0 - Word-0 (SGI_DATIN0D)

Offset

Register	Offset
SGI_DATIN0D	20Ch

Function
Input Data register 0 - Word-0

Diagram



Fields

Field	Function
31-0 DATIN0D	Input Data register **NOTE** This field is volatile.

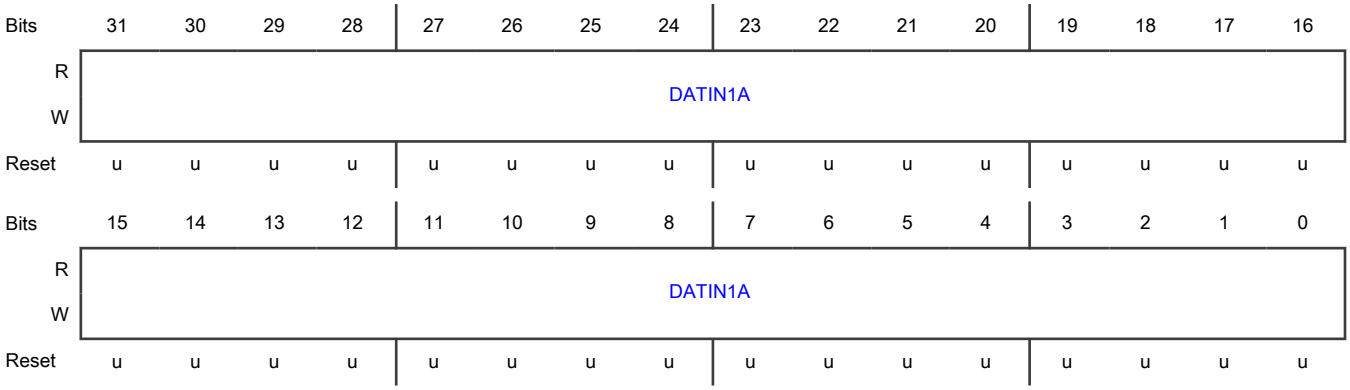
15.5.1.6 Input Data register 1 - Word-3 (SGI_DATIN1A)

Offset

Register	Offset
SGI_DATIN1A	210h

Function
Input Data register 1 - Word-3

Diagram



Fields

Field	Function
31-0 DATIN1A	Input Data register **NOTE** This field is volatile.

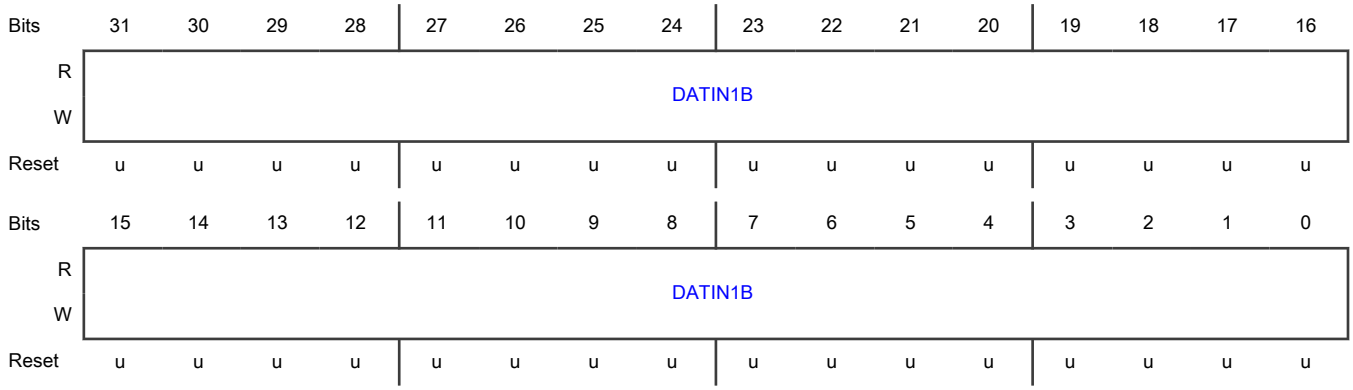
15.5.1.7 Input Data register 1 - Word-2 (SGI_DATIN1B)

Offset

Register	Offset
SGI_DATIN1B	214h

Function
Input Data register 1 - Word-2

Diagram



Fields

Field	Function
31-0 DATIN1B	Input Data register **NOTE** This field is volatile.

15.5.1.8 Input Data register 1 - Word-1 (SGI_DATIN1C)

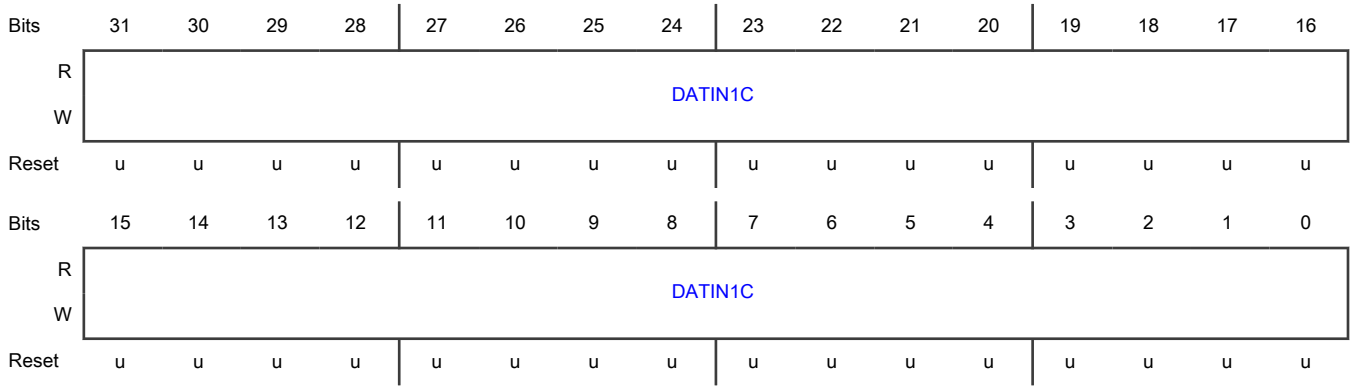
Offset

Register	Offset
SGI_DATIN1C	218h

Function

Input Data register 1 - Word-1

Diagram



Fields

Field	Function
31-0 DATIN1C	Input Data register **NOTE** This field is volatile.

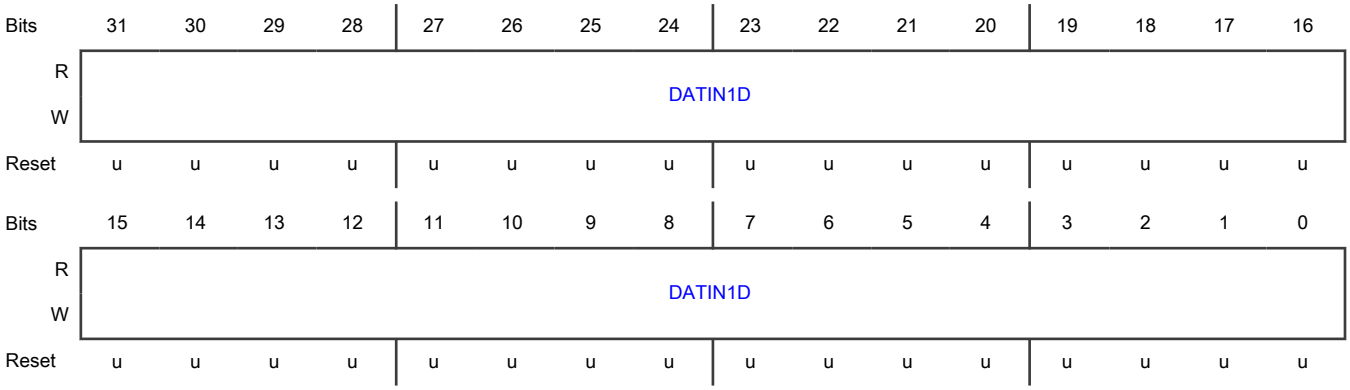
15.5.1.9 Input Data register 1 - Word-0 (SGI_DATIN1D)

Offset

Register	Offset
SGI_DATIN1D	21Ch

Function
Input Data register 1 - Word-0

Diagram



Fields

Field	Function
31-0 DATIN1D	Input Data register **NOTE** This field is volatile.

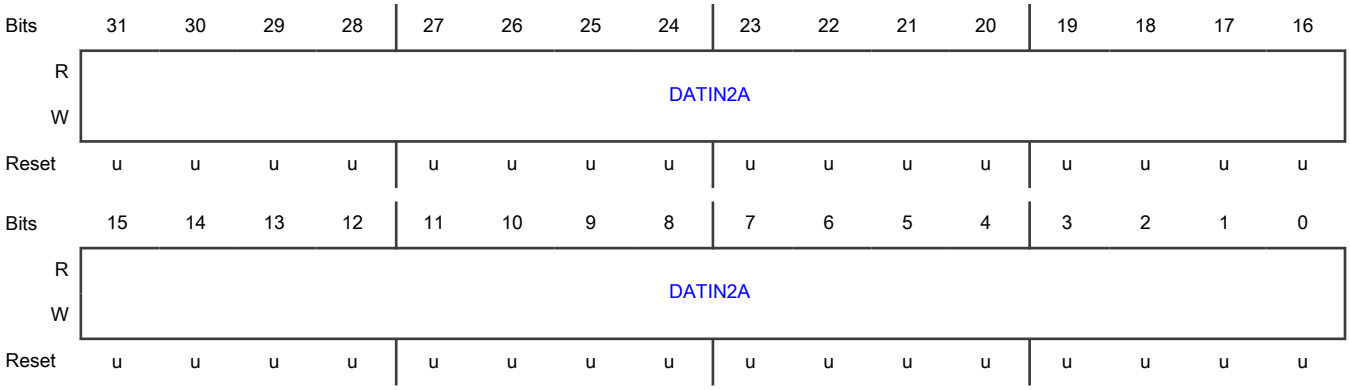
15.5.1.10 Input Data register 2 - Word-3 (SGI_DATIN2A)

Offset

Register	Offset
SGI_DATIN2A	220h

Function
Input Data register 2 - Word-3

Diagram



Fields

Field	Function
31-0 DATIN2A	Input Data register **NOTE** This field is volatile.

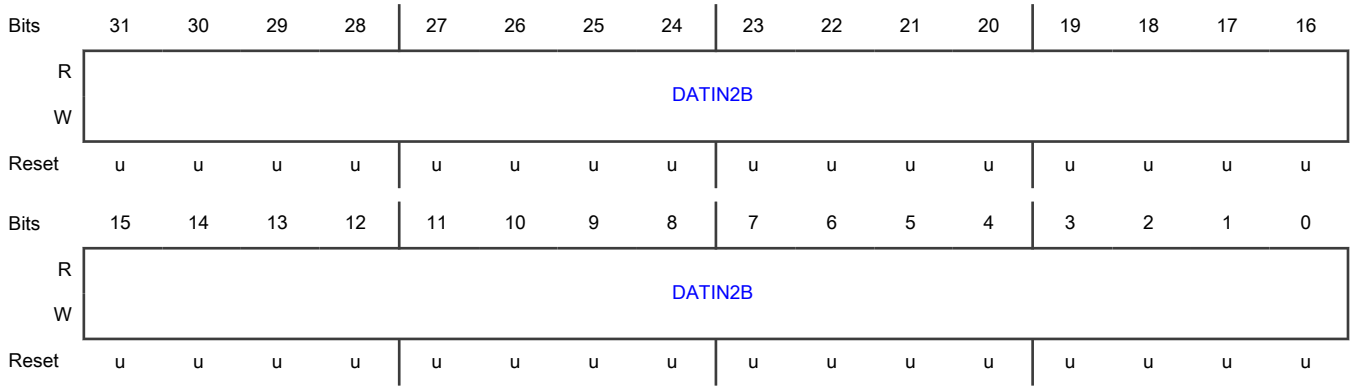
15.5.1.11 Input Data register 2 - Word-2 (SGI_DATIN2B)

Offset

Register	Offset
SGI_DATIN2B	224h

Function
Input Data register 2 - Word-2

Diagram



Fields

Field	Function
31-0 DATIN2B	Input Data register **NOTE** This field is volatile.

15.5.1.12 Input Data register 2 - Word-1 (SGI_DATIN2C)

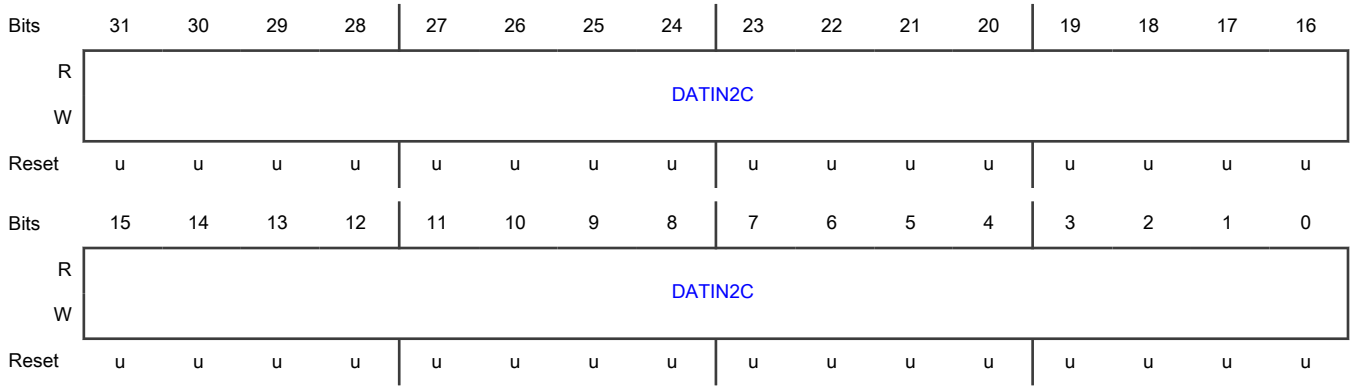
Offset

Register	Offset
SGI_DATIN2C	228h

Function

Input Data register 2 - Word-1

Diagram



Fields

Field	Function
31-0 DATIN2C	Input Data register **NOTE** This field is volatile.

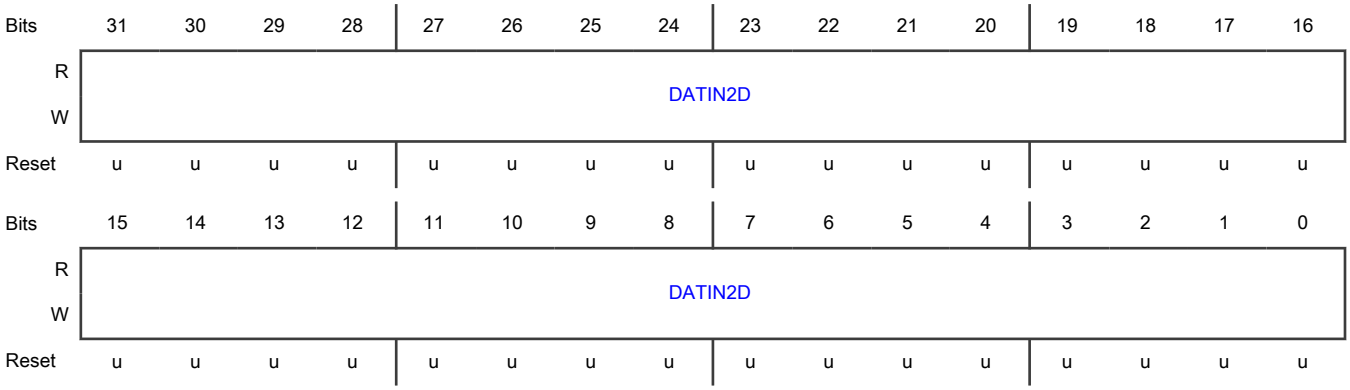
15.5.1.13 Input Data register 2 - Word-0 (SGI_DATIN2D)

Offset

Register	Offset
SGI_DATIN2D	22Ch

Function
Input Data register 2 - Word-0

Diagram



Fields

Field	Function
31-0 DATIN2D	Input Data register **NOTE** This field is volatile.

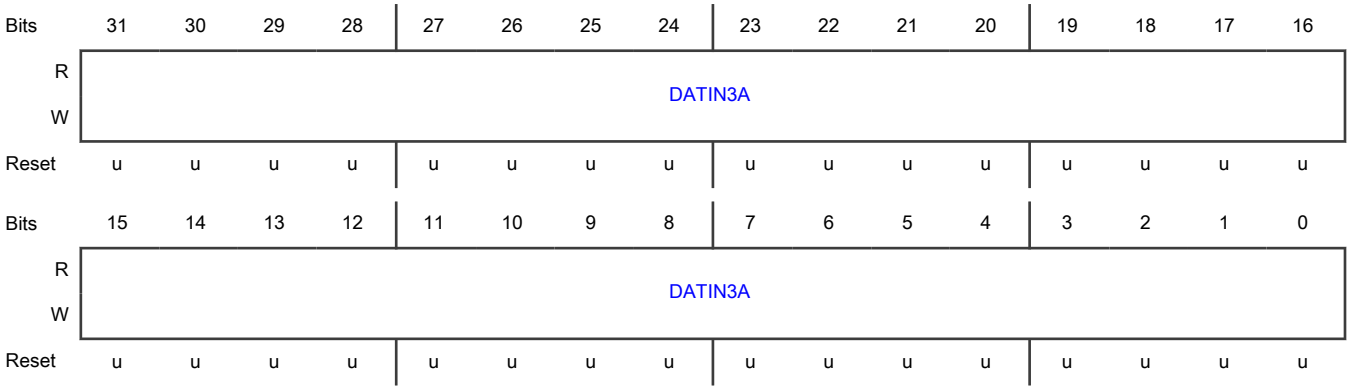
15.5.1.14 Input Data register 3 - Word-3 (SGI_DATIN3A)

Offset

Register	Offset
SGI_DATIN3A	230h

Function
Input Data register 3 - Word-3

Diagram



Fields

Field	Function
31-0 DATIN3A	Input Data register **NOTE** This field is volatile.

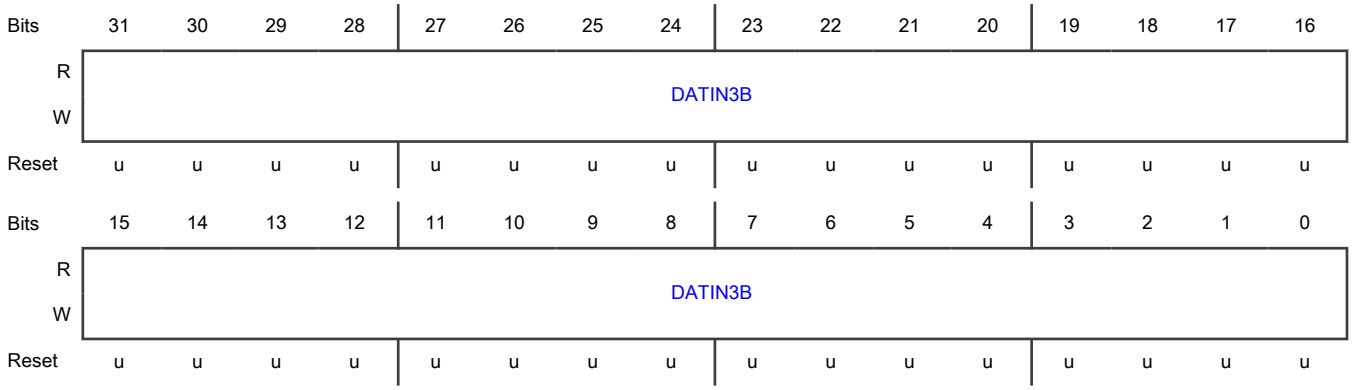
15.5.1.15 Input Data register 3 - Word-2 (SGI_DATIN3B)

Offset

Register	Offset
SGI_DATIN3B	234h

Function
Input Data register 3 - Word-2

Diagram



Fields

Field	Function
31-0 DATIN3B	Input Data register **NOTE** This field is volatile.

15.5.1.16 Input Data register 3 - Word-1 (SGI_DATIN3C)

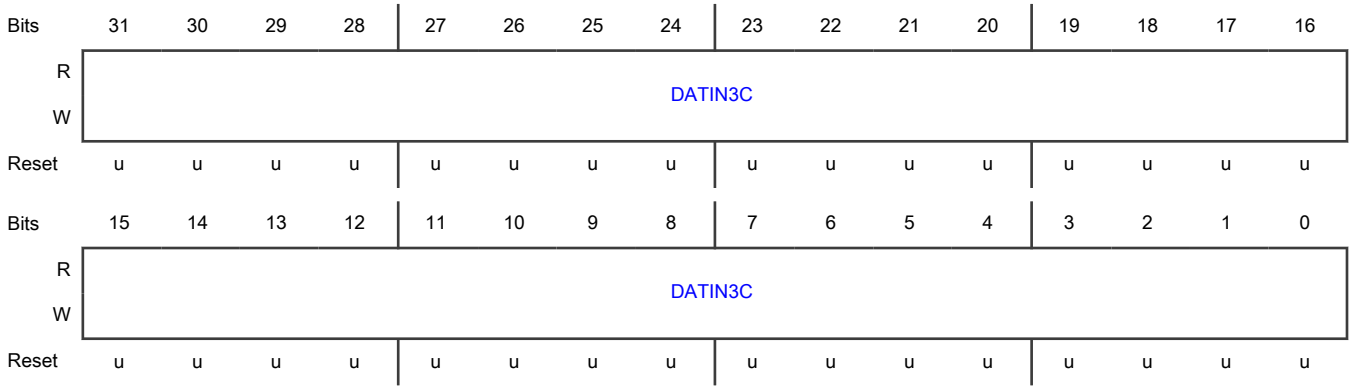
Offset

Register	Offset
SGI_DATIN3C	238h

Function

Input Data register 3 - Word-1

Diagram



Fields

Field	Function
31-0 DATIN3C	Input Data register **NOTE** This field is volatile.

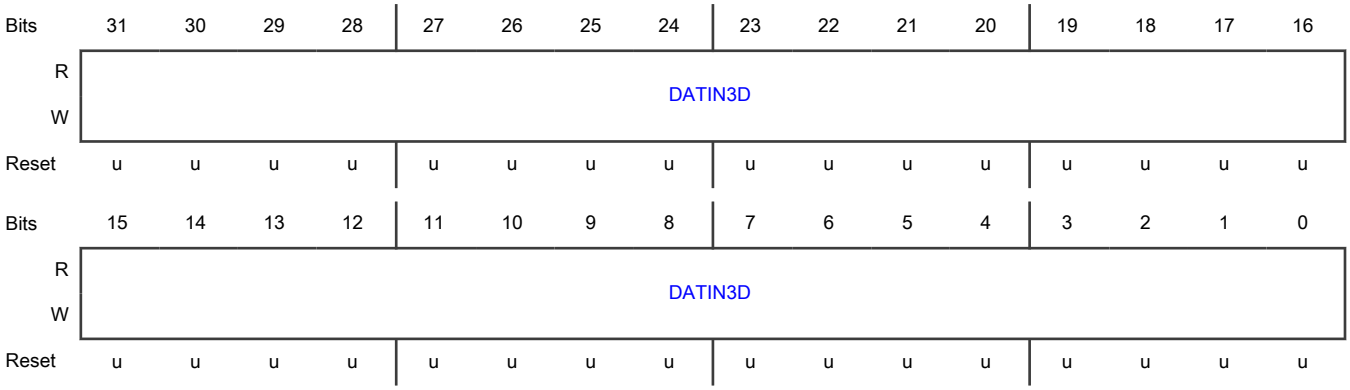
15.5.1.17 Input Data register 3 - Word-0 (SGI_DATIN3D)

Offset

Register	Offset
SGI_DATIN3D	23Ch

Function
Input Data register 3 - Word-0

Diagram



Fields

Field	Function
31-0 DATIN3D	Input Data register **NOTE** This field is volatile.

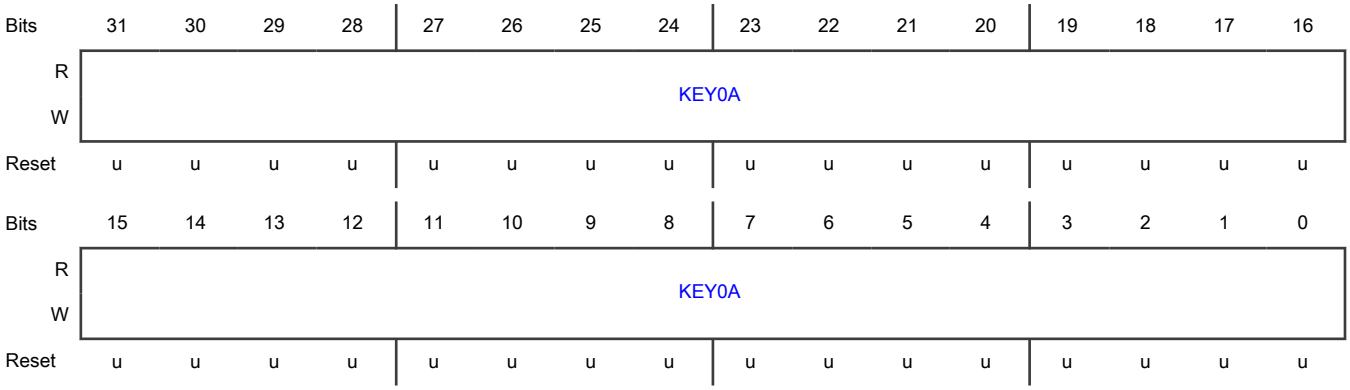
15.5.1.18 Input Key register 0 - Word-3 (SGI_KEY0A)

Offset

Register	Offset
SGI_KEY0A	240h

Function
Input Key register 0 - Word-3

Diagram



Fields

Field	Function
31-0 KEY0A	Input Key register **NOTE** This field is volatile.

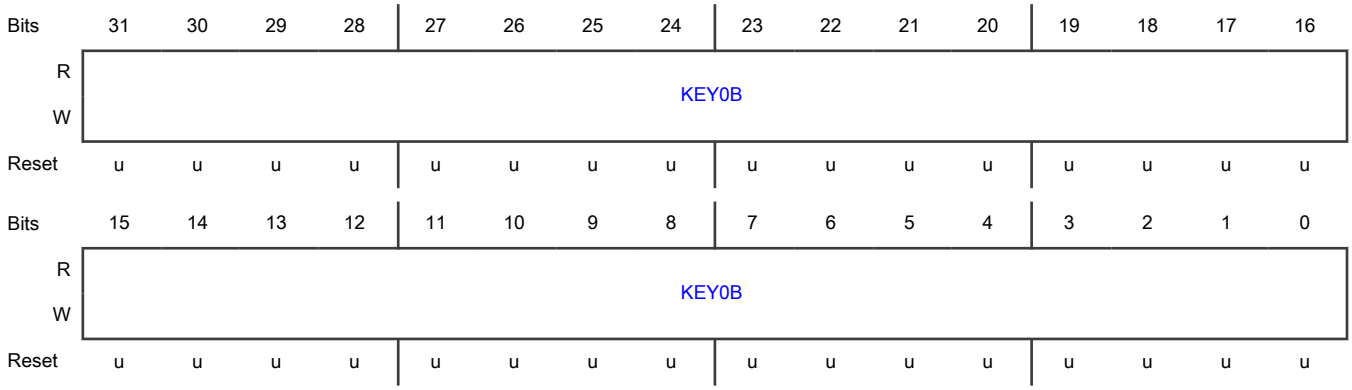
15.5.1.19 Input Key register 0 - Word-2 (SGI_KEY0B)

Offset

Register	Offset
SGI_KEY0B	244h

Function
Input Key register 0 - Word-2

Diagram



Fields

Field	Function
31-0 KEY0B	Input Key register **NOTE** This field is volatile.

15.5.1.20 Input Key register 0 - Word-1 (SGI_KEY0C)

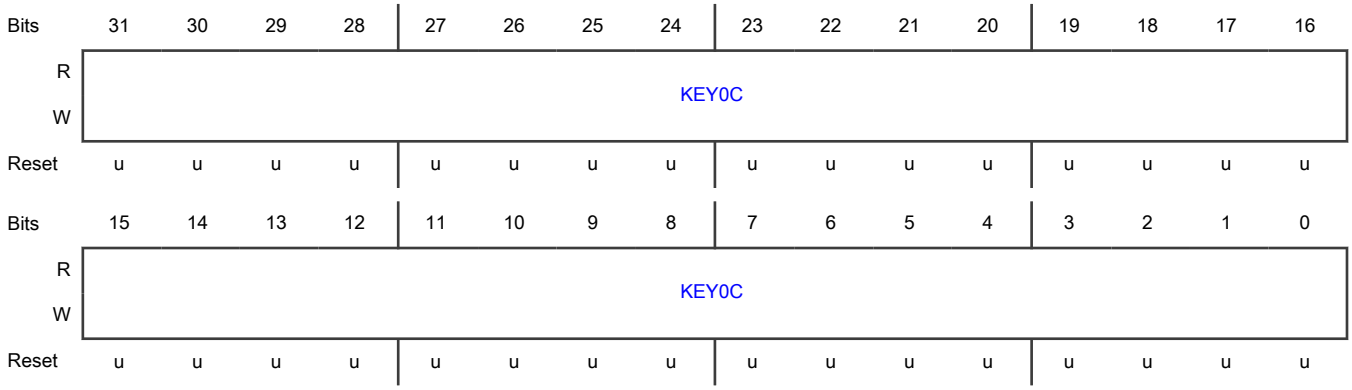
Offset

Register	Offset
SGI_KEY0C	248h

Function

Input Key register 0 - Word-1

Diagram



Fields

Field	Function
31-0 KEY0C	Input Key register **NOTE** This field is volatile.

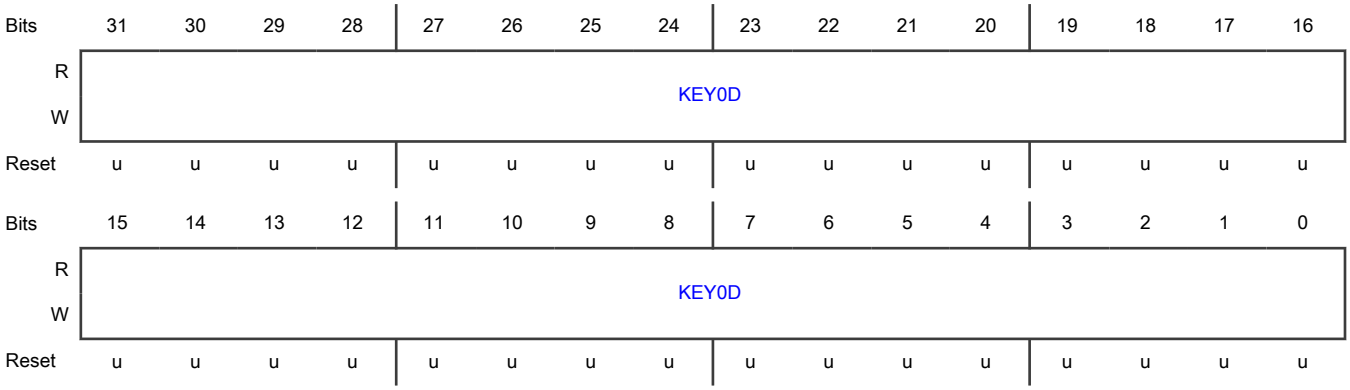
15.5.1.21 Input Key register 0 - Word-0 (SGI_KEY0D)

Offset

Register	Offset
SGI_KEY0D	24Ch

Function
Input Key register 0 - Word-0

Diagram



Fields

Field	Function
31-0 KEY0D	Input Key register **NOTE** This field is volatile.

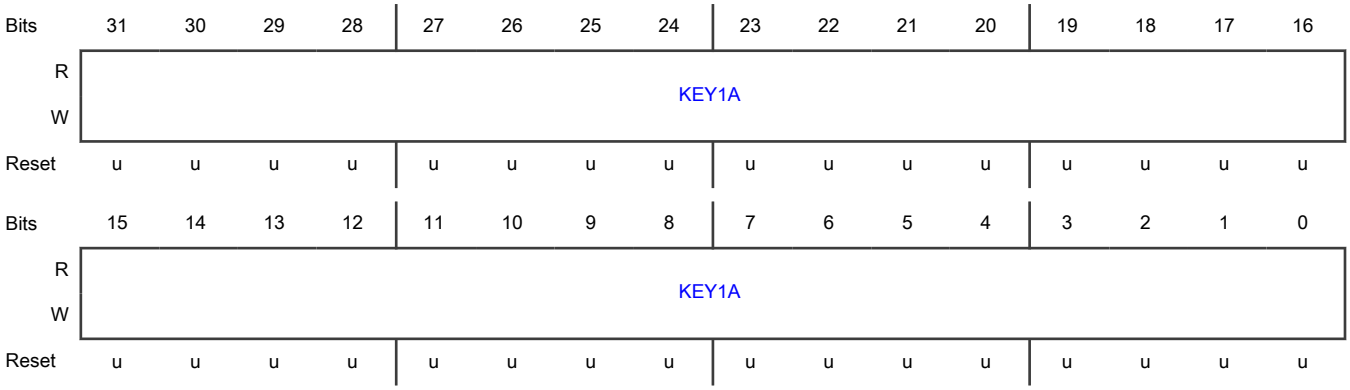
15.5.1.22 Input Key register 1 - Word-3 (SGI_KEY1A)

Offset

Register	Offset
SGI_KEY1A	250h

Function
Input Key register 1 - Word-3

Diagram



Fields

Field	Function
31-0 KEY1A	Input Key register **NOTE** This field is volatile.

15.5.1.23 Input Key register 1 - Word-2 (SGI_KEY1B)

Offset

Register	Offset
SGI_KEY1B	254h

Function
Input Key register 1 - Word-2

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	KEY1B															
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	KEY1B															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Fields

Field	Function
31-0 KEY1B	Input Key register **NOTE** This field is volatile.

15.5.1.24 Input Key register 1 - Word-1 (SGI_KEY1C)

Offset

Register	Offset
SGI_KEY1C	258h

Function

Input Key register 1 - Word-1

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	KEY1C															
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	KEY1C															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Fields

Field	Function
31-0 KEY1C	Input Key register **NOTE** This field is volatile.

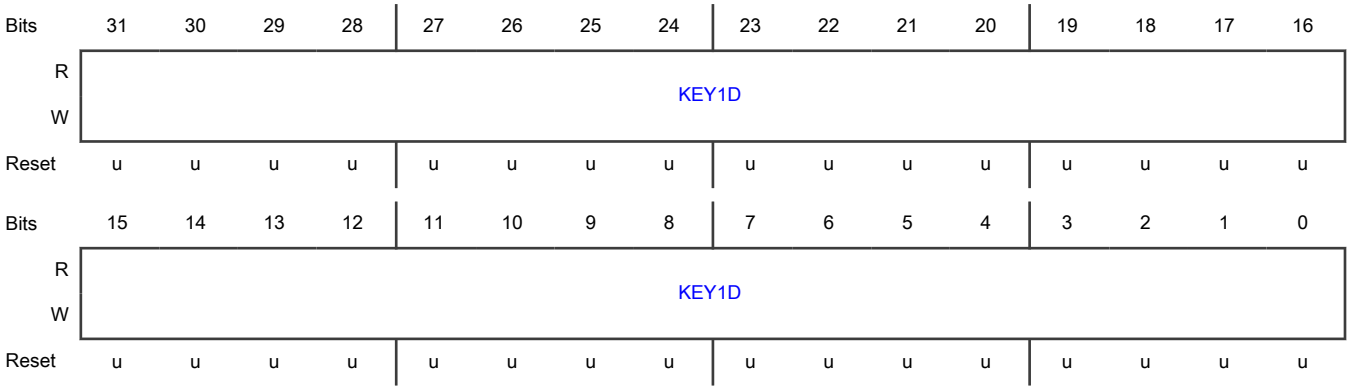
15.5.1.25 Input Key register 1 - Word-0 (SGI_KEY1D)

Offset

Register	Offset
SGI_KEY1D	25Ch

Function
Input Key register 1 - Word-0

Diagram



Fields

Field	Function
31-0 KEY1D	Input Key register **NOTE** This field is volatile.

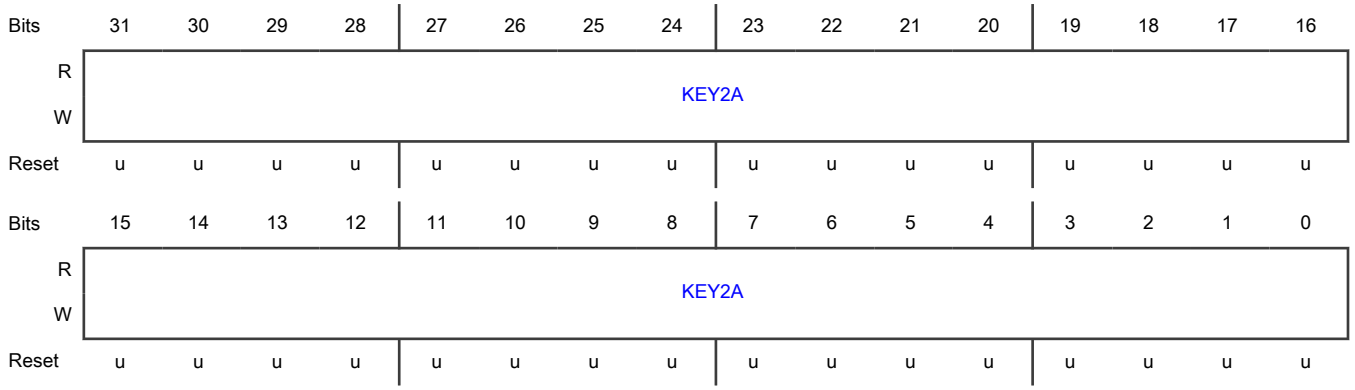
15.5.1.26 Input Key register 2 - Word-3 (SGI_KEY2A)

Offset

Register	Offset
SGI_KEY2A	260h

Function
Input Key register 2 - Word-3

Diagram



Fields

Field	Function
31-0 KEY2A	Input Key register **NOTE** This field is volatile.

15.5.1.27 Input Key register 2 - Word-2 (SGI_KEY2B)

Offset

Register	Offset
SGI_KEY2B	264h

Function
Input Key register 2 - Word-2

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	KEY2B															
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	KEY2B															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Fields

Field	Function
31-0 KEY2B	Input Key register **NOTE** This field is volatile.

15.5.1.28 Input Key register 2 - Word-1 (SGI_KEY2C)

Offset

Register	Offset
SGI_KEY2C	268h

Function

Input Key register 2 - Word-1

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	KEY2C															
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	KEY2C															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Fields

Field	Function
31-0 KEY2C	Input Key register **NOTE** This field is volatile.

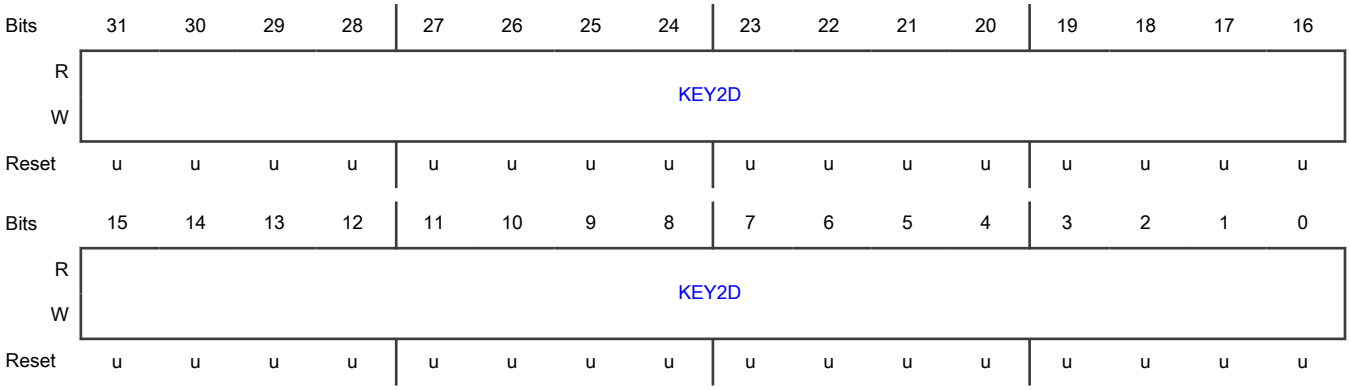
15.5.1.29 Input Key register 2 - Word-0 (SGI_KEY2D)

Offset

Register	Offset
SGI_KEY2D	26Ch

Function
Input Key register 2 - Word-0

Diagram



Fields

Field	Function
31-0 KEY2D	Input Key register **NOTE** This field is volatile.

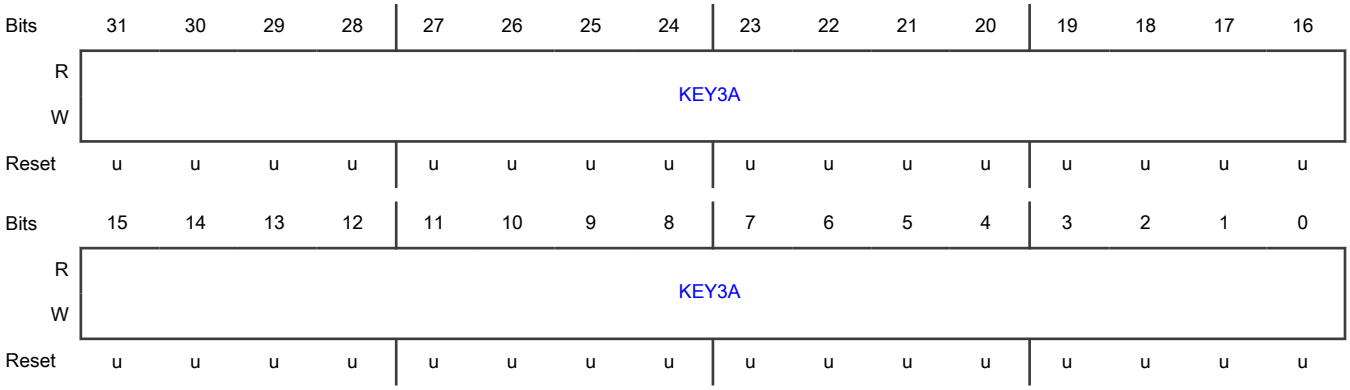
15.5.1.30 Input Key register 3 - Word-3 (SGI_KEY3A)

Offset

Register	Offset
SGI_KEY3A	270h

Function
Input Key register 3 - Word-3

Diagram



Fields

Field	Function
31-0 KEY3A	Input Key register **NOTE** This field is volatile.

15.5.1.31 Input Key register 3 - Word-2 (SGI_KEY3B)

Offset

Register	Offset
SGI_KEY3B	274h

Function
Input Key register 3 - Word-2

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	KEY3B															
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	KEY3B															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Fields

Field	Function
31-0 KEY3B	Input Key register **NOTE** This field is volatile.

15.5.1.32 Input Key register 3 - Word-1 (SGI_KEY3C)

Offset

Register	Offset
SGI_KEY3C	278h

Function

Input Key register 3 - Word-1

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	KEY3C															
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	KEY3C															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Fields

Field	Function
31-0 KEY3C	Input Key register **NOTE** This field is volatile.

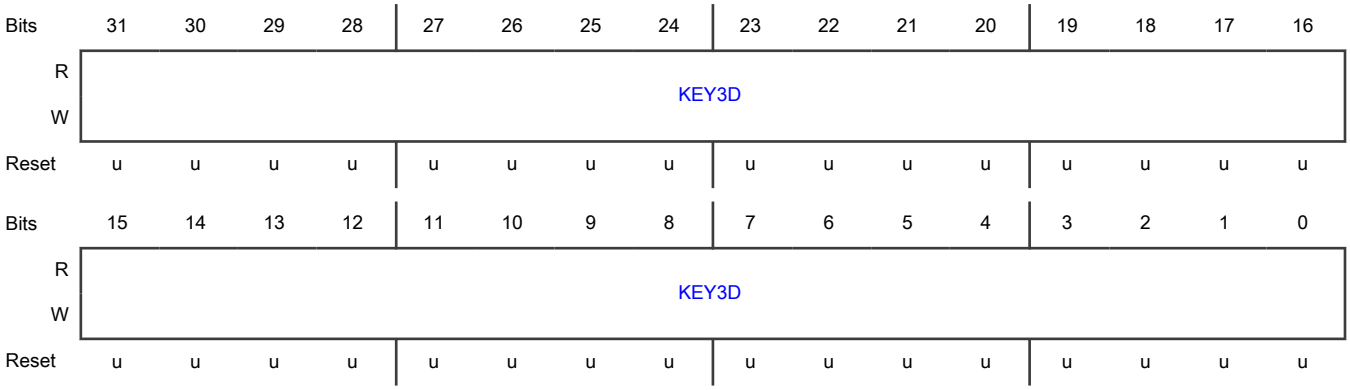
15.5.1.33 Input Key register 3 - Word-0 (SGI_KEY3D)

Offset

Register	Offset
SGI_KEY3D	27Ch

Function
Input Key register 3 - Word-0

Diagram



Fields

Field	Function
31-0 KEY3D	Input Key register **NOTE** This field is volatile.

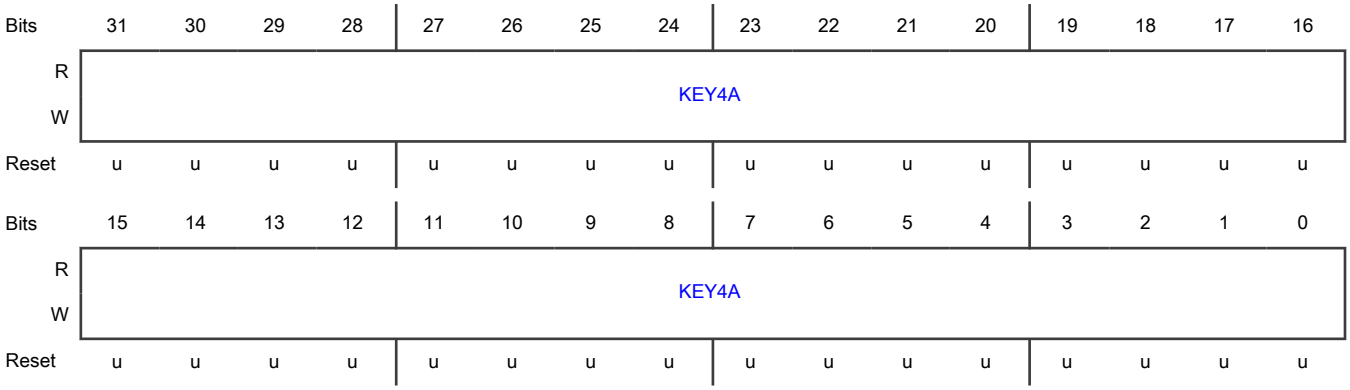
15.5.1.34 Input Key register 4 - Word-3 (SGI_KEY4A)

Offset

Register	Offset
SGI_KEY4A	280h

Function
Input Key register 4 - Word-3

Diagram



Fields

Field	Function
31-0 KEY4A	Input Key register **NOTE** This field is volatile.

15.5.1.35 Input Key register 4 - Word-2 (SGI_KEY4B)

Offset

Register	Offset
SGI_KEY4B	284h

Function
Input Key register 4 - Word-2

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	KEY4B															
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	KEY4B															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Fields

Field	Function
31-0 KEY4B	Input Key register **NOTE** This field is volatile.

15.5.1.36 Input Key register 4 - Word-1 (SGI_KEY4C)

Offset

Register	Offset
SGI_KEY4C	288h

Function

Input Key register 4 - Word-1

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	KEY4C															
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	KEY4C															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Fields

Field	Function
31-0 KEY4C	Input Key register **NOTE** This field is volatile.

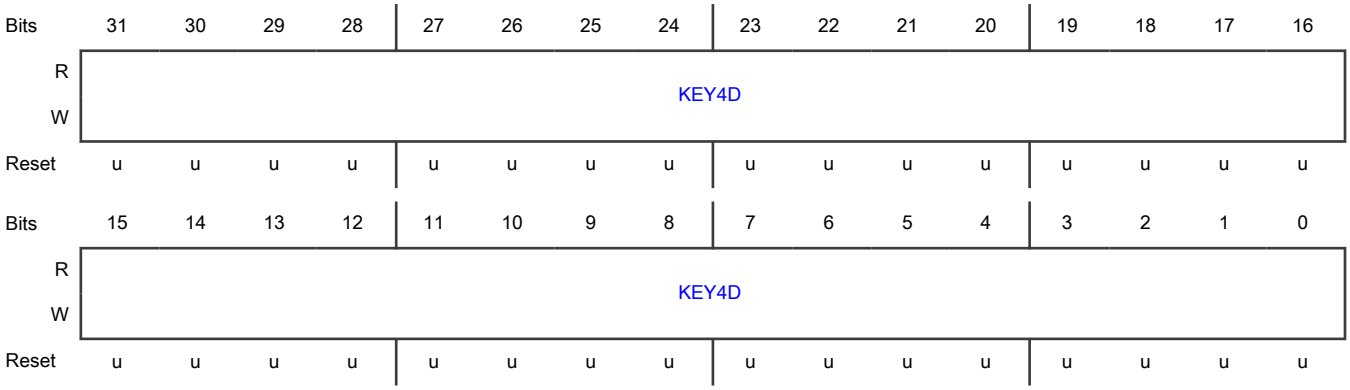
15.5.1.37 Input Key register 4 - Word-0 (SGI_KEY4D)

Offset

Register	Offset
SGI_KEY4D	28Ch

Function
Input Key register 4 - Word-0

Diagram



Fields

Field	Function
31-0 KEY4D	Input Key register **NOTE** This field is volatile.

15.5.1.38 Input Key register 5 - Word-3 (SGI_KEY5A)

Offset

Register	Offset
SGI_KEY5A	290h

Function
Input Key register 5 - Word-3

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	KEY5A															
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	KEY5A															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Fields

Field	Function
31-0 KEY5A	Input Key register **NOTE** This field is volatile.

15.5.1.39 Input Key register 5 - Word-2 (SGI_KEY5B)

Offset

Register	Offset
SGI_KEY5B	294h

Function
Input Key register 5 - Word-2

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	KEY5B															
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	KEY5B															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Fields

Field	Function
31-0 KEY5B	Input Key register **NOTE** This field is volatile.

15.5.1.40 Input Key register 5 - Word-1 (SGI_KEY5C)

Offset

Register	Offset
SGI_KEY5C	298h

Function

Input Key register 5 - Word-1

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	KEY5C															
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	KEY5C															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Fields

Field	Function
31-0 KEY5C	Input Key register **NOTE** This field is volatile.

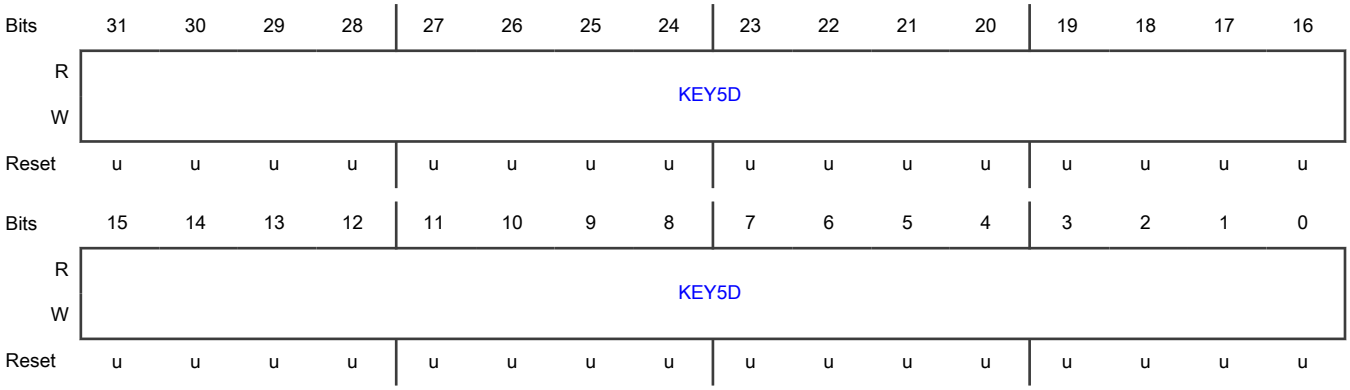
15.5.1.41 Input Key register 5 - Word-0 (SGI_KEY5D)

Offset

Register	Offset
SGI_KEY5D	29Ch

Function
Input Key register 5 - Word-0

Diagram



Fields

Field	Function
31-0 KEY5D	Input Key register **NOTE** This field is volatile.

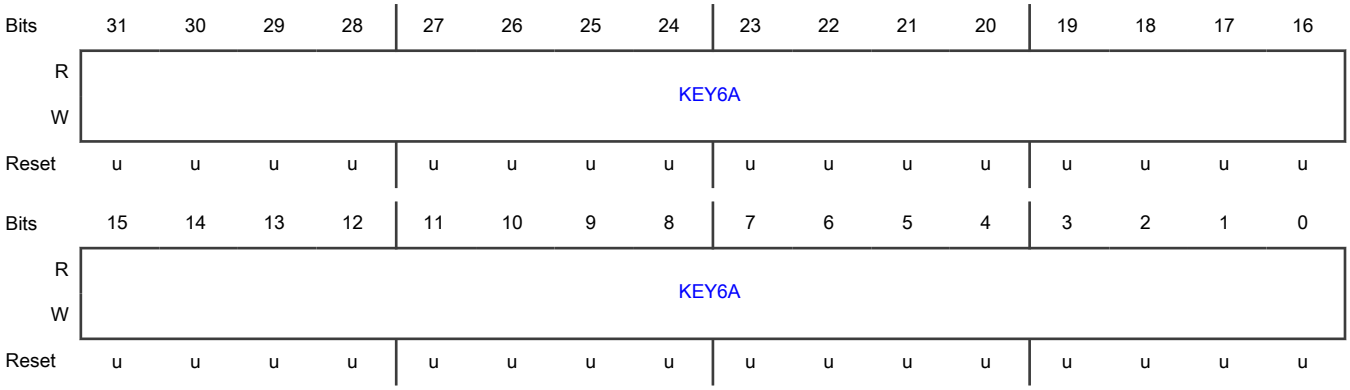
15.5.1.42 Input Key register 6 - Word-3 (SGI_KEY6A)

Offset

Register	Offset
SGI_KEY6A	2A0h

Function
Input Key register 6 - Word-3

Diagram



Fields

Field	Function
31-0 KEY6A	Input Key register **NOTE** This field is volatile.

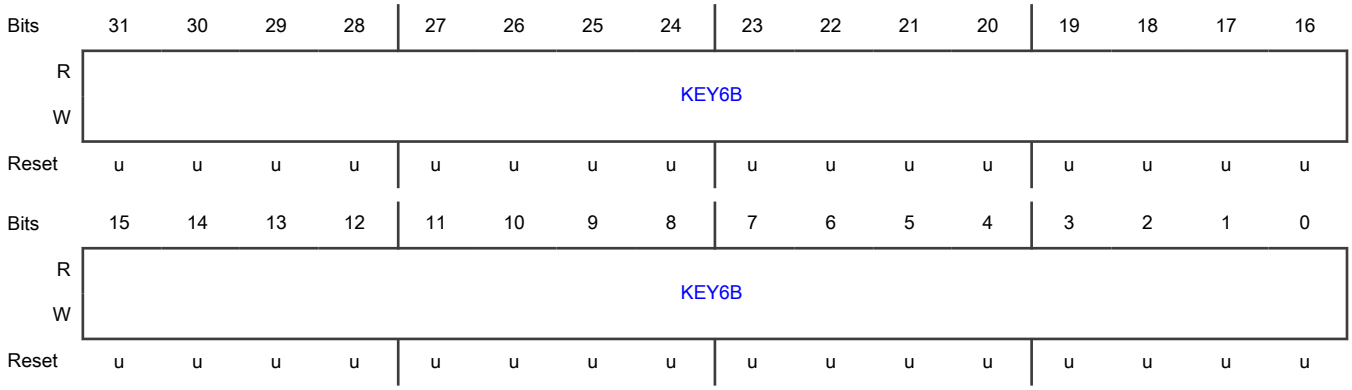
15.5.1.43 Input Key register 6 - Word-2 (SGI_KEY6B)

Offset

Register	Offset
SGI_KEY6B	2A4h

Function
Input Key register 6 - Word-2

Diagram



Fields

Field	Function
31-0 KEY6B	Input Key register **NOTE** This field is volatile.

15.5.1.44 Input Key register 6 - Word-1 (SGI_KEY6C)

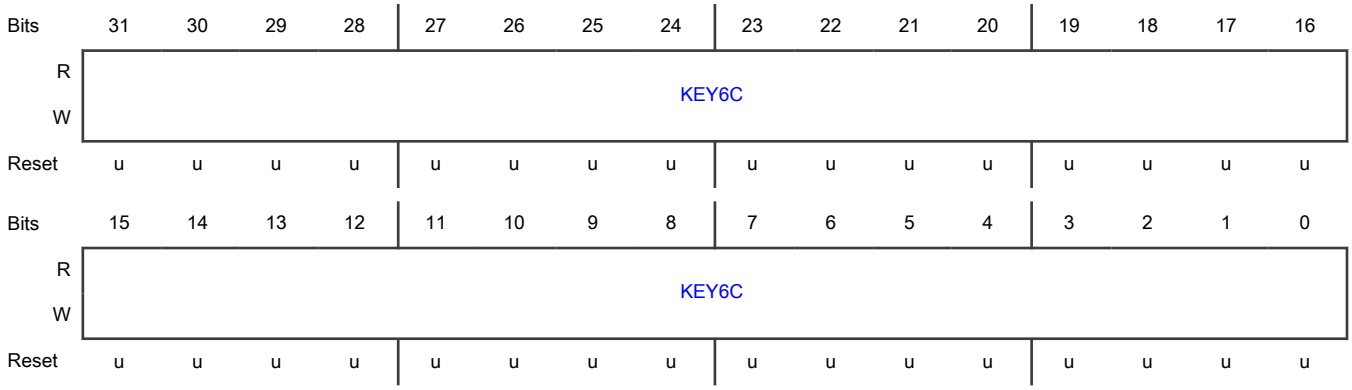
Offset

Register	Offset
SGI_KEY6C	2A8h

Function

Input Key register 6 - Word-1

Diagram



Fields

Field	Function
31-0 KEY6C	Input Key register **NOTE** This field is volatile.

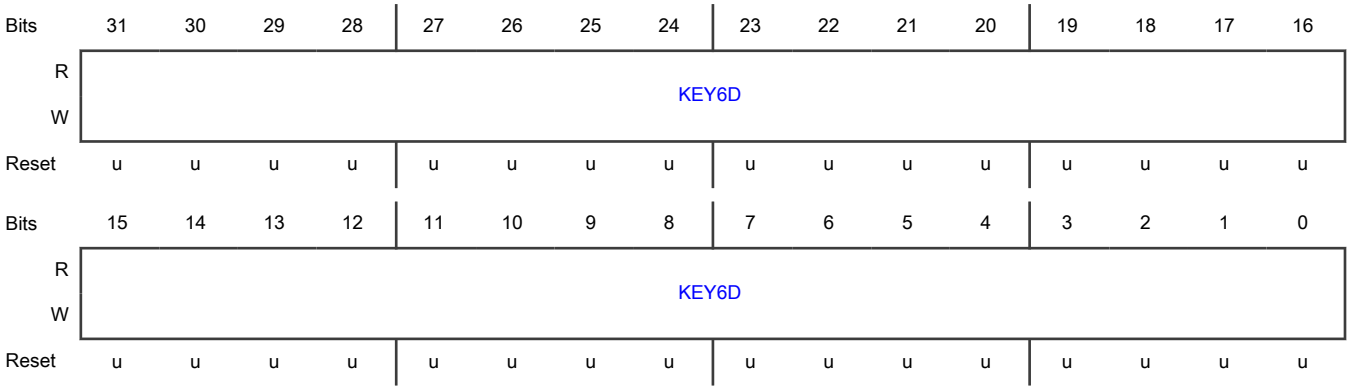
15.5.1.45 Input Key register 6 - Word-0 (SGI_KEY6D)

Offset

Register	Offset
SGI_KEY6D	2ACh

Function
Input Key register 6 - Word-0

Diagram



Fields

Field	Function
31-0 KEY6D	Input Key register **NOTE** This field is volatile.

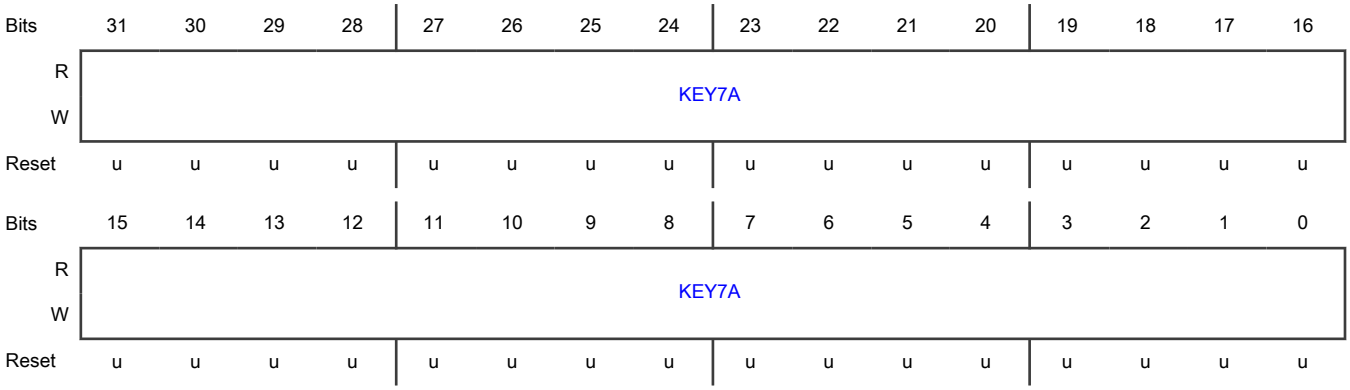
15.5.1.46 Input Key register 7 - Word-3 (SGI_KEY7A)

Offset

Register	Offset
SGI_KEY7A	2B0h

Function
Input Key register 7 - Word-3

Diagram



Fields

Field	Function
31-0 KEY7A	Input Key register **NOTE** This field is volatile.

15.5.1.47 Input Key register 7 - Word-2 (SGI_KEY7B)

Offset

Register	Offset
SGI_KEY7B	2B4h

Function
Input Key register 7 - Word-2

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	KEY7B															
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	KEY7B															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Fields

Field	Function
31-0 KEY7B	Input Key register **NOTE** This field is volatile.

15.5.1.48 Input Key register 7 - Word-1 (SGI_KEY7C)

Offset

Register	Offset
SGI_KEY7C	2B8h

Function

Input Key register 7 - Word-1

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	KEY7C															
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	KEY7C															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Fields

Field	Function
31-0 KEY7C	Input Key register **NOTE** This field is volatile.

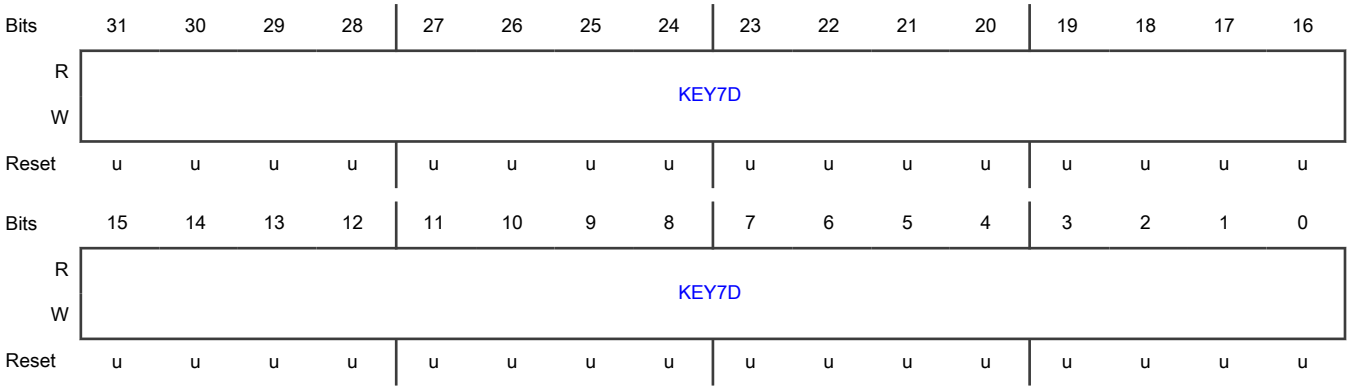
15.5.1.49 Input Key register 7 - Word-0 (SGI_KEY7D)

Offset

Register	Offset
SGI_KEY7D	2BCh

Function
Input Key register 7 - Word-0

Diagram



Fields

Field	Function
31-0 KEY7D	Input Key register **NOTE** This field is volatile.

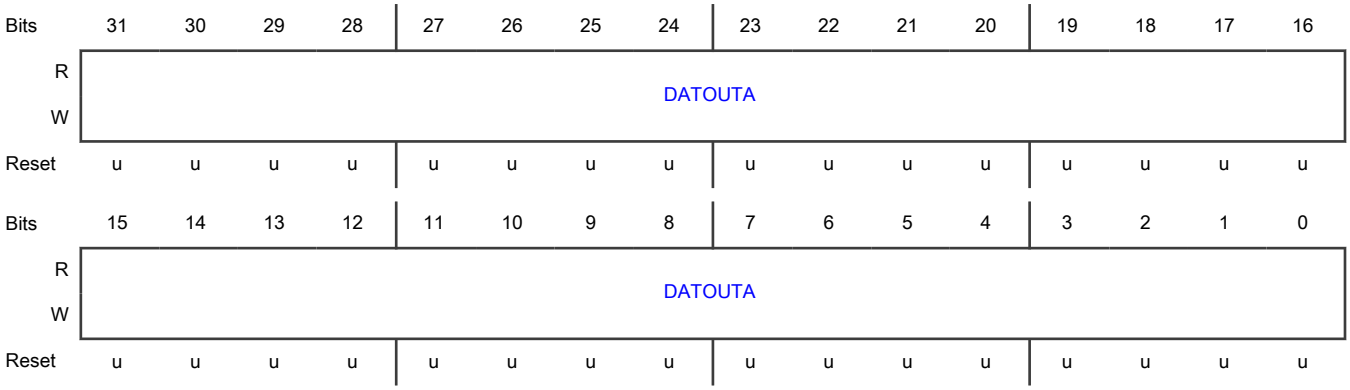
15.5.1.50 Output Data register - Word-3 (SGI_DATOUTA)

Offset

Register	Offset
SGI_DATOUTA	2C0h

Function
Output Data register - Word-3

Diagram



Fields

Field	Function
31-0 DATOUTA	Output Data register **NOTE** This field is volatile.

15.5.1.51 Output Data register - Word-2 (SGI_DATOUTB)

Offset

Register	Offset
SGI_DATOUTB	2C4h

Function
Output Data register - Word-2

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	DATOUTB															
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	DATOUTB															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Fields

Field	Function
31-0 DATOUTB	Output Data register **NOTE** This field is volatile.

15.5.1.52 Output Data register - Word-1 (SGI_DATOUTC)

Offset

Register	Offset
SGI_DATOUTC	2C8h

Function

Output Data register - Word-1

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	DATOUTC															
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	DATOUTC															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Fields

Field	Function
31-0 DATOUTC	Output Data register **NOTE** This field is volatile.

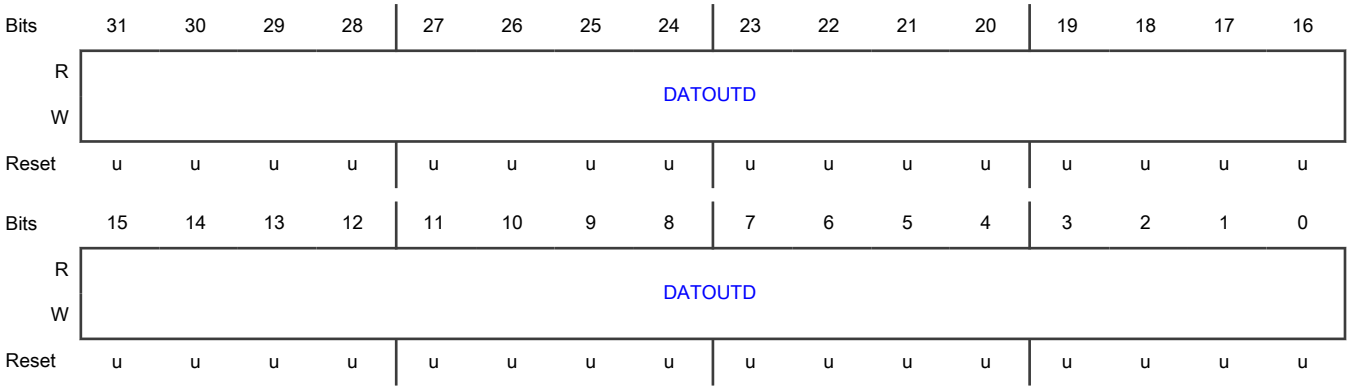
15.5.1.53 Output Data register - Word-0 (SGI_DATOUTD)

Offset

Register	Offset
SGI_DATOUTD	2CCh

Function
Output Data register - Word-0

Diagram



Fields

Field	Function
31-0 DATOUTD	Output Data register **NOTE** This field is volatile.

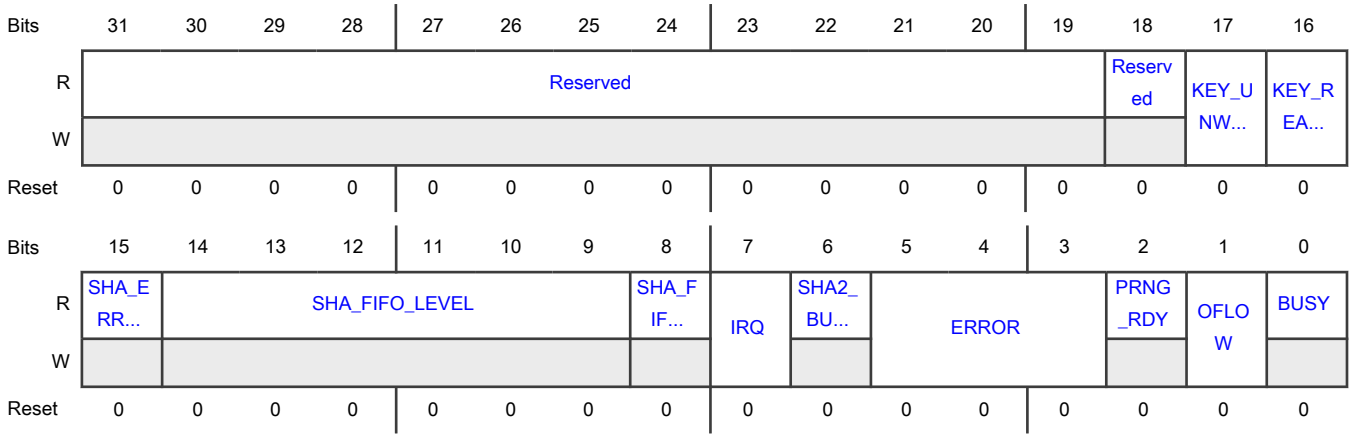
15.5.1.54 Status register (SGI_STATUS)

Offset

Register	Offset
SGI_STATUS	C00h

Function
Status register

Diagram



Fields

Field	Function
31-19 —	Reserved
18 —	Reserved
17 KEY_UNWRAP_ERR	KEY UNWRAP ERROR , sticky, cleared only with reset or flush **NOTE** This field is volatile. 0b - No error 1b - Error
16 KEY_READ_ERR	KEY SFR READ ERROR, sticky, cleared only with reset or flush **NOTE** This field is volatile.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - No error 1b - Error
15 SHA_ERROR	SHA error (see SHA errors for details) **NOTE** This field is volatile. 0b - No error 1b - Error
14-9 SHA_FIFO_LEVEL	SHA FIFO level (0x00 indicates empty, and 0x1F indicates full) **NOTE** This field is volatile.
8 SHA_FIFO_FULL	Indicates if SHA FIFO is full **NOTE** This field is volatile. 0b - Not full 1b - Full
7 IRQ	Interrupt status bit **NOTE** This field is volatile. **NOTE** This field behaves differently for register reads and writes. When reading 0b - No IRQ 1b - IRQ When writing 0b - Clear the flag 1b - No effect
6 SHA2_BUSY	Indicates if SHA2 kernal is busy **NOTE** This field is volatile. 0b - Not busy 1b - Busy

Table continues on the next page...

Table continued from the previous page...

Field	Function
5-3 ERROR	Error detected **NOTE** This field is volatile. 000b - Error (all values other than 0x05 indicate ERROR) 101b - No error
2 PRNG_RDY	Indicates if PRNG is ready after boot-up phase **NOTE** This field is volatile. 0b - Not ready 1b - Ready
1 OFLOW	Overflow in INCR operation flag **NOTE** This field is volatile. **NOTE** This field behaves differently for register reads and writes. When reading 0b - No overflow 1b - Overflow When writing 0b - Clear the flag 1b - No effect
0 BUSY	Combined busy flag that remains high until end of calculation **NOTE** This field is volatile. 0b - Not busy 1b - Busy

15.5.1.55 Calculation counter (SGI_COUNT)

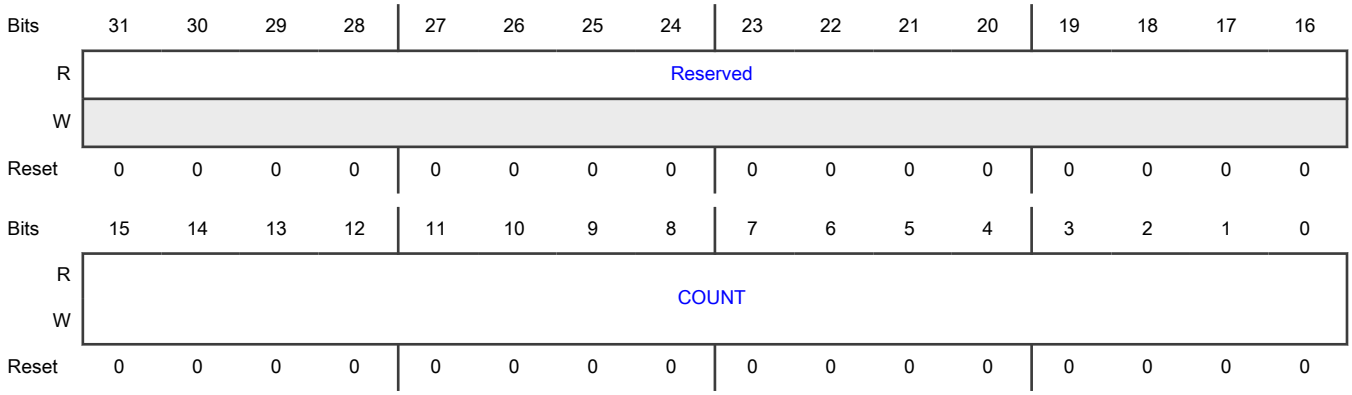
Offset

Register	Offset
SGI_COUNT	C04h

Function

Calculation counter

Diagram



Fields

Field	Function
31-16 —	Reserved
15-0 COUNT	Calculation counter, incremented with each calculation start. **NOTE** This field is volatile.

15.5.1.56 Key checksum register (SGI_KEYCHK)

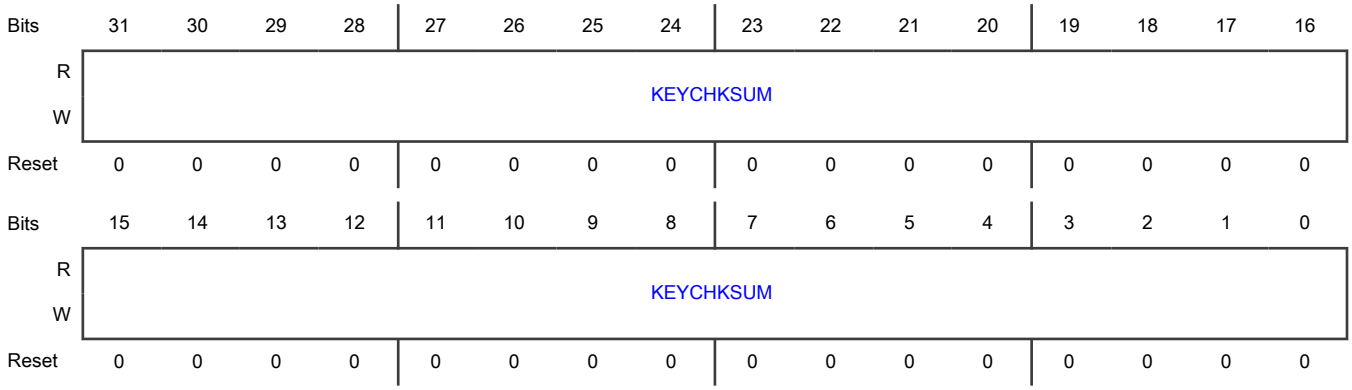
Offset

Register	Offset
SGI_KEYCHK	C08h

Function

Key checksum register

Diagram



Fields

Field	Function
31-0 KEYCHKSUM	Key checksum (32-bit). **NOTE** This field is volatile.

15.5.1.57 SGI Control register (SGI_CTRL)

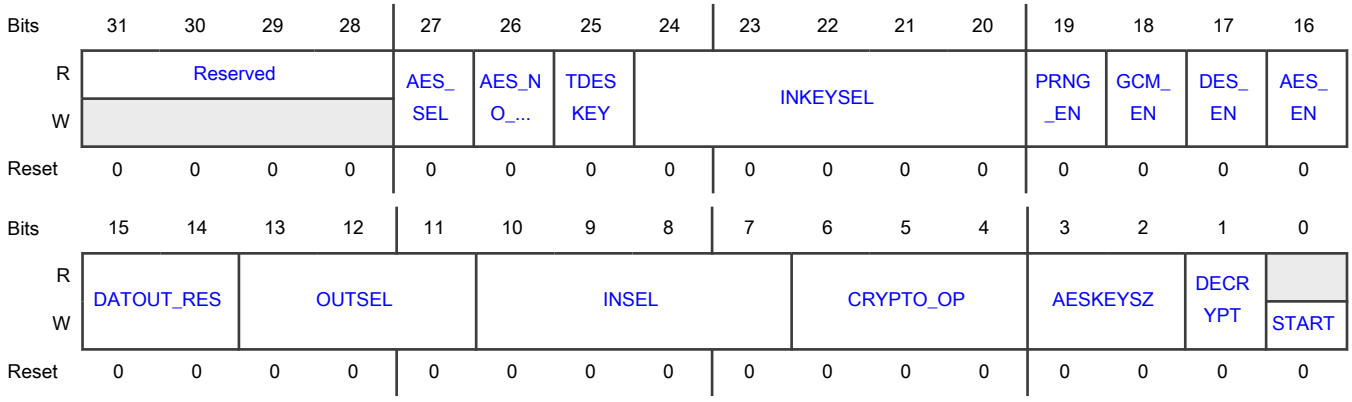
Offset

Register	Offset
SGI_CTRL	D00h

Function

SGI Control register

Diagram



Fields

Field	Function
31-28 —	Reserved
27 AES_SEL	AES Dual Selection **NOTE** This field is volatile. 0b - First AES selected 1b - Second AES selected (when enabled)
26 AES_NO_KL	AES No decryption key schedule **NOTE** This field is volatile. 0b - new AES key will be loaded 1b - No AES key will be loaded, and previously loaded key will be used.
25 TDESKEY	Triple-DES Key Configuration **NOTE** This field is volatile. 0b - 2-key TDES 1b - 3-key TDES
24-20 INKEYSEL	Input key selection Sets the base address for the input Key. **NOTE** This field is volatile.
19 PRNG_EN	PRNG Enable (only if SGI has internal PRNG) **NOTE** This field is volatile. 0b - PRNG Disabled 1b - PRNG Enabled
18 GCM_EN	GFMUL Kernel Enable **NOTE** This field is volatile. 0b - GFMUL disabled 1b - GFMUL enabled

Table continues on the next page...

Table continued from the previous page...

Field	Function
17 DES_EN	DES Kernel Enable **NOTE** This field is volatile. 0b - DES disabled 1b - DES enabled
16 AES_EN	AES Kernel Enable **NOTE** This field is volatile. 0b - AES disabled 1b - AES enabled
15-14 DATOUT_RES	Kernels data out options **NOTE** This field is volatile. 00b - END_UP 01b - START_UP 10b - TRIGGER_UP 11b - NO_UP
13-11 OUTSEL	**NOTE** This field is volatile. 000b - DATOUT = 'Kernel Res' 001b - DATOUT = 'Kernel Res' ^ DATIN[0] 010b - DATOUT = 'Kernel Res' ^ DATIN[1]* 011b - DATOUT = 'Kernel Res' ^ DATIN[2]* 100b - DATOUT = 'Kernel Res' ^DATIN[3]* 101b-111b - others - DATOUT = 'Kernel Res' * - only if DATIN[num] exists, else [0]
10-7 INSEL	**NOTE** This field is volatile. 0000b - DATIN[0] 0001b - DATIN[1]* 0010b - DATIN[2]* 0011b - DATIN[3]*

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0100b - DATIN[0] ^ DATOUT 0101b - DATIN[1] ^ DATOUT* 0110b - DATIN[2] ^ DATOUT* 0111b - DATIN[3] ^ DATOUT* 1000b - DATOUT 1001b-1111b - others - DATIN[0] * - only if DATIN[num] exists, else [0]
6-4 CRYPTO_OP	Sets 'Crypto Operation' type **NOTE** This field is volatile. 000b - AES 001b - DES (If Included) 010b - TDES (If Included) 011b - GFMUL(If Included) 100b - SHA2 (If Included) 101b - CMAC (If Included) 110b-111b - others - RFU (Defaults to 1st available OP)
3-2 AESKEYSZ	Sets AES key size **NOTE** This field is volatile. 00b - AES-128 10b - AES-256 11b - RFU (defaults to AES-128)
1 DECRYPT	Sets Cipher direction(AES and DES) **NOTE** This field is volatile. 0b - Encryption 1b - Decryption
0 START	Start crypto operation **NOTE** This field is volatile. 0b - Clr has no effect 1b - Set to start operation

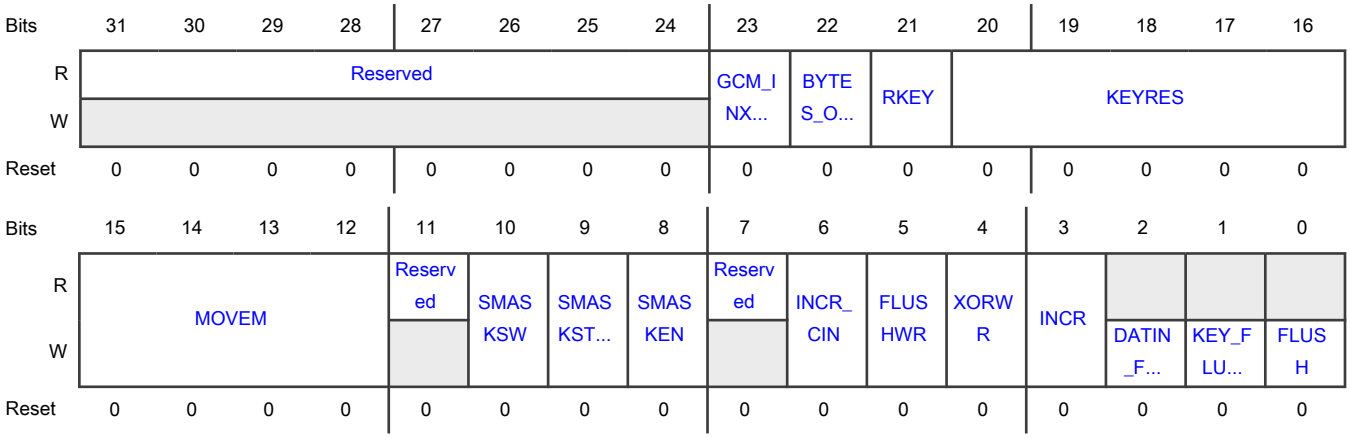
15.5.1.58 SGI Control register 2 (SGI_CTRL2)

Offset

Register	Offset
SGI_CTRL2	D04h

Function
SGI Control register 2

Diagram



Fields

Field	Function
31-24 —	Reserved
23 GCM_INXOR	GCM INXOR **NOTE** This field is volatile. 0b - GCM INXOR disabled 1b - GCM INXOR enabled
22 BYTES_ORDER	Byte order of regbank read/write data **NOTE** This field is volatile. 0b - Normal 1b - Swapped
21	Crypto result location

Table continues on the next page...

Table continued from the previous page...

Field	Function
RKEY	**NOTE** This field is volatile. 0b - DATOUT register bank 1b - KEY register bank
20-16 KEYRES	Selects key registers to be updated when rkey=1 Sets the base address of the Key register for crypto result storage, to be used with 'RKEY'. **NOTE** This field is volatile.
15-12 MOVEM	4-bit optional input for MOVEM feature **NOTE** This field is volatile.
11 —	Reserved
10 SMASKSW	SFRMASK MASK control **NOTE** This field is volatile. 0b - SFR MASK output directly controlled by HW mask generator 1b - SFR MASK output directly controlled by SW
9 SMASKSTEP	SFRSEED increment control **NOTE** This field is volatile. 0b - SFRSEED increments every regbank access 1b - SFRSEED increments every regbank access PLUS when SFRSEED in read
8 SMASKEN	SFRMASK Enable **NOTE** This field is volatile. 0b - SFRMASK feature Disabled 1b - SFRMASK feature Enabled
7 —	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
6 INCR_CIN	Increment Carry-In control **NOTE** This field is volatile. 0b - Carry-In for INCR is 1 1b - Carry-In for INCR is overflow from previous INCR operation
5 FLUSHWR	Flush Write control **NOTE** This field is volatile. 0b - Flush-Write disabled 1b - Flush-Write enabled
4 XORWR	Write-XOR control **NOTE** This field is volatile. 0b - XOR-On-Write disabled 1b - XOR-On-Write enabled
3 INCR	Increment(Triggered by SFR write) **NOTE** This field is volatile. 0b - INCR-On-Write disabled 1b - INCR-On-Write enabled
2 DATIN_FLUSH	Start DATIN register-bank Flush **NOTE** This field is volatile. 0b - Clr has no effect 1b - Set to start flush
1 KEY_FLUSH	Start KEY register-bank Flush **NOTE** This field is volatile. 0b - Clr has no effect 1b - Set to start flush
0	Start Full SGI Flush

Table continues on the next page...

Table continued from the previous page...

Field	Function
FLUSH	**NOTE** This field is volatile. 0b - Clr has no effect 1b - Set to start flush

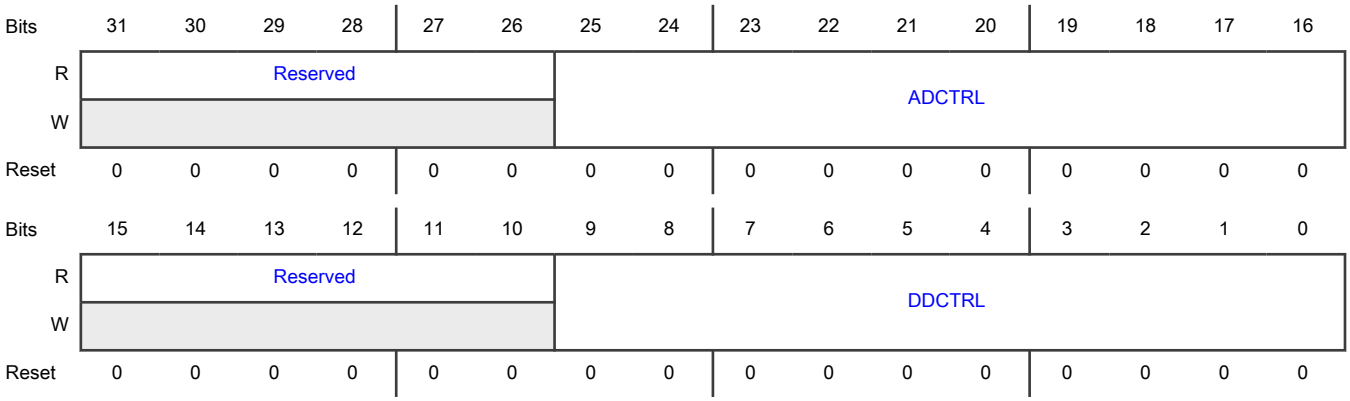
15.5.1.59 Configuration of dummy controls (SGI_DUMMY_CTRL)

Offset

Register	Offset
SGI_DUMMY_CTRL	D08h

Function
Configuration of dummy controls

Diagram



Fields

Field	Function
31-26 —	Reserved
25-16 ADCTRL	AES dummy control **NOTE** Please refer to the relevant kernel document for details on dummy ctrl. Not all kernels support dummy cycles.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	NOTE This field is volatile.
15-10 —	Reserved
9-0 DDCTRL	DES dummy control **NOTE** Please refer to the relevant kernel document for details on dummy ctrl. Not all kernels support dummy cycles. **NOTE** This field is volatile.

15.5.1.60 Sofware Assisted Masking register (SGI_SFR_SW_MASK)

Offset

Register	Offset
SGI_SFR_SW_MASK	D0Ch

Function

Sofware Assisted Masking register

Diagram

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

Fields

Field	Function
31-0 SFR_MASK_VAL	Seed/mask used for sw level masking **NOTE** This field is volatile.

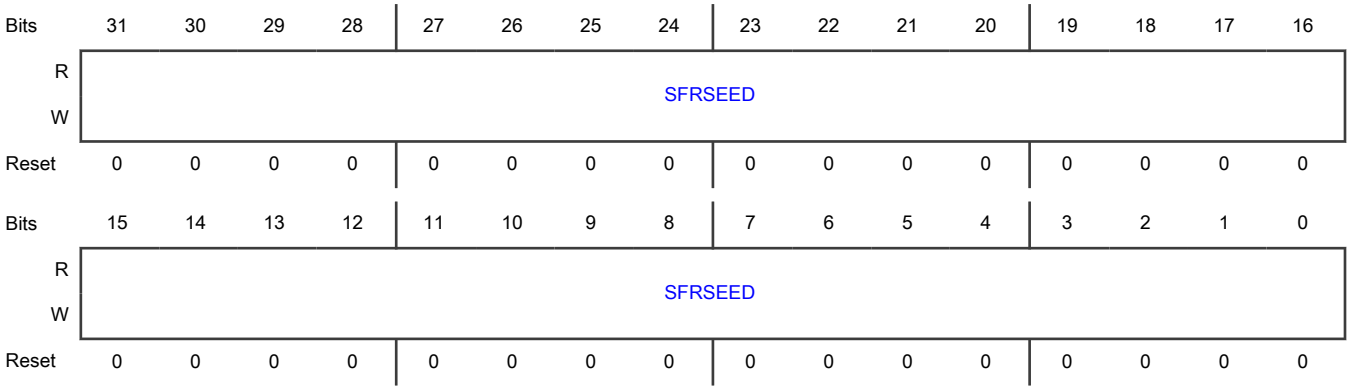
15.5.1.61 SFRSEED register for SFRMASK feature (SGI_SFRSEED)

Offset

Register	Offset
SGI_SFRSEED	D10h

Function
SFRSEED register for SFRMASK feature

Diagram



Fields

Field	Function
31-0 SFRSEED	Seed/mask used for sw level masking **NOTE** This field is volatile.

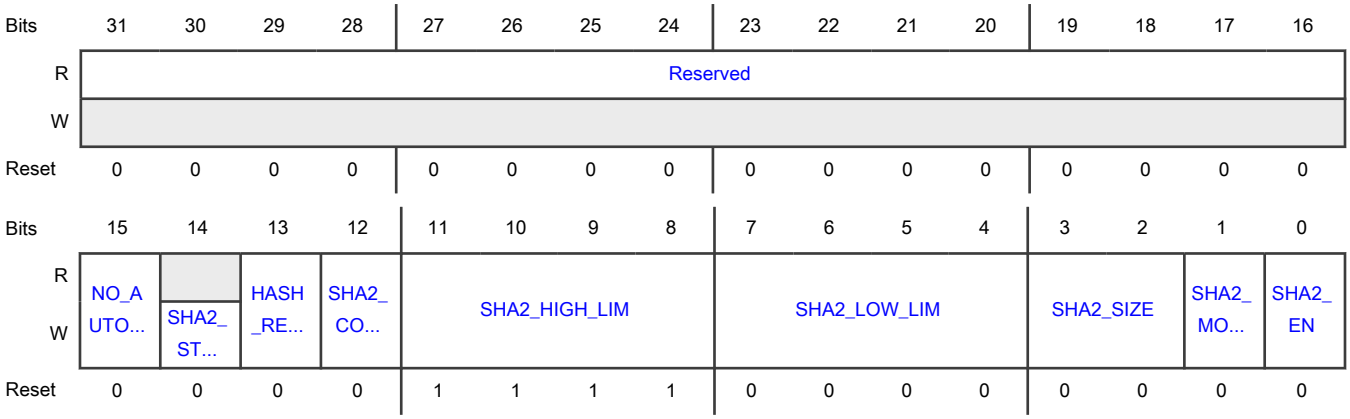
15.5.1.62 SHA Control Register (SGI_SHA2_CTRL)

Offset

Register	Offset
SGI_SHA2_CTRL	D14h

Function
SHA Control Register

Diagram



Fields

Field	Function
31-16 —	Reserved
15 NO_AUTO_INIT	SHA no automatic HASH initialisation **NOTE** This field is volatile. 0b - SHA automatic HASH initialisation 1b - No SHA automatic HASH initialisation
14 SHA2_STOP	STOP SHA AUTO mode **NOTE** This field is volatile. 0b - Keep running 1b - Stop auto mode
13	SHA HASH reload

Table continues on the next page...

Table continued from the previous page...

Field	Function
HASH_RELOAD	**NOTE** This field is volatile. 0b - No HASH reload 1b - HASH reload enabled
12 SHA2_COUNT_EN	SHA Calculation counter enable **NOTE** This field is volatile. 0b - SHA operation DOES NOT increment COUNT 1b - SHA operation DOES increment count
11-8 SHA2_HIGH_LIM	SHA FIFO high limit **NOTE** This field is volatile.
7-4 SHA2_LOW_LIM	SHA FIFO low limit **NOTE** This field is volatile.
3-2 SHA2_SIZE	Indicates SHA size **NOTE** This field is volatile. 00b - SHA-224 01b - SHA-256 10b - SHA-384(or SHA-224 if SHA-256 only) 11b - SHA-512 (or SHA-256 if SHA-256 only)
1 SHA2_MODE	SHA mode normal or automatic **NOTE** This field is volatile. 0b - SHA NORM Mode 1b - SHA AUTO Mode
0 SHA2_EN	SHA enable **NOTE** This field is volatile.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - SHA disabled 1b - SHA enabled

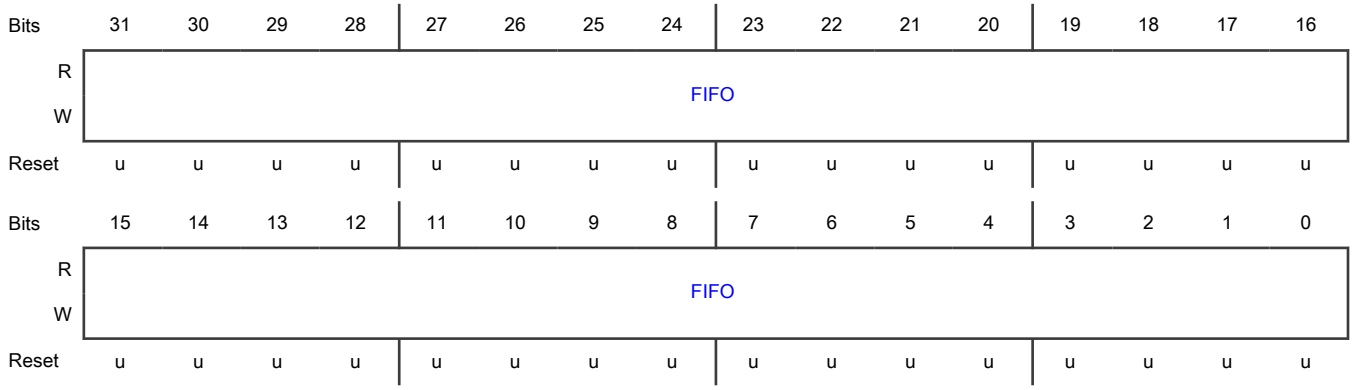
15.5.1.63 SHA FIFO lower-bank low (SGI_SHA_FIFO)

Offset

Register	Offset
SGI_SHA_FIFO	D18h

Function
SHA FIFO lower-bank low

Diagram



Fields

Field	Function
31-0 FIFO	SHA FIFO register **NOTE** This field is volatile.

15.5.1.64 SHA Configuration Reg (SGI_CONFIG)

Offset

Register	Offset
SGI_CONFIG	D1Ch

Function

SHA Configuration Reg

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved				SFR_SW_...	SPB_MAS...	SPB_SUP...	SHA_256...	Reserved		EDC	NUM_KEY			NUM_DATIN	
W																
Reset	0	0	0	0	u	u	u	u	0	0	u	u	u	u	u	u

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BUS_WID...	Reserved	FA	Reserved	COUNT_S...	KEY_DIG...	INTERNA...	HAS_GFM...	HAS_CMAC	HAS_MOV...	HAS_SHA	HAS_DES	HAS_AES	CC	CHINA	ROW
W																
Reset	u	0	u	0	u	u	u	u	u	u	u	u	u	u	u	u

Fields

Field	Function
31-28 —	Reserved
27 SFR_SW_MASK	ID_CFG_SGI_USE_SFR_SW_MASK is set **NOTE** This field is volatile.
26 SPB_MASKING	ID_CFG_SGI_SPB_MASKING is set **NOTE** This field is volatile.
25 SPB_SUPPORT	ID_CFG_SGI_SPB_SUPPORT is set **NOTE** This field is volatile.
24 SHA_256_ONLY	HAS SHA-256 ONLY **NOTE** This field is volatile.
23-22 —	Reserved
21 EDC	DATIN to KERNEL End-to-end EDC is enabled

Table continues on the next page...

Table continued from the previous page...

Field	Function
	NOTE This field is volatile.
20-18 NUM_KEY	NUMBER OR KEY REGBANKS **NOTE** This field is volatile.
17-16 NUM_DATIN	NUMBER OF DATIN REGBANKS **NOTE** This field is volatile.
15 BUS_WIDTH	0 - BUS_WIDTH=16, 1 - BUS_WIDTH=32 **NOTE** This field is volatile.
14 —	Reserved
13 FA	HAS FA protection **NOTE** This field is volatile.
12 —	Reserved
11 COUNT_SIZE	0 - COUNT=16, 1 - COUNT=32 **NOTE** This field is volatile.
10 KEY_DIGEST	HAS KEY DIGEST **NOTE** This field is volatile.
9 INTERNAL_PRNG	HAS INTERNAL PRNG **NOTE** This field is volatile.
8 HAS_GFMUL	HAS GFMUL **NOTE** This field is volatile.

Table continues on the next page...

Table continued from the previous page...

Field	Function
7 HAS_CMAC	HAS CMAC **NOTE** This field is volatile.
6 HAS_MOVEM	HAS MOVEM **NOTE** This field is volatile.
5 HAS_SHA	HAS SHA **NOTE** This field is volatile.
4 HAS_DES	HAS DES **NOTE** This field is volatile.
3 HAS_AES	HAS AES **NOTE** This field is volatile.
2 CC	SGI Diversified for 'CC' **NOTE** This field is volatile.
1 CHINA	SGI Diversified for 'CHINA' **NOTE** This field is volatile.
0 ROW	SGI Diversified for 'ROW' **NOTE** This field is volatile.

15.5.1.65 SHA Configuration 2 Reg (SGI_CONFIG2)

Offset

Register	Offset
SGI_CONFIG2	D20h

Function

SHA Configuration 2 Reg

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved								DES_NUM_SBOXES				DES_USED			
W																
Reset	0	0	0	0	0	0	0	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	Reserved					AES_KEYSIZE		AES_NUM_SBOXES					AES_USED			
W																
Reset	0	0	0	0	0	u	u	u	u	u	u	u	u	u	u	u

Fields

Field	Function
31-25 —	Reserved
24-20 DES_NUM_SBOXES	Number of DES sboxes **NOTE** This field is volatile.
19-16 DES_USED	**NOTE** This field is volatile. 0000b - Dakar 0001b - Danube 0010b - Depicta 0011b - Digi 0100b - Date 0101b - Desert 0110b-1111b - RFU
15-11 —	Reserved
10-9 AES_KEYSIZE	Indicates which AES key size has been selected. **NOTE** This field is volatile.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - 128 0nly 10b - 256 only 11b - All key sizes
8-4 AES_NUM_SBOXES	Number of AES sboxes **NOTE** This field is volatile.
3-0 AES_USED	**NOTE** This field is volatile. 0000b - Apollo 0001b - Aegis 0010b - Ayna 0011b - Athenium 0100b - Ajax 0101b - Aegis_hs 0110b - Athenium_hs 0111b - ATE 1000b - ATOM 1001b - Asterix 1010b-1111b - RFU

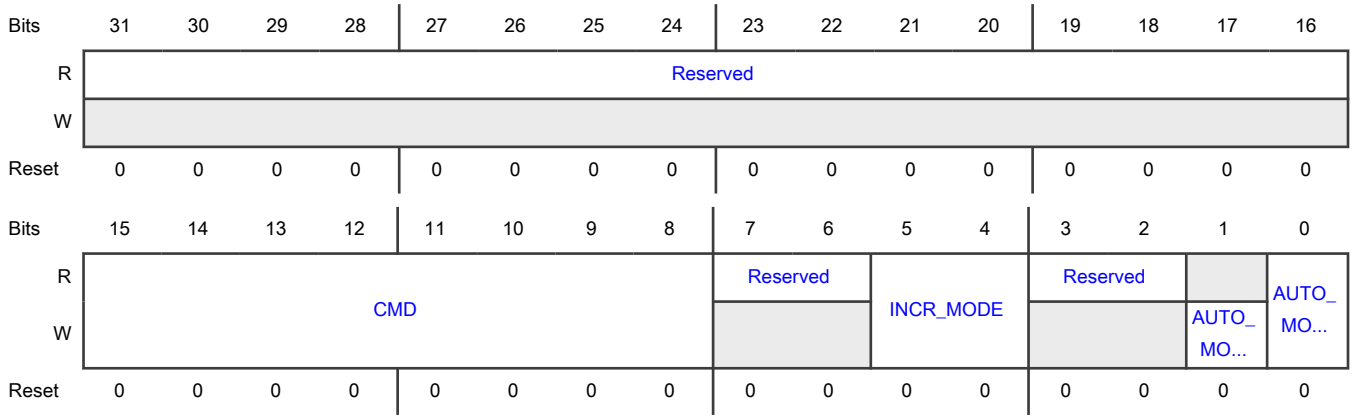
15.5.1.66 SGI Auto Mode Control register (SGI_AUTO_MODE)

Offset

Register	Offset
SGI_AUTO_MODE	D24h

Function
SGI Auto Mode Control register

Diagram



Fields

Field	Function
31-16 —	Reserved
15-8 CMD	Auto mode of operation 0000_0000b - ECB mode 0000_0001b - CTR mode 0000_0010b - CBC mode 0000_0011b - CBCMAC mode 0001_0000b - Key Wrap/Unwrap (128 bit key data) 0001_0001b - Key Wrap/Unwrap (256 bit key data)
7-6 —	Reserved
5-4 INCR_MODE	CTR increment mode 00b - 2**32 increment mode 01b - 2**64 increment mode 10b - 2**96 increment mode 11b - 2**128 increment mode
3-2 —	Reserved
1 AUTO_MODE_STOP	auto_mode_stop **NOTE** This field is volatile.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	1b - exit auto mode as soon as the data has been emptied
0 AUTO_MODE_EN	auto_start_en 1b - auto mode has been selected

15.5.1.67 SGI Auto Mode Control register (SGI_AUTO_DMA_CTRL)

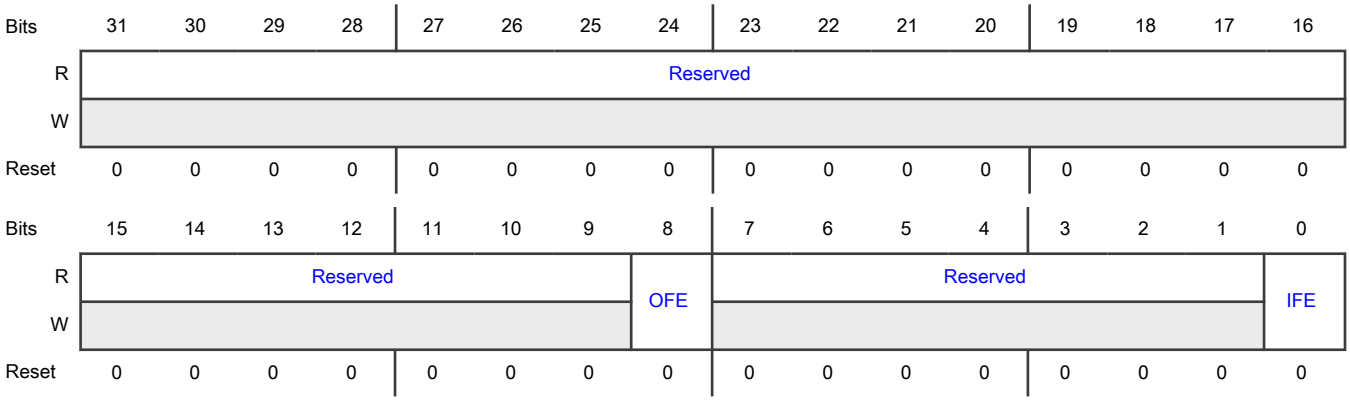
Offset

Register	Offset
SGI_AUTO_DMA_CTRL	D28h

Function

SGI Auto Mode Control register

Diagram



Fields

Field	Function
31-16 —	Reserved
15-9 —	Reserved
8 OFE	Ouput FIFO DMA Enable 0b - DMA handshake disabled

Table continues on the next page...

Table continued from the previous page...

Field	Function
	1b - DMA handshake enabled
7-1 —	Reserved
0 IFE	Input FIFO DMA Enable 0b - DMA handshake disabled 1b - DMA handshake enabled

15.5.1.68 SGI internal PRNG SW seeding register (SGI_PRNG_SW_SEED)

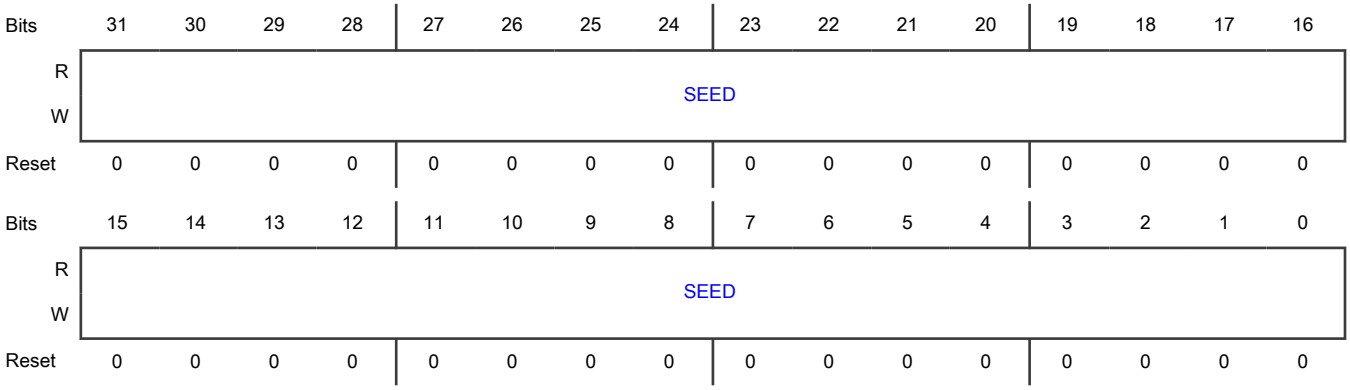
Offset

Register	Offset
SGI_PRNG_SW_SEED	D30h

Function

SGI internal PRNG SW seeding register

Diagram



Fields

Field	Function
31-0 SEED	32-bits SEED field. A write to the SEED field will seed the internal PRNG

15.5.1.69 SGI Key Control SFR (SGI_KEY_CTRL)

Offset

Register	Offset
SGI_KEY_CTRL	D40h

Function
SGI Key Control SFR

Diagram

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

Fields

Field	Function
31-0 KEY_WO	SGI Key control register(1-bit per KEY SFR) 1'b0 - Key SFR is readable 1'b1 - Key SFR is not-readable(write-only) **NOTE** This field is volatile.

15.5.1.70 Wrapped key read SFR (SGI_KEY_WRAP)

Offset

Register	Offset
SGI_KEY_WRAP	D50h

Function
Wrapped key read SFR

Diagram

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

Fields

Field	Function
31-0 KW_DATA	Field contains wrapped key, auto-updated by HW for each word **NOTE** This field is volatile.

15.5.1.71 SGI Version (SGI_VERSION)

Offset

Register	Offset
SGI_VERSION	F08h

Function
SGI Version

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved														MILESTONE	
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	X				Y1				Y2				Z			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-18 —	Reserved
17-16 MILESTONE	Release milestone **NOTE** This field is volatile. 00b - PREL 01b - BR 10b - SI 11b - GO
15-12 X	Major revision number in X.Y1Y2.Z, e.g. 1.20.3. **NOTE** This field is volatile.
11-8 Y1	Minor revision number 1 in X.Y1Y2.Z, e.g. 1.20.3. **NOTE** This field is volatile.
7-4 Y2	Minor revision number 2 in X.Y1Y2.Z, e.g. 1.20.3. **NOTE** This field is volatile.
3-0 Z	Extended revision number in X.Y1Y2.Z, e.g. 1.20.3. **NOTE** This field is volatile.

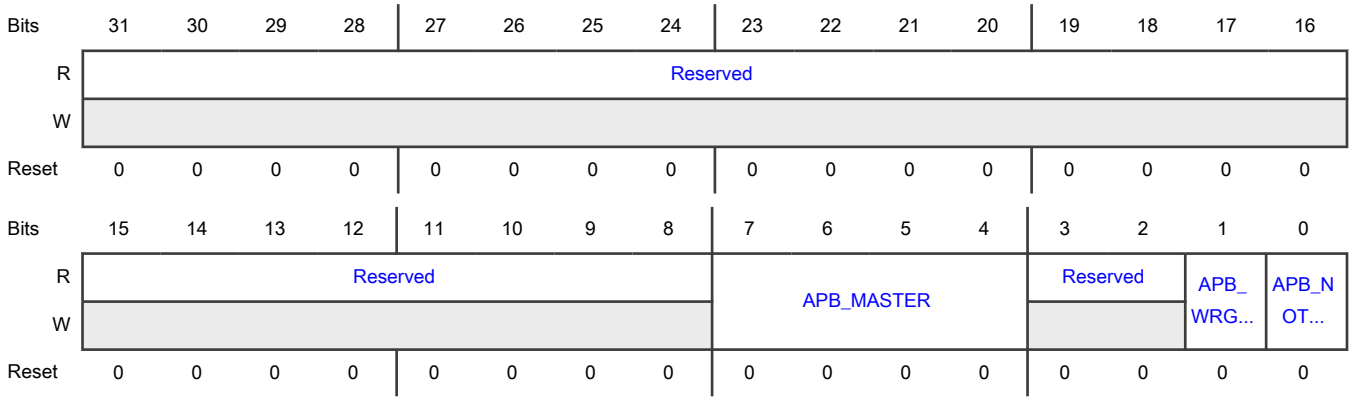
15.5.1.72 Access Error (SGI_ACCESS_ERR)

Offset

Register	Offset
SGI_ACCESS_ERR	FC0h

Function
Access Error

Diagram



Fields

Field	Function
31-16 —	Reserved
15-8 —	Reserved
7-4 APB_MASTER	APB Master that triggered first APB error (APB_WRGMD or APB_NOTAV) **NOTE** This field is volatile.
3-2 —	Reserved
1 APB_WRGMD	APB Error: Wrong access mode **NOTE** This field is volatile.
0 APB_NOTAV	APB Error: address not available **NOTE** This field is volatile.

15.5.1.73 Clear Access Error (SGI_ACCESS_ERR_CLR)

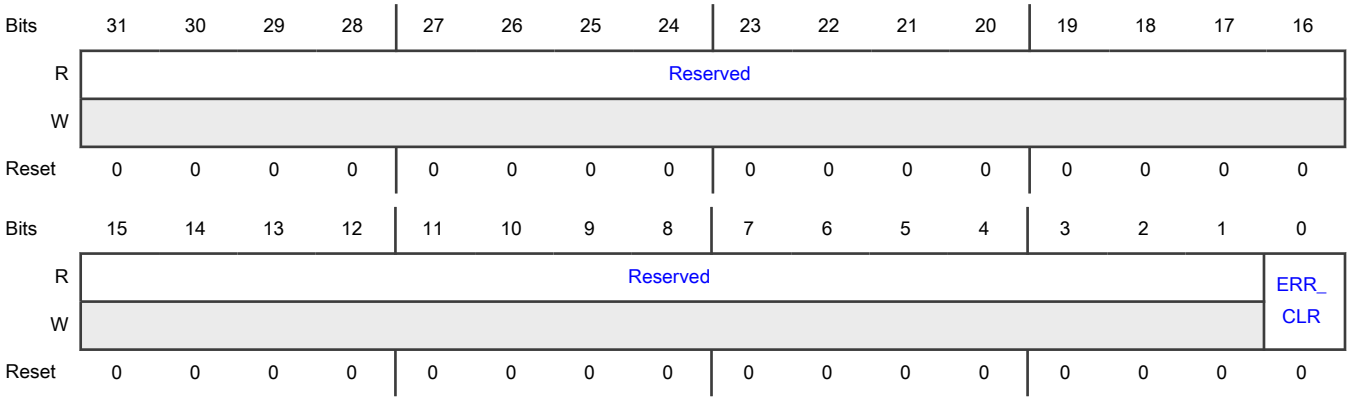
Offset

Register	Offset
SGI_ACCESS_ERR_CLR	FC4h

Function

Clear Access Error

Diagram



Fields

Field	Function
31-1 —	Reserved
0 ERR_CLR	Write to reset SGI_ACCESS_ERR SFR. **NOTE** This field is volatile.

15.5.1.74 Interrupt status (SGI_INT_STATUS)

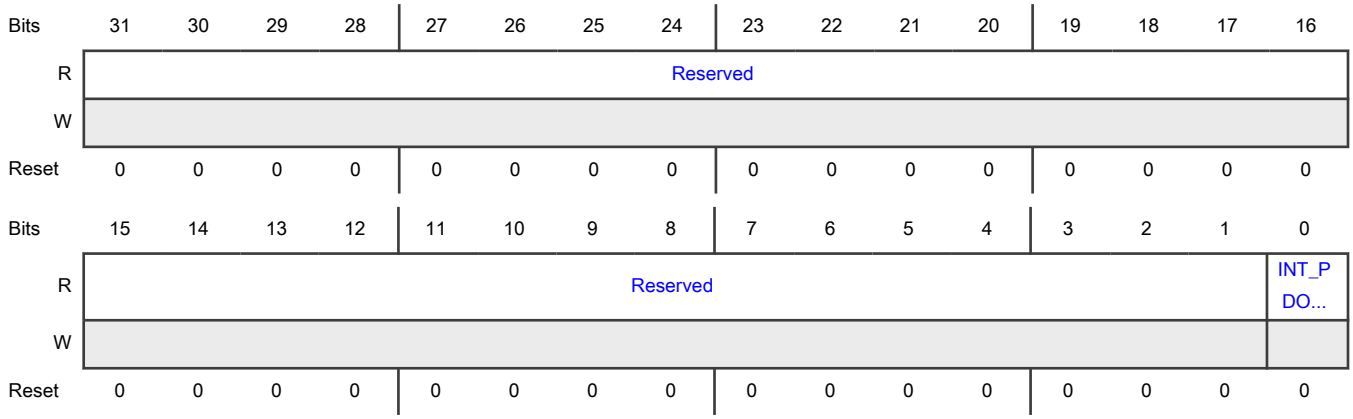
Offset

Register	Offset
SGI_INT_STATUS	FE0h

Function

Interrupt status

Diagram



Fields

Field	Function
31-16 —	Reserved
15-1 —	Reserved
0 INT_PDONE	Interrupt status flag: INT_PDONE is set independent from the interrupt enable SGI_INT_ENABLE.INT_EN. In case SGI_INT_ENABLE.INT_EN=1, an interrupt towards the CPU is triggered when INT_PDONE is set (level triggered). INT_PDONE is not cleared by SGI hardware but has to be cleared by software **NOTE** This field is volatile.

15.5.1.75 Interrupt enable (SGI_INT_ENABLE)

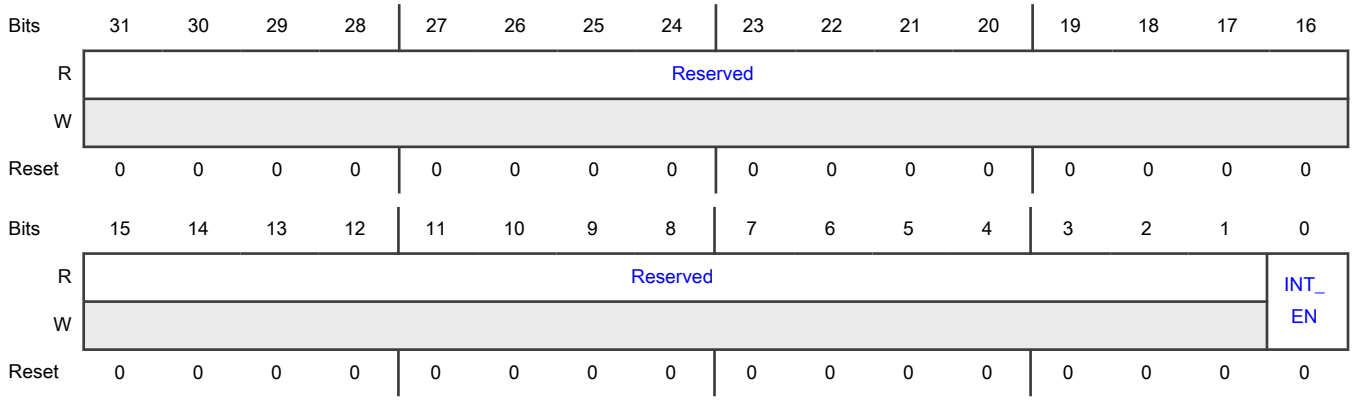
Offset

Register	Offset
SGI_INT_ENABLE	FE4h

Function

Interrupt enable

Diagram



Fields

Field	Function
31-1 —	Reserved
0 INT_EN	Interrupt enable bit

15.5.1.76 Interrupt status clear (SGI_INT_STATUS_CLR)

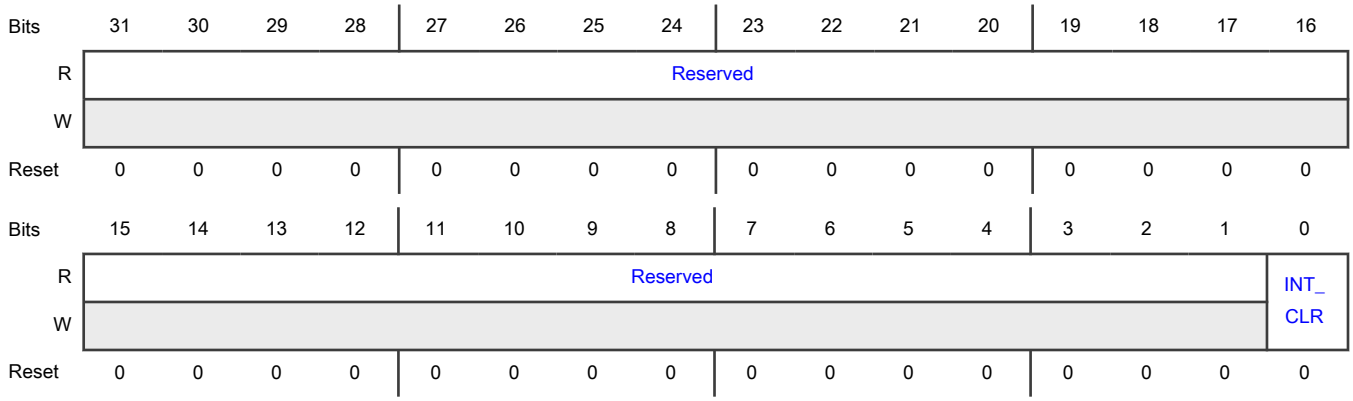
Offset

Register	Offset
SGI_INT_STATUS_CLR	FE8h

Function

Interrupt status clear

Diagram



Fields

Field	Function
31-1 —	Reserved
0 INT_CLR	Write to clear interrupt status flag (SGI_INT_STATUS.INT_PDONE=0). **NOTE** This field is volatile.

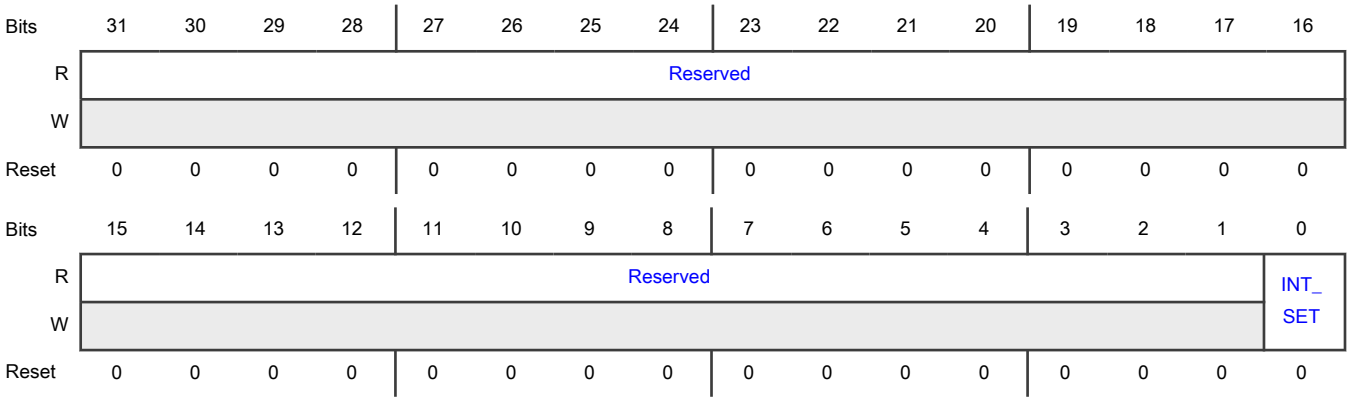
15.5.1.77 Interrupt status set (SGI_INT_STATUS_SET)

Offset

Register	Offset
SGI_INT_STATUS_SET	FECh

Function
Interrupt status set

Diagram



Fields

Field	Function
31-1 —	Reserved
0 INT_SET	Write to set interrupt status flag (SGI_INT_STATUS.INT_PDONE=1) to trigger a SGI interrupt via software, e.g. for debug purposes. **NOTE** This field is volatile.

15.5.1.78 Module ID (SGI_MODULE_ID)

Offset

Register	Offset
SGI_MODULE_ID	FFCh

Function
Module ID

Diagram

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

Fields

Field	Function
31-0 PLACEHOLDER	Module ID

15.6 Terms and definitions

NOTE

The following terms will only have the given meaning when used in this document within single quotes (i.e. 'crypto operation')

Table 110.

Term	Meaning
register bank	The N *128-bit bank of registers that are used to store data/ keys
bank	One of the 'N' 128-bit register banks, also referred to as bank[n], where 'n' is an integer in range {0…N}

Table continues on the next page...

Table 110. (continued)

Term	Meaning
block	One of the two 64-bit parts of a 'Bank'. These are referred to as 'Upper Block(UB)' and 'Lower Block(LB)'.
bus access	A bus 'read' or 'write'
crypto operation	A 'crypto operation' refers to an AES, DES, SHA, GFMUL or CMAC operation(including when only doing a TRIGGER_UP operation)
exored/exoring	The performing of the logical XOR operation
flush/flushed	Refers to loading 'blocks' in the 'register bank' with random data
kernel	Refers to AES,DES or GFMUL coprocessor engine
share	One of the two entities used in splitting secure data using a masking scheme.
words	One of the four 32-bit words within a 'bank', numbered w0 to w3.
xorwr	The feature that allows writing a register where the written value is the xor of the 'current value of the register' and the 'value being written' (new_value = current_value ^ write_value)
Full-Flush	The sequence of flushing all register banks, and kernels.
Error-Flush	A 'Full Flush' triggered as response to a 'security alert'.
Auto-Flush	A 'Full Flush' triggered in response to 'prng_ready' asserting.
User-Flush	A 'Full Flush' triggered in response to 'i_flush' asserting.

Table 111.

Acronym	Explanation
AES	Advanced Encryption Standard
DES	Data Encryption Standard
DFF	D-type Flip-Flop
EDC	Error Detection and Correction
GCM	Galois/Counter Multiplier
HW	Hardware
IV	Initialisation Vector
NC	No Change
RNG	Random Number Generator
RO	Read-Only

Table continues on the next page...

Table 111. (continued)

Acronym	Explanation
SFR	Special Function Register
SHA	Secure Hash Algorithm
SW	Software
TDES	Triple-DES
TI	Threshold Implementation
WO	Write-Only

Chapter 16
Public-Key Crypto Coprocessor (PKC)

16.1 Chip-specific PKC information

Table 112. Reference links to related information[1]

Topic	Related module	References
Full description	PKC	PKC
System memory map		System memory map
Clocking		See the chapter "Clocking"
Signal multiplexing	Port control	See the chapter "Signal multiplexing"
System debug		See the chapter "Debug"

1. For all the reference sections and chapters mentioned in this table, refer the Reference Manual.

16.1.1 Module instance

This device has one instance of the PKC module.

16.1.2 Security considerations

For secure applications, NXP recommends that this module is configured for secure and privileged access only to prevent unintentional or malicious modification of the system operation by nonsecure software.

16.1.3 Parity error check

PKC supports parity checking on it's RAM. Any parity errors that are detected are reported through the Error Recording Module (ERM). See the *ERM* chapter in *MCXA 255, and A256 Reference Manual*.

16.1.4 PKC RAM

System SRAM A1 and A2 are shared with PKC. PKC RAM interface is 64-bit. SRAM A1 is connected to the lower 32 bits of PKC RAM interface, and SRAM A2 is connected to the higher 32 bits. The RAM_INTERLEAVE[INTERLEAVE] bit in SYSCON should be set when CPU and other bus initiators read or write PKC RAM to operate with PKC.

16.2 Overview

PKC (Public-Key Crypto Coprocessor) is a security IP hardware, which can perform basic arithmetic and logical operations on multi-precision integers. Its purpose is to provide hardware acceleration to the software when executing public-key cryptography computations. The interfacing is done mainly using special function registers (SFRs) that provide the following functionality:

- Control and configuration SFRs to configure and start a PKC calculation
- Status SFRs to report status information (busy, carry, zero-result, fault detected)
- Layer 0 (L0): parameter set registers to define the calculation mode and operands (via start address and length)
- Layer 1 (L1): micro-coding SFRs to configure and start a PKC calculation via its micro-coding interface
- Layer 2 (L2): universal pointer fetch registers to define the DMA pointer addresses and amount of calculations that are executed in a series

The PKC kernel consists of the arithmetic unit (layer 0 (L0)) and a micro-coding state machine (layer 1 (L1)). The arithmetic unit supports the multiplication of a PKC word (64 bits) with a multi-precision integer (MPI) and addition, subtraction, shift operation and logical functions (AND, OR, XOR) of two multi-precision integers. The micro-coding state machine is used to define more complex

calculation functions, like full-size multiplication or modular multiplication. Via the universal-pointer fetch complete sequences of L0 and L1 operations can be executed.

For any kind of PKC operation the data needs to be first loaded into the PKC RAM. Afterward, software can configure the PKC via the SFR-IF and start the operation. During the execution of an operation PKC automatically fetches data via the RAM and the AHB L2 interfaces. Once the operation is completed the results can be read from the PKC RAM.

16.3 Block diagram

The PKC mainly consists of an interface shell (PKC control) and the PKC kernel. The interface shell connects the PKC kernel to the SFRs and implements the hardware for the universal-pointer fetch.

The PKC is interfaced via its advanced peripheral bus (APB) using special function registers (SFRs).

The PKC implements an AHB master to provide DMA access for the universal pointer unit in layer 2 (L2). This AHB interface requires only read accesses. A second read/write DMA to the PKC accessible RAM (PKC RAM) is required for arithmetic calculation of the PKC kernel itself in layer 0 (L0).

The PKC provides further error and interrupt signals that need to be connected to the relevant system controller. A basic overview of the PKC interfaces and its typical integration into the system is shown in Figure 50.

The PKC kernel consists of the arithmetic unit in L0 and a micro-coding state machine in layer 1 (L1). The arithmetic unit supports the multiplication of a 64 bit PKC word with a multi-precision integer (MPI) and addition, subtraction, shift/rotate operation and logical functions (AND, OR, XOR only) of two multi-precision integers. The micro-coding state machine is used to define more complex calculation functions, like full-size multiplication or modular multiplication. Via the universal-pointer fetch complete sequences of L0 and L1 operations can be executed.

The following is the PKC block diagram:

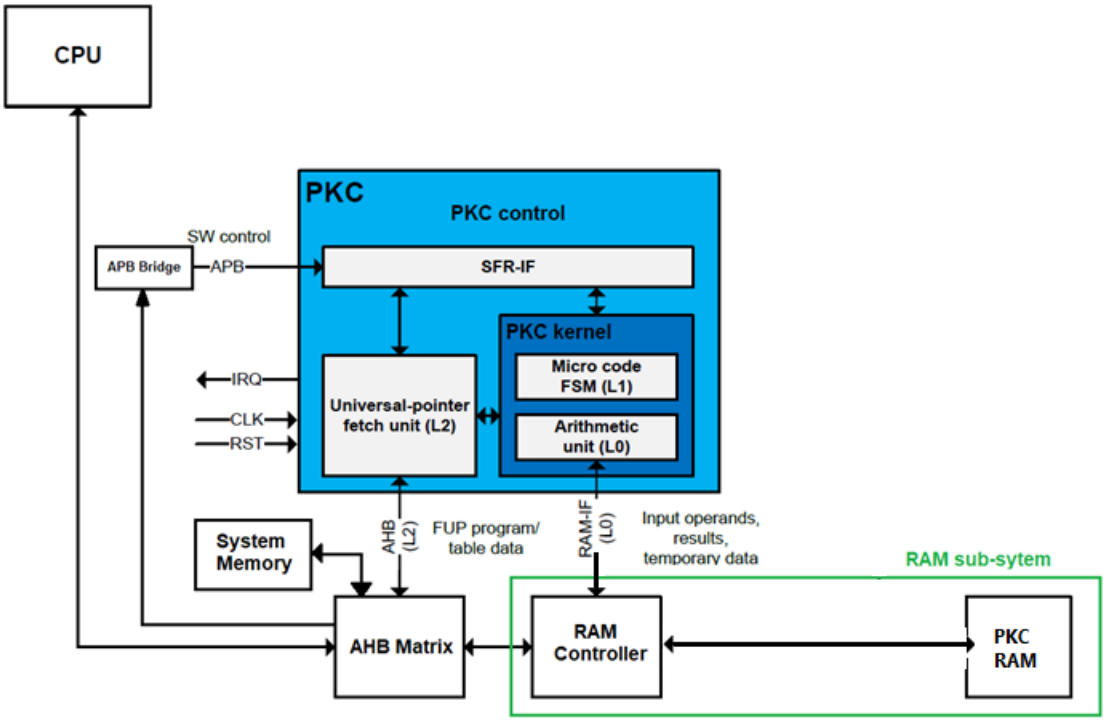


Figure 50. Overview of the PKC interfaces and its typical integration into the system.

16.4 Functional description

A PKC calculation is executed using the following steps, e.g., for a single addition R=Y+Z using the direct programming mode (L0):

1. Release PKC coprocessor from reset (could also be done later but increases the noise source during parameter and operand loading)
2. Load addition operands to PKC accessible RAM
3. Prepare parameter set for PKC addition, e.g. using parameter set 1
    a. Calculation mode: addition (PKC_MODE1)
    b. Start address of Y operand (PKC_XYPTR1[YPTR]
    c. Start address of Z operand (PKC_ZRPTR1[ZPTR])
    d. Start address of result area (PKC_ZRPTR1[RPTR])
    e. Length of operands (PKC_LEN1[LEN])
    f. Start calculation by setting the 'go' bit for the chosen parameter set: PKC_CTRL[GOD1]=1
4. Load addition operands to PKC accessible RAM
5. Poll for end of PKC calculation using the combined 'busy' flag: PKC_STATUS[ACTIV]
6. Read result of addition from RAM at address PKC_ZRPTR[RPTR]
7. Check PKC status register for carry (PKC_STATUS[CARRY]), and zero (PKC_STATUS[ZERO])

16.4.1 PKC control parameter sets

The PKC Control implements two parameter sets to define the PKC calculation. The parameter set contains all required information to define a single calculation: calculation mode, location of all inputs, length of inputs. Two calculations can be defined using the two dedicated parameter sets to support fast switching between two calculations.

A parameter set contains the following fields:

- PKC_MODEi[MODE] (8 bits) - calculation mode or start address for micro code (MC)
- PKC_XYPTRi[XPTR] (16 bits) - start address for X operand
- PKC_XYPTRi[YPTR] (16 bits) - start address for Y operand
- PKC_ZRPTRi[ZPTR] (16 bits) - start address for Z operand or constant operand for CONST or shift and rotate calculations (lower 8 bits only)
- PKC_ZRPTRi[RPTR] (16 bits) - start address for calculation result R
- PKC_LENi[LEN] (16 bits) - length of Y operand, defines also length of Z and R (may differ from length defined in SFR but depends on the chosen calculation mode, e.g. for op-code 0x01 (multiply-accumulate) Z is one PKC word bigger and R is two PKC words bigger than Y)
- PKC_LENi[MCLEN] (16 bits) - loop counter for MC execution, e.g. for MC pattern (MC sequence) for plain multiplication

The following is an example of plain multiplication (fixed MC pattern 0x13, parameter set 1):

```
@ RAM address 0x2004_0000
X: 0xE9794B15AAF151CF991D686D31D95C20263E5D725DE2CB9B9C51AEFBC167E8C2
@ RAM address 0x2004_0200
Y: 0x5A4D00195A39B947F732CB60CE97B0CA83D54C982ED5EC06ADB0785842550D31
@ address 0x2004_0400
R: 0x525ADDF852BEDFD7E81B1E7BCAF241DBC9582598B710677515542B839D462C87_
     531FC208AF12E6D76066B522F209506E228A834A83067B6F632C3B34981F6722

PKC_MODE1.MODE = 0x013 --> start address of Plain multiplication
PKC_XYPTR1.XPTR = 0x0000 --> start address of X operand in RAM
PKC_XYPTR1.YPTR = 0x0200 --> start address of Y operand in RAM
PKC_ZRPTR1.ZPTR --> not used/required
PKC_ZRPTR1.RPTR = 0x0400 --> start address of result R in RAM
PKC_LEN1.LEN = 0x0020 --> length of Y operand (32 bytes = 4 PKC-words of 64 bits)
```

```
PKC_LEN1.MCLEN = 0x0020 --> length of X operand (32 bytes = 4 PKC-words of 64 bits)
PKC_CTRL.GF2CONV = 0 --> execute in non-GF(2)

Calculation is started with PKC_CTRL.GOM1 --> MC pattern calculation with parameter set 1
```

The granularity of the pointers is one byte. The implementation is such that the pointers used by the CPU can be reused.

Example:

- CPU address of data word in PKC RAM: 0x2204_12EF
- PKC pointer value, e.g., for PKC_XYPTRi[XPTR]: 0x12EF

As the minimum operand size of the PKC coprocessor is 64 bits the least significant 3 bits are ignored and not forwarded to the PKC kernel. The same holds for the most significant bits that are not required due to the limitation in PKC RAM. Even though the bits are redundant for the PKC all bits are stored within the PKC SFRs.

All operands in RAM and the pointers and length registers need to be aligned to the word size of the PKC kernel.

Table 113. Alignment for operands in RAM, pointers and length registers

PKC_CTRL[REDMUL]	Multiplier size / PKC word	Security alarm in case PKC_LENi[LEN] (or MCLEN if used) is ...	Operand and register (pointer/length) alignment
00b	default (64 bit)	< 0x08	default value referring to native multiplier size indicated via PKC_VERSION[MULSIZE] = 10b (64 bit)
01b	-	-	Reserved (triggers an error)
10b	64 bit	< 0x08	64 bit, least significant 3 bits [2:0] of pointer/length registers are ignored, for shift/rotate operation only PKC_ZPTRi[5:0] is evaluated, the upper bits are ignored (no error is triggered if set)
11b	-	-	Reserved (triggers an error)

The security alarm for PKC_LENi[MCLEN] is only triggered in a L1 operation when DecrTBNZ is used with a zero MCLEN value.

16.4.1.1 Layer 0: Operating PKC through parameter set interface (L0)

With L0 the basic operations of the PKC coprocessor can be started, e.g. simple addition, shift and logical operations or multiplication of a single PKC word with a long integer. A single calculation is started by setting the write only PKC_CTRL[GOD1] bit for parameter set 1 and PKC_CTRL[GOD2] for parameter set 2. The PKC_STATUS[GOANY] flag is set automatically when a calculation start has been triggered and cleared by HW as soon as the PKC kernel has acknowledged the start of the calculation. The end of the calculation is indicated by PKC_STATUS[ACTIV]=0 assuming there is no other pending calculation.

See PKC arithmetic unit (L0) for details of Layer 0 calculation modes supported by the PKC arithmetic unit.

16.4.1.2 Layer 1: Operating PKC through micro code interface (L1)

PKC micro codes can be used to execute complex calculations (e.g. modular multiplication or addition) that consist of a combination of single PKC Control and L0 calculations. Beside the L0 calculations, loops, conditional and unconditional jumps and simple pointer adaptations (increment/decrement) are supported. The operand pointers are defined using one of the two available parameter sets. The micro code can be started either using parameter set 1 or parameter set 2 via the dedicated control flags PKC_CTRL[GOM1] and PKC_CTRL[GOM2].

Execution of a PKC micro code is done by defining the 8 bit start address (via the PKC_MODE1 SFR) that is used as a pointer within the MC state machine. The PKC coprocessor kernel provides hard-coded calculation modes to implement commonly used

operations, for example, a full size multiplication, modular addition, subtraction, reduction and multiplication in in Z_N (which is normal computation modulo N) and in Galois Field GF(2) (which are computations in binary field).

User programmable micro code is not supported in this version of the IP as the size of the flexible MC table is zero. Attempts to read or write the PKC_MCDATA register will trigger a security alarm.

The calculation itself is started using a dedicated write-only control flag PKC_CTRL[GOM1] or PKC_CTRL[GOM2]. The PKC_STATUS[GOANY] flag is set when GOM1 or GOM2is set and cleared by HW after the micro execution has been started. Setting a GOM bit while GOANY is set will trigger a security alarm. The currently used parameter set can be updated after associated STATUS[LOCKED] bit is cleared. The parameter set is latched, such that pointer modifications (increment/decrement) during the MC execution do not overwrite the initial parameter set.

The end of a micro code execution is indicated by ACTIV=0.

In addition to the parameter-set registers, two extra internal pointer registers S and T are available during micro-code execution. The two pointer registers S and T can be used, e.g., as temporary storage (i.e., loaded with value from other pointer register via a MC instruction) and are reset to zero with the start of each L1 calculation triggered by setting either GOM1 or GOM2.

The parameter set registers including the start address and loop counter are latched with the start of a MC operation and can be overwritten once the MC start has been acknowledged.

For detailed information on the Micro Code Layer see Micro code (MC) interface (L1).

16.4.1.3 Layer 2: Operating PKC through universal pointer fetch unit (L2)

The PKC Control is able to perform a predefined number of calculations with different parameters without interaction of the CPU. To achieve this, two universal pointers (PKC_UPTRand PKC_UPTRT) and a counter register (PKC_ULEN) have been implemented. By this a flow of PKC calculations can be defined and executed in a series to implement complex cryptographic operations, e.g. EC point arithmetic. Using the universal pointer fetch single PKC calculation (L0) and PKC micro codes (L1) can be executed in any order.

The universal pointer, a 32 bit address pointer, is used to fetch the parameters for the single PKC operation. The parameters can be located in any embedded memory (ROM, RAM, Flash) that is accessible via the L2 DMA. All memory access restrictions that are valid for the L2 DMA are also true for the universal pointer fetch. SFRs cannot be accessed using the PKC universal pointer fetch unit. In case of an invalid L2 DMA access (e.g. access to an invalid memory area) triggered by the PKC L2 calculation and detected by the system a security alarm is triggered and PKC_ACCESS_ERR[AHB] error flag is set.

The pipe operation is started by setting the PKC_CTRL[GOU] control bit. The end of the pipe calculation is indicated via the PKC_STAT[ACTIV] status flag. Between start and end of the pipe operation no further CPU interaction is required. The operand length is taken from parameter set 1 or parameter set 2 depending on CALCparam0[6] and needs to be defined upfront. The length SFRs cannot be overwritten while GOU is set. A violation triggers a security alarm. The loop counter (MCLEN) for the MC execution needs to be initialized via SFR write before starting the universal pointer fetch operation as well. The loop counter must not be overwritten during the complete universal pointer fetch calculation.

A single PKC kernel calculation (L0 or L1) triggered by universal pointer fetch can be repeated up to 15 times using the parameter CALCparam0[3:0].

PKC_UPTR, PKC_UPTRT and PKC_ULEN must not be updated during the L2 calculation (while GOANY=1). Updates of these SFRs during a calculation are not covered by HW but must be ensured by the SW operating the PKC.

The PKC pipe operation is based on the two universal pointer registers PKC_UPTR, PKC_UPTRT and PKC_ULEN that defines the number of calculations in a row. Starting a universal pointer fetch with PKC_ULEN equal 0 will trigger a security alarm.

PKC_UPTR points at the start address of the PKC Universal Pointer fetch program (short FUP program). The FUP program parameters for a single calculation have to be stored and are fetched in the following order:

- CALCparam0, CALCparam1, XPTRind, YPTRind, ZPTRind, RPTRind (= data entry) or
- CALCparam0, CALCparam1, CRC32[0], CRC32[1], CRC32[2], CRC32[3] (= CRC entry (when CRC mode is enabled))

CALCparam0 and CALCparam1 define the calculation mode and potential universal pointer control options for future use. The bit fields of CALCparam0 are described in Table 114

Table 114. CALCparam0 bit fields.

Bits	Description
3:0	Repeat counter that defines how often the operation/pattern is repeated (redundant for CRC entry).
4	0 → Indicates a data entry. The 4 bytes following CALCparam1 define the table indexes for the operand pointer (X, Y, Z, R). 1 → Indicates a CRC entry. The CRC32 preload value is provided with the 4 bytes following CALCparam1. CALCparam1 is redundant for a CRC entry.
5	Reserved
6	0 → Operand length and micro code loop counter are taken from parameter set 1 (redundant for CRC entry). 1 → Operand length and micro code loop counter are taken from parameter set 2 (redundant for CRC entry).
7	0 → Direct PKC calculation (L0) is executed. CALCparam1 defines the calculation mode. 1 → Micro code execution (L1) is started. CALCparam1 defines the start address.

Each parameter of the FUP program has a size of one byte. The index parameters (XPTRind,…) are multiplied by two and added to PKC_UPTRT to fetch the selected parameters from the FUP table. PKC_UPTR is incremented automatically by one for each byte that is fetched via the L2 DMA. It is mandatory that the FUP table and program pointers and as such also the underlying parameters are 16-Bit word aligned as the least significant bit of PKC_UPTR and PKC_UPTRT is ignored.

PKC_UPTRT is used as a pointer to the start address of the FUP table. The FUP table contains the pointers, lengths and operands (for CONST) for the internal parameter registers used for the pipe operation. The table entries must have a size of two bytes. The two bytes variables have to be stored with the endianness that corresponds to the CPU architecture. Up to 256 successive 2-bytes variables are supported. PKC_UPTRT remains unchanged during the complete pipe operation.

NOTE

When modifying the content of the FUP program it is recommended to update the PKC_UPTR register (via an SFR write access) prior to starting a new L2 calculation to clear the internal code-fetch buffer.

NOTE

If the cache is enabled and the FUP table content is updated, it is recommended to invalidate the cache prior to starting a new L2 calculation.

The PKC_ULEN SFR defines the number of FUP parameter sets (FUP program entries) which shall be used for calculation in serial order without interaction of the CPU. After each parameter set fetch this register is decremented, the calculations are stopped when the PKC_ULEN register has been decremented to zero and the last operation has been completed.

The write-only PKC_CTRL[GOU] control bit is used to start the parameter fetch via L2 DMA using the universal pointer. The calculation starts immediately after the complete parameter set is loaded. During the calculation of the PKC new parameters for the next operation are fetched from ROM, RAM or Flash as long as PKC_ULEN is not zero.

The following steps are needed to perform a PKC pipe operation:

1. Store all needed operands in the RAM (accessible by PKC coprocessor)
2. Calculate the pointers and lengths for the parameter sets and define the MODE for the calculation.
3. Store these 2 byte parameters (word aligned) in the FUP table in any embedded memory.
4. Store the FUP program in the order described above in any embedded memory.
5. Calculate the checksum of the FUP program and store it at the end of the FUP program parameter bytes (optional: only when FUP program integrity check via CRC checksum is used)

6. Set PKC_ULEN to the number of calculations to be done.
7. Set PKC_UPTR to the start address of the FUP program and PKC_UPTRT to the start address of the FUP table.
8. Set the PKC_CTRL[GOU] bit to start the parameter fetch and the calculation.

When PKC_ULEN has been decremented to zero, GOANY is cleared by the PKC Control after the start of the last operation in the series. The end of the calculation is indicated by clearing the flag PKC_STATUS[ACTIV] and setting the PKC_INT_STATUS[INT_PDONE] bit. Setting one of the GOx bits while GOANY is set will trigger a security alarm.

To reduce the amount of required DMA accesses by the PKC Control a cache is implemented to latch the last used parameter values. With this, the parameters for the next calculations can be written during the running calculation to minimize the gap between two calculations.

The usage of the cache is configurable and can be disabled completely such that for each single PKC operation (either atomic macro function or micro code instruction) all calculation parameters are fetched via the universal pointer fetch. The cache can be cleared using a dedicated write only control bit (PKC_CTRL[CLRCACHE]). The cache is also invalidated if the SFR PKC_UPTRT is written or the cache is disabled via PKC_CTRL[CACHE_EN].

In case the FUP table content in RAM or Flash is updated it is mandatory to invalidate the cache to ensure that old calculation parameters available in the cache are no longer used.

NOTE

When executing two FUP programs from consecutive memory locations, one could in principle omit the update of the PKC_UPTR pointer prior to starting the second FUP program (as PKC_UPTR is auto incremented and will already point to the correct location). However, it is important to note that this will not work if the cache is invalidated in between (i.e., by asserting PKC_CTRL[CLRCACHE] or writing to SFR PKC_UPTR).

For integrity protection of the fetched FUP program a CRC32 checksum calculation can be enabled. Bit 4 in CALCparam0 indicates if the entry is a data entry (bit 4 is 0) or a CRC entry (bit 4 is 1).

In case of a CRC entry, all other bits of CALCparam0 and CALCparam1 are redundant and ignored. The remaining four bytes of the entry contain the CRC-32 pre-load value in little endian notation.

The CRC-32 is also known as AUTODIN-II (e.g., used in Ethernet, AAL5) with the following generator polynomial: x32+x26+x23+x22+x16+x12+x11+x10+x8+x7+x5+x4+x2+x+1. CRC input data order is lsb first. The CRC check is disabled with each start of a new L2 calculation, triggered by setting PKC_CTRL[GOU]. The first fetch of a CRC entry activates the CRC calculation. This allows implementing FUP programs with and without CRC check (security/performance trade-off).

If the CRC calculation is not active and a CRC entry is fetched, the pre-load value is loaded into the CRC checksum register in the CRC unit. At the end of the FUP program (UPLEN equal to 0) the CRC checksum register should contain zero.

If the CRC calculation is already active and a CRC entry is fetched, the new pre-load value is XORed to the CRC checksum register (expected to be zero) such that the zero check is only needed at the very end of the FUP program.

The input bytes for the CRC calculation are processed in the same order as stored in the FUP program, resp. read and processed during the L2 calculation. In case of a data entry the byte order is: CALCparam0, CALCparam1, XPTRind, YPTRind, ZPTRind, RPTRind .

For CRC entries (CALCparam0.4=1) CALCparam0 and CALCparam1 are not used for the CRC checksum calculation but skipped, only the data entries are considered to (re-)initialize the CRC checksum. This integrity-protection concept supports multiple entry and exit points in the FUP program, easing extendability and reuse of FUP programs.

Example - initialization with pre-calculated CRC init value

1. CRC entry (init to 0x663da777): 0x10 0x00 0x77 0xA7 0x3D 0x66
2. FUP calculation 1 (addition): 0x00 0x0A 0x01 0x23 0x45 0x67
3. FUP calculation 2 (shift right): 0x00 0x15 0x89 0xab 0xcd 0xef
4. Final CRC32 equal 0x00000000

Example - init with zero, adding expected CRC32 at the end of the FUP program:

1. CRC entry 1 (init to 0x00000000): 0x10 0x00 0x00 0x00 0x00 0x00

2. FUP calculation 1 (addition): 0x00 0x0A 0x01 0x23 0x45 0x67
3. FUP calculation 2 (shift right): 0x00 0x15 0x89 0xab 0xcd 0xef
4. CRC entry 2 (zero checksum using the calculated CRC32 (0x3AE03616)): 0x10 0x00 0x16 0x36 0xE0 0x3A
5. Final CRC32 equal 0x00000000

16.4.2 PKC Kernel Description

The PKC coprocessor kernel is divided into a PKC arithmetic unit and the PKC interface shell.

The PKC arithmetic unit supports the multiplication of a PKC word (64 bits) with a multi-precision integer (MPI) and addition, subtraction, shift operation and the logical functions (AND, OR, XOR) of two multi-precision integers.

The PKC interface shell connects the arithmetic unit to the interface register sets. Furthermore a micro coding (MC) interface is provided to define more complex calculation functions, like full-size multiplication or modular multiplication.

16.4.2.1 PKC arithmetic unit (L0)

The PKC arithmetic unit is the main building block for the arithmetic operations to support public key cryptography. Table 115 lists all the calculation modes (L0 operations) supported by the PKC arithmetic unit.

Table 115. L0 calculation modes supported by the PKC arithmetic unit

Op-Code	Symbol	Operation	Description
0x00	MUL	R = X_0 · Y	Pure multiplication of a PKC word X_0 by a MPI Y
0x02	MAC	R = X_0 · Y + Z	Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z
0x03	MAC_NEG	R = X_0 · Y - Z	Multiply-Subtract of a PKC word X_0 with a MPI Y and a MPI Z
0x04	MUL_GF2	R = X_0 · Y {GF(2)}	Pure multiplication of a PKC word X_0 by a MPI Y over GF(2)
0x06	MAC_GF2	R = X_0 · Y + Z {GF(2)}	Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z over GF(2)
0x09	NEG	R = -Z	Two's complement of MPI Z
0x0a	ADD	R = Y + Z	Addition of a MPI Y with a MPI Z
0x0b	SUB	R = Y - Z	Subtraction of a MPI Y with a MPI Z
0x0d	AND	R = Y AND Z	Logical AND of a MPI Y with a MPI Z
0x0e	OR	R = Y OR Z	Logical OR of a MPI Y with a MPI Z
0x0f	XOR	R = Y XOR Z	Logical XOR of a MPI Y with a MPI Z
0x10	MAC_CONST_GF2	R = X_0 · Y + CONST {GF(2)}	Multiply-Accumulate of a PKC word X_0 with a MPI Y and a CONST over GF(2)
0x12	MAC_CONST	R = X_0 · Y + CONST	Multiply-Accumulate of a PKC word X_0 with a MPI Y and a CONST
0x13	MAC_NEG_CONST	R = X_0 · Y - CONST	Multiply-Subtract of a PKC word X_0 with a MPI Y and a CONST
0x14	SHL	R = Y << CONST	Shift left of a MPI Y by CONST positions
0x15	SHR	R = Y >> CONST	Shift right of a MPI Y by CONST positions

Table continues on the next page...

Table 115. L0 calculation modes supported by the PKC arithmetic unit (continued)

Op-Code	Symbol	Operation	Description
0x16	ROTL	R = rotateleft (Y) by CONST	Rotate left of a MPI Y by CONST positions
0x17	ROTR	R = rotateright (Y) by CONST	Rotate right of a MPI Y by CONST positions
0x1a	ADD_CONST	R = Y + CONST	Addition of a MPI Y with a CONST
0x1b	SUB_CONST	R = Y - CONST	Subtraction of a MPI Y with a CONST
0x1d	AND_CONST	R = Y AND CONST	Logical AND of a MPI Y with a CONST
0x1e	OR_CONST	R = Y OR CONST	Logical OR of a MPI Y with a CONST
0x1f	XOR_CONST	R = Y XOR CONST	Logical XOR of a MPI Y with a CONST
0x20	MUL1	Reg_i = $X_0 \cdot Y_0$ mod 2^wordsize	Plain multiplication of 1 PKC word modulo word size
0x22	MACC	R = $X_0 \cdot Y$ + {c,Z}	Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z extended by carry overhead
0x24	MUL1_GF2	Reg_i = $X_0 \cdot Y_0$ mod 2^wordsize {GF(2)}	Plain multiplication of 1 PKC word modulo word size over GF(2)
0x26	MACC_GF2	R = $X_0 \cdot Y$ + Z {GF(2)}	Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z over GF(2)
0x2A	ADDC	R = (c,Y) + Z	Addition of a MPI Y incl. carry overhead with a MPI Z
0x2B	SUBC	R = (c,Y) - Z	Subtraction of a MPI Y incl. carry overhead with a MPI Z
0x2D	LSB0s	T = LSB0s(Z)	Determines the number of consecutive zero bits (up to 8) of MPI Z starting from LSB
0x2F	MSB0s	T = MSB0s(Z)	Determines the number of consecutive zero bits (up to 8) of MPI Z starting from MSB
0x3E	CONST	R = CONST	Initializes memory with a CONST
0x4B	CMP	Y-Z	Compares two MPIs by subtracting MPI Z from MPI Y
0x62	MACCR	R = (Reg_i $\cdot$ Y + {c,Z}) >> wordsize	Multiply-Accumulate of the internal register Reg_i with a MPI Y and a MPI Z extended by carry overhead
0x66	MACCR_GF2	R = (Reg_i $\cdot$ Y + Z) >> wordsize {GF(2)}	Multiply-Accumulate of the internal register Reg_i with a MPI Y and a MPI Z over GF(2)
0x6A	ADD_Z0	R = Y + Z_0	Addition of a MPI Y with a single PKC word Z_0
0x6F	XOR_Z0	R = Y XOR Z_0	XOR of a MPI Y with a single PKC word Z_0

Detailed description of the L0 calculation modes

Table 116. Plain Multiplication

Pure multiplication of a PKC word X_0 by a MPI Y		
$R = X_0 \cdot Y$	Op-code = 0x00	Symbol: MUL
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
Result	RLEN = YLEN + 1 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR and RPTR == YPTR	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	0x04 - pure multiplication over GF(2) / MUL_GF2	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 117. Plain Multiplication with Addition

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z. MPI Z must be one PKC word longer than Y.		
$R = X_0 \cdot Y + Z$	Op-code = 0x02	Symbol: MAC
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	

Table continues on the next page...

Table 117. Plain Multiplication with Addition (continued)

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z. MPI Z must be one PKC word longer than Y.		
$R = X_0 \cdot Y + Z$	Op-code = 0x02	Symbol: MAC
	ZLEN = YLEN + 1 $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN + 2 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR, RPTR == YPTR and RPTR == ZPTR	
Updated status flags[1]	CARRY=1 in case of overflow (equals lsb(R[RLEN-1])), ZERO	
GF2 conversion opcode	0x06 - multiply-accumulate over GF(2) / MAC_GF2	
Execution time	4·(YLEN+2) PKC clock cycle	

1. Carry overflow is stored in carry flag and in RAM, in the most significant word of the result (R[RLEN-1]).

Table 118. Plain Multiplication with Subtraction

Multiply-Subtract of a PKC word X_0 with a MPI Y and a MPI Z. MPI Z must be one PKC word longer than Y.		
$R = X_0 \cdot Y - Z$	Op-code = 0x03	Symbol: MAC_NEG
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN + 1 $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN + 2 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR, RPTR == YPTR and RPTR == ZPTR	

Table continues on the next page...

Table 118. Plain Multiplication with Subtraction (continued)

Multiply-Subtract of a PKC word X_0 with a MPI Y and a MPI Z. MPI Z must be one PKC word longer than Y.		
$R = X_0 \cdot Y - Z$	Op-code = 0x03	Symbol: MAC_NEG
Updated status flags[1]	CARRY=1 in case result is negative (equals lsb(R[RLEN-1])), ZERO	
GF2 conversion opcode	0x06 - multiply-accumulate over GF(2) / MAC_GF2	
Execution time	4·(YLEN+2) PKC clock cycle	

1. Carry overflow is stored in carry flag and in RAM, in the most significant word of the result (R[RLEN-1]).
   A CARRY=1 indicates a negative result in two's complement notation.

Table 119. Plain Multiplication in GF2

Pure multiplication of a PKC word X_0 with a MPI Y over GF(2).		
$R = X_0 \cdot Y$ {GF(2)}	Op-code = 0x04	Symbol: MUL_GF2
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
Result	RLEN = YLEN + 1 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR and RPTR == YPTR	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	0x04 (no impact)	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 120. Plain Multiplication with Addition in GF2

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z over GF(2). MPI Z must be one PKC word longer than Y.		
$R = X_0 \cdot Y + Z$ {GF(2)}	Op-code = 0x06	Symbol: MAC_GF2
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN + 1 $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN + 2 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR, RPTR == YPTR and RPTR == ZPTR	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	0x06 (no impact)	
Execution time	4·(YLEN+2) PKC clock cycle	

Table 121. Negation

Two's complement of MPI Z		
R = -Z	Op-code = 0x09	Symbol: NEG
Parameter	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Z in bytes
Operands	ZLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = ZLEN	

Table continues on the next page...

Table 121. Negation (continued)

Two's complement of MPI Z		
R = -Z	Op-code = 0x09	Symbol: NEG
	$R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == ZPTR	
Updated status flags[1]	CARRY=1 in case result is negative (only equal '1' if ZERO=1), ZERO	
GF2 conversion opcode	0x09 (no impact)	
Execution time	4·(YLEN+1) PKC clock cycle	

1. A CARRY=1 indicates a negative result in two's complement notation.

   Applying NEG(Z) on Z=0 will result in R=0 and CARRY=0

Table 122. Addition

Addition of a MPI Y and a MPI Z .		
R = Y + Z	Op-code = 0x0A	Symbol: ADD
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y and Z in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR	
Updated status flags	CARRY=1 in case of overflow, ZERO	

Table continues on the next page...

Table 122. Addition (continued)

Addition of a MPI Y and a MPI Z .		
R = Y + Z	Op-code = 0x0A	Symbol: ADD
GF2 conversion opcode	0x0f - logical xor / XOR	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 123. Subtraction

Subtraction of a MPI Y and a MPI Z .		
R = Y - Z	Op-code = 0x0B	Symbol: SUB
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y and Z in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR	
Updated status flags[1]	CARRY=1 in case of negative result, ZERO	
GF2 conversion opcode	0x0f - logical xor / XOR	
Execution time	4·(YLEN+1) PKC clock cycle	

1. A CARRY=1 indicates a negative result in two's complement notation.

Table 124. Logical AND/OR/XOR

Logical AND/OR/XOR of a MPI Y and a MPI Z .		
R = Y AND Z R = Y OR Z R = Y XOR Z	Op-code = 0x0D Op-code = 0x0E Op-code = 0x0F	Symbol: AND Symbol: OR Symbol: XOR
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y and Z in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	0x09 (no impact)	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 125. Plain Multiplication with Addition of a constant in GF2

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a CONST over GF(2). CONST = {0,...,255}.		
$R = X_0 \cdot Y + CONST$ {GF(2)}	Op-code = 0x10	Symbol: MAC_CONST_GF2
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Constant operand
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes

Table continues on the next page...

Table 125. Plain Multiplication with Addition of a constant in GF2 (continued)

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a CONST over GF(2). CONST = {0,...,255}.		
$R = X_0 \cdot Y + CONST$ {GF(2)}	Op-code = 0x10	Symbol: MAC_CONST_GF2
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	CONST = ZPTR[7:0]	
Result	RLEN = YLEN + 1 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR and RPTR == YPTR	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	0x10 (no impact)	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 126. Plain Multiplication with Addition of a constant

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a CONST. CONST = {0,...,255}.		
$R = X_0 \cdot Y + CONST$	Op-code = 0x12	Symbol: MAC_CONST
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Constant operand
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	CONST = ZPTR[7:0]	
Result	RLEN = YLEN + 2 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR and RPTR == YPTR	

Table continues on the next page...

Table 126. Plain Multiplication with Addition of a constant (continued)

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a CONST. CONST = {0,...,255}.		
$R = X_0 \cdot Y + CONST$	Op-code = 0x12	Symbol: MAC_CONST
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	0x10 - Multiply-Accumulate with a CONST in GF(2) / MAC_CONST_GF2	
Execution time	4·(YLEN+2) PKC clock cycle	

Table 127. Plain Multiplication with Subtraction of a constant

Multiply-Subtract of a PKC word X_0 with a MPI Y and a CONST. CONST = {0,...,255}.		
$R = X_0 \cdot Y - CONST$	Op-code = 0x13	Symbol: MAC_NEG_CONST
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Constant operand
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	CONST = ZPTR[7:0]	
Result	RLEN = YLEN + 2 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR and RPTR == YPTR	
Updated status flags[1]	CARRY=1 in case of negative result, ZERO	
GF2 conversion opcode	0x10 - Multiply-Accumulate with a CONST in GF(2) / MAC_CONST_GF2	
Execution time	4·(YLEN+2) PKC clock cycle	

1. A CARRY=1 indicates a negative result in two's complement notation.

Table 128. Shift and Rotate

Shift/Rotate left/right a MPI Y by CONST positions. CONST = {0,...,63}. Redundant most significant bits of CONST are ignored.		
R = shiftleft (Y) by CONST R = shiftright (Y) by CONST R = rotateleft (Y) by CONST R = rotateright (Y) by CONST	Op-code = 0x14 Op-code = 0x15 Op-code = 0x16 Op-code = 0x17	Symbol: SHL Symbol: SHR Symbol: ROTL Symbol: ROTR
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Shift/rotate factor, max. value depends on REDMUL setting
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	CONST = ZPTR[5:0] for REDMUL == 0x0 or 0x2	
Result	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR	
Updated status flags	CARRY=1 in case shifted/rotated out bit is '1', ZERO	
GF2 conversion opcode	no impact	
Execution time	SHL: 4·(YLEN+1) PKC clock cycle SHL, ROTL, ROTR: 4·(YLEN+2) PKC clock cycle	

NOTE

Internal register Reg_i is updated by SHR, ROTL and ROTR calculation with intermediate data:

- For SHR and CONST unequal zero $Reg_i = (Y << (wordsize - CONST))_{YLEN-1}$, with wordsize = 64-bit for REDMUL == 0x0 or 0x2. For CONST = 0 $Reg_i = Y_{YLEN-1}$
- For ROTL $Reg_i = (Y << CONST)_0$.
- For ROTR and CONST unequal zero $Reg_i = (Y << (wordsize - CONST))_0$, with wordsize = 64-bit for REDMUL == 0x0 or 0x2. For CONST = 0 $Reg_i = Y_{YLEN-1}$

Table 129. Addition with CONST

Addition of a MPI Y with a CONST. CONST = {0,...,255}.		
R = Y + CONST	Op-code = 0x1A	Symbol: ADD_CONST
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Value of constant operand CONST.
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	CONST = ZPTR[7:0]	
Result	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR	
Updated status flags	CARRY=1 in case of overflow, ZERO	
GF2 conversion opcode	0x1F - Logical XOR with constant / XOR_CONST	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 130. Subtraction with CONST

Subtraction of a MPI Y with a CONST. CONST = {0,...,255}.		
R = Y - CONST	Op-code = 0x1B	Symbol: SUB_CONST
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Value of constant operand CONST.
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	CONST = ZPTR[7:0]	
Result	RLEN = YLEN	

Table continues on the next page...

Table 130. Subtraction with CONST (continued)

Subtraction of a MPI Y with a CONST. CONST = {0,...,255}.		
R = Y - CONST	Op-code = 0x1B	Symbol: SUB_CONST
	$R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR	
Updated status flags[1]	CARRY=1 in case of negative result, ZERO	
GF2 conversion opcode	0x1F - Logical XOR with constant / XOR_CONST	
Execution time	4·(YLEN+1) PKC clock cycle	

1. A CARRY=1 indicates a negative result in two's complement notation.

Table 131. Logical AND/OR/XOR with CONST

Logical AND/OR/XOR of a MPI Y with a CONST. CONST = {0,...,255}.		
R = Y AND CONST R = Y OR CONST R = Y XOR CONST	Op-code = 0x1D Op-code = 0x1E Op-code = 0x1F	Symbol: AND_CONST Symbol: OR_CONST Symbol: XOR_CONST
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Value of constant operand CONST expanded to every byte.
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = LEN $CONST = ZPTR[7:0] \cdot 2^{8 \cdot (ZLEN-1)} + ZPTR[7:0] \cdot 2^{8 \cdot (ZLEN-2)} + ... + ZPTR[7:0] \cdot 2^8 + ZPTR[7:0]$	
Result	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	no impact	

Table continues on the next page...

Table 131. Logical AND/OR/XOR with CONST (continued)

Logical AND/OR/XOR of a MPI Y with a CONST. CONST = {0,...,255}.		
R = Y AND CONST R = Y OR CONST R = Y XOR CONST	Op-code = 0x1D Op-code = 0x1E Op-code = 0x1F	Symbol: AND_CONST Symbol: OR_CONST Symbol: XOR_CONST
Execution time	4·(YLEN+1) PKC clock cycle	

Table 132. Plain Multiplication modulo PKC wordsize

Plain multiplication of a PKC word X_0 by a PKC word Y_0 modulo PKC wordsize. Result is stored in internal register Reg_i		
$Reg_i = X_0 \cdot Y_0 \bmod 2^{wordsize}$	Op-code = 0x20	Symbol: MUL1
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to operand Y_0: Byte address of Y_0
	LEN[1]	don't care --> set to 1 PKC word, depends on REDMUL configuration
Operands	X_0	
	Y_0	
Result[2]	Reg_i	
Result-in-place	n/a	
Updated status flags	no result flags updated	
GF2 conversion opcode	0x24 - plain multiplication of a PKC word modulo wordsize over GF(2) / MUL1_GF2	
Execution time	4 PKC clock cycle	

1. Operand length is fixed to one PKC word (64-bit for REDMUL == 0x0 or 0x2). PKC_LENi.LEN is ignored.
2. Result is written to internal register Reg_i.

Table 133. Multiplication with Addition incl. previous carry overhead

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z extended by carry overhead. MPI Y and MPI Z must have equal length.		
$R = X_0 \cdot Y + \{c,Z\}$	Op-code = 0x22	Symbol: MACC
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0

Table continues on the next page...

Table 133. Multiplication with Addition incl. previous carry overhead (continued)

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z extended by carry overhead. MPI Y and MPI Z must have equal length.		
$R = X_0 \cdot Y + \{c,Z\}$	Op-code = 0x22	Symbol: MACC
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y and Z in bytes
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN $\{c,Z\} = CARRY \cdot b^{ZLEN} + Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN + 1 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR, RPTR == YPTR and RPTR == ZPTR	
Updated status flags	CARRY=1 in case of overflow, ZERO	
GF2 conversion opcode	0x26 - multiply-accumulate over GF(2) / MACC_GF2	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 134. Plain Multiplication modulo PKC wordsize in GF2

Plain multiplication of a PKC word X_0 by a PKC word Y_0 modulo PKC wordsize in GF(2). Result is stored in internal register Reg_i		
$Reg_i = X_0 \cdot Y_0 \bmod 2^{wordsize}$ {GF(2)}	Op-code = 0x24	Symbol: MUL1_GF2
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to operand Y_0: Byte address of Y_0
	LEN[1]	don't care --> set to 1 PKC word, depends on REDMUL configuration
Operands	X_0	
	Y_0	
Result[2]	Reg_i	
Result-in-place	n/a	

Table continues on the next page...

Table 134. Plain Multiplication modulo PKC wordsize in GF2 (continued)

Plain multiplication of a PKC word X_0 by a PKC word Y_0 modulo PKC wordsize in GF(2). Result is stored in internal register Reg_i		
$Reg_i = X_0 \cdot Y_0 \bmod 2^{wordsize}$ {GF(2)}	Op-code = 0x24	Symbol: MUL1_GF2
Updated status flags	no result flags updated	
GF2 conversion opcode	no impact	
Execution time	4 PKC clock cycle	

1. Operand length is fixed to one PKC word (64-bit for REDMUL == 0x0 or 0x2). PKC_LENi.LEN is ignored.
2. Result is written to internal register Reg_i.

Table 135. Multiplication with Addition in GF2

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z . MPI Y and MPI Z must have equal length.		
$R = X_0 \cdot Y + \{c,Z\}$ {GF(2)}	Op-code = 0x26	Symbol: MACC_GF2
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y and Z in bytes
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN + 1 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR, RPTR == YPTR and RPTR == ZPTR	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	no impact	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 136. Addition incl. previous carry overhead

Addition of a MPI Y extended by carry overhead and a MPI Z.		
R = {c,Y} + Z	Op-code = 0x2A	Symbol: ADDC
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y and Z in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $\{c,Y\} = CARRY \cdot b^{YLEN} + Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR	
Updated status flags	CARRY=1 in case of overflow, ZERO	
GF2 conversion opcode	0x0f - logical xor / XOR	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 137. Subtraction incl. previous carry overhead

Subtraction of a MPI Y extended by carry overhead and a MPI Z.		
R = {c,Y} - Z	Op-code = 0x2A	Symbol: SUBC
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y and Z in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $\{c,Y\} = CARRY \cdot b^{YLEN} + Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN	

Table continues on the next page...

Table 137. Subtraction incl. previous carry overhead (continued)

Subtraction of a MPI Y extended by carry overhead and a MPI Z.		
R = {c,Y} - Z	Op-code = 0x2A	Symbol: SUBC
	$Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR	
Updated status flags[1]	CARRY=1 in case of overflow or negative result, ZERO	
GF2 conversion opcode	0x0f - logical xor / XOR	
Execution time	4·(YLEN+1) PKC clock cycle	

1. CARRY=1 represents an overflow only in case input carry c has been set. If input carry c=0 a set CARRY flag for this calculation represents a negative result in two's complement notation.

Table 138. Least significant zero bits

Determines the number of consecutive zero bits (up to 8) of MPI Z starting from lsb. No data is written to RAM, only flags and internal T-register are updated. The result T is in the range of 0 to 8. T can only be used with L1 calculations.		
T = LSB0s(Z)	Op-code = 0x2D	Symbol: LSB0s
Parameter	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	LEN	Length of operand Z in bytes
Operands	ZLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result[1]	T-register	
Result-in-place	n/a	
Updated status flags	CARRY=0, ZERO=1 if LSB0s(Z)=0 resp. least significant bit of Z operand is '1' --> Z is odd	
GF2 conversion opcode	no impact	
Execution time	16 PKC clock cycle	

1. In case this operation is used in a L1 operation, register T is assigned and stable only after a CondJump. This is required for all MC instructions following the LSB0s calculation that read or write the T pointer: InDecPtr(Ptr_Sel=T), PreLoadT, Execute(Mode={LSB0s, MSB0s, calculation modes with CONST + ConfLoad(ConstSrc=1)})

Table 139. Most significant zero bits

Determines the number of consecutive zero bits (up to 8) of MPI Z starting from msb. No data is written to RAM, only flags and internal T-register are updated. The result T is in the range of 0 to 8. T can only be used with L1 calculations.		
T = MSB0s(Z)	**Op-code = 0x2E**	**Symbol: MSB0s**
Parameter	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	LEN	Length of operand Z in bytes
Operands	ZLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result[1]	T-register	
Result-in-place	n/a	
Updated status flags	CARRY=0, ZERO=1 if MSB0s(Z)=0 resp. most significant bit of Z operand is '1'	
GF2 conversion opcode	no impact	
Execution time	16 PKC clock cycle	

1. In case this operation is used in a L1 operation T is assigned and stable only after a CondJump. This is required for all MC instructions following the MSB0s calculation that read or write the T pointer: InDecPtr(Ptr_Sel=T), PreLoadT, Execute(Mode={LSB0s, MSB0s, calculation modes with CONST + ConfLoad(ConstSrc=1)})

Table 140. Initialization function

Initializes memory with a CONST. CONST = {0,...,255}.		
R = CONST	**Op-code = 0x3E**	**Symbol: CONST**
Parameter	ZPTR	Value of constant operand CONST expanded to every byte
	LEN	Length of result area R in bytes
Operands	ZLEN = LEN $CONST = ZPTR[7:0] \cdot 2^{8 \cdot (ZLEN-1)} + ZPTR[7:0] \cdot 2^{8 \cdot (ZLEN-2)} + ... + ZPTR[7:0] \cdot 2^8 + ZPTR[7:0]$	
Result	RLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	n/a	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	no impact	

Table continues on the next page...

Table 140. Initialization function (continued)

Initializes memory with a CONST. CONST = {0,...,255}.		
R = CONST	Op-code = 0x3E	Symbol: CONST
Execution time	4·(YLEN+1) PKC clock cycle	

Table 141. Compare

Compares two MPIs by subtracting MPI Z from MPI Y. No result is written to RAM, only flags are updated. Start address of MPI Y and MPI Z must not be equal.		
Y - Z	Op-code = 0x4B	Symbol: CMP
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	LEN	Length of operand Y and Z in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	n/a - only CARRY and ZERO status flag is updated	
Result-in-place	n/a	
Updated status flags	CARRY=1 in case of Z > Y, ZERO in case Y == Z	
GF2 conversion opcode	no impact	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 142. Multiplication with Addition incl. previous carry overhead ignoring least significant result word

Multiply-Accumulate of the internal register Reg_i with a MPI Y and a MPI Z extended by carry overhead ignoring least significant result word. MPI Z must be one word longer than Y. R is one PKC word bigger than Y but the least significant result word R[0] is withdrawn such that the resulting R has the same length as MPI Y.		
$R = Reg_i \cdot Y + \{c,Z\} >> wordsize$	Op-code = 0x62	Symbol: MACCR
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0

Table continues on the next page...

Table 142. Multiplication with Addition incl. previous carry overhead ignoring least significant result word (continued)

Multiply-Accumulate of the internal register Reg_i with a MPI Y and a MPI Z extended by carry overhead ignoring least significant result word. MPI Z must be one word longer than Y. R is one PKC word bigger than Y but the least significant result word R[0] is withdrawn such that the resulting R has the same length as MPI Y.		
$R = Reg_i \cdot Y + \{c,Z\} >> wordsize$	Op-code = 0x62	Symbol: MACCR
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	Reg_i	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN+1 $\{c,Z\} = CARRY \cdot b^{ZLEN} + Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN $R = R_{RLEN} \cdot b^{RLEN-1} + R_{RLEN-1} \cdot b^{RLEN-2} + ... + R_2 \cdot b + R_1$	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR	
Updated status flags	CARRY=1 in case of overflow, ZERO	
GF2 conversion opcode	0x66 - multiply-accumulate over GF(2) / MACCR_GF2	
Execution time	4·(YLEN+1) PKC clock cycle	

NOTE

Theoretically the result can have a carry overflow of 0x2 that is not covered in the carry status flag (CARRY=0 in this case). However in the typical use case of the Montgomery Multiplication and Reduction this can never happen. In case MACCR operation is used in a different context the max. possible result has to be considered.

An operation that writes to Reg_i (e.g. MUL1 (0x20)) has to be executed directly before executing MACCR. In case Reg_i is not initialized by a proper preceding calculation the result of the MACCR calculation may be undefined.

Table 143. Multiplication with Addition ignoring least significant result word in GF(2)

Multiply-Accumulate of the internal register Reg_i with a MPI Y and a MPI Z ignoring least significant result word in GF(2). MPI Z must be one word longer than Y. R is one PKC word bigger than Y but the least significant result word R[0] is withdrawn such that the resulting R has the same length as MPI Y.		
$R = Reg_i \cdot Y + Z >>$ wordsize {GF(2)}	Op-code = 0x66	Symbol: MACCR_GF2
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0

Table continues on the next page...

Table 143. Multiplication with Addition ignoring least significant result word in GF(2) (continued)

Multiply-Accumulate of the internal register Reg_i with a MPI Y and a MPI Z ignoring least significant result word in GF(2). MPI Z must be one word longer than Y. R is one PKC word bigger than Y but the least significant result word R[0] is withdrawn such that the resulting R has the same length as MPI Y.		
$R = Reg_i \cdot Y + Z >>$ wordsize {GF(2)}	Op-code = 0x66	Symbol: MACCR_GF2
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	Reg_i	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN+1 $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN $R = R_{RLEN} \cdot b^{RLEN-1} + R_{RLEN-1} \cdot b^{RLEN-2} + ... + R_2 \cdot b + R_1$	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	no impact	
Execution time	4·(YLEN+1) PKC clock cycle	

NOTE

An operation that writes to Reg_i (e.g. MUL1_GF2 (0x24)) has to be executed directly before executing MACCR_GF2. In case Reg_i is not initialized by a proper preceding calculation the result of the MACCR_GF2 calculation may be undefined.

Table 144. Addition with single PKC word Z_0

Addition of a MPI Y with a single PKC word Z_0 .		
$R = Y + Z_0$	Op-code = 0x6A	Symbol: ADD_Z0
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to single PKC word operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2)	

Table continues on the next page...

Table 144. Addition with single PKC word Z_0 (continued)

Addition of a MPI Y with a single PKC word Z_0 .		
$R = Y + Z_0$	Op-code = 0x6A	Symbol: ADD_Z0
	$Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	Z_0	
Result	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR	
Updated status flags	CARRY=1 in case of overflow, ZERO	
GF2 conversion opcode	0x6f - logical xor with single PKC word Z / XOR_Z0	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 145. XOR with single PKC word Z_0

XOR of a MPI Y with a single PKC word Z_0 .		
$R = Y$ XOR Z_0	Op-code = 0x6F	Symbol: XOR_Z0
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to single PKC word operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	Z_0	
Result	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	no impact	

Table continues on the next page...

Table 145. XOR with single PKC word Z_0 (continued)

XOR of a MPI Y with a single PKC word Z_0 .		
R = Y XOR Z_0	Op-code = 0x6F	Symbol: XOR_Z0
Execution time	4·(YLEN+1) PKC clock cycle	

16.4.2.2 Micro code (MC) interface (L1)

The MC programming interface uses simple microcode instructions to:

- Initialize a PKC L0 calculation (increment/decrement pointer, select pointer for calculation)
- Start a PKC L0 calculation
- Decrement a loop counter and branch if not zero
- Load pointer into internal shadow T register (incl. auto-increment/decrement)
- Select parameter sets
- Unconditional jumps and conditional jumps based on carry and zero flag

Two internal pointers S (S-Ptr) and T (T-Ptr) are implemented to support pointer handling, e.g. to backup and recover a pointer. S and T pointer can be loaded and updated with the PreLoadT instruction and InDecPtr only (they are not directly accessible through the register memory map). The T pointer can furthermore be used as alternative to the CONST value, defined via ConfLoad. On top the result of the LSB0s and MSB0s operation is stored in T pointer automatically. S pointer is for storage only. With each start of a new L1 calculation the internal pointer registers S-Ptr and T-Ptr are reset to zero.

All pointer and length values used in the micro-code layer have word granularity that depends on the PKC_CTRL[REDMUL] setting, e.g.:

- X pointer is PKC_XYPTRi[XPTR] >> 3 (REDMUL == 0x0 or 0x2)
- operand length is PKC_LENi[LEN] >> 3 (REDMUL == 0x0 or 0x2)
- loop counter LC is PKC_LENi[MCLEN] >> 3 (REDMUL == 0x0 or 0x2)

The coding of the micro code instructions is shown in the Table 146.

Table 146. PKC micro code instructions

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	MC instruction
1	Mode (L0 operation)							Execute
0	1	Addr (relative address +31,...,-32)						DecrTBNZ
0	0	1	Const_Src	Conf_Load				ConfLoad
0	0	0	1	Incr	Ptr_Sel			InDecPtr
0	0	0	0	1	Cond			CondJump (uses both bytes)
Addr (relative address +127,...,-128)								
0	0	0	0	0	1	Reserved		Reserved

Table continues on the next page...

Table 146. PKC micro code instructions (continued)

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	MC instruction
0	0	0	0	0	0	1	Const	PreLoadT (two bytes)
Exch	T2Ptr	AddEN	IncrEN	Add_Incr	Ptr_Sel			
8-bit Constant value								
0	0	0	0	0	0	0	1	Jump (two bytes)
Addr (relative address +127,...,-128)								
0	0	0	0	0	0	0	0	Exit

MC instruction: Execute

Starts execution of the L0 operation indicated by the parameter Mode. When trying to start a new operation while a previous one is still ongoing, the Execute instruction will block (until next operation can be started). Table 147 describes the usage of the different parameters of the Execute instruction.

Table 147. Description of the parameters used within the Execute instruction

Parameter	Description
Mode[6:0]	Op-code of native L0 operation that should be started (see PKC arithmetic unit (L0) for a list of all available L0 op-codes).

MC instruction: DecrTBNZ

The DecrTBNZ (Decrement, Test and Branch if Not Zero) instruction decrements the internal loop-counter register (LC) and performs a relative jump (i.e., +31 to -32 is added to program counter) in the MC table in case the loop counter is not zero. For example, if the DecrTBNZ instruction is located at address a and the offset value (relative address) is b (in two's complement representation), the final jump address where the next instruction will be fetched from is a + b. In case the loop counter is zero after the decrement, it is reloaded again with the preload value from PKC_LENi.MCLEN and increments the program counter by one (i.e., next instruction in table is executed afterwards). Table 148 describes the usage of the different parameters of the DecrTBNZ instruction. In case the DecrTBNZ instruction is executed on a zero loop counter value a security alarm is triggered.

Note: The DecrTBNZ has no relation to the internal T pointer of the MC program interface. The T pointer is neither read nor updated by the DecrTBNZ MC instruction.

Table 148. Description of the parameters used within the DecrTBNZ instruction

Parameter	Description
Addr[5:0]	Relative address that is used in case of a jump. The address is in the range of +31 to -32 and is added to the current program counter.

MC instruction: ConfLoad

The ConfLoad instruction configures the loading of the operands, i.e., how pointers from a register set are loaded into the PKC kernel. The main purpose of this configuration is to be able to use different pointers for different PKC kernel operands or the result, e.g., to be able to use an operand previously stored at RPTR as an input operand (X, Y, Z) for a following calculation or to write the result of a calculation to RAM address defined by an input operand pointer (e.g. ZPTR). A dedicated flag is used to indicate the source of the constant value provided to the kernel (either lower byte of ZPTRor T-Ptr is used). The specified configuration on

a ConfLoad instruction stays active until the next ConfLoad instruction is executed or the pattern is exited (default configuration is used). Table 149 describes the usage of the different parameters of the ConfLoad instruction.

Table 149. Description of the parameters used within the ConfLoad instruction

Parameter	Description
Const_Src	Define source of the constant value provided to the PKC kernel • 0: constant value is taken from lower byte of ZPTR • 1: constant value is taken from lower byte of T-Ptr
Conf_Load[3:0]	Selects a configuration that is used for loading operands into the PKC kernel (e.g., using configuration 0001 changes behavior of addition operation from R = Y + Z to R = R + Z). • 0000: XYZR ☐ XYZR • 0001: ZRZR ☐ XYZR • 0010: XYRR ☐ XYZR • 0011: XYXY ☐ XYZR • 0100: XXYX ☐ XYZR • 0101: XZRR ☐ XYZR • 0110: XRXR ☐ XYZR • 0111: XXYR ☐ XYZR • 1000: XZRZ ☐ XYZR • 1001: XYRY ☐ XYZR • 1010: YRZZ ☐ XYZR • 1011: Reserved (XYZR ☐ XYZR) • 1100: Reserved (XYZR ☐ XYZR) • 1101: Reserved (XYZR ☐ XYZR) • 1110: Reserved (XYZR ☐ XYZR) • 1111: Reserved (XYZR ☐ XYZR) • 1111: Reserved (XYZR ☐ XYZR)

MC Instruction: InDecPtr

The InDecPtr instruction increments or decrements a selected pointer/parameter. Table 150 describes the usage of the different parameters of the InDecPtr instruction.

Table 150. Description of the parameters used within the InDecPtr instruction

Parameter	Description
Incr	Selects increment or decrement 0: decrement pointer/parameter selected via Ptr_Sel 1: increment pointer/parameter selected via Ptr_Sel
Ptr_Sel[2:0]	Selects a pointer/parameter

Table continues on the next page...

Table 150. Description of the parameters used within the InDecPtr instruction (continued)

Parameter	Description
	• 000: select pointer X • 001: select pointer Y • 010: select pointer Z • 011: select pointer R • 100: select pointer S • 101: select pointer T • 110: select parameter LEN (operand length) • 111: select parameter LC (loop counter)

MC instruction: CondJump

This instruction performs a conditional jump and consists of two bytes. The first byte contains the jump condition; different jump conditions can be selected: carry flag, zero flag, combination of carry and zero flag. The second byte contains the relative address that is added to the program counter in case the defined jump condition is fulfilled. For example, if the CondJump instruction is located at address a and a + 1 and the offset value (relative address) is b (in two's complement representation), the final jump address where the next instruction will be fetched from is a + 1 + b. If the jump condition is not fulfilled, the program counter is incremented by one (i.e., next instruction in the MC table is executed). It is important to note that the CondJump instruction blocks until the current operation has completely finished (i.e., carry and zero flag of current operation are valid), which typically takes 8 extra PKC clock cycles. Table 151 describes the usage of the different parameters of the CondJump instruction.

Table 151. Description of the parameters used within the CondJump instruction

Parameter	Description
Cond[2:0]	Defines the jump condition (selection of flag(s) and state(s)) • 0xy: jump if zero flag = x and carry flag = y • 10y: jump if zero flag = y • 11y: jump if carry flag = y
Addr[7:0]	Relative address that is used in case jump condition is fulfilled. The address is in the range of +127 to -128 and is added to the current program counter.

MC instruction: PreLoadT

The PreLoadT instruction allows transferring data to and from the internal T register (T-Ptr) inside the MC layer and consists of two bytes. This includes also an option to load 8 bit constants into the T register. Data can also be manipulated during the transfer (increment/decrement, addition/subtraction) and exchanged with data from another pointer register/parameter. Table 152 describes the usage of the different parameters of the PreLoadT instruction.

Table 152. Description of the parameters used within the PreLoadT instruction

Parameter	Description
Const	Defines if second byte contains configuration options for transferring data between selected pointer/parameter and T register or an 8 bit constant value

Table continues on the next page...

Table 152. Description of the parameters used within the PreLoadT instruction (continued)

Parameter	Description
	• 0: second byte contains configuration options for transferring data • 1: second byte contains 8 bit constant value that is loaded into T register
Const[7:0]	8 bit constant value loaded into T register - padded with 0s (Const = 1)
Exch	Enables exchanging of data between T register and selected pointer/parameter, with optional data manipulation (T2Ptr is not considered). • 0: exchanging of data is disabled • 1: exchanging of data is enabled, i.e., in addition to normal data transfer Ptr = T (T2Ptr = 0) (e.g., T = Ptr - T; Ptr = T (T2Ptr = 0; AddEN = 1; IncrEN = X; Add_Incr = 0)) T = Ptr (T2Ptr = 1) (e.g., Ptr = T - Ptr; T = Ptr (T2Ptr = 1; AddEN = 1; IncrEN = X; Add_Incr = 0) Pointer assignments are done concurrently.
T2Ptr	Defines if data is loaded from selected pointer/parameter to T register or the other way around, with optional data manipulation • 0: data loaded into T register from selected pointer/parameter T = Ptr - T (AddEN = 1; IncrEN = X; Add_Incr = 0) T = Ptr + T (AddEN = 1; IncrEN = X; Add_Incr = 1) T = Ptr - 1 (AddEN = 0; IncrEN = 1; Add_Incr = 0) T = Ptr + 1 (AddEN = 0; IncrEN = 1; Add_Incr = 1) T = Ptr (AddEN = 0; IncrEN = 0; Add_Incr = X) • 1: data stored from T register to selected pointer/parameter Ptr = T - Ptr (AddEN = 1; IncrEN = X; Add_Incr = 0) Ptr = T + Ptr (AddEN = 1; IncrEN = X; Add_Incr = 1) Ptr = T - 1 (AddEN = 0; IncrEN = 1; Add_Incr = 0) Ptr = T + 1 (AddEN = 0; IncrEN = 1; Add_Incr = 1) Ptr = T (AddEN = 0; IncrEN = 0; Add_Incr = X)
AddEN	Enables/disables adding/subtracting T register or selected pointer/parameter • 0: adding/subtracting disabled • 1: adding (Add_Incr = 1) / subtracting (Add_Incr = 0) enabled
IncrEN	Enables/disables incrementing/decrementing T register or selected pointer/parameter. Ignored if AddEN = 1 • 0: incrementing/decrementing disabled • 1: incrementing (Add_Incr = 1) / decrementing (Add_Incr = 0) enabled

Table continues on the next page...

Table 152. Description of the parameters used within the PreLoadT instruction (continued)

Parameter	Description
Add_Incr	Selects added/incremented or subtracted/decremented operation • 0: subtracting/decrementing • 1: adding/incrementing
Ptr_Sel[2:0]	Selects pointer/parameter that should be used for transferring/exchanging data with T register • 000: select pointer X • 001: select pointer Y • 010: select pointer Z • 011: select pointer R • 100: select pointer S • 101: select pointer T • 110: select parameter LEN • 111: select parameter LC

The condition 'Ptr_Sel = T' is a special case as in this case the Exch and T2Ptr parameters are don't care. Table 153 shows how pointers are updated when PreLoadT is operated with 'Ptr_Sel != T' and 'Ptr_Sel = T'.

Table 153. Description of pointer update during PreLoadT instruction for Ptr_Sel != T and Ptr_Sel = T

Exch	T2Ptr	AddEN	IncrEN	Add_Incr	Ptr_Sel != T	Ptr_Sel = T
1	0	1	X	0	T = Ptr - T; Ptr = T	T = 0
1	0	1	X	1	T = Ptr + T; Ptr = T	T = 2T
1	0	0	1	0	T = Ptr - 1; Ptr = T	T = T - 1
1	0	0	1	1	T = Ptr + 1; Ptr = T	T = T + 1
1	0	0	0	X	T = Ptr; Ptr = T	T = T
0	0	1	X	0	T = Ptr - T	T = 0
0	0	1	X	1	T = Ptr + T	T = 2T
0	0	0	1	0	T = Ptr - 1	T = T - 1
0	0	0	1	1	T = Ptr + 1	T = T + 1
0	0	0	0	X	T = Ptr	T = T
0	1	1	X	0	Ptr = T - Ptr	T = 0
0	1	1	X	1	Ptr = T + Ptr	T = 2T
0	1	0	1	0	Ptr = T - 1	T = T - 1
0	1	0	1	1	Ptr = T + 1	T = T + 1
0	1	0	0	X	Ptr = T	T = T
1	1	1	X	0	Ptr = T - Ptr; T = Ptr	T = 0

Table continues on the next page...

Table 153. Description of pointer update during PreLoadT instruction for Ptr_Sel != T and Ptr_Sel = T (continued)

Exch	T2Ptr	AddEN	IncrEN	Add_Incr	Ptr_Sel != T	Ptr_Sel = T
1	1	1	X	1	Ptr = T + Ptr; T = Ptr	T = 2T
1	1	0	1	0	Ptr = T - 1; T = Ptr	T = T -1
1	1	0	1	1	Ptr = T + 1; T = Ptr	T = T + 1
1	1	0	0	X	Ptr = T; T = Ptr	T = T

MC instruction: Jump

This instruction performs an unconditional jump and consists of two bytes. The first byte contains only the op-code, the second byte contains the relative address that is added to the program counter. For example, if the Jump instruction is located at address a and a+1 and the offset value (relative address) is b (in two's complement representation), the final jump address where the next instruction will be fetched from is a + 1 + b. Table 154 describes the usage of the different parameters of the Jump instruction.

Table 154. Description of the parameters used within the Jump instruction

Parameter	Description
Addr[7:0]	Relative address (+127 to -128) that is added to the program counter

MC instruction: Exit

The Exit instruction marks the end of a pattern. The execution of the pattern stops. The Exit instruction has no parameters. When the execution of a pattern is stopped, configuration values (potentially changed during the execution of a pattern) are set back to default (e.g., Conf_Load is set to 0000 and Const_Src to 0).

Any MC instruction not listed above will trigger a security alarm.

16.4.2.3 Fixed L1 MC Patterns

The PKC provides the fixed MC patterns that are listed in Table 155.

Table 155. Fixed L1 MC patterns

Start-Addr	MC size [bytes]	Symbol	Operation	Description
0x00	19	MM	R = X · Y mod Z	Modular Multiplication of a MPI X with a MPI Y modulo a MPI Z (Montgomery Reduction)
0x00	19	MM_GF2	R = X · Y mod Z {GF(2)}	Modular Multiplication of a MPI X with a MPI Y modulo a MPI Z (Montgomery Reduction) over GF(2)
0x13	10	PM	R = X · Y	Plain Multiplication of a MPI X with a MPI Y
0x13	10	PM_GF2	R = X · Y {GF(2)}	Plain Multiplication of a MPI X with a MPI Y over GF(2)
0x1D	4	PMA	R = X · Y + Z	Plain Multiplication with Addition of a MPI X with a MPI Y and a MPI Z
0x1D	4	PMA_GF2	R = X · Y + Z {GF(2)}	Plain Multiplication with Addition of a MPI X with a MPI Y and a MPI Z over GF(2)

Table continues on the next page...

Table 155. Fixed L1 MC patterns (continued)

Start-Addr	MC size [bytes]	Symbol	Operation	Description
0x21	9	MA	R = Y + Z mod X	Modular Addition of a MPI Y with a MPI Z modulo a MPI X
0x2A	9	MS	R = Y - Z mod X	Modular Subtraction of a MPI Y with a MPI Z modulo a MPI X
0x33	32	MR	R = X mod Z	Modular Reduction of a MPI X and a MPI Z
0x33	32	MR_GF2	R = X mod Z {GF(2)}	Modular Reduction of a MPI X and a MPI Z over GF(2)
0x53	10	MMP2	$R = X \cdot Y \mod 2^{LEN(Y)}$	Modular Multiplication of a MPI X and a MPI Y modulo $2^{LEN(Y)}$
0x5A	10	MMAP2	$R = X \cdot Y + Z \mod 2^{LEN(Y)}$	Modular Multiplication with Addition of a MPI X with a MPI Y and a MPI Z modulo $2^{LEN(Y)}$
0x5D	64	MI	$R = Y^{-1} \cdot 2^k \mod X$	Modular Inversion of a MPI Y module a MPI X
0x5D	64	MI_GF2	$R = Y^{-1} \cdot 2^k \mod X$ {GF(2)}	Modular Inversion of a MPI Y module a MPI X over GF(2)
0x9D	10	PM_PATCH	$R = X \cdot Y$	Plain Multiplication of a MPI X with a MPI Y (patched version) NOTE: same functionality as PM, but using patched code that supports corner case: MPI X > 1 PKC word and MPI Y = 1 PKC word (slightly longer execution time)
0x9D	10	PM_PATCH_GF2	$R = X \cdot Y$ {GF(2)}	Plain Multiplication of a MPI X with a MPI Y over GF(2) (patched version) NOTE: same functionality as PM_GF2, but using patched code that supports corner case: MPI X > 1 PKC word and MPI Y = 1 PKC word (slightly longer execution time)
0xA7	28	GCD	R = GCD(Y, Z)	Greatest common divider (GCD) of MPI Y and MPI Z

Detailed description of the fixed L1 calculation modes

Table 156. Modular Multiplication

Modular multiplication of a MPI X with a MPI Y modulo a MPI Z. MPI Y and MPI Z must have equal length. Montgomery algorithm is used for reduction.		
$R = X \cdot Y \mod Z$	Start-Addr = 0x00	Symbol: MM / MM_GF2
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0

Table continues on the next page...

Table 156. Modular Multiplication (continued)

Modular multiplication of a MPI X with a MPI Y modulo a MPI Z. MPI Y and MPI Z must have equal length. Montgomery algorithm is used for reduction.		
R = X · Y mod Z	Start-Addr = 0x00	Symbol: MM / MM_GF2
		Multiplicative Inverse ~Z has to be stored in the PKC word directly before Z_0.
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y and Z in bytes
	MCLEN	Length of operand X in bytes
Operands[1]	XLEN = MCLEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
	$\sim Z = (-Z)^{-1} \bmod 2^{wordsize}$; wordsize = 64 for REDMUL == 0x0 or 0x2	
Result[2]	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result space	RLEN+1	
Result-in-place	Only possible if MCLEN == 1 PKC word for RPTR == XPTR and RPTR == YPTR. Not possible for RTPR == ZPTR.	
Updated status flags	CARRY=0, ZERO	
GF2 conversion	Modular Multiplication in GF(2) / MM_GF2	
Execution time	4·XLEN·(2·YLEN+3)+8 PKC clock cycle; In case of optional final reduction the cycle count is increased by (YLEN+1), not applicable for MM_GF2	

MC code	`ModMul :  ConfLoad(0,XYZR)`
	`  Execute(0x00)`
	`ModMul_Loop :  InDecPtr(0,Z)`
	`  ConfLoad(0,ZRZR)`
	`  Execute(0x20)`
	`  InDecPtr(1,Z)`
	`  ConfLoad(0,XZRR)`
	`  Execute(0x62)`
	`  DecrTBNZ(ModMul_Cont)`	

Table continues on the next page...

Table 156. Modular Multiplication (continued)

Modular multiplication of a MPI X with a MPI Y modulo a MPI Z. MPI Y and MPI Z must have equal length. Montgomery algorithm is used for reduction.		
R = X · Y mod Z	Start-Addr = 0x00	Symbol: MM / MM_GF2

	`ModMul_CondSub : ConfLoad(0,ZRZR)`
	`CondJump(C=0)`
	`(ModMul_Exit)`
	`Execute(0x2B)`
	`ModMul_Exit : Exit()`
	`ModMul_Cont : InDecPtr(1,X)`
	`ConfLoad(0,XYRR)`
	`Execute(0x22)`
	`Jump()`
	`(ModMul_Loop)`	

1. In case XLEN>1 and YLEN=1 the least-significant PKC word of the result R is read before it is written the very first time. This can lead to reading uninitialized memory. If this is an issue the least-significant PKC word of the result R needs to be initialized upfront before calling the operation.
2. The result R might be bigger than the modulus (R ≥ Z). The reduction only guarantees that the result fits to the wordsize of the modulus. → $R < 2^{YLEN \cdot wordsize}$

   The length of the result is equal RLEN(=YLEN) but for the calculation one extra PKC word on top of the result R is required. The width of the word depends on PKC_CTRL.REDMUL.

NOTE

There may happen a final reduction step at the end of the calculation that can be used for timing attacks. Refer to the PKC application note how to prevent side channel attacks based on this effect.

In case the operands X and Y have been transformed to Montgomery Space, also the result R is in Montgomery Space (represented by []):

$[X] \cdot [Y] \bmod Z = X \cdot Q \cdot Y \cdot Q \cdot Q^{-1} \bmod Z = R \cdot Q \bmod Z = [R]$; with $Q = 2^{MCLEN \cdot wordsize}$

This is only valid for MCLEN==LEN. If MCLEN ≠ LEN special care has to be taken. Refer to the PKC application note in this case.

Table 157. Plain Multiplication

Plain multiplication of a MPI X with a MPI Y		
R = X · Y	Start-Addr = 0x13	Symbol: PM / PM_GF2
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
	MCLEN	Length of operand X in bytes
Operands[1]	XLEN = MCLEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	

Table continues on the next page...

Table 157. Plain Multiplication (continued)

Plain multiplication of a MPI X with a MPI Y		
R = X · Y	Start-Addr = 0x13	Symbol: PM / PM_GF2
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
Result	RLEN = XLEN+YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result space	RLEN	
Result-in-place	Only possible if MCLEN == 1 PKC word for RPTR == XPTR and RPTR == YPTR.	
Updated status flags[2]	CARRY=0, ZERO	
GF2 conversion	Plain Multiplication in GF(2) / PM_GF2	
Execution time	4·XLEN·(YLEN+1) PKC clock cycle;	

MC code	`PlainMul :  ConfLoad (0,XYZR)`
	`  Execute(0x00)`
	`PlainMul_Loop :  DecrTBNZ(PlainMul_Cont )`
	`PlainMul_Exit :  Exit()`
	`PlainMul_Cont :  InDecPtr(1,X)`
	`  InDecPtr (1,R)`
	`  ConfLoad (0,XYRR)`
	`PlainMac_Entry :  Execute(0x22 )`
	`  Jump()`
	`  (PlainMul_Loop)`	

1. In case XLEN>1 also YLEN needs to be >1 (XLEN and YLEN can still have different values). This can be achieved, e.g., by extending Y with another PKC word set to zero ($Y_1 = 0$) resulting in YLEN=2.
2. ZERO=1 indicates only a zero result in the last L0 calculation, resp. the last MACC (Execute(0x22)) for the most significant X word. In case XLEN>1 it can happen that ZERO is set even though the complete result R is unequal zero. To verify if for XLEN>1 the complete result R is zero a succeeding CMP calculation (op-code 0x4B) could be used.

Table 158. Plain Multiplication with Addition

Plain multiplication of a MPI X with a MPI Y and addition of a MPI Z. MPI Y and MPI Z must have equal length.		
R = X · Y + Z	Start-Addr = 0x1D	Symbol: PMA / PMA_GF2
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0

Table continues on the next page...

Table 158. Plain Multiplication with Addition (continued)

Plain multiplication of a MPI X with a MPI Y and addition of a MPI Z. MPI Y and MPI Z must have equal length.		
R = X · Y + Z	Start-Addr = 0x1D	Symbol: PMA / PMA_GF2
	LEN	Length of operand Y and Z in bytes
	MCLEN	Length of operand X in bytes
Operands[1]	XLEN = MCLEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = XLEN+YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result space	RLEN	
Result-in-place	Only possible if MCLEN == 1 PKC word for RPTR == XPTR and RPTR == YPTR. Always possible for RPTR == ZPTR.	
Updated status flags[2]	CARRY=0, ZERO	
GF2 conversion	Plain Multiplication with Addition in GF(2) / PMA_GF2	
Execution time	4·XLEN·(YLEN+1) PKC clock cycle;	

MC code	`PlainMac :  ConfLoad(0,XYZR)`
	`Execute(0x2D) // to clear carry bit`
	`Jump()`
	`(PlainMac_Entry)`	

1. In case XLEN>1 also YLEN needs to be >1 (XLEN and YLEN can still have different values). This can be achieved, e.g., by extending Y with another PKC word set to zero ($Y_1 = 0$) resulting in YLEN=2.
2. ZERO=1 indicates only a zero result in the last L0 calculation, resp. the last MACC (Execute(0x22)) for the most significant X word. In case XLEN>1 it can happen that ZERO is set even though the complete result R is unequal zero. To verify if for XLEN>1 the complete result R is zero a succeeding CMP calculation (op-code 0x4B) could be used.

Table 159. Modular Addition

Modular Addition of a MPI Y with a MPI Z modulo a MPI X. MPI X, MPI Y and MPI Z must have equal length.		
R = Y + Z mod X	Start-Addr = 0x21	Symbol: MA
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand X, Y and Z in bytes
	MCLEN	don't care
Operands[1]	XLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	YLEN = XLEN $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = XLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result[2]	RLEN = XLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result space	RLEN	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR. Not possible for RPTR == XPTR.	
Updated status flags	CARRY=0, ZERO	
GF2 conversion	not supported --> behaves like L0 calculation with op-code 0x0f - logical xor / XOR	
Execution time	4·(YLEN+3 + n·(YLEN+3)) PKC clock cycle; with n equals amount of reduction loops (n=0 in case Y+Z < $2^{YLEN \cdot wordsize}$)	

MC code	`ModAdd :  ConfLoad(0,XYZR)`
	`Execute(0x0A)`
	`ConfLoad(0,XRXR)`
	`CondJump(C=0)`
	`(ModAdd_Exit)`
	`ModAdd_Sub :  Execute(0x2B)`
	`CondJump(C=1)`	

Table continues on the next page...

Table 159. Modular Addition (continued)

Modular Addition of a MPI Y with a MPI Z modulo a MPI X. MPI X, MPI Y and MPI Z must have equal length.		
R = Y + Z mod X	Start-Addr = 0x21	Symbol: MA
	(ModAdd_Sub) ModAdd_Exit : Exit()	

1. Carry overflow during addition is corrected by subtracting the modulus X from the result. If X is small a lot of subtraction loops might be required. X should be prepared accordingly, e.g. shifted so that the msb of X is '1'.
2. The result R might be bigger than the modulus (R ≥ X). The reduction only guarantees that the result fits to the word-size of the modulus. → $R < 2^{YLEN \cdot wordsize}$

Table 160. Modular Subtraction

Modular Subtraction of a MPI Y by a MPI Z modulo a MPI X. MPI X, MPI Y and MPI Z must have equal length.		
R = Y - Z mod X	Start-Addr = 0x2A	Symbol: MS
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand X, Y and Z in bytes
	MCLEN	don't care
Operands[1]	XLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	YLEN = XLEN $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = XLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result[2]	RLEN = XLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result space	RLEN	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR. Not possible for RPTR == XPTR.	
Updated status flags	CARRY=0, ZERO	

Table continues on the next page...

Table 160. Modular Subtraction (continued)

Modular Subtraction of a MPI Y by a MPI Z modulo a MPI X. MPI X, MPI Y and MPI Z must have equal length.		
R = Y - Z mod X	Start-Addr = 0x2A	Symbol: MS
GF2 conversion	not supported --> behaves like L0 calculation with op-code 0x0f - logical xor / XOR	
Execution time	4·(YLEN+3 + n·(YLEN+3)) PKC clock cycle; with n equals amount of reduction loops (n=0 in case Y-Z ≥ 0)	

MC Code	`ModSub : ConfLoad(0,XYZR)`
	`Execute(0x0B)`
	`ConfLoad(0,XRXR)`
	`CondJump(C=0)`
	`(ModSub_Exit)`
	`ModSub_Add : Execute(0x2A)`
	`CondJump(C=1)`
	`(ModSub_Add)`
	`ModSub_Exit : Exit()`	

1. Carry overflow in subtraction is reduced by adding the modulus X to the result. If X is small a lot of addition loops might be required. X should be prepared accordingly, e.g. shifted so that the msb of X is '1'.
2. In case the operand Y Is bigger than X and Z the result R might be bigger than the modulus (R ≥ X). If Y is smaller than X or Z the result is smaller than the modulus X.

Table 161. Modular Reduction

Modular Reduction of a MPI X and a MPI Z. Montgomery algorithm is used for reduction.		
R = X mod Z	Start-Addr = 0x33	Symbol: MR / MR_GF2
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0 Multiplicative Inverse ~Z has to be stored in the PKC word directly before Z_0.
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Z in bytes
	MCLEN	Length of operand X in bytes
Operands[1]	XLEN = MCLEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	ZLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
	$\sim Z = (-Z)^{-1} \bmod 2^{wordsize}$; wordsize = 64 for REDMUL == 0x0 or 0x2	
Result[2]	RLEN = ZLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	

Table continues on the next page...

Table 161. Modular Reduction (continued)

Modular Reduction of a MPI X and a MPI Z. Montgomery algorithm is used for reduction.		
R = X mod Z	Start-Addr = 0x33	Symbol: MR / MR_GF2
Result space	RLEN+1	
Result-in-place	Not possible for RPTR == XPTR and RTPR == ZPTR.	
Updated status flags	CARRY=0, ZERO	
GF2 conversion	Modular Reduction in GF(2) / MR_GF2	
Execution time	4·(XLEN·(2·ZLEN+6) + MAX(ZLEN+1, 4)) PKC clock cycle;	
MC Code		

```
ModRed :          ConfLoad(1,XYZR)
                  Execute(0x3E)
                  PreLoadT(0)
                  (T = R)
                  PreLoadT(0)
                  (T = T + LEN)
                  PreLoadT(0)
                  (Y = T)
                  PreLoadT(1)
                  (T = 1)
ModRed_Loop :     PreLoadT(0)
                  (T = T + LEN; LEN = T)
                  PreLoadT(0)
                  (T <--> S)
                  ConfLoad(1,XYXY)
                  Execute(0x3E)
                  PreLoadT(0)
                  (T <--> S)
                  PreLoadT(0)
                  (T <--> LEN)
                  ConfLoad(0,XRXR)
                  Execute(0x6A)
                  InDecPtr(0,LEN)
                  InDecPtr(0,Z)
                  ConfLoad(0,ZRZR)
                  Execute(0x20)
                  InDecPtr(1,Z)
                  ConfLoad(0,XZRR)
                  Execute(0x62)
                  InDecPtr(1,X)
                  DecrTBNZ(ModRed_Loop)
                  Exit()
```

1. In case XLEN>1 and YLEN=1 the least-significant PKC word of the result R is read before it is written the very first time. This can lead to reading uninitialized memory. If this is an issue the least-significant PKC word of the result R needs to be initialized upfront before calling the operation.
2. The reduction ensures that the result is smaller or equal to the modulus (i.e., R ≤ Z).

   The length of the result is equal RLEN(=ZLEN) but for the calculation one extra PKC word on top of the result R is required. The width of the word depends on PKC_CTRL.REDMUL.

NOTE

MontRed(X) = MM(X,1) = X · Q^{-1} mod Z; with $Q=2^{XLEN \cdot wordsize}$

To obtain the result in normal representation (without the Q^{-1} term), a Modular Montgomery multiplication with Q^2 is required

→ X mod Z = MM(MontRed(X), Q^2 mod Z)

Because Q^2 is typically only known for the $Q=2^{YLEN \cdot wordsize}$ the length of the X operand should be equal ZLEN or be adapted accordingly (refer to PKC application note).

Table 162. Modular Multiplication Modulo Power of 2

Modular multiplication of a MPI X with a MPI Y modulo 2^{YLEN}.		
$R = X \cdot Y \bmod 2^{YLEN}$	Start-Addr = 0x53	Symbol: MMP2
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
	MCLEN	Length of operand X in bytes
Operands[1]	XLEN = MCLEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
Result[2]	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result space	RLEN+1	
Result-in-place	Only possible if MCLEN == 1 PKC word for RPTR == XPTR and RPTR == YPTR.	
Updated status flags[3]	CARRY=0, ZERO	
GF2 conversion	not supported	
Execution time	4·XLEN·(2·YLEN-XLEN+3)/2 PKC clock cycle;	

MC code	`ModMulP2 :  Execute(0x00)`
	`ModMulP2_Loop :  DecrTBNZ(ModMulP2_Cont)`
	`Exit()`
	`ModMulP2_Cont :  InDecPtr(1,X)`
	`InDecPtr(1,R)`
	`InDecPtr(0,LEN)`
	`ConfLoad(0,XYRR)`	

Table continues on the next page...

Table 162. Modular Multiplication Modulo Power of 2 (continued)

Modular multiplication of a MPI X with a MPI Y modulo 2^{YLEN}.		
$R = X \cdot Y \bmod 2^{YLEN}$	Start-Addr = 0x53	Symbol: MMP2

	`ModMulAddP2_Entry : Execute(0x22)`
	`Jump()`
	`(ModMulP2_Loop)`	

1. This micro code program only works for YLEN ≥ XLEN.
2. The length of the result is equal RLEN(=YLEN) but for the calculation one extra PKC word on top of the result R is required. The width of the word depends on PKC_CTRL.REDMUL.
3. ZERO=1 indicates only a zero result in the last L0 calculation, resp. the last MACC(Execute(0x22)) for the most significant X word. In case XLEN>1 it can happen that ZERO is set even though the complete result R is unequal zero. To verify if for XLEN>1 the complete result is zero a succeeding CMP calculation (op-code 0x4B) could be used.

Table 163. Modular Multiplication with Addition Modulo Power of 2

Modular multiplication with addition of a MPI X with a MPI Y and a MPI Z modulo 2^{YLEN}. MPI Y and MPI Z must have equal length.		
$R = X \cdot Y + Z \bmod 2^{YLEN}$	Start-Addr = 0x5A	Symbol: MMAP2
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y and Z in bytes (≥ MCLEN)
	MCLEN	Length of operand X in bytes
Operands[1]	XLEN = MCLEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result[2]	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result space	RLEN+1	
Result-in-place	Only possible if MCLEN == 1 PKC word for RPTR == XPTR and RPTR == YPTR. Always possible for RPTR == ZPTR.	

Table continues on the next page...

Table 163. Modular Multiplication with Addition Modulo Power of 2 (continued)

Modular multiplication with addition of a MPI X with a MPI Y and a MPI Z modulo 2^{YLEN}. MPI Y and MPI Z must have equal length.		
$R = X \cdot Y + Z \bmod 2^{YLEN}$	Start-Addr = 0x5A	Symbol: MMAP2
Updated status flags[3]	CARRY=0, ZERO	
GF2 conversion	not supported	
Execution time	4·XLEN·(2·YLEN-XLEN+3)/2 PKC clock cycle;	

MC code	`ModMulP2 : Execute(0x00)`
	`ModMulP2_Loop : DecrTBNZ(ModMulP2_Cont)`
	`Exit()`
	`ModMulP2_Cont : InDecPtr(1,X)`
	`InDecPtr(1,R)`
	`InDecPtr(0,LEN)`
	`ConfLoad(0,XYRR)`
	`ModMulAddP2_Entry : Execute(0x22)`
	`Jump()`
	`(ModMulP2_Loop)`	

1. This micro code program only works for YLEN ≥ XLEN.
2. The length of the result is equal RLEN(=YLEN) but for the calculation one extra PKC word on top of the result R is required. The width of the word depends on PKC_CTRL.REDMUL.
3. ZERO=1 indicates only a zero result in the last L0 calculation, resp. the last MACC(Execute(0x22)) for the most significant X word. In case XLEN>1 it can happen that ZERO is set even though the complete result R is unequal zero. To verify if for XLEN>1 the complete result is zero a succeeding CMP calculation (op-code 0x4B) could be used.

Table 164. Modular Inversion

Modular Inversion of a MPI Y modulo an MPI X. Almost Montgomery inversion algorithm used.		
$R = Y^{-1} \cdot 2^k \bmod X$	Start-Addr = 0x5D	Symbol: MI / MI_GF2
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0 MPI X gets overwritten
	YPTR	Pointer to MPI operand Y: Byte address of Y_0 MPI Y gets overwritten
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0 MPI Z needs to be initialized to 1
	RPTR	Pointer to result area R: Byte address of R_0 k is stored in the PKC word directly before R_0 {MPI R, k} needs to be initialized to 0

Table continues on the next page...

Table 164. Modular Inversion (continued)

Modular Inversion of a MPI Y modulo an MPI X. Almost Montgomery inversion algorithm used.		
$R = Y^{-1} \cdot 2^k \bmod X$	Start-Addr = 0x5D	Symbol: MI / MI_GF2
	LEN	Length of operand X, Y and Z in bytes
	MCLEN	32 << 3 (REDMUL == 0x0 or 0x2) bytes
Operands	XLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	YLEN = XLEN $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = XLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = XLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
	kLEN = 1 PKC word $k = k_0$	
Result space	RLEN+kLEN	
Result-in-place	No	
Updated status flags	CARRY=0, ZERO	
GF2 conversion	Modular Inversion in GF(2) / MI_GF2	
Execution time	data dependent	

NOTE

1. The following input relation has to be fulfilled: MPI Z = 1, {MPI R, k} = 0, MCLEN = 31 PKC words (the actual value depends on PKC_CTRL.REDMUL).
2. Output relation of this operation: MPI X and MPI Y get overwritten.

Table 165. Plain Multiplication (Patched version)

Plain multiplication of a MPI X with a MPI Y		
$R = X \cdot Y$	Start-Addr = 0x9D	Symbol: PM_PATCH / PM_PATCH_GF2
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0

Table continues on the next page...

Table 165. Plain Multiplication (Patched version) (continued)

Plain multiplication of a MPI X with a MPI Y		
R = X · Y	Start-Addr = 0x9D	Symbol: PM_PATCH / PM_PATCH_GF2
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
	MCLEN	Length of operand X in bytes
Operands	XLEN = MCLEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
Result	RLEN = XLEN+YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result space	RLEN	
Result-in-place	Only possible if MCLEN == 1 PKC word for RPTR == XPTR and RPTR == YPTR.	
Updated status flags[1]	CARRY=0, ZERO	
GF2 conversion	Plain Multiplication in GF(2) / PM_PATCH_GF2	
Execution time	4·XLEN·(YLEN+2) PKC clock cycle;	

MC code	`PlainMulPatch :  Execute(0x00)`
	`PlainMulPatch_Loop :  DecrTBNZ(PlainMulPatch_Cont)`
	`PlainMulPatch_Exit :  Exit()`
	`PlainMulPatch_Cont :  InDecPtr(1,X)`
	`InDecPtr (1,R)`
	`ConfLoad (0,XYRR)`
	`PlainMacPatch :  Execute(0x20)`
	`Execute(0x22)`
	`Jump()`
	`(PlainMulPatch_Loop)`	

1. ZERO=1 indicates only a zero result in the last L0 calculation, resp. the last MACC (Execute(0x22)) for the most significant X word. In case XLEN>1 it can happen that ZERO is set even though the complete result R is unequal zero. To verify if for XLEN>1 the complete result R is zero a succeeding CMP calculation (op-code 0x4B) could be used.

Table 166. Greatest Common Divider

Greatest Common Divider (GCD) of MPI Y and MPI Z		
R = GCD(Y, Z)	Start-Addr = 0xA7	Symbol: GCD
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0 (needs to point to same address as YPTR)
	YPTR	Pointer to MPI operand Y: Byte address of Y_0 MPI Y gets overwritten
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0 (needs to point to same address as RPTR) MPI Z gets overwritten
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand X, Y and Z in bytes
	MCLEN	don't care
Operands	XLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	YLEN = XLEN $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = XLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = XLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result space	RLEN	
Result-in-place	Yes (since RPTR == ZPTR).	
Updated status flags	CARRY, ZERO=1	
GF2 conversion	not supported	
Execution time	data dependent	

NOTE

1. The following input relation has to be fulfilled: XPTR = YPTR and ZPTR = RPTR.
2. Output relation of this operation: MPI X = MPI Y = MPI Z = MPI R (i.e., MPI Y and MPI Z get overwritten).

16.4.3 Security

This section provides a brief summary of the security related aspects of PKC, general considerations, error conditions triggered by PKC and software-configurable countermeasures.

Error Conditions

The following types of error conditions trigger a reset of the PKC-CTRL and the PKC coprocessor kernel. The error that triggered this reset is latched in the PKC_ACCESS_ERR register and is only cleared by setting PKC_ACCESS_ERR_CLR[ERR_CLR] or via a global reset (not (PKC_ACCESS_ERR_CLR[PKC_SOFT_RST[SOFT_RST]). The single error bits are accumulated. If a new error occurs while the previous one is still indicated the new error flag is set on top. This holds for all bits in PKC_ACCESS_ERR.

Bit definition for block specific PKC_ACCESS_ERR fields:

- PKCC - Error in PKC coprocessor kernel (e.g. state machine or illegal calculation mode (could be triggered by SW)) or L0 bus errors
- FDET - Error in PKC control state machine
- CTRL - Error in GO handling (could also be triggered by SW)
- UCRC - Error in UP CRC check (could also be triggered by SW)
- BLK_ERR[7:4] - reserved

Bit definition for AHB specific PKC_ACCESS_ERR:

- AHB - AHB master error (AHB_AMBA_ERROR) - L2

Bit definition for APB specific PKC_ACCESS_ERR fields:

- APB_WRGMD - wrong access rights or wrong read-write information: Access not allowed in the current mode or write access to a read-only register or vice versa. APB interface returns zero on data read port for invalid read access.

NOTE

Reading from write-only SFRs will not trigger an error, instead zero on data read port is returned.

- APB_NOTAV - unavailable address: There is no SFR or memory located at the accessed address. APB interface returns zero on data read port for invalid read access.

The APB master that triggered the first APB_WRGMD/APB_NOTAV error is stored in the field PKC_ACCESS_ERR.APB_MASTER.

NOTE

The PKC_ACCESS_ERR[APB_MASTER] bit field always reads 0.

The PKC_ACCESS_ERR[APB_WRGMD]/[NOTAV]/[MASTER] fields are not updated in case a second APB error triggers but only for the first occurrences until cleared via PKC_ACCESS_ERR_CLR or a global reset.

All other bit fields of the PKC_ACCESS_ERR SFR are reserved, and always read zero.

The following events will trigger a security alert that resets the PKC including the PKC kernel and asserts the pkc_err signal. All error outputs listed as output in the following table stay high until cleared by reset (only POR and chip/main reset) or PKC_ACCESS_ERR_CLR[ERR_CLR].

Table 167. List of PKC security alerts

Security alert	PKC_ACCESS_ERR flag
SFR access	
SFR access to not implemented SFR	APB_NOTAV
SFR write access to read_only SFR	APB_WRGMD
DMA access	

Table continues on the next page...

Table 167. List of PKC security alerts (continued)

Security alert	PKC_ACCESS_ERR flag
System feedback error as response to an AHB access. L2 only	AHB
PKC kernel errors	
Executing PKC kernel operation with illegal calculation mode	PKCC
Executing PKC kernel operation with illegal operand length LEN (zero or too short for selected multiplier size)	PKCC
Executing DecrTBNZ on zero loop counter value (e.g. input via MCLEN)	PKCC
REDMUL setting selects multiplier size that is bigger than native multiplier size, e.g. REDMUL=11b for 64-bit native multiplier.	PKCC
Error detection	PKCC
Executing CMP calculation with YPTR equal ZPTR.	PKCC
Executing Reserved MC instruction in L1 calculation.	PKCC
PKC RAM parity error.	PKCC
PKC-CTRL errors	
Error detection	FDET
Set PKC_CTRL[GOx] while GOANY is set.	CTRL
Write access to parameter set 1 (PKC_MODE1, PKC_XYPTR1, PKC_ZRPTR1, PKC_LEN1) while PKC_STATUS.LOCKED[0] is set.	CTRL
Write access to parameter set 2 (PKC_MODE2, PKC_XYPTR2, PKC_ZRPTR2, PKC_LEN2) while bit 1 of PKC_STATUS[LOCKED is set.	CTRL
More than one GO bit is set for CTRL write.	CTRL
PKC_CTRL[GOx] is set while PKC_CTRL[RESET] is set. (PKC_CTRL[RESET] is required to be cleared before setting GOx. RESET cannot be cleared in the same write that sets a GOx bit).	CTRL
Write access to PKC_CFG while PKC_CTRL[RESET] is cleared.	CTRL
Write to PKC_LEN1 or PKC_LEN2 during L2 calculation while PKC_CTRL[GOANY] is set.	CTRL
Set PKC_CTRL[GOU] while PKC_ULEN[LEN] is equal zero.	CTRL
Error in CRC integrity check of L2 calculation (universal pointer fetch).	UCRC

Table continues on the next page...

Table 167. List of PKC security alerts (continued)

Security alert	PKC_ACCESS_ERR flag
PKC Stop Error, attempt to change the PKC_CTRL[STOP] bit before it is stable (see Handling of PKC_CTRL[RESET] and PKC_CTRL[STOP]).	CTRL
PKC Reset Error, attempt to change the PKC_CTRL.RESET bit before it is stable (see Handling of PKC_CTRL[RESET] and PKC_CTRL[STOP]).	CTRL

16.4.4 Interrupt, exception and DMA

Interrupts

The PKC Control provides a level triggered interrupt to indicate the end of a calculation. The high active interrupt request signal `pkc_int` is raised in case the interrupt is enabled (PKC_INT_ENABLE[EN_PDONE] is set due to previous PKC_INT_SET_ENABLE[EN_PDONE] SFR write) and one of the following conditions is triggered:

- end of a native calculation (L0)
- end of microcode calculation (L1)
- end of universal pointer calculation (L2)
- Interrupt triggered by SW via SFR PKC_INT_SET_STATUS[INT_PDONE], e.g. for test purposes

The interrupt request signal stays high unless cleared by reset (all resets except the PKC kernel reset (PKC_CTRL[RESET])) or via SW using SFR PKC_INT_CLR_STATUS[INT_PDONE].

PKC_INT_STATUS[INT_PDONE] will be set by HW or SW independently from PKC_INT_ENABLE[EN_PDONE]. PKC_INT_ENABLE[EN_PDONE] shall only gate the interrupt request signal that is connected to the interrupt controller.

```
pkc_int = PKC_INT_STATUS.INT_PDONE && PKC_INT_ENABLE.EN_DONE
```

Exceptions

The PKC_CTRL notifies the system block on any security alarm, resp. misbehavior of the PKC. The system decides how to react, e.g. trigger global reset, raise a CPU exception or ignore. The possible security alarms are listed in table Table 167

DMA / Memory Interfaces

The PKC Control provides an AHB (L2) master port for the DMA access of the Universal Pointer Fetch (read-only). Typically two windows for two different memories (e.g. RAM and NV) are accessed during a PKC universal pointer fetch calculation, one for the program code and one for the calculation parameters.

The PKC kernel requires a direct and fast DMA access to the accessible RAM (read-write) via RAM-IF (L0).

16.4.5 Clocking

The IP is a fully synchronous design.

The PKC Control supports an APB clock (port name `clk_apb`) and a functional clock for the kernel (port name `clk_pkc`).

16.4.6 Reset

There is a single main hard reset connected from the system to PKC-CTRL

- POR/System reset (rst_main_sys_n)

POR reset is a Power-On-Reset and asserted once by the system during the start up. System reset is asserted when the system decides to reset the PKC-CTRL and PKC (usually due to a security violation). This resets PKC-CTRL and PKC completely.

There are two soft resets defined in the SFR Interface:

- PKC_SOFT_RST[SOFT_RST], which resets the PKC-CTRL including all sub-blocks (except SFR PKC_ACCESS_ERR),
- PKC_CTRL[RESET], which resets only the PKC kernel (incl. the internal carry (c) and result register Reg_i that are e.g. used as input for opcode 0x62 (R=(Reg_i * Y + {c,Z}) >> 64)).

In case of a security alarm the PKC_CTRL including all sub-blocks are reset as well. The behavior is the same as for the SOFT_RST. The error indicating SFR PKC_ACCESS_ERR is not reset. The security reset releases also the error source such that the PKC_CTRL is again functional after the reset was triggered.

Table 168. Resets and impact for PKC-CTRL and kernel

Reset type	Reset Signal	Impact on PKC_CTRL	Impact on PKC kernel
Global system reset	rst_main_sys_n	Full block is reset.	Kernel is reset (Note 1), due to reset of PKC-CTRL.RST, clk_pkc is stopped
Soft reset SW (PKC_SOFT_RST)	N/A	Reset full block except PKC_ACCESS_ERR SFR.	Kernel is reset, due to reset of PKC-CTRL.RST, clk_pkc is stopped
Security alarm reset	pkc_err	Reset full block except PKC_ACCESS_ERR SFR.	Kernel is reset, due to reset of PKC-CTRL.RST, clk_pkc is stopped
Kernel reset (PKC_CTRL[RESET])	N/A	Running PKC calculation is interrupted. Read-only PKC_STATUS SFR is reset.	Kernel is reset, clk_pkc is stopped

Note 1: Uninitialized registers are discharged to their intrinsic value in case power off was long enough, influenced by environment (PVT conditions).

Note 2: For safely interrupting any ongoing operation (i.e., ensuring all pending memory requests have been completed) and completely resetting the IP afterwards the following approach needs to be used:

1. Ensure PKC kernel clock is enabled by setting PKC_CTRL[STOP]=0.
2. Poll value of PKC_CTRL[STOP] until it has taken over internally the value (i.e., reads as 0). This ensures kernel clock has been properly enabled by the system (important if PKC-CTRL and PKC kernel block run at different clock speeds).
3. Interrupt ongoing operation via PKC_CTRL[RESET]=1.
4. Poll value of PKC_CTRL[RESET] until it has taken over internally the value (i.e., reads as 1). This ensures all pending memory requests have been completed and PKC kernel has entered reset state (important if PKC-CTRL and PKC kernel block run at different clock speeds).
5. Perform soft reset via PKC_SOFT_RST[SOFT_RST]=1
6. Clear any pending access errors via PKC_ACCESS_ERR_CLR[ERR_CLR]=1

16.5 Handling of PKC_CTRL[RESET] and PKC_CTRL[STOP]

The PKC internal reset and STOP features are controlled via the PKC_CTRL[RESET] and PKC_CTRL[STOP] bits respectively.

Internally the PKC_CTRL[RESET] is implemented as a synchronous reset of the PKC kernel, therefore careful handling of this control bit and the PKC_CTRL[STOP] is required.

To ensure that PKC_CTRL[RESET] and PKC_CTRL[STOP] bit writes have been appropriately propagated to the PKC kernel and ongoing memory accesses have successfully completed, the software should poll the modified bit until the changed value is read back from the PKC_CTRL register.

This requirement is due to the PKC kernel being fed internally from a different clock source than the register interface.

It is important to note that enabling of the STOP feature while changing the PKC Reset is not supported, the STOP feature should first be disabled before changing the PKC_CTRL[RESET] state.

Table Table 169 lists Invalid writes to the PKC_CTRL register modifying the RESET and STOP bit together which should be avoided.

Table 169. Invalid PKC_CTRL writes

Current State	Updated State
PKC_CTRL[RESET] = 1 PKC_CTR[STOP] = 1	PKC_CTRL[RESET] = 0 PKC_CTR[STOP] = 1
PKC_CTRL[RESET] = 0 PKC_CTR[STOP] = 1	PKC_CTRL[RESET] = 1 PKC_CTR[STOP] = 1
PKC_CTRL[RESET] = 1 PKC_CTR[STOP] = 0	PKC_CTRL[RESET] = 0 PKC_CTR[STOP] = 1

The PKC also ensures ongoing memory accesses at its L2 AHB and L0 RAM interfaces are completed before accepting a kernel reset (PKC_CTRL[RESET]=1) or clock-stop (PKC_CTRL[STOP]=1) request:

- Setting PKC_CTRL[RESET] flag: PKC will stop sending new read/write requests via its L2 AHB and L0 RAM interfaces and wait until any ongoing request has completed (i.e, wait until last AHB request has been acknowledged by AHB L2 slave via corresponding hready_l2=1).
- Setting PKC_CTRL[STOP] flag: PKC will stop sending new read/write request via its L0 RAM interface and wait until any ongoing request has completed.

16.6 PKC register descriptions

16.6.1 PKC memory map

PKC0.PKC base address: 400E_A000h

Offset	Register	Width (In bits)	Access	Reset value
0h	Status Register (PKC_STATUS)	32	R	0000_0000h
4h	Control Register (PKC_CTRL)	32	RW	0000_0001h
8h	Configuration register (PKC_CFG)	32	RW	0000_0719h
10h	Mode register, parameter set 1 (PKC_MODE1)	32	RW	0000_0000h
14h	X+Y pointer register, parameter set 1 (PKC_XYPTR1)	32	RW	0000_0000h
18h	Z+R pointer register, parameter set 1 (PKC_ZRPTR1)	32	RW	0000_0000h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
1Ch	Length register, parameter set 1 (PKC_LEN1)	32	RW	0000_0000h
20h	Mode register, parameter set 2 (PKC_MODE2)	32	RW	0000_0000h
24h	X+Y pointer register, parameter set 2 (PKC_XYPTR2)	32	RW	0000_0000h
28h	Z+R pointer register, parameter set 2 (PKC_ZRPTR2)	32	RW	0000_0000h
2Ch	Length register, parameter set 2 (PKC_LEN2)	32	RW	0000_0000h
40h	Universal pointer FUP program (PKC_UPTR)	32	RW	0000_0000h
44h	Universal pointer FUP table (PKC_UPTRT)	32	RW	0000_0000h
48h	Universal pointer length (PKC_ULEN)	32	RW	0000_0000h
50h	MC pattern data interface (PKC_MCDATA)	32	RW	0000_0000h
60h	PKC version register (PKC_VERSION)	32	R	0000_00BEh
FB0h	Software reset (PKC_SOFT_RST)	32	W	0000_0000h
FC0h	Access Error (PKC_ACCESS_ERR)	32	R	0000_0000h
FC4h	Clear Access Error (PKC_ACCESS_ERR_CLR)	32	W	0000_0000h
FD8h	Interrupt enable clear (PKC_INT_CLR_ENABLE)	32	W	0000_0000h
FDCh	Interrupt enable set (PKC_INT_SET_ENABLE)	32	W	0000_0000h
FE0h	Interrupt status (PKC_INT_STATUS)	32	R	0000_0000h
FE4h	Interrupt enable (PKC_INT_ENABLE)	32	R	0000_0000h
FE8h	Interrupt status clear (PKC_INT_CLR_STATUS)	32	W	0000_0000h
FECh	Interrupt status set (PKC_INT_SET_STATUS)	32	W	0000_0000h
FFCh	Module ID (PKC_MODULE_ID)	32	R	E103_1280h

16.6.2 Status Register (PKC_STATUS)

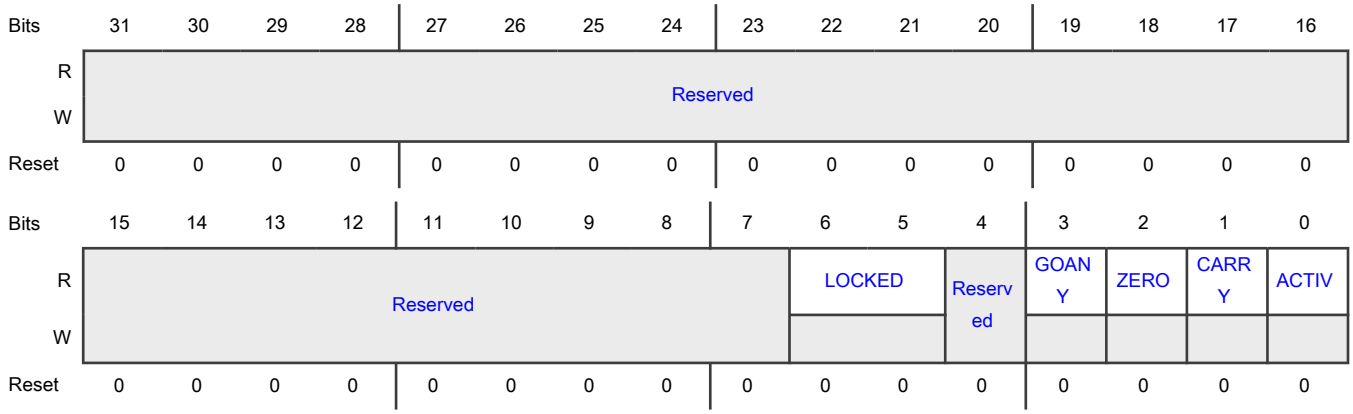
Offset

Register	Offset
PKC_STATUS	0h

Function

Contains PKC primary status information.

Diagram



Fields

Field	Function
31-7 —	Reserved
6-5 LOCKED	Parameter set locked Indicates if parameter set is locked due to a pending calculation start or can be overwritten. x1b- parameter set 1 is locked 1xb- parameter set 2 is locked LOCKED[0] is set in case a PKC calculation is started with parameter set 1 via PKC_CTRL[GOD1] or PKC_CTRL[GOM1]. In case one of the parameter set 1 SFRs PKC_MODE1, PKC_XYPTR1, PKC_ZRPTR1, PKC_LEN1 is updated while LOCKED[0] is set a PKC security alarm will be triggered (PKC_ACCESS_ERR[CTRL] is set). LOCKED[1] is set in case a PKC calculation is started with parameter set 2 via PKC_CTRL[GOD2] or PKC_CTRL[GOM2]. In case one of the parameter set 2 SFRs PKC_MODE2, PKC_XYPTR2, PKC_ZRPTR2, PKC_LEN2 is updated while LOCKED[1] is set a PKC security alarm will be triggered (PKC_ACCESS_ERR[CTRL] is set). Bits 1:0 are not set for a universal pointer fetch (layer2) PKC calculation started via PKC_CTRL[GOU]. All parameter set SFRs can be updated during a universal pointer fetch operation, except PKC_LEN1 and PKC_LEN2. Updating PKC_LEN1 or PKC_LEN2 during a layer2 calculation indicated by PKC_STATUS[GOANY] will trigger a PKC security alarm (PKC_ACCESS_ERR[CTRL] is set). Bits 1:0 are cleared with the start of the calculation in parallel with the 1-to-0 transition of PKC_STATUS[GOANY] and in case PKC_CTRL[RESET] is set. **NOTE** This field is volatile.
4 —	Reserved
3 GOANY	Combined GO status flag

Table continues on the next page...

Table continued from the previous page...

Field	Function
	Set in case either PKC_CTRL[GOD1], [GOD2], [GOM1], [GOM2] or [GOU] is set. The 1-to-0 transition of GOANY indicates that a calculation has been started and that a new GO bit can be set. If GOANY is cleared also all PKC_STATUS[LOCKED] bits are cleared to indicate that the parameter set can be updated. GOANY is always '0' in case PKC_CTRL.RESET is set. **NOTE** This field is volatile.
2 ZERO	Zero result flag Set by the PKC at the end of a calculation in case the result of the calculation is equal zero. ZERO is updated for each PKC calculation mode, except for MUL1 (opcode 0x20) and MUL1_GF2 (opcode 0x24). **NOTE** A set ZERO flag only indicates that the result written to PKC RAM is zero, but does not imply that there is no overflow (i.e., CARRY flag set). If PKC is in reset (PKC_CTRL.RESET=1), ZERO is set to logic 0 (default). **NOTE** This field is volatile.
1 CARRY	Carry overflow flag Set by the PKC at the end of a calculation in case • an addition or multiplication with addition operation has been executed and an overflow in the most significant bit has occurred. • a subtraction or multiplication with subtraction operation has been executed with negative result in twos complement notation. • a shift or rotate operation has been executed (most/least significant shifted/rotated bit is stored in CARRY flag). • a LSB0s or MSB0s operation has been executed (CARRY indicates zero least/most-significant zero bits). • a compare operation have been executed with Z>Y. CARRY is updated for each PKC calculation mode, except for MUL1 (opcode 0x20) and MUL1_GF2 (opcode 0x24). This holds also for operation modes where CARRY is always zero, e.g. logical operations (AND, OR, ...). If PKC is in reset (PKC_CTRL.RESET=1), CARRY is set to logic 0 (default). **NOTE** This field is volatile.
0 ACTIV	PKC ACTIV ACTIV = 1 signals that a calculation is in progress or about to start. At the end of a calculation, ACTIV is automatically reset to logic 0 in case no further GO bit is set. If the next PKC operation has been started by setting a GO bit during a calculation, ACTIV remains high. ACTIV is always '1' in case PKC_STATUS[GOANY] is set.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	ACTIV is always '0' in case PKC_CTRL_RESET register is set. While ACTIV is '1' the PKC is active also in using the PKC RAM. Concurrent accesses to PKC RAM via the CPU is still possible, but may reduce the performance of the PKC calculation. Write accesses to the operand and result area defined by the PKC pointer and length registers is prohibited during a running calculation and may result in wrong calculation results. **NOTE** This field is volatile.

16.6.3 Control Register (PKC_CTRL)

Offset

Register	Offset
PKC_CTRL	4h

Function

Controls primary operation of ELS.

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	Reserved				REDMUL		CACHE_EN		GF2CONV						STOP	RESET
W								CLRCACHE		GOU	GOM2	GOM1	GOD2	GOD1		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Fields

Field	Function
31-12 —	Reserved
11-10 REDMUL	Reduced multiplier mode Defines the operand width processed by the PKC coprocessor. This IP version only supports 64-bit operand width, i.e., REDMUL = 0x0 or 0x2.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	NOTE Although operand width is fixed the following points need to be considered: For hard coded layer1 calculation modular multiplication and modular reduction the pre-computed inverse of the module depends on the PKC-word size setting and as such is not the same for different REDMUL configurations. The operand pointers and length definitions are independent from the REDMUL setting as long as they are 64-bit aligned. The amount of ignored least significant bits of the pointer and length SFRs depends on REDMUL. As a consequence, also the minimum supported operand length is defined via bits 1 and 2 of PKC_LEN[LEN] and bits 1,2 if PKC_LEN[MCLEN] are affected by REDMUL. 00b - full size mode, 3 least significant bits of pointer and length are ignored, minimum supported length 0x0008 01b - Reserved - Error Generated if selected 10b - 64-bit mode, 3 least significant bits of pointer and length are ignored, minimum supported length 0x0008 11b - Reserved - Error Generated if selected
9 CACHE_EN	Enable universal pointer cache If CACHE_EN=1, the cache for the universal pointer parameters is enabled. In case a parameter value is found in the cache (from a previous fetch) no DMA access is triggered. As such the amount of DMA accesses for the parameter fetch vary between 0 and 4. To further optimize the cache utilization not used parameters, e.g. XPTR for a plain addition (opcode 0x0A), could be defined equal to a used one (e.g. equal YPTR or RPTR) or a previously fetched parameter. The universal pointer cache can store up to 6 16-bit parameter values. In case the cache is full the entry that has not been used longest is overwritten by the newly fetched parameter. The cache can be enabled and disabled at any point in time, even during a running universal pointer fetch calculation.
8 CLRCACHE	Clear universal pointer cache Invalidates the cache such that all previously fetched parameters are withdrawn and have to be fetched again via DMA accesses. CLRCACHE can be triggered at any point in time, even during a running universal pointer fetch calculation. CLRCACHE is write-only and always read '0'. CLRCACHE has no impact in case the cache is disabled via PKC_CTRL[CACHE_EN]=0 or in case no layer2 calculations are executed. Beside CLRCACHE the cache is also invalidated when it is disabled (PKC_CTRL[CACHE_EN]=0) or in case of any write access to PKC_UPTRT. **NOTE** This field is volatile.
7 GF2CONV	Convert to GF2 calculation modes If GF2CONV is set, operations are mapped to their GF(2) equivalent operation modes. GF2CONV influences all PKC operation modes layer0/1/2. GF2CONV can be updated at any point in time, even during a running calculation. However the change is only applied when a new calculation is started via PKC_CTRL.GO*.

Table continues on the next page...

Table continued from the previous page...

Field	Function
6 GOU	Control bit to start pipe operation If GOU is set PKC will start the pipe / layer2 operation (parameter fetch and calculation) described in section 'PKC Universal Pointer Fetch Operation'. Section 'Operating the PKC via the Universal Pointer Fetch Unit (layer 2)' describes how to start layer2 operations. GOU is a write-only bit and always read '0'. It is only allowed to set GOU when PKC_STATUS[GOANY] and PKC_CTRL[RESET] are cleared, PKC_ULEN>0 and PKC_UPTR(T) point to valid addresses (defined by MMU). Otherwise a security alarm is triggered (PKC_ACCESS_ERR[CTRL] or PKC_ACCESS_ERR[AHB] is set). An alarm is also triggered in case more than one GO bit is set for the PKC_CTRL SFR write access (PKC_ACCESS_ERR[CTRL] is set). **NOTE** This field is volatile.
5 GOM2	Control bit to start MC pattern using parameter set 2 If GOM2 is set PKC will start a MC pattern / layer1 operation using parameter set 2 (PKC_MODE2, PKC_XYPTR2, PKC_ZRPTR2, PKC_LEN2). Section Layer 1: Operating PKC through micro code interface (L1) describes how to start layer1 operations. GOM2 is a write-only bit and always read '0'. It is only allowed to set GOM2 when PKC_STATUS[GOANY] and PKC_CTRL[RESET] are cleared. Otherwise a security alarm is triggered (PKC_ACCESS_ERR[CTRL] is set). An alarm is also triggered in case more than one GO bit is set for the PKC_CTRL SFR write access (PKC_ACCESS_ERR[CTRL] is set). **NOTE** This field is volatile.
4 GOM1	Control bit to start MC pattern using parameter set 1 If GOM1 is set PKC will start a MC pattern / layer1 operation using parameter set 1 (PKC_MODE1, PKC_XYPTR1, PKC_ZRPTR1, PKC_LEN1). Section 'Operating the PKC via the Micro Code interface (layer 1)' describes how to start layer1 operations. GOM1 is a write-only bit and always read '0'. It is only allowed to set GOM1 when PKC_STATUS[GOANY] and PKC_CTRL[RESET] are cleared. Otherwise a security alarm is triggered (PKC_ACCESS_ERR[CTRL] is set). An alarm is also triggered in case more than one GO bit is set for the PKC_CTRL SFR write access (PKC_ACCESS_ERR[CTRL] is set). **NOTE** This field is volatile.
3 GOD2	Control bit to start direct operation using parameter set 2 If GOD2 is set PKC will start a direct / layer0 operation using parameter set 2 (PKC_MODE2, PKC_XYPTR2, PKC_ZRPTR2, PKC_LEN2). Section 'Operating the PKC via the Parameter Set interface (layer 0)' describes how to start layer0 operations. GOD2 is a write-only bit and always read '0'. It is only allowed to set GOD2 when PKC_STATUS.GOANY and PKC_CTRL.RESET are cleared. Otherwise a security alarm is triggered(PKC_ACCESS_ERR[CTRL] is set). An alarm is also triggered in case more than one GO bit is set for the PKC_CTRL SFR write access (PKC_ACCESS_ERR.CTRL is set).

Table continues on the next page...

Table continued from the previous page...

Field	Function
	NOTE This field is volatile.
2 GOD1	Control bit to start direct operation using parameter set 1 If GOD1 is set, PKC will start a direct / layer0 operation using parameter set 1 (PKC_MODE1, PKC_XYPTR1, PKC_ZRPTR1, PKC_LEN1). Section 'Operating the PKC via the Parameter Set interface (layer 0)' describes how to start layer0 operations. GOD1 is a write-only bit and always read '0'. It is only allowed to set GOD1 when PKC_STATUS[GOANY] and PKC_CTRL[RESET] are cleared. Otherwise a security alarm is triggered (PKC_ACCESS_ERR[CTRL] is set). An alarm is also triggered in case more than one GO bit is set for the PKC_CTRL SFR write access (PKC_ACCESS_ERR[CTRL] is set). **NOTE** This field is volatile.
1 STOP	Freeze PKC calculation STOP=1 freezes all PKC activity incl. RAM accesses and reduces the PKC power consumption to its minimum. The difference compared to the reset of the PKC is that a stopped calculation can be continued when STOP is released (reset to '0') again. The status flags are not affected by the STOP control bit. **NOTE** When writing a new value to the STOP flag one should poll it afterwards until the new value is read back to ensure the change has been taken over properly internally. **NOTE** This field is volatile.
0 RESET	PKC reset control bit RESET=1 enforces the PKC's reset state during which a calculation cannot be started and by which any ongoing calculation process is stopped. RESET can be set/cleared by the CPU in order to switch between PKC reset and calculation enable. RESET=1 is the default state after a chip reset. RESET=1 disables PKC activity and accordingly also provides its power-saving state. When RESET is set to '1' the read-only status flags PKC_STATUS.CARRY and PKC_STATUS[ZERO] are cleared. If required these flags have to be evaluated before RESET is set. The status bits PKC_STATUS[GOANY] and PKC_STATUS[LOCKED] are reset as well in case RESET is set. RESET does not have an impact on the content of any other PKC SFRs. The parameter set and universal pointer SFRs can be read and written also while RESET is set. Software must clear bit RESET to enable the PKC before it can start a calculation via one of the GO bits in PKC_CTRL. Any attempt to start a calculation via a GO bit while RESET=1 will trigger a PKC security alarm (PKC_ACCESS_ERR[CTRL] is set). The PKC configuration setting in PKC_CFG can only be updated while RESET=1. A write access to PKC_CFG while RESET=0 will trigger a PKC security alarm (PKC_ACCESS_ERR[CTRL] is set). **NOTE** When writing a new value to the RESET flag one should poll it afterwards until the new value is read back to ensure the change has been taken over properly internally.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	NOTE This field is volatile.

16.6.4 Configuration register (PKC_CFG)

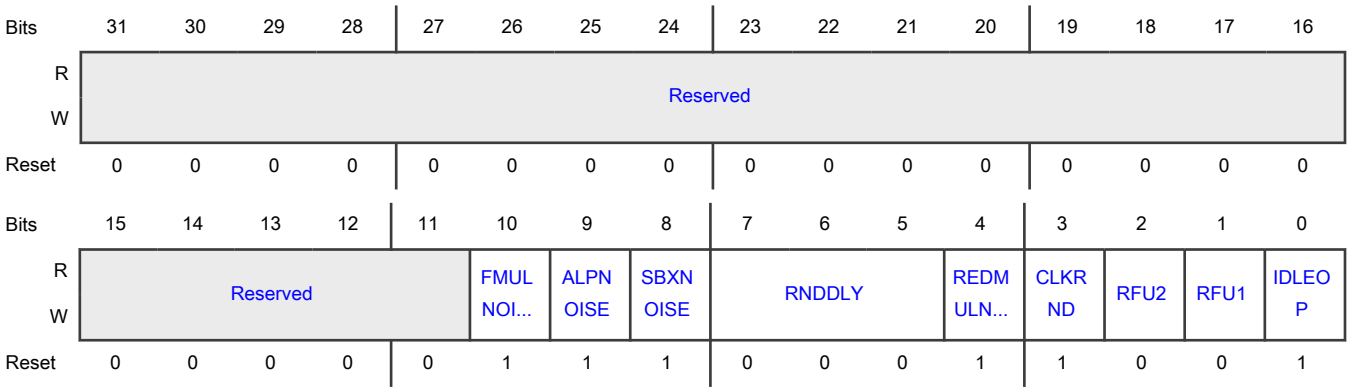
Offset

Register	Offset
PKC_CFG	8h

Function

Configuration register.

Diagram



Fields

Field	Function
31-11 —	Reserved
10 FMULNOISE	Noise feature not available in this version (flag is don't care).
9 ALPNOISE	Noise feature not available in this version (flag is don't care).
8 SBXNOISE	Noise feature not available in this version (flag is don't care).

Table continues on the next page...

Table continued from the previous page...

Field	Function
7-5 RNDDLY	Random delay feature not available in this version (flag is don't care).
4 REDMULNOISE	Noise in reduced multiplier mode feature not available in this version (flag is don't care).
3 CLKRND	Clock randomization feature not available in this version (flag is don't care).
2 RFU2	RFU
1 RFU1	RFU
0 IDLEOP	Idle operation feature not available in this version (flag is don't care).

16.6.5 Mode register, parameter set 1 (PKC_MODE1)

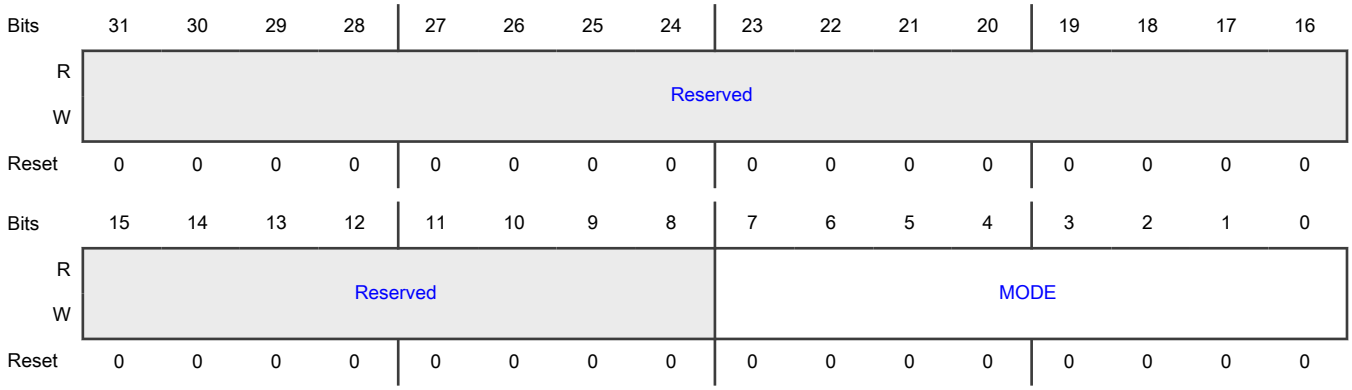
Offset

Register	Offset
PKC_MODE1	10h

Function

Mode register, parameter set 1

Diagram



Fields

Field	Function
31-8 —	Reserved
7-0 MODE	Calculation Mode / MC Start address Calculation mode of direct calculation (layer0) are listed in a table in Section 'PKC arithmetic unit (layer 0)'. In case of a not supported calculation mode for layer0 calculation a PKC security alarm is triggered (PKC_ACCESS_ERR[PKCC] is set). MC start addresses for hard coded MC pattern are listed in a table in Section 'Fixed Layer 1 MC Patterns'. The start address for the flexible MC pattern depends on the customized MC code previously stored using SFR PKC_MCDATA (data) and PKC_MODE1 (address). **NOTE** This IP version does not support flexible MC patterns (only hard coded ones). Accessing SFR PKC_MCDATA will trigger an error. **NOTE** This field is volatile.

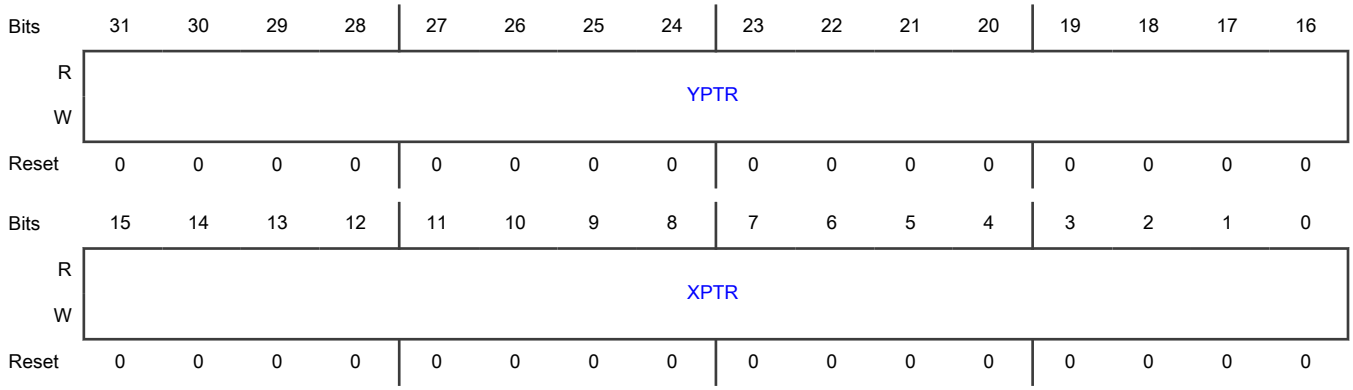
16.6.6 X+Y pointer register, parameter set 1 (PKC_XYPTR1)

Offset

Register	Offset
PKC_XYPTR1	14h

Function
X+Y pointer register, parameter set 1

Diagram



Fields

Field	Function
31-16 YPTR	Start address of Y operand in PKCRAM with byte granularity Least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size.
15-0 XPTR	Start address of X operand in PKCRAM with byte granularity Least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size.

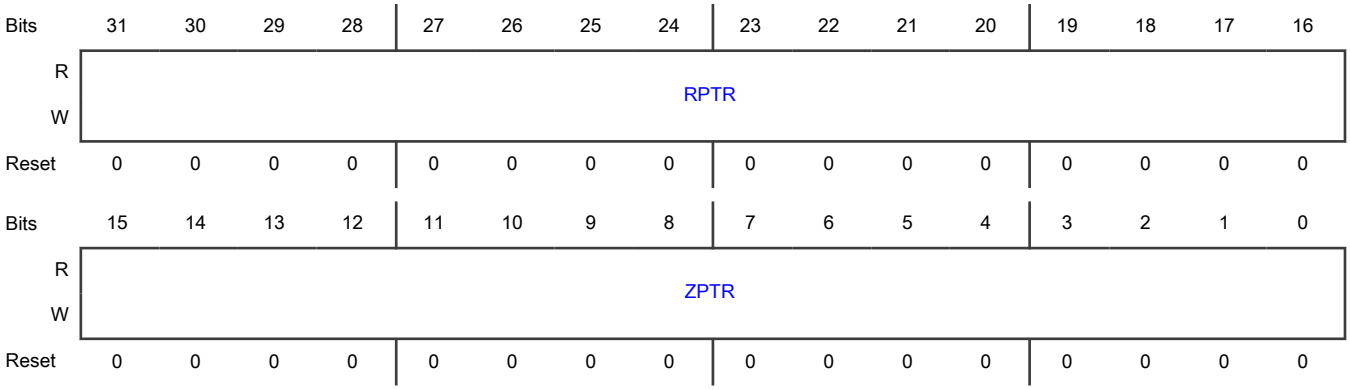
16.6.7 Z+R pointer register, parameter set 1 (PKC_ZRPTR1)

Offset

Register	Offset
PKC_ZRPTR1	18h

Function
Z+R pointer register, parameter set 1

Diagram



Fields

Field	Function
31-16 RPTR	Start address of R result in PKCRAM with byte granularity Least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size.
15-0 ZPTR	Start address of Z operand in PKCRAM with byte granularity or constant for calculation modes using CONST

Table continues on the next page...

Table continued from the previous page...

Field	Function
	If ZPTR is used as address pointer the least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size. If ZPTR is used as CONST operand the high byte is ignored and only ZPTR[7:0] is used for the calculation. For shift/rotate operation further most significant bits are ignored depending on PKC_CTRL[REDMUL].

16.6.8 Length register, parameter set 1 (PKC_LEN1)

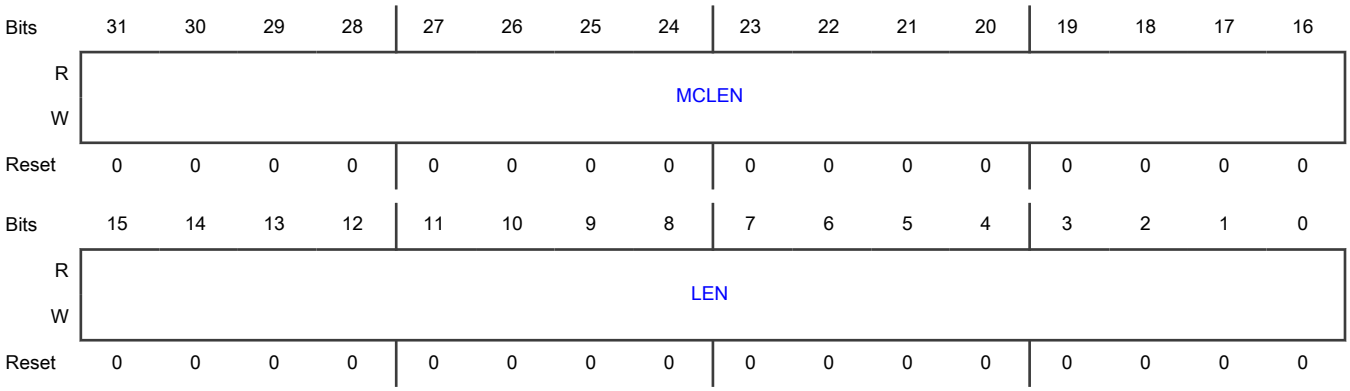
Offset

Register	Offset
PKC_LEN1	1Ch

Function

Length register, parameter set 1

Diagram



Fields

Field	Function
31-16 MCLEN	Loop counter for microcode pattern Defines the length of the loop counter that can be used in layer1 calculation mode, e.g. in MC opcode DecrTBNZ. For the hard coded MC patterns Modular Multiplication (MC start address 0x00), Plain Multiplication (0x13), Plain Multiplication with Addition (0x1D) and Modular Reduction (0x33) MCLEN defines the length of the X operand in bytes. The least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size. In case MCLEN is too short such that a DecrTBNZ MC opcode is executed on a zero loop counter value a PKC security alarm is triggered (PKC_ACCESS_ERR[PKCC] is set).

Table continues on the next page...

Table continued from the previous page...

Field	Function
15-0 LEN	Operand length Defines the length of the operands and the result in bytes. The length of Y, Z and R depend furthermore on the selected calculation mode. The least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size. In case LEN is too short such that the resulting length depending on PKC_CTRL[REDMUL] is zero a PKC security alarm is triggered (PKC_ACCESS_ERR[PKCC] is set).

16.6.9 Mode register, parameter set 2 (PKC_MODE2)

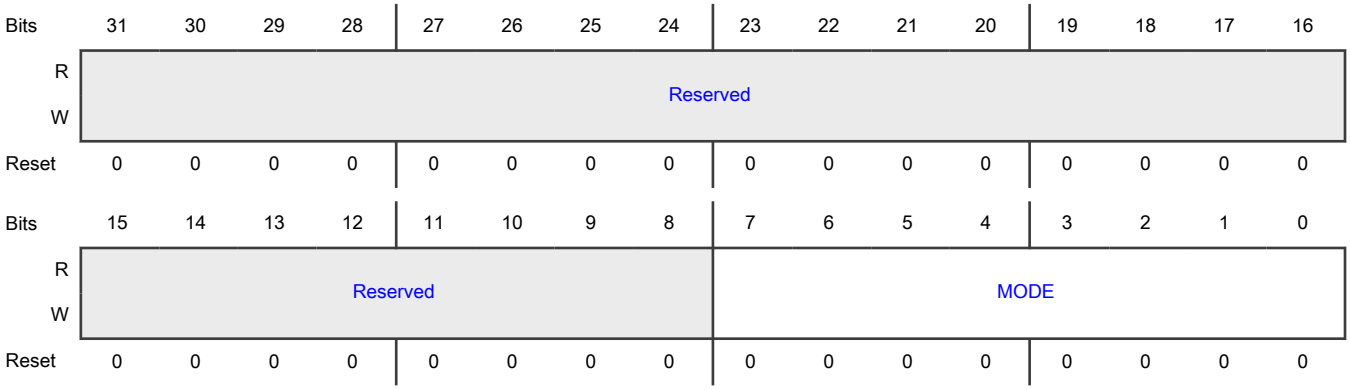
Offset

Register	Offset
PKC_MODE2	20h

Function

Mode register, parameter set 2

Diagram



Fields

Field	Function
31-8 —	Reserved
7-0 MODE	Calculation Mode / MC Start address Calculation mode of direct calculation (layer0) are listed in a table in Section 'PKC arithmetic unit (layer 0)'. In case of a not supported calculation mode for layer0 calculation a PKC security alarm is triggered

Table continues on the next page...

Table continued from the previous page...

Field	Function
	(PKC_ACCESS_ERR[PKCC] is set). MC start addresses for hard coded MC pattern are listed in a table in Section 'Fixed Layer 1 MC Patterns'. The start address for the flexible MC pattern depends on the customized MC code previously stored using SFR PKC_MCDATA (data) and PKC_MODE1 (address). **NOTE** This IP version does not support flexible MC patterns (only hard coded ones). Accessing SFR PKC_MCDATA will trigger an error.

16.6.10 X+Y pointer register, parameter set 2 (PKC_XYPTR2)

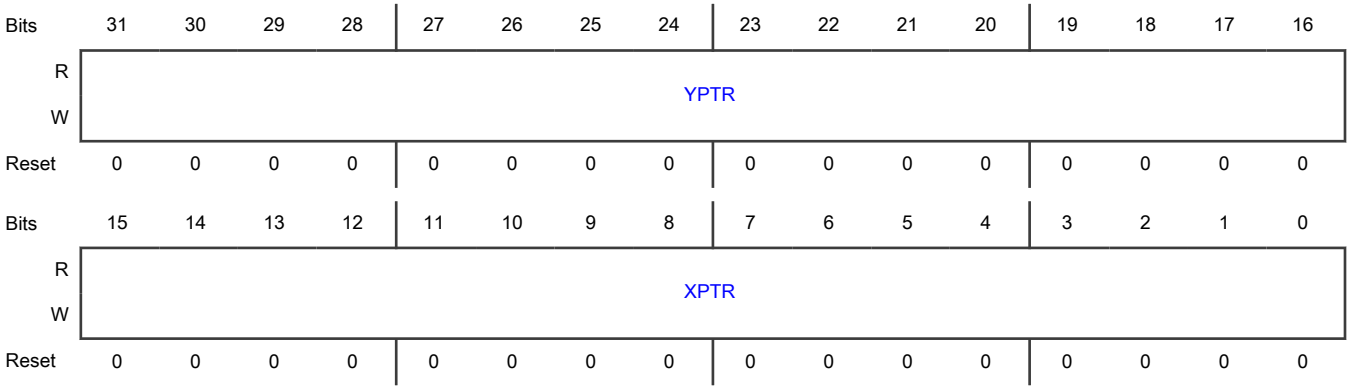
Offset

Register	Offset
PKC_XYPTR2	24h

Function

X+Y pointer register, parameter set 2

Diagram



Fields

Field	Function
31-16 YPTR	Start address of Y operand in PKCRAM with byte granularity Least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size.
15-0 XPTR	Start address of X operand in PKCRAM with byte granularity Least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size.

16.6.11 Z+R pointer register, parameter set 2 (PKC_ZRPTR2)

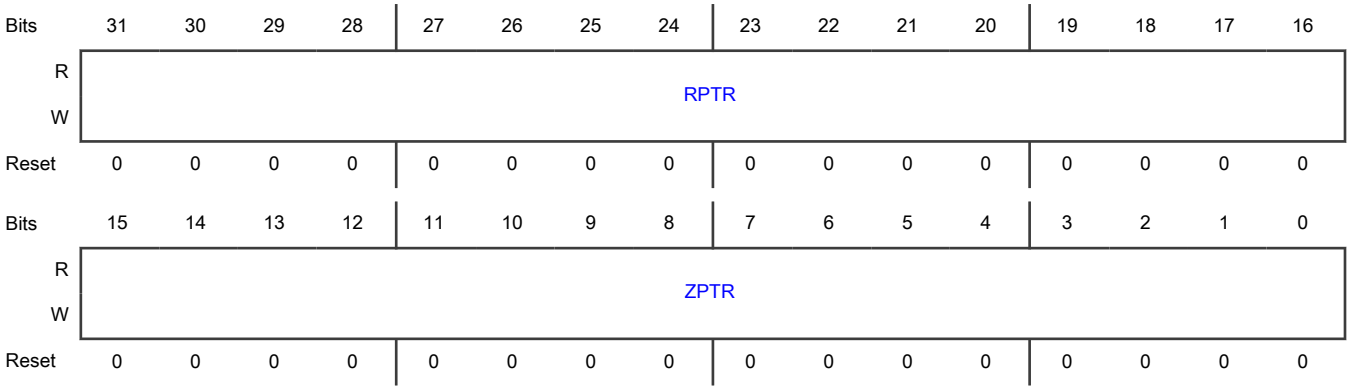
Offset

Register	Offset
PKC_ZRPTR2	28h

Function

Z+R pointer register, parameter set 2

Diagram



Fields

Field	Function
31-16 RPTR	Start address of R result in PKCRAM with byte granularity Least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size.
15-0 ZPTR	Start address of Z operand in PKCRAM with byte granularity or constant for calculation modes using CONST If ZPTR is used as address pointer the least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size. If ZPTR is used as CONST operand the high byte is ignored and only bits 7:0 of ZPTR are used for the calculation. For shift/rotate operation further most significant bits are ignored depending on PKC_CTRL[REDMUL].

16.6.12 Length register, parameter set 2 (PKC_LEN2)

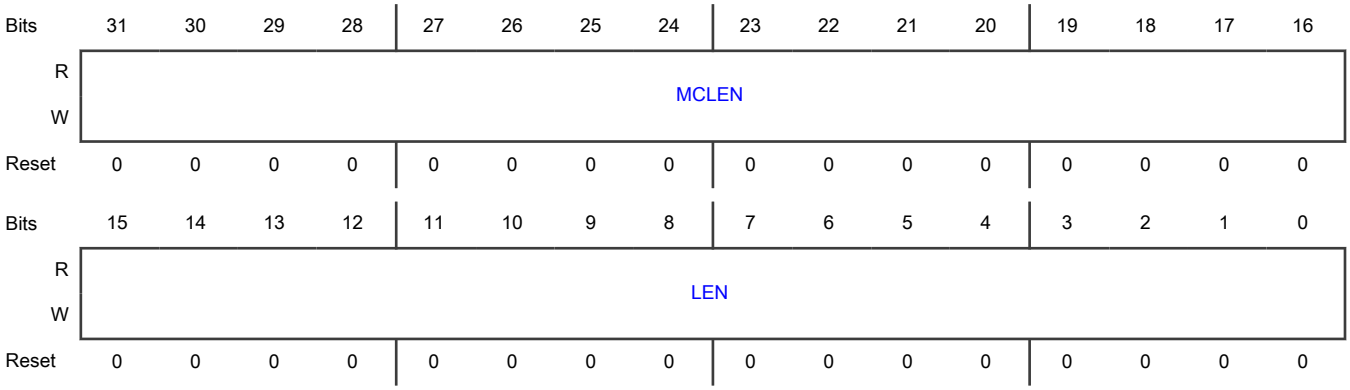
Offset

Register	Offset
PKC_LEN2	2Ch

Function

Length register, parameter set 2

Diagram



Fields

Field	Function
31-16 MCLEN	Loop counter for microcode pattern MCLEN defines the length of the loop counter that can be used in layer1 calculation mode, e.g. in MC opcode DecrTBNZ. For the hard coded MC patterns Modular Multiplication (MC start address 0x00). The least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size. In case MCLEN is too short such that a DecrTBNZ MC opcode is executed on a zero loop counter value a PKC security alarm is triggered (PKC_ACCESS_ERR[PKCC] is set).
15-0 LEN	Operand length LEN defines the length of the operands and the result in bytes. The length of Y, Z and R depend furthermore on the selected calculation mode. The least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size. In case LEN is too short such that the resulting length depending on PKC_CTRL[REDMUL] is zero a PKC security alarm is triggered (PKC_ACCESS_ERR[PKCC] is set).

16.6.13 Universal pointer FUP program (PKC_UPTR)

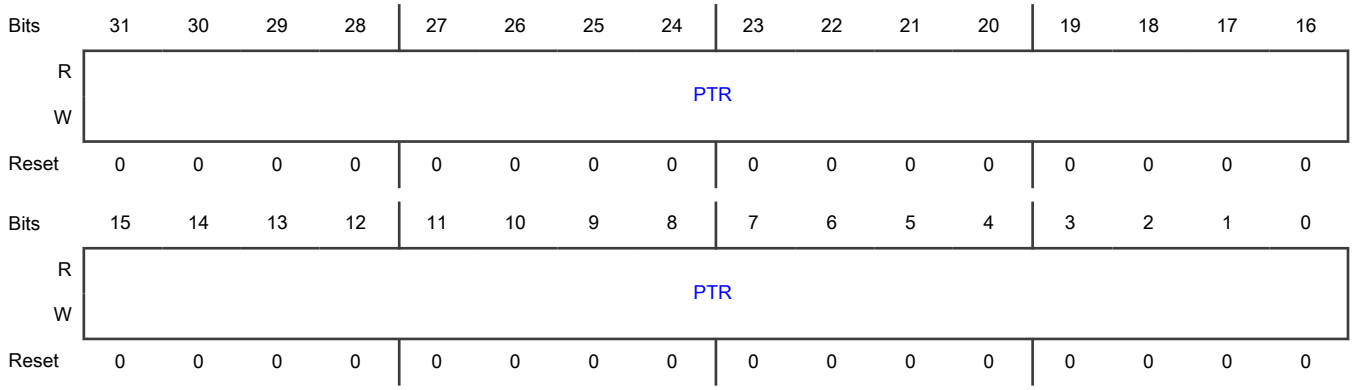
Offset

Register	Offset
PKC_UPTR	40h

Function

Universal pointer FUP program

Diagram



Fields

Field	Function
31-0 PTR	Pointer to start address of PKC FUP program PKC_UPTR needs to be defined before starting a universal pointer PKC calculation (layer2) via PKC_CTRL[GOU]. The pointer address needs to be valid and the memory space the pointer addresses needs to be enabled for PKC access by the system. Otherwise a security alarm is triggered (PKC_ACCESS_ERR[AHB] is set). TPKC_UPTR must not be updated during a running layer2 calculation while PKC_STATUS[GOANY] is set. Update of this SFR during a universal pointer fetch calculation may result in wrong calculation results and/or security alarms. PKC_UPTR is updated by HW during a layer2 calculation, resp. incremented by 6 for each processed FUP program entry. The least significant bit of PKC_UPTR is ignored as the FUP program has to be stored at even addresses. **NOTE** After modifying the FUP program it is recommended to update the PKC_UPTR register (via an SFR write access) prior to starting a new layer2 calculation to clear the internal code-fetch buffer. **NOTE** This field is volatile.

16.6.14 Universal pointer FUP table (PKC_UPTRT)

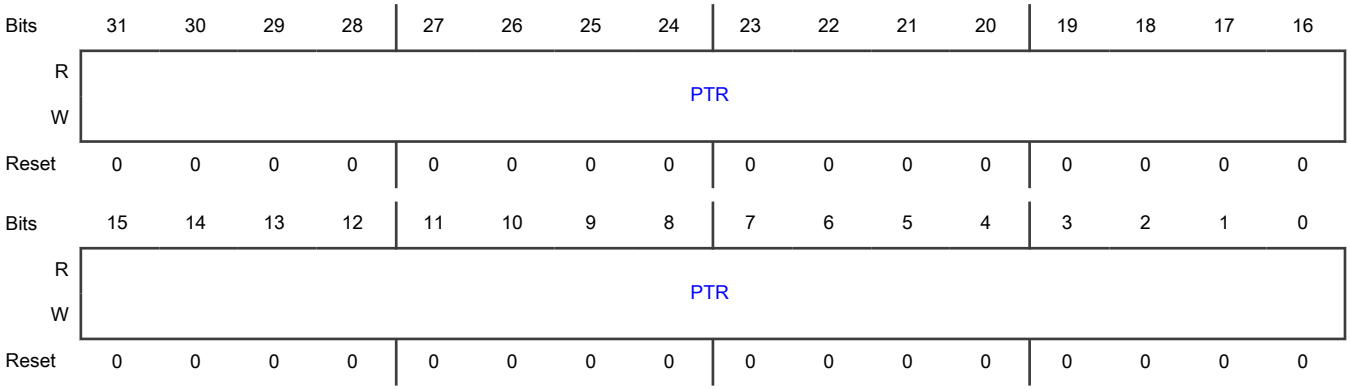
Offset

Register	Offset
PKC_UPTRT	44h

Function

Universal pointer FUP table

Diagram



Fields

Field	Function
31-0 PTR	Pointer to start address of PKC FUP table PKC_UPTRT needs to be defined before starting a universal pointer PKC calculation (layer2) via PKC_CTRL[GOU]. The pointer address needs to be valid and the memory space the pointer addresses needs to be enabled for PKC access by the system. Otherwise a security alarm is triggered (PKC_ACCESS_ERR[AHB] is set). PKC_UPTRT must not be updated during a running layer2 calculation while PKC_STATUS.GOANY is set. Update of this SFR during a universal pointer fetch calculation may result in wrong calculation results and/or security alarms. PKC_UPTRT is not updated by HW during a layer2 calculation. The least significant bit of PKC_UPTRT is ignored as the FUP table has to be stored at even addresses. Any SFR write access to PKC_UPTRT triggers the invalidation of the universal pointer cache, similar to PKC_CTRL[CLRCACHE]. **NOTE** After modifying the FUP table it is recommended to invalidate the universal pointer cache (if it is enabled) prior to starting a new layer2 calculation.

16.6.15 Universal pointer length (PKC_ULEN)

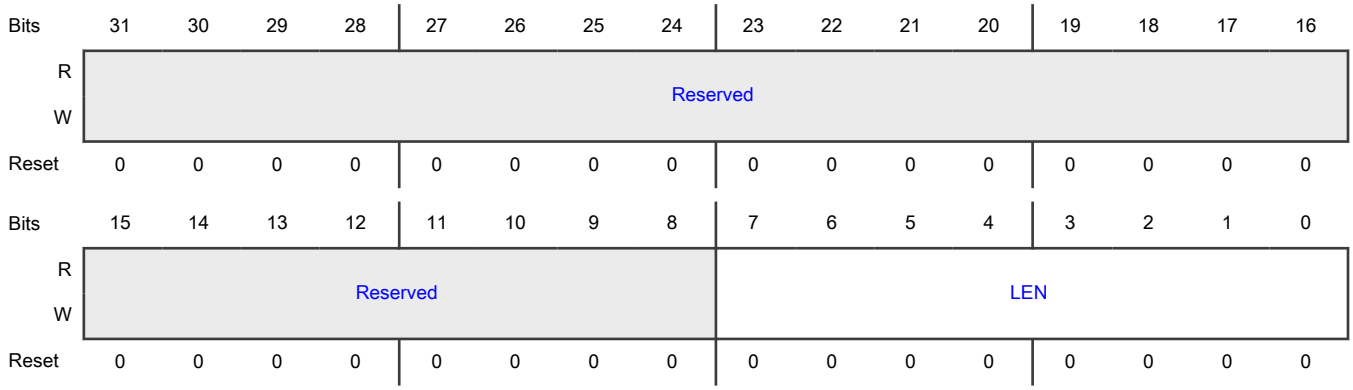
Offset

Register	Offset
PKC_ULEN	48h

Function

Universal pointer length

Diagram



Fields

Field	Function
31-8 —	Reserved
7-0 LEN	Length of universal pointer calculation PKC_ULEN defines how many FUP program entries shall be processed for one layer2 calculation started via PKC_CTRL[GOU]. The FUP program entries include layer0 calculations, layer1 calculations and CRC entries for FUP program integrity protection. PKC_ULEN must not be updated during a running layer2 calculation while PKC_STATUS[GOANY] is set. Update of this SFR during a universal pointer fetch calculation may result in wrong calculation results and/or security alarms. Starting a universal pointer fetch calculation via PKC_CTRL[GOU] while PKC_ULEN is zero triggers a security alarm (PKC_ACCESS_ERR[CTRL] is set). PKC_ULEN is updated by HW during a layer2 calculation, resp. decremented for each processed FUP program entry. When PKC_ULEN has been decremented to zero PKC_STATUS[GOANY] is cleared to indicate the start of the last calculation. When the PKC has finalized its last pipe calculation PKC_STATUS[ACTIV] is cleared and the interrupt status bit PKC_INT_STATUS[INT_PDONE] is set. **NOTE** This field is volatile.

16.6.16 MC pattern data interface (PKC_MCDATA)

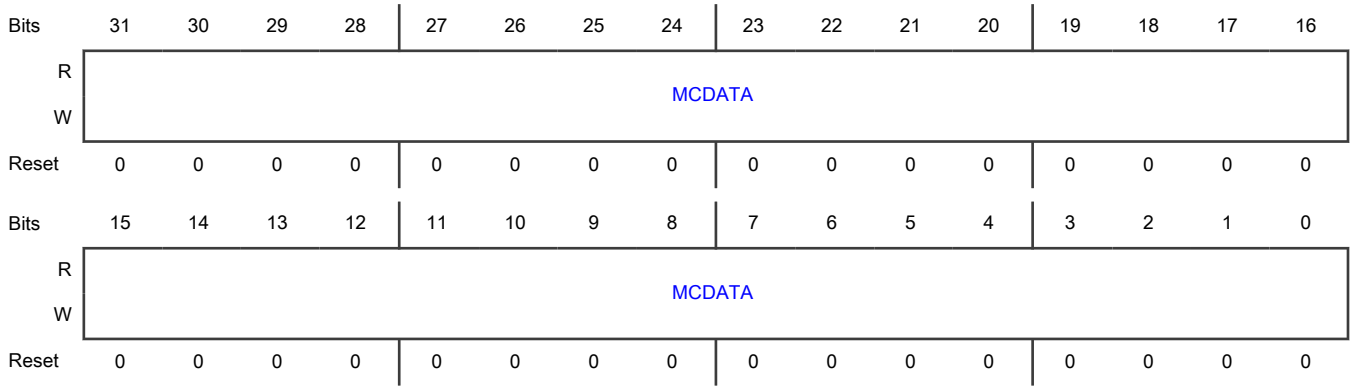
Offset

Register	Offset
PKC_MCDATA	50h

Function

MC pattern data interface

Diagram



Fields

Field	Function
31-0 MCDATA	Microcode read/write data This IP version does not support flexible MC patterns (only hard coded ones). Any read or write access to PKC_MCDATA triggers a security alarm (PKC_ACCESS_ERR[CTRL] is set). **NOTE** When trying to read from PKC_MCDATA a data value of 0 is returned. Any SFR access to PKC_MCDATA increments the PKC_MODE1 SFR by 4. **NOTE** This field is volatile.

16.6.17 PKC version register (PKC_VERSION)

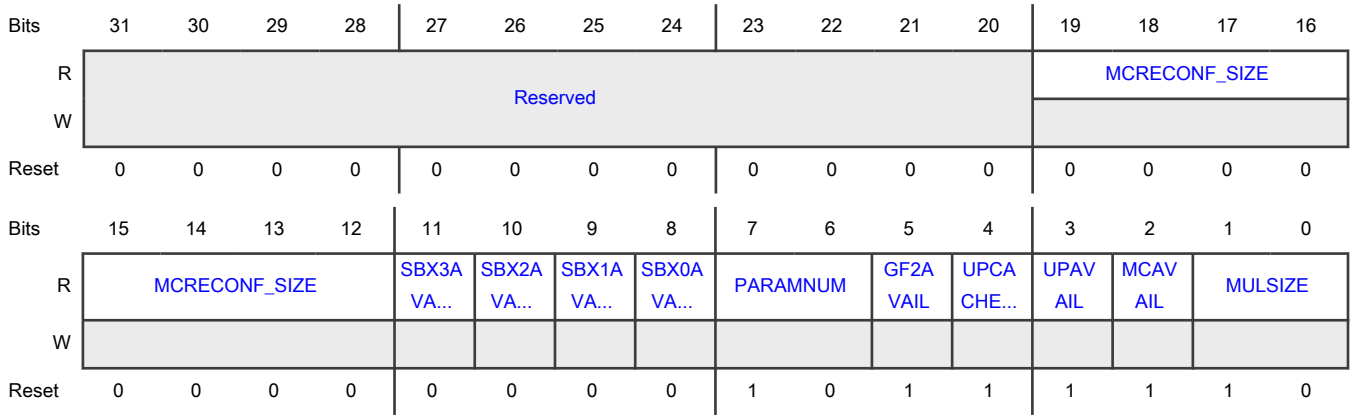
Offset

Register	Offset
PKC_VERSION	60h

Function

PKC version register

Diagram



Fields

Field	Function
31-20 —	Reserved
19-12 MCRECONF_SIZE	Size of reconfigurable MC table in bytes.
11 SBX3AVAIL	SBX3 operation is available
10 SBX2AVAIL	SBX2 operation is available
9 SBX1AVAIL	SBX1 operation is available
8 SBX0AVAIL	SBX0 operation is available
7-6 PARAMNUM	Number of parameter sets for real calculation
5 GF2AVAIL	GF2 calculation modes are available
4 UPCACHEAVAIL	UP cache is available

Table continues on the next page...

Table continued from the previous page...

Field	Function
3 UPAVAIL	UP feature (layer2 calculation) is available
2 MCAVAIL	MC feature (layer1 calculation) is available
1-0 MULSIZE	Native multiplier size and operand granularity 01b - Reserved 10b - 64-bit multiplier 11b - Reserved

16.6.18 Software reset (PKC_SOFT_RST)

Offset

Register	Offset
PKC_SOFT_RST	FB0h

Function

Software reset

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															SOFT_RST
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-1 —	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
0 SOFT_RST	Write 1 to reset module (0 has no effect). All running and pending PKC calculation are stopped. All PKC SFRs are reset except PKC_ACCESS_ERR. **NOTE** This field is volatile.

16.6.19 Access Error (PKC_ACCESS_ERR)

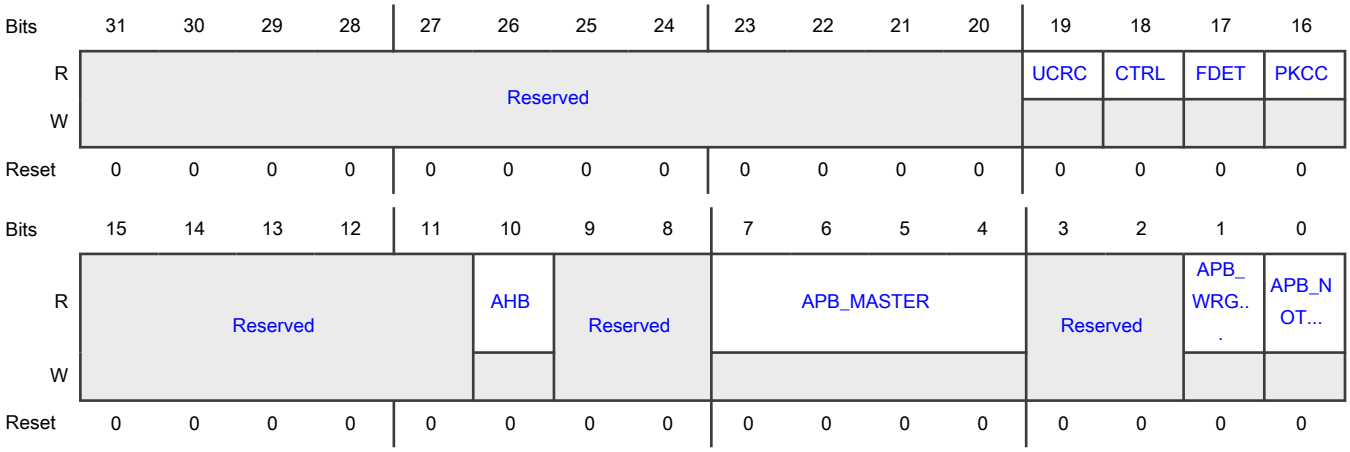
Offset

Register	Offset
PKC_ACCESS_ERR	FC0h

Function

Access Error

Diagram



Fields

Field	Function
31-20 —	Reserved
19 UCRC	Error in layer2 CRC check. **NOTE** This field is volatile.

Table continues on the next page...

Table continued from the previous page...

Field	Function
18 CTRL	Error in PKC software control **NOTE** This field is volatile.
17 FDET	Error due to error detection circuitry **NOTE** This field is volatile.
16 PKCC	Error in PKC coprocessor kernel **NOTE** This field is volatile.
15-11 —	Reserved
10 AHB	AHB Error Invalid AHB access Layer2 Only **NOTE** This field is volatile.
9-8 —	Reserved
7-4 APB_MASTER	APB Master that triggered first APB error (APB_WRGMD or APB_NOTAV) **NOTE** This bit field always reads 0. **NOTE** This field is volatile.
3-2 —	Reserved
1 APB_WRGMD	APB Error Wrong access mode **NOTE** This field is volatile.
0	APB Error

Table continues on the next page...

Table continued from the previous page...

Field	Function
APB_NOTAV	Address not available **NOTE** This field is volatile.

16.6.20 Clear Access Error (PKC_ACCESS_ERR_CLR)

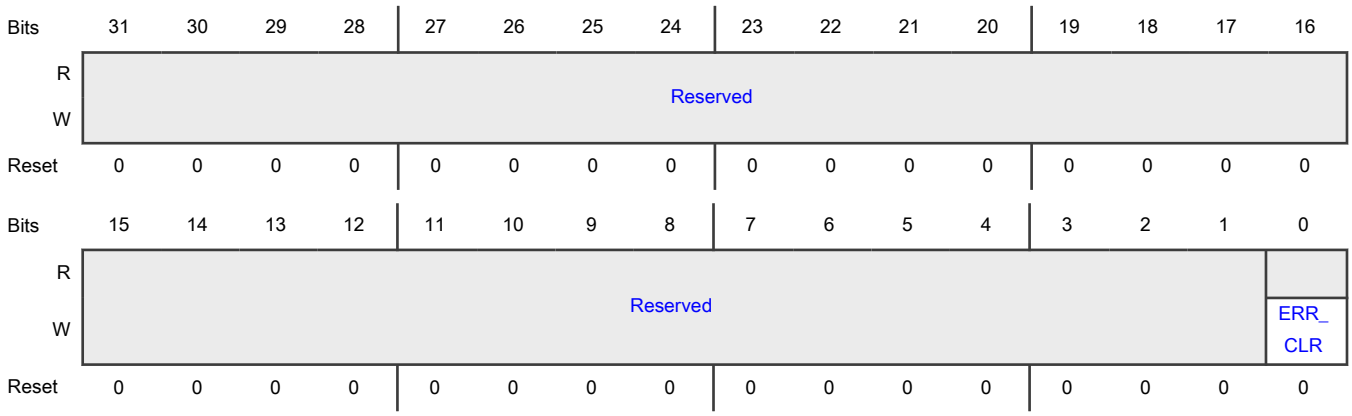
Offset

Register	Offset
PKC_ACCESS_ERR_CLR	FC4h

Function

Clear Access Error

Diagram



Fields

Field	Function
31-1 —	Reserved
0 ERR_CLR	Write 1 to reset PKC_ACCESS_ERR SFR. **NOTE** This field is volatile.

16.6.21 Interrupt enable clear (PKC_INT_CLR_ENABLE)

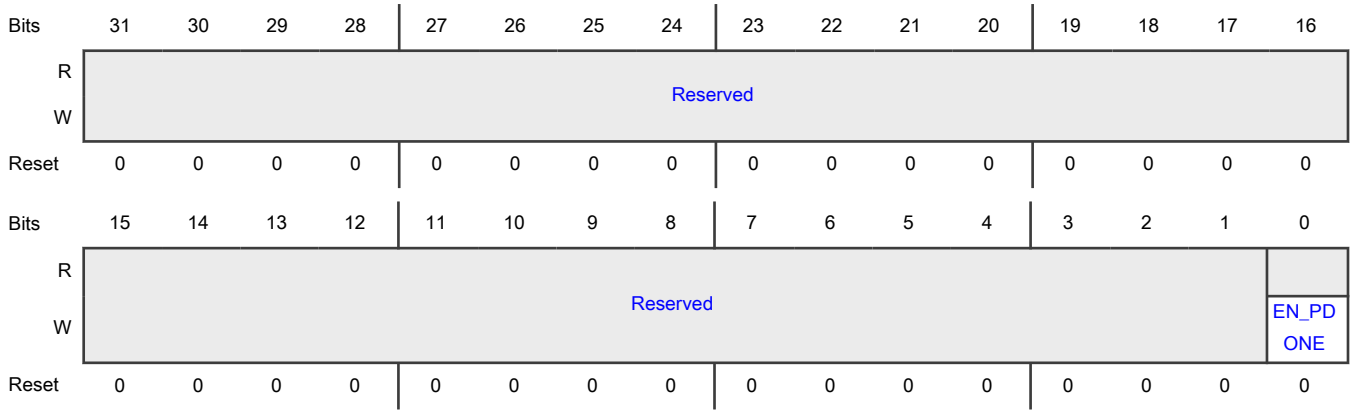
Offset

Register	Offset
PKC_INT_CLR_ENABLE	FD8h

Function

Interrupt enable clear

Diagram



Fields

Field	Function
31-1 —	Reserved
0 EN_PDONE	Write to clear PDONE interrupt enable flag (PKC_INT_ENABLE[EN_PDONE]=0). **NOTE** This field is volatile.

16.6.22 Interrupt enable set (PKC_INT_SET_ENABLE)

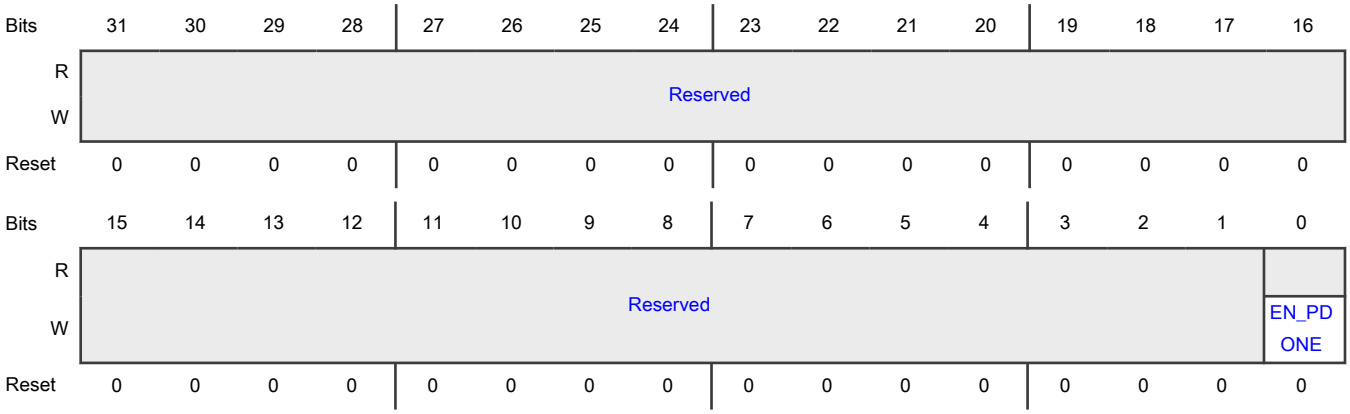
Offset

Register	Offset
PKC_INT_SET_ENABLE	FDCh

Function

Interrupt enable set

Diagram



Fields

Field	Function
31-1 —	Reserved
0 EN_PDONE	Write to set PDONE interrupt enable flag (PKC_INT_ENABLE[EN_PDONE]=1). **NOTE** This field is volatile.

16.6.23 Interrupt status (PKC_INT_STATUS)

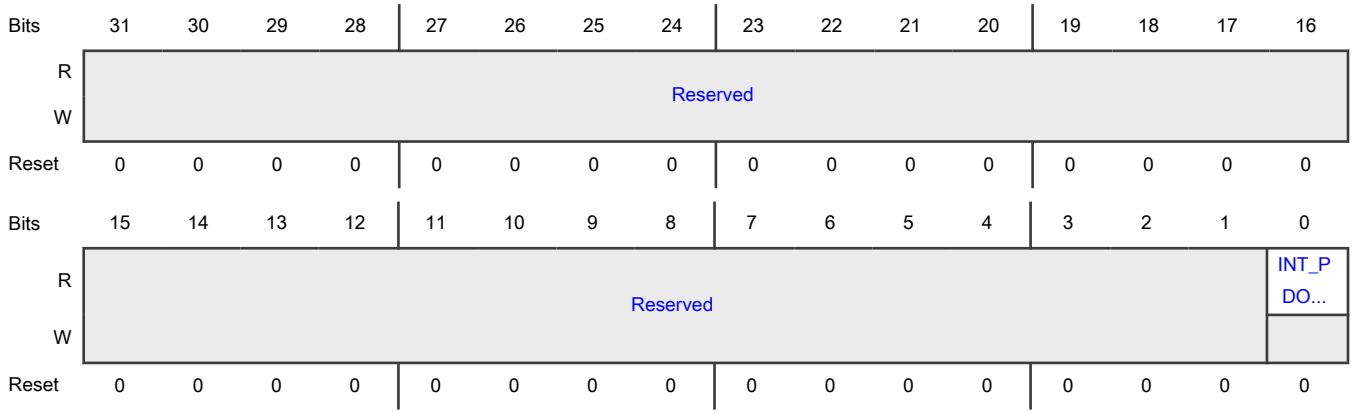
Offset

Register	Offset
PKC_INT_STATUS	FE0h

Function

Interrupt status

Diagram



Fields

Field	Function
31-1 —	Reserved
0 INT_PDONE	End-of-computation status flag INT_PDONE is set after EACH single PKC layer0 or layer1 calculation. In case of a universal pointer calculation (layer2) INT_PDONE is set at the end of the pipe calculation when PKC_ULEN has been decremented to zero and the final PKC calculation has completed. INT_PDONE is set independently from the interrupt enable PKC_INT_ENABLE[EN_PDONE]. In case PKC_INT_ENABLE[EN_PDONE]=1 an interrupt towards the CPU is triggered when INT_PDONE is set (level triggered). INT_PDONE is not cleared by PKC hardware but has to be cleared by software, except in case of a reset (chip/block reset, PKC_SOFT_RST, PKC security alarm). **NOTE** This field is volatile.

16.6.24 Interrupt enable (PKC_INT_ENABLE)

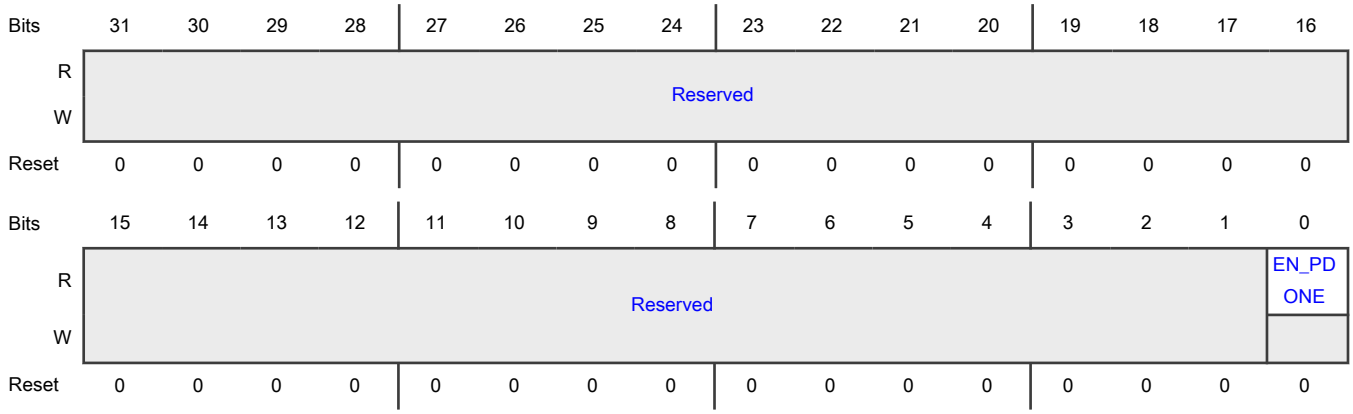
Offset

Register	Offset
PKC_INT_ENABLE	FE4h

Function

Interrupt enable

Diagram



Fields

Field	Function
31-1 —	Reserved
0 EN_PDONE	PDONE interrupt enable flag If EN_PDONE=1, an interrupt is triggered every time PKC_INT_STATUS[INT_PDONE] is set. Otherwise the interrupt generation is suppressed. **NOTE** This field is volatile.

16.6.25 Interrupt status clear (PKC_INT_CLR_STATUS)

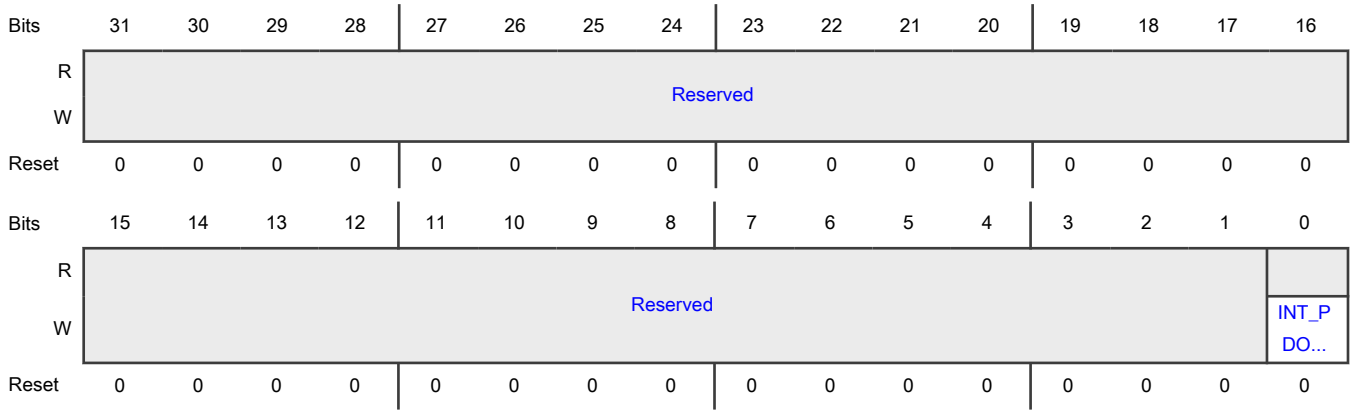
Offset

Register	Offset
PKC_INT_CLR_STATUS	FE8h

Function

Interrupt status clear

Diagram



Fields

Field	Function
31-1 —	Reserved
0 INT_PDONE	Write to clear End-of-computation status flag (PKC_INT_STATUS[INT_PDONE]=0). **NOTE** This field is volatile.

16.6.26 Interrupt status set (PKC_INT_SET_STATUS)

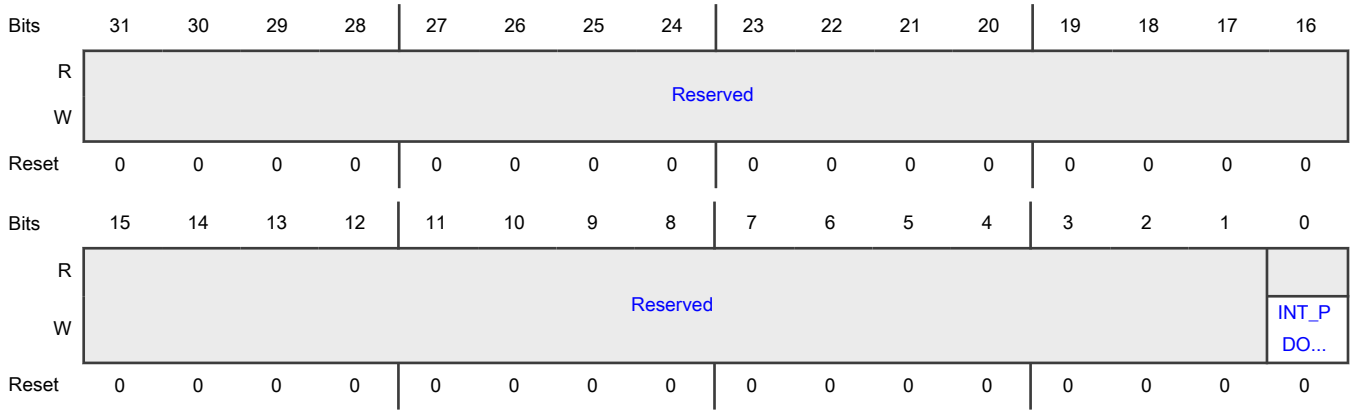
Offset

Register	Offset
PKC_INT_SET_STATUS	FECh

Function

Interrupt status set

Diagram



Fields

Field	Function
31-1 —	Reserved
0 INT_PDONE	Write to set End-of-computation status flag (PKC_INT_STATUS[INT_PDONE]=1) to trigger a PKC interrupt via software, e.g. for debug purposes. **NOTE** This field is volatile.

16.6.27 Module ID (PKC_MODULE_ID)

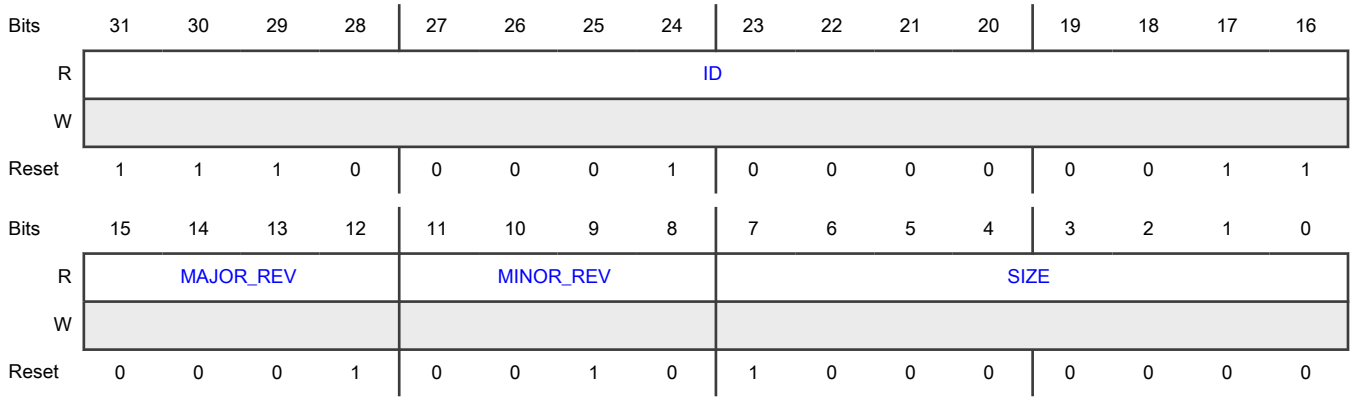
Offset

Register	Offset
PKC_MODULE_ID	FFCh

Function

Module ID

Diagram



Fields

Field	Function
31-16 ID	Module ID
15-12 MAJOR_REV	Major revision
11-8 MINOR_REV	Minor revision
7-0 SIZE	Address space of the IP

Chapter 17
True Random Generator (TRNG)

17.1 Chip-specific TRNG information

Table 170. Reference links to related information[1]

Topic	Related module	References
Full description	TRNG	TRNG
System memory map		System memory map
Clocking		See the chapter "Clocking"
Signal multiplexing	Port control	See the chapter "Signal multiplexing"
System debug		See the chapter "Debug"

1. For all the reference sections and chapters mentioned in this table, refer the Reference Manual.

17.1.1 Module instance

This device has one instance of the TRNG module.

17.1.2 Security considerations

For secure applications, NXP recommends that this module is configured for secure and privileged access only to prevent unintentional or malicious modification of the system operation by nonsecure software.

17.2 Overview

The True Random Number Generator (TRNG) is a hardware accelerator module that generates 256-bit entropy as needed by an entropy-consuming module or by other post-processing functions. An appropriate entropy consumer is a deterministic random bit generator (DRBG) that can be implemented to achieve both randomness and cryptographic-strength using the TRNG output as its seed. The DRBG is not part of this module.

The entropy generated by a TRNG is intended to be used by functions (for example, to seed DRBG), whose output then can be used by functions that generate secret keys, per-message secrets, random challenges, and other similar quantities used in cryptographic algorithms. In each of these cases, it is important that a random number be difficult to guess or predict. It is important that a random number is at least as difficult to predict as it is difficult to break the cryptographic algorithm with which it is being used. This stringent requirement is particularly difficult to fulfill if the entropy source from a TRNG contains bias and/or correlation. To increase the trustworthiness and quality of the generated random data, DRBGs are often used to post-process the output of a TRNG.

This document describes the TRNG design functionality and usage.

Note that before entropy can be obtained from the TRNG, it must be initialized by setting the appropriate TRNG registers.

17.2.1 Block Diagram

The TRNG contains the following submodules: register interface, counters, config registers, health tests, and free-running oscillators (OSC).

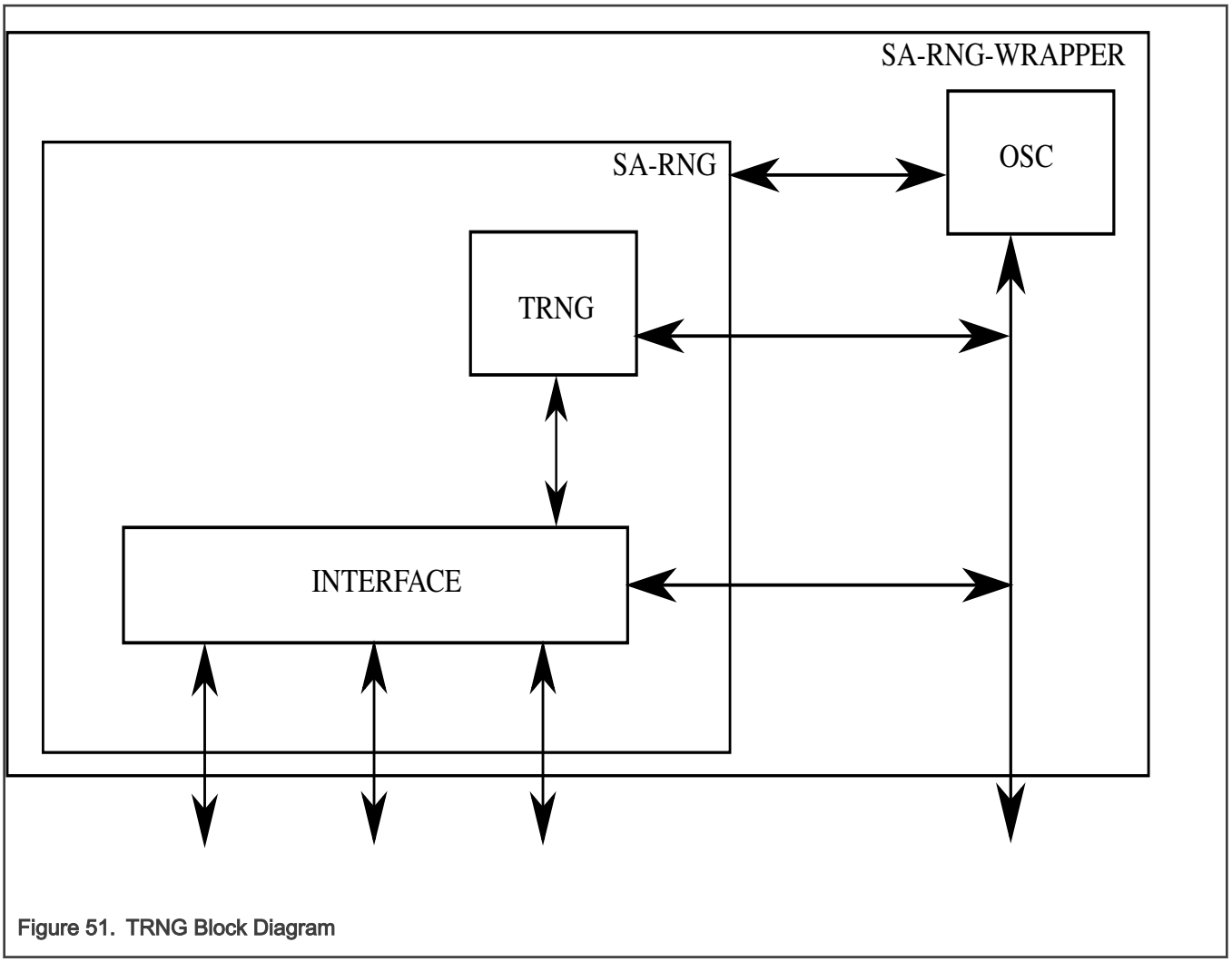


Figure 51. TRNG Block Diagram

17.3 Functional Description

The TRNG consists of several functional submodules. Its overall functionality can be described from the top-level in terms of generating entropy for seed generation.

TRNG generates random bits by collecting bits from a random noise source. This noise causes various small changes in the period of the oscillator. Therefore, if the count of the ring oscillator clock cycles is sampled after a known period of time, this count will vary each time the sample is taken. By using the variation in this count over a large number of samples, random bits can be derived. In addition to generating entropy, the TRNG also performs several statistical tests on its output.

17.4 Register Interface Usage

The TRNG register interface is used to read and write registers within TRNG for the following purposes:

- to configure TRNG
- to read the generated entropy

Note that accesses to registers must be 32-bit reads or writes.

17.5 Operation

The TRNG is designed to operate as a peripheral and can be controlled via the register interface. In order to write to many TRNG registers, the MCTL register must be set to programming mode (TRNG_MCTL[PRGM] = 1). After a power-on reset (POR) or an asynchronous system hard reset, the TRNG resets to programming mode. TRNG will not generate entropy until it has transitioned from programming mode to run mode.

There are two operations that the user (programmer/integrator) will want to do with a TRNG.

1. Initialize.

   Set up the parameters to proper values and then start generation of the first block of entropy. This is done after reset or any time the user wants to update operating parameters and/or built-in self test parameter bounds. Part of this step involves clearing of registers FRQCNT and OSC2_FRQCNT by performing a read to their corresponding address location, prior to transitioning TRNG programming mode into run mode.

   For more details, see Operation Programming Flow

2. Read entropy and generate next block of entropy.

   This is the normal flow of operation and is looped until enough entropy has been generated.

17.5.1 Operation Programming Flow

After reset, the TRNG will be in programming mode with the ok-to-stop bit asserted (MCTL[TSTOP_OK] = 1). Below is an example of a normal initialization flow.

1. Initialize TRNG:
   a. Write the desired sample size and entropy delay values to the SDCTL register.
   b. Program TRNG's self-test bounds registers. Refer to Sample Size Adjustment Programming Flow for recommended values based on sample size.
   c. Optionally, clear the FRQCNT and OSC2_FRQCNT registers by reading them.
   d. Optionally, enable interrupts in the INT_MASK register.
   e. Set the MCTL[TRNG_ACC] bit to enable access to the entropy registers and other critical TRNG registers.
2. Move TRNG into run mode:
   a. Clear MCTL[PRGM] to remove the TRNG from programming mode. This will also start entropy generation.
3. Poll TRNG (MCTL) to check if valid entropy was generated or error occured:
   a. Wait for valid entropy by polling the MCTL[TSTOP_OK] bit or waiting for a TRNG interrupt.
   b. Check whether bit MCTL[ENT_VAL] is set to indicate that the entropy is valid (no errors), or else, whether bit MCTL[ERR] is set to indicate error has occured during entropy generation.
   c. Read the 256-bit entropy from ENT0 to ENT7.

      NOTE

      Reading ENT7 will reset the entropy, so it should always be read last.

   d. Repeat this process as needed to read additional entropy values.

In Run Mode (MCTL[PRGM] = 0), the TRNG will regenerate entropy automatically after every ENT7 read. Entropy generation can be disabled by setting the TRNG back to programming mode (MCTL[PRGM]=1). MCTL[TRNG_ACC] can also be cleared, to stop the TRNG and access to TRNG registers at any point.

17.5.2 Interrupt Programming Flow

The TRNG has three interrupts that are managed using three TRNG registers: INT_CTRL, INT_STATUS and INT_MASK registers. The interrupts are FRQ_CT_FAIL, ENT_VAL and HW_ERR. By default, these interrupts are disabled. They have to be individually enabled, as needed, to work.

For example, to enable the ENT_VAL interrupt, the corresponding bit in the INT_MASK register must be set (INT_MASK[ENT_VAL] = 1). When a new interrupt is triggered, the corresponding bit in the INT_STATUS register will assert (INT_STATUS[ENT_VAL]). To clear an asserted interrupt, the corresponding bit in the INT_CTRL register is written with value 0 (INT_CTRL[ENT_VAL] = 0).

After a power cycle or reset, the TRNG interrupts need to be enabled again. Note that these interrupt bits correspond to the MCTL[FCT_FAIL], MCTL[ENT_VAL] and MCTL[ERR]. These MCTL bits will always assert/deassert regardless of whether interrupts are masked or temporarily enabled/disabled. The interrupt registers mirror these MCTL bits but are independent.

Below is an example of an interrupt setup and programming flow for ENT_VAL, HW_ERR, and FRQ_CT_FAIL interrupts can be enabled in the same way.

1. Initialize TRNG:
    a. Write the desired sample size and entropy delay values to the SDCTL register.
    b. Program TRNG's self-test bounds registers. Refer to Sample Size Adjustment Programming Flow for recommended values based on sample size.
    c. Optionally, clear the FRQCNT and OSC2_FRQCNT registers by reading them.
    d. Optionally, enable interrupts in the INT_MASK register
    e. Set the MCTL[TRNG_ACC] bit to enable access to the entropy registers and other critical TRNG registers.
    f. Turn on the ENT_VAL interrupt functionality by setting INT_CTRL[ENT_VAL] = 1 and INT_MASK[ENT_VAL] = 1.
2. Move TRNG into run mode:
    a. Clear MCTL[PRGM] to remove the TRNG from programming mode. This will also start entropy generation.
3. Wait for an interrupt from TRNG and check whether a valid entropy was generated or error occured:
    a. When entropy is ready, the active-low interrupt is expected to trigger. This should also be reflected in the interrupt status register (INT_STATUS[ENT_VAL] = 1).
    b. Check whether bit MCTL[ENT_VAL] is set to indicate that the entropy is valid (no errors), or else, whether bit MCTL[ERR] is set to indicate error has occured during entropy generation.
    c. Read the 256-bit entropy from ENT0 to ENT7.

NOTE

Reading ENT7 will reset the entropy, so it should always be read last.

    d. Reading the ENT7 register will clear the ENT_VAL interrupt. This should also be reflected in the interrupt status register (INT_STATUS[ENT_VAL] = 0).
    e. Repeat this process as needed to read additional entropy values.

17.5.3 Sample Size Adjustment Programming Flow

The sample size (SDCTL[SAMP_SIZE]) parameter can be adjusted to speed up or to limit the number of bits that an application wants to use. The greater the sample size, the more sensitive the self tests will be to entropy loss. However, this slows down entropy generation. The tradeoff between the speed versus the quality of entropy needs to be balanced per application.

Every time the sample size parameter is updated, the bounds of the built-in self tests (BIST) must also be updated to match the acceptable error ranges for the new sample size.

Below are the emperical options for some sample size parameters and their corresponding self bounds adjustment, mentioned in both table and psuedo-code formats.

Table 171. Bound values for the BIST For Some Sample Sizes

Configuration Register	SDCTL[SAMP_SIZE] Write Value		
	128	256	512
SCMISC	0x0001001D	0x0001001F	0x00010020
SCML	0x003D005E	0x005600AB	0x007A013D
SCR1L	0x00270027	0x0038003F	0x0050006B
SCR2L	0x00190018	0x00260026	0x0037003E
SCR3L	0x00120011	0x001A0019	0x00270027

Psuedo-code: Programming values for the BIST For Some Sample Sizes

```
// example Seed Control Register (SDCTL) parameters
samp_size    = 128;  // decimal

// target rejection ratio = 1.00E-06

if ( samp_size == 128 ) {
    reg32_write( TRNG_SCMISC, 0x0001001D ); // retry once, long run max 29

                                            // min range max
    reg32_write( TRNG_SCML,   0x003D005E ); //  33    61  94        monobit limit]
    reg32_write( TRNG_SCR1L,  0x00270027 ); //   0    39  39 [run length 1  limit]
    reg32_write( TRNG_SCR2L,  0x00190018 ); //  -1    25  24 [run length 2  limit]
    reg32_write( TRNG_SCR3L,  0x00120011 ); //  -1    18  17 [run length 3  limit]

} else if ( samp_size == 256 ) {
    reg32_write( TRNG_SCMISC, 0x0001001F ); // retry once, long run max 31

                                            // min range max
    reg32_write( TRNG_SCML,   0x005600AB ); //  85    86 171 [     monobit  limit]
    reg32_write( TRNG_SCR1L,  0x0038003f ); //   7    56  63 [run length 1  limit]
    reg32_write( TRNG_SCR2L,  0x00260026 ); //   0    38  38 [run length 2  limit]
    reg32_write( TRNG_SCR3L,  0x001A0019 ); //  -1    26  25 [run length 3  limit]


} else if ( samp_size == 512 ) {
    reg32_write( TRNG_SCMISC, 0x00010020 ); // retry once, long run max 32

                                            //  min range  max
    reg32_write( TRNG_SCML,   0x007A013D ); //  195   122  317 [     monobit  limit]
    reg32_write( TRNG_SCR1L,  0x0050006B ); //   27    80  107 [run length 1  limit]
    reg32_write( TRNG_SCR2L,  0x0037003E ); //    7    55   62 [run length 2  limit]
    reg32_write( TRNG_SCR3L,  0x00270027 ); //    0    39   39 [run length 3  limit]

}

// adjust delay
reg32_write( TRNG_SDCTL, ( (reg32_read( TRNG_SDCTL ) & 0xFFFF0000) | (uint32_t)samp_size ));
```

After the sample size parameter has been set, entropy can be requested normally without generating any errors.

17.6 Other Applications

The TRNG can also be used by a post-processing pseudo-random number generator function. For example, TRNG can be used to seed a hardware- or software-based implementation of a deterministic random bit generator (DRBG) as defined by SP800-90A.

17.7 TRNG register descriptions

All accesses of undefined addresses always return zero and generate a bus error. Writes to undefined and read-only addresses are ignored. Undefined addresses are those undocumented, protected or reserved addresses within the range of the addresses defined in the memory map below. The register addresses shown in the Memory Map below represent how these registers are accessed over the register bus as 32-bit words. Accesses to registers must use full-word (32-bit) reads or writes.

The format and fields in each 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.

17.7.1 TRNG memory map

TRNG0 base address: 400E_C000h

Offset	Register	Width (In bits)	Access	Reset value
0h	Miscellaneous Control Register (MCTL)	32	RW	0001_2000h
4h	Statistical Check Miscellaneous Register (SCMISC)	32	RW	0001_001Fh
8h	Poker Range Register (PKRRNG)	32	RW	0000_023Ah
Ch	Poker Maximum Limit Register (PKRMAX)	32	RW	0000_0640h
Ch	Poker Square Calculation Result Register (PKRSQ)	32	R	0000_0000h
10h	Seed Control Register (SDCTL)	32	RW	0C80_0200h
14h	Sparse Bit Limit Register (SBLIM)	32	RW	0000_003Fh
14h	Total Samples Register (TOTSAM)	32	R	0000_0000h
18h	Frequency Count Minimum Limit Register (FRQMIN)	32	RW	0000_0640h
18h	Oscillator-2 Frequency Count Register (OSC2_FRQCNT)	32	R	0000_0000h
1Ch	Frequency Count Register (FRQCNT)	32	R	0000_0000h
1Ch	Frequency Count Maximum Limit Register (FRQMAX)	32	RW	0000_6400h
20h	Statistical Check Monobit Count Register (SCMC)	32	R	0000_0000h
20h	Statistical Check Monobit Limit Register (SCML)	32	RW	0078_013Ch
24h	Statistical Check Run Length 1 Count Register (SCR1C)	32	R	0000_0000h
24h	Statistical Check Run Length 1 Limit Register (SCR1L)	32	RW	004F_006Bh
28h	Statistical Check Run Length 2 Count Register (SCR2C)	32	R	0000_0000h
28h	Statistical Check Run Length 2 Limit Register (SCR2L)	32	RW	0036_003Eh
2Ch	Statistical Check Run Length 3 Count Register (SCR3C)	32	R	0000_0000h
2Ch	Statistical Check Run Length 3 Limit Register (SCR3L)	32	RW	002D_0037h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
30h	Statistical Check Run Length 4 Count Register (SCR4C)	32	R	0000_0000h
30h	Statistical Check Run Length 4 Limit Register (SCR4L)	32	RW	001B_001Ah
34h	Statistical Check Run Length 5 Count Register (SCR5C)	32	R	0000_0000h
34h	Statistical Check Run Length 5 Limit Register (SCR5L)	32	RW	0013_0012h
38h	Statistical Check Run Length 6+ Count Register (SCR6PC)	32	R	0000_0000h
38h	Statistical Check Run Length 6+ Limit Register (SCR6PL)	32	RW	0012_0011h
3Ch	Status Register (STATUS)	32	R	0000_0000h
40h - 5Ch	Entropy Read Register (ENT0 - ENT7)	32	R	0000_0000h
80h	Statistical Check Poker Count 1 and 0 Register (PKRCNT10)	32	R	0000_0000h
84h	Statistical Check Poker Count 3 and 2 Register (PKRCNT32)	32	R	0000_0000h
88h	Statistical Check Poker Count 5 and 4 Register (PKRCNT54)	32	R	0000_0000h
8Ch	Statistical Check Poker Count 7 and 6 Register (PKRCNT76)	32	R	0000_0000h
90h	Statistical Check Poker Count 9 and 8 Register (PKRCNT98)	32	R	0000_0000h
94h	Statistical Check Poker Count B and A Register (PKRCNTBA)	32	R	0000_0000h
98h	Statistical Check Poker Count D and C Register (PKRCNTDC)	32	R	0000_0000h
9Ch	Statistical Check Poker Count F and E Register (PKRCNTFE)	32	R	0000_0000h
A0h	Security Configuration Register (SEC_CFG)	32	RW	0000_0000h
A4h	Interrupt Control Register (INT_CTRL)	32	RW	See section
A8h	Mask Register (INT_MASK)	32	RW	0000_0000h
ACh	Interrupt Status Register (INT_STATUS)	32	R	0000_0000h
B0h	Common Security Error Register (CSER)	32	R	0000_0000h
B4h	Common Security Clear Register (CSCLR)	32	W	0000_0000h
ECh	TRNG Oscillator 2 Control Register (OSC2_CTL)	32	RW	0000_1001h
F0h	Version ID Register (MS) (VID1)	32	R	0030_140Ch
F4h	Version ID Register (LS) (VID2)	32	R	0C0A_0000h

17.7.2 Miscellaneous Control Register (MCTL)

Offset

Register	Offset
MCTL	0h

Function

This register is intended to be used for programming, configuring, and testing the TRNG. It is the main register to read/write in order to enable entropy generation, to stop entropy generation, and to block access to entropy registers. This is done via the TRNG_ACC and PRGM bit below.

The Miscellaneous Control Register is a read/write register used to control the True Random Number Generator (TRNG) access, operation and test.

NOTE

Note that in many cases two TRNG 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 MCTL register.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	INTG_ERR	Reserved														PRGM
W																
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	OSC2_FA...	Reserved	TSTOP_OK	ERR	Reserved	ENT_VAL	FCT_VAL	FCT_FAIL	Reserved		TRNG_ACC		OSC_DIV		Reserved	
W				W1C						RST_DEF		DIS_SLF...				
Reset	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31 INTG_ERR	Integrity Error An internal fault has been detected. 0b - TRNG detected no internal bit error 1b - TRNG detected internal bit error(s)
30-17 —	Reserved
16 PRGM	Program Mode No Entropy value will be generated while the TRNG is in Program Mode. Note that different TRNG registers are accessible at the same address depending on whether PRGM is set to 1 or 0. This is noted in the TRNG register descriptions. **NOTE** If PRGM is set to 1, then TRNG_ACC should also be set to 1. 0b - TRNG is in Run Mode

Table continues on the next page...

Table continued from the previous page...

Field	Function
	1b - TRNG is in Program Mode
15 OSC2_FAIL	Oscillator 2 Failure OSC2_FAIL. 0b - Oscillator 2 is running. 1b - Oscillator 2 has failed (see OSC2_CTL[OSC_FAILSAFE_LMT]).
14 —	Reserved
13 TSTOP_OK	TRNG is ok to stop Software can check that this bit is a 1 before transitioning pd_main.trng0 to low power mode (pd_main.trng0 clock stopped). pd_main.trng0 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 pd_main.trng0 clock is stopped while the TRNG ring oscillator is running, the oscillator will continue running even though the pd_main.trng0 clock is stopped. TSTOP_OK is set when the TRNG ring oscillator is not running. Setting the MCTL[PRGM] bit will also cause TSTOP_OK to assert. 0b - TRNG is generating entropy and is not ok to stop 1b - TRNG is not generating entropy and is ok to stop
12 ERR	Error Status Read: Error status. Write: Write 1 to clear errors. Writing 0 has no effect. 0b - No error 1b - Error detected
11 —	Reserved
10 ENT_VAL	Entropy Valid Will set after an entropy value is generated and only if TRNG_ACC bit is set. Will be cleared at most one (1) bus clock cycle after reading the ENT7 register. (ENT0 through ENT6 should be read before reading ENT7). 0b - Entropy is not valid 1b - Entropy is valid
9 FCT_VAL	Frequency Count Valid Indicates that a valid frequency count may be read from FRQCNT. 0b - Frequency Count is not valid 1b - Frequency Count is valid
8	Frequency Count Fail

Table continues on the next page...

Table continued from the previous page...

Field	Function
FCT_FAIL	The frequency counter has detected a failure. This may be due to improper programming of the FRQMAX and/or FRQMIN registers, or a hardware failure in the ring oscillator. This error may be cleared by writing a 1 to the ERR bit. See OSC2_FAIL to determine if the failure is due to oscillator 1 or oscillator 2. • In Single-Osc1 mode (OSC2_CTL[TRNG_ENT_CTL] = 00b): — if FCT_FAIL = 1, oscillator 1 has failed • In Dual-Osc mode (OSC2_CTL[TRNG_ENT_CTL] = 01b): — if FCT_FAIL = 1, oscillator 1 has failed — if OSC2_FAIL = 1, oscillator 2 has failed and in addition, FCT_FAIL also gets set • In Single-Osc2 mode (OSC2_CTL[TRNG_ENT_CTL] = 10b): — if FCT_FAIL = 1, oscillator 2 has failed
7 —	Reserved
6 RST_DEF	Reset Defaults This bit is writable only if PRGM bit is 1, or, PRGM bit is being written to 1 simultaneously while writing to this bit. This bit has higher priority if both this and MCTL[DIS_SLF_TST] bits are written simultaneously. Reading this bit always produces a 0. This is a self-clearing bit. 0b - No impact. 1b - Writing a 1 to this bit clears various TRNG registers, and bits within registers, to their default state.
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 ENT0-ENT7 registers. The Entropy value may be read once the ENT VAL bit is asserted. Also see ENTa register descriptions (For a = 0 to 7). **NOTE** If PRGM is set to 1, then TRNG_ACC should also be set to 1.
4 DIS_SLF_TST	Disable Self-Tests Writing a 1 to this bit configures various TRNG registers with values that effectively disable internal self-tests. This bit is writable only if PRGM bit is 1, or, PRGM bit is being written to 1 simultaneously while to writing to this bit. This bit has lower priority if both this and MCTL[RST_DEF] bits are being written simultaneously. Reading this bit always produces a 0. This is a self-clearing bit.
3-2 OSC_DIV	Oscillator1 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 reset by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0x0 in this register.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - use ring oscillator with no divide 01b - use ring oscillator divided-by-2 10b - use ring oscillator divided-by-4 11b - use ring oscillator divided-by-8
1-0 —	Reserved

17.7.3 Statistical Check Miscellaneous Register (SCMISC)

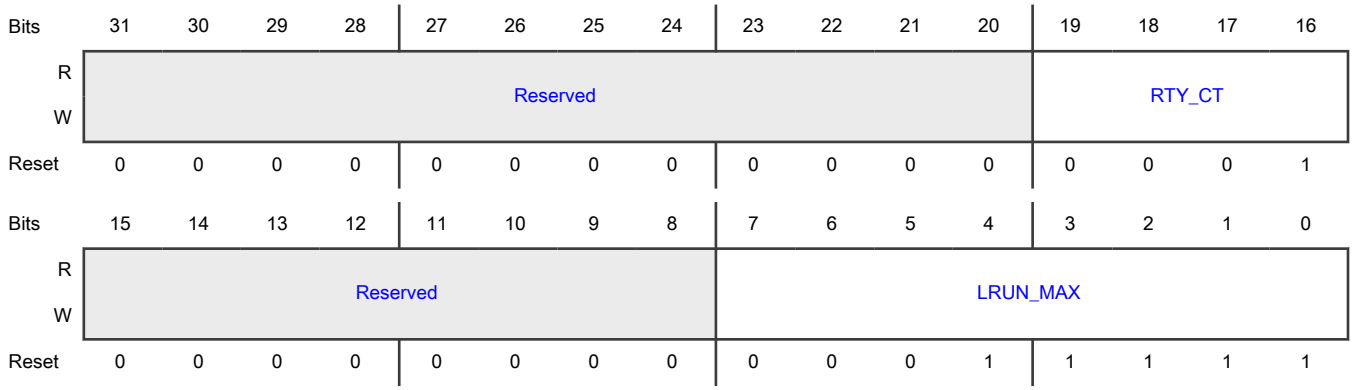
Offset

Register	Offset
SCMISC	4h

Function

The Statistical Check Miscellaneous Register contains the Long Run Maximum Limit value and the Retry Count value. This register is accessible only when the MCTL[PRGM] bit is 1, otherwise this register will read zeroes, and cannot be written.

Diagram



Fields

Field	Function
31-20 —	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
19-16 RTY_CT	Retry count. If a statistical check fails during the TRNG Entropy Generation, the RTY_CT value indicates the number of times a retry should occur before generating an error. This field is writable only if MCTL[PRGM] bit is 1. This field will read zeroes if MCTL[PRGM] = 0. This field is reset by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0x1 in this field.
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 Entropy generation. This field is writable only if MCTL[PRGM] bit is 1. This field will read zeroes if MCTL[PRGM] = 0. This field is reset by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0xFF in this field.

17.7.4 Poker Range Register (PKRRNG)

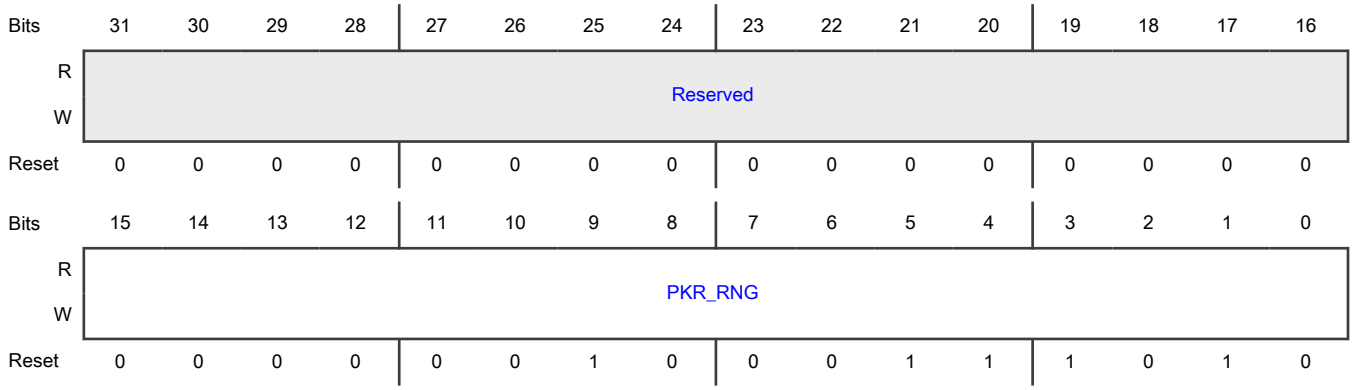
Offset

Register	Offset
PKRRNG	8h

Function

The 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. This field is reset by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0xFFFF in this field.

Diagram



Fields

Field	Function
31-16	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	
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 PKRMAX[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 MCTL[PRGM] bit is 1. This field will read zeroes if MCTL[PRGM] = 0. This field is reset by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0xFFFF in this field. Note that the minimum allowable Poker result is PKR_MAX - PKR_RNG + 1.

17.7.5 Poker Maximum Limit Register (PKRMAX)

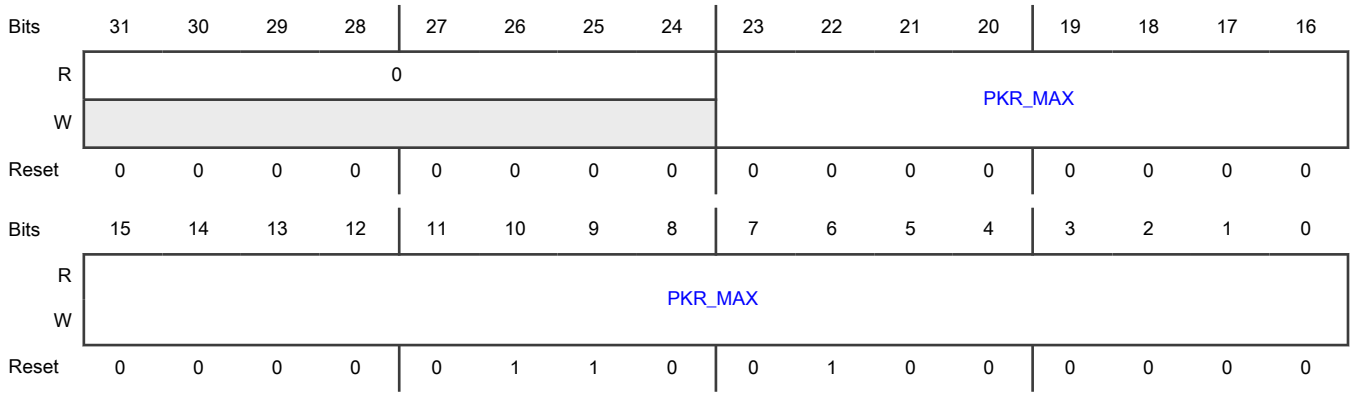
Offset

Register	Offset	Description
PKRMAX	Ch	Accessible at this address when MCTL[PRGM] = 1]

Function

The Poker Maximum Limit Register defines Maximum Limit allowable during the TRNG Statistical Check Poker Test. Note that this offset (0x0C) is used as PKRMAX only if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this offset is used as the PKRSQ readback register.

Diagram



Fields

Field	Function
31-24 —	Reserved
23-0	Poker Maximum Limit.

Table continues on the next page...

Table continued from the previous page...

Field	Function
PKR_MAX	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 PKRMAX[PKR_MAX] register. This field is writable only if MCTL[PRGM] bit is 1. This register is reset by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0xFFFF in this field. Note that the PKRMAX and PKRRNG registers combined are used to define the minimum allowable Poker result, which is PKR_MAX - PKR_RNG + 1. Note that if MCTL[PRGM] bit is 0, this register address is used to read the Poker Test Square Calculation result in register PKRSQ, as defined in the following section.

17.7.6 Poker Square Calculation Result Register (PKRSQ)

Offset

Register	Offset	Description
PKRSQ	Ch	Accessible at this address when MCTL[PRGM] = 0]

Function

The 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 PKRMAX value and decreases toward a final result, which is read here. For the Poker Test to pass, this final result must be less than the programmed PKRRNG value. Note that this offset (0x0C) is used as PKRMAX if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this offset is used as PKRSQ readback register, as described here.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0								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

Fields

Field	Function
31-24 —	Reserved
23-0	Poker Square Calculation Result.

Table continues on the next page...

Table continued from the previous page...

Field	Function
PKR_SQ	During the TRNG Statistical Checks, a "Poker Test" is run which starts with the value PKRMAX[PKR_MAX]. This value decreases according to a "sum of squares" algorithm, and must remain greater than zero, but less than the PKRRNG[PKR_RNG] limit. The resulting value may be read through this register, if MCTL[PRGM] bit is 0. Note that if MCTL[PRGM] bit is 1, this register address is used to access the Poker Test Maximum Limit in register PKRMAX, as defined in the previous section.

17.7.7 Seed Control Register (SDCTL)

Offset

Register	Offset
SDCTL	10h

Function

The 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).

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	0	0	1	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-16 ENT_DLY	Entropy Delay. Defines the length (in system clocks) of each Entropy sample taken. This field is writable only if MCTL[PRGM] bit is 1. This field will read zeroes if MCTL[PRGM] = 0. This field is reset upon assertion of TRNG's asynchronous reset input.
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 MCTL[PRGM] bit is 1. This field will read zeroes if MCTL[PRGM] = 0. This field is reset by writing the MCTL[RST_DEF] bit to 1.

17.7.8 Sparse Bit Limit Register (SBLIM)

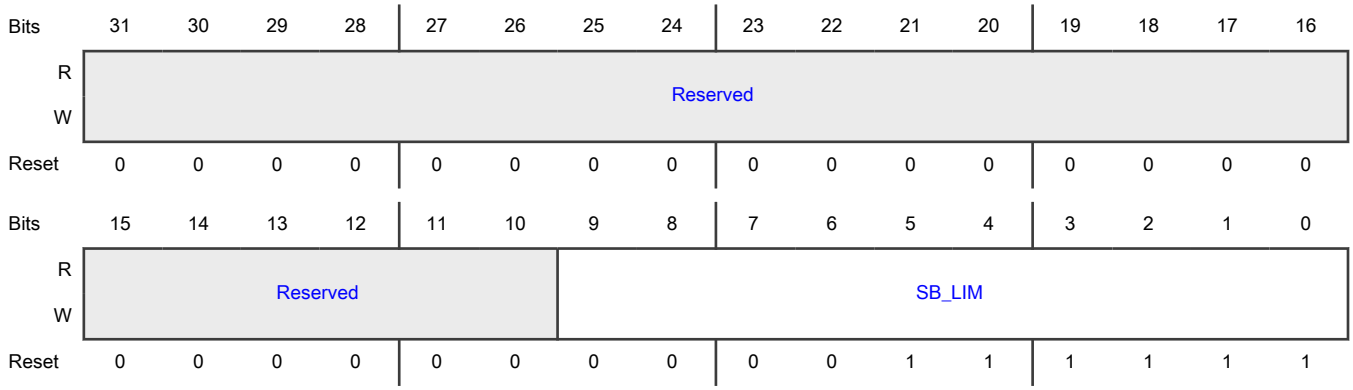
Offset

Register	Offset	Description
SBLIM	14h	Accessible at this address when MCTL[PRGM] = 1]

Function

The Sparse Bit Limit Register is used when Von Neumann sampling is selected during Entropy Generation. It defines the maximum number of consecutive Von Neumann samples which may be discarded before an error is generated. Note that this address (0x14) is used as SBLIM only if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this address is used as TOTSAM readback register.

Diagram



Fields

Field	Function
31-10 —	Reserved
9-0 SB_LIM	Sparse Bit Limit. During Von Neumann sampling (if enabled by MCTL[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 MCTL[PRGM] bit is 1. This register is reset by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0x3F in this field. Note that if MCTL[PRGM] bit is 0, this register address is used to read the Total Samples count in register TOTSAM, as defined in the following section.

17.7.9 Total Samples Register (TOTSAM)

Offset

Register	Offset	Description
TOTSAM	14h	Accessible at this address when MCTL[PRGM] = 0]

Function

The Total Samples Register is a read-only register used to read the total number of samples taken during Entropy generation. It is used to give an indication of how often a sample is actually used during Von Neumann sampling. Note that this offset (0x14) is used as SBLIM if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this offset is used as TOTSAM readback register, as described here.

Diagram

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

Fields

Field	Function
31-20 —	Reserved
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 MCTL[PRGM] bit is 0.

17.7.10 Frequency Count Minimum Limit Register (FRQMIN)

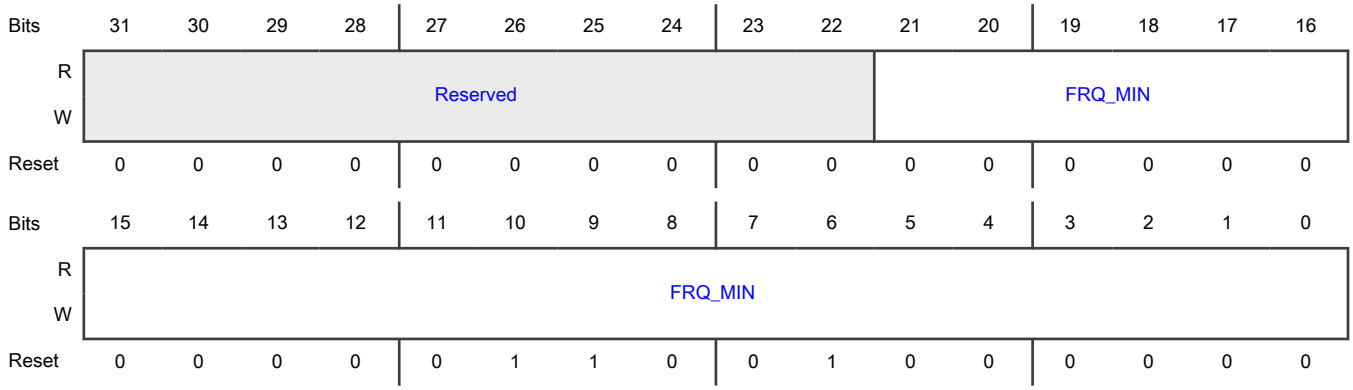
Offset

Register	Offset	Description
FRQMIN	18h	Accessible at this address when MCTL[PRGM] = 1]

Function

The 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 MCTL[FCT_FAIL] and an error is generated. Note that this address (0x18) is used as FRQMIN only if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this address is used as OSC2_FRQCNT readback register when OSC2_CTL[TRNG_ENT_CTL] = 10b. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0x0 in this field.

Diagram



Fields

Field	Function
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 MCTL[PRGM] bit is 1. This field will read zeroes if MCTL[PRGM] = 0. This field is cleared to its reset value at POR. Note that if MCTL[PRGM] bit is 0 and OSC2_CTL[TRNG_ENT_CTL] = 10b, this register address is used to read the Oscillator-2 Frequency Count result in register OSC2_FRQCNT, as defined in the OSC2_FRQ_CT field. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0x0 in this field.

17.7.11 Oscillator-2 Frequency Count Register (OSC2_FRQCNT)

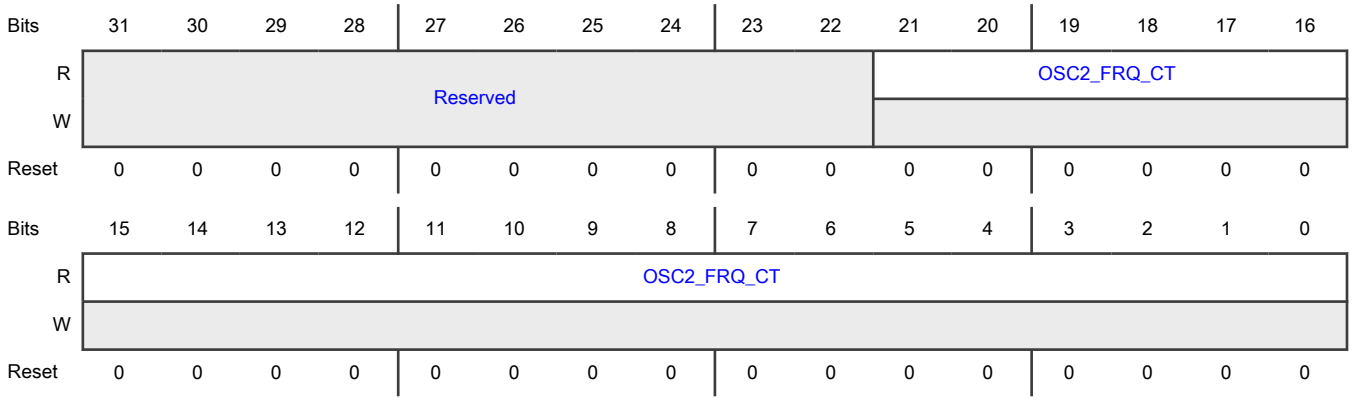
Offset

Register	Offset	Description
OSC2_FRQCNT	18h	Accessible at this address when MCTL[PRGM] = 0]

Function

The Oscillator-2 Frequency Count Register is a read-only register used to read the oscillator-2 frequency counter within the TRNG entropy generator. It will read all zeroes unless PRGM[MCTL] = 0 and MCTL[TRNG_ACC] = 1. Note that this offset (0x18) is used as FRQMIN if MCTL[PRGM] is 1. If MCTL[PRGM] is 0 and OSC2_CTL[TRNG_ENT_CTL] = 10b, this offset is used as live OSC2_FRQCNT readback register, as described here.

Diagram



Fields

Field	Function
31-22 —	Reserved
21-0 OSC2_FRQ_CT	Frequency Count. If MCTL[TRNG_ACC] = 1, Reads a sample frequency count taken during entropy generation when OSC2_CTL[TRNG_ENT_CTL] = 10b. Requires MCTL[PRGM] = 0. Note that if MCTL[PRGM] bit is 1, this register address is used to access the Frequency Minimum Limit in register FRQMIN, as defined in the FRQ_MIN field. This register must be read once, as a full 32-bit value. A subsequent read is only valid when MCTL[FCT_VAL] bit is set. A read clears this register. The next read when MCTL[FCT_VAL] bit is set must be treated as a different value from the previous one, even though the counts might be the same. If OSC2_CTL[TRNG_ENT_CTL] = 10b, OSC2_FRQ_CT shows the frequency count from TRNG oscillator 2. If OSC2_CTL[TRNG_ENT_CTL] = 00b or 01b, OSC2_FRQ_CT is 0.

17.7.12 Frequency Count Register (FRQCNT)

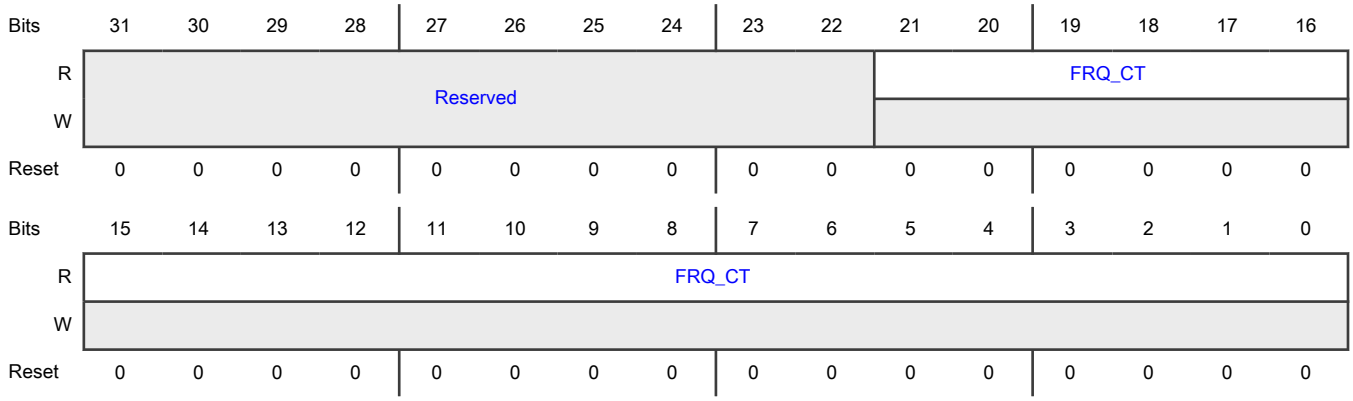
Offset

Register	Offset	Description
FRQCNT	1Ch	Accessible at this address when MCTL[PRGM] = 0]

Function

The 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 PRGM[MCTL] = 0 and MCTL[TRNG_ACC] = 1. Note that this offset (0x1C) is used as FRQMAX if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this offset is used as live FRQCNT readback register, as described here.

Diagram



Fields

Field	Function
31-22 —	Reserved
21-0 FRQ_CT	Frequency Count. If MCTL[TRNG_ACC] = 1, Reads a sample frequency count taken during entropy generation. Requires MCTL[PRGM] = 0. Note that if MCTL[PRGM] bit is 1, this register address is used to access the Frequency Maximum Limit in register FRQMAX, as defined in the FRQ_MAX field. This register must be read once, as a full 32-bit value. A subsequent read is only valid when MCTL[FCT_VAL] bit is set. A read clears this register. The next read when MCTL[FCT_VAL] bit is set must be treated as a different value from the previous one, even though the counts might be the same. If OSC2_CTL[TRNG_ENT_CTL] = 00b or 01b, FRQ_CT shows the frequency count from TRNG oscillator 1. If OSC2_CTL[TRNG_ENT_CTL] = 10b, FRQ_CT = 0, unless MCTL[FCT_FAIL] bit is set in which case it shows the frequency count from TRNG oscillator 2.

17.7.13 Frequency Count Maximum Limit Register (FRQMAX)

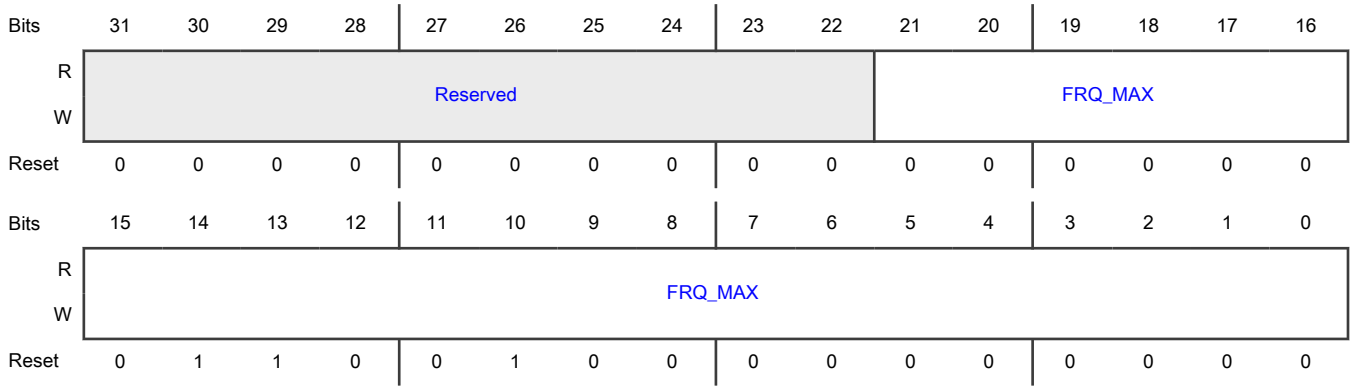
Offset

Register	Offset	Description
FRQMAX	1Ch	Accessible at this address when MCTL[PRGM] = 1]

Function

The 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 MCTL[FCT_FAIL] and an error is generated. Note that this address (0x1C) is used as FRQMAX only if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this address is used as FRQCNT readback register. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0x3FFFFF in this field.

Diagram



Fields

Field	Function
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 MCTL[PRGM] bit is 1. This field is cleared to its reset value at POR. Note that if MCTL[PRGM] bit is 0, this register address is used to read the Frequency Count result in register FRQCNT, as defined in the FRQ_CT field. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0x3FFFFF in this field.

17.7.14 Statistical Check Monobit Count Register (SCMC)

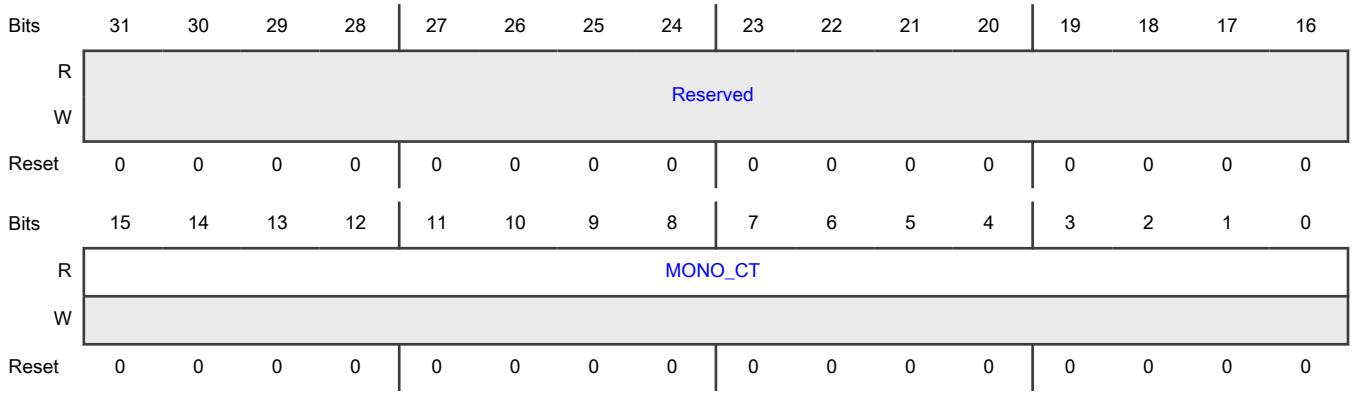
Offset

Register	Offset	Description
SCMC	20h	Accessible at this address when MCTL[PRGM] = 0

Function

The 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 SCML[MONO_MAX], and is decremented each time a one is sampled. Note that this offset (0x20) is used as SCML if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this offset is used as SCMC readback register, as described here.

Diagram



Fields

Field	Function
31-16 —	Reserved
15-0 MONO_CT	Monobit Count. Reads the final Monobit count after entropy generation. Requires MCTL[PRGM] = 0. Note that if MCTL[PRGM] bit is 1, this register address is used to access the Statistical Check Monobit Limit in register SCML, as defined in the next section.

17.7.15 Statistical Check Monobit Limit Register (SCML)

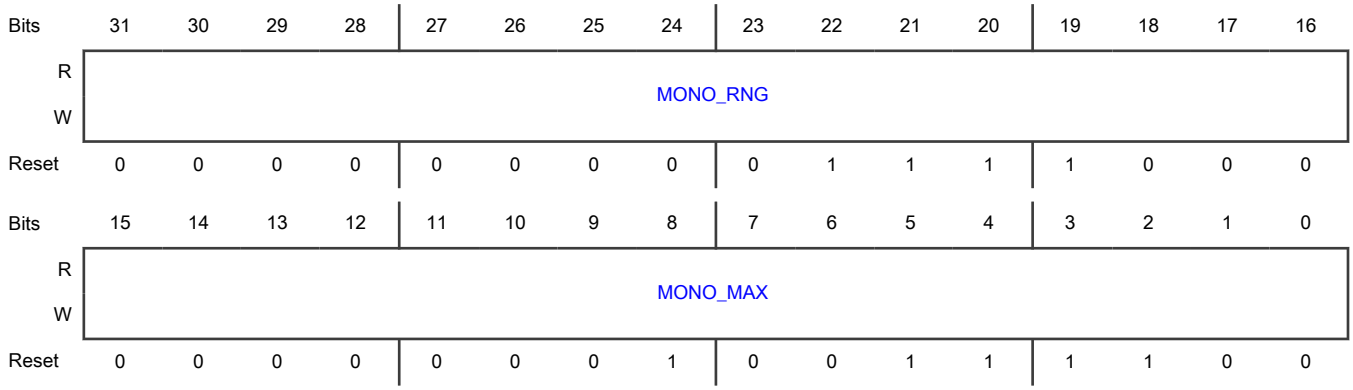
Offset

Register	Offset	Description
SCML	20h	Accessible at this address when MCTL[PRGM] = 1

Function

The 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 (MONO_MAX - MONO_RNG). If this test fails, the Retry Counter in SCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this offset (0x20) is used as SCML only if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this offset is used as SCMC readback register.

Diagram



Fields

Field	Function
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 the reset value by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0xFFFF in this field.
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 reset by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0xFFFF in this field.

17.7.16 Statistical Check Run Length 1 Count Register (SCR1C)

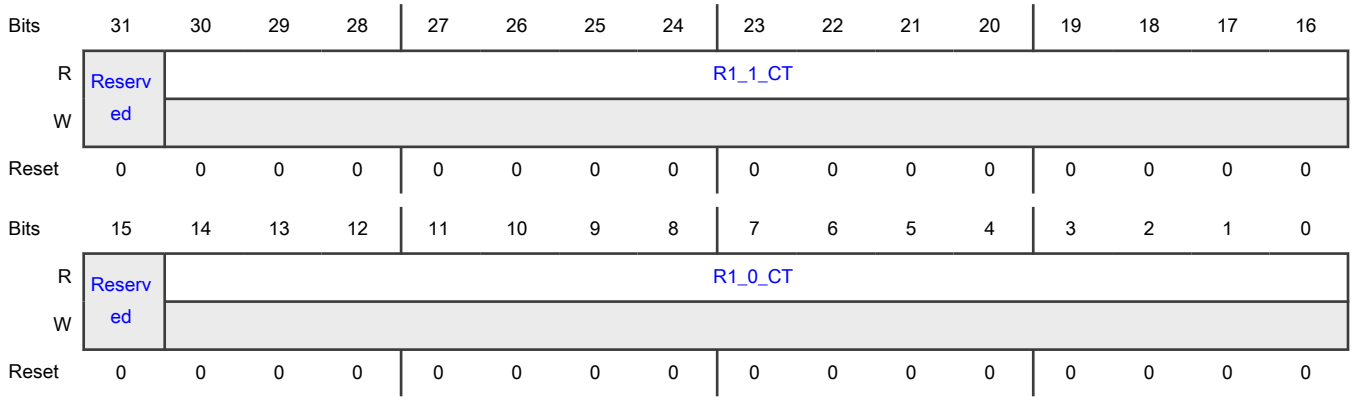
Offset

Register	Offset	Description
SCR1C	24h	Accessible at this address when MCTL[PRGM] = 0

Function

The 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 SCR1L[RUN1_MAX]. The R1_1_CT decrements each time a single one is sampled (preceded by a zero and followed by a zero). The R1_0_CT decrements each time a single zero is sampled (preceded by a one and followed by a one). Note that this offset (0x24) is used as SCR1L if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this offset is used as SCR1C readback register, as described here.

Diagram



Fields

Field	Function
31 —	Reserved
30-16 R1_1_CT	Runs of One, Length 1 Count Reads the final Runs of Ones, length 1 count after entropy generation. Requires MCTL[PRGM] = 0.
15 —	Reserved
14-0 R1_0_CT	Runs of Zero, Length 1 Count Reads the final Runs of Zeroes, length 1 count after entropy generation. Requires MCTL[PRGM] = 0.

17.7.17 Statistical Check Run Length 1 Limit Register (SCR1L)

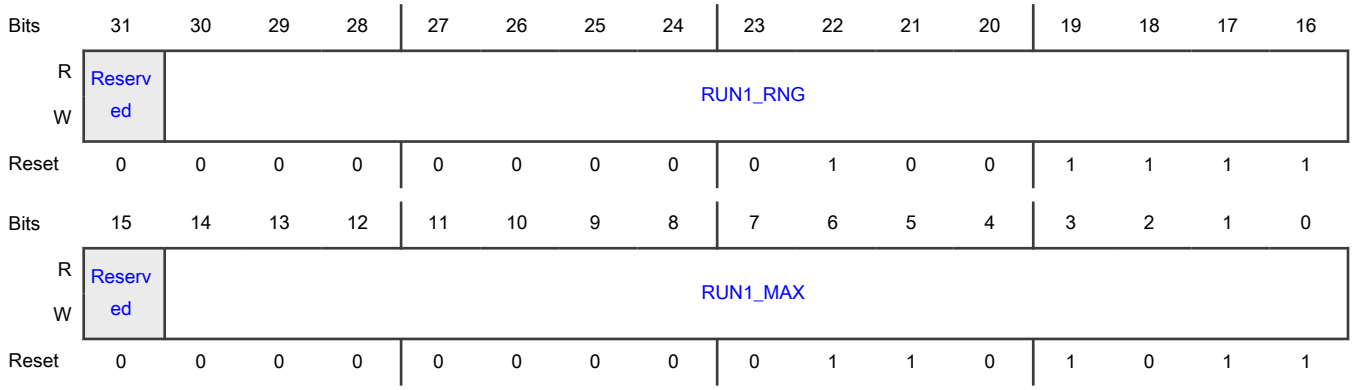
Offset

Register	Offset	Description
SCR1L	24h	Accessible at this address when MCTL[PRGM] = 1

Function

The 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 (RUN1_MAX - RUN1_RNG). If this test fails, the Retry Counter in SCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (0x24) is used as SCR1L only if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this address is used as SCR1C readback register.

Diagram



Fields

Field	Function
31 —	Reserved
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 reset by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0x7FFF in this field.
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 reset by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0x7FFE in this field.

17.7.18 Statistical Check Run Length 2 Count Register (SCR2C)

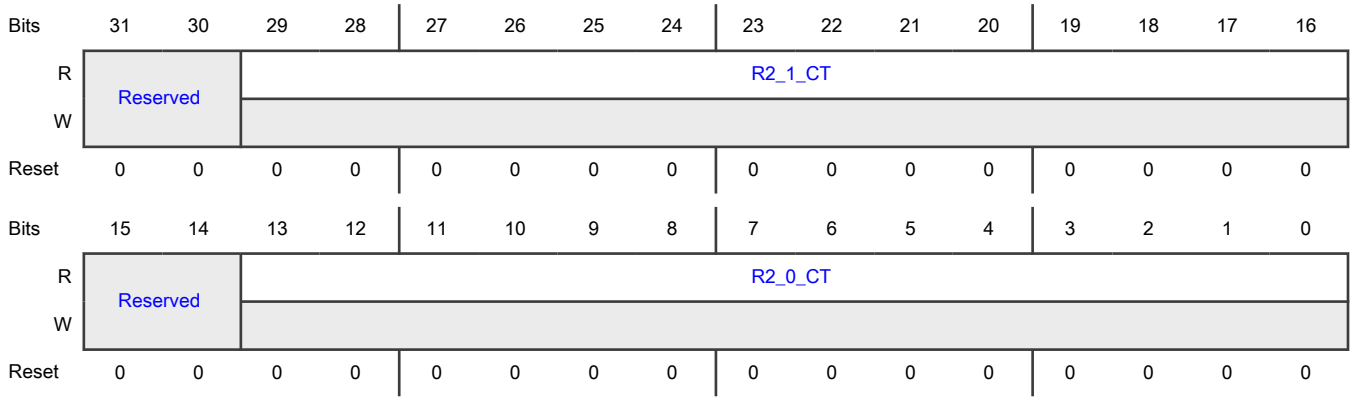
Offset

Register	Offset	Description
SCR2C	28h	Accessible at this address when MCTL[PRGM] = 0

Function

The 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 SCR2L[RUN2_MAX]. The R2_1_CT decrements each time two consecutive ones are sampled (preceded by a zero and followed by a zero). The R2_0_CT decrements each time two consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this offset (0x28) is used as SCR2L if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this offset is used as SCR2C readback register, as described here.

Diagram



Fields

Field	Function
31-30 —	Reserved
29-16 R2_1_CT	Runs of One, Length 2 Count. Reads the final Runs of Ones, length 2 count after entropy generation. Requires MCTL[PRGM] = 0.
15-14 —	Reserved
13-0 R2_0_CT	Runs of Zero, Length 2 Count. Reads the final Runs of Zeroes, length 2 count after entropy generation. Requires MCTL[PRGM] = 0.

17.7.19 Statistical Check Run Length 2 Limit Register (SCR2L)

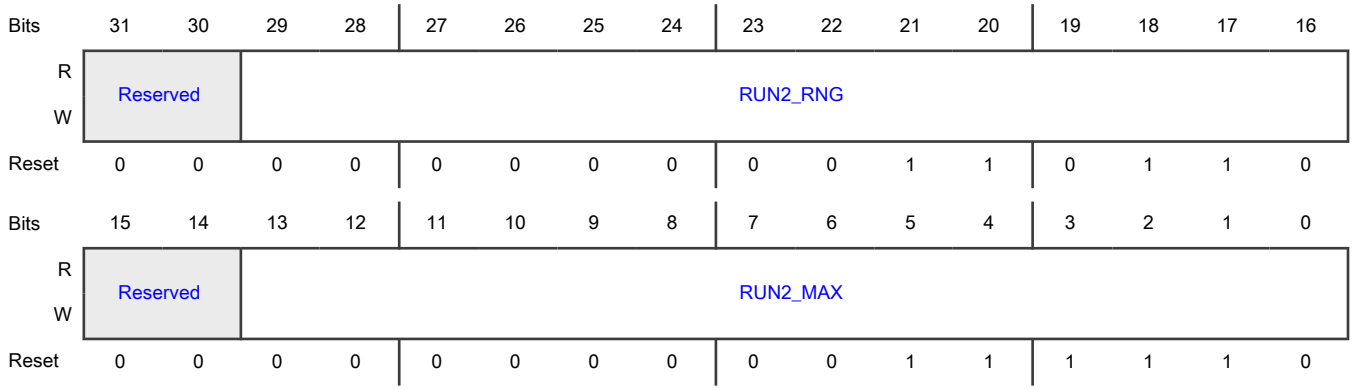
Offset

Register	Offset	Description
SCR2L	28h	Accessible at this address when MCTL[PRGM] = 1

Function

The 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 (RUN2_MAX - RUN2_RNG). If this test fails, the Retry Counter in SCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (0x28) is used as SCR2L only if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this address is used as SCR2C readback register.

Diagram



Fields

Field	Function
31-30 —	Reserved
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 reset by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0x3FFF in this field.
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 reset by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0x3FFE in this field.

17.7.20 Statistical Check Run Length 3 Count Register (SCR3C)

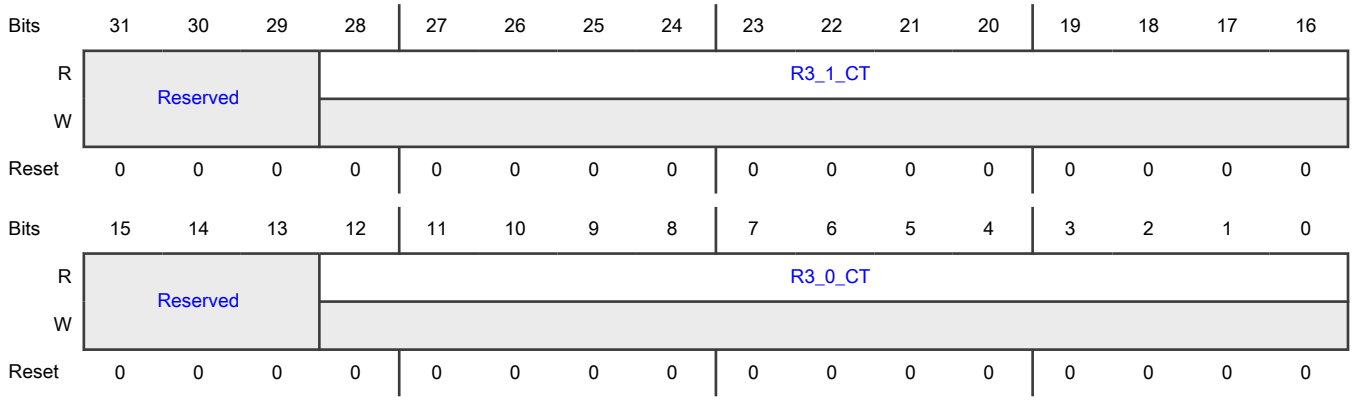
Offset

Register	Offset	Description
SCR3C	2Ch	Accessible at this address when MCTL[PRGM] = 0

Function

The 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 SCR3L[RUN3_MAX]. The R3_1_CT decrements each time three consecutive ones are sampled (preceded by a zero and followed by a zero). The R3_0_CT decrements each time three consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this offset (0x2C) is used as SCR3L if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this offset is used as SCR3C readback register, as described here.

Diagram



Fields

Field	Function
31-29 —	Reserved
28-16 R3_1_CT	Runs of Ones, Length 3 Count. Reads the final Runs of Ones, length 3 count after entropy generation. Requires MCTL[PRGM] = 0.
15-13 —	Reserved
12-0 R3_0_CT	Runs of Zeroes, Length 3 Count. Reads the final Runs of Zeroes, length 3 count after entropy generation. Requires MCTL[PRGM] = 0.

17.7.21 Statistical Check Run Length 3 Limit Register (SCR3L)

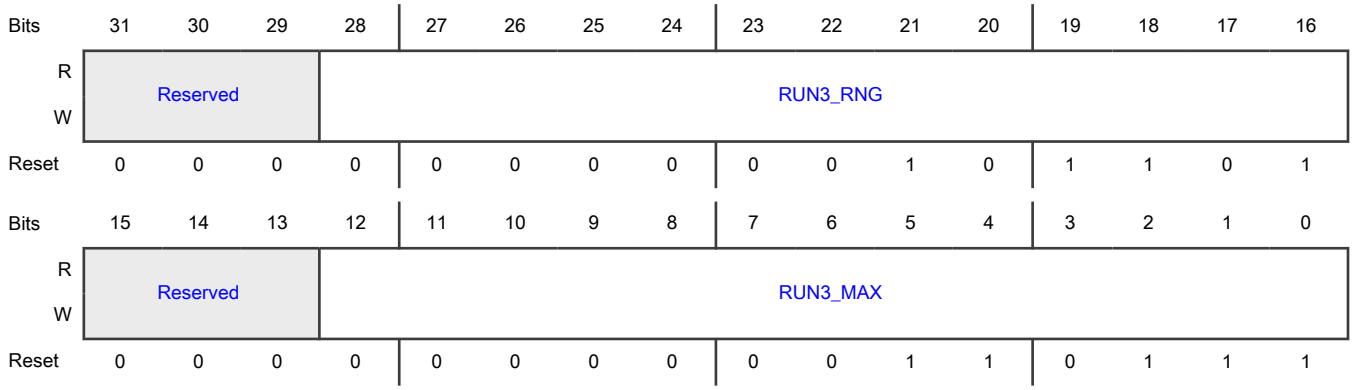
Offset

Register	Offset	Description
SCR3L	2Ch	Accessible at this address when MCTL[PRGM] = 1

Function

The 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 (RUN3_MAX - RUN3_RNG). If this test fails, the Retry Counter in SCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (0x2C) is used as SCR3L only if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this address is used as SCR3C readback register.

Diagram



Fields

Field	Function
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 reset by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0x1FFF in this field.
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 reset by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0x1FFE in this field.

17.7.22 Statistical Check Run Length 4 Count Register (SCR4C)

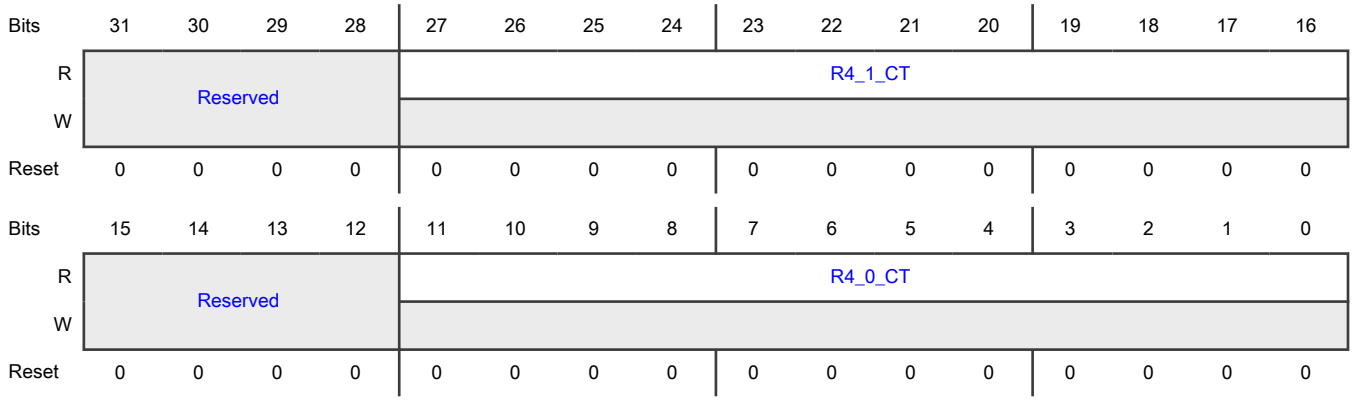
Offset

Register	Offset	Description
SCR4C	30h	Accessible at this address when MCTL[PRGM] = 0

Function

The 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 SCR4L[RUN4_MAX]. The R4_1_CT decrements each time four consecutive ones are sampled (preceded by a zero and followed by a zero). The R4_0_CT decrements each time four consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this offset (0x30) is used as SCR4L if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this offset is used as SCR4C readback register, as described here.

Diagram



Fields

Field	Function
31-28 —	Reserved
27-16 R4_1_CT	Runs of One, Length 4 Count. Reads the final Runs of Ones, length 4 count after entropy generation. Requires MCTL[PRGM] = 0.
15-12 —	Reserved
11-0 R4_0_CT	Runs of Zero, Length 4 Count. Reads the final Runs of Ones, length 4 count after entropy generation. Requires MCTL[PRGM] = 0.

17.7.23 Statistical Check Run Length 4 Limit Register (SCR4L)

Offset

Register	Offset	Description
SCR4L	30h	Accessible at this address when MCTL[PRGM] = 1

Function

The 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 (RUN4_MAX - RUN4_RNG). If this test fails, the Retry Counter in SCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (0x30) is used as SCR4L only if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this address is used as SCR4C readback register.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved				RUN4_RNG											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved				RUN4_MAX											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0

Fields

Field	Function
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 reset by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0xFFF in this field.
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 reset by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0xFFE in this field.

17.7.24 Statistical Check Run Length 5 Count Register (SCR5C)

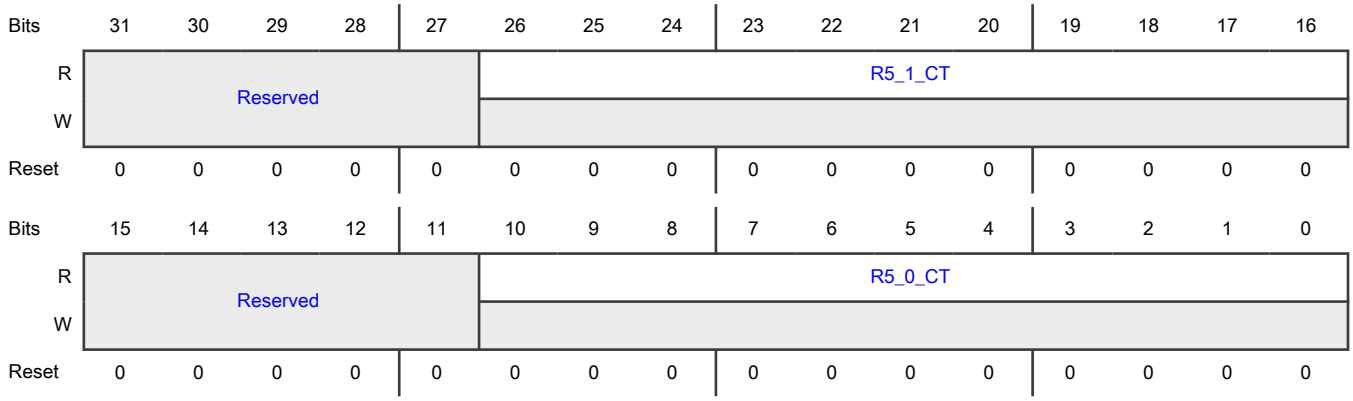
Offset

Register	Offset	Description
SCR5C	34h	Accessible at this address when MCTL[PRGM] = 0

Function

The 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 SCR5L[RUN5_MAX]. The R5_1_CT decrements each time five consecutive ones are sampled (preceded by a zero and followed by a zero). The R5_0_CT decrements each time five consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this offset (0x34) is used as SCR5L if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this offset is used as SCR5C readback register, as described here.

Diagram



Fields

Field	Function
31-27 —	Reserved
26-16 R5_1_CT	Runs of One, Length 5 Count. Reads the final Runs of Ones, length 5 count after entropy generation. Requires MCTL[PRGM] = 0.
15-11 —	Reserved
10-0 R5_0_CT	Runs of Zero, Length 5 Count. Reads the final Runs of Ones, length 5 count after entropy generation. Requires MCTL[PRGM] = 0.

17.7.25 Statistical Check Run Length 5 Limit Register (SCR5L)

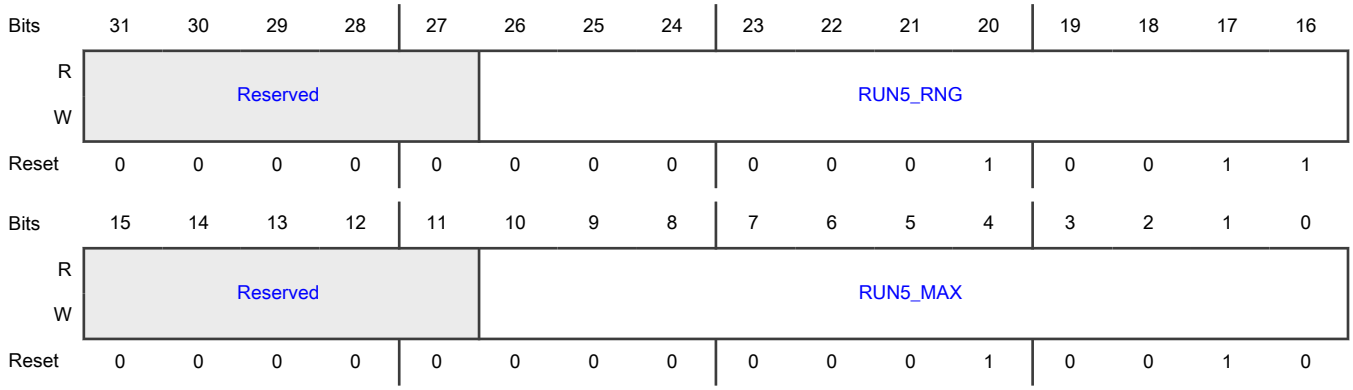
Offset

Register	Offset	Description
SCR5L	34h	Accessible at this address when MCTL[PRGM] = 1

Function

The 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 (RUN5_MAX - RUN5_RNG). If this test fails, the Retry Counter in SCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (0x34) is used as SCR5L only if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this address is used as SCR5C readback register.

Diagram



Fields

Field	Function
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 reset by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0x7FF in this field.
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 reset by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0x7FE in this field.

17.7.26 Statistical Check Run Length 6+ Count Register (SCR6PC)

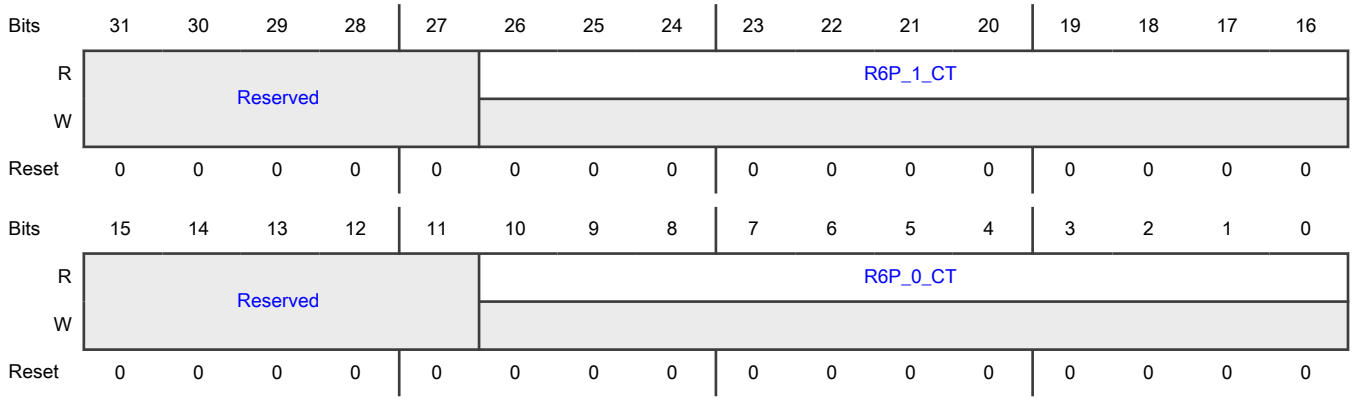
Offset

Register	Offset	Description
SCR6PC	38h	Accessible at this address when MCTL[PRGM] = 0

Function

The 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 SCR6PL[RUN6P_MAX]. The R6P_1_CT decrements each time six or more consecutive ones are sampled (preceded by a zero and followed by a zero). The R6P_0_CT decrements each time six or more consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this offset (0x38) is used as SCR6PL if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this offset is used as SCR6PC readback register, as described here.

Diagram



Fields

Field	Function
31-27 —	Reserved
26-16 R6P_1_CT	Runs of One, Length 6+ Count. Reads the final Runs of Ones, length 6+ count after entropy generation. Requires MCTL[PRGM] = 0.
15-11 —	Reserved
10-0 R6P_0_CT	Runs of Zero, Length 6+ Count. Reads the final Runs of Ones, length 6+ count after entropy generation. Requires MCTL[PRGM] = 0.

17.7.27 Statistical Check Run Length 6+ Limit Register (SCR6PL)

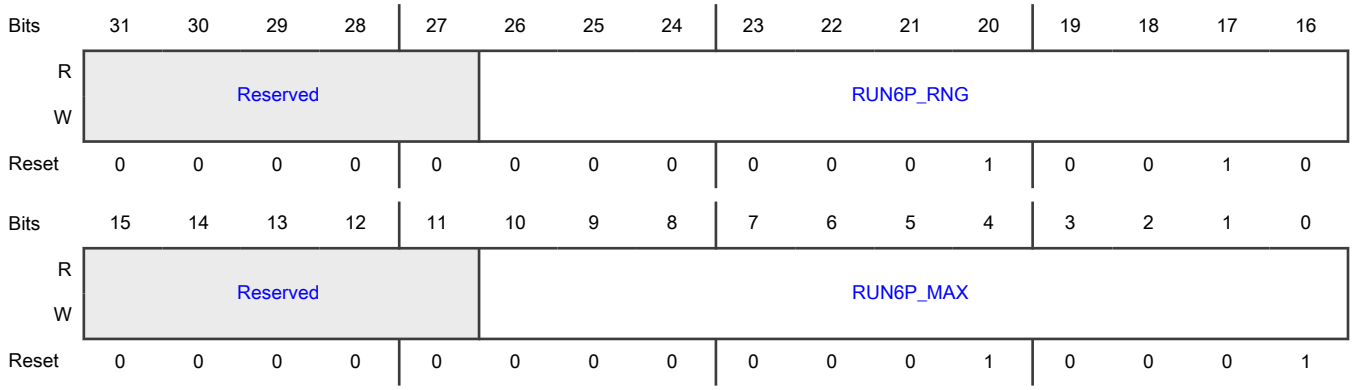
Offset

Register	Offset	Description
SCR6PL	38h	Accessible at this address when MCTL[PRGM] = 1

Function

The 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 (RUN6P_MAX - RUN6P_RNG). If this test fails, the Retry Counter in SCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this offset (0x38) is used as SCR6PL only if MCTL[PRGM] is 1. If MCTL[PRGM] is 0, this offset is used as SCR6PC readback register.

Diagram



Fields

Field	Function
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 the reset value by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0x7FF in this field.
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 reset by writing the MCTL[RST_DEF] bit to 1. Writing 1 to bit MCTL[DIS_SLF_TST] loads a value of 0x7FE in this field.

17.7.28 Status Register (STATUS)

Offset

Register	Offset
STATUS	3Ch

Function

Various statistical tests are run as a normal part of the TRNG's entropy generation process. The least-significant 16 bits of the STATUS register reflect the result of each of these tests. The status of these bits will be valid when the TRNG has finished its entropy generation process. Software can determine when this occurs by polling MCTL[ENT_VAL].

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_CT 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 TRNG. By default, RETRY_CT is initialized to 1, but software can increase the retry count by writing to SCMISC[RTY_CT].

All 0s will be returned if this register address is read while the TRNG is in Program Mode (MCTL[PRGM] = 1). If this register is read while the TRNG is in Run Mode (MCTL[PRGM] = 0), the value returned will be formatted as follows.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved												RETRY_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	TFMB	TFP	TFLR	TFSB	TF6PBR1	TF6PBR0	TF5BR1	TF5BR0	TF4BR1	TF4BR0	TF3BR1	TF3BR0	TF2BR1	TF2BR0	TF1BR1	TF1BR0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-20 —	Reserved
19-16 RETRY_CT	RETRY COUNT. This represents the current number of entropy generation retries left before a statistical test failure will cause the TRNG to generate an error condition.
15 TFMB	Test Fail, Mono Bit. 0b - The Mono Bit Test has passed 1b - The Mono Bit Test has failed
14 TFP	Test Fail, Poker. 0b - The Poker Test has passed 1b - The Poker Test has failed
13 TFLR	Test Fail, Long Run. 0b - The Long Run Test has passed 1b - The Long Run Test has failed
12 TFSB	Test Fail, Sparse Bit. 0b - The Sparse Bit Test has passed 1b - The Sparse Bit Test has failed
11	Test Fail, 6 Plus Bit Run, Sampling 1s.

Table continues on the next page...

Table continued from the previous page...

Field	Function
TF6PBR1	0b - The 6 Plus Bit Run, Sampling 1s Test has passed 1b - The 6 Plus Bit Run, Sampling 1s Test has failed
10 TF6PBR0	Test Fail, 6 Plus Bit Run, Sampling 0s. 0b - The 6 Plus Bit Run, Sampling 0s Test has passed 1b - the 6 Plus Bit Run, Sampling 0s Test has failed
9 TF5BR1	Test Fail, 5-Bit Run, Sampling 1s. If TF5BR1=1, the 5-Bit Run, Sampling 1s Test has failed. 0b - The 5-Bit Run, Sampling 1s Test has passed 1b - The 5-Bit Run, Sampling 1s Test has failed
8 TF5BR0	Test Fail, 5-Bit Run, Sampling 0s. 0b - The 5-Bit Run, Sampling 0s Test has passed 1b - The 5-Bit Run, Sampling 0s Test has failed
7 TF4BR1	Test Fail, 4-Bit Run, Sampling 1s. 0b - The 4-Bit Run, Sampling 1s Test has passed 1b - The 4-Bit Run, Sampling 1s Test has failed
6 TF4BR0	Test Fail, 4-Bit Run, Sampling 0s 0b - The 4-Bit Run, Sampling 0s Test has passed 1b - The 4-Bit Run, Sampling 0s Test has failed
5 TF3BR1	Test Fail Test Fail, 3-Bit Run, Sampling 1s. 0b - The 3-Bit Run, Sampling 1s Test has passed 1b - The 3-Bit Run, Sampling 1s Test has failed
4 TF3BR0	Test Fail, 3-Bit Run, Sampling 0s. 0b - The 3-Bit Run, Sampling 0s Test has passed 1b - The 3-Bit Run, Sampling 0s Test has failed
3 TF2BR1	Test Fail, 2-Bit Run, Sampling 1s. 0b - The 2-Bit Run, Sampling 1s Test has passed 1b - The 2-Bit Run, Sampling 1s Test has failed
2 TF2BR0	Test Fail, 2-Bit Run, Sampling 0s. 0b - The 2-Bit Run, Sampling 0s Test has passed 1b - The 2-Bit Run, Sampling 0s Test has failed

Table continues on the next page...

Table continued from the previous page...

Field	Function
1 TF1BR1	Test Fail, 1-Bit Run, Sampling 1s. 0b - The 1-Bit Run, Sampling 1s Test has passed 1b - The 1-Bit Run, Sampling 1s Test has failed
0 TF1BR0	Test Fail, 1-Bit Run, Sampling 0s. 0b - The 1-Bit Run, Sampling 0s Test has passed 1b - The 1-Bit Run, Sampling 0s Test has failed

17.7.29 Entropy Read Register (ENT0 - ENT7)

Offset

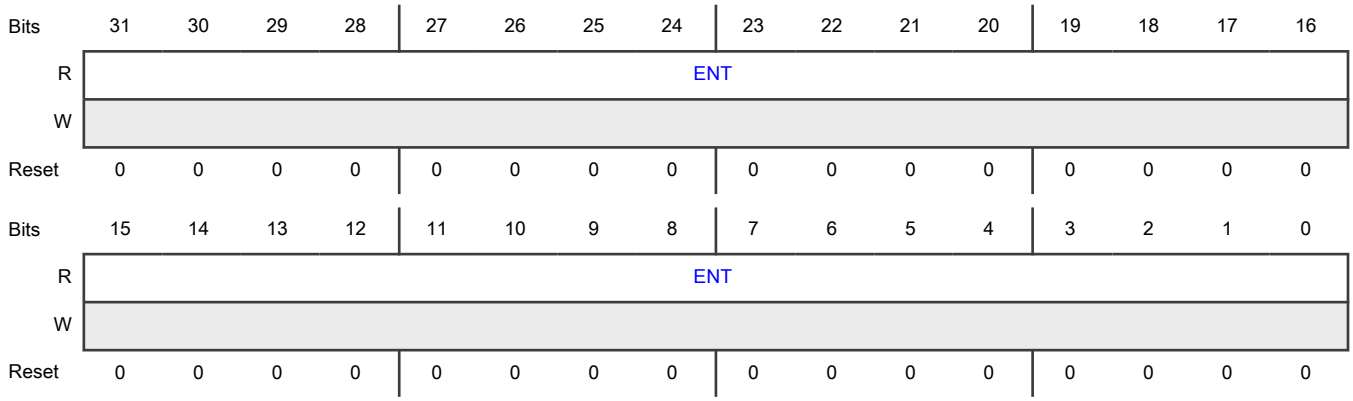
For a = 0 to 7:

Register	Offset	Description
ENTa	40h + (a × 4h)	Word a

Function

The TRNG can be programmed to generate an entropy value that is readable via the register interface. To do this, set the MCTL[TRNG_ACC] bit to 1. Once the entropy value has been generated, the MCTL[ENT_VAL] bit will be set to 1. At this point, ENT0 through ENT7 may be read to retrieve the 256-bit entropy value. Note that once ENT7 is read, the entropy value will be cleared and a new value will begin generation, so it is important that ENT7 be read last. These registers are readable only when MCTL[PRGM] = 0 (Run Mode) , MCTL[TRNG_ACC] = 1 (TRNG access mode) and MCTL[ENT_VAL] = 1. After at most one (1) bus clock cycle of reading a valid ENT7 register value, reading any ENT0 through ENT7 register will return zeros.

Diagram



Fields

Field	Function
31-0 ENT	Entropy Value. Will be non-zero only if MCTL[PRGM] = 0 (Run Mode) and MCTL[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.

17.7.30 Statistical Check Poker Count 1 and 0 Register (PKRCNT10)

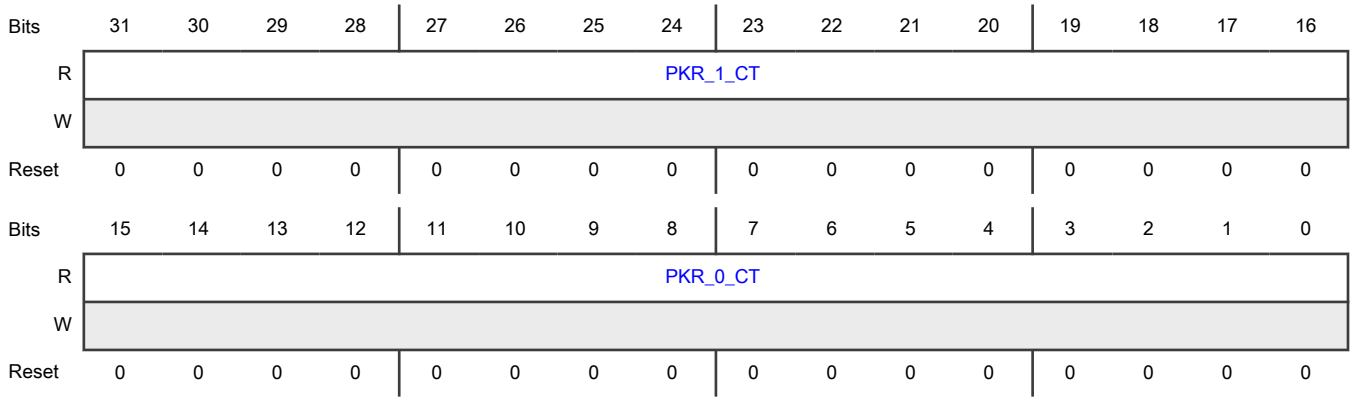
Offset

Register	Offset
PKRCNT10	80h

Function

The 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 MCTL[PRGM] is 0, otherwise zeroes will be read.

Diagram



Fields

Field	Function
31-16 PKR_1_CT	Poker 1h Count. Total number of nibbles of sample data which were found to be 1h. Requires MCTL[PRGM] = 0.
15-0 PKR_0_CT	Poker 0h Count. Total number of nibbles of sample data which were found to be 0h. Requires MCTL[PRGM] = 0.

17.7.31 Statistical Check Poker Count 3 and 2 Register (PKRCNT32)

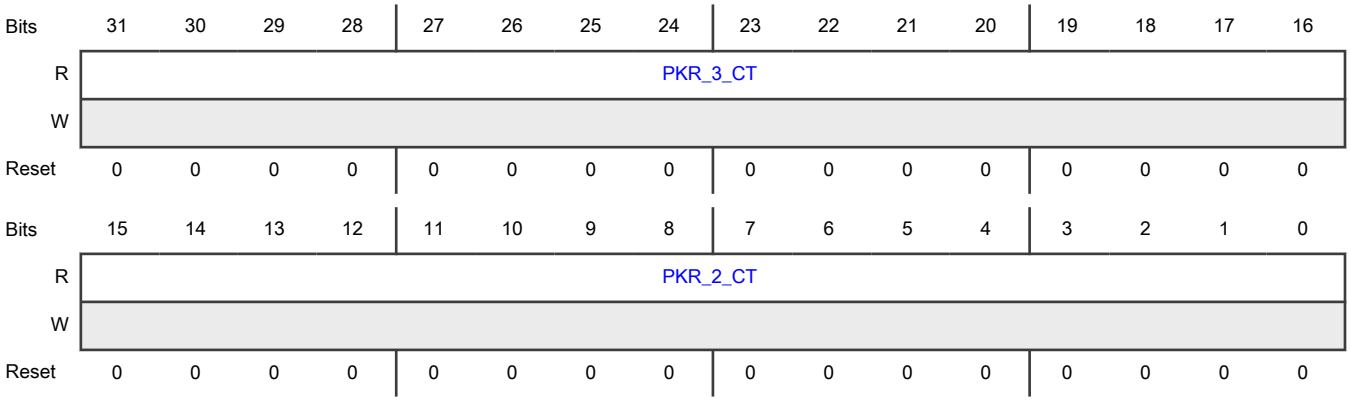
Offset

Register	Offset
PKRCNT32	84h

Function

The 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 MCTL[PRGM] is 0, otherwise zeroes will be read.

Diagram



Fields

Field	Function
31-16 PKR_3_CT	Poker 3h Count. Total number of nibbles of sample data which were found to be 3h. Requires MCTL[PRGM] = 0.
15-0 PKR_2_CT	Poker 2h Count. Total number of nibbles of sample data which were found to be 2h. Requires MCTL[PRGM] = 0.

17.7.32 Statistical Check Poker Count 5 and 4 Register (PKRCNT54)

Offset

Register	Offset
PKRCNT54	88h

Function

The 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 MCTL[PRGM] is 0, otherwise zeroes will be read.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PKR_5_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	PKR_4_CT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-16 PKR_5_CT	Poker 5h Count. Total number of nibbles of sample data which were found to be 5h. Requires MCTL[PRGM] = 0.
15-0 PKR_4_CT	Poker 4h Count. Total number of nibbles of sample data which were found to be 4h. Requires MCTL[PRGM] = 0.

17.7.33 Statistical Check Poker Count 7 and 6 Register (PKRCNT76)

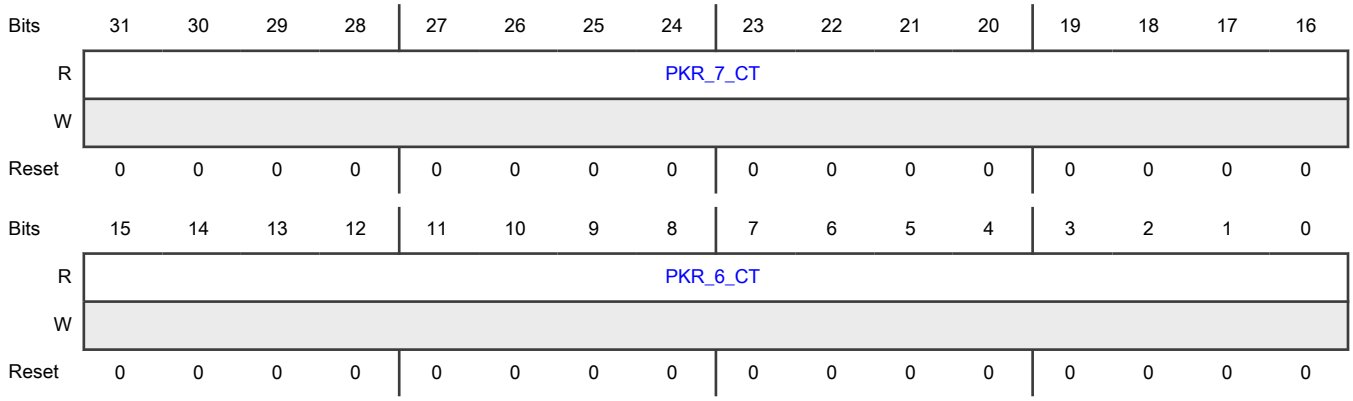
Offset

Register	Offset
PKRCNT76	8Ch

Function

The 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 MCTL[PRGM] is 0, otherwise zeroes will be read.

Diagram



Fields

Field	Function
31-16 PKR_7_CT	Poker 7h Count. Total number of nibbles of sample data which were found to be 7h. Requires MCTL[PRGM] = 0.
15-0 PKR_6_CT	Poker 6h Count. Total number of nibbles of sample data which were found to be 6h. Requires MCTL[PRGM] = 0.

17.7.34 Statistical Check Poker Count 9 and 8 Register (PKRCNT98)

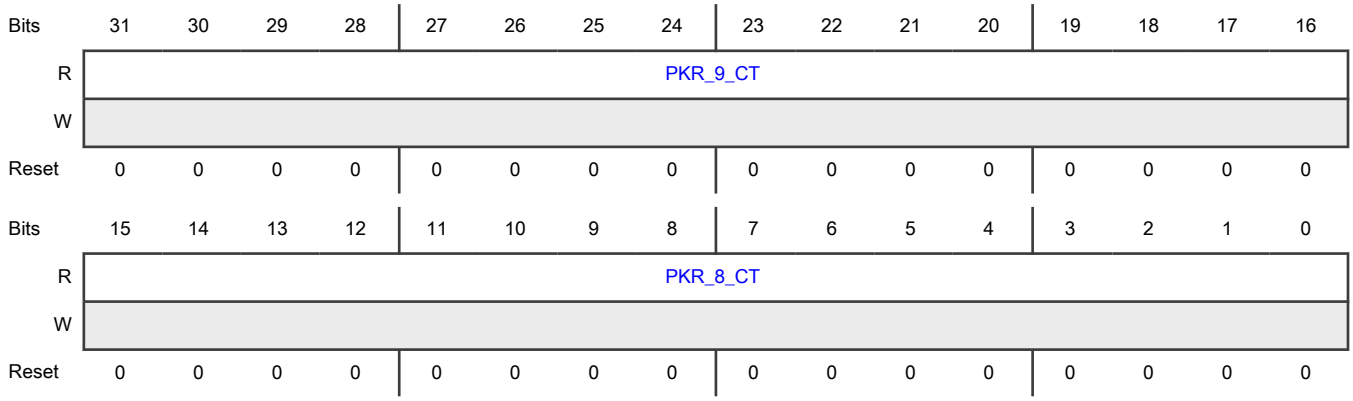
Offset

Register	Offset
PKRCNT98	90h

Function

The 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 MCTL[PRGM] is 0, otherwise zeroes will be read.

Diagram



Fields

Field	Function
31-16 PKR_9_CT	Poker 9h Count Total number of nibbles of sample data which were found to be 9h. Requires MCTL[PRGM] = 0.
15-0 PKR_8_CT	Poker 8h Count Total number of nibbles of sample data which were found to be 8h. Requires MCTL[PRGM] = 0.

17.7.35 Statistical Check Poker Count B and A Register (PKRCNTBA)

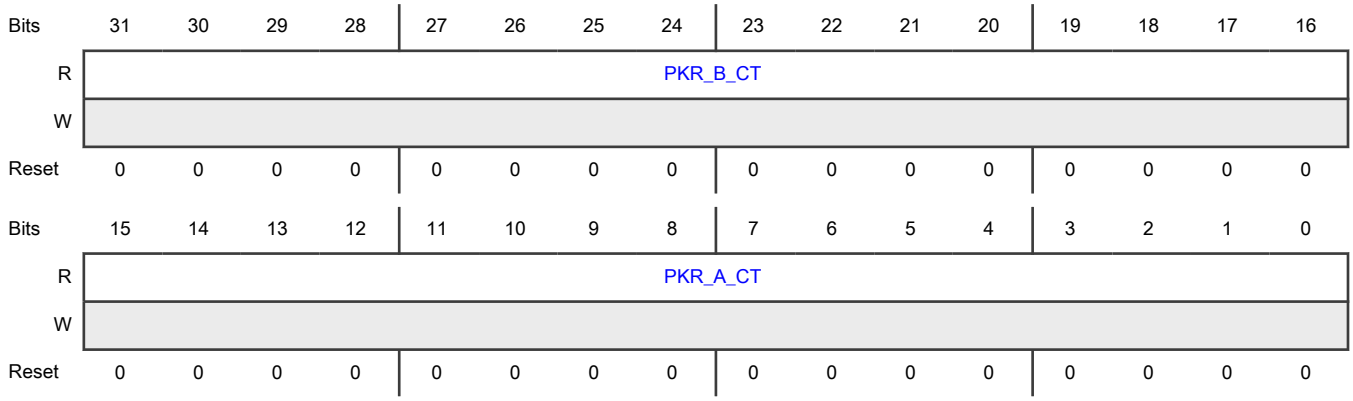
Offset

Register	Offset
PKRCNTBA	94h

Function

The 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 MCTL[PRGM] is 0, otherwise zeroes will be read.

Diagram



Fields

Field	Function
31-16 PKR_B_CT	Poker Bh Count Total number of nibbles of sample data which were found to be Bh. Requires MCTL[PRGM] = 0.
15-0 PKR_A_CT	Poker Ah Count Total number of nibbles of sample data which were found to be Ah. Requires MCTL[PRGM] = 0.

17.7.36 Statistical Check Poker Count D and C Register (PKRCNTDC)

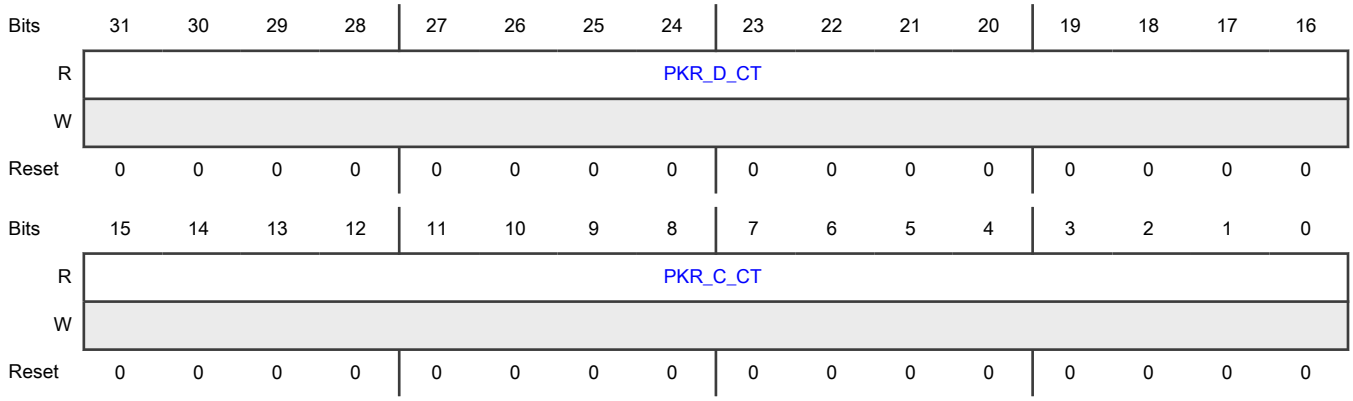
Offset

Register	Offset
PKRCNTDC	98h

Function

The 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 MCTL[PRGM] is 0, otherwise zeroes will be read.

Diagram



Fields

Field	Function
31-16 PKR_D_CT	Poker Dh Count Total number of nibbles of sample data which were found to be Dh. Requires MCTL[PRGM] = 0.
15-0 PKR_C_CT	Poker Ch Count Total number of nibbles of sample data which were found to be Ch. Requires MCTL[PRGM] = 0.

17.7.37 Statistical Check Poker Count F and E Register (PKRCNTFE)

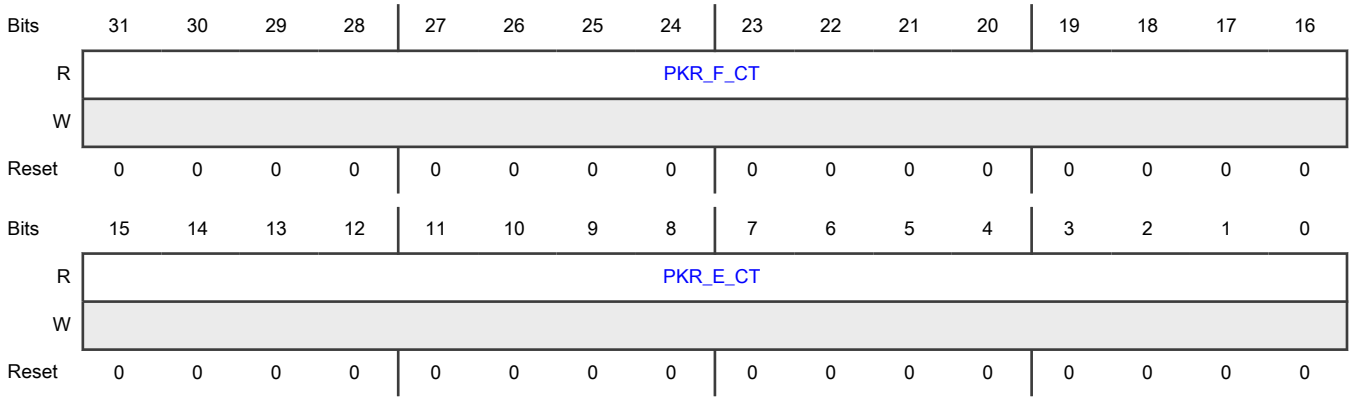
Offset

Register	Offset
PKRCNTFE	9Ch

Function

The 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 MCTL[PRGM] is 0, otherwise zeroes will be read.

Diagram



Fields

Field	Function
31-16 PKR_F_CT	Poker Fh Count. Total number of nibbles of sample data which were found to be Fh. Requires MCTL[PRGM] = 0.
15-0 PKR_E_CT	Poker Eh Count. Total number of nibbles of sample data which were found to be Eh. Requires MCTL[PRGM] = 0.

17.7.38 Security Configuration Register (SEC_CFG)

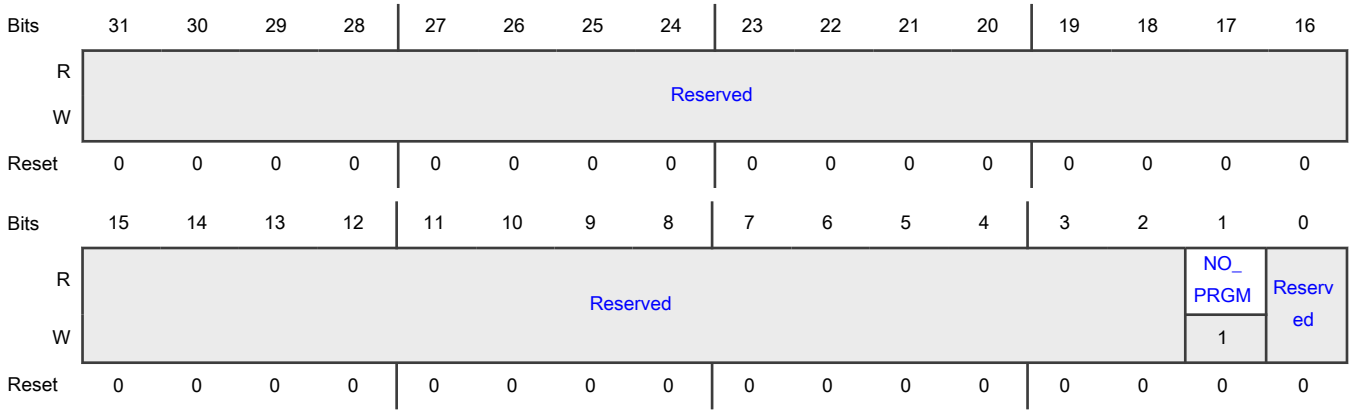
Offset

Register	Offset
SEC_CFG	A0h

Function

The Security Configuration Register used to control global configuration of TRNG.

Diagram



Fields

Field	Function
31-2 —	Reserved
1 NO_PRGM	If set, below mentioned TRNG configuration registers cannot be programmed: • Oscillator 2 Control Register (OSC2_CTL): — TRNG Entropy Generation Control [1:0] — Oscillator 2 Divider [3:2] — Oscillator Fail Safe Limit [13:12] — Oscillator Fail Safe Test [14] • TRNG Seed Control Register (SDCTL) • TRNG Frequency Count Minimum Limit Register (FRQMIN) • TRNG Frequency Count Maximum Limit Register (FRQMAX) • TRNG Statistical Check Monobit Limit Register (SCML) • TRNG Statistical Check Run Length 1 Limit Register (SCR1L) • TRNG Statistical Check Run Length 2 Limit Register (SCR2L) • TRNG Statistical Check Run Length 3 Limit Register (SCR3L) • TRNG Miscellaneous Control Register (MCTL): — Sample Mode [1:0] — Oscillator Divider [3:2] — Reset Defaults [6] — Force System Clock [7] — Long Runs Continuation Mode [14] After this bit has been written to a 1, it cannot be changed. The bit is reset to 0 by an asynchronous reset. 0b - TRNG configuration registers can be modified. 1b - TRNG configuration registers cannot be modified.
0 —	Reserved

17.7.39 Interrupt Control Register (INT_CTRL)

Offset

Register	Offset
INT_CTRL	A4h

Function

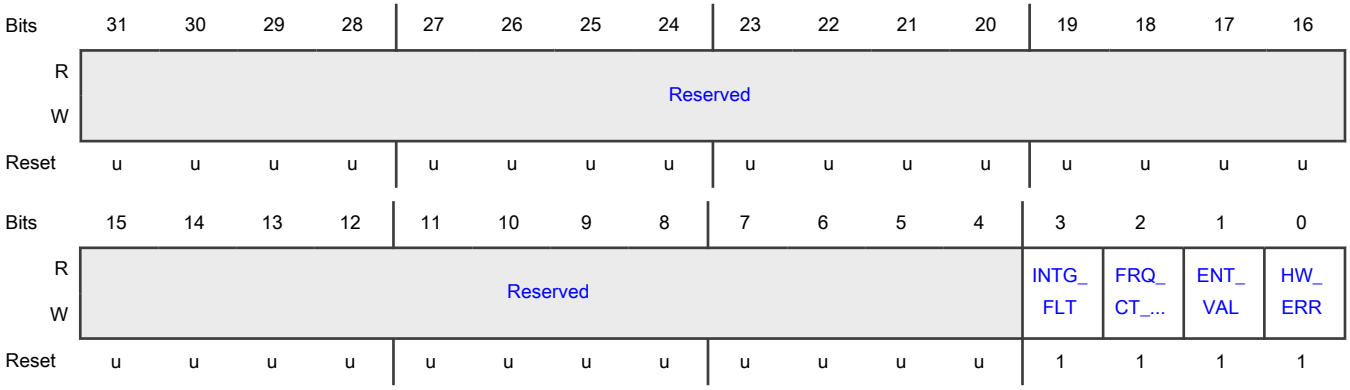
The Interrupt Control Register is a read/write register used to clear the status for the four interrupts generated by the TRNG. See INT_STATUS register description.

The purpose of the INT_CTRL register is to allow a pending interrupt to be cleared. To perform this function the specific interrupt bit is written 0. At this time the INT_STATUS bit for the corresponding interrupt, and the intterupt output (if no other interrupts were pending) will de-assert.

The INT_CTRL bit for the interrupt will remain 0 as long as the interrupt source that caused the interrupt remains asserted. Once the source is deasserted the INT_CRTL bit will return 1 and remain there for the next interrupt.

Writing 1 can NOT cause generation of interrupts, or have any other effect.

Diagram



Fields

Field	Function
31-4 —	Reserved
3 INTG_FLT	Clear the INTG_FLT interrupt. 0b - Clears the INT_STATUS[INTG_FLT] bit. Will automatically set after writing. 1b - Enables the INT_STATUS[INTG_FLT] bit to be set, thereby enabling interrupt generation for the INTG_FLT condition.
2 FRQ_CT_FAIL	Clear the FRQ_CT_FAIL interrupt. 0b - Clears the INT_STATUS[FRQ_CT_FAIL] bit. Will automatically set after writing. 1b - Enables the INT_STATUS[FRQ_CT_FAIL] bit to be set, thereby enabling interrupt generation for the FRQ_CT_FAIL condition.
1 ENT_VAL	Clear the ENT_VAL interrupt. 0b - Clears the INT_STATUS[ENT_VAL] bit. Will automatically set after writing. 1b - Enables the INT_STATUS[ENT_VAL] bit to be set, thereby enabling interrupt generation for the ENT_VAL condition.
0	Clear the HW_ERR interrupt.

Table continues on the next page...

Table continued from the previous page...

Field	Function
HW_ERR	0b - Clears the INT_STATUS[HW_ERR] bit. Will automatically set after writing. 1b - Enables the INT_STATUS[HW_ERR] bit to be set, thereby enabling interrupt generation for the HW_ERR condition.

17.7.40 Mask Register (INT_MASK)

Offset

Register	Offset
INT_MASK	A8h

Function

The Interrupt Mask Register (actually an enable register) is a read/write register used to enable the status reporting of the four interrupts generated by the TRNG. See INT_STATUS register description. Each interrupt can be masked/disabled by clearing the corresponding bit in the INT_MASK register. Only setting this bit high will re-enable the interrupt in the status register. Even if the interrupt is cleared or masked, interrupt status information can be read from the MCTL register.

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												INTG_FLT	FRQ_CT_...	ENT_VAL	HW_ERR
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-4 —	Reserved
3 INTG_FLT	Mask the INTG_FLT interrupt. 0b - INTG_FLT interrupt is disabled. 1b - INTG_FLT interrupt is enabled.

Table continues on the next page...

Table continued from the previous page...

Field	Function
2 FRQ_CT_FAIL	Mask the FRQ_CT_FAIL interrupt. 0b - FRQ_CT_FAIL interrupt is disabled. 1b - FRQ_CT_FAIL interrupt is enabled.
1 ENT_VAL	Mask the ENT_VAL interrupt. 0b - ENT_VAL interrupt is disabled. 1b - ENT_VAL interrupt is enabled.
0 HW_ERR	Mask the HW_ERR interrupt. 0b - HW_ERR interrupt is disabled. 1b - HW_ERR interrupt is enabled.

17.7.41 Interrupt Status Register (INT_STATUS)

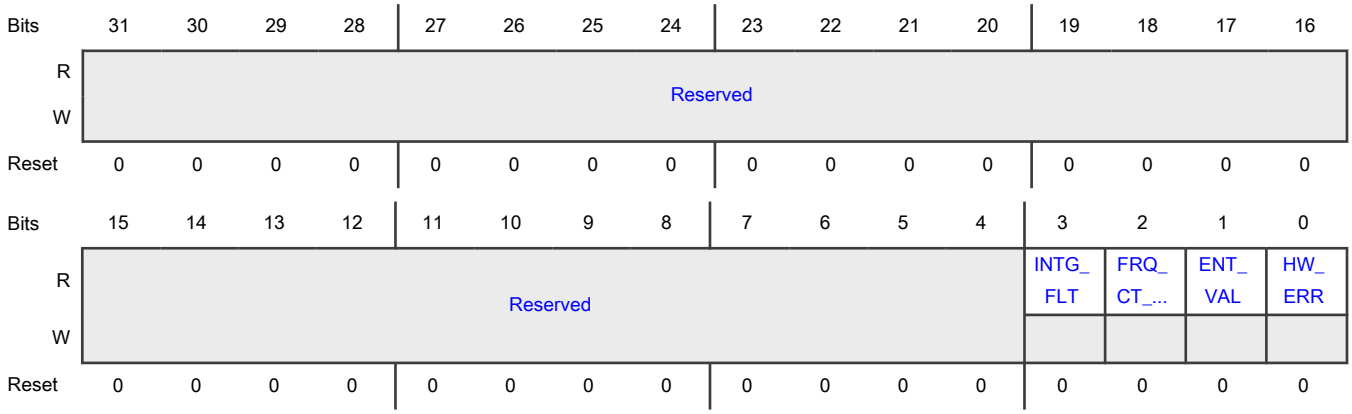
Offset

Register	Offset
INT_STATUS	ACh

Function

The Interrupt Status Register is used to provide status for the three interrupts generated by the TRNG. These interrupts can be used to provide alert when either Frequency Count has failed or Entropy is Valid or an HW Error has occurred. Each interrupt can be temporarily cleared by writing 0 to the corresponding bit in the INT_CTRL register. To mask the interrupts, write 0 to the corresponding bits in the INT_MASK register. The description of the interrupts is defined in the MCTL register description. Even if the interrupt is cleared or masked, interrupt status information can be read from the MCTL register.

Diagram



Fields

Field	Function
31-4 —	Reserved
3 INTG_FLT	Integrity Fault. An internal fault has occurred. 0b - No internal fault has been detected. 1b - TRNG has detected internal fault.
2 FRQ_CT_FAIL	Frequency Count Fail. The frequency counter has detected a failure. This may be due to improper programming of the FRQMAX and/or FRQMIN registers, or a hardware failure in the ring oscillator. 0b - No hardware nor self test frequency errors. 1b - The frequency counter has detected a failure.
1 ENT_VAL	Entropy Valid. Will assert after an entropy value is generated and only if TRNG_ACC bit is set. Will be cleared when ENT7 is read. (ENT0 through ENT6 should be read before reading ENT7). 0b - Busy generating entropy. Any value read from the Entropy registers is invalid. 1b - Values read from the Entropy registers are valid.
0 HW_ERR	Read: TRNG Error. Any error in the TRNG will trigger this interrupt. 0b - No error. 1b - Error detected.

17.7.42 Common Security Error Register (CSER)

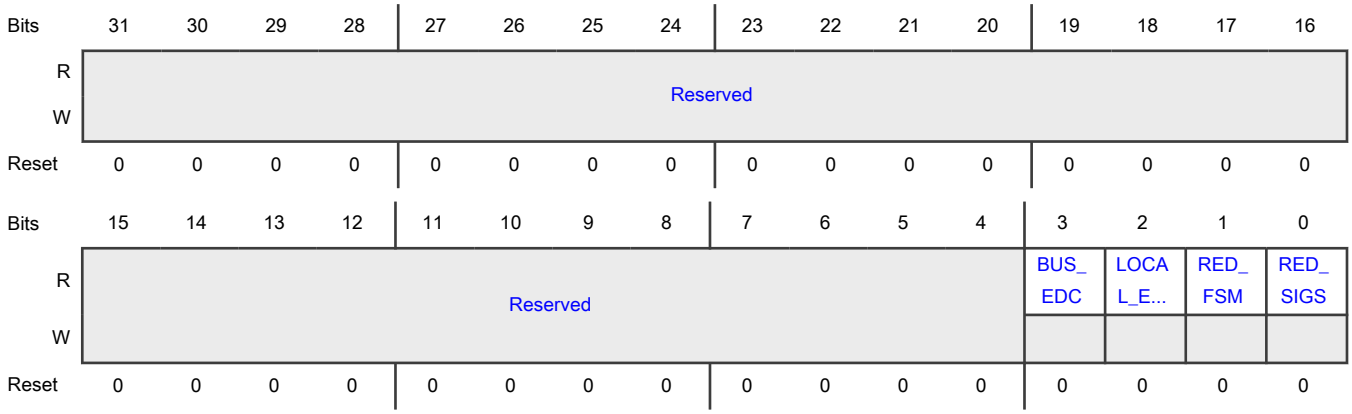
Offset

Register	Offset
CSER	B0h

Function

The Common Security Error Register is used to provide status for the Integrity Errors/Faults (H/W countermeasure) detected by TRNG. These faults get detected due to bit error occurring inside TRNG caused as a result of an external attack on the TRNG h/w. Whenever any bit of this register gets set, it causes TRNG to report a high-severity event to a central event handler. Each Integrity Fault can be temporarily cleared by writing 0 to the corresponding bit in the CSCLR register.

Diagram



Fields

Field	Function
31-4 —	Reserved
3 BUS_EDC	Bus-EDC error/fault detected This error/fault gets set whenever the EDC received via register/bus interface doesn't match EDC calculated on the write data received on the register interface. 0b - No Bus-EDC error/fault detected. 1b - Bus-EDC error/fault detected.
2 LOCAL_EDC	Local-EDC error/fault detected This error/fault gets set whenever the continuously-generated Local-EDC of a register deemed security critical doesn't match the stored-EDC that was generated at the time the corresponding register was updated. 0b - No Local-EDC error/fault detected. 1b - Local-EDC error/fault detected.
1 RED_FSM	Redundant FSM error/fault detected This error/fault gets set whenever, the current state of an FSM and its redundant state values are not same. Writing 1 to MCTL[ERR] followed by writing 1 to CSCLR[RED_FSM] will clear the fault. 0b - No redundant FSM error/fault 1b - Redundant FSM error/fault detected.
0 RED_SIGS	Redundant Signals error/fault Detected This error/fault gets set when true and redundant-inverted version of input port is found to be in same logical state. 0b - No redundant signal error/fault 1b - Redundant signal error/fault detected.

17.7.43 Common Security Clear Register (CSCLR)

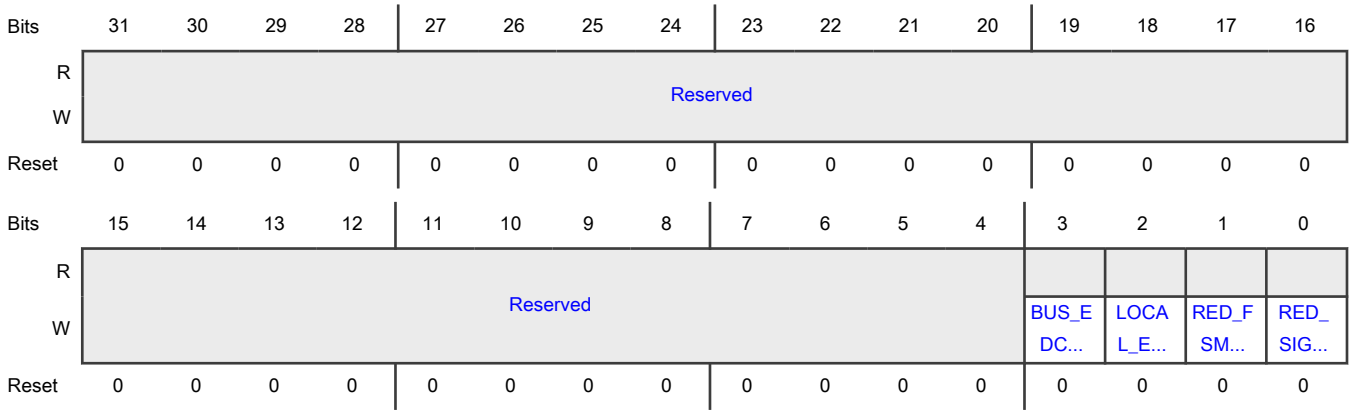
Offset

Register	Offset
CSCLR	B4h

Function

The Common Security Clear Register is used to clear the Integrity Errors/Faults (H/W countermeasure) reported via CSER register of TRNG. The act of writing a '1' shall have the effect of temporarily clearing the corresponding bit in the SEH_CSER. A write of '0' shall be ignored. A read of this register shall result in a Bus error.

Diagram



Fields

Field	Function
31-4 —	Reserved
3 BUS_EDC_CLR	Read only: Bus-EDC error/fault detected. This error/fault gets set whenever the received bus EDC doesn't match to EDC generated on the received data on register interface. 0b - No effect, ignored 1b - Clears the CSER[BUS_EDC] bit.
2 LOCAL_EDC_CLR	Read only: Local-EDC error/fault detected. This error/fault gets set whenever the EDC received via register/bus interface doesn't match EDC calculated on the write data received on the register interface. 0b - No effect, ignored 1b - Clears the CSER[LOCAL_EDC] bit.
1 RED_FSM_CLR	Read only: Redundant FSM error/fault detected. This error/fault gets set whenever, the current state of an FSM and its redundant state values are not same.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - No effect, ignored 1b - Clears the CSER[RED_FSM] bit.
0 RED_SIGS_CLR	Redundant Signals error/fault Detected Read Only: This error/fault gets set when true and redundant-inverted version of input port is found to be in same logical state. 0b - No effect, ignored 1b - Clears the CSER[RED_SIGS] bit.

17.7.44 TRNG Oscillator 2 Control Register (OSC2_CTL)

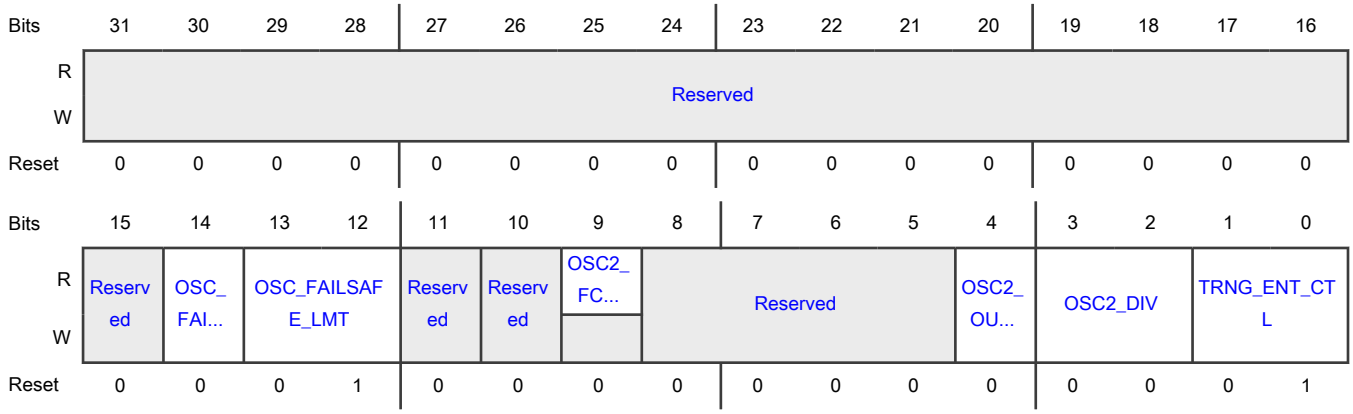
Offset

Register	Offset
OSC2_CTL	ECh

Function

The TRNG Oscillator 2 Control Register controls various functions related to the second free-running oscillator in the TRNG. OSC2_CTL is only write-able if MCTL[PRGM]=1.

Diagram



Fields

Field	Function
31-15 —	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
14 OSC_FAILSAFE_TEST	Oscillator fail safe test. This is used only for testing the OSC_FAILSAFE_LMT logic. This field is writable only if MCTL[PRGM] bit is 1. This field is reset by writing the RST_DEF bit to 1. 0b - No impact. 1b - Disables oscillator 2 while in dual-oscillator mode (TRNG_ENT_CTL = 01b).
13-12 OSC_FAILSAFE_LMT	Oscillator fail safe limit. Sets the failsafe limit for determining oscillator failure. This field is writable only if MCTL[PRGM] bit is 1. This field is reset by writing the RST_DEF bit to 1. In dual-oscillator mode (TRNG_ENT_CTL = 01b), if oscillator 2 is not running, both MCTL[FCT_FAIL] and MCTL[OSC2_FAIL] will be set after N system clocks, as follows: 00b - The limit N is 4096 (2^12) system clocks. 01b - The limit N is 65536 (2^16) system clocks. (default) 10b - N is 2^20 system clocks. 11b - N is 2^22 system clocks (full range of the counter being used).
11 —	Reserved
10 —	Reserved
9 OSC2_FCT_VAL	TRNG Oscillator 2 Frequency Count Valid 0b - Frequency count is invalid. 1b - If TRNG_ENT_CTL = 10b, valid frequency count may be read from OSC2_FRQCNT.
8-5 —	Reserved
4 OSC2_OUT_EN	Oscillator 2 Clock Output Enable This field is reset by writing the RST_DEF bit to 1. 0b - Ring oscillator 2 output is gated to an output pad. 1b - Allows external viewing of divided-by-2 ring oscillator 2 if MCTL[PRGM] = 1 mode is also selected, else ring oscillator 2 output is gated to an output pad.
3-2 OSC2_DIV	Oscillator 2 Divide. Determines the amount of dividing done to the ring oscillator 2 before it is used by the TRNG. This field is writable only if MCTL[PRGM] is 1. This field is reset by writing the RST_DEF bit to 1. 00b - Use ring oscillator 2 with no divide 01b - Use ring oscillator 2 divided-by-2

Table continues on the next page...

Table continued from the previous page...

Field	Function
	10b - Use ring oscillator 2 divided-by-4 11b - Use ring oscillator 2 divided-by-8
1-0 TRNG_ENT_CTL	TRNG entropy generation control. This field is writable only if MCTL[PRGM] is 1. This field is reset by writing the RST_DEF bit to 1. 00b - Single oscillator mode, using OSC1 (default) 01b - Dual oscillator mode 10b - Single oscillator mode, using OSC2 11b - Unused, (bit field cannot be written to this value)

17.7.45 Version ID Register (MS) (VID1)

Offset

Register	Offset
VID1	F0h

Function

The Version ID Register (VID1) is an MSW read only register used to identify the version of the TRNG in use. This register as well as VID2 should both be read to verify the expected version.

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	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MAJ_REV								MIN_REV							
W																
Reset	0	0	0	1	0	1	0	0	0	0	0	0	1	1	0	0

Fields

Field	Function
31-16	Shows the IP ID.

Table continues on the next page...

Table continued from the previous page...

Field	Function
IP_ID	0000_0000_0011_0000b - ID for TRNG.
15-8 MAJ_REV	Shows the IP's Major revision of the TRNG. • A value of 00010000b indicates TRNG has Dual-Oscillators. • A value of 00000100b indicates TRNG with 256-bit entropy size. • A value of 00000010b indicates TRNG with 128-bit entropy size. • A value of 00000001b indicates TRNG with 512-bit entropy size. 0001_0100b - Major revision number for TRNG.
7-0 MIN_REV	Shows the IP's Minor revision of the TRNG. 0000_1100b - Minor revision number for TRNG.

17.7.46 Version ID Register (LS) (VID2)

Offset

Register	Offset
VID2	F4h

Function

The Version ID Register (VID2) is an LSW read only register used to identify the architecture of the TRNG in use. This register as well as VID1 should both be read to verify the expected version.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ERA								INTG_OPT							
W																
Reset	0	0	0	0	1	1	0	0	0	0	0	0	1	0	1	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

Fields

Field	Function
31-24 ERA	Shows the ERA of the TRNG. 0000_1100b - ERA of the TRNG.
23-16 INTG_OPT	Shows the integration options for the TRNG. 0000_1010b - INTG_OPT for TRNG.
15-8 ECO_REV	Shows the IP's ECO revision of the TRNG. 0000_0000b - TRNG_ECO_REV for TRNG.
7-0 CONFIG_OPT	Shows the IP's Configuaration options for the TRNG. 0000_0000b - TRNG_CONFIG_OPT for TRNG.

Appendix A
Release notes

A.1 About This Manual changes

No substantial content changes.

A.2 Introduction changes

- Overview, added frequency 240 MHz.
- Figure 5
  - Core: Added 240 MHz
  - Memories: Added "Flash w/ Secure Installer "
  - HMI: Added "Segment LCD 4 x 44"
  - Connectivity: Changed from "Up to 4x LPI2C" to "4x LPI2C", "Up to 5xLPUART" to "5x or 6x LPUART ", "CAN-FD" to "1x or 2x CAN -FD".
  - Advanced Motor Control: Changed from "Up to 2xFlexPWM" to "2x FlexPWM (4x submodule)", "Up to 2x eQDC" to "2x eQDC", "Up to 2x AOI" to "2x AOI"
- Features
  - Changed from "180 MHz at Over Drive mode (1.2 V)" to "180/240 MHz at Over Drive mode (1.2 V)"
  - Removed "MAU (Math Accelerator Unit), supports trigonometric, reciprocal, square, square root, sine, cosine and arctan algorithms"
  - Changed from "4 wait states at 180 MHz (OD mode), 1 wait state at 45 MHz (MD mode)" to "5 wait states at 240 MHz (OD mode), 4 wait states at 180 MHz (OD mode), 1 wait state at 45 MHz (MD mode)"
  - Revised "Security"
  - Revised "Communication Interfaces for Connectivity"
  - Added HMI

A.3 Core overview changes

No substantial content changes.

A.4 Security Overview changes

No substantial content changes.

A.5 Life Cycle States chapter changes

- Updated Table 17.

A.6 ROM API chapter changes

No substantial content changes.

A.7 ISP chapter changes

No substantial content changes.

A.8 Boot ROM changes

No substantial content changes.

A.9 Secure Installer chapter changes

No substantial content changes.

A.10 Debug Mailbox (DBGMB)

A.10.1 Debug Mailbox bridge topic changes

No substantial content changes.

A.10.2 DBGMB module changes

No substantial content changes.

A.11 System Controller (SYSCON)

A.11.1 SYSCON bridge topic changes

No substantial content changes.

A.11.2 SYSCON module changes

No substantial content changes.

A.12 Code Watchdog Timer (CDOG)

A.12.1 CDOG bridge topic changes

No substantial content changes.

A.12.2 CDOG module changes

No substantial content changes.

A.13 Digital Tamper (TDET)

A.13.1 TDET bridge topic changes

No substantial content changes.

A.13.2 Digital Tamper Detect (TDET) module changes

No substantial content changes.

A.14 GLIKEY

A.14.1 GLIKEY bridge topic changes

No substantial content changes.

A.14.2 GLIKEY module changes

No substantial content changes.

A.15 Secure Generic Interface (SGI)

A.15.1 SGI bridge topic changes

No substantial content changes.

A.15.2 SGI module changes

No substantial content changes.

A.16 Public-Key Crypto Coprocessor (PKC)

A.16.1 PKC bridge topic changes

No substantial content changes.

A.16.2 PKC module changes

No substantial content changes.

A.17 True Random Generator (TRNG)

A.17.1 TRNG bridge topic changes

No substantial content changes.

A.17.2 TRNG module changes

No substantial content changes.

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

EdgeLock — is a trademark of NXP B.V.

I2C-bus — logo is a trademark of NXP B.V.

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

For more information, please visit: https://www.nxp.com

Date of release: 17 November 2025
Document identifier: MCXAP144M240F61SRM