Public Version

CC2538 System-on-Chip Solution for 2.4-GHz IEEE 802.15.4 and ZigBee®/ZigBee IP® Applications

Texas Instruments CC2538™ Family of Products

Version C

User's Guide

Literature Number: SWRU319C
April 2012–Revised May 2013

WARNING: EXPORT NOTICE

Recipient agrees to not knowingly export or re-export, directly or indirectly, any product or technical data (as defined by the U.S., EU, and other Export Administration Regulations) including software, or any controlled product restricted by other applicable national regulations, received from Disclosing party under this Agreement, or any direct product of such technology, to any destination to which such export or re-export is restricted or prohibited by U.S. or other applicable laws, without obtaining prior authorisation from U.S. Department of Commerce and other competent Government authorities to the extent required by those laws. This provision shall survive termination or expiration of this Agreement.

According to our best knowledge of the state and end-use of this product or technology, and in compliance with the export control regulations of dual-use goods in force in the origin and exporting countries, this technology is classified as follows:

US ECCN: 5E002

EU ECCN: 5E002

And may require export or re-export license for shipping it in compliance with the applicable regulations of certain countries.

Contents

List of Figures

List of Tables

Read This First

About This Document

This user's guide provides information on how to use the CC2538 and describes the functional blocks of the system-on-chip (SoC) designed around the ARM® Cortex™-M3 core and an IEEE 802.15.4 radio.

Audience

This manual is intended for system software developers, hardware designers, and application developers.

About This Manual

This document is organized into sections that correspond to each major feature. This document does not contain performance characteristics of the device or modules. These are gathered in the corresponding device data sheet.

Related Documents

The following related documents are available on the CC2538 product page at www.ti.com:

- CC2538 Data Sheet
- CC2538 Errata
- CC2538 ROM User's Guide
- CC2538 Peripheral Driver Library User's Guide

The following related documents are also referenced:

- IEEE Standard 1149.1-Test Access Port and Boundary-Scan Architecture

This list of documents was current as of publication date. Check the web site for additional documentation, including application notes and white papers.

Document Conventions

Table 0-1. Document Conventions

Register	Meaning
General Register Notation	
Register	APB registers are indicated in uppercase bold. For example, **GPTIMER_CFG** configures the global operation of general purpose timers. If a register name contains a lowercase n, it represents more than one register. For example, **GPTIMER_TnMR** represents any of the timer A or B mode control registers: **GPTIMER_TAMR, GPTIMER_TBMR.**
Bit	A single bit in a register
Bit field	Two or more consecutive and related bits
Offset 0xnnn	A hexadecimal increment to the address of a register, relative to the base address of that module as specified in

Table 0-1. Document Conventions (continued)

Register	Meaning
Register N	Registers are numbered consecutively throughout the document to aid in referencing them. The register number has no meaning to software.
Reserved	Register bits marked reserved are reserved for future use. In most cases, reserved bits are set to 0; however, user software should not rely on the value of a reserved bit. To provide software compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
yy:xx	The range of register bits inclusive from xx to yy. For example, 31:15 means bits 15 through 31 in that register.
Register Bit or Field Types	This value in the register bit diagram indicates whether software running on the controller can change the value of the bit field.
RC	Software can read this field. The bit or field is cleared by hardware after reading the bit or field.
RO	Software can read this field. Always write the chip reset value.
R/W	Software can read or write this field.
R/WC	Software can read or write this field. Writing to it with any value clears the register.
R/W1C	Software can read or write this field. Writing 0 to a W1C bit does not affect the bit value in the register. Writing 1 clears the value of the bit in the register; the remaining bits remain unchanged. This register type is primarily used for clearing interrupt status bits where the read operation provides the interrupt status and the write of the read value clears only the interrupts being reported at the time the register was read.
R/W1S	Software can read or write 1 to this field. Writing 0 to a R/W1S bit does not affect the bit value in the register.
W1C	Software can write this field. Writing 0 to a W1C bit does not affect the bit value in the register. Writing 1 clears the value of the bit in the register; the remaining bits remain unchanged. Reading the register returns no meaningful data. This register is typically used to clear the corresponding bit in an interrupt register.
WO	Only a write by software is valid; a read of the register returns no meaningful data.
Register Bit or Field Reset Value	This value in the register bit diagram shows the bit or field value after any reset, unless noted.
1	Bit is cleared to 0 on chip reset.
0	Bit is set to 1 on chip reset.
–	Nondeterministic
Pin and Signal Notation	
[ ]	Pin alternate function; a pin defaults to the signal without the brackets.
Pin	Refers to the physical connection on the package.
Signal	Refers to the electrical signal encoding of a pin.
Assert a signal	Change the value of the signal from the logically False state to the logically True state. For active high signals, the asserted signal value is 1 (high); for active low signals, the asserted signal value is 0 (low). The active polarity (high or low) is defined by the signal name (see SIGNAL and $\overline{\text{SIGNAL}}$ below).
deassert a signal	Change the value of the signal from the logically True state to the logically False state.
$\overline{\text{SIGNAL}}$	Signal names are in uppercase and in the Courier font. An overbar on a signal name indicates that it is active low. To assert $\overline{\text{SIGNAL}}$ is to drive it low; to deassert $\overline{\text{SIGNAL}}$ is to drive it high

Table 0-1. Document Conventions (continued)

Register	Meaning
SIGNAL	Signal names are in uppercase and in the Courier font. An active high signal has no overbar. To assert SIGNAL is to drive it high; to deassert SIGNAL is to drive it low.
Numbers	
X	An uppercase X indicates any of several values is allowed, where X can be any legal pattern. For example, a binary value of 0X00 can be 0100 or 0000, a hex value of 0xX is 0x0 or 0x1, and so on.
0x	Hexadecimal numbers have a prefix of 0x. For example, 0x00FF is the hexadecimal number FF. All other numbers within register tables are assumed to be binary. Within conceptual information, binary numbers are indicated with a b suffix (for example, 1011b) and decimal numbers are written without a prefix or suffix.

The Device

The CC2538 integrates an IEEE 802.15.4 (2.4 GHz) radio, ARM Cortex-M3 processor, security acceleration, and enough flash and RAM to run the ZigBee IP® stack and SE2.0 profile without requiring external memory. The CC2538 also includes hardware support for ECC and RSA public key calculations and is aimed at Smart Grid applications such as ZigBee Smart Energy 2.0. The CC2538 device familiy currently supports TI's ZigBee PRO stack Z-Stack™ with associated profiles and the ZigBee IP stack with Smart Energy 2.0 profile.

The usage is, however, not limited to these protocols alone. The CC2538 is, for example, also suitable for 6LoWPAN and wireless HART implementations.

The CC2538 is suited for systems where very low power consumption is required. Very low-power sleep modes are available. Short transition times between operating modes further enable low power consumption.

For a complete feature list of any of the devices, see the corresponding data sheet.

Community Resources

All technical support is channeled through the TI Product Information Centers (PIC) - www.ti.com/support. To send an E-mail request, please enter your contact information, along with your request at the following link – PIC request form.

The following link connects to TI community resources. Linked contents are provided "AS IS" by the respective contributors. They do not constitute TI specifications and do not necessarily reflect TI's views; see TI's Terms of Use.

TI Embedded Processors Wiki — Texas Instruments Embedded Processors Wiki

Established to assist developers using the many Embedded Processors from TI to get started, help each other innovate, and foster the growth of general knowledge about the hardware and software surrounding these devices.

Register, Field, and Bit Calls

The naming convention applied for a call consists of:

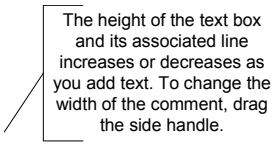
- For a register call: <*Module name*>.<*Register name*>; for example: UART.UASR
- For a bit field call:
  - <*Module name*>.<*Register name*>[*End:Start*] <*Field name*> *field*; for example, UART.UASR[4:0] SPEED bit field
  - <*Field name*> *field* <*Module name*>.<*Register name*>[*End:Start*]; for example, SPEED bit field UART.UASR[4:0]
- For a bit call:
  - <*Module name*>.<*Register name*>[*pos*] <*Bit name*> *bit*; for example, UART.UASR[5] BIT_BY_CHAR bit
  - <*Bit name*> *bit* <*Module name*>.<*Register name*>[*pos*]; for example, BIT_BY_CHAR bit UART.UASR[5]

To help the reader navigate the document, each register call is hyperlinked to its register description in the register manual section. After each register description, a table summarizes all hyperlinked register calls.

To navigate in the PDF documents, see Acrobat Reader Tips.

Flow Chart Rules

Flow charts follow the following rules:

Shape	Name	Definition
	Process	Any computational steps or processing function of a program; defined operation(s) causing change in value, form, or location of information
	Decision	A decision or switching-type operation that determines which of a number of alternate paths is followed
	Predefined process or sub-process	One or more named operations or program steps specified in a subroutine or another set of flow charts
	Data or I/O	General I/O function; information available for processing (input) or recording of processed information (output)
	Terminator	Terminal point in a flow chart: start, stop, halt, delay, or interrupt; may show exit from a closed subroutine
	Annotation	Additional descriptive clarification, comment
	On page connector (reference)	Exit to, or entry from, another part of chart in the same page
	Off page connector (reference)	The flow continues on a different page.
	Summing Junction	Logical AND
	Or	Logical OR
	Parallel mode (ISO)	Beginning or end of two or more simultaneous operations
	Flow Line	Lines indicate the sequence of steps and the direction of flow.

Acrobat Reader Tips

Acrobat includes two methods to search for words in a PDF:

- The Find toolbar provides a basic set of options to locate a word in the current PDF.
- The Search window lists words or partial words that match your text in the current PDF.

These guidelines apply to Acrobat Reader 5.x, 6.0, and 7.0.

For more information on Acrobat Reader search features, see the Adobe Reader Help.

To search for words in a document using the Find dialog box:

1. Open the document.
2. To display the Find toolbar, right-click in the toolbar area and select Find.
3. In the Find box, type the word, words, or partial words for which you want to search.
4. From the Find Options menu, select options as desired.
5. To view each search result, click the Find toolbar, the Find Previous button, or the Find Next button to go backward or forward through the document.

To search for words in a document using the Search PDF window:

1. Open the document.
2. Click the Search button on the File toolbar or right-click on your document and select Search.
3. Type the word, words, or part of a word for which you want to search.
4. Click Search.
5. The results appear in page order and, if applicable, show a few words of context. Each result displays an icon to identify the type of occurrence. All other searchable areas display the Search Result icon.
6. To display the page that contains a search result, click an item in the Results list. The occurrence is highlighted.
7. To navigate to the next result, choose Edit > Search Results > Next Result (or Ctrl+G).
8. To navigate to the previous result, choose Edit > Search Results > Previous Result (or Shift+Ctrl+G).

Navigate through your previous view

To retrace your path within an Adobe PDF document:

- For the previous view: Choose View > Go To > Previous View or Alt+Left Arrow.
- For the next view: Choose View > Go To > Next View or Alt+Right Arrow. The Next View command is available only if you have chosen Previous View.

If you view the PDF document in a browser, use options on the Navigation toolbar to move between views.

- Right-click the toolbar area, and then choose Navigation.
- Click the Go To Previous View button or the Go To Next View button.

NOTE: This navigation tip is useful to return to your previous view after clicking on a register call hyperlink.

Architectural Overview

The CC2538 device family provides solutions for a wide range of applications. To help the user develop these applications, this user's guide focuses on the use of the different building blocks of the CC2538 device family. For detailed device descriptions, complete feature lists, and performance numbers, see the data sheet. To provide easy access to relevant information, the following subsections guide the reader to the different chapters in this guide.

The CC2538 system-on-chip (SoC) is tailored to the requirements of complex mesh networking Internet Of Things applications of which ZigBee® Smart Energy 2.0 is the most demanding on the hardware. The combination of 32-MHz ARM® Cortex™-M3 processing, sufficient memory, and a wide selection of peripherals makes the CC2538 device family ideal for single-chip implementation of ZigBee nodes.

Topic **Page**

1.1 Target Applications

The CC2538 family is positioned for low-power wireless applications such as:

- IEEE 802.15.4 Radio Networks
- ZigBee Smart Energy 1.x and to 2.0 profiles
- Home and building automation
- Intelligent lighting systems
- Internet of Things - 6LoWPAN
- Industrial control and monitoring
- Consumer electronics
- Health care

1.2 Overview

Figure 1-1 shows the different building blocks of the CC2538.

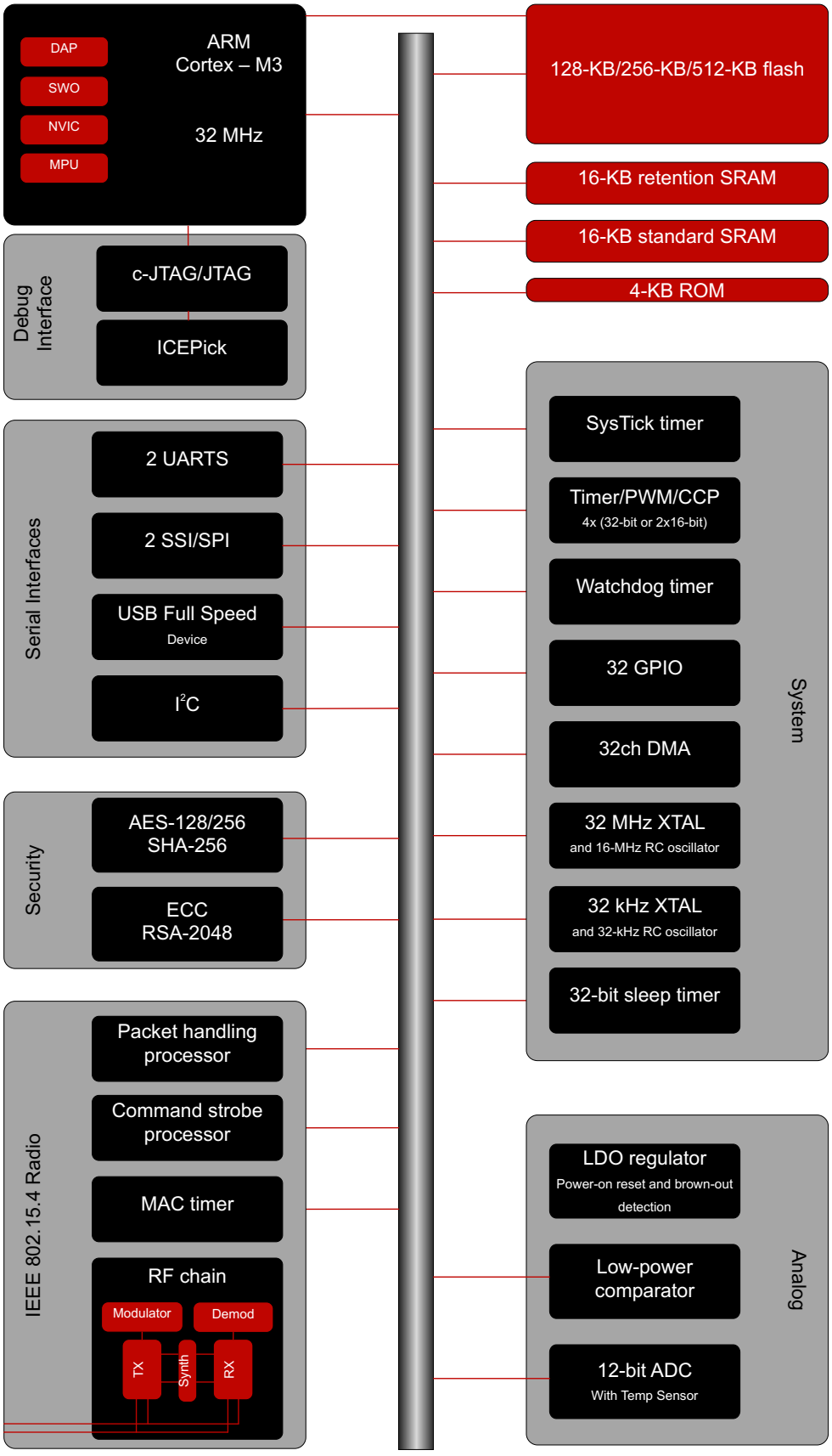


Figure 1-1. CC2538 Block Diagram

The CC2538 device has the following features:

- ARM Cortex-M3 processor core
  - 32-MHz operation
  - ARM Cortex SysTick timer
  - Nested vectored interrupt controller (NVIC)
- On-chip memory
  - Up to 512KB single-cycle flash memory up to 16 MHz; a prefetch buffer improves performance above 16 MHz
  - Up to 32KB single-cycle SRAM
    - Boot loader and utility library in ROM
- Advanced serial integration
  - USB 2.0 full-speed (FS) device (12 Mbps)
  - Two universal asynchronous receiver/transmitters (UARTs) with IrDA, 9-bit (one UART with modem flow control)
  - One inter-integrated circuit (I^2C™) module
  - Two synchronous serial interface modules (SSIs)
- System integration
  - Direct memory access (DMA) controller
  - System control and clocks including on-chip 16-MHz oscillator and 32-MHz crystal oscillator
  - Four 32-bit timers (up to eight 16-bit) with pulse width modulation (PWM) capability and synchronization
  - 32-bit, 32-kHz sleep timer
  - Watchdog timer
  - Clock loss detection circuit
  - 32 GPIOs
  - 8 GPIOs with analog capability; 4 of the GPIOs have 20-mA drive capability
  - Fully flexible pin muxing allows use as GPIO or any peripheral function
  - Power-management system with powerful low-power modes down to 500 nA with pin wakeup and retention
- Analog
  - 12-bit analog-to-digital converter (ADC) with eight analog input channels
  - Low-power analog comparator
  - On-chip temperature and supply voltage sensing with the ADC
  - On-chip voltage regulator
- IEEE 1149.7 compliant cJTAG with legacy 1149.1 JTAG® support
- 56-pin QFN 8x8 mm package
- Extended (–40°C to 125°C) temperature range

For applications requiring extreme conservation of power, the CC2538 device features a power-management system to efficiently power down the CC2538 device to a low-power state during extended periods of inactivity. A power-up and power-down sequencer, a 32-bit sleep timer (which is a real-time clock [RTC]) with interrupt and 16KB of RAM with retention in all power modes positions the CC2538 microcontroller perfectly for battery applications.

In addition, the CC2538 microcontroller offers the advantages of ARM's widely available development tools, SoC infrastructure IP applications, and a large user community. Additionally, the microcontroller uses ARM's Thumb®-compatible Thumb-2 instruction set to reduce memory requirements and, thereby, cost.

TI offers a complete solution to get to market quickly, with evaluation and development boards, white papers and application notes, an easy-to-use peripheral driver library, and a strong support, sales, and distributor network.

1.3 Functional Overview

The following sections provide an overview of the features of the CC2538 microcontroller. The page number in parentheses indicates where that feature is discussed in detail.

1.3.1 ARM Cortex-M3

The following sections provide an overview of the ARM Cortex-M3 processor core and instruction set, the integrated system timer (SysTick), and the NVIC.

1.3.1.1 Processor Core

The CC2538 is designed around an ARM Cortex-M3 processor core. The ARM Cortex-M3 processor provides the core for a high-performance, low-cost platform that meets the needs of minimal memory implementation, reduced pin count, and low power consumption, while delivering outstanding computational performance and exceptional system response to interrupts.

- 32-bit ARM Cortex-M3 architecture optimized for small-footprint embedded applications
- Outstanding processing performance combined with fast interrupt handling
- Thumb-2 mixed 16- and 32-bit instruction set delivers the high performance expected of a 32-bit ARM core in a compact memory size usually associated with 8- and 16-bit devices, typically in the range of a few kilobytes of memory for microcontroller-class applications
  - Single-cycle multiply instruction and hardware divide
  - Atomic bit manipulation (bit-banding), delivering maximum memory use and streamlined peripheral control
  - Unaligned data access, enabling data to be efficiently packed into memory
- Fast code execution permits slower processor clock or increases sleep mode time.
- Harvard architecture characterized by separate buses for instruction and data
- Efficient processor core, system and memories
- Hardware division and fast multiplier
- Deterministic, high-performance interrupt handling for time-critical applications
- Memory protection unit (MPU) provides a privileged mode for protected operating system functionality.
- Enhanced system debug with extensive breakpoint capabilities and debugging through power modes
- cJTAG reduces the number of pins required for debugging.
- Ultra-low power consumption with integrated sleep modes
- 32-MHz operation

1.3.1.2 Memory Map

A memory map lists the location of instructions and data in memory. The memory map for the CC2538 can be found in the Memories section of this User's Guide. Register addresses are given as a hexadecimal increment, relative to the base address of the module as shown in the memory map.

1.3.1.3 System Timer (SysTick)

ARM Cortex-M3 includes an integrated system timer, SysTick. SysTick provides a simple, 24-bit, clear-on-write, decrementing, wrap-on-zero counter with a flexible control mechanism. The counter can be used in several different ways; for example:

- An RTOS tick timer that fires at a programmable rate (for example, 100 Hz) and invokes a SysTick routine
- A high-speed alarm timer using the system clock 11

- A variable rate alarm or signal timer—the duration is range-dependent on the reference clock used and the dynamic range of the counter.
- A simple counter used to measure time to completion and time used
- An internal clock-source control based on missing or meeting durations

1.3.1.4 Nested Vector Interrupt Controller

The CC2538 controller includes the ARM NVIC. The NVIC and Cortex-M3 prioritize and handle all exceptions in handler mode. The processor state is automatically stored to the stack on an exception and automatically restored from the stack at the end of the interrupt service routine (ISR). The interrupt vector is fetched in parallel to the state saving, thus enabling efficient interrupt entry. The processor supports tail-chaining, meaning that back-to-back interrupts can be performed without the overhead of state saving and restoration. Software can set eight priority levels on seven exceptions (system handlers) and CC2538 interrupts.

- Deterministic, fast interrupt processing: always 12 cycles, or just 6 cycles with tail-chaining
- External nonmaskable interrupt (NMI) signal available for immediate execution of NMI handler for safety critical applications
- Dynamically re-prioritizable interrupts
- Exceptional interrupt handling through hardware implementation of required register manipulations

1.3.1.5 System Control Block

The system control block (SCB) provides system implementation information and system control, including configuration, control, and reporting of system exceptions.

1.3.1.6 MPU

The MPU supports the standard ARM7 protected memory system architecture (PMSA) model. The MPU provides full support for protection regions, overlapping protection regions, access permissions, and exporting memory attributes to the system.

1.3.2 On-Chip Memory

The following sections describe the on-chip memory modules.

1.3.2.1 SRAM

The CC2538 provides a 16KB block of single-cycle on-chip SRAM with full retention in all power modes. In addition, some variants offer an additional 16KB of single-cycle on-chip SRAM without retention in the lowest power modes.

Because read-modify-write (RMW) operations are very time consuming, ARM has introduced bit-banding technology in the Cortex-M3 processor. With a bit-band-enabled processor, certain regions in the memory map (SRAM and peripheral space) can use address aliases to access individual bits in a single, atomic operation.

Data can be transferred to and from the SRAM using the micro DMA (µDMA) controller.

1.3.2.2 Flash Memory

The flash block provides in-circuit programmable nonvolatile program memory for the device. The flash memory is organized as a set of 2KB pages that can be individually erased. Erasing a block causes the entire contents of the block to be reset to all 1s. These pages can be individually protected. Read-only blocks cannot be erased or programmed, protecting the contents of those blocks from being modified. In addition to holding program code and constants, the nonvolatile memory allows the application to save data that must be preserved such that it is available after restarting the device. Using this feature one can, for example, use saved network-specific data to avoid the need for a full start-up and network find-and-join process.

1.3.2.3 ROM

The ROM is preprogrammed with a serial boot loader (SPI or UART). For applications that require in-field programmability, the royalty-free CC2538 boot loader can act as an application loader and support in-field firmware updates.

1.3.3 Radio

The CC2538 device family provides a highly integrated low-power IEEE 802.15.4-compliant radio transceiver. The radio subsystem provides an interface between the MCU and the radio which makes it possible to issue commands, read status, and automate and sequence radio events. The radio also includes a packet-filtering and address-recognition module.

1.3.4 AES Engine with 128, 192 256 Bit Key Support

- CCM, GCM, CTR, CBC-MAC, ECB modes of operation
- SHA-256 hash function
- Secure key storage memory
- High throughput, low latency
- Public key accelerator
- Elliptic Curve Cryptography (ECC) and RSA-2048
- Support for RSA-2048 makes it ideal for ESIs
- Keeps the key exchange algorithms out of the CPU cycle budget and reduces energy consumption

1.3.5 Programmable Timers

Programmable timers can be used to count or time external events that drive the timer input pins. Each 16- or 32-bit GPTM block provides two 16-bit timers or counters that can be configured to operate independently as timers or event counters, or configured to operate as one 32-bit timer.

The general-purpose timer module (GPTM) contains four 16- or 32-bit GPTM blocks with the following functional options:

- 16- or 32-bit operating modes:
  - 16- or 32-bit programmable one-shot timer
  - 16- or 32-bit programmable periodic timer
  - 16-bit general-purpose timer with an 8-bit prescaler
  - 16-bit input-edge count- or time-capture modes with an 8-bit prescaler
  - 16-bit PWM mode with an 8-bit prescaler and software-programmable output inversion of the PWM signal
- Count up or down
- Eight 16- or 32-bit capture compare PWM pins (CCP)
- Daisy-chaining of timer modules to allow a single timer to initiate multiple timing events
- Timer synchronization allows selected timers to start counting on the same clock cycle.
- User-enabled stalling when the microcontroller asserts CPU Halt flag during debug
- Ability to determine the elapsed time between the assertion of the timer interrupt and entry into the interrupt service routine.
- Efficient transfers using the µDMA controller:
  - Dedicated channel for each timer
  - Burst request generated on timer interrupt

1.3.5.1 MAC Timer

The MAC timer is specially designed for supporting an IEEE 802.15.4 MAC or other time-slotted protocol in software. The MAC timer has a configurable timer period and a 24-bit overflow counter that can be used to keep track of the number of periods that have transpired. A 40-bit capture register is also used to record the exact time at which a start-of-frame delimiter is received or transmitted or the exact time at which transmission ends, as well as two 16-bit output compare registers and two 24-bit overflow compare registers that can send various command strobes (start RX, start TX, and so forth) at specific times to the radio modules.

1.3.5.2 Watchdog Timer

A watchdog timer is used to regain control when a system fails due to a software error or because an external device does not respond in the expected way. When enabled by software, the watchdog timer must be cleared periodically; otherwise, the watchdog timer resets the CC2538 device when it times out.

1.3.5.3 Sleep Timer

The sleep timer is an ultralow-power 32-bit timer that counts 32-kHz crystal oscillator or 32-kHz RC oscillator periods. The sleep timer runs continuously in all operating modes except power mode 3 (PM3). Typical applications of this timer are as a real-time counter or as a wake-up timer for coming out of power mode 1 (PM1) or power mode 2 (PM2).

- 32-bit real-time seconds counter (RTC) with 1/32,768 second resolution
- 32-bit RTC match register for timed wake-up and interrupt generation

1.3.5.4 CCP Pins

CCP pins can be used by the GPTM to time or count external events using the CCP pin as an input. Alternatively, the GPTM can generate a simple PWM output on the CCP pin.

Any of the GPIOs can be programmed to operate in the following modes:

- Capture: The GPTM is incremented or decremented by programmed events on the CCP input. The GPTM captures and stores the current timer value when a programmed event occurs.
- Compare: The GPTM is incremented or decremented by programmed events on the CCP input. The GPTM compares the current value with a stored value and generates an interrupt when a match occurs.
- PWM: The GPTM is incremented or decremented by the system clock. A PWM signal is generated based on a match between the counter value and a value stored in a match register; the PWM signal is then output on the CCP pin.

1.3.6 Direct Memory Access

The CC2538 microcontroller includes a DMA controller, known as µDMA. The µDMA controller provides a way to offload data transfer tasks from the Cortex-M3 processor, allowing for more efficient use of the processor and the available bus bandwidth. The µDMA controller can perform transfers between memory and peripherals. It has dedicated channels for each supported on-chip module and can be programmed to automatically perform transfers between peripherals and memory as the peripheral is ready to transfer more data. The µDMA controller provides the following features:

- ARM PrimeCell 32-channel configurable µDMA controller
- Support for memory-to-memory, memory-to-peripheral, and peripheral-to-memory in multiple transfer modes:
  - Basic for simple transfer scenarios
  - Ping-pong for continuous data flow
  - Scatter-gather for a programmable list of arbitrary transfers initiated from a single request
- Highly flexible and configurable channel operation:
  - Independently configured and operated channels

  - Dedicated channels for supported on-chip modules
  - Primary and secondary channel assignments
  - Flexible channel assignments
  - One channel each for RX and TX path for bidirectional modules
  - Dedicated channel for software-initiated transfers
  - Per-channel configurable priority scheme
  - Optional software-initiated requests for any channel
- Two levels of priority
- Design optimizations for improved bus access performance between the µDMA controller and the processor core:
  - µDMA controller access is subordinate to core access
  - RAM striping
  - Peripheral bus segmentation
- Data sizes of 8, 16, and 32 bits
- Transfer size is programmable in binary steps from 1 to 1024.
- Source and destination address increment size of byte, half-word, word, or no increment
- Maskable peripheral requests
- Interrupt on transfer completion, with a separate interrupt per channel

1.3.7 System Control and Clock

System control determines the overall operation of the CC2538 device. System control provides information about the CC2538 device, controls power-saving features, controls the clocking of the CC2538 device and individual peripherals, and handles reset detection and reporting.

- Device identification information: version, part number, SRAM size, flash memory size, and so forth
- Power control:
  - On-chip fixed low drop-out (LDO) voltage regulator
  - Handles the power-up and power-down sequencing and control for the core digital logic and analog circuits
  - Low-power options for the CC2538 microcontroller: Sleep and deep-sleep modes with clock gating
  - Low-power options for on-chip modules: Software controls shutdown of individual peripherals and memory
    - General-purpose input/output (GPIO) states are retained in all power modes
    - 16KB RAM and configuration registers are retained in all power modes
  - Control pin option for control of external DC-DC regulator
  - Configurable wakeup from sleep timer or any GPIO interrupt
  - 3.3-V supply power-on-reset (POR) and 1.8-V brown-out reset (BOR) detection and reset generation
- Multiple clock sources for microcontroller system clock:
  - RC oscillator (RCOSC16M): On-chip resource providing a 16-MHz frequency:
    - Automatically calibrated against the 32-MHz XTAL oscillator when running
    - Software power-down control for low-power modes
  - 32-MHz crystal oscillator (XOSC32M): A frequency-accurate clock source by one of two means: an external single-ended clock source is connected to the XOSC32M_Q1 input pin, or an external crystal is connected across the XOSC32M_Q1 input and XOSC32M_Q2 output pins.
  - Internal 32-kHz RC oscillator: on chip resource providing a 30 kHz frequency, used during power-saving modes
    - Automatically calibrated against the 32-MHz XTAL oscillator when running
  - 32.768-kHz crystal oscillator: A frequency-accurate clock source by one of two means: an external

single-ended clock source is connected to the XOSC32K_Q1 input pin, or an external crystal is connected across the XOSC32K_Q1 input and XOSC32K_Q2 output pins.
    - Ideal for accurate RTC operation or synchronous network timing
  - CPU and periphery clock division options down to 256 kHz
- Flexible reset sources
  - POR
  - Reset pin assertion
  - BOR detector alerts to system power drops
  - Software reset
  - Watchdog timer reset

1.3.8 Serial Communications Peripherals

The CC2538 device supports both asynchronous and synchronous serial communications with:

- USB 2.0 FS device
- Two UARTs with 9-bit
- I^2C module
- Two SSI

The following sections provide more detail on each of these communications functions.

1.3.8.1 USB

Universal serial bus (USB) is a serial bus standard designed to allow peripherals to be connected and disconnected using a standardized interface.

The CC2538 device supports the USB 2.0 FS configuration in device mode and has the following features:

- Complies with USB-IF certification standards
- USB 2.0 full speed (12 Mbps) operation with integrated PHY
- 4 transfer types: control, interrupt, bulk, and isochronous
- Five IN and five OUT configurable endpoints
- Support for packet sizes between 8 to 256 bytes and remote wake-up.
- 1KB of dedicated endpoint memory flexibly assigned to the different endpoints
- Efficient transfers using the µDMA controller

1.3.8.2 UART

A UART is an integrated circuit used for RS-232C serial communications, containing a transmitter (parallel-to-serial converter) and a receiver (serial-to-parallel converter), each clocked separately.

The CC2538 microcontroller includes two fully programmable 16C550-type UARTs. Although the functionality is similar to a 16C550 UART, this UART design is not register compatible. The UART can generate individually masked interrupts from the receive (RX), transmit (TX), modem flow control, and error conditions. The module generates a single combined interrupt when any of the interrupts are asserted and are unmasked.

The two UARTs have the following features:

- Programmable baud-rate generator allowing speeds up to 2 Mbps for regular speed (divide by 16) and 4 Mbps for high speed (divide by 8)
- Separate 16x8 TX and RX FIFOs to reduce CPU interrupt service loading
- Programmable FIFO length, including 1-byte deep operation providing conventional double-buffered interface
- FIFO trigger levels of 1/8, 1/4, 1/2, 3/4, and 7/8
- Standard asynchronous communication bits for start, stop, and parity

- Line-break generation and detection
- Fully programmable serial interface characteristics:
  - 5, 6, 7, or 8 data bits
  - Even, odd, stick, or no-parity bit generation and detection
  - 1 or 2 stop-bit generation
- Full modem handshake support (on UART1)
- Modem flow control (on UART1)
- LIN protocol support
- EIA-485 9-bit support
- Standard FIFO-level and end-of-transmission (EoT) interrupts
- Efficient transfers using the µDMA controller:
  - Separate channels for TX and RX
  - Receive single request asserted when data is in the FIFO; burst request asserted at programmed FIFO level
  - Transmit single request asserted when there is space in the FIFO; burst request asserted at programmed FIFO level

1.3.8.3 I^2C

The I^2C bus provides bidirectional data transfer through a 2-wire design (a serial data line SDA and a serial clock line SCL). The I^2C bus interfaces to external I^2C devices such as serial memory (RAMs and ROMs), networking devices, LCDs, tone generators, and so on. The I^2C bus may also be used for system testing and diagnostic purposes in product development and manufacturing.

Each device on the I^2C bus can be designated as a master or a slave. Each I^2C module supports both sending and receiving data as either a master or a slave and can operate simultaneously as both a master and a slave. Both the I^2C master and slave can generate interrupts.

The CC2538 microcontroller includes an I^2C module with the following features:

- Devices on the I^2C bus can be designated as either a master or a slave:
  - Supports both transmitting and receiving data as either a master or a slave
  - Supports simultaneous master and slave operation
- Four I^2C modes:
  - Master transmit
  - Master receive
  - Slave transmit
  - Slave receive
- Two transmission speeds: Standard (100 Kbps) and fast (400 Kbps)
- Clock low time-out interrupt
- Dual slave address capability
- Master and slave interrupt generation:
  - Master generates interrupts when a TX or RX operation completes (or aborts due to an error).
  - Slave generates interrupts when data is transferred or requested by a master or when a START or STOP condition is detected.
  - Master with arbitration and clock synchronization, multimaster support, and 7-bit addressing mode

1.3.8.4 SSI

An SSI module is a 4-wire bidirectional communications interface that converts data between parallel and serial. The SSI performs serial-to-parallel conversion on data received from a peripheral device, and parallel-to-serial conversion on data transmitted to a peripheral device. The SSI can be configured as either a master or slave device. As a slave device, the SSI can also be configured to disable its output, which allows coupling of a master device with multiple slave devices. The TX and RX paths are buffered with separate internal FIFOs.

The SSI also includes a programmable bit rate clock divider and prescaler to generate the output serial clock derived from the input clock of the SSI. Bit rates are generated based on the input clock, and the maximum bit rate is determined by the connected peripheral.

The CC2538 includes two SSI modules with the following features:

- Programmable interface operation for Freescale SPI, MICROWIRE, or TI synchronous serial interfaces
- Master or slave operation
- Programmable clock bit rate and prescaler
- Separate TX and RX FIFOs, each 16 bits wide and 8 locations deep
- Programmable data frame size from 4 to 16 bits
- Internal loopback test mode for diagnostic and debug testing
- Standard FIFO-based interrupts and EoT interrupt
- Efficient transfers using the µDMA controller:
  - Separate channels for TX and RX
  - Receive single request asserted when data is in the FIFO; burst request asserted when FIFO contains four entries
  - Transmit single request asserted when there is space in the FIFO; burst request asserted when FIFO contains four entries

1.3.9 Programmable GPIOs

GPIO pins offer flexibility for a variety of connections. The CC2538 GPIO module is comprised of four GPIO blocks, each corresponding to an individual GPIO port. The GPIO module supports CC2538 programmable I/O pins. The number of GPIOs available depends on the peripherals being used (see Chapter 5 for the signals available to each GPIO pin).

- Up to 32 GPIOs, depending on configuration
- 4 pins with 20-mA drive strength, 28 pins with 4-mA drive strength
- Fully flexible digital pin muxing allows use as GPIO or any of several peripheral functions
- Programmable control for GPIO interrupts:
  - Interrupt generation masking per pin
  - Edge-triggered on rising or falling
- Bit masking in read and write operations through address lines
- Can be used to initiate a µDMA transfer
- Pin state can be retained during all sleep modes
- Pins configured as digital inputs are Schmitt-triggered
- Programmable control for GPIO pad configuration:
  - Weak pull up or pull down resistors
  - Digital input enables

1.3.10 Analog

The CC2538 microcontroller provides analog functions integrated into the device, including:

- A 12-bit ADC with eight analog input channels

- Low-power analog comparator
- On-chip voltage regulator

The following provides more detail on these analog functions.

1.3.10.1 ADC

An ADC is a peripheral that converts a continuous analog voltage to a discrete digital number. The ADC module features 12-bit conversion resolution and supports eight input channels plus an internal division of the battery voltage and a temperature sensor.

- Eight shared analog input channels
- 12-bit precision ADC with up to 11.5 ENOB
- Single-ended and differential-input configurations
- On-chip internal temperature sensor
- Periodic sampling or conversion over a sequence of channels
- Converter uses an internal regulated reference, AVDD or an external reference
- Efficient transfers using the µDMA controller
  - Dedicated channel for each sample sequencer

1.3.10.2 Analog Comparator

An analog comparator is a peripheral that compares two analog voltages, two external pin inputs, and provides a logical output that signals the comparison result. The CC2538 microcontroller provides an independent integrated and low-power analog comparator that can be active in all power modes. The comparator output is mapped into the digital I/O port, and the MCU can treat the comparator output as a regular digital input.

1.3.10.3 Random Number Generator

The random number generator (RNG) uses a 16-bit LFSR to generate pseudorandom numbers, which can be read by the CPU or used directly by the command strobe processor. The RNG can be seeded with random data from noise in the radio ADC.

1.3.11 cJTAG, JTAG and SWO

The Joint Test Action Group (JTAG) port is an IEEE standard that defines a test access port (TAP) and Boundary Scan Architecture for digital integrated circuits and provides a standardized serial interface for controlling the associated test logic. The TAP, Instruction Register (IR), and Data Registers (DR) can be used to test the interconnections of assembled printed circuit boards and obtain manufacturing information on the components. The JTAG port also provides a means of accessing and controlling design-for-test features such as I/O pin observation and control, scan testing, and debugging. The compact JTAG (cJTAG) interface has the following features:

- IEEE 1149.1-1990-compatible TAP controller
- IEEE 1149.7 cJTAG interface
- ICEPick™ JTAG router
- Four-bit IR chain for storing JTAG instructions
- IEEE standard instructions: BYPASS, IDCODE, SAMPLE and PRELOAD, EXTEST and INTEST
- ARM additional instructions: APACC, DPACC, and ABORT

1.3.12 Packaging and Temperature

- 56-pin 8x8 mm QFN package
- Extended operating temperature from –40°C to +125°C

The Cortex-M3 Processor

This chapter describes the Cortex-M3 processor.

Topic **Page**

2.1 The Cortex-M3 Processor Introduction

The ARM® Cortex™-M3 processor provides a high-performance, low-cost platform that meets the system requirements of minimal memory implementation, reduced pin count, and low power consumption, while delivering outstanding computational performance and exceptional system response to interrupts. Features include:

- 32-bit ARM Cortex-M3 architecture optimized for small-footprint embedded applications
- Outstanding processing performance combined with fast interrupt handling
- Thumb®-2 mixed 16- and 32-bit instruction set delivers the high performance expected of a 32-bit ARM core in a compact memory size usually associated with 8- and 16-bit devices, typically in the range of a few kilobytes of memory for microcontroller-class applications:
  - Single-cycle multiply instruction and hardware divide
  - Atomic bit manipulation (bit-banding), delivering maximum memory use and streamlined peripheral control
  - Unaligned data access, enabling data to be efficiently packed into memory
- Fast code execution permits slower processor clock or increases sleep mode time.
- Harvard architecture characterized by separate buses for instruction and data
- Efficient processor core, system and memories
- Hardware division and fast digital-signal-processing orientated multiply accumulate
- Saturating arithmetic for signal processing
- Deterministic, high-performance interrupt handling for time-critical applications
- Memory protection unit (MPU) to provide a privileged mode for protected operating system functionality
- Enhanced system debug with extensive breakpoint and trace capabilities
- Serial wire trace reduce the number of pins required for debugging and tracing.
- Migration from the ARM7™ processor family for better performance and power efficiency
- Optimized for single-cycle flash memory use
- Ultra-low power consumption with integrated sleep modes
- 32-MHz operation
- 1.25 DMIPS/MHz

The CC2538 builds on this core to bring high-performance 32-bit computing to cost-sensitive embedded microcontroller applications, such as factory automation and control, industrial control power devices, building and home automation, and stepper motor control.

This chapter provides information on the CC2538 implementation of the Cortex-M3 processor, including the programming model, the memory model, the exception model, fault handling, and power management.

For technical details on the instruction set, see the *Cortex™-M3 Instruction Set Technical User's Manual*.

2.2 Block Diagram

The Cortex-M3 processor is built on a high-performance processor core, with a 3-stage pipeline Harvard architecture, making it ideal for demanding embedded applications. The processor delivers exceptional power efficiency through an efficient instruction set and extensively optimized design, which provides high-end processing hardware. The instruction set includes a range of single-cycle and SIMD multiplication and multiply-with-accumulate capabilities, saturating arithmetic, and dedicated hardware division.

To facilitate the design of cost-sensitive devices, the Cortex-M3 processor implements tightly coupled system components that reduce processor area while significantly improving interrupt handling and system debug capabilities. The Cortex-M3 processor implements a version of the Thumb® instruction set based on Thumb-2 technology, thus ensuring high code density and reduced program memory requirements. The Cortex-M3 instruction set provides the exceptional performance expected of a modern 32-bit architecture, with the high code density of 8-bit and 16-bit microcontrollers.

The Cortex-M3 processor closely integrates a nested vector interrupt controller (NVIC) to deliver industry-leading interrupt performance. The CC2538 NVIC includes a nonmaskable interrupt (NMI) and provides eight interrupt priority levels. The tight integration of the processor core and NVIC provides fast execution of interrupt service routines (ISRs), dramatically reducing interrupt latency. The hardware stacking of registers and the ability to suspend load-multiple and store-multiple operations further reduce interrupt latency. Interrupt handlers do not require any assembler stubs, thus removing code overhead from the ISRs. Tail-chaining optimization also significantly reduces the overhead when switching from one ISR to another. To optimize low-power designs, the NVIC integrates with the sleep modes, including deep-sleep mode, which enables the entire device to be rapidly powered down.

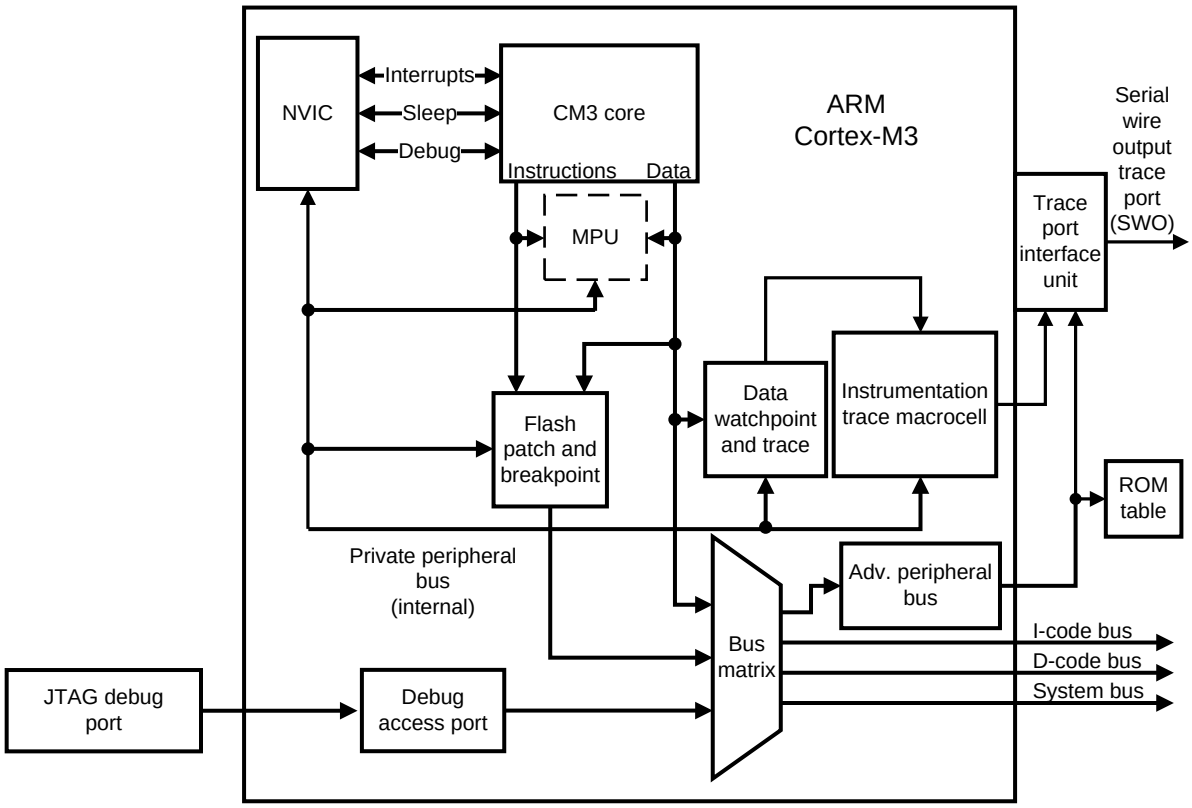


Figure 2-1. CPU Block Diagram

2.3 Overview

2.3.1 System-Level Interface

The Cortex-M3 processor provides multiple interfaces using AMBA™ technology to provide high-speed, low-latency memory accesses. The core supports unaligned data accesses and implements atomic bit manipulation that enables faster peripheral controls, system spinlocks, and thread-safe Boolean data handling.

An MPU in the Cortex-M3 processor provides fine-grain memory control, enabling applications to implement security privilege levels and separate code, data and stack on a task-by-task basis.

2.3.2 Integrated Configurable Debug

The Cortex-M3 processor implements a complete hardware debug solution, providing high system visibility of the processor and memory through a traditional JTAG® port.

For system trace, the processor integrates an instrumentation trace macrocell (ITM) alongside data watch points and a profiling unit. To enable simple and cost-effective profiling of the system trace events, a serial wire viewer (SWV) can export a stream of software-generated messages, data trace, and profiling information through a single pin.

The flash patch and breakpoint unit (FPB) provides up to eight hardware breakpoint comparators that debuggers can use. The comparators in the FPB also provide remap functions of up to eight words in the program code in the CODE memory region. Remap functions enable patching of applications stored in a read-only area of flash memory into another area of on-chip SRAM or flash memory. If a patch is required, the application programs the FPB to remap a number of addresses. When those addresses are accessed, the accesses are redirected to a remap table specified in the FPB configuration.

For more information on the Cortex-M3 debug capabilities, see the *ARM® Debug Interface V5 Architecture Specification*.

2.3.3 Trace Port Interface Unit

The trace port interface unit (TPIU) acts as a bridge between the Cortex-M3 trace data from the ITM, and an off-chip trace port analyzer (see Figure 2-2).

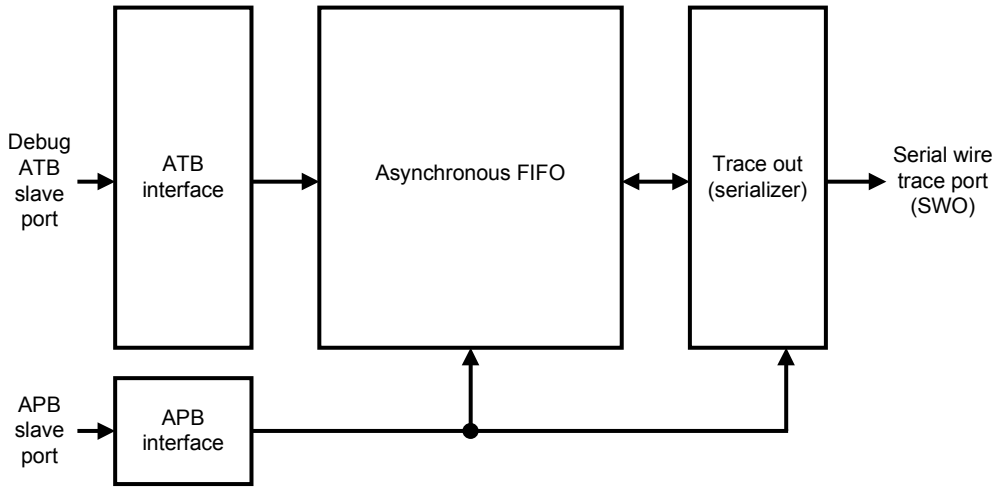


Figure 2-2. TPIU Block Diagram

2.3.4 Cortex-M3 System Component Details

The Cortex-M3 includes the following system components:

- SysTick: A 24-bit count-down timer that can be used as a real-time operating system (RTOS) tick timer or as a simple counter (see Section 3.2.1)
- NVIC: An embedded interrupt controller (INTC) that supports low-latency interrupt processing (see Section 3.2.2)
- System control block (SCB): The programming model interface to the processor. The SCB provides system implementation information and system control, including configuration, control, and reporting of system exceptions (see Section 3.2.3).
- MPU: Improves system reliability by defining the memory attributes for different memory regions. The MPU provides up to eight different regions and an optional predefined background region (see Section 3.2.4).

2.4 Programming Model

This section describes the Cortex-M3 programming model. In addition to the descriptions of the individual core registers, information about the processor modes and privilege levels for software execution and stacks is included.

2.4.1 Processor Mode and Privilege Levels for Software Execution

The Cortex-M3 has two modes of operation:

- Thread mode: Used to execute application software. The processor enters thread mode when it comes out of reset.
- Handler mode: Used to handle exceptions. When the processor completes exception processing, it returns to thread mode.

In addition, the Cortex-M3 has two privilege levels, unprivileged and privileged.

- In unprivileged mode, software has the following restrictions:
  - Limited access to the MSR and MRS instructions and no use of the CPS instruction
  - No access to the system timer, NVIC, or system control block
  - Possibly restricted access to memory or peripherals
- In privileged mode, software can use all the instructions and has access to all resources.

In thread mode, the CONTROL register (see Control Register (CONTROL)) controls whether software execution is privileged or unprivileged. In handler mode, software execution is always privileged.

Only privileged software can write to the CONTROL register to change the privilege level for software execution in thread mode. Unprivileged software can use the SVC instruction to make a supervisor call to transfer control to privileged software.

2.4.2 Stacks

The processor uses a full descending stack, meaning that the stack pointer indicates the last stacked item on the memory. When the processor pushes a new item onto the stack, it decrements the stack pointer and then writes the item to the new memory location. The processor implements two stacks: the main stack and the process stack, with a pointer for each held in independent registers (see the SP register on Stack Pointer (SP), *Stack Pointer (SP)*).

In thread mode, the CONTROL register (see Control Register (CONTROL)) controls whether the processor uses the main stack or the process stack. In handler mode, the processor always uses the main stack. Table 2-1 lists the options for processor operations.

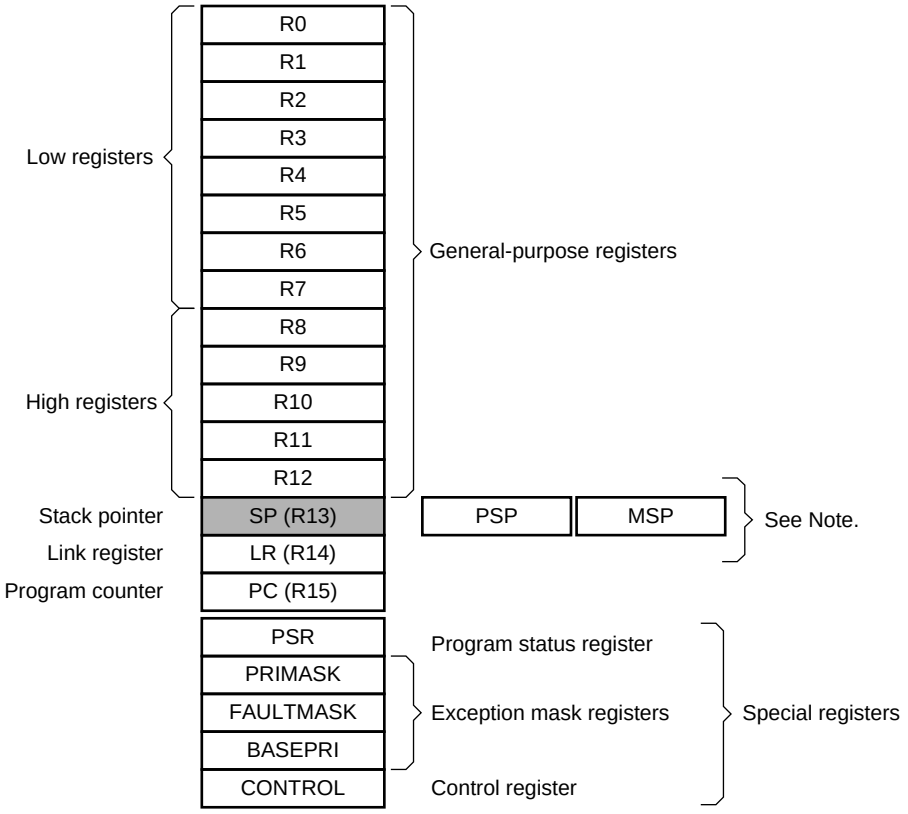
Table 2-1. Summary of Processor Mode, Privilege Level, and Stack Use

Processor Mode	Use	Privilege Level	Stack Used
Thread	Applications	Privileged or unprivileged (1)	Main stack or process stack
Handler	Exception handlers	Always privileged	Main stack

(1) See CONTROL (Control Register (CONTROL)).

2.4.3 Register Map

Figure 2-3 shows the Cortex-M3 register set. Table 2-2 lists the core registers. The core registers are not memory mapped and are accessed by register name, so the base address is N/A (not applicable) and there is no offset.



NOTE:: Banked version of SP

Figure 2-3. Cortex-M3 Register Set

Table 2-2. Processor Register Map

Offset	Name	Type	Reset	Description	Link
–	R0	R/W	–	Cortex general-purpose register 0	See Cortex General-Purpose Register 0 (R0) .
–	R1	R/W	–	Cortex general-purpose register 1	See Cortex General-Purpose Register 1 (R1).
–	R2	R/W	–	Cortex general-purpose register 2	See Cortex General-Purpose Register 2 (R2).
–	R3	R/W	–	Cortex general-purpose register 3	See Cortex General-Purpose Register 3 (R3).
–	R4	R/W	–	Cortex general-purpose register 4	See Cortex General-Purpose Register 4 (R4).
–	R5	R/W	–	Cortex general-purpose register 5	See Cortex General-Purpose Register 5 (R5).
–	R6	R/W	–	Cortex general-purpose register 6	See Cortex General-Purpose Register 6 (R6).
–	R7	R/W	–	Cortex general-purpose register 7	See Cortex General-Purpose Register 7 (R7).
–	R8	R/W	–	Cortex general-purpose register 8	See Cortex General-Purpose Register 8 (R8).
–	R9	R/W	–	Cortex general-purpose register 9	See Cortex General-Purpose Register 9 (R9).
–	R10	R/W	–	Cortex general-purpose register 10	See Cortex General-Purpose Register 10 (R10) .

Table 2-2. Processor Register Map (continued)

Offset	Name	Type	Reset	Description	Link
–	R11	R/W	–	Cortex general-purpose register 11	See Cortex General-Purpose Register 11 (R11) .
–	R12	R/W	–	Cortex general-purpose register 12	See Cortex General-Purpose Register 12 (R12).
–	SP	R/W	–	Stack pointer	See Stack Pointer (SP).
–	LR	R/W	0xFFFF FFFF	Link register	See Link Register (LR).
–	PC	R/W	–	Program counter	See Program Counter (PC).
–	PSR	R/W	0x0100 0000	Program status register	See Program Status Register (PSR).
–	PRIMASK	R/W	0x0000 0000	Priority mask register	See Priority Mask Register (PRIMASK).
–	FAULTMASK	R/W	0x0000 0000	Fault mask register	See Fault Mask Register (FAULTMASK).
–	BASEPRI	R/W	0x0000 0000	Base priority mask register	See Base Priority Mask Register (BASEPRI).
–	CONTROL	R/W	0x0000 0000	Control register	See Control Register (CONTROL).

2.4.4 Register Descriptions

This section lists and describes the Cortex-M3 registers, in the order shown in Figure 2-3. The core registers are not memory mapped and are accessed by register name rather than offset.

NOTE: The register type shown in the register descriptions refers to type during program execution in thread mode and handler mode. Debug access can differ.

Cortex General-Purpose Register 0 (R0)

Address Offset		**Reset**	–
Physical Address		**Instance**	
Description	The Rn registers are 32-bit general-purpose registers for data operations and can be accessed from either privileged or unprivileged mode.		
Type	R/W		

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

Bits	Field Name	Description	Type	Reset
31:0	DATA	Register data	RW	—

Cortex General-Purpose Register 1 (R1)

Address Offset		**Reset**	–
Physical Address		**Instance**	
Description	The Rn registers are 32-bit general-purpose registers for data operations and can be accessed from either privileged or unprivileged mode.		
Type	R/W		

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

Bits	Field Name	Description	Type	Reset
31:0	DATA	Register data	RW	—

Cortex General-Purpose Register 2 (R2)

Address Offset		**Reset**	–
Physical Address		**Instance**	
Description	The Rn registers are 32-bit general-purpose registers for data operations and can be accessed from either privileged or unprivileged mode.		
Type	R/W		

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

Bits	Field Name	Description	Type	Reset
31:0	DATA	Register data	RW	—

Cortex General-Purpose Register 3 (R3)

Address Offset		**Reset**	–
Physical Address		**Instance**	
Description	The Rn registers are 32-bit general-purpose registers for data operations and can be accessed from either privileged or unprivileged mode.		
Type	R/W		

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

Bits	Field Name	Description	Type	Reset
31:0	DATA	Register data	RW	—

Cortex General-Purpose Register 4 (R4)

Address Offset		**Reset**	–
Physical Address		**Instance**	
Description	The Rn registers are 32-bit general-purpose registers for data operations and can be accessed from either privileged or unprivileged mode.		
Type	R/W		

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

Bits	Field Name	Description	Type	Reset
31:0	DATA	Register data	RW	—

Cortex General-Purpose Register 5 (R5)

Address Offset		**Reset**	–
Physical Address		**Instance**	
Description	The Rn registers are 32-bit general-purpose registers for data operations and can be accessed from either privileged or unprivileged mode.		
Type	R/W		

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

Bits	Field Name	Description	Type	Reset
31:0	DATA	Register data	RW	—

Cortex General-Purpose Register 6 (R6)

Address Offset		**Reset**	–
Physical Address		**Instance**	
Description	The Rn registers are 32-bit general-purpose registers for data operations and can be accessed from either privileged or unprivileged mode.		
Type	R/W		

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

Bits	Field Name	Description	Type	Reset
31:0	DATA	Register data	RW	—

Cortex General-Purpose Register 7 (R7)

Address Offset		**Reset**	–
Physical Address		**Instance**	
Description	The Rn registers are 32-bit general-purpose registers for data operations and can be accessed from either privileged or unprivileged mode.		
Type	R/W		

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

Bits	Field Name	Description	Type	Reset
31:0	DATA	Register data	RW	—

Cortex General-Purpose Register 8 (R8)

Address Offset		**Reset**	–
Physical Address		**Instance**	
Description	The Rn registers are 32-bit general-purpose registers for data operations and can be accessed from either privileged or unprivileged mode.		
Type	R/W		

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

Bits	Field Name	Description	Type	Reset
31:0	DATA	Register data	RW	—

Cortex General-Purpose Register 9 (R9)

Address Offset		**Reset**	–
Physical Address		**Instance**	
Description	The Rn registers are 32-bit general-purpose registers for data operations and can be accessed from either privileged or unprivileged mode.		
Type	R/W		

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

Bits	Field Name	Description	Type	Reset
31:0	DATA	Register data	RW	—

Cortex General-Purpose Register 10 (R10)

Address Offset		**Reset**	–
Physical Address		**Instance**	
Description	The Rn registers are 32-bit general-purpose registers for data operations and can be accessed from either privileged or unprivileged mode.		
Type	R/W		

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

Bits	Field Name	Description	Type	Reset
31:0	DATA	Register data	RW	—

Cortex General-Purpose Register 11 (R11)

Address Offset		**Reset**	–
Physical Address		**Instance**	
Description	The Rn registers are 32-bit general-purpose registers for data operations and can be accessed from either privileged or unprivileged mode.		
Type	R/W		

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

Bits	Field Name	Description	Type	Reset
31:0	DATA	Register data	RW	—

Cortex General-Purpose Register 12 (R12)

Address Offset		**Reset**	–
Physical Address		**Instance**	
Description	The Rn registers are 32-bit general-purpose registers for data operations and can be accessed from either privileged or unprivileged mode.		
Type	R/W		

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

Bits	Field Name	Description	Type	Reset
31:0	DATA	Register data	RW	—

Stack Pointer (SP)

Address Offset **Reset** –

Physical Address **Instance**

Description

The Stack Pointer (SP) is register R13. In thread mode, the function of this register changes depending on the ASP bit in the Control Register (CONTROL) register. When the ASP bit is clear, this register is the Main Stack Pointer (MSP). When the ASP bit is set, this register is the Process Stack Pointer (PSP). On reset, the ASP bit is clear, and the processor loads the MSP with the value from address 0x0000 0000. The MSP can only be accessed in privileged mode; the PSP can be accessed in either privileged or unprivileged mode.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	SP	This field is the address of the stack pointer.	RW	—

Link Register (LR)

Address Offset **Reset** 0xFFFF FFFF

Physical Address **Instance**

Description

The Link Register (LR) is register R14, and it stores the return information for subroutines, function calls, and exceptions. LR can be accessed from either privileged or unprivileged mode.

EXC_RETURN is loaded into LR on exception entry. See Table 5-3 for the values and description.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	LINK	This field is the return address.	RW	0xFFFF FFFF

Program Counter (PC)

Address Offset **Reset** –

Physical Address **Instance**

Description

The Program Counter (PC) is register R15, and it contains the current program address. On reset, the processor loads the PC with the value of the reset vector, which is at address 0x0000 0004. Bit 0 of the reset vector is loaded into the THUMB bit of the EPSR register at reset and must be 1. The PC register can be accessed in either privileged or unprivileged mode

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	PC	This field is the current program address.	RW	—

Table 2-3. PSR Register Combinations

Register	Type	Combination
PSR	R/W (1) (2)	APSR, EPSR, and IPSR
IEPSR	RO	EPSR and IPSR
IAPSR	R/W	APSR and IPSR
EAPSR	R/W	APSR and EPSR

(1) The processor ignores writes to the IPSR bits.

(2) Reads of the EPSR bits return 0, and the processor ignores writes to these bits.

Program Status Register (PSR)

Address Offset **Reset** 0x0100 0000

Physical Address **Instance**

Description

Note: This register is also referred to as xPSR.

The Program Status Register (PSR) has three functions, and the register bits are assigned to the different functions:

- Application Program Status Register (APSR), bits 31:27
- Execution Program Status Register (EPSR), bits 26:24, 15:10
- Interrupt Program Status Register (IPSR), bits 6:0

The PSR, IPSR, and EPSR registers can be accessed only in privileged mode; the APSR register can be accessed in privileged or unprivileged mode.

APSR contains the current state of the condition flags from previous instruction executions.

EPSR contains the Thumb state bit and the execution state bits for the If-Then (IT) instruction or the Interruptible-Continuable Instruction (ICI) field for an interrupted load multiple or store multiple instruction. Attempts to read the EPSR directly through application software using the MSR instruction always return 0. Attempts to write the EPSR using the MSR instruction in application software are always ignored. Fault handlers can examine the EPSR value in the stacked PSR to determine the operation that faulted (see Section 5.1.7).

IPSR contains the exception type number of the current ISR.

These registers can be accessed individually or as a combination of any two or all three registers, using the register name as an argument to the MSR or MRS instructions. For example, all of the registers can be read using PSR with the MRS instruction, or APSR only can be written to using APSR with the MSR instruction. Program Status Register (PSR) shows the possible register combinations for the PSR. See the MRS and MSR instruction descriptions in the *Cortex™-M3 Instruction Set Technical User's Manual* for more information about how to access the program status registers.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
N	Z	C	V	Q	ICI / IT		THUMB	RESERVED								ICI / IT						RESERVED		ISRNUM							

Bits	Field Name	Description	Type	Reset
31	N	APSR Negative or Less Flag	R/W	0
		Value **Description** 1 The previous operation result was negative or less than. 0 The previous operation result was positive, zero, greater than, or equal		
		The value of this bit is meaningful only when accessing PSR or APSR.		
30	Z	APSR Zero Flag	R/W	0
		Value **Description** 1 The previous operation result was zero.		

Bits	Field Name	Description	Type	Reset
		0 The previous operation result was nonzero. The value of this bit is meaningful only when accessing PSR or APSR.		
29	C	APSR Carry or Borrow Flag **Value** **Description** 1 The previous add operation resulted in a carry bit or the previous subtract operation did not result in a borrow bit. 0 The previous add operation did not result in a carry bit or the previous subtract operation resulted in a borrow bit. The value of this bit is meaningful only when accessing PSR or APSR.	R/W	0
28	V	APSR Overflow Flag **Value** **Description** 1 The previous operation resulted in an overflow. 0 The previous operation did not result in an overflow. The value of this bit is meaningful only when accessing PSR or APSR.	R/W	0
27	Q	APSR DSP Overflow and Saturation Flag **Value** **Description** 1 DSP overflow or saturation has occurred. 0 DSP overflow or saturation has not occurred since reset or since the bit was last cleared. The value of this bit is meaningful only when accessing PSR or APSR. This bit is cleared by software using an MRS instruction.	R/W	0
26:25	ICI / IT	EPSR ICI / IT status These bits, along with bits 15:10, contain the ICI field for an interrupted load multiple or store multiple instruction or the execution state bits of the IT instruction. When EPSR holds the ICI execution state, bits 26:25 are 0. The If-Then block contains up to four instructions following an IT instruction. Each instruction in the block is conditional. The conditions for the instructions are either all the same, or some can be the inverse of others. See the *Cortex™-M3 Instruction Set Technical User's Manual* for more information. The value of this field is meaningful only when accessing PSR or EPSR.	RO	0x0
24	THUMB	EPSR Thumb state This bit indicates the Thumb state and should always be set. The following can clear the THUMB bit: • The BLX, BX and POP{PC} instructions • Restoration from the stacked xPSR value on an exception return • Bit 0 of the vector value on an exception entry or reset Attempting to execute instructions when this bit is clear results in a fault or lockup. For more information, see Section 5.2.4. The value of this bit is meaningful only when accessing PSR or EPSR.	RO	1
23:16	RESERVED	Reserved	RO	0x00

Bits	Field Name	Description	Type	Reset
15:10	ICI / IT	EPSR ICI / IT status These bits, along with bits 26:25, contain the Interruptible-Continuable Instruction (ICI) field for an interrupted load multiple or store multiple instruction or the execution state bits of the IT instruction. When an interrupt occurs during the execution of an LDM, STM, PUSH, or POP instruction, the processor stops the load multiple or store multiple instruction operation temporarily and stores the next register operand in the multiple operation to bits 15:12. After servicing the interrupt, the processor returns to the register pointed to by bits 15:12 and resumes execution of the multiple load or store instruction. When EPSR holds the ICI execution state, bits 11:10 are 0. The If-Then block contains up to four instructions following a 16-bit IT instruction. Each instruction in the block is conditional. The conditions for the instructions are either all the same, or some can be the inverse of others. See the *Cortex™-M3 Instruction Set Technical User's Manual* for more information. The value of this field is meaningful only when accessing PSR or EPSR.	RO	0x0
9:7	RESERVED	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.	RO	0x0
6:0	ISRNUM	IPSR ISR Number This field contains the exception type number of the current ISR. **Value** **Description** 0x00 Thread mode 0x01 Reserved 0x02 NMI 0x03 Hard fault 0x04 Memory management fault 0x05 Bus fault 0x06 Usage fault 0x07-0x0A Reserved 0x0B SVCall 0x0C Reserved for debug 0x0D Reserved 0x0E PendSV 0x0F SysTick 0x10 Interrupt vector 0 0x11 Interrupt vector 1 ... ... 0x46 Interrupt vector 54 0x47-0x7F Reserved For more information, see Section 5.1.2. The value of this field is meaningful only when accessing PSR or IPSR.	RO	0x00

Priority Mask Register (PRIMASK)

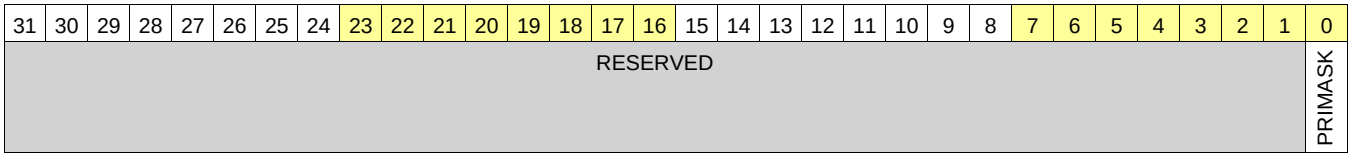
Address Offset **Reset** 0x0000 0000

Physical Address **Instance**

Description

The Priority Mask (PRIMASK) register prevents activation of all exceptions with programmable priority. Reset, nonmaskable interrupt (NMI), and hard fault are the only exceptions with fixed priority. Exceptions should be disabled when they might impact the timing of critical tasks. This register is accessible only in privileged mode. The MSR and MRS instructions are used to access the PRIMASK register, and the CPS instruction may be used to change the value of the PRIMASK register. For more information on these instructions, see the *Cortex™-M3 Instruction Set Technical User's Manual*. For more information on exception priority levels, see Section 5.1.2.

Type R/W



Bits	Field Name	Description	Type	Reset
31:1	RESERVED	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.	RO	0x0000 000
0	PRIMASK	Priority Mask **Value** **Description** 1 Prevents the activation of all exceptions with configurable priority. 0 No effect.	R/W	0

Fault Mask Register (FAULTMASK)

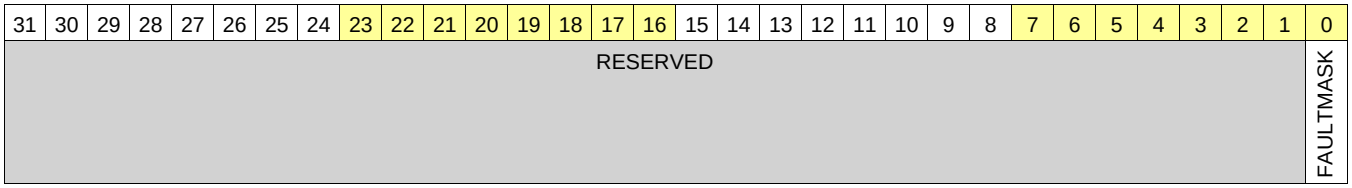
Address Offset **Reset** 0x0000 0000

Physical Address **Instance**

Description

The Fault Mask FAULTMASK register prevents activation of all exceptions except for the NMI. Exceptions should be disabled when they might impact the timing of critical tasks. This register is accessible only in privileged mode. The MSR and MRS instructions are used to access the FAULTMASK register, and the CPS instruction may be used to change the value of the FAULTMASK register. See the *Cortex™-M3 Instruction Set Technical User's Manual* for more information on these instructions. For more information on exception priority levels, see Section 5.1.2.

Type R/W



Bits	Field Name	Description	Type	Reset
31:1	RESERVED	Reserved	RO	0x0000 000
0	FAULTMASK	Fault Mask **Value** **Description** 1 Prevents the activation of all exceptions except for NMI. 0 No effect. The processor clears the FAULTMASK bit on exit from any exception handler except the NMI handler.	R/W	0

Base Priority Mask Register (BASEPRI)

Address Offset **Reset** 0x0000 0000

Physical Address **Instance**

Description

The Base Priority Mask BASEPRI register defines the minimum priority for exception processing. When BASEPRI is set to a nonzero value, it prevents the activation of all exceptions with the same or lower priority level as the BASEPRI value. Exceptions should be disabled when they might impact the timing of critical tasks. This register is accessible only in privileged mode. For more information on exception priority levels, see Section 5.1.2.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	Reserved	RO	0x0000 00
7:5	BASEPRI	Base Priority Any exception that has a programmable priority level with the same or lower priority as the value of this field is masked. The PRIMASK register can be used to mask all exceptions with programmable priority levels. Higher priority exceptions have lower priority levels. **Value** **Description** 0x0 All exceptions are unmasked. 0x1 All exceptions with priority levels 1–7 are masked. 0x2 All exceptions with priority levels 2–7 are masked. 0x3 All exceptions with priority levels 3–7 are masked. 0x4 All exceptions with priority levels 4–7 are masked. 0x5 All exceptions with priority levels 5–7 are masked. 0x6 All exceptions with priority levels 6 and 7 are masked. 0x7 All exceptions with priority level 7 are masked.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x0

Control Register (CONTROL)

Address Offset **Reset** 0x0000 0000

Physical Address **Instance**

Description

The CONTROL register controls the stack used and the privilege level for software execution when the processor is in thread mode. This register is accessible only in privileged mode.

Handler mode always uses MSP, so the processor ignores explicit writes to the ASP bit of the CONTROL register when in handler mode. The exception entry and return mechanisms automatically update the CONTROL register based on the EXC_RETURN value (see Table 5-3). In an OS environment, threads running in thread mode should use the process stack and the kernel and exception handlers should use the main stack. By default, thread mode uses MSP. To switch the stack pointer used in thread mode to PSP, either use the MSR instruction to set the ASP bit, as detailed in the *Cortex™-M3 Instruction Set Technical User's Manual*, or perform an exception return to thread mode with the appropriate EXC_RETURN value, as shown in Table 5-3.

Note: When changing the stack pointer, software must use an ISB instruction immediately after the MSR instruction, ensuring that instructions after the ISB instruction executes use the new stack pointer. See the *Cortex™-M3 Instruction Set Technical User's Manual.*

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	Reserved	RO	0x0000 000
1	ASP	Active Stack Pointer **Value** **Description** 1 PSP is the current stack pointer. 0 MSP is the current stack pointer In handler mode, this bit reads as zero and ignores writes. The Cortex-M3 updates this bit automatically on exception return.	R/W	0
0	TMPL	Thread Mode Privilege Level **Value** **Description** 1 Unprivileged software can be executed in thread mode. 0 Only privileged software can be executed in thread mode.	R/W	0

2.4.5 Exceptions and Interrupts

The Cortex-M3 processor supports interrupts and system exceptions. The processor and the NVIC prioritize and handle all exceptions. An exception changes the normal flow of software control. The processor uses handler mode to handle all exceptions except for reset. For more information, see Section 5.1.7.

The NVIC registers control interrupt handling. For more information, see Section 3.2.2.

2.4.6 Data Types

The Cortex-M3 supports 32-bit words, 16-bit halfwords, and 8-bit bytes. The processor also supports 64-bit data transfer instructions. All instruction and data memory accesses are little endian. For more information, see Section 4.1.1.

2.5 Instruction Set Summary

The processor implements a version of the Thumb instruction set. Table 2-4 lists the supported instructions.

NOTE: In Table 2-4:

- Angle brackets, <>, enclose alternative forms of the operand.
- Braces, {}, enclose optional operands.
- The Operands column is not exhaustive.
- Op2 is a flexible second operand that can be either a register or a constant.
- Most instructions can use an optional condition code suffix.

For more information on the instructions and operands, see the instruction descriptions in the *Cortex™-M3 Instruction Set Technical User's Manual*.

NOTE: This summary table is copied from the the *Cortex™-M3 Instruction Set Technical User's Manual*. Changes made in the manual must be made in Table 2-4.

Table 2-4. Cortex-M3 Instruction Summary

Mnemonic	Operands	Brief Description	Flags
ADC, ADCS	{Rd,} Rn, Op2	Add with carry	N, Z, C, V
ADD, ADDS	{Rd,} Rn, Op2	Add	N, Z, C, V
ADD, ADDW	{Rd,} Rn , #imm12	Add	N, Z, C, V
ADR	Rd, label	Load PC-relative address	–
AND, ANDS	{Rd,} Rn, Op2	Logical AND	N, Z, C
ASR, ASRS	Rd, Rm, <Rs|#n>	Arithmetic shift right	N, Z, C
B	label	Branch	–
BFC	Rd, #lsb, #width	Bit field clear	–
BFI	Rd, Rn, #lsb, #width	Bit field insert	–
BIC, BICS	{Rd,} Rn, Op2	Bit clear	N, Z, C
BKPT	#imm	Breakpoint	–
BL	label	Branch with link	–
BLX	Rm	Branch indirect with link	–
BX	Rm	Branch indirect	–
CBNZ	Rn, label	Compare and branch if nonzero	–
CBZ	Rn, label	Compare and branch if zero	–
CLREX	–	Clear exclusive	–

Table 2-4. Cortex-M3 Instruction Summary (continued)

Mnemonic	Operands	Brief Description	Flags
CLZ	Rd, Rm	Count leading zeros	–
CMN	Rn, Op2	Compare negative	N, Z, C, V
CMP	Rn, Op2	Compare	N, Z, C, V
CPSID	i	Change processor state, disable interrupts	–
CPSIE	i	Change processor state, enable interrupts	–
DMB	–	Data memory barrier	–
DSB	–	Data synchronization barrier	–
EOR, EORS	{Rd,} Rn, Op2	Exclusive OR	N, Z, C
ISB	–	Instruction synchronization barrier	–
IT	–	If-Then condition block	–
LDM	Rn{!}, reglist	Load multiple registers, increment after	–
LDMDB, LDMEA	Rn{!}, reglist	Load multiple registers, decrement before	–
LDMFD, LDMIA	Rn{!}, reglist	Load multiple registers, increment after	–
LDR	Rt, [Rn, #offset]	Load register with word	–
LDRB, LDRBT	Rt, [Rn, #offset]	Load register with byte	–
LDRD	Rt, Rt2, [Rn, #offset]	Load register with 2 bytes	–
LDREX	Rt, [Rn, #offset]	Load register exclusive	–
LDREXB	Rt, [Rn]	Load register exclusive with byte	–
LDREXH	Rt, [Rn]	Load register exclusive with halfword	–
LDRH, LDRHT	Rt, [Rn, #offset]	Load register with halfword	–
LDRSB, LDRSBT	Rt, [Rn, #offset]	Load register with signed byte	–
LDRSH, LDRSHT	Rt, [Rn, #offset]	Load register with signed halfword	–
LDRT	Rt, [Rn, #offset]	Load register with word	–
LSL, LSLS	Rd, Rm, <Rs|#n>	Logical shift left	N, Z, C
LSR, LSRS	Rd, Rm, <Rs|#n>	Logical shift right	N, Z, C
MLA	Rd, Rn, Rm, Ra	Multiply with accumulate, 32-bit result	–
MLS	Rd, Rn, Rm, Ra	Multiply and subtract, 32-bit result	–
MOV, MOVS	Rd, Op2	Move	N, Z, C
MOV, MOVW	Rd, #imm16	Move 16-bit constant	N, Z, C
MOVT	Rd, #imm16	Move top	–
MRS	Rd, spec_reg	Move from special register to general register	–
MSR	spec_reg, Rm	Move from general register to special register	N, Z, C, V
MUL, MULS	{Rd,} Rn, Rm	Multiply, 32-bit result	N, Z
MVN, MVNS	Rd, Op2	Move NOT	N, Z, C
NOP	–	No operation	–
ORN, ORNS	{Rd,} Rn, Op2	Logical OR NOT	N, Z, C
ORR, ORRS	{Rd,} Rn, Op2	Logical OR	N, Z, C
POP	reglist	Pop registers from stack	–

Table 2-4. Cortex-M3 Instruction Summary (continued)

Mnemonic	Operands	Brief Description	Flags
PUSH	reglist	Push registers onto stack	–
RBIT	Rd, Rn	Reverse bits	–
REV	Rd, Rn	Reverse byte order in a word	–
REV16	Rd, Rn	Reverse byte order in each halfword	–
REVSH	Rd, Rn	Reverse byte order in bottom halfword and sign extend	–
ROR, RORS	Rd, Rm, <Rs|#n>	Rotate right	N, Z, C
RRX, RRXS	Rd, Rm	Rotate right with extend	N, Z, C
RSB, RSBS	{Rd,} Rn, Op2	Reverse subtract	N, Z, C, V
SBC, SBCS	{Rd,} Rn, Op2	Subtract with carry	N, Z, C, V
SBFX	Rd, Rn, #lsb, #width	Signed bit field extract	–
SDIV	{Rd,} Rn, Rm	Signed divide	–
SEV	–	Send event	–
SMLAL	RdLo, RdHi, Rn, Rm	Signed multiply with accumulate (32 × 32 + 64), 64-bit result	–
SMULL	RdLo, RdHi, Rn, Rm	Signed multiply (32 × 32), 64-bit result	–
SSAT	Rd, #n, Rm {,shift #s}	Signed saturate	Q
STM	Rn{!}, reglist	Store multiple registers, increment after	–
STMDB, STMEA	Rn{!}, reglist	Store multiple registers, decrement before	–
STMFD, STMIA	Rn{!}, reglist	Store multiple registers, increment after	–
STR	Rt, [Rn {, #offset}]	Store register word	–
STRB, STRBT	Rt, [Rn {, #offset}]	Store register byte	–
STRD	Rt, Rt2, [Rn {, #offset}]	Store register two words	–
STREX	Rt, Rt, [Rn {, #offset}]	Store register exclusive	–
STREXB	Rd, Rt, [Rn]	Store register exclusive byte	–
STREXH	Rd, Rt, [Rn]	Store register exclusive halfword	–
STRH, STRHT	Rt, [Rn {, #offset}]	Store register halfword	–
STRSB, STRSBT	Rt, [Rn {, #offset}]	Store register signed byte	–
STRSH, STRSHT	Rt, [Rn {, #offset}]	Store register signed halfword	–
STRT	Rt, [Rn {, #offset}]	Store register word	–
SUB, SUBS	{Rd,} Rn, Op2	Subtract	N, Z, C, V
SUB, SUBW	{Rd,} Rn, #imm12	Subtract 12-bit constant	N, Z, C, V
SVC	#imm	Supervisor call	–
SXTB	{Rd,} Rm {,ROR #n}	Sign extend a byte	–
SXTH	{Rd,} Rm {,ROR #n}	Sign extend a halfword	–
TBB	[Rn, Rm]	Table branch byte	–
TBH	[Rn, Rm, LSL #1]	Table branch halfword	–
TEQ	Rn, Op2	Test equivalence	N, Z, C
TST	Rn, Op2	Test	N, Z, C
UBFX	Rd, Rn, #lsb, #width	Unsigned bit field extract	–
UDIV	{Rd,} Rn, Rm	Unsigned divide	–
UMLAL	RdLo, RdHi, Rn, Rm	Unsigned multiply with accumulate (32 × 32 + 32 + 32), 64-bit result	–

Table 2-4. Cortex-M3 Instruction Summary (continued)

Mnemonic	Operands	Brief Description	Flags
UMULL	RdLo, RdHi, Rn, Rm	Unsigned multiply (32 × 2), 64-bit result	–
USAT	Rd, #n, Rm {,shift #s}	Unsigned saturate	Q
UXTB	{Rd,} Rm, {,ROR #n}	Zero extend a byte	–
UXTH	{Rd,} Rm, {,ROR #n}	Zero extend a halfword	–
USAT	Rd, #n, Rm {,shift #s}	Unsigned saturate	Q
UXTB	{Rd,} Rm {,ROR #n}	Zero extend a byte	–
UXTH	{Rd,} Rm {,ROR #n}	Zero extend a halfword	–
WFE	–	Wait for event	–
WFI	–	Wait for interrupt	–

Cortex™-M3 Peripherals

This chapter describes the Cortex™-M3 peripherals.

Topic **Page**

3.1 Cortex™-M3 Peripherals Introduction

This chapter provides information on the CC2538 implementation of the Cortex-M3 processor peripherals, including:

- System timer (SysTick) (see SysTick): Provides a simple, 24-bit clear-on-write, decrementing, wrap-on-zero counter with a flexible control mechanism.
- Nested vectored interrupt controller (NVIC) (see NVIC):
  - Facilitates low-latency exception and interrupt handling
  - Works with system controller (see Chapter 7) to control power management
  - Implements system control registers
- M3 system control block (SCB) (see SCB): Provides system implementation information and system control, including configuration, control, and reporting of system exceptions.
- Memory protection unit (MPU) (see MPU): Supports the standard ARMv7 protected memory system architecture (PMSA) model. The MPU provides full support for protection regions, overlapping protection regions, access permissions, and exporting memory attributes to the system.

Table 3-1 lists the address map of the private peripheral bus (PPB). Some peripheral register regions are split into two address regions, as indicated by two addresses listed.

Table 3-1. Core Peripheral Register Regions

ADDRESS	CORE PERIPHERAL	LINK
0xE000 E010 – 0xE000 E01F	System Timer (SysTick)	SysTick
0xE000 E100 – 0xE000 E4EF 0xE000 EF00 – 0xE000 EF03	Nested Vectored Interrupt Controller (NVIC)	NVIC
0xE000 E008 – 0xE000 E00F 0xE000 ED00 – 0xE000 ED3F	System Control Block (SCB)	SCB
0xE000 ED90 – 0xE000 EDB8	Memory Protection Unit (MPU)	MPU

3.2 Functional Description

This chapter provides information on the CC2538 implementation of the Cortex-M3 processor peripherals:

- SysTick
- NVIC
- SCB
- MPU

3.2.1 SysTick

Cortex-M3 includes an integrated system timer, SysTick, which provides a simple, 24-bit clear-on-write, decrementing, wrap-on-zero counter with a flexible control mechanism. The counter can be used in several different ways. For example, the counter can be:

- An RTOS tick timer that fires at a programmable rate (for example, 100 Hz) and invokes a SysTick routine.
- A high-speed alarm timer using the system clock.
- A variable rate alarm or signal timer—the duration is range-dependent on the reference clock used and the dynamic range of the counter.
- A simple counter used to measure time to completion and time used.
- An internal clock source control based on missing and/or meeting durations. The **COUNT** bit in the **STCTRL** control and status register can be used to determine if an action completed within a set duration, as part of a dynamic clock management control loop.

The timer consists of three registers:

- **SysTick Control and Status (STCTRL)**: A control and status counter to configure its clock, enable the counter, enable the SysTick interrupt, and determine counter status

- **SysTick Reload Value (STRELOAD)**: The reload value for the counter, used to provide the wrap value of the counter
- **SysTick Current Value (STCURRENT)**: The current value of the counter

When enabled, the timer counts down on each clock from the reload value to 0, reloads (wraps) to the value in the **STRELOAD** register on the next clock edge, then decrements on subsequent clocks. Clearing the **STRELOAD** register disables the counter on the next wrap. When the counter reaches 0, the **COUNT** status bit is set. The **COUNT** bit clears on reads.

Writing to the **STCURRENT** register clears the register and the **COUNT** status bit. The write does not trigger the SysTick exception logic. On a read, the current value is the value of the register at the time the register is accessed.

The SysTick counter runs on the system clock. If this clock signal is stopped for low-power mode, the SysTick counter stops. Ensure that software uses aligned word accesses to access the **SysTick** registers.

NOTE: When the processor is halted for debugging, the counter does not decrement.

3.2.2 NVIC

This section describes the NVIC and the registers it uses. The NVIC supports:

- 48 interrupts in alternate interrupt mapping mode.
- 147 interrupt in regular interrupt mapping mode.
- A programmable priority level of 0 to 7 for each interrupt. A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority.
- Low-latency exception and interrupt handling
- Level and pulse detection of interrupt signals
- Dynamic reprioritization of interrupts
- Grouping of priority values into group priority and sub priority fields
- Interrupt tail-chaining
- An external nonmaskable interrupt (NMI)

The processor automatically stacks its state on exception entry and un-stacks this state on exception exit, with no instruction overhead, providing low latency exception handling.

3.2.2.1 Level-Sensitive and Pulse Interrupts

The processor supports both level-sensitive and pulse interrupts. Pulse interrupts are also described as edge-triggered interrupts.

A level-sensitive interrupt is held asserted until the peripheral de-asserts the interrupt signal. Typically this happens because the interrupt service routine (ISR) accesses the peripheral, causing it to clear the interrupt request. A pulse interrupt is an interrupt signal sampled synchronously on the rising edge of the processor clock. To ensure the NVIC detects the interrupt, the peripheral must assert the interrupt signal for at least one clock cycle, during which the NVIC detects the pulse and latches the interrupt.

When the processor enters the ISR, it automatically removes the pending state from the interrupt (for more information, see Hardware and Software Control of Interrupts). For a level-sensitive interrupt, if the signal is not de-asserted before the processor returns from the ISR, the interrupt becomes pending again, and the processor must execute its ISR again. As a result, the peripheral can hold the interrupt signal asserted until it no longer needs servicing.

3.2.2.2 Hardware and Software Control of Interrupts

The Cortex-M3 processor latches all interrupts. A peripheral interrupt becomes pending for one of the following reasons:

- The NVIC detects that the interrupt signal is asserted and the interrupt is not active.
- The NVIC detects a rising edge on the interrupt signal.

- Software writes to the corresponding interrupt set-pending register bit, or to the **Software Trigger Interrupt (SWTRIG)** register to make a software-generated interrupt pending. See the **INT** bit in the **PEND0** register in Interrupt 0–31 Set Pending (PEND0), or **SWTRIG** in Software Trigger Interrupt (SWTRIG).

A pending interrupt remains pending until one of the following:

- The processor enters the ISR for the interrupt, changing the state of the interrupt from pending to active. Then:
  - For a level-sensitive interrupt, when the processor returns from the ISR, the NVIC samples the interrupt signal. If the signal is asserted, the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR. Otherwise, the state of the interrupt changes to inactive.
  - For a pulse interrupt, the NVIC continues to monitor the interrupt signal, and if this is pulsed the state of the interrupt changes to pending and active. In this case, when the processor returns from the ISR the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR. If the interrupt signal is not pulsed while the processor is in the ISR, when the processor returns from the ISR the state of the interrupt changes to inactive.
- Software writes to the corresponding interrupt clear-pending register bit:
  - For a level-sensitive interrupt, if the interrupt signal is still asserted, the state of the interrupt does not change. Otherwise, the state of the interrupt changes to inactive.
  - For a pulse interrupt, the state of the interrupt changes to inactive if the state was pending, or to active if the state was active and pending.

3.2.3 SCB

The SCB provides system implementation information and system control, including configuration, control, and reporting of the system exceptions.

3.2.4 MPU

This section describes the MPU. The MPU divides the memory map into a number of regions and defines the location, size, access permissions, and memory attributes of each region. The MPU supports independent attribute settings for each region, overlapping regions, and export of memory attributes to the system.

The memory attributes affect the behavior of memory accesses to the region. The Cortex-M3 MPU defines eight separate memory regions, 0 to 7, and a background region.

When memory regions overlap, a memory access is affected by the attributes of the region with the highest number. For example, the attributes for region 7 take precedence over the attributes of any region that overlaps region 7.

The background region has the same memory access attributes as the default memory map, but is accessible from privileged software only.

The Cortex-M3 MPU memory map is unified, meaning that instruction accesses and data accesses have the same region settings.

If a program accesses a memory location that is prohibited by the MPU, the processor generates a memory management fault, causing a fault exception and possibly causing termination of the process in an OS environment. In an OS environment, the kernel can update the MPU region setting dynamically based on the process to be executed. Typically, an embedded OS uses the MPU for memory protection.

Configuration of MPU regions is based on memory types (for more information, see Section 4.1.1).

Table 3-2 shows the possible MPU region attributes. For guidelines for programming a microcontroller implementation, see MPU Configuration for a CC2538 Microcontroller.

Table 3-2. Memory Attributes Summary

MEMORY TYPE	DESCRIPTION
Strongly ordered	All accesses to strongly ordered memory occur in program order.
Device	Memory-mapped peripherals
Normal	Normal memory

To avoid unexpected behavior, disable the interrupts before updating the attributes of a region that the interrupt handlers might access.

Ensure that software uses aligned accesses of the correct size to access MPU registers:

- Except for the **MPU Region Attribute and Size (MPUATTR)** register, all MPU registers must be accessed with aligned word accesses.
- The **MPUATTR** register can be accessed with byte or aligned halfword or word accesses.

The processor does not support unaligned accesses to MPU registers.

When setting up the MPU, and if the MPU has previously been programmed, disable unused regions to prevent any previous region settings from affecting the new MPU setup.

3.2.4.1 Updating an MPU Region

To update the attributes for an MPU region, the **MPU Region Number (MPUNUMBER)**, **MPU Region Base Address (MPUBASE)** and **MPUATTR** registers must be updated. Each register can be programmed separately or with a multiple-word write to program all of these registers. The **MPUBASEx** and **MPUATTRx** aliases can be used to program up to four regions simultaneously using an **STM** instruction.

3.2.4.1.1 Updating an MPU Region Using Separate Words

This example simple code configures one region:

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
LDR R0,=MPUNUMBER        ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]       ; Region Number
STR R4, [R0, #0x4]       ; Region Base Address
STRH R2, [R0, #0x8]      ; Region Size and Enable
STRH R3, [R0, #0xA]      ; Region Attribute
```

Disable a region before writing new region settings to the MPU if the region being changed was previously enabled. For example:

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
LDR R0,=MPUNUMBER         ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]        ; Region Number
BIC R2, R2, #1            ; Disable
STRH R2, [R0, #0x8]       ; Region Size and Enable
STR R4, [R0, #0x4]        ; Region Base Address
STRH R3, [R0, #0xA]       ; Region Attribute
ORR R2, #1                ; Enable
STRH R2, [R0, #0x8]       ; Region Size and Enable
```

Software must use memory barrier instructions:

- Before MPU setup, if there might be outstanding memory transfers, such as buffered writes, that might be affected by the change in MPU settings.
- After MPU setup, if it includes memory transfers that must use the new MPU settings.

However, memory barrier instructions are not required if the MPU setup process starts by entering an exception handler, or is followed by an exception return, because the exception entry and exception return mechanisms cause memory barrier behavior.

Software does not need any memory barrier instructions during MPU setup, because it accesses the MPU through the PPB, which is a strongly ordered memory region.

For example, if all of the memory access behavior is intended to take effect immediately after the programming sequence, then a **DSB** instruction and an **ISB** instruction should be used. A **DSB** is required after changing MPU settings, such as at the end of context switch. An **ISB** is required if the code that programs the MPU region or regions is entered using a branch or call. If the programming sequence is entered using a return from exception, or by taking an exception, then an **ISB** is not required.

3.2.4.1.2 Updating an MPU Region Using Multiple-Word Writes

The MPU can be programmed directly using multiple-word writes, depending how the information is divided. Consider the following reprogramming:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
LDR R0, =MPUNUMBER  ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]  ; Region Number
STR R2, [R0, #0x4]  ; Region Base Address
STR R3, [R0, #0x8]  ; Region Attribute, Size and Enable
```

An **STM** instruction can be used to optimize this:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
LDR R0, =MPUNUMBER  ; 0xE000ED98, MPU region number register
STM R0, {R1-R3}     ; Region number, address, attribute, size and enable
```

This operation can be done in two words for prepacked information, meaning that the **MPU Region Base Address (MPUBASE)** register (see MPU Region Base Address (MPUBASE)) contains the required region number and has the **VALID** bit set. This method can be used when the data is statically packed, for example in a boot loader:

```
; R1 = address and region number in one
; R2 = size and attributes in one
LDR R0, =MPUBASE    ; 0xE000ED9C, MPU Region Base register
STR R1, [R0, #0x0]  ; Region base address and region number combined
                    ; with VALID (bit 4) set
STR R2, [R0, #0x4]  ; Region Attribute, Size and Enable
```

3.2.4.1.3 Subregions

Regions of 256 bytes or more are divided into eight equal-sized subregions. Set the corresponding bit in the **SRD** field of the **MPU Region Attribute and Size (MPUATTR)** register (see MPU Region Attribute and Size (MPUATTR)) to disable a subregion. The least-significant bit (LSB) of the **SRD** field controls the first subregion, and the most-significant bit (MSB) controls the last subregion. Disabling a subregion means another region overlapping the disabled range matches instead. If no other enabled region overlaps the disabled subregion, the MPU issues a fault.

Regions of 32, 64, and 128 bytes do not support subregions. With regions of these sizes, the **SRD** field must be configured to 0x00, otherwise the MPU behavior is unpredictable.

3.2.4.1.3.1 Example of SRD Use

Two regions with the same base address overlap. Region 1 is 128KB, and region 2 is 512KB. To ensure the attributes from region 1 apply to the first 128-KB region, configure the **SRD** field for region 2 to 0x03 to disable the first two subregions, as Figure 3-1 shows.

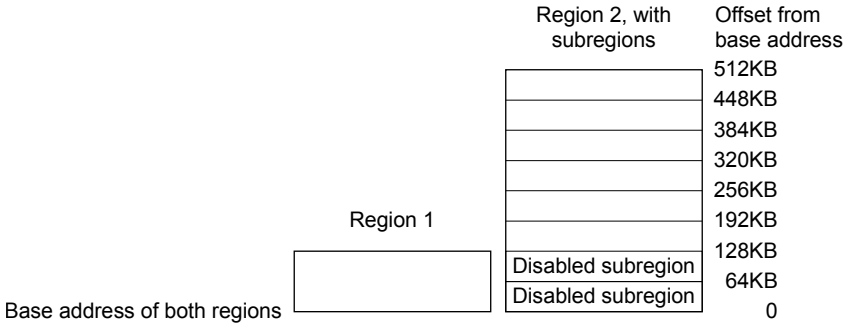


Figure 3-1. SRD Use Example

3.2.4.2 MPU Access Permission Attributes

The access permission bits, **TEX**, **S**, **C**, **B**, **AP**, and **XN** of the **MPUATTR** register, control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, then the MPU generates a permission fault.

Table 3-3 lists the encodings for the **TEX**, **C**, **B**, and **S** access permission bits. All encodings are shown for completeness; however, the current implementation of the Cortex-M3 does not support the concept of cacheability or shareability. For information on programming the MPU for CC2538 implementations, see MPU Configuration for a CC2538 Microcontroller.

Table 3-3. TEX, S, C, and B Bit Field Encoding

TEX	S	C	B	MEMORY TYPE	SHAREABILITY	OTHER ATTRIBUTES
000b	x (1)	0	0	Strongly Ordered	Shareable	–
000	x	0	1	Device	Shareable	–
000	0	1	0	Normal	Not shareable	Outer and inner write-through. No write allocate.
000	1	1	0	Normal	Shareable	
000	0	1	1	Normal	Not shareable	
000	1	1	1	Normal	Shareable	
001	0	0	0	Normal	Not shareable	Outer and inner noncacheable
001	1	0	0	Normal	Shareable	
001	x	0	1	Reserved encoding	–	–
001	x	1	0	Reserved encoding	–	–
001	0	1	1	Normal	Not shareable	Outer and inner write-back. Write and read allocate.
001	1	1	1	Normal	Shareable	
010	x	0	0	Device	Not shareable	Nonshared device
010	x	0	1	Reserved encoding	–	–
010	x	1	x	Reserved encoding	–	–
1BB	0	A	A	Normal	Not shareable	Cached memory (BB = outer policy, AA = inner policy). See Table 3-4 for the encoding of the AA and BB bits.
1BB	1	A	A	Normal	Shareable	

(1) The MPU ignores the value of this bit.

Table 3-4 lists the cache policy for memory attribute encodings with a **TEX** value in the range of 0x4 to 0x7.

Table 3-4. Cache Policy for Memory Attribute Encoding

ENCODING, AA or BB	CORRESPONDING CACHE POLICY
00	Noncacheable
01	Write back, write and read allocate
10	Write through, no write allocate
11	Write back, no write allocate

Table 3-5 lists the **AP** encodings in the **MPUATTR** register that define the access permissions for privileged and unprivileged software.

Table 3-5. AP Bit Field Encoding

AP BIT FIELD	PRIVILEGED PERMISSIONS	UNPRIVILEGED PERMISSIONS	DESCRIPTION
000	No access	No access	All accesses generate a permission fault.
001	R/W	No access	Access from privileged software only
010	R/W	RO	Writes by unprivileged software generate a permission fault.
011	R/W	R/W	Full access
100	Unpredictable	Unpredictable	Reserved
101	RO	No access	Reads by privileged software only
110	RO	RO	Read-only, by privileged or unprivileged software
111	RO	RO	Read-only, by privileged or unprivileged software

3.2.4.2.1 MPU Configuration for a CC2538 Microcontroller

CC2538 microcontrollers have only a single processor and no caches. As a result, the MPU should be programmed as shown in Table 3-6.

Table 3-6. Memory Region Attributes for a CC2538 Microcontroller

MEMORY REGION	TEX	C	B	S	MEMORY TYPE AND ATTRIBUTES
Flash memory	b000	1	0	0	Normal memory, nonshareable, write-through
Internal SRAM	b000	1	0	1	Normal memory, shareable, write-through
External SRAM	b000	1	1	1	Normal memory, shareable, write-back, write-allocate
Peripherals	b000	0	1	1	Device memory, shareable

In current CC2538 microcontroller implementations, the shareability and cache policy attributes do not affect the system behavior. However, using these settings for the MPU regions can make the application code more portable. The values given in Table 3-6 are for typical situations.

3.2.4.3 MPU Mismatch

When an access violates the MPU permissions, the processor generates a memory management fault (for more information, see Section 2.4.5). The **MFAULTSTAT** register indicates the cause of the fault. For more information, see Configurable Fault Status (FAULTSTAT).

3.3 Register Map

Table 3-7 lists the Cortex-M3 Peripheral SysTick, NVIC, MPU, and SCB registers. The offset listed is a hexadecimal increment to the register address, relative to the core peripherals base address of 0xE000 E000 (ending address of 0xE000 EFFF).

NOTE: Register spaces that are not used are reserved for future or internal use. Software should not modify any reserved memory address.

Table 3-7. Peripherals Register Map

OFFSET	NAME	TYPE	RESET	DESCRIPTION	LINK
SysTick Registers					
0x010	STCTRL	R/W	0x0000 0004	SysTick Control and Status Register	SysTick Control and Status Register (STCTRL)
0x014	STRELOAD	R/W	0x0000 0000	SysTick Reload Value Register	SysTick Reload Value Register (STRELOAD)
0x018	STCURRENT	R/WC	0x0000 0000	SysTick Current Value Register	SysTick Current Value Register (STCURRENT)
NVIC Registers					
0x100	EN0	R/W	0x0000 0000	Interrupt 0–31 Set Enable	Interrupt 0–31 Set Enable (EN0)
0x104	EN1	R/W	0x0000 0000	Interrupt 32–63 Set Enable	Interrupt 32–63 Set Enable (EN1)
0x108	EN2	R/W	0x0000 0000	Interrupt 64–95 Set Enable	Interrupt 64–95 Set Enable (EN2)
0x10C	EN3	R/W	0x0000 0000	Interrupt 96–127 Set Enable	Interrupt 96–127 Set Enable (EN3)
0x110	EN4	R/W	0x0000 0000	Interrupt 128–147 Set Enable	Interrupt 128–147 Set Enable (EN4)
0x180	DIS0	R/W	0x0000 0000	Interrupt 0–31 Clear Enable	Interrupt 0–31 Clear Enable (DIS0)
0x184	DIS1	R/W	0x0000 0000	Interrupt 32–63 Clear Enable	Interrupt 32–63 Clear Enable (DIS1)
0x188	DIS2	R/W	0x0000 0000	Interrupt 64–95 Clear Enable	Interrupt 64–95 Clear Enable (DIS2)
0x18C	DIS3	R/W	0x0000 0000	Interrupt 96–127 Clear Enable	Interrupt 96–127 Clear Enable (DIS3)
0x190	DIS4	R/W	0x0000 0000	Interrupt 128–147 Clear Enable	Interrupt 128–147 Clear Enable (DIS4)
0x200	PEND0	R/W	0x0000 0000	Interrupt 0–31 Set Pending	Interrupt 0–31 Set Pending (PEND0)
0x204	PEND1	R/W	0x0000 0000	Interrupt 32–63 Set Pending	Interrupt 32–63 Set Pending (PEND1)
0x208	PEND2	R/W	0x0000 0000	Interrupt 64–95 Set Pending	Interrupt 64–95 Set Pending (PEND2)
0x20C	PEND3	R/W	0x0000 0000	Interrupt 96–127 Set Pending	Interrupt 96–127 Set Pending (PEND3)
0x210	PEND4	R/W	0x0000 0000	Interrupt 128–147 Set Pending	Interrupt 128–147 Set Pending (PEND4)
0x280	UNPEND0	R/W	0x0000 0000	Interrupt 0–31 Clear Pending	Interrupt 0–31 Clear Pending (UNPEND0)
0x284	UNPEND1	R/W	0x0000 0000	Interrupt 32–63 Clear Pending	Interrupt 32–63 Clear Pending (UNPEND1)
0x288	UNPEND2	R/W	0x0000 0000	Interrupt 64–95 Clear Pending	Interrupt 64–95 Clear Pending (UNPEND2)
0x28C	UNPEND3	R/W	0x0000 0000	Interrupt 96–127 Clear Pending	Interrupt 96–127 Clear Pending (UNPEND3)
0x290	UNPEND4	R/W	0x0000 0000	Interrupt 128–147 Clear Pending	Interrupt 128–147 Clear Pending (UNPEND4)
0x300	ACTIVE0	RO	0x0000 0000	Interrupt 0–31 Active Bit	Interrupt 0–31 Active Bit (ACTIVE0)
0x304	ACTIVE1	RO	0x0000 0000	Interrupt 32–63 Active Bit	Interrupt 32–63 Active Bit (ACTIVE1)

Table 3-7. Peripherals Register Map (continued)

OFFSET	NAME	TYPE	RESET	DESCRIPTION	LINK
0x308	ACTIVE2	RO	0x0000 0000	Interrupt 64–95 Active Bit	Interrupt 64–95 Active Bit (ACTIVE2)
0x30C	ACTIVE3	RO	0x0000 0000	Interrupt 96–127 Active Bit	Interrupt 96–127 Active Bit (ACTIVE3)
0x310	ACTIVE4	RO	0x0000 0000	Interrupt 128–147 Active Bit	Interrupt 128–147 Active Bit (ACTIVE4)
0x400	PRI0	R/W	0x0000 0000	Interrupt 0–3 Priority	Interrupt 0–3 Priority (PRI0)
0x404	PRI1	R/W	0x0000 0000	Interrupt 4–7 Priority	Interrupt 4–7 Priority (PRI1)
0x408	PRI2	R/W	0x0000 0000	Interrupt 8–11 Priority	Interrupt 8–11 Priority (PRI2)
0x40C	PRI3	R/W	0x0000 0000	Interrupt 12–15 Priority	Interrupt 12–15 Priority (PRI3)
0x410	PRI4	R/W	0x0000 0000	Interrupt 16–19 Priority	Interrupt 16–19 Priority (PRI4)
0x414	PRI5	R/W	0x0000 0000	Interrupt 20–23 Priority	Interrupt 20–23 Priority (PRI5)
0x418	PRI6	R/W	0x0000 0000	Interrupt 24–27 Priority	Interrupt 24–27 Priority (PRI6)
0x41C	PRI7	R/W	0x0000 0000	Interrupt 28–31 Priority	Interrupt 28–31 Priority (PRI7)
0x420	PRI8	R/W	0x0000 0000	Interrupt 32–35 Priority	Interrupt 32–35 Priority (PRI8)
0x424	PRI9	R/W	0x0000 0000	Interrupt 36–39 Priority	Interrupt 36–39 Priority (PRI9)
0x428	PRI10	R/W	0x0000 0000	Interrupt 40–43 Priority	Interrupt 40–43 Priority (PRI10)
0x42C	PRI11	R/W	0x0000 0000	Interrupt 44–47 Priority	Interrupt 44–47 Priority (PRI11)
0x430	PRI12	R/W	0x0000 0000	Interrupt 48–51 Priority	Interrupt 48–51 Priority (PRI12)
0x434	PRI13	R/W	0x0000 0000	Interrupt 52–55 Priority	Interrupt 52–55 Priority (PRI13)
0x438	PRI14	R/W	0x0000 0000	Interrupt 56–59 Priority	Interrupt 56–59 Priority (PRI14)
0x43C	PRI15	R/W	0x0000 0000	Interrupt 60–63 Priority	Interrupt 60–63 Priority (PRI15)
0x440	PRI16	R/W	0x0000 0000	Interrupt 64–67 Priority	Interrupt 64–67 Priority (PRI16)
0x444	PRI17	R/W	0x0000 0000	Interrupt 68–71 Priority	Interrupt 68–71 Priority (PRI17)
0x448	PRI18	R/W	0x0000 0000	Interrupt 72–75 Priority	Interrupt 72–75 Priority (PRI18)
0x44C	PRI19	R/W	0x0000 0000	Interrupt 76–79 Priority	Interrupt 76–75 Priority (PRI19)
0x450	PRI20	R/W	0x0000 0000	Interrupt 80–83 Priority	Interrupt 80–83 Priority (PRI20)
0x454	PRI21	R/W	0x0000 0000	Interrupt 84–87 Priority	Interrupt 84–87 Priority (PRI21)
0x458	PRI22	R/W	0x0000 0000	Interrupt 88–91 Priority	Interrupt 88–91 Priority (PRI22)
0x45C	PRI23	R/W	0x0000 0000	Interrupt 92–95 Priority	Interrupt 92–95 Priority (PRI23)
0x460	PRI24	R/W	0x0000 0000	Interrupt 96–99 Priority	Interrupt 96–99 Priority (PRI24)
0x464	PRI25	R/W	0x0000 0000	Interrupt 100–103 Priority	Interrupt 100–103 Priority (PRI25)
0x468	PRI26	R/W	0x0000 0000	Interrupt 104–107 Priority	Interrupt 104–107 Priority (PRI26)
0x46C	PRI27	R/W	0x0000 0000	Interrupt 108–111 Priority	Interrupt 108–111 Priority (PRI27)
0x470	PRI28	R/W	0x0000 0000	Interrupt 112–115 Priority	Interrupt 112–115 Priority (PRI28)
0x474	PRI29	R/W	0x0000 0000	Interrupt 116–119 Priority	Interrupt 116–119 Priority (PRI29)
0x478	PRI30	R/W	0x0000 0000	Interrupt 120–123 Priority	Interrupt 120–123 Priority (PRI30)
0x47C	PRI31	R/W	0x0000 0000	Interrupt 124–127 Priority	Interrupt 124–127 Priority (PRI31)
0x480	PRI32	R/W	0x0000 0000	Interrupt 128–131 Priority	Interrupt 128–131 Priority (PRI32)
0x484	PRI33	R/W	0x0000 0000	Interrupt 132–135 Priority	Interrupt 132–135 Priority (PRI33)
0x488	PRI34	R/W	0x0000 0000	Interrupt 136–139 Priority	Interrupt 136–139 Priority (PRI34)

Table 3-7. Peripherals Register Map (continued)

OFFSET	NAME	TYPE	RESET	DESCRIPTION	LINK
0x48C	PRI35	R/W	0x0000 0000	Interrupt 140–143 Priority	Interrupt 140–143 Priority (PRI35)
0x490	PRI36	R/W	0x0000 0000	Interrupt 144–147 Priority	Interrupt 144–147 Priority (PRI36)
0xF00	SWTRIG	WO	0x0000 0000	Software Trigger Interrupt	Software Trigger Interrupt (SWTRIG)
SCB Registers					
0x008	ACTLR	R/W	0x0000 0000	Auxiliary Control	Auxiliary Control (ACTLR)
0xD00	CPUID	RO	0x412F C230	CPU ID Base	CPU ID Base (CPUID)
0xD04	INTCTRL	R/W	0x0000 0000	Interrupt Control and State	Interrupt Control and State (INTCTRL)
0xD08	VTABLE	R/W	0x0000 0000	Vector Table Offset	Vector Table Offset (VTABLE)
0xD0C	APINT	R/W	0xFA05 0000	Application Interrupt and Reset Control	Application Interrupt and Reset Control (APINT)
0xD10	SYSCTRL	R/W	0x0000 0000	System Control	System Control (SYSCTRL)
0xD14	CFGCTRL	R/W	0x0000 0200	Configuration and Control	Configuration and Control (CFGCTRL)
0xD18	SYSPRI1	R/W	0x0000 0000	System Handler Priority 1	System Handler Priority 1 (SYSPRI1)
0xD1C	SYSPRI2	R/W	0x0000 0000	System Handler Priority 2	System Handler Priority 2 (SYSPRI2)
0xD20	SYSPRI3	R/W	0x0000 0000	System Handler Priority 3	System Handler Priority 3 (SYSPRI3)
0xD24	SYSHNDCTRL	R/W	0x0000 0000	System Handler Control and State	System Handler Control and State (SYSHNDCTRL)
0xD28	FAULTSTAT	R/W1C	0x0000 0000	Configurable Fault Status	Configurable Fault Status (FAULTSTAT)
0xD2C	HFAULTSTAT	R/W1C	0x0000 0000	Hard Fault Status	Hard Fault Status (HFAULTSTAT)
0xD34	MMADDR	R/W	–	Memory Management Fault Address	Memory Management Fault Address (MMADDR)
0xD38	FAULTADDR	R/W	–	Bus Fault Address	Bus Fault Address (FAULTADDR)
MPU Registers					
0xD90	MPUTYPE	RO	0x0000 0800	MPU Type	MPU Type (MPUTYPE)
0xD94	MPUCTRL	R/W	0x0000 0000	MPU Control	MPU Control (MPUCTRL)
0xD98	MPUNUMBER	R/W	0x0000 0000	MPU Region Number	MPU Region Number (MPUNUMBER)
0xD9C	MPUBASE	R/W	0x0000 0000	MPU Region Base Address	MPU Region Base Address (MPUBASE)
0xDA0	MPUATTR	R/W	0x0000 0000	MPU Region Attribute and Size	MPU Region Attribute and Size (MPUATTR)
0xDA4	MPUBASE1	R/W	0x0000 0000	MPU Region Base Address Alias 1	MPU Region Base Address Alias 1 (MPUBASE1)
0xDA8	MPUATTR1	R/W	0x0000 0000	MPU Region Attribute and Size Alias 1	MPU Region Attribute and Size Alias 1 (MPUATTR1), Offset 0xDA8
0xDAC	MPUBASE2	R/W	0x0000 0000	MPU Region Base Address Alias 2	MPU Region Base Address Alias 2 (MPUBASE2)
0xDB0	MPUATTR2	R/W	0x0000 0000	MPU Region Attribute and Size Alias 2	MPU Region Attribute and Size Alias 2 (MPUATTR2)
0xDB4	MPUBASE3	R/W	0x0000 0000	MPU Region Base Address Alias 3	MPU Region Base Address Alias 3 (MPUBASE3)
0xDB8	MPUATTR3	R/W	0x0000 0000	MPU Region Attribute and Size Alias 3	MPU Region Attribute and Size Alias 3 (MPUATTR3)

3.4 SysTick Register Descriptions

This section lists and describes the System Timer registers, in numerical order by address offset.

SysTick Control and Status Register (STCTRL)

Address Offset	0x010	**Reset**	0x0000 0004
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The SysTick **STCTRL** register enables the SysTick features.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED															COUNT	RESERVED													CLK_SRC	INTEN	ENABLE

Bits	Field Name	Description	Type	Reset
31:17	RESERVED	Reserved	RO	0x000
16	COUNT	Count flag **Value** **Description** 0 The SysTick timer has not counted to 0 since the last time this bit was read. 1 The SysTick timer has counted to 0 since the last time this bit was read. This bit is cleared by a read of the register or if the **STCURRENT** register is written with any value. If read by the debugger using the DAP, this bit is cleared only if the **MasterType** bit in the **AHB-AP Control Register** is clear. Otherwise, the **COUNT** bit is not changed by the debugger read. For more information on **MasterType**, see the *ARM Debug Interface V5 Architecture Specification*.	RO	0
15:3	RESERVED	Reserved	RO	0x000
2	CLK_SRC	Clock source **Value** **Description** 0 External reference clock (not implemented for CC2538 microcontrollers) 1 System clock Because an external reference clock is not implemented, this bit must be set in order for SysTick to operate.	R/W	1
1	INTEN	Interrupt enable **Value** **Description** 0 Interrupt generation is disabled. Software can use the **COUNT** bit to determine if the counter has ever reached 0. 1 An interrupt is generated to the NVIC when SysTick counts to 0.	R/W	0
0	ENABLE	Enable **Value** **Description** 0 The counter is disabled. 1 Enables SysTick to operate in a multiple-shot way. That is, the counter loads the **RELOAD** value and begins counting down. On reaching 0, the **COUNT** bit is set and an interrupt is generated if enabled by **INTEN**. The counter then loads the **RELOAD** value again and begins counting.	R/W	0

SysTick Reload Value Register (STRELOAD)

Address Offset	0x014	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **STRELOAD** register specifies the start value to load into the **SysTick Current Value (STCURRENT)** register when the counter reaches 0. The start value can be between 0x1 and 0x00FF FFFF. A start value of 0 is possible but has no effect because the SysTick interrupt and the **COUNT** bit are activated when counting from 1 to 0.

SysTick can be configured as a multiple-shot timer, repeated over and over, firing every N+1 clock pulses, where N is any value from 1 to 0x00FF FFFF. For example, if a tick interrupt is required every 100 clock pulses, 99 must be written into the **RELOAD** field.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:24	RESERVED	Reserved	RO	0x00
23:0	RELOAD	Reload value Value to load into the **SysTick Current Value (STCURRENT)** register when the counter reaches 0.	R/W	0x00 0000

SysTick Current Value Register (STCURRENT)

Address Offset	0x018	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **STCURRENT** register contains the current value of the SysTick counter.

Type R/WC

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

Bits	Field Name	Description	Type	Reset
31:24	Reserved	Reserved	RO	0x00
23:0	CURRENT	Current value This field contains the current value at the time the register is accessed. No read-modify-write protection is provided, so change with care. This register is write-clear. Writing to it with any value clears the register. Clearing this register also clears the **COUNT** bit of the **STCTRL** register	R/WC	0x00 0000

3.5 NVIC Register Descriptions

This section lists and describes the NVIC registers, in numerical order by address offset.

The NVIC registers can only be fully accessed from privileged mode, but interrupts can be pended while in unprivileged mode by enabling the **Configuration and Control (CFGCTRL)** register. Any other unprivileged mode access causes a bus fault.

Ensure that software uses correctly aligned register accesses. The processor does not support unaligned accesses to NVIC registers.

An interrupt can enter the pending state even if it is disabled.

Before programming the **VTABLE** register to relocate the vector table, ensure the vector table entries of the new vector table are set up for fault handlers, NMI, and all enabled exceptions such as interrupts. For more information, see Vector Table Offset (VTABLE).

Interrupt 0–31 Set Enable (EN0)

Address Offset	0x100	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

See Table 5-2 for interrupt assignments.

If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt regardless of its priority.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt enable	R/W	0x0000 0000
		Value **Description** 0 On a read, indicates the interrupt is disabled. On a write, no effect. 1 On a read, indicates the interrupt is enabled. On a write, enables the interrupt. A bit can be cleared only by setting the corresponding **INT[n]** bit in the **DISn** register.		

Interrupt 32–63 Set Enable (EN1)

Address Offset	0x104	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **EN1** register enables interrupts and shows which interrupts are enabled. Bit 0 corresponds to Interrupt 32; bit 31 corresponds to Interrupt 63. See Table 5-2 for interrupt assignments.

If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt regardless of its priority.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt enable	R/W	0x0000 0000
		Value **Description** 0 On a read, indicates the interrupt is disabled. On a write, no effect. 1 On a read, indicates the interrupt is enabled. On a write, enables the interrupt. A bit can be cleared only by setting the corresponding **INT[n]** bit in the **DIS1** register.		

Interrupt 64–95 Set Enable (EN2)

Address Offset	0x108	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **EN2** register enables interrupts and shows which interrupts are enabled. Bit 0 corresponds to Interrupt 64; bit 31 corresponds to Interrupt 95. See Table 5-2 for interrupt assignments.

If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt regardless of its priority.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt enable	R/W	0x0000 0000
		Value **Description** 0 On a read, indicates the interrupt is disabled. On a write, no effect. 1 On a read, indicates the interrupt is enabled. On a write, enables the interrupt. A bit can be cleared only by setting the corresponding **INT[n]** bit in the **DISn** register.		

Interrupt 96–127 Set Enable (EN3)

Address Offset	0x10C	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **EN3** register enables interrupts and shows which interrupts are enabled. Bit 0 corresponds to Interrupt 96; bit 31 corresponds to Interrupt 127. See Table 5-2 for interrupt assignments.

If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt regardless of its priority.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt enable	R/W	0x0000 0000
		Value **Description** 0 On a read, indicates the interrupt is disabled. On a write, no effect. 1 On a read, indicates the interrupt is enabled. On a write, enables the interrupt. A bit can be cleared only by setting the corresponding **INT[n]** bit in the **DISn** register.		

Interrupt 128–147 Set Enable (EN4)

Address Offset	0x110	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **EN4** register enables interrupts and shows which interrupts are enabled. Bit 0 corresponds to Interrupt 128; bit 19 corresponds to Interrupt 147. See Table 5-2 for interrupt assignments.

If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt regardless of its priority.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:20	RESERVED	Reserved	RO	0x000
19:0	INT	Interrupt enable	R/W	0x0 0000
		Value **Description** 0 On a read, indicates the interrupt is disabled. On a write, no effect. 1 On a read, indicates the interrupt is enabled. On a write, enables the interrupt. A bit can be cleared only by setting the corresponding **INT[n]** bit in the **DISn** register.		

Interrupt 0–31 Clear Enable (DIS0)

Address Offset	0x180	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

See Table 5-2 for interrupt assignments.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt disable	R/W	0x0000 0000
		Value **Description** 0 On a read, indicates the interrupt is disabled. On a write, no effect. 1 On a read, indicates the interrupt is enabled. On a write, clears the corresponding **INT[n]** bit in the **EN0** register, disabling interrupt [n].		

Interrupt 32–63 Clear Enable (DIS1)

Address Offset	0x184	**Reset**	0xE000 E000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **DIS1** register disables interrupts. Bit 0 corresponds to Interrupt 32; bit 31 corresponds to Interrupt 63. See Table 5-2 for interrupt assignments.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt disable	R/W	0x0000 0000

Value	Description
0	On a read, indicates the interrupt is disabled. On a write, no effect.
1	On a read, indicates the interrupt is enabled. On a write, clears the corresponding **INT[n]** bit in the **EN1** register, disabling interrupt [n]

Interrupt 64–95 Clear Enable (DIS2)

Address Offset	0x188	**Reset**	0xE000 E000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **DIS2** register disables interrupts. Bit 0 corresponds to Interrupt 64; bit 31 corresponds to Interrupt 95. See Table 5-2 for interrupt assignments.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt disable	R/W	0x0000 0000

Value	Description
0	On a read, indicates the interrupt is disabled. On a write, no effect.
1	On a read, indicates the interrupt is enabled. On a write, clears the corresponding **INT[n]** bit in the **EN2** register, disabling interrupt [n]

Interrupt 96–127 Clear Enable (DIS3)

Address Offset	0x18C	**Reset**	0xE000 E000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **DIS3** register disables interrupts. Bit 0 corresponds to Interrupt 96; bit 31 corresponds to Interrupt 127. See Table 5-2 for interrupt assignments.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt disable	R/W	0x0000 0000
		Value **Description**		
		0 On a read, indicates the interrupt is disabled. On a write, no effect.		
		1 On a read, indicates the interrupt is enabled. On a write, clears the corresponding **INT[n]** bit in the **EN3** register, disabling interrupt [n]		

Interrupt 128–147 Clear Enable (DIS4)

Address Offset	0x190	**Reset**	0xE000 E000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **DIS4** register disables interrupts. Bit 0 corresponds to Interrupt 128; bit 19 corresponds to Interrupt 147. See Table 5-2 for interrupt assignments.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:20	RESERVED	Reserved	RO	0x000
19:0	INT	Interrupt disable	R/W	0x0 0000
		Value **Description**		
		0 On a read, indicates the interrupt is disabled. On a write, no effect.		
		1 On a read, indicates the interrupt is enabled. On a write, clears the corresponding **INT[n]** bit in the **EN4** register, disabling interrupt [n]		

Interrupt 0–31 Set Pending (PEND0)

Address Offset	0x200	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

See Table 5-2 for interrupt assignments.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt set pending	R/W	0x0000 0000
		Value **Description**		
		0 On a read, indicates that the interrupt is not pending. On a write, no effect.		

Bits	Field Name	Description	Type	Reset
		1 On a read, indicates that the interrupt is pending. On a write, the corresponding interrupt is set to pending even if it is disabled. If the corresponding interrupt is already pending, setting a bit has no effect. A bit can be cleared only by setting the corresponding **INT[n]** bit in the **UNPEND0** register.		

Interrupt 32–63 Set Pending (PEND1)

Address Offset	0x204	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PEND1** register forces interrupts into the pending state and shows which interrupts are pending. Bit 0 corresponds to Interrupt 32; bit 31 corresponds to Interrupt 63. See Table 5-2 for interrupt assignments.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt set pending **Value** **Description** 0 On a read, indicates that the interrupt is not pending. On a write, no effect. 1 On a read, indicates that the interrupt is pending. On a write, the corresponding interrupt is set to pending even if it is disabled. If the corresponding interrupt is already pending, setting a bit has no effect. A bit can be cleared only by setting the corresponding **INT[n]** bit in the **UNPEND1** register.	R/W	0x0000 0000

Interrupt 64–95 Set Pending (PEND2)

Address Offset	0x208	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PEND2** register forces interrupts into the pending state and shows which interrupts are pending. Bit 0 corresponds to Interrupt 64; bit 31 corresponds to Interrupt 95. See Table 5-2 for interrupt assignments.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt set pending **Value** **Description** 0 On a read, indicates that the interrupt is not pending. On a write, no effect. 1 On a read, indicates that the interrupt is pending. On a write, the corresponding interrupt is set to pending even if it is disabled.	R/W	0x0000 0000

Bits	Field Name	Description	Type	Reset
		If the corresponding interrupt is already pending, setting a bit has no effect. A bit can be cleared only by setting the corresponding **INT[n]** bit in the **UNPEND2** register.		

Interrupt 96–127 Set Pending (PEND3)

Address Offset	0x20C	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PEND3** register forces interrupts into the pending state and shows which interrupts are pending. Bit 0 corresponds to Interrupt 96; bit 31 corresponds to Interrupt 127. See Table 5-2 for interrupt assignments.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt set pending **Value** **Description** 0 On a read, indicates that the interrupt is not pending. On a write, no effect. 1 On a read, indicates that the interrupt is pending. On a write, the corresponding interrupt is set to pending even if it is disabled. If the corresponding interrupt is already pending, setting a bit has no effect. A bit can be cleared only by setting the corresponding **INT[n]** bit in the **UNPEND3** register.	R/W	0x0000 0000

Interrupt 128–147 Set Pending (PEND4)

Address Offset	0x210	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PEND4** register forces interrupts into the pending state and shows which interrupts are pending. Bit 0 corresponds to Interrupt 128; bit 19 corresponds to Interrupt 147. See Table 5-2 for interrupt assignments.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:20	RESERVED	Reserved	RO	0x000
19:0	INT	Interrupt set pending **Value** **Description** 0 On a read, indicates that the interrupt is not pending. On a write, no effect. 1 On a read, indicates that the interrupt is pending. On a write, the corresponding interrupt is set to pending even if it is disabled.	R/W	0x0 0000

Bits	Field Name	Description	Type	Reset
		If the corresponding interrupt is already pending, setting a bit has no effect. A bit can be cleared only by setting the corresponding **INT[n]** bit in the **UNPEND4** register.		

Interrupt 0–31 Clear Pending (UNPEND0)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x280	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

See Table 5-2 for interrupt assignments.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt clear pending	R/W	0x0000 0000
		Value **Description** 0 On a read, indicates that the interrupt is not pending. On a write, no effect. 1 On a read, indicates that the interrupt is pending. On a write, clears the corresponding **INT[n]** bit in the **PEND0** register, so that interrupt [n] is no longer pending. Setting a bit does not affect the active state of the corresponding interrupt.		

Interrupt 32–63 Clear Pending (UNPEND1)

Address Offset	0x284	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **UNPEND1** register shows which interrupts are pending and removes the pending state from interrupts. Bit 0 corresponds to Interrupt 32; bit 31 corresponds to Interrupt 63. See Table 5-2 for interrupt assignments.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt clear pending	R/W	0x0000 0000
		Value **Description** 0 On a read, indicates that the interrupt is not pending. On a write, no effect. 1 On a read, indicates that the interrupt is pending. On a write, clears the corresponding **INT[n]** bit in the **PEND1** register, so that interrupt [n] is no longer pending. Setting a bit does not affect the active state of the corresponding interrupt.		

Interrupt 64–95 Clear Pending (UNPEND2)

Address Offset	0x288	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **UNPEND2** register shows which interrupts are pending and removes the pending state from interrupts. Bit 0 corresponds to Interrupt 64; bit 31 corresponds to Interrupt 95. See Table 5-2 for interrupt assignments.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt clear pending	R/W	0x0000 0000

Value	Description
0	On a read, indicates that the interrupt is not pending. On a write, no effect.
1	On a read, indicates that the interrupt is pending. On a write, clears the corresponding **INT[n]** bit in the **PEND2** register, so that interrupt [n] is no longer pending. Setting a bit does not affect the active state of the corresponding interrupt.

Interrupt 96–127 Clear Pending (UNPEND3)

Address Offset	0x28C	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **UNPEND3** register shows which interrupts are pending and removes the pending state from interrupts. Bit 0 corresponds to Interrupt 96; bit 31 corresponds to Interrupt 127. See Table 5-2 for interrupt assignments.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt clear pending	R/W	0x0000 0000

Value	Description
0	On a read, indicates that the interrupt is not pending. On a write, no effect.
1	On a read, indicates that the interrupt is pending. On a write, clears the corresponding **INT[n]** bit in the **PEND3** register, so that interrupt [n] is no longer pending. Setting a bit does not affect the active state of the corresponding interrupt.

Interrupt 128–147 Clear Pending (UNPEND4)

Address Offset	0x290	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **UNPEND4** register shows which interrupts are pending and removes the pending state from interrupts. Bit 0 corresponds to Interrupt 128; bit 19 corresponds to Interrupt 147. See Table 5-2 for interrupt assignments.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:20	RESERVED	Reserved	RO	0x0
19:0	INT	Interrupt clear pending	R/W	0x0 0000

Value	Description
0	On a read, indicates that the interrupt is not pending. On a write, no effect.
1	On a read, indicates that the interrupt is pending. On a write, clears the corresponding **INT[n]** bit in the **PEND4** register, so that interrupt [n] is no longer pending. Setting a bit does not affect the active state of the corresponding interrupt.

Interrupt 0–31 Active Bit (ACTIVE0)

Address Offset	0x300	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

See Table 5-2 for interrupt assignments.

Caution: Do not manually set or clear the bits in this register.

Type RO

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt active	RO	0x0000 0000

Value	Description
0	The corresponding interrupt is not active.
1	The corresponding interrupt is active, or active and pending.

Interrupt 32–63 Active Bit (ACTIVE1)

Address Offset	0x304	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **ACTIVE1** register indicates which interrupts are active. Bit 0 corresponds to Interrupt 32; bit 31 corresponds to Interrupt 63. See Table 5-2 for interrupt assignments.

Caution: Do not manually set or clear the bits in this register.

Type RO

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt active	RO	0x0000 0000
		Value **Description**		
		0 The corresponding interrupt is not active.		
		1 The corresponding interrupt is active, or active and pending.		

Interrupt 64–95 Active Bit (ACTIVE2)

Address Offset	0x308	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **ACTIVE2** register indicates which interrupts are active. Bit 0 corresponds to Interrupt 64; bit 31 corresponds to Interrupt 95. See Table 5-2 for interrupt assignments.

Caution: Do not manually set or clear the bits in this register.

Type RO

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt active	RO	0x0000 0000
		Value **Description**		
		0 The corresponding interrupt is not active.		
		1 The corresponding interrupt is active, or active and pending.		

Interrupt 96–127 Active Bit (ACTIVE3)

Address Offset	0x30C	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **ACTIVE3** register indicates which interrupts are active. Bit 0 corresponds to Interrupt 96; bit 31 corresponds to Interrupt 127. See Table 5-2 for interrupt assignments.

Caution: Do not manually set or clear the bits in this register.

Type RO

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

Bits	Field Name	Description	Type	Reset
31:0	INT	Interrupt active	RO	0x0000 0000
		Value **Description**		
		0 The corresponding interrupt is not active.		
		1 The corresponding interrupt is active, or active and pending.		

Interrupt 128–147 Active Bit (ACTIVE4)

Address Offset	0x310	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **ACTIVE4** register indicates which interrupts are active. Bit 0 corresponds to Interrupt 128; bit 19 corresponds to Interrupt 147. See Table 5-2 for interrupt assignments.

Caution: Do not manually set or clear the bits in this register.

Type RO

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

Bits	Field Name	Description	Type	Reset
31:20	RESERVED	Reserved	RO	0x000
19:0	INT	Interrupt active	RO	0x0 0000
		Value **Description**		
		0 The corresponding interrupt is not active.		
		1 The corresponding interrupt is active, or active and pending.		

Interrupt 0–3 Priority (PRI0)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x400	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00

Bits	Field Name	Description	Type	Reset
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 4–7 Priority (PRI1)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address		**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00

Bits	Field Name	Description	Type	Reset
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 8–11 Priority (PRI2)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x408	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00

Bits	Field Name	Description	Type	Reset
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 12–15 Priority (PRI3)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x40C	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 16–19 Priority (PRI4)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x410	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 20–23 Priority (PRI5)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x414	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 24–27 Priority (PRI6)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x418	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 28–31 Priority (PRI7)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x41C	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 32–35 Priority (PRI8)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x420	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 36–39 Priority (PRI9)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x424	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 40–43 Priority (PRI10)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x428	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 44–47 Priority (PRI11)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x42C	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 48–51 Priority (PRI12)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x430	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 52–55 Priority (PRI13)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x434	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 56–59 Priority (PRI14)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x438	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 60–63 Priority (PRI15)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x43C	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 64–67 Priority (PRI16)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x440	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 68–71 Priority (PRI17)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x444	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 72–75 Priority (PRI18)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x448	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 76–75 Priority (PRI19)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x44C	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 80–83 Priority (PRI20)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x450	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 84–87 Priority (PRI21)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x454	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 88–91 Priority (PRI22)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x458	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 92–95 Priority (PRI23)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x45C	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 96–99 Priority (PRI24)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x460	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 100–103 Priority (PRI25)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x464	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 104–107 Priority (PRI26)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x468	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 108–111 Priority (PRI27)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x46C	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 112–115 Priority (PRI28)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x470	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 116–119 Priority (PRI29)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x474	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 120–123 Priority (PRI30)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x478	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 124–127 Priority (PRI31)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x47C	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 128–131 Priority (PRI32)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x480	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 132–135 Priority (PRI33)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x484	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 136–139 Priority (PRI34)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x488	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 140–143 Priority (PRI35)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x48C	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Interrupt 144–147 Priority (PRI36)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0x490	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

PRIn Register Bit Field	Interrupt
Bits 31:29	Interrupt [4n+3]
Bits 23:21	Interrupt [4n+2]
Bits 15:13	Interrupt [4n+1]
Bits 7:5	Interrupt [4n]

See Table 5-2 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The **PRIGROUP** field in the **Application Interrupt and Reset Control (APINT)** register (see Application Interrupt and Reset Control (APINT)) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can be accessed only from privileged mode.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTD			RESERVED					INTC			RESERVED					INTB			RESERVED					INTA			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	INTD	Interrupt priority for interrupt [4n+3] This field holds a priority value, 0–7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	INTC	Interrupt priority for interrupt [4n+2] This field holds a priority value, 0–7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x00
15:13	INTB	Interrupt priority for interrupt [4n+1] This field holds a priority value, 0–7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x00
7:5	INTA	Interrupt priority for interrupt [4n] This field holds a priority value, 0–7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

Software Trigger Interrupt (SWTRIG)

Address Offset	0xF00	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: Only privileged software can enable unprivileged access to the **SWTRIG** register.

Writing an interrupt number to the **SWTRIG** register generates a Software Generated Interrupt (SGI). See Table 5-2 for interrupt assignments.

When the **MAINPEND** bit in the **Configuration and Control (CFGCTRL)** register (see Configuration and Control (CFGCTRL)) is set, unprivileged software can access the **SWTRIG** register.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:6	RESERVED	Reserved	RO	0x000 0000
5:0	INTID	Interrupt ID This field holds the interrupt ID of the required SGI. For example, a value of 0x3 generates an interrupt on IRQ3.	WO	0x00

3.5.1 System Control Block (SCB) Register Descriptions

This section lists and describes the SCB registers, in numerical order by address offset. The SCB registers can be accessed only from privileged mode.

All registers must be accessed with aligned word accesses except for the **FAULTSTAT** and **SYSPRI1** through **SYSPRI3** registers, which can be accessed with byte or aligned half word or word accesses. The processor does not support unaligned accesses to system control block registers.

Auxiliary Control (ACTLR)

Address Offset	0x008	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **ACTLR** register provides disable bits for IT folding, write buffer use for accesses to the default memory map, and interruption of multiple cycle instructions. By default, this register is set to provide optimum performance from the Cortex-M3 processor and does not normally require modification.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																													DISFOLD	DISWBUF	DISMCYC

Bits	Field Name	Description		Type	Reset
31:3	RESERVED	Reserved		RO	0x0000 0000
2	DISFOLD	Disable IT folding		R/W	0
		Value	**Description**		
		0	No effect		
		1	Disables IT folding		

Bits	Field Name	Description	Type	Reset
		In some situations, the processor can start executing the first instruction in an IT block while it is still executing the IT instruction. This behavior is called IT folding, and improves performance, However, IT folding can cause jitter in looping. If a task must avoid jitter, set the **DISFOLD** bit before executing the task, to disable IT folding.		
1	DISWBUF	Disable write buffer **Value** **Description** 0 No effect 1 Disables write buffer use during default memory map accesses. In this situation, all bus faults are precise bus faults but performance is decreased because any store to memory must complete before the processor can execute the next instruction. This bit only affects write buffers implemented in the Cortex-M3 processor.	R/W	0
0	DISMCYC	Disable interrupts of multiple cycle instructions **Value** **Description** 0 No effect 1 Disables interruption of load multiple and store multiple instructions. In this situation, the interrupt latency of the processor is increased because any LDM or STM must complete before the processor can stack the current state and enter the interrupt handler.	R/W	0

CPU ID Base (CPUID)

Address Offset	0xD00	**Reset**	0x412F.C230
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **CPUID** register contains the ARM® Cortex-M3 processor part number, version, and implementation information.

Type RO

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IMP								VAR				CON				PARTNO												REV			

Bits	Field Name	Description	Type	Reset
31:24	IMP	Implementer code **Value** **Description** 0x41 ARM	RO	0x41
23:20	VAR	Variant number **Value** **Description** 0x2 The rn value in the rnpn product revision identifier; for example, the 2 in r2p0.	RO	0x2
19:16	CON	Constant **Value** **Description** 0xF Always reads as 0xF.	RO	0xF
15:4	PARTNO	Part number **Value** **Description** 0xC23 Cortex-M3 processor.	RO	0xC23
3:0	REV	Revision number **Value** **Description** 0x0 The pn value in the rnpn product revision identifier; for example, the 0 in r2p0.	RO	0x0

Interrupt Control and State (INTCTRL)

Address Offset	0xD04	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **INCTRL** register provides a set-pending bit for the NMI exception, and set-pending and clear-pending bits for the PendSV and SysTick exceptions. In addition, bits in this register indicate the exception number of the exception being processed, whether there are preempted active exceptions, the exception number of the highest priority pending exception, and whether any interrupts are pending.

When writing to **INCTRL**, the effect is unpredictable when writing 1 to both the **PENDSV** and **UNPENDSV** bits, or writing 1 to both the **PENDSTSET** and **PENDSTCLR** bits.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NMISET	RESERVED		PENDSV	UNPENDSV	PENDSTSET	PENDSTCLR	RESERVED	ISRPRE	ISRPEND	RESERVED			VECPEND							RETBASE	RESERVED			VECACT							

Bits	Field Name	Description	Type	Reset
31	NMISET	NMI set pending **Value** **Description** 0 On a read, indicates an NMI exception is not pending. On a write, no effect. 1 On a read, indicates an NMI exception is pending. On a write, changes the NMI exception state to pending. Because NMI is the highest-priority exception, normally the processor enters the NMI exception handler as soon as it registers the setting of this bit, and clears this bit on entering the interrupt handler. A read of this bit by the NMI exception handler returns 1 only if the **NMI** signal is reasserted while the processor is executing that handler.	R/W	0
30:29	RESERVED	Reserved	RO	0x0
28	PENDSV	PendSV set pending **Value** **Description** 0 On a read, indicates a PendSV exception is not pending. On a write, no effect. 1 On a read, indicates a PendSV exception is pending. On a write, changes the PendSV exception state to pending. Setting this bit is the only way to set the PendSV exception state to pending. This bit is cleared by writing 1 to the **UNPENDSV** bit.	R/W	0
27	UNPENDSV	PendSV clear pending **Value** **Description** 0 On a write, no effect. 1 On a write, removes the pending state from the PendSV exception. This bit is write only; on a register read, its value is unknown.	WO	0
26	PENDSTSET	SysTick set pending **Value** **Description** 0 On a read, indicates a SysTick exception is not pending. On a write, no effect. 1 On a read, indicates a SysTick exception is pending. On a write, changes the SysTick exception state to pending. This bit is cleared by writing 1 to the **PENDSTCLR** bit.	R/W	0

Bits	Field Name	Description	Type	Reset
25	PENDSTCLR	SysTick clear pending **Value** **Description** 0 On a write, no effect. 1 On a write, removes the pending state from the SysTick exception. This bit is write only; on a register read, its value is unknown.	WO	0
24	RESERVED	Reserved	RO	0
23	ISRPRE	Debug interrupt handling **Value** **Description** 0 The release from halt does not take an interrupt. 1 The release from halt takes an interrupt. This bit is only meaningful in debug mode and reads as 0 when the processor is not in debug mode.	RO	0
22	ISRPEND	Interrupt pending **Value** **Description** 0 No interrupt is pending. 1 An interrupt is pending. This bit provides status for all interrupts excluding NMI and faults.	RO	0
21:19	RESERVED	Reserved	RO	0x0
18:12	VECPEND	Interrupt pending vector number This field contains the exception number of the highest priority pending enabled exception. The value indicated by this field includes the effect of the **BASEPRI** and **FAULTMASK** registers, but not any effect of the **PRIMASK** register **Value** **Description** 0x00 No exceptions are pending 0x01 Reserved 0x02 NMI 0x03 Hard fault 0x04 Memory management fault 0x05 Bus fault 0x06 Usage fault 0x07–0x0A Reserved 0x0B SVCall 0x0C Reserved for debug 0x0D Reserved 0x0E PendSV 0x0F SysTick 0x10 Interrupt vector 0 0x11 Interrupt vector 1 ... ... 0x46 Interrupt vector 54 0x47–0x7F Reserved	RO	0x00
11	RETBASE	Return to base **Value** **Description** 0 There are preempted active exceptions to execute. 1 There are no active exceptions, or the currently executing exception is the only active exception.	RO	0

Bits	Field Name	Description	Type	Reset
		This bit provides status for all interrupts excluding NMI and faults. This bit only has meaning if the processor is currently executing an ISR (the **Interrupt Program Status (IPSR)** register is nonzero).		
10:7	RESERVED	Reserved	RO	0x0
6:0	VECACT	Interrupt pending vector number This field contains the active exception number. The exception numbers can be found in the description for the **VECPEND** field. If this field is clear, the processor is in thread mode. This field contains the same value as the **ISRNUM** field in the **IPSR** register. Subtract 16 from this value to obtain the IRQ number required to index into the **Interrupt Set Enable (ENn)**, **Interrupt Clear Enable (DISn)**, **Interrupt Set Pending (PENDn)**, **Interrupt Clear Pending (UNPENDn)**, and **Interrupt Priority (PRIn)** registers (see Program Status Register (PSR)).	RO	0x00

Vector Table Offset (VTABLE)

Address Offset	0xD08	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **VTABLE** register indicates the offset of the vector table base address from memory address 0x0000 0000.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:30	RESERVED	Reserved	RO	0x0
29	BASE	Vector table base **Value** **Description** 0 The vector table is in the code memory region. 1 The vector table is in the SRAM memory region.	R/W	0
28:9	OFFSET	Vector table offset When configuring the **OFFSET** field, the offset must be aligned to the number of exception entries in the vector table. Because there are 54 interrupts, the offset must be aligned on a 512-byte boundary.	R/W	0x0.0000
8:0	RESERVED	Reserved	RO	0x00

Interrupt Priority Levels

PRIGROUP Bit Field	Binary Point [(1)].	Group Priority Field	Subpriority Field	Group Priorities	Subpriorities
0x0–0x4	bxxx.	[7:5]	None	8	1
0x5	bxx.y	[7:6]	[5]	4	2
0x6	bx.yy	[7]	[6:5]	2	4
0x7	b.yyy	None	[7:5]	1	8

(1) **INTx** field showing the binary point. An x denotes a group priority field bit, and a y denotes a subpriority field bit.

Application Interrupt and Reset Control (APINT)

Address Offset	0xD0C	**Reset**	0xFA05.0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **APINT** register provides priority grouping control for the exception model, endian status for data accesses, and reset control of the system. To write to this register, 0x05FA must be written to the **VECTKEY** field, otherwise the write is ignored.

The **PRIGROUP** field indicates the position of the binary point that splits the **INTx** fields in the **Interrupt Priority (PRIx)** registers into separate group priority and subpriority fields. Interrupt Priority Levels shows how the **PRIGROUP** value controls this split. The bit numbers in the Group Priority Field and Subpriority Field columns in the table refer to the bits in the **INTA** field. For the **INTB** field, the corresponding bits are 15:13; for the **INTC** field, the corresponding bits are 23:21; and for the **INTD** field, the corresponding bits are 31:29.

Note: Determining preemption of an exception uses only the group priority field.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VECTKEY																ENDIANESS	RESERVED				PRIGROUP			RESERVED					SYSRESREQ	VECTCLRACT	VECTRESET

Bits	Field Name	Description	Type	Reset
31:16	VECTKEY	Register key This field is used to guard against accidental writes to this register. 0x05FA must be written to this field to change the bits in this register. On a read, 0xFA05 is returned.	R/W	0xFA05
15	ENDIANESS	Data endianness The CC2538 implementation uses only little-endian mode so this is cleared to 0.	RO	0
14:11	RESERVED	Reserved	RO	0x0
10:8	PRIGROUP	Interrupt priority grouping This field determines the split of group priority from subpriority (for more information, see Interrupt Priority Levels).	R/W	0x0
7:3	RESERVED	Reserved	RO	0x00
2	SYSRESREQ	System reset request **Value** **Description** 0 No effect 1 Resets the core and all on-chip peripherals except the debug interface. This bit is automatically cleared during the reset of the core and reads as 0.	WO	0
1	VECTCLRACT	Clear active NMI / fault This bit is reserved for debug use and reads as 0. This bit must be written as 0, otherwise behavior is unpredictable.	WO	0
0	VECTRESET	System reset This bit is reserved for debug use and reads as 0. This bit must be written as 0, otherwise behavior is unpredictable.	WO	0

System Control (SYSCTRL)

Address Offset	0xD10	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **SYSCTRL** register controls features of entry to and exit from low-power state.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																											SEVONPEND	RESERVED	SLEEPDEEP	SLEEPEXIT	RESERVED

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	Reserved	RO	0x000 0000
4	SEVONPEND	Wake up on pending	R/W	0
		Value **Description** 0 Only enabled interrupts or events can wake up the processor; disabled interrupts are excluded. 1 Enabled events and all interrupts, including disabled interrupts, can wake up the processor.		
		When an event or interrupt enters the pending state, the event signal wakes up the processor from **WFE**. If the processor is not waiting for an event, the event is registered and affects the next **WFE**. The processor also wakes up on execution of a **SEV** instruction or an external event.		
3	RESERVED	Reserved	RO	0
2	SLEEPDEEP	Deep sleep enable	R/W	0
		Value **Description** 0 Use sleep mode as the low-power mode. 1 Use deep-sleep mode as the low-power mode.		
1	SLEEPEXIT	Sleep on ISR exit	R/W	0
		Value **Description** 0 When returning from handler mode to thread mode, do not sleep when returning to thread mode. 1 When returning from handler mode to thread mode, enter sleep or deep sleep when returning from an ISR.		
		Setting this bit enables an interrupt-driven application to avoid returning to an empty main application.		
0	RESERVED	Reserved	RO	0

Configuration and Control (CFGCTRL)

Address Offset	0xD14	**Reset**	0x0000 0200
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **CFGCTRL** register controls entry to thread mode and enables: the handlers for NMI, hard fault and faults escalated by the **FAULTMASK** register to ignore bus faults; trapping of divide by zero and unaligned accesses; and access to the **SWTRIG** register by unprivileged software (see Software Trigger Interrupt (SWTRIG)).

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																						STKALIGN	BFHFNMIGN	RESERVED			DIV0	UNALIGNED	RESERVED	MAINPEND	BASETHR

Bits	Field Name	Description	Type	Reset
31:10	Reserved	Reserved	RO	0x0000 00
9	STKALIGN	Stack alignment on exception entry **Value** **Description** 0 The stack is 4-byte aligned. 1 The stack is 8-byte aligned. On exception entry, the processor uses bit 9 of the stacked **PSR** to indicate the stack alignment. On return from the exception, it uses this stacked bit to restore the correct stack alignment.	R/W	1
8	BFHFNMIGN	Ignore bus fault in NMI and fault This bit enables handlers with priority –1 or –2 to ignore data bus faults caused by load and store instructions. The setting of this bit applies to the hard fault, NMI, and **FAULTMASK** escalated handlers. **Value** **Description** 0 Data bus faults caused by load and store instructions cause a lock-up. 1 Handlers running at priority –1 and –2 ignore data bus faults caused by load and store instructions. Set this bit only when the handler and its data are in absolutely safe memory. The normal use of this bit is to probe system devices and bridges to detect control path problems and fix them.	R/W	0
7:5	Reserved	Reserved	RO	0x0
4	DIV0	Trap on divide by 0 This bit enables faulting or halting when the processor executes an **SDIV** or **UDIV** instruction with a divisor of 0. **Value** **Description** 0 Do not trap on divide by 0. A divide by 0 returns a quotient of 0. 1 Trap on divide by 0.	R/W	0
3	UNALIGNED	Trap on unaligned access **Value** **Description** 0 Do not trap on unaligned half word and word accesses. 1 Trap on unaligned half word and word accesses. An unaligned access generates a usage fault. Unaligned **LDM**, **STM**, **LDRD**, and **STRD** instructions always fault regardless of whether **UNALIGNED** is set.	R/W	0
2	Reserved	Reserved	RO	0
1	MAINPEND	Allow main interrupt trigger **Value** **Description** 0 Disables unprivileged software access to the **SWTRIG** register. 1 Enables unprivileged software access to the **SWTRIG** register (for more information, see ??).	R/W	0
0	BASETHR	Thread state control **Value** **Description** 0 The processor can enter thread mode only when no exception is active. 1 The processor can enter thread mode from any level under the control of an EXC_RETURN value (for more information, see ??).	R/W	0

System Handler Priority 1 (SYSPRI1)

Address Offset	0xD18	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **SYSPRI1** register configures the priority level, 0 to 7 of the usage fault, bus fault, and memory management fault exception handlers. This register is byte-accessible.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED								USAGE			RESERVED					BUS			RESERVED					MEM			RESERVED				

Bits	Field Name	Description	Type	Reset
31:24	RESERVED	Reserved	RO	0x00
23:21	USAGE	Usage fault priority This field configures the priority level of the usage fault. Configurable priority values are in the range 0–7, with lower values having higher priority.	R/W	0x0
20:16	RESERVED	Reserved	RO	0x0
15:13	BUS	Bus fault priority This field configures the priority level of the bus fault. Configurable priority values are in the range 0–7, with lower values having higher priority.	R/W	0x0
12:8	RESERVED	Reserved	RO	0x0
7:5	MEM	Memory management fault priority This field configures the priority level of the memory management fault. Configurable priority values are in the range 0–7, with lower values having higher priority.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x0

System Handler Priority 2 (SYSPRI2)

Address Offset	0xD1C	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **SYSPRI2** register configures the priority level, 0 to 7 of the SVCall handler. This register is byte-accessible.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:29	SVC	SVCall priority This field configures the priority level of SVCall. Configurable priority values are in the range 0–7, with lower values having higher priority.	R/W	0x0
28:0	RESERVED	Reserved	RO	0x000 0000

System Handler Priority 3 (SYSPRI3)

Address Offset	0xD20	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **SYSPRI3** register configures the priority level, 0 to 7 of the SysTick exception and PendSV handlers. This register is byte-accessible.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TICK			RESERVED					PENDSV			RESERVED													DEBUG			RESERVED				

Bits	Field Name	Description	Type	Reset
31:29	TICK	SysTick exception priority This field configures the priority level of the SysTick exception. Configurable priority values are in the range 0–7, with lower values having higher priority.	R/W	0x0
28:24	RESERVED	Reserved	RO	0x00
23:21	PENDSV	PendSV priority This field configures the priority level of PendSV. Configurable priority values are in the range 0–7, with lower values having higher priority.	R/W	0x0
20:8	RESERVED	Reserved	RO	0x0000
7:5	DEBUG	Debug priority This field configures the priority level of debug. Configurable priority values are in the range 0–7, with lower values having higher priority.	R/W	0x0
4:0	RESERVED	Reserved	RO	0x00

System Handler Control and State (SYSHNDCTRL)

Address Offset	0xD24	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **SYSHNDCTRL** register enables the system handlers, and indicates the pending status of the usage fault, bus fault, memory management fault, and SVC exceptions as well as the active status of the system handlers.

If a system handler is disabled and the corresponding fault occurs, the processor treats the fault as a hard fault.

This register can be modified to change the pending or active status of system exceptions. An OS kernel can write to the active bits to perform a context switch that changes the current exception type.

Caution: Software that changes the value of an active bit in this register without correct adjustment to the stacked content can cause the processor to generate a fault exception. Ensure software that writes to this register retains and subsequently restores the current active status. If the value of a bit in this register must be modified after enabling the system handlers, a read-modify-write procedure must be used to ensure that only the required bit is modified.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED													USAGE	BUS	MEM	SVC	BUSP	MEMP	USAGEP	TICK	PNDSV	RESERVED	MON	SVCA	RESERVED				USGA	RESERVED	BUSA	MEMA

Bits	Field Name	Description	Type	Reset
31:19	RESERVED	Reserved	RO	0x000
18	USAGE	Usage fault enable **Value** **Description**	R/W	0

Bits	Field Name	Description	Type	Reset
		0 Disables the usage fault exception.		
		1 Enables the usage fault exception.		
17	BUS	Bus fault enable	R/W	0
		Value **Description**		
		0 Disables the bus fault exception.		
		1 Enables the bus fault exception.		
16	MEM	Memory management fault enable	R/W	0
		Value **Description**		
		0 Disables the memory management fault exception.		
		1 Enables the memory management fault exception.		
15	SVC	SVC call pending	R/W	0
		Value **Description**		
		0 An SVC call exception is not pending.		
		1 An SVC call exception is pending.		
		This bit can be modified to change the pending status of the SVC call exception.		
14	BUSP	Bus fault pending	R/W	0
		Value **Description**		
		0 A bus fault exception is not pending.		
		1 A bus fault exception is pending.		
		This bit can be modified to change the pending status of the bus fault exception.		
13	MEMP	Memory management fault pending	R/W	0
		Value **Description**		
		0 A memory management fault exception is not pending.		
		1 A memory management fault exception is pending.		
		This bit can be modified to change the pending status of the memory management fault exception.		
12	USAGEP	Usage fault pending	R/W	0
		Value **Description**		
		0 A usage fault exception is not pending.		
		1 A usage fault exception is pending.		
		This bit can be modified to change the pending status of the usage fault exception.		
11	TICK	SysTick exception active	R/W	0
		Value **Description**		
		0 A SysTick exception is not active.		
		1 A SysTick exception is active.		
		This bit can be modified to change the active status of the SysTick exception; however, see the Caution above before setting this bit.		
10	PNDSV	PendSV exception active	R/W	0
		Value **Description**		
		0 A PendSV exception is not active.		
		1 A PendSV exception is active.		
		This bit can be modified to change the active status of the PendSV exception; however, see the Caution above before setting this bit.		
9	RESERVED	Reserved	RO	0
8	MON	Debug monitor active	R/W	0
		Value **Description**		
		0 The debug monitor is not active.		
		1 The debug monitor is active.		

Bits	Field Name	Description	Type	Reset
7	SVCA	SVC call active **Value** **Description** 0 SVC call is not active. 1 SVC call is active. This bit can be modified to change the active status of the SVC call exception; however, see the Caution above before setting this bit.	R/W	0
6:4	RESERVED	Reserved	RO	0x0
3	USGA	Usage fault active **Value** **Description** 0 Usage fault is not active. 1 Usage fault is active. This bit can be modified to change the active status of the usage fault exception; however, see the Caution above before setting this bit.	R/W	0
2	RESERVED	Reserved	RO	0
1	BUSA	Bus fault active **Value** **Description** 0 Bus fault is not active. 1 Bus fault is active. This bit can be modified to change the active status of the bus fault exception; however, see the Caution above before setting this bit.	R/W	0
0	MEMA	Memory management fault active **Value** **Description** 0 Memory management fault is not active. 1 Memory management fault is active. This bit can be modified to change the active status of the memory management fault exception; however, see the Caution above before setting this bit.	R/W	0

Configurable Fault Status (FAULTSTAT)

Address Offset	0xD28	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **FAULTSTAT** register indicates the cause of a memory management fault, bus fault, or usage fault. Each of these functions is assigned to a subregister as follows:

- **Usage Fault Status (UFAULTSTAT)**, bits 31:16
- **Bus Fault Status (BFAULTSTAT)**, bits 15:8
- **Memory Management Fault Status (MFAULTSTAT)**, bits 7:0

FAULTSTAT is byte accessible. **FAULTSTAT** or its subregisters can be accessed as follows:

- The complete **FAULTSTAT** register, with a word access to offset 0xD28
- The **MFAULTSTAT** register, with a byte access to offset 0xD28
- The **MFAULTSTAT** and **BFAULTSTAT** registers, with a halfword access to offset 0xD28
- The **BFAULTSTAT** register, with a byte access to offset 0xD29
- The **UFAULTSTAT** register, with a halfword access to offset 0xD2A

Bits are cleared by setting them to 1.

In a fault handler, the true faulting address can be determined by:

1. Read and save the **Memory Management Fault Address (MMADDR)** or **Bus Fault Address (FAULTADDR)** value.
2. Read the **MMARV** bit in **MFAULTSTAT**, or the **BFARV** bit in **BFAULTSTAT** to determine if the contents of **MMADDR** or **FAULTADDR** are valid.

Software must follow this sequence because another higher priority exception might change the **MMADDR** or **FAULTADDR** value. For example, if a higher priority handler preempts the current fault handler, the other fault might change the value of the **MMADDR** or **FAULTADDR** register.

Type R/W1C

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED						DIV0	UNALIGN	RESERVED				NOCP	INVPC	INVSTAT	UNDEF	BFARV	RESERVED		BSTKE	BUSTKE	IMPRE	PRECISE	IBUS	MMARV	RESERVED		MSTKE	MUSTKE	RESERVED	DERR	IERR

Bits	Field Name	Description	Type	Reset
31:26	RESERVED	Reserved	RO	0x00
25	DIV0	Divide-by-zero usage fault **Value** **Description** 0 No divide-by-zero fault has occurred, or divide-by-zero trapping is not enabled. 1 The processor has executed an **SDIV** or **UDIV** instruction with a divisor of 0. When this bit is set, the **PC** value stacked for the exception return points to the instruction that performed the divide by 0. Trapping on divide-by-zero is enabled by setting the **DIV0** bit in the **Configuration and Control (CFGCTRL)** register (see Configuration and Control (CFGCTRL)). This bit is cleared by setting it to 1.	R/W1C	0
24	UNALIGN	Unaligned access usage fault **Value** **Description** 0 No unaligned access fault has occurred, or unaligned access trapping is not enabled. 1 The processor has made an unaligned memory access. Unaligned **LDM**, **STM**, **LDRD**, and **STRD** instructions always fault regardless of the configuration of this bit. Trapping on unaligned access is enabled by setting the **UNALIGNED** bit in the **CFGCTRL** register (see Configuration and Control (CFGCTRL)). This bit is cleared by setting it to 1.	R/W1C	0

Bits	Field Name	Description	Type	Reset
23:20	RESERVED	Reserved	RO	0x00
19	NOCP	No coprocessor usage fault **Value** **Description** 0 A usage fault has not been caused by trying to access a coprocessor. 1 The processor has tried to access a coprocessor. This bit is cleared by setting it to 1.	R/W1C	0
18	INVPC	Invalid PC load usage fault **Value** **Description** 0 A usage fault has not been caused by trying to load an invalid PC value. 1 The processor has tried an illegal load of EXC_RETURN to the PC as a result of an invalid context or an invalid EXC_RETURN value. When this bit is set, the **PC** value stacked for the exception return points to the instruction that tried to perform the illegal load of the **PC**. This bit is cleared by setting it to 1.	R/W1C	0
17	INVSTAT	Invalid state usage fault **Value** **Description** 0 A usage fault has not been caused by an invalid state. 1 The processor has tried to execute an instruction that makes illegal use of the **EPSR** register. When this bit is set, the **PC** value stacked for the exception return points to the instruction that attempted the illegal use of the **Execution Program Status Register (EPSR)** register. This bit is not set if an undefined instruction uses the **EPSR** register. This bit is cleared by setting it to 1.	R/W1C	0
16	UNDEF	Undefined instruction usage fault **Value** **Description** 0 A usage fault has not been caused by an undefined instruction. 1 The processor has tried to execute an undefined instruction. When this bit is set, the **PC** value stacked for the exception return points to the undefined instruction. An undefined instruction is an instruction that the processor cannot decode. This bit is cleared by setting it to 1.	R/W1C	0
15	BFARV	Bus fault address register valid **Value** **Description** 0 The value in the **Bus Fault Address (FAULTADDR)** register is not a valid fault address. 1 The **FAULTADDR** register is holding a valid fault address. This bit is set after a bus fault, where the address is known. Other faults can clear this bit, such as a memory management fault occurring later. If a bus fault occurs and is escalated to a hard fault because of priority, the hard fault handler must clear this bit. This action prevents problems if returning to a stacked active bus fault handler whose **FAULTADDR** register value has been overwritten. This bit is cleared by setting it to 1.	R/W1C	0
14:13	RESERVED	Reserved	RO	0

Bits	Field Name	Description	Type	Reset
12	BSTKE	Stack bus fault **Value** **Description** 0 No bus fault has occurred on stacking for exception entry. 1 Stacking for an exception entry has caused one or more bus faults. When this bit is set, the **SP** is still adjusted but the values in the context area on the stack might be incorrect. A fault address is not written to the **FAULTADDR** register. This bit is cleared by setting it to 1.	R/W1C	0
11	BUSTKE	Unstack bus fault **Value** **Description** 0 No bus fault has occurred on unstacking for a return from exception. 1 Unstacking for a return from exception has caused one or more bus faults. This fault is chained to the handler. Thus, when this bit is set, the original return stack is still present. The **SP** is not adjusted from the failing return, a new save is not performed, and a fault address is not written to the **FAULTADDR** register. This bit is cleared by setting it to 1.	R/W1C	0
10	IMPRE	Imprecise data bus error **Value** **Description** 0 An imprecise data bus error has not occurred. 1 A data bus error has occurred, but the return address in the stack frame is not related to the instruction that caused the error. When this bit is set, a fault address is not written to the **FAULTADDR** register. This fault is asynchronous. Therefore, if the fault is detected when the priority of the current process is higher than the bus fault priority, the bus fault becomes pending and becomes active only when the processor returns from all higher-priority processes. If a precise fault occurs before the processor enters the handler for the imprecise bus fault, the handler detects that both the **IMPRE** bit is set and one of the precise fault status bits is set. This bit is cleared by setting it to 1.	R/W1C	0
9	PRECISE	Precise data bus error **Value** **Description** 0 A precise data bus error has not occurred. 1 A data bus error has occurred, and the **PC** value stacked for the exception return points to the instruction that caused the fault. When this bit is set, the fault address is written to the **FAULTADDR** register. This bit is cleared by setting it to 1.	R/W1C	0
8	IBUS	Instruction bus error **Value** **Description** 0 An instruction bus error has not occurred. 1 An instruction bus error has occurred. The processor detects the instruction bus error on prefetching an instruction, but sets this bit only if it attempts to issue the faulting instruction .When this bit is set, a fault address is not written to the **FAULTADDR** register. This bit is cleared by setting it to 1.	R/W1C	0

Bits	Field Name	Description	Type	Reset
7	MMARV	Memory management fault address register valid **Value** **Description** 0 The value in the **Memory Management Fault Address (MMADDR)** register is not a valid fault address. 1 The **MMADDR** register is holding a valid fault address. If a memory management fault occurs and is escalated to a hard fault because of priority, the hard fault handler must clear this bit. This action prevents problems if returning to a stacked active memory management fault handler whose **MMADDR** register value has been overwritten. This bit is cleared by setting it to 1.	R/W1C	0
6:5	RESERVED	Reserved	RO	0
4	MSTKE	Stack access violation **Value** **Description** 0 No memory management fault has occurred on stacking for exception entry. 1 Stacking for an exception entry has caused one or more access violations. When this bit is set, the **SP** is still adjusted but the values in the context area on the stack might be incorrect. A fault address is not written to the **MMADDR** register. This bit is cleared by setting it to 1.	R/W1C	0
3	MUSTKE	Unstack access violation **Value** **Description** 0 No memory management fault has occurred on unstacking for a return from exception. 1 Unstacking for a return from exception has caused one or more access violations. This fault is chained to the handler. Thus, when this bit is set, the original return stack is still present. The **SP** is not adjusted from the failing return, a new save is not performed, and a fault address is not written to the **MMADDR** register. This bit is cleared by setting it to 1.	R/W1C	0
2	RESERVED	Reserved	RO	0
1	DERR	Data access violation **Value** **Description** 0 A data access violation has not occurred. 1 The processor tried to load or store at a location that does not permit the operation. When this bit is set, the **PC** value stacked for the exception return points to the faulting instruction and the address of the attempted access is written to the **MMADDR** register. This bit is cleared by setting it to 1.	R/W1C	0
0	IERR	Instruction access violation **Value** **Description** 0 An instruction access violation has not occurred. 1 The processor tried an instruction fetch from a location that does not permit execution. This fault occurs on any access to an XN region, even when the MPU is disabled or not present. When this bit is set, the **PC** value stacked for the exception return points to the faulting instruction and the address of the attempted access is not written to the **MMADDR** register. This bit is cleared by setting it to 1.	R/W1C	0

Hard Fault Status (HFAULTSTAT)

Address Offset	0xD2C	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **HFAULTSTAT** register gives information about events that activate the hard fault handler.

Bits are cleared by writing 1 to them.

Type R/W1C

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DBG	FORCED	RESERVED																												VECT	RESERVED

Bits	Field Name	Description	Type	Reset
31	DBG	Debug event This bit is reserved for debug use. This bit must be written as 0; otherwise, behavior is unpredictable.	R/W1C	0
30	FORCED	Forced hard fault **Value** **Description** 0 No forced hard fault has occurred. 1 A forced hard fault has been generated by escalation of a fault with configurable priority that cannot be handled, either because of priority or because it is disabled. When this bit is set, the hard fault handler must read the other fault status registers to find the cause of the fault. This bit is cleared by setting it to 1.	R/W1C	0
29:2	RESERVED	Reserved	RO	0x000 0000
1	VECT	Vector table read fault **Value** **Description** 0 No bus fault has occurred on a vector table read. 1 A bus fault occurred on a vector table read. This error is always handled by the hard fault handler. When this bit is set, the **PC** value stacked for the exception return points to the instruction that was preempted by the exception. This bit is cleared by setting it to 1.	R/W1C	0
0	RESERVED	Reserved	RO	0

Memory Management Fault Address (MMADDR)

Address Offset	0xD34	**Reset**	–
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **MMADDR** register contains the address of the location that generated a memory management fault. When an unaligned access faults, the address in the **MMADDR** register is the actual address that faulted. Because a single read or write instruction can be split into multiple aligned accesses, the fault address can be any address in the range of the requested access size. Bits in the **Memory Management Fault Status (MFAULTSTAT)** register indicate the cause of the fault and whether the value in the **MMADDR** register is valid (see Configurable Fault Status (FAULTSTAT)).

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:0	ADDR	Fault address When the **MMARV** bit of the **MFAULTSTAT** register is set, this field holds the address of the location that generated the memory management fault.	R/W	–

Bus Fault Address (FAULTADDR)

Address Offset	0xD38	**Reset**	–
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **FAULTADDR** register contains the address of the location that generated a bus fault. When an unaligned access faults, the address in the **FAULTADDR** register is the one requested by the instruction, even if it is not the address of the fault. Bits in the **Bus Fault Status (BFAULTSTAT)** register indicate the cause of the fault and whether the value in the **FAULTADDR** register is valid (see Configurable Fault Status (FAULTSTAT)).

Type

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

Bits	Field Name	Description	Type	Reset
31:0	ADDR	Fault address When the **FAULTADDRV** bit of the **BFAULTSTAT** register is set, this field holds the address of the location that generated the bus fault.	R/W	–

3.5.2 MPU Register Descriptions

This section lists and describes the MPU registers, in numerical order by address offset.

The MPU registers can be accessed only from privileged mode.

MPU Type (MPUTYPE)

Address Offset	0xD90	**Reset**	0x0000 0800
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **MPUTYPE** register indicates whether the MPU is present, and if so, how many regions it supports.

Type

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED								IREGION								DREGION								RESERVED							SEPARATE

Bits	Field Name	Description	Type	Reset
31:24	RESERVED	Reserved	RO	0x00
23:16	IREGION	Number of I regions This field indicates the number of supported MPU instruction regions. This field always contains 0x00. The MPU memory map is unified and is described by the **DREGION** field.	RO	0x00
15:8	DREGION	Number of D regions **Value** **Description** 0 Indicates there are eight supported MPU data regions.	RO	0x08

Bits	Field Name	Description	Type	Reset
7:1	RESERVED	Reserved	RO	0x00
0	SEPARATE	Separate or unified MPU	RO	0
		Value **Description**		
		0 Indicates the MPU is unified.		

MPU Control (MPUCTRL)

Address Offset	0xD94	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **MPUCTRL** register enables the MPU, enables the default memory map background region, and enables use of the MPU when in the hard fault, NMI, and **Fault Mask Register (FAULTMASK)** escalated handlers.

When the **ENABLE** and **PRIVDEFEN** bits are both set:

- For privileged accesses, the default memory map is as described in Section 4.1. Any access by privileged software that does not address an enabled memory region behaves as defined by the default memory map.
- Any access by unprivileged software that does not address an enabled memory region causes a memory management fault.

Execute Never (XN) and strongly ordered rules always apply to the system control space regardless of the value of the **ENABLE** bit.

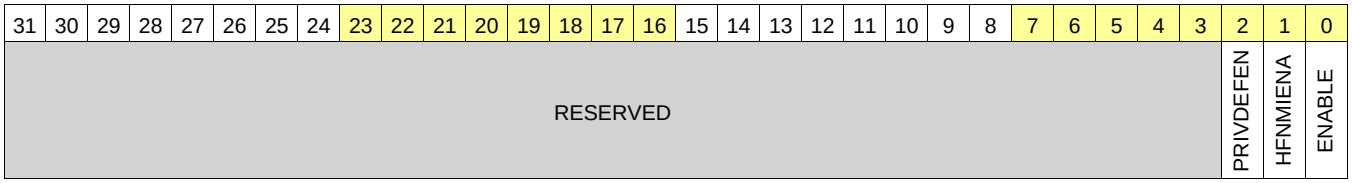
When the **ENABLE** bit is set, at least one region of the memory map must be enabled for the system to function unless the **PRIVDEFEN** bit is set. If the **PRIVDEFEN** bit is set and no regions are enabled, then only privileged software can operate.

When the **ENABLE** bit is clear, the system uses the default memory map, which has the same memory attributes as if the MPU is not implemented (for more information, see Table 4-2). The default memory map applies to accesses from both privileged and unprivileged software.

When the MPU is enabled, accesses to the system control space and vector table are always permitted. Other areas are accessible based on regions and whether **PRIVDEFEN** is set.

Unless the **HFNMIENA** bit is set, the MPU is not enabled when the processor is executing the handler for an exception with priority –1 or –2. These priorities are possible only when handling a hard fault or NMI exception or when **FAULTMASK** is enabled. Setting the **HFNMIENA** bit enables the MPU when operating with these two priorities.

Type R/W



Bits	Field Name	Description	Type	Reset
31:3	RESERVED	Reserved	RO	0x0000 0000
2	PRIVDEFEN	MPU default region This bit enables privileged software access to the default memory map.	R/W	0
		Value **Description**		
		0 If the MPU is enabled, this bit disables use of the default memory map. Any memory access to a location not covered by any enabled region causes a fault.		
		1 If the MPU is enabled, this bit enables use of the default memory map as a background region for privileged software accesses.		
		When this bit is set, the background region acts as if it is region number –1. Any region that is defined and enabled has priority over this default map. If the MPU is disabled, the processor ignores this bit.		
1	HFNMIENA	MPU enabled during faults This bit controls the operation of the MPU during hard fault, NMI, and **FAULTMASK** handlers.	R/W	0
		Value **Description**		

Bits	Field Name	Description	Type	Reset
		0 The MPU is disabled during hard fault, NMI, and **FAULTMASK** handlers, regardless of the value of the **ENABLE** bit. 1 The MPU is enabled during hard fault, NMI, and **FAULTMASK** handlers. When the MPU is disabled and this bit is set, the resulting behavior is unpredictable.		
0	ENABLE	MPU enable **Value** **Description** 0 The MPU is disabled. 1 The MPU is enabled. When the MPU is disabled and the **HFNMIENA** bit is set, the resulting behavior is unpredictable.	R/W	0

MPU Region Number (MPUNUMBER)

Address Offset	0xD98	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **MPUNUMBER** register selects which memory region is referenced by the **MPU Region Base Address (MPUBASE)** and **MPU Region Attribute and Size (MPUATTR)** registers. Normally, the required region number should be written to this register before accessing the **MPUBASE** or **MPUATTR** register. However, the region number can be changed by writing to the **MPUBASE** register with the **VALID** bit set (see MPU Region Base Address (MPUBASE)). This write updates the value of the **REGION** field.

Type R/W

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

Bits	Field Name	Description	Type	Reset
31:3	RESERVED	Reserved	RO	0x0000 0000
2:0	NUMBER	MPU region to access This field indicates the MPU region referenced by the **MPUBASE** and **MPUATTR** registers. The MPU supports eight memory regions.	R/W	0x0

MPU Region Base Address (MPUBASE)

Address Offset	0xD9C	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **MPUBASE** register defines the base address of the MPU region selected by the **MPU Region Number (MPUNUMBER)** register and can update the value of the **MPUNUMBER** register. To change the current region number and update the **MPUNUMBER** register, write the **MPUBASE** register with the **VALID** bit set.

The **ADDR** field is bits 31:N of the **MPUBASE** register. Bits (N–1):5 are reserved. The region size, as specified by the **SIZE** field in the **MPU Region Attribute and Size (MPUATTR)** register, defines the value of N where:
$N = Log_2$(region size in bytes)

If the region size is configured to 4GB in the **MPUATTR** register, there is no valid **ADDR** field. In this case, the region occupies the complete memory map, and the base address is 0x0000 0000.

The base address is aligned to the size of the region. For example, a 64-KB region must be aligned on a multiple of 64KB, for example, at 0x0001 0000 or 0x0002 0000.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDR																											VALID	RESERVED	REGION			

Bits	Field Name	Description	Type	Reset
31:5	ADDR	Base address mask Bits 31:N in this field contain the region base address. The value of N depends on the region size, as shown above. The remaining bits (N–1):5 are reserved.	R/W	0x000 0000
4	VALID	Region number valid **Value** **Description** 0 The **MPUNUMBER** register is not changed and the processor updates the base address for the region specified in the **MPUNUMBER** register and ignores the value of the **REGION** field. 1 The **MPUNUMBER** register is updated with the value of the **REGION** field and the base address is updated for the region specified in the **REGION** field.	WO	0
3	RESERVED	Reserved	RO	0
2:0	REGION	Region number On a write, contains the value to be written to the **MPUNUMBER** register. On a read, returns the current region number in the **MPUNUMBER** register.	R/W	0x0

MPU Region Base Address Alias 1 (MPUBASE1)

Address Offset	0xDA4	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **MPUBASE** register defines the base address of the MPU region selected by the **MPU Region Number (MPUNUMBER)** register and can update the value of the **MPUNUMBER** register. To change the current region number and update the **MPUNUMBER** register, write the **MPUBASE** register with the **VALID** bit set.

The **ADDR** field is bits 31:N of the **MPUBASE** register. Bits (N–1):5 are reserved. The region size, as specified by the **SIZE** field in the **MPU Region Attribute and Size (MPUATTR)** register, defines the value of N where:
$N = Log_2$(Region size in bytes)

If the region size is configured to 4GB in the **MPUATTR** register, there is no valid **ADDR** field. In this case, the region occupies the complete memory map, and the base address is 0x0000 0000.

The base address is aligned to the size of the region. For example, a 64-KB region must be aligned on a multiple of 64KB, for example, at 0x0001 0000 or 0x0002 0000.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDR																											VALID	RESERVED	REGION			

Bits	Field Name	Description	Type	Reset
31:5	ADDR	Base address mask Bits 31:N in this field contain the region base address. The value of N depends on the region size, as shown above. The remaining bits (N–1):5 are reserved.	R/W	0x000 0000
4	VALID	Region number valid **Value** **Description**	WO	0

Bits	Field Name	Description	Type	Reset
		0 The **MPUNUMBER** register is not changed and the processor updates the base address for the region specified in the **MPUNUMBER** register and ignores the value of the **REGION** field. 1 The **MPUNUMBER** register is updated with the value of the **REGION** field and the base address is updated for the region specified in the **REGION** field.		
3	RESERVED	Reserved	RO	0
2:0	REGION	Region number On a write, contains the value to be written to the **MPUNUMBER** register. On a read, returns the current region number in the **MPUNUMBER** register.	R/W	0x0

MPU Region Base Address Alias 2 (MPUBASE2)

Address Offset	0xDAC	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

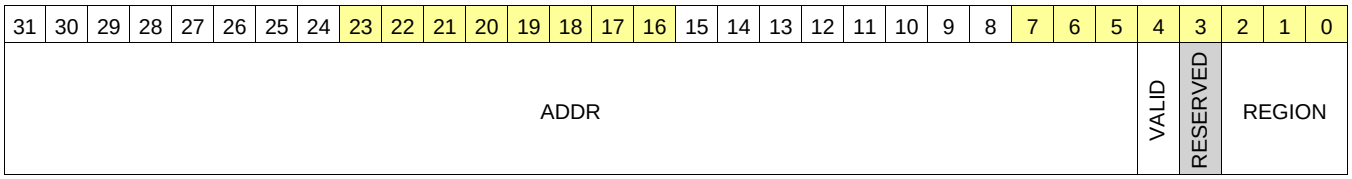
The **MPUBASE** register defines the base address of the MPU region selected by the **MPU Region Number (MPUNUMBER)** register and can update the value of the **MPUNUMBER** register. To change the current region number and update the **MPUNUMBER** register, write the **MPUBASE** register with the **VALID** bit set.

The **ADDR** field is bits 31:N of the **MPUBASE** register. Bits (N–1):5 are reserved. The region size, as specified by the **SIZE** field in the **MPU Region Attribute and Size (MPUATTR)** register, defines the value of N where:
$N = Log_2$(region size in bytes)

If the region size is configured to 4GB in the **MPUATTR** register, there is no valid **ADDR** field. In this case, the region occupies the complete memory map, and the base address is 0x0000 0000.

The base address is aligned to the size of the region. For example, a 64-KB region must be aligned on a multiple of 64KB, for example, at 0x0001 0000 or 0x0002 0000.

Type R/W



Bits	Field Name	Description	Type	Reset
31:5	ADDR	Base address mask Bits 31:N in this field contain the region base address. The value of N depends on the region size, as shown above. The remaining bits (N–1):5 are reserved.	R/W	0x000 0000
4	VALID	Region number valid **Value** **Description** 0 The **MPUNUMBER** register is not changed and the processor updates the base address for the region specified in the **MPUNUMBER** register and ignores the value of the **REGION** field. 1 The **MPUNUMBER** register is updated with the value of the **REGION** field and the base address is updated for the region specified in the **REGION** field.	WO	0
3	RESERVED	Reserved	RO	0
2:0	REGION	Region number On a write, contains the value to be written to the **MPUNUMBER** register. On a read, returns the current region number in the **MPUNUMBER** register.	R/W	0x0

MPU Region Base Address Alias 3 (MPUBASE3)

Address Offset	0xDB4	**Reset**	0x0000 0000
Physical Address	0xE000 E000	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **MPUBASE** register defines the base address of the MPU region selected by the **MPU Region Number (MPUNUMBER)** register and can update the value of the **MPUNUMBER** register. To change the current region number and update the **MPUNUMBER** register, write the **MPUBASE** register with the **VALID** bit set.

The **ADDR** field is bits 31:N of the **MPUBASE** register. Bits (N–1):5 are reserved. The region size, as specified by the **SIZE** field in the **MPU Region Attribute and Size (MPUATTR)** register, defines the value of N where:
$N = Log_2$(region size in bytes)

If the region size is configured to 4GB in the **MPUATTR** register, there is no valid **ADDR** field. In this case, the region occupies the complete memory map, and the base address is 0x0000 0000.

The base address is aligned to the size of the region. For example, a 64-KB region must be aligned on a multiple of 64KB, for example, at 0x0001 0000 or 0x0002 0000.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDR																											VALID	RESERVED	REGION			

Bits	Field Name	Description	Type	Reset
31:5	ADDR	Base address mask Bits 31:N in this field contain the region base address. The value of N depends on the region size, as shown above. The remaining bits (N–1):5 are reserved.	R/W	0x000 0000
4	VALID	Region number valid **Value** **Description** 0 The **MPUNUMBER** register is not changed and the processor updates the base address for the region specified in the **MPUNUMBER** register and ignores the value of the **REGION** field. 1 The **MPUNUMBER** register is updated with the value of the **REGION** field and the base address is updated for the region specified in the **REGION** field.	WO	0
3	RESERVED	Reserved	RO	0
2:0	REGION	Region number On a write, contains the value to be written to the **MPUNUMBER** register. On a read, returns the current region number in the **MPUNUMBER** register.	R/W	0x0

Example SIZE Field Values

SIZE Encoding	Region Size	Value of N [(1)]	Note
00100b (0x4)	32 B	5	Minimum permitted size
01001b (0x9)	1KB	10	–
10011b (0x13)	1MB	20	–
11101b (0x1D)	1GB	30	–
11111b (0x1F)	4GB	No valid **ADDR** field in the **MPUBASE** register; the region occupies the complete memory map.	Maximum possible size

(1) This refers to the N parameter in the **MPUBASE** register (see MPU Region Base Address (MPUBASE)).

MPU Region Attribute and Size (MPUATTR)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0xDA0	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **MPUATTR** register defines the region size and memory attributes of the MPU region specified by the **MPU Region Number (MPUNUMBER)** register and enables that region and any subregions.

The **MPUATTR** register is accessible using word or half word accesses with the most-significant half word holding the region attributes and the least-significant half word holding the region size and the region and subregion enable bits.

The MPU access permission attribute bits, **XN**, **AP**, **TEX**, **S**, **C**, and **B**, control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, then the MPU generates a permission fault.

The **SIZE** field defines the size of the MPU memory region specified by the **MPUNUMBER** register as follows:
(Region size in bytes) = $2^{(SIZE + 1)}$

The smallest permitted region size is 32 bytes, corresponding to a **SIZE** value of 4. Example SIZE Field Values gives an example of **SIZE** values with the corresponding region size and value of N in the **MPU Region Base Address (MPUBASE)** register.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED			XN	RESERVED	AP			RESERVED		TEX			S	C	B	SRD								RESERVED		SIZE					ENABLE

Bits	Field Name	Description	Type	Reset
31:29	RESERVED	Reserved	RO	0x00
28	XN	Instruction access disable **Value** **Description** 0 Instruction fetches are enabled. 1 Instruction fetches are disabled.	R/W	0
27	RESERVED	Reserved	RO	0
26:24	AP	Access privilege For information on using this bit field, see Table 3-5.	R/W	0
23:22	RESERVED	Reserved	RO	0x0
21:19	TEX	Type extension mask For information on using this bit field, see Table 3-3.	R/W	0x0
18	S	Shareable For information on using this bit, see Table 3-3.	R/W	0
17	C	Cacheable For information on using this bit, see Table 3-3.	R/W	0
16	B	Bufferable For information on using this bit, see Table 3-3.	R/W	0
15:8	SRD	Subregion Disable bits **Value** **Description** 0 The corresponding subregion is enabled. 1 The corresponding subregion is disabled. Region sizes of 128 bytes and less do not support subregions. When writing the attributes for such a region, configure the **SRD** field as 0x00. For more information, see Subregions.	R/W	0x00
7:6	RESERVED	Reserved	RO	0x0
5:1	SIZE	Region size mask The **SIZE** field defines the size of the MPU memory region specified by the **MPUNUMBER** register. For more information, see Example SIZE Field Values.	R/W	0x00

Bits	Field Name	Description	Type	Reset
0	ENABLE	Region enable	R/W	0
		Value **Description**		
		0 The region is disabled.		
		1 The region is enabled.		

Example SIZE Field Values

SIZE Encoding	Region Size	Value of N [(1)]	Note
00100b (0x4)	32 B	5	Minimum permitted size
01001b (0x9)	1KB	10	–
10011b (0x13)	1MB	20	–
11101b (0x1D)	1GB	30	–
11111b (0x1F)	4GB	No valid **ADDR** field in the **MPUBASE** register; the region occupies the complete memory map.	Maximum possible size

(1) This refers to the N parameter in the **MPUBASE** register (see MPU Region Base Address (MPUBASE)).

MPU Region Attribute and Size Alias 1 (MPUATTR1), Offset 0xDA8

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0xDA8	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **MPUATTR** register defines the region size and memory attributes of the MPU region specified by the **MPU Region Number (MPUNUMBER)** register and enables that region and any subregions.

The **MPUATTR** register is accessible using word or half word accesses with the most-significant half word holding the region attributes and the least-significant half word holding the region size and the region and subregion enable bits.

The MPU access permission attribute bits, **XN**, **AP**, **TEX**, **S**, **C**, and **B**, control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, then the MPU generates a permission fault.

The **SIZE** field defines the size of the MPU memory region specified by the **MPUNUMBER** register as follows:
(Region size in bytes) = $2^{(SIZE + 1)}$

The smallest permitted region size is 32 bytes, corresponding to a **SIZE** value of 4. Example SIZE Field Values gives example **SIZE** values with the corresponding region size and value of N in the **MPU Region Base Address (MPUBASE)** register.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED			XN	RESERVED	AP			RESERVED		TEX			S	C	B	SRD								RESERVED		SIZE					ENABLE

Bits	Field Name	Description	Type	Reset
31:29	RESERVED	Reserved	RO	0x00
28	XN	Instruction access disable	R/W	0
		Value **Description**		
		0 Instruction fetches are enabled.		
		1 Instruction fetches are disabled.		
27	RESERVED	Reserved	RO	0
26:24	AP	Access privilege For information on using this bit field, see Table 3-5.	R/W	0
23:22	RESERVED	Reserved	RO	0x0
21:19	TEX	Type extension mask For information on using this bit field, see Table 3-3.	R/W	0x0

Bits	Field Name	Description	Type	Reset
18	S	Shareable For information on using this bit, see Table 3-3.	R/W	0
17	C	Cacheable For information on using this bit, see Table 3-3.	R/W	0
16	B	Bufferable For information on using this bit, see Table 3-3.	R/W	0
15:8	SRD	Subregion disable bits **Value** **Description** 0 The corresponding subregion is enabled. 1 The corresponding subregion is disabled. Region sizes of 128 bytes and less do not support subregions. When writing the attributes for such a region, configure the **SRD** field as 0x00. For more information, see Subregions.	R/W	0x00
7:6	RESERVED	Reserved	RO	0x0
5:1	SIZE	Region size mask The **SIZE** field defines the size of the MPU memory region specified by the **MPUNUMBER** register. For more information, see Example SIZE Field Values.	R/W	0x00
0	ENABLE	Region enable **Value** **Description** 0 The region is disabled. 1 The region is enabled.	R/W	0

Example SIZE Field Values

SIZE Encoding	Region Size	Value of N [1]	Note
00100b (0x4)	32 B	5	Minimum permitted size
01001b (0x9)	1KB	10	–
10011b (0x13)	1MB	20	–
11101b (0x1D)	1GB	30	–
11111b (0x1F)	4GB	No valid **ADDR** field in **MPUBASE**; the region occupies the complete memory map.	Maximum possible size

(1) This refers to the N parameter in the **MPUBASE** register (see MPU Region Base Address (MPUBASE)).

MPU Region Attribute and Size Alias 2 (MPUATTR2)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0xDB0	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **MPUATTR** register defines the region size and memory attributes of the MPU region specified by the **MPU Region Number (MPUNUMBER)** register and enables that region and any subregions.

The **MPUATTR** register is accessible using word or half word accesses with the most-significant half word holding the region attributes and the least-significant half word holding the region size and the region and subregion enable bits.

The MPU access permission attribute bits, **XN**, **AP**, **TEX**, **S**, **C**, and **B**, control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, then the MPU generates a permission fault.

The **SIZE** field defines the size of the MPU memory region specified by the **MPUNUMBER** register as follows:
(Region size in bytes) = $2^{(SIZE + 1)}$

The smallest permitted region size is 32 bytes, corresponding to a **SIZE** value of 4. Example SIZE Field Values gives example **SIZE** values with the corresponding region size and value of N in the **MPU Region Base Address (MPUBASE)** register.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED			XN	RESERVED	AP			RESERVED		TEX			S	C	B	SRD								RESERVED		SIZE					ENABLE

Bits	Field Name	Description	Type	Reset
31:29	RESERVED	Reserved	RO	0x00
28	XN	Instruction access disable **Value** **Description** 0 Instruction fetches are enabled. 1 Instruction fetches are disabled.	R/W	0
27	RESERVED	Reserved	RO	0
26:24	AP	Access privilege For information on using this bit field, see Table 3-5.	R/W	0
23:22	RESERVED	Reserved	RO	0x0
21:19	TEX	Type extension mask For information on using this bit field, see Table 3-3.	R/W	0x0
18	S	Shareable For information on using this bit, see Table 3-3.	R/W	0
17	C	Cacheable For information on using this bit, see Table 3-3.	R/W	0
16	B	Bufferable For information on using this bit, see Table 3-3.	R/W	0
15:8	SRD	Subregion disable bits **Value** **Description** 0 The corresponding subregion is enabled. 1 The corresponding subregion is disabled. Region sizes of 128 bytes and less do not support subregions. When writing the attributes for such a region, configure the **SRD** field as 0x00. For more information, see Subregions.	R/W	0x00
7:6	RESERVED	Reserved	RO	0x0
5:1	SIZE	Region size mask The **SIZE** field defines the size of the MPU memory region specified by the **MPUNUMBER** register. For more information, see Example SIZE Field Values.	R/W	0x00
0	ENABLE	Region enable **Value** **Description** 0 The region is disabled. 1 The region is enabled.	R/W	0

Example SIZE Field Values

SIZE Encoding	Region Size	Value of N [1]	Note
00100b (0x4)	32 B	5	Minimum permitted size
01001b (0x9)	1KB	10	-
10011b (0x13)	1MB	20	-
11101b (0x1D)	1GB	30	-
11111b (0x1F)	4GB	No valid **ADDR** field in **MPUBASE**; the region occupies the complete memory map.	Maximum possible size

(1) This refers to the N parameter in the **MPUBASE** register (see MPU Region Base Address (MPUBASE)).

MPU Region Attribute and Size Alias 3 (MPUATTR3)

Address Offset	0xE000 E000	**Reset**	0x0000 0000
Physical Address	0xDB8	**Instance**	

Description

Note: This register can be accessed only from privileged mode.

The **MPUATTR** register defines the region size and memory attributes of the MPU region specified by the **MPU Region Number (MPUNUMBER)** register and enables that region and any subregions.

The **MPUATTR** register is accessible using word or half word accesses with the most-significant half word holding the region attributes and the least-significant half word holding the region size and the region and subregion enable bits.

The MPU access permission attribute bits, **XN**, **AP**, **TEX**, **S**, **C**, and **B**, control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, then the MPU generates a permission fault.

The **SIZE** field defines the size of the MPU memory region specified by the **MPUNUMBER** register as follows:
(Region size in bytes) = $2^{(SIZE + 1)}$

The smallest permitted region size is 32 bytes, corresponding to a **SIZE** value of 4. Example SIZE Field Values gives example **SIZE** values with the corresponding region size and value of N in the **MPU Region Base Address (MPUBASE)** register.

Type R/W

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED			XN	RESERVED	AP			RESERVED		TEX			S	C	B	SRD								RESERVED		SIZE					ENABLE

Bits	Field Name	Description	Type	Reset
31:29	RESERVED	Reserved	RO	0x00
28	XN	Instruction access disable **Value** **Description** 0 Instruction fetches are enabled. 1 Instruction fetches are disabled.	R/W	0
27	RESERVED	Reserved	RO	0
26:24	AP	Access privilege For information on using this bit field, see Table 3-5.	R/W	0
23:22	RESERVED	Reserved	RO	0x0
21:19	TEX	Type extension mask For information on using this bit field, see Table 3-3.	R/W	0x0
18	S	Shareable For information on using this bit, see Table 3-3.	R/W	0
17	C	Cacheable For information on using this bit, see Table 3-3.	R/W	0
16	B	Bufferable For information on using this bit, see Table 3-3.	R/W	0
15:8	SRD	Subregion disable bits **Value** **Description** 0 The corresponding subregion is enabled. 1 The corresponding subregion is disabled. Region sizes of 128 bytes and less do not support subregions. When writing the attributes for such a region, configure the **SRD** field as 0x00. For more information, see Subregions.	R/W	0x00
7:6	RESERVED	Reserved	RO	0x0
5:1	SIZE	Region size mask The **SIZE** field defines the size of the MPU memory region specified by the **MPUNUMBER** register. For more information, see Example SIZE Field Values.	R/W	0x00

Bits	Field Name	Description		Type	Reset
0	ENABLE	Region enable		R/W	0
		Value	**Description**		
		0	The region is disabled.		
		1	The region is enabled.		

Memory Map

This chapter describes the memory map.

Topic **Page**

4.1 Memory Model

This section describes the processor memory map, the behavior of memory accesses, and the bit-banding features. The processor has a fixed memory map that provides up to 4GB of addressable memory.

Table 4-1 provides the memory map for the CC2538 controller. In this manual, register addresses are given as a hexadecimal increment, relative to the base address of the module, as shown in the memory map.

The regions for SRAM and peripherals include bit-band regions. Bit-banding provides atomic operations to bit data (see Section 4.1.5, *Bit-Banding*).

The processor reserves regions of the private peripheral bus (PPB) address range for core peripheral registers (see Chapter 3).

NOTE: Within the memory map, all reserved space returns a bus fault when read or written.

Table 4-1. Memory Map

Start	End	Description	For Details
Memory 0x0000 0000 through 0x3FFF FFFF			
0x0000 0000	0x0001 FFFF	Reserved for ROM	See Chapter 8
0x0020 0000	0x0027 FFFF	On-chip flash	See Chapter 8
0x0028 000	0x1FFF FFFF	Reserved	
0x2000 0000	0x2000 3FFF	Bit-banded on-chip SRAM	See Chapter 8
0x2000 4000	0x2000 7FFF	Bit-banded on-chip low leakage SRAM	See Chapter 8
0x2001 0000	0x200F FFFF	Reserved	
0x2010 0000	0x21FF FFFF	Reserved	
0x2200 0000	0x221F FFFF	Bit-band alias of bit-banded on-chip SRAM starting at 0x2000 0000	See Chapter 8
0x2220 0000	0x3FFF FFFF	Reserved	
Peripherals 0x4000 0000 through 0x4400 67FF			
0x4000 0000	0x4000 7FFF	Reserved	
0x4000 8000	0x4000 8FFF	SSI0	See Chapter 19
0x4000 9000	0x4000 9FFF	SSI1	See Chapter 19
0x4000 A000	0x4000 BFFF	Reserved	
0x4000 C000	0x4000 CFFF	UART0	See Chapter 18
0x4000 D000	0x4000 DFFF	UART1	See Chapter 18
0x4000 E000	0x4001 FFFF	Reserved	
0x4002 0000	0x4002 00FF	I^2C master	See Chapter 20
0x4002 0800	0x4002 08FF	I^2C slave	See Chapter 20
0x4002 9000	0x4002 FFFF	Reserved	
0x4003 0000	0x4003 0FFF	GPTimer 0	See Chapter 11
0x4003 1000	0x4003 1FFF	GPTimer 1	See Chapter 11
0x4003 2000	0x4003 2FFF	GPTimer 2	See Chapter 11
0x4003 3000	0x4003 3FFF	GPTimer 3	See Chapter 11
0x4003 4000	0x4008 7FFF	Reserved	
0x4008 8000	0x4008 8FFF	RF core	
0x4008 9000	0x4008 9FFF	USB	See Chapter 21
0x4008 A000	0x4008 AFFF	Reserved	
0x4008 B000	0x4008 BFFF	AES	See Chapter 22
0x4008 C000	0x400D 1FFF	Reserved	

Table 4-1. Memory Map (continued)

Start	End	Description	For Details
0x400D 2000	0x400D 2FFF	System control	See Chapter 7
0x400D 3000	0x400D 3FFF	Flash memory control	See Chapter 8
0x400D 4000	0x400D 4FFF	I/O muxing	See Section 9.1.1
0x400D 5000	0x400D 5FFF	WDT, sleep timer	See Chapter 14 See Chapter 13
0x400D 6000	0x400D 6FFF	Analog subsystem	See Chapter 17
0x400D 7000	0x400D 7FFF	SoC ADC	See Chapter 15
0x400D 8000	0x400D 8FFF	Reserved	
0x400D 9000	0x400D 9FFF	GPIO port A	See Section 9.2
0x400D A000	0x400D AFFF	GPIO port B	See Section 9.2
0x400D B000	0x400D BFFF	GPIO port C	See Section 9.2
0x400D C000	0x400D CFFF	GPIO port D	See Section 9.2
0x400D D000	0x400F EFFF	Reserved	
0x400F F000	0x400F FFFF	µDMA	See Section 10.1
0x4010 0000	0x43FF FFFF	Reserved	
0x4400 4000	0x4400 4FFF	Authentic PKA	
0x4400 5000	0x4400 5FFF	Reserved	
0x4400 6000	0x4400 67FF	PKA SRAM	
0x4401 0000	0x44010030	CCTEST/OBSSEL	
Private Peripheral Bus 0xE000 0000 through 0xE004 0FFF			
0xE000 0000	0xE000 0FFF	Instrumentation trace macrocell (ITM)	
0xE000 1000	0xE000 1FFF	Data watchpoint and trace (DWT)	
0xE000 2000	0xE000 2FFF	Flash patch and breakpoint (FPB)	
0xE000 3000	0xE000 DFFF	Reserved	
0xE000 E000	0xE000 EFFF	Cortex™-M3 peripherals (SysTick, NVIC, MPU, and SCB)	See Chapter 3
0xE000 F000	0xE003 FFFF	Reserved	
0xE004 0000	0xE004 0FFF	Trace port interface unit (TPIU)	
0xE004 2000	0xFFFF FFFF	Reserved	

4.1.1 Memory Regions, Types, and Attributes

The memory map and the programming of the microprocessor unit (MPU) split the memory map into regions. Each region has a defined memory type, and some regions have additional memory attributes. The memory type and attributes determine the behavior of accesses to the region.

The memory types are:

- Normal: The processor can reorder transactions for efficiency and perform speculative reads.
- Device: The processor preserves a transaction order relative to other transactions to the device or strongly ordered memory types.
- Strongly ordered: The processor preserves a transaction order relative to all other transactions.

The different ordering requirements for both device and strongly ordered memory mean that the memory system can buffer a write to device memory but must not buffer a write to strongly ordered memory.

An additional memory attribute is Execute Never (XN), which means the processor prevents instruction accesses. A fault exception is generated only when an instruction is executed from an XN region.

4.1.2 Memory System Ordering of Memory Accesses

For most memory accesses caused by explicit memory access instructions, the memory system does not ensure that the order in which the accesses complete matches the program order of the instructions, providing the order does not affect the behavior of the instruction sequence. Normally, if correct program execution depends on two memory accesses completing in program order, software must insert a memory barrier instruction between the memory access instructions (see Section 4.1.4, *Software Ordering of Memory Accesses*).

However, the memory system does ensure ordering of accesses to Device and Strongly Ordered memory. For two memory access instructions A1 and A2, if both A1 and A2 are accesses to either device or strongly ordered memory, and if A1 occurs before A2 in program order, A1 is always observed before A2.

4.1.3 Behavior of Memory Accesses

Table 4-2 shows the behavior of accesses to each region in the memory map. For more information on memory types and the XN attribute, see Section 4.1.1, *Memory Regions, Types, and Attributes*. CC2538 devices may have reserved memory areas within the address ranges listed in Table 4-2 (for more information, see Table 4-1).

Table 4-2. Memory Access Behavior

Address Range	Memory Region	Memory Type	Execute Never (XN)	Description
0x0000 0000 through 0x1FFF FFFF	Code	Normal	–	This executable region is for program code. Data can also be stored here.
0x2000 0000 through 0x3FFF FFFF	SRAM	Normal	–	This executable region is for data. Code can also be stored here. This region includes bit band and bit band alias areas (see Table 4-3).
0x4000 0000 through 0x5FFF FFFF	Peripheral	Device	XN	This region includes bit band and bit band alias areas (see Table 4-4).
0x6000 0000 through 0x9FFF FFFF	External RAM	Normal	–	This executable region is for data.
0xA000 0000 through 0xDFFF FFFF	External device	Device	XN	This region is for external device memory.
0xE000 0000 through 0xE00F FFFF	Private peripheral bus	Strongly ordered	XN	This region includes the NVIC, system timer, and system control block.
0xE010 0000 through 0xFFFF FFFF	Reserved	–	–	–

The code, SRAM, and external RAM regions can hold programs. However, it is recommended that programs always use the code region because the Cortex-M3 has separate buses that can perform instruction fetches and data accesses simultaneously.

The MPU can override the default memory access behavior described in this section. For more information, see Section 3.2.4.

The Cortex-M3 prefetches instructions ahead of execution and speculatively prefetches from branch target addresses.

4.1.4 Software Ordering of Memory Accesses

The order of instructions in the program flow does not always ensure the order of the corresponding memory transactions for the following reasons:

- The processor can reorder some memory accesses to improve efficiency, providing this does not affect the behavior of the instruction sequence.
- The processor has multiple bus interfaces.

- Memory or devices in the memory map have different wait states.
- Some memory accesses are buffered or speculative.

Memory System Ordering of Memory Accesses describes the cases where the memory system ensures the order of memory accesses. Otherwise, if the order of memory accesses is critical, software must include memory barrier instructions to force that ordering. The Cortex-M3 has the following memory barrier instructions:

- The Data Memory Barrier (DMB) instruction ensures that outstanding memory transactions complete before subsequent memory transactions.
- The Data Synchronization Barrier (DSB) instruction ensures that outstanding memory transactions complete before subsequent instructions execute.
- The Instruction Synchronization Barrier (ISB) instruction ensures that the effect of all completed memory transactions is recognizable by subsequent instructions.

Memory barrier instructions can be used in the following situations:

- MPU programming
  - If the MPU settings are changed and the change must be effective on the next instruction, use a DSB instruction to ensure the effect of the MPU occurs immediately at the end of context switching.
  - Use a DSB instruction followed by an ISB instruction to ensure the new MPU setting takes effect immediately after programming the MPU region or regions, if the MPU configuration code was accessed using a branch or call. If the MPU configuration code is entered using exception mechanisms, then an ISB instruction is not required.
- Vector table – If the program changes an entry in the vector table and then enables the corresponding exception, use a DMB instruction between the operations. The DMB instruction ensures that if the exception is taken immediately after being enabled, the processor uses the new exception vector.
- Self-modifying code – If a program contains self-modifying code, use an ISB instruction immediately after the code modification in the program. The ISB instruction ensures that execution of a subsequent instruction uses the updated program.
- Memory map switching – If the system contains a memory map switching mechanism, use a DSB instruction after switching the memory map in the program. The DSB instruction ensures that execution of a subsequent instruction uses the updated memory map.
- Dynamic exception priority change – When an exception priority must change when the exception is pending or active, use DSB instructions after the change. The change then takes effect when the DSB instruction completes.

Memory accesses to strongly ordered memory, such as the system control block, do not require the use of DMB instructions.

For more information on the memory barrier instructions, see the *Cortex-M3 Instruction Set Technical User's Manual*.

4.1.5 Bit-Banding

A bit-band region maps each word in a bit-band alias region to a single bit in the bit-band region. The bit-band regions occupy the lowest 1MB of the SRAM and peripheral memory regions. Accesses to the 32-MB SRAM alias region map to the 1-MB SRAM bit-band region, as shown in Table 4-3. Accesses to the 32-MB peripheral alias region map to the 1-MB peripheral bit-band region, as shown in Table 4-4. Figure 4-1 shows the specific address range of the bit-band regions.

NOTE: A word access to the SRAM or the peripheral bit-band alias region maps to a single bit in the SRAM or peripheral bit-band region.

A word access to a bit-band address results in a word access to the underlying memory, and similarly for halfword and byte accesses. This allows bit-band accesses to match the access requirements of the underlying peripheral.

Table 4-3. SRAM Memory Bit-Banding Regions

Address Range	Memory Region	Instruction and Data Accesses
0x2000 0000 through 0x200F FFFF	SRAM bit-band region	Direct accesses to this memory range behave as SRAM memory accesses, but this region is also bit addressable through bit-band alias.
0x2200 0000 through 0x23FF FFFF	SRAM bit-band alias	Data accesses to this region are remapped to bit-band region. A write operation is performed as read-modify-write. Instruction accesses are not remapped.

Table 4-4. Peripheral Memory Bit-Banding Regions

Address Range	Memory Region	Instruction and Data Accesses
0x4000 0000 through 0x400F FFFF	Peripheral bit-band region	Direct accesses to this memory range behave as peripheral memory accesses, but this region is also bit addressable through bit-band alias.
0x4200 0000 through 0x43FF FFFF	Peripheral bit-band alias	Data accesses to this region are remapped to bit-band region. A write operation is performed as read-modify-write. Instruction accesses are not permitted.

The following formula shows how the alias region maps onto the bit-band region:

```
bit_word_offset = (byte_offset x 32) + (bit_number x 4)
bit_word_addr = bit_band_base + bit_word_offset
```

where:

- bit_word_offset
  The position of the target bit in the bit-band memory region
- bit_word_addr
  The address of the word in the alias memory region that maps to the targeted bit
- bit_band_base
  The starting address of the alias region
- byte_offset
  The number of the byte in the bit-band region that contains the targeted bit
- bit_number
  The bit position, [7:0], of the targeted bit

Figure 4-1 shows examples of bit-band mapping between the SRAM bit-band alias region and the SRAM bit-band region:

- The alias word at 0x23FF FFE0 maps to bit 0 of the bit-band byte at 0x200F FFFF:
  `0x23FF FFE0 = 0x2200 0000 + (0x000F FFFF*32) + (0*4)`
- The alias word at 0x23FF FFFC maps to bit 7 of the bit-band byte at 0x200F FFFF: `0x23FF FFFC = 0x2200 0000 + (0x000F FFFF*32) + (7*4)`
- The alias word at 0x2200 0000 maps to bit 0 of the bit-band byte at 0x2000 0000:
  `0x2200 0000 = 0x2200 0000 + (0*32) + (0*4)`
- The alias word at 0x2200 001C maps to bit 7 of the bit-band byte at 0x2000 0000:
  `0x2200 001C = 0x2200 0000 + (0*32) + (7*4)`

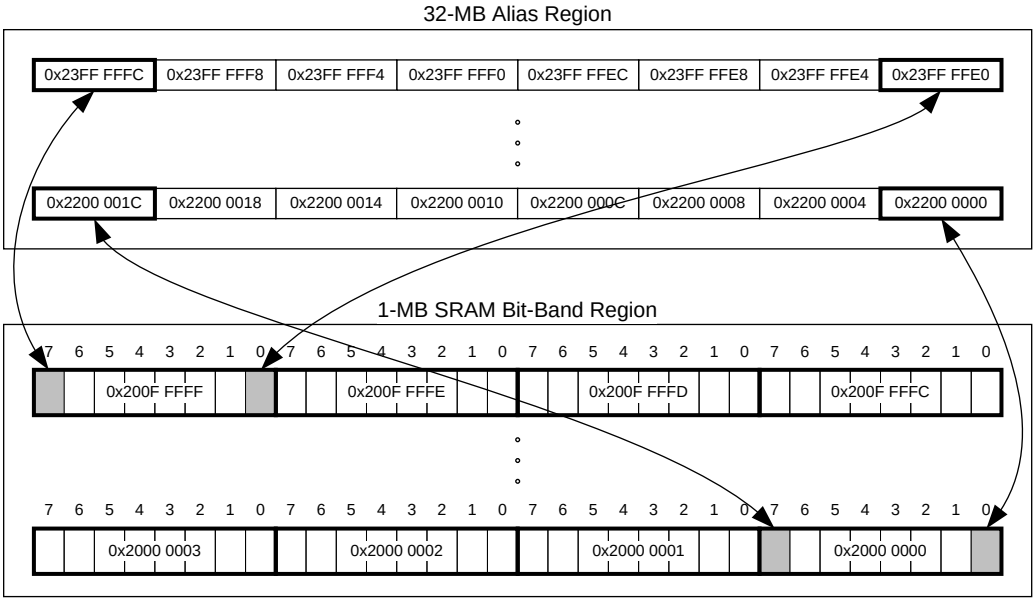


Figure 4-1. Bit-Band Mapping

4.1.5.1 Directly Accessing an Alias Region

Writing to a word in the alias region updates a single bit in the bit-band region.

Bit [0] of the value written to a word in the alias region determines the value written to the targeted bit in the bit-band region. Writing a value with bit 0 set writes a 1 to the bit-band bit, and writing a value with bit 0 clear writes a 0 to the bit-band bit.

Bits [31:1] of the alias word have no effect on the bit-band bit. Writing 0x01 has the same effect as writing 0xFF. Writing 0x00 has the same effect as writing 0x0E.

When reading a word in the alias region, 0x0000 0000 indicates that the targeted bit in the bit-band region is clear and 0x0000 0001 indicates that the targeted bit in the bit-band region is set.

4.1.5.2 Directly Accessing a Bit-Band Region

Section 4.1.3, *Behavior of Memory Accesses*, describes the behavior of direct byte, halfword, or word accesses to the bit-band regions.

4.1.6 Data Storage

The processor views memory as a linear collection of bytes numbered in ascending order from zero. For example, bytes 0–3 hold the first stored word, and bytes 4–7 hold the second stored word. Data is stored in little-endian format, with the least-significant byte (LSByte) of a word stored at the lowest-numbered byte, and the most-significant byte (MSByte) stored at the highest-numbered byte. Figure 4-2 shows how data is stored.

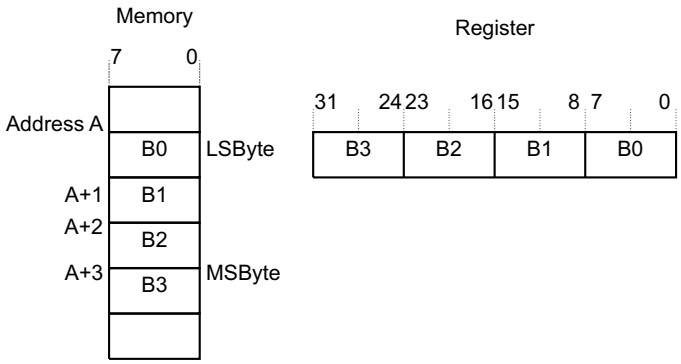


Figure 4-2. Data Storage

4.1.7 Synchronization Primitives

The Cortex-M3 instruction set includes pairs of synchronization primitives which provide a nonblocking mechanism that a thread or process can use to obtain exclusive access to a memory location. Software can use these primitives to perform an ensured read-modify-write memory update sequence or for a semaphore mechanism.

A pair of synchronization primitives consists of the following instructions:

- A Load-Exclusive instruction, which is used to read the value of a memory location and requests exclusive access to that location.
- A Store-Exclusive instruction, which is used to attempt to write to the same memory location and returns a status bit to a register. If this status bit is clear, it indicates that the thread or process gained exclusive access to the memory and the write succeeds; if this status bit is set, it indicates that the thread or process did not gain exclusive access to the memory and no write was performed.

The pairs of Load-Exclusive and Store-Exclusive instructions are:

- The word instructions LDREX and STREX
- The halfword instructions LDREXH and STREXH
- The byte instructions LDREXB and STREXB

Software must use a Load-Exclusive instruction with the corresponding Store-Exclusive instruction.

To perform an exclusive read-modify-write of a memory location, software must:

1. Use a Load-Exclusive instruction to read the value of the location.
2. Modify the value, as required.
3. Use a Store-Exclusive instruction to attempt to write the new value back to the memory location.
4. Test the returned status bit. If the status bit is clear, the read-modify-write completed successfully. If the status bit is set, no write was performed, which indicates that the value returned at Step 1 might be out of date. Software must retry the entire read-modify-write sequence.

Software can use the synchronization primitives to implement a semaphore as follows:

1. Use a Load-Exclusive instruction to read from the semaphore address to check whether the semaphore is free.
2. If the semaphore is free, use a Store-Exclusive to write the claim value to the semaphore address.
3. If the returned status bit from Step 2 indicates that the Store-Exclusive succeeded, then software has claimed the semaphore. However, if the Store-Exclusive failed, another process might have claimed the semaphore after the software performed Step 1.

The Cortex-M3 includes an exclusive access monitor that tags the fact that the processor has executed a Load-Exclusive instruction. The processor removes its exclusive access tag if one of the following conditions occurs:

- The processor executes a CLREX instruction.

- The processor executes a Store-Exclusive instruction, regardless of whether the write succeeds.
- An exception occurs, which means the processor can resolve semaphore conflicts between different threads.

For more information about the synchronization primitive instructions, see the *Cortex-M3 Instruction Set Technical User's Manual*.

Interrupts

This chapter describes the interrupts.

Topic **Page**

5.1 Exception Model

The ARM® Cortex™-M3 processor and the nested vectored interrupt controller (NVIC) prioritize and handle all exceptions in handler mode. The processor state is automatically stored to the stack on an exception and automatically restored from the stack at the end of the interrupt service routine (ISR). The vector is fetched in parallel to state saving, thus enabling efficient interrupt entry. The processor supports tail-chaining, which enables back-to-back interrupts to be performed without the overhead of state saving and restoration.

The CC2538 has 48 interrupts that are spread across a range of 147 possible ARM® Cortex™-M3 interrupt inputs in a regular interrupt map setting.

The CC2538 also provides an alternate interrupt map setting for a more compact mapping arrangement. The alternate mapping can be enabled by setting **SYS_CTRL_I_MAP.ALTMAP** bit. With the alternate interrupt map enabled, the system interrupts are remapped to a smaller range between 0 and 47.

Table 5-1 lists all exception types. Software can set eight priority levels on seven of these exceptions (system handlers) as well as on CC2538 interrupts (listed in Table 5-2).

Priorities on the system handlers are set with the NVIC **System Handler Priority n (SYSPRIn)** registers. Interrupts are enabled through the NVIC **Interrupt Set Enable n (ENn)** register and prioritized with the NVIC **Interrupt Priority n (PRIn)** registers. Priorities can be grouped by splitting priority levels into preemption priorities and subpriorities. All the interrupt registers are described in Section 3.2.2.

Internally, the highest user-programmable priority (0) is treated as third priority, after a reset, and a hard fault, in that order. Note that 0 is the default priority for all the programmable priorities.

CAUTION

After a write to clear an interrupt source, it may take several processor cycles for the NVIC to detect the interrupt source deassert. Thus if the interrupt clear is done as the last action in an interrupt handler, it is possible for the interrupt handler to complete while the NVIC detects the interrupt as still asserted, causing the interrupt handler to be re-entered errantly. This situation can be avoided by either clearing the interrupt source at the beginning of the interrupt handler or by performing a read or write after the write to clear the interrupt source (and flush the write buffer).

For more information on exceptions and interrupts, see Section 3.2.2.

5.1.1 Exception States

Each exception is in one of the following states:

- **Inactive:** The exception is not active and not pending.
- **Pending**: The exception is waiting to be serviced by the processor. An interrupt request from a peripheral or from software can change the state of the corresponding interrupt to pending.
- **Active**: An exception is being serviced by the processor but has not completed. Note: An exception handler can interrupt the execution of another exception handler. In this case, both exceptions are in the active state.
- **Active and Pending**: The exception is being serviced by the processor, and there is a pending exception from the same source.

5.1.2 Exception Types

The exception types are:

- **Reset**: Reset is invoked on power up or a warm reset. The exception model treats reset as a special form of exception. When reset is asserted, the operation of the processor stops, potentially at any point in an instruction. When reset is deasserted, execution restarts from the address provided by the reset

entry in the vector table. Execution restarts as privileged execution in thread mode.

- **Hard Fault:** A hard fault is an exception that occurs because of an error during exception processing, or because an exception cannot be managed by any other exception mechanism. Hard faults have a fixed priority of –1, meaning they have higher priority than any exception with configurable priority.
- **Memory Management Fault:** A memory management fault is an exception that occurs because of a memory protection-related fault, including access violation and no match. The MPU or the fixed memory protection constraints determine this fault, for both instruction and data memory transactions. This fault is used to abort instruction accesses to Execute Never (XN) memory regions, even if the MPU is disabled.
- **Bus Fault:** A bus fault is an exception that occurs because of a memory-related fault for an instruction or data memory transaction such as a prefetch fault or a memory access fault. This fault can be enabled or disabled.
- **Usage Fault:** A usage fault is an exception that occurs because of a fault related to instruction execution, such as:
  - An undefined instruction
  - An illegal unaligned access
  - Invalid state on instruction execution
  - An error on exception return

  An unaligned address on a word or halfword memory access or division by 0 can cause a usage fault when the core is properly configured.
- **SVCall:** A supervisor call (SVC) is an exception that is triggered by the SVC instruction. In an OS environment, applications can use SVC instructions to access OS kernel functions and device drivers.
- **Debug Monitor:** This exception is caused by the debug monitor (when not halting). This exception is active only when enabled. This exception does not activate if it is a lower priority than the current activation.
- **PendSV:** PendSV is a pendable, interrupt-driven request for system-level service. In an OS environment, use PendSV for context switching when no other exception is active. PendSV is triggered using the **Interrupt Control and State (INTCTRL)** register.
- **SysTick:** A SysTick exception is an exception that the system timer generates when it reaches 0 when it is enabled to generate an interrupt. Software can also generate a SysTick exception using the **Interrupt Control and State (INTCTRL)** register. In an OS environment, the processor can use this exception as system tick.
- **Interrupt (IRQ):** An interrupt, or IRQ, is an exception signaled by a peripheral or generated by a software request and fed through the NVIC (prioritized). All interrupts are asynchronous to instruction execution. In the system, peripherals use interrupts to communicate with the processor. Table 5-2 lists the interrupts on the CC2538 controller.

For an asynchronous exception, other than reset, the processor can execute another instruction between when the exception is triggered and when the processor enters the exception handler.

Privileged software can disable the exceptions that Table 5-1 shows as having configurable priority (see the **SYSHNDCTRL** register on System Handler Control and State (SYSHNDCTRL) and the **DIS0** register in Interrupt 0–31 Clear Enable (DIS0)).

For more information about hard faults, memory management faults, bus faults, and usage faults, see Section 5.2.

Table 5-1. Exception Types

Exception Type	Vector Number	Priority[1]	Vector Address or Offset[2]	Activation
–	0	–	0x0000 0000	Stack top is loaded from the first entry of the vector table on reset.
Reset	1	–3 (highest)	0x0000 0004	Asynchronous

(1) 0 is the default priority for all the programmable priorities.
(2) See Section 5.1.4.

Table 5-1. Exception Types (continued)

Exception Type	Vector Number	Priority[1]	Vector Address or Offset[2]	Activation
–	–	–	–	–
Hard fault	3	–1	0x0000 000C	–
Memory management	4	Programmable[3]	0x0000 0010	Synchronous
Bus fault	5	Programmable	0x0000 0014	Synchronous when precise and asynchronous when imprecise
Usage fault	6	Programmable	0x0000 0018	Synchronous
–	7–10	–	–	Reserved
SVCall	11	Programmable	0x0000 002C	Synchronous
Debug monitor	12	Programmable	0x0000 0030	Synchronous
–	13	–	–	Reserved
PendSV	14	Programmable	0x0000 0038	Asynchronous
SysTick	15	Programmable	0x0000 003C	Asynchronous
Interrupts	16 and above	Programmable[4]	0x0000 0040 and above	Asynchronous

(3) See **SYSPRI1** on System Handler Priority 1 (SYSPRI1).
(4) See **PRIn** registers on Interrupt 0–3 Priority (PRI0).

Table 5-2. Interrupts

Vector Number	Interrupt Number (Bit in Interrupt Registers)	Vector Address or Offset	Description
0-15	-	0x0000 0000 - 0x0000 003C	Processor exceptions
16	0	0x0000 0040	GPIO port A
17	1	0x0000 0044	GPIO port B
18	2	0x0000 0048	GPIO port C
19	3	0x0000 004C	GPIO port D
20	4	0x0000 0050	Reserved
21	5	0x0000 0054	UART0
22	6	0x0000 0058	UART1
23	7	0x0000 005C	SSI0
24	8	0x0000 0060	I^2C
25	9	0x0000 0064	Reserved
26	10	0x0000 0068	Reserved
27	11	0x0000 006C	Reserved
28	12	0x0000 0070	Reserved
29	13	0x0000 0074	Reserved
30	14	0x0000 0078	ADC
31	15	0x0000 007C	Reserved
32	16	0x0000 0080	Reserved
33	17	0x0000 0084	Reserved
34	18	0x0000 0088	Watchdog Timer
35	19	0x0000 008C	GPTimer 0A
36	20	0x0000 0090	GPTimer 0B
37	21	0x0000 0094	GPTimer 1A
38	22	0x0000 0098	GPTimer 1B
39	23	0x0000 009C	GPTimer 2A
40	24	0x0000 00A0	GPTimer 2B
41	25	0x0000 00A4	Analog Comparator

Table 5-2. Interrupts (continued)

Vector Number	Interrupt Number (Bit in Interrupt Registers)	Vector Address or Offset	Description
42	26	0x0000 00A8	RF TX/RX (Alternate)
43	27	0x0000 00AC	RF Error (Alternate)
44	28	0x0000 00B0	System Control
45	29	0x0000 00B4	Flash memory control
46	30	0x0000 00B8	AES (Alternate)
47	31	0x0000 00BC	PKA (Alternate)
48	32	0x0000 00C0	SM Timer (Alternate)
49	33	0x0000 00C4	MAC Timer (Alternate)
50	34	0x0000 00C8	SSI1
51	35	0x0000 00CC	GPTimer 3A
52	36	0x0000 00D0	GPTimer 3B
53	37	0x0000 00D4	Reserved
55	39	0x0000 00DC	Reserved
59	43	0x0000 00EC	Reserved
60	44	0x0000 00F0	Reserved
61	45	0x0000 00F4	Reserved
62	46	0x0000 00F8	µDMA software
63	47	0x0000 00FC	µDMA error
64	48	0x0000 0100	Reserved
65	49	0x0000 0104	Reserved
66	50	0x0000 0108	Reserved
67	51	0x0000 010C	Reserved
68	52	0x0000 0110	Reserved
69	53	0x0000 0114	Reserved
70	54	0x0000 0118	Reserved
71	55	0x0000 011C	Reserved
72	56	0x0000 0120	Reserved
73	57	0x0000 0124	Reserved
74	58	0x0000 0128	Reserved
75	59	0x0000 012C	Reserved
76	60	0x0000 0130	Reserved
77	61	0x0000 0134	Reserved
78	62	0x0000 0138	Reserved
79	63	0x0000 013C	Reserved
80	64	0x0000 0140	Reserved
81	65	0x0000 0144	Reserved
82	66	0x0000 0148	Reserved
83	67	0x0000 014C	Reserved
84	68	0x0000 0150	Reserved
85	69	0x0000 0154	Reserved
86	70	0x0000 0158	Reserved
87	71	0x0000 015C	Reserved
88	72	0x0000 0160	Reserved
89	73	0x0000 0164	Reserved
90	74	0x0000 0168	Reserved
91	75	0x0000 016C	Reserved
92	76	0x0000 070	Reserved

Table 5-2. Interrupts (continued)

Vector Number	Interrupt Number (Bit in Interrupt Registers)	Vector Address or Offset	Description
93	77	0x0000 0174	Reserved
94	78	0x0000 0178	Reserved
95	79	0x0000 017C	Reserved
96	80	0x0000 0180	Reserved
97	81	0x0000 0184	Reserved
98	82	0x0000 0188	Reserved
99	83	0x0000 018C	Reserved
100	84	0x0000 0190	Reserved
101	85	0x0000 0194	Reserved
102	86	0x0000 0198	Reserved
103	87	0x0000 019C	Reserved
104	88	0x0000 0200	Reserved
105	89	0x0000 0204	Reserved
106	90	0x0000 0208	Reserved
107	91	0x0000 020C	Reserved
108	92	0x0000 0210	Reserved
109	93	0x0000 0214	Reserved
110	94	0x0000 0218	Reserved
111	95	0x0000 021C	Reserved
112	96	0x0000 0220	Reserved
113	97	0x0000 0224	Reserved
114	98	0x0000 0228	Reserved
115	99	0x0000 022C	Reserved
116	100	0x0000 0230	Reserved
117	101	0x0000 0234	Reserved
118	102	0x0000 0238	Reserved
119	103	0x0000 023C	Reserved
120	104	0x0000 0240	Reserved
121	105	0x0000 0244	Reserved
123	106	0x0000 0248	Reserved
124	107	0x0000 024C	Reserved
125	108	0x0000 0250	Reserved
126	109	0x0000 0254	Reserved
127	110	0x0000 0258	Reserved
128	111	0x0000 025C	Reserved
129	112	0x0000 0260	Reserved
130	113	0x0000 0264	Reserved
131	114	0x0000 0268	Reserved
132	115	0x0000 026C	Reserved
133	116	0x0000 0270	Reserved
134	117	0x0000 0274	Reserved
135	118	0x0000 0278	Reserved
136	119	0x0000 027C	Reserved
137	120	0x0000 0280	Reserved
138	121	0x0000 0284	Reserved
139	123	0x0000 0288	Reserved
140	124	0x0000 028C	Reserved

Table 5-2. Interrupts (continued)

Vector Number	Interrupt Number (Bit in Interrupt Registers)	Vector Address or Offset	Description
141	125	0x0000 0290	Reserved
142	126	0x0000 0294	Reserved
143	127	0x0000 0298	Reserved
144	128	0x0000 029C	Reserved
145	129	0x0000 0300	Reserved
146	130	0x0000 0304	Reserved
147	131	0x0000 0308	Reserved
148	132	0x0000 030C	Reserved
149	133	0x0000 0310	Reserved
150	134	0x0000 0314	Reserved
151	135	0x0000 0318	Reserved
152	136	0x0000 031C	Reserved
153	137	0x0000 0320	Reserved
154	138	0x0000 0324	Reserved
155	139	0x0000 0328	Reserved
156	140	0x0000 032C	USB
157	141	0x0000 0330	RF Core Rx/Tx
158	142	0x0000 0334	RF Core Error
159	143	0x0000 0348	AES
160	144	0x0000 034C	PKA
161	145	0x0000 0350	SM Timer
162	146	0x0000 0354	MAC Timer
163	147	0x0000 0358	Reserved

5.1.3 Exception Handlers

The processor handles exceptions using:

- **Interrupt Service Routines (ISRs):** Interrupts (IRQx) are the exceptions handled by ISRs.
- **Fault Handlers:** Hard fault, memory management fault, usage fault, and bus fault are fault exceptions handled by the fault handlers.
- **System Handlers:** PendSV, SVCall, SysTick, and the fault exceptions are all system exceptions that are handled by system handlers.

5.1.4 Vector Table

The vector table contains the reset value of the stack pointer and the start addresses, also called exception vectors, for all exception handlers. The vector table is constructed using the vector address or offset shown in Table 5-1. Figure 5-1 shows the order of the exception vectors in the vector table. The least-significant bit (LSB) of each vector must be 1, indicating that the exception handler is Thumb® code.

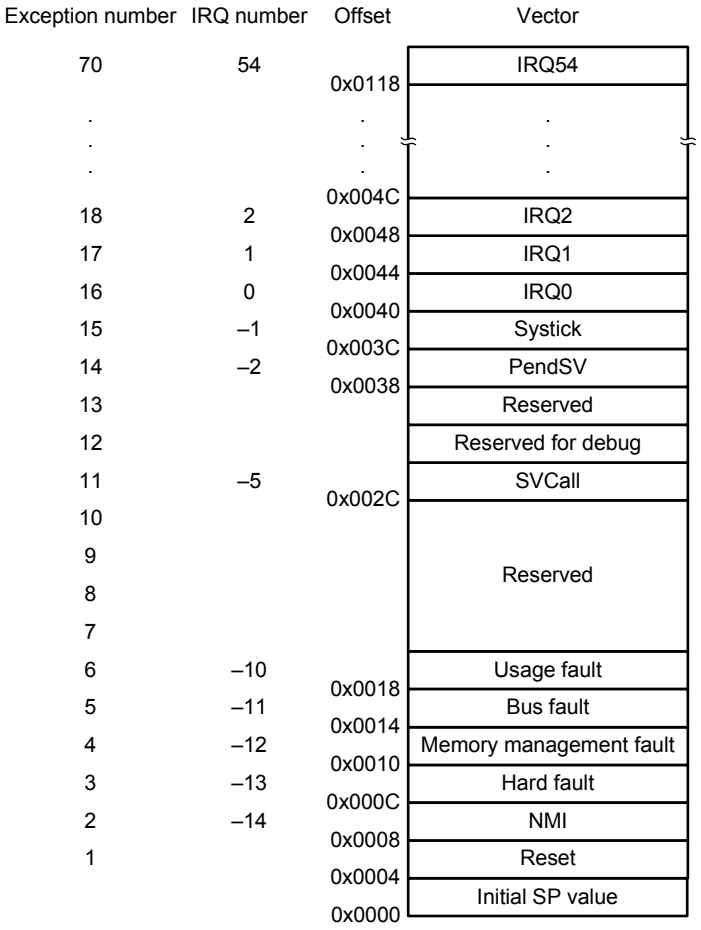


Figure 5-1. Vector Table

On system reset, the vector table is fixed at address 0x0000 0000. Privileged software can write to the **Vector Table Offset (VTABLE)** register to relocate the vector table start address to a different memory location, in the range 0x0000 0200 to 0x3FFF FE00 (see Section 5.1.4). When configuring the **VTABLE** register, the offset must be aligned on a 512-byte boundary.

5.1.5 Exception Priorities

As Table 5-1 shows, all exceptions have an associated priority, with a lower priority value indicating a higher priority and configurable priorities for all exceptions except reset, and hard fault. If software does not configure any priorities, then all exceptions with a configurable priority have a priority of 0. For information about configuring exception priorities, see System Handler Priority 1 (SYSPRI1) and Interrupt 0–3 Priority (PRI0).

NOTE: Configurable priority values for the CC2538 implementation are in the range 0 to 7. This means that the Reset and Hard fault exceptions, with fixed negative priority values, always have higher priority than any other exception.

For example, assigning a higher priority value to IRQ[0] and a lower priority value to IRQ[1] means that IRQ[1] has higher priority than IRQ[0]. If IRQ[1] and IRQ[0] are asserted, IRQ[1] is processed before IRQ[0].

If multiple pending exceptions have the same priority, the pending exception with the lowest exception number takes precedence. For example, if IRQ[0] and IRQ[1] are pending and have the same priority, then IRQ[0] is processed before IRQ[1].

When the processor is executing an exception handler, the exception handler is preempted if a higher priority exception occurs. If an exception occurs with the same priority as the exception being handled, the handler is not preempted, irrespective of the exception number. However, the status of the new interrupt changes to pending.

5.1.6 Interrupt Priority Grouping

To increase priority control in systems with interrupts, the NVIC supports priority grouping. This grouping divides each interrupt priority register entry into two fields:

- An upper field that defines the group priority
- A lower field that defines a subpriority within the group

Only the group priority determines preemption of interrupt exceptions. When the processor is executing an interrupt exception handler, another interrupt with the same group priority as the interrupt being handled does not preempt the handler.

If multiple pending interrupts have the same group priority, the subpriority field determines the order in which they are processed. If multiple pending interrupts have the same group priority and subpriority, the interrupt with the lowest IRQ number is processed first.

For information about splitting the interrupt priority fields into group priority and subpriority, see Application Interrupt and Reset Control (APINT).

5.1.7 Exception Entry and Return

Descriptions of exception handling use the following terms:

- **Preemption:** When the processor is executing an exception handler, an exception can preempt the exception handler if its priority is higher than the priority of the exception being handled. For more information about preemption by an interrupt, see Section 5.1.6. When one exception preempts another, the exceptions are called nested exceptions. For more information, see Section 5.1.7.1.
- **Return:** Return occurs when the exception handler is completed, and there is no pending exception with sufficient priority to be serviced and the completed exception handler was not handling a late-arriving exception. The processor pops the stack and restores the processor state to the state it had before the interrupt occurred. For more information, see Section 5.1.7.2.
- **Tail-Chaining:** This mechanism speeds up exception servicing. When an exception handler completes, if a pending exception meets the requirements for exception entry, the stack pop is skipped and control transfers to the new exception handler.
- **Late-Arriving:** This mechanism speeds up preemption. If a higher-priority exception occurs during state saving for a previous exception, the processor switches to handle the higher-priority exception and initiates the vector fetch for that exception. State saving is not affected by late arrival because the state saved is the same for both exceptions. Therefore, the state saving continues uninterrupted. The processor can accept a late-arriving exception until the first instruction of the exception handler of the original exception enters the execute stage of the processor. When the late-arriving exception returns from the exception handler, the normal tail-chaining rules apply.

5.1.7.1 Exception Entry

Exception entry occurs when there is a pending exception with sufficient priority and either the processor is in thread mode or the new exception is of higher priority than the exception being handled, in which case the new exception preempts the original exception.

When one exception preempts another, the exceptions are nested.

Sufficient priority means the exception has more priority than any limits set by the mask registers (see **PRIMASK** on Priority Mask Register (PRIMASK), **FAULTMASK** on Fault Mask Register (FAULTMASK), and **BASEPRI** on Base Priority Mask Register (BASEPRI)). An exception with less priority than this is pending but is not handled by the processor.

When the processor takes an exception, unless the exception is a tail-chained or a late-arriving exception, the processor pushes information onto the current stack. This operation is referred to as stacking and the structure of eight data words is referred to as stack frame.

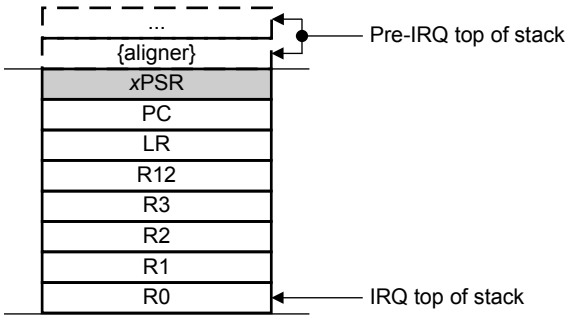


Figure 5-2. Exception Stack Frame

Immediately after stacking, the stack pointer indicates the lowest address in the stack frame.

The stack frame includes the return address, which is the address of the next instruction in the interrupted program. This value is restored to the **PC** at exception return so that the interrupted program resumes.

In parallel to the stacking operation, the processor performs a vector fetch that reads the exception handler start address from the vector table. When stacking completes, the processor starts executing the exception handler. At the same time, the processor writes an EXC_RETURN value to the **LR**, indicating which stack pointer corresponds to the stack frame and what operation mode the processor was in before the entry occurred.

If no higher-priority exception occurs during exception entry, the processor starts executing the exception handler and automatically changes the status of the corresponding pending interrupt to active.

If another higher-priority exception occurs during exception entry, known as late arrival, the processor starts executing the exception handler for this exception and does not change the pending status of the earlier exception.

5.1.7.2 Exception Return

Exception return occurs when the processor is in handler mode and executes one of the following instructions to load the EXC_RETURN value into the **PC**:

- An **LDM** or **POP** instruction that loads the **PC**
- A **BX** instruction using any register
- An **LDR** instruction with the **PC** as the destination

EXC_RETURN is the value loaded into the **LR** on exception entry. The exception mechanism relies on this value to detect when the processor completes an exception handler. The lowest 4 bits of this value provide information on the return stack and processor mode. Table 5-3 shows the EXC_RETURN values with a description of the exception return behavior.

EXC_RETURN bits 31:4 are all set. When this value is loaded into the **PC**, it indicates to the processor that the exception is complete, and the processor initiates the appropriate exception return sequence.

Table 5-3. Exception Return Behavior

EXC_RETURN[31:0]	Description
0xFFFF FFF0	Reserved
0xFFFF FFF1	Return to Handler mode. Exception return uses state from **MSP**. Execution uses **MSP** after return.
0xFFFF FFF2–0xFFFF FFF8	Reserved
0xFFFF FFF9	Return to Thread mode. Exception return uses state from **MSP**. Execution uses **MSP** after return.
0xFFFF FFFA–0xFFFF FFFC	Reserved

Table 5-3. Exception Return Behavior (continued)

EXC_RETURN[31:0]	Description
0xFFFF FFFD	Return to Thread mode. Exception return uses state from **PSP**. Execution uses **PSP** after return.
0xFFFF FFFE–0xFFFF FFFF	Reserved

5.2 Fault Handling

Faults are a subset of the exceptions (see Section 5.1). The following conditions generate a fault:

- A bus error on an instruction fetch or vector table load or a data access
- An internally detected error such as an undefined instruction or an attempt to change state with a **BX** instruction
- Attempting to execute an instruction from a memory region marked as nonexecutable (XN)
- An MPU fault because of a privilege violation or an attempt to access an unmanaged region

5.2.1 Fault Types

Table 5-4 shows the types of fault, the handler used for the fault, the corresponding fault status register, and the register bit that indicates the fault has occurred. For more information about the fault status registers, see Configurable Fault Status (FAULTSTAT).

Table 5-4. Faults

Fault	Handler	Fault Status Register	Bit Name
Bus error on a vector read	Hard fault	**Hard Fault Status (HFAULTSTAT)**	**VECT**
Fault escalated to a hard fault	Hard fault	**Hard Fault Status (HFAULTSTAT)**	**FORCED**
MPU or default memory mismatch on instruction access	Memory management fault	**Memory Management Fault Status (MFAULTSTAT)**	**IERR** (1)
MPU or default memory mismatch on data access	Memory management fault	**Memory Management Fault Status (MFAULTSTAT)**	**DERR**
MPU or default memory mismatch on exception stacking	Memory management fault	**Memory Management Fault Status (MFAULTSTAT)**	**MSTKE**
MPU or default memory mismatch on exception unstacking	Memory management fault	**Memory Management Fault Status (MFAULTSTAT)**	**MUSTKE**
Bus error during exception stacking	Bus fault	**Bus Fault Status (BFAULTSTAT)**	**BSTKE**
Bus error during exception unstacking	Bus fault	**Bus Fault Status (BFAULTSTAT)**	**BUSTKE**
Bus error during instruction prefetch	Bus fault	**Bus Fault Status (BFAULTSTAT)**	**IBUS**
Precise data bus error	Bus fault	**Bus Fault Status (BFAULTSTAT)**	**PRECISE**
Imprecise data bus error	Bus fault	**Bus Fault Status (BFAULTSTAT)**	**IMPRE**
Attempt to access a coprocessor	Usage fault	**Usage Fault Status (UFAULTSTAT)**	**NOCP**
Undefined instruction	Usage fault	**Usage Fault Status (UFAULTSTAT)**	**UNDEF**
Attempt to enter an invalid instruction set state (2)	Usage fault	**Usage Fault Status (UFAULTSTAT)**	**INVSTAT**

(1) Occurs on an access to an XN region even if the MPU is disabled.

(2) Attempting to use an instruction set other than the Thumb instruction set, or returning to a non load-store-multiple instruction with **ICI** continuation.

Table 5-4. Faults (continued)

Fault	Handler	Fault Status Register	Bit Name
Invalid EXC_RETURN value	Usage fault	**Usage Fault Status (UFAULTSTAT)**	**INVPC**
Illegal unaligned load or store	Usage fault	**Usage Fault Status (UFAULTSTAT)**	**UNALIGN**
Divide by 0	Usage fault	**Usage Fault Status (UFAULTSTAT)**	**DIV0**

5.2.2 Fault Escalation and Hard Faults

All fault exceptions except for hard fault have configurable exception priority (see **SYSPRI1** on System Handler Priority 1 (SYSPRI1)). Software can disable execution of the handlers for these faults (see **SYSHNDCTRL** on System Handler Control and State (SYSHNDCTRL)).

Usually, the exception priority, together with the values of the exception mask registers, determines whether the processor enters the fault handler, and whether a fault handler can preempt another fault handler as described in Section 5.1.

In some situations, a fault with configurable priority is treated as a hard fault. This process is called priority escalation, and the fault is described as escalated to hard fault. Escalation to hard fault occurs when:

- A fault handler causes the same kind of fault as the one it is servicing. This escalation to hard fault occurs because a fault handler cannot preempt itself because it must have the same priority as the current priority level.
- A fault handler causes a fault with the same or lower priority as the fault it is servicing. This situation happens because the handler for the new fault cannot preempt the fault handler that is currently executing.
- An exception handler causes a fault for which the priority is the same as or lower than the exception that is currently executing.
- A fault occurs and the handler for that fault is not enabled.

If a bus fault occurs during a stack push when entering a bus fault handler, the bus fault does not escalate to a hard fault. Thus, if a corrupted stack causes a fault, the fault handler executes even though the stack push for the handler failed. The fault handler operates but the stack contents are corrupted.

5.2.3 Fault Status Registers and Fault Address Registers

The fault status registers indicate the cause of a fault. For bus faults and memory management faults, the fault address register indicates the address accessed by the operation that caused the fault, as shown in Table 5-5.

Table 5-5. Fault Status and Fault Address Registers

Handler	Status Register Name	Address Register Name	Register Description
Hard fault	**Hard Fault Status (HFAULTSTAT)**	–	Hard Fault Status (HFAULTSTAT)
Memory management fault	**Memory Management Fault Status (MFAULTSTAT)**	**Memory Management Fault Address (MMADDR)**	Configurable Fault Status (FAULTSTAT) Memory Management Fault Address (MMADDR)
Bus fault	**Bus Fault Status (BFAULTSTAT)**	**Bus Fault Address (FAULTADDR)**	Configurable Fault Status (FAULTSTAT) Bus Fault Address (FAULTADDR)
Usage fault	**Usage Fault Status (UFAULTSTAT)**	–	Configurable Fault Status (FAULTSTAT)

5.2.4 Lockup

The processor enters a lockup state if a hard fault occurs when executing the hard fault handlers. When the processor is in the lockup state, it does not execute any instructions. The processor remains in lockup state until it is reset or it is halted by a debugger

5.2.5 PKA Interrupt

The PKA interrupt is the pka_int[1] signal from the PKA module. This signal is active whenever the PKA is idle (not busy). Because the PKA is idle after reset, this signal is active soon after reset. The interrupt can be disabled through the corresponding bit in a CM3 interrupt disable register, and is disabled at power up. PKA interrupt programming should consider that the CM3 allows an active but disabled interrupt to pend. That is, an pended interrupt can occur as soon as the PKA interrupt is enabled, even when the PKA is busy. The PKA ISR can test bit 15 of the **PKA_FUNCTION** register to confirm that the PKA interrupt is actually active.

JTAG Interface

This chapter describes the cJTAG and JTAG interface for on-chip debug support.

Topic **Page**

6.1 Debug Port

The debug interface supports both standard 4-pin JTAG as well as 2-pin cJTAG. Figure 6-1 shows the top level debug component connectivity through JTAG.

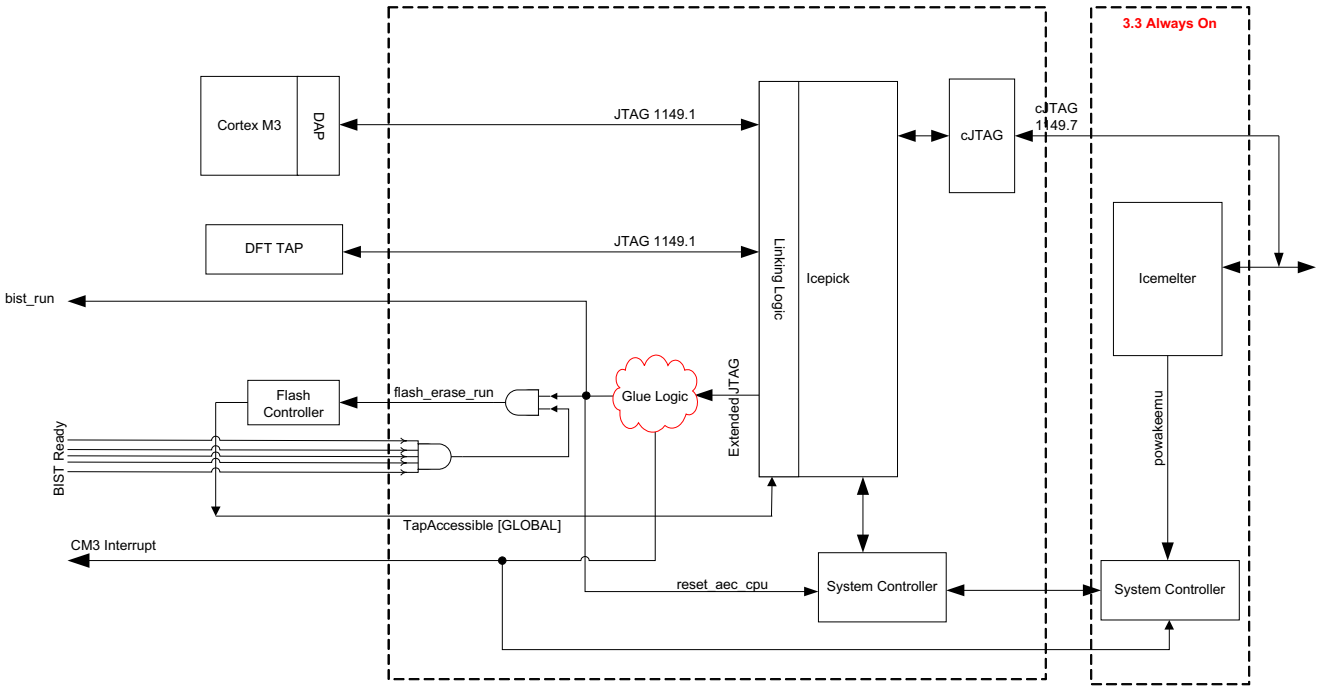


Figure 6-1. Test/Debug System Top Level Diagram

6.2 IEEE 1149.7

The default configuration after power up is 2 pin cJTAG (IEEE 1149.7) mode. An IEEE 1149.7 adapter module runs a 2-pin communication protocol on top of an IEEE 1149.1 JTAG test access port (TAP). The debug-IP logic serializes the IEEE 1149.1 transactions, using a variety of compression formats, to reduce the number of pins needed to implement a JTAG debug port. The device implements only a subset of the IEEE 1149.7 protocol (see Table 6-2).

The IEEE 1149.7 communication protocol can switch between serial and parallel formats. The parallel format consists of the IEEE 1149.1 signals, TCK, TMS, TDI, and TDO. The serial format uses two signals, TCK and TMSC. The values of the TMS, TDI, TDO, and RTCK signals are multiplexed on the bidirectional pin, jtag_tms_tmsc, by the cJTAG logic. The TDO information is returned during the packet transmission.

To switch the target debug interface to standard 4 pin JTAG, see section 6.1.2 for details.

NOTE: Only Oscan0 and Oscan4 formats can be used if the device requires stalls.

Table 6-1 describes the IEEE 1149.7 signals.

Table 6-1. IEEE 1149.7 Signals

Pin Name	Internal Signal Name	Type(1)	Pull Type (2)	Function	Description
jtag_tck	TCK	I	PD	Test clock	This is the test clock used to drive an IEEE 1149.1 TAP state-machine and logic. This is a free-running clock.
jtag_tms_tmsc	TMSC	I/O	PU	Test mode control and data scan	Compressed JTAG packet. The TMSC signal is driven from the chip only when the TCK signal is low. Logic inside the chip maintains the signal level while the TCK signal is high.

(1) I = Input; O = Output

(2) PU = internal pull up; PD = internal pull down

Table 6-2 summarizes the IEEE 1149.7 features subset supported by the device.

Table 6-2. IEEE 1149.7 Features Subset

	IEEE 1149.7 Feature	Device Support	Comment
Configuration	Class 4 TAP	✓	Supports 2-pin operation
	Class 5 TAP	–	No BDX or CDX channel
Optional components	FRST	–	Functional reset (done via ICEPick™)
	TRST	–	Test reset
	RDBK capability	–	Read back of register data
	Aux pin functions	–	Reuse of TDI and TDO pins
	TCKWID	–	Programmable TCK width
Scan formats	JScan0	✓	Parallel mode
	JScan1	✓	Parallel with firewall
	JScan2	✓	Parallel with super-bypass select
	JScan3	✓	Parallel with register select
	SScan0	–	Segmented scan
	SScan1	–	Segmented scan + stalls
	SScan2	–	Segmented scan
	SScan3	–	Segmented scan + stalls
	Oscan0	✓	Support stalls
	Oscan1	✓	Nonstall mode
	Oscan2	✓	Bidirectional transfers
	Oscan3	✓	Host to target only
	Oscan4	✓	Support stalls
	Oscan5	✓	Pipelined
	Oscan6	✓	Bidirectional transfers
	Oscan7	✓	Host to target only
	Mscan	–	Multi devices mode + stalls
Power control	Power down logic	–	Handled by ICEMelter

6.3 Switching Debug Interface from 2-pin cJTAG to 4-pin JTAG

Besides the default cJTAG mode of operation, the target debug interface can also be switched to standard JTAG (IEEE 1149.1) mode of operation. Switching requires three steps, opening the command window, storing format JScan0 and closing command window.

Before the cJTAG module accepts any commands, the control level must be set to 2 and locked. The following steps describe the procedure in detail:

Step 1. Opening Command Window

(a) Scan_IR (bypass, end in Pause DR) - load benign opcode into the Instruction Register.

(b) Goto_Scan(via Update_DR, end in Pause DR) - this is the first ZBS.

(c) Goto_Scan(via Update_DR, end in Pause DR) - this is the second ZBS.

(d) Scan_DR(1 bit, end in Pause DR) - This locks the control level at 2.

Opening the command window decouples the device TAP. Then, until the command window closes, all cJTAG message traffic occurs between the DTS controller and the cj1149_7 TS module but not the reset of the device.

The TMS sequence from Run-Test-Idle is (1 1 0 0*n 1 1 1 0 1 0) for the IR shift to end in Pause DR.

The TMS sequence from Pause DR to Pause DR is (1 1 1 0 1 0) (Done twice).

The TMS sequence from Pause DR through shift to Pause DR is (1 0 0 1 0).

Step 2. Store Scan Format
Once the command window is open, commands can be issued. The opcode for STFMT is 3. It assumes the TAP state is starting from Pause DR. For a different scan format, such as Oscan1, change CP2 to 9 (Oscans are 8 + the number).

(a) Scan_DR(3 bits of 1, end in Pause DR) - load CP1 with 3.

(b) Goto_Scan(via Update_DR to Pause DR) - Complete CP1 by going through update.

(c) Scan_DR(16 bits of 1, end in Pause DR) - load CP2 with 16.

(d) Goto_Scan(via Update_DR to Pause DR) - Complete CP2 by going through update.

Assuming PauseDR starting state, the TMS Sequence is (10001110111010). This does a 3-bit shift for CP0 (STFMT) and a 0 bit shift for CP1 (JScan0).

Step 3. Close Command Window
The command window can be closed by doing an IR scan, going to Test Logic Reset or by an ECL command. The ECL command is a sub-command of the STMC (opcode 0) command. Again we assuming the TAP state is starting from Pause DR.

(a) Goto_Scan(via Update_DR to Pause DR) - Does a Zero Bit scan to load CP1 with 0.

(b) Scan_DR(1 bit of 1, end in Pause DR) - load CP2 with 1

(c) Goto_Scan(via Update_DR to Pause DR) - Complete CP2 by going through update.

Now that the command window is closed, the device TAP is usually coupled, so any subsequenct scans (IR or DR) are issued to the device TAP, not the cj1149_7 TS module. The TMS sequence is (1 1 1 0 1 0) and (1 0 1 1 1 0 1 0).

NOTE: When switching from 2-pin to 4-pin mode, the hardware will configure PB6 as TDI and PB7 as TDO. No additional steps are required by the user to have TDI and TDO mapped as GPIO. Previous configurations of PB6 and PB7 will be overridden.

6.4 Debugger Connection

6.4.1 ICEPick Module

The debugger connects to the device through its JTAG interface. The first level of debug interface seen by the debugger is the IEEE 1149.7 adapter connected to the ICEPick module embedded in the debug subsystem.

NOTE: ICEPick version C (ICEPick-C) is used in the device.

Complex system designs typically have multiple processors, each having a JTAG TAP embedded in the processor. The ICEPick module manages these TAPs and the power, reset, and clock controls for modules that have TAPs.

The ICEPick module is visible only from the debugger point of view, and thus cannot be programmed by application software. The debugger can configure ICEPick through its own TAP controller. The ICEPick TAP has an instruction length of 6 bits and is the primary TAP. It is used to control and monitor the other secondary TAPs.

ICEPick provides the following debug capabilities:

- Debug connect logic for enabling or disabling most ICEPick instructions
- Dynamic TAP insertion
  - Serially linking up to 16 TAP controllers
  - Individually selecting one or more of the TAPs for scan without disrupting the instruction register (IR) state of other TAPs
- Power, reset, and clock management
  - Provides the power and clock states of each emulation domain
  - Provides debugger control of the power, clock, and reset control of the processor. The processor can force on the processor domain power and clocks, and prohibit the domain from being clock-gated or powered down while a debugger is connected.
  - Applies system warm reset
  - Provides local warm reset of the processor
  - Provides global and local warm reset blocking

The ICEPick module implements a connect register, which must be configured with a predefined key to enable the full set of JTAG instructions. When the debug connect key is properly programmed, ICEPick signals and subsystems emulation logics should be turned on. ICEPick ID code is 0x8B96402F.

6.4.1.1 Default Boot Mode

In ICEPick-only configuration, none of the secondary TAPs are selected. The ICEPick TAP is the only TAP between device-level TDI and TDO pins and debug boot mode is ICEPick only mode.

6.4.1.2 CM3 DAP Access

On the device, CM3 is the only debug TAP connected to the ICEPick. The CM3 DAP can be accessed using ICEPick with secondary debug TAP 0 ID.

6.5 Primary Debug Support

6.5.1 Processor Native Debug Support

6.5.1.1 Cortex™-M3 MPU

The Cortex-M3 processor supports the following native debug features:

- Program halt and stepping
- Hardware breakpoints, breakpoint instruction
- Data watch point on access to data add, add range, and data value
- Register value accesses
- Debug monitor exception
- Memories accesses

For more information about Cortex-M3 native debug support features, see the *Cortex-M3 Technical Reference Manual*.

6.5.2 Suspend

The device supports a suspend feature, which provides a way to stop a "closely coupled" hardware process running on a peripheral-IP when the host processor enters debug state. The suspend mechanism is important for debug to ensure that peripheral-IPs operate in a lock-step manner with a host controller processor.

6.6 Debug Access Security Through ICEPick

The top most page the Flash contains the lock bit information. A value of 1 enables the access to all the secondary/slave DAP/TAP connected to the ICEPick while 0 disables all the access.

After having locked debug access, a mass erase of the device, followed by an external reset, must be performed to regain control of the chip.

6.6.1 Unlocking the Debug Interface

Step 1: Initiate flash mass erase

- Scan “Public Connect Sequence (with 0x07 IR followed by 0x89 DR)” for ICEPick. Refer to ICEPick functional spec for details on “Public Connect Sequence”.
- Scan “Private Connect Sequence (with 0x1F IR followed by 0x01 DR)” for ICEPick.
- Do IR scan 001101 (0x0D) followed by IR scan 001110 (0x0E) for ICEPick.
- Monitor the status register for 0x0E IR. Look for bits [0:4] & [6] being high and bit [5] being low. When this condition is true, the debug unlock sequence is complete.
- After the status mentioned above is achieved, issue an ICEPick instruction 111110 (0x3E) to clear the debug unlock command.
- Any other sequence of the IRs for ICEPick, will be ignored by the glue logic and flash erase will not be executed.

Step 2: Perform the external reset

- The debug interface is not unlocked unless an external reset is performed.

Following 0x0E IR, the data register read (DR) reflects status of various activities related to the flash mass erase command. Table 6-4 shows the description.

Table 6-3. Data Register description for instruction 0x0E(Read Only)

Bits	Name	Type	Description
31:7	NA		Not Used
6	flash_erase_run	R	This reflects the command issued to the flash controller for the mass erase 1 - Command issued 0 Command is not issued
5	flash_erase_busy	R	1 - Mass erase is in progress This bit only takes effect after bit 6 is high. After bit 5 goes high, wait for 10 TCK before checking this bit.
0:4	reserved	R	reserved

6.7 CM3 Debug Interrupt

CM3 debug interrupt generation is implemented, using an extended JTAG interface on ICEPick. Instruction 0x0A of ICEPick with 8-bit data register is implemented to control the CM3 debug interrupt. The LSB of the data register drives the debug interrupt.

Following are the steps assert/de-asset the debug interrupt (an active high signal):

- Scan “Public Connect Sequence (with 0x07 IR followed by 0x89 DR)” for ICEPick. Refer to ICEPick functional spec for details on “Public Connect Sequence”.
- Scan “Private Connect Sequence (with 0x1F IR followed by 0x01 DR)” for ICEPick.
- Do IR scan of 0x0A followed by 32-bit DR scan to set the LSB to 1 for ICEPick. This will set the interrupt to CM3 high. A DR scan of 8-bit with LSB 0 will de-assert the interrupt.

Table 6-4. Data Register Description for 0x0A

Bits	Name	Description
31:1	NA	Not Used
0	CM3_debug_interrupt	1 - Assert an interrupt to CM3 (pulling the signal high) 0 - De-assert interrupt to CM3 (pulling the signal down)

System Control

The system control module configures the overall operation of the CC2538 device. Configurable features include reset control, power control, clock control, and low-power modes. Low-power operation is enabled through different operating modes (power modes). Ultralow-power operation is obtained by turning off the power supply to modules to avoid static (leakage) power consumption and also by using clock gating and turning off oscillators to reduce dynamic power consumption.

Topic **Page**

7.1 Power Management

Power management enables optimized power consumption for an application. Power management is based on three levels of power saving actions:

- Clock gating of unused / not required peripheral clocks
- Power down of clock sources
- Power down the power supply

Clock gating is the simplest power saving action and also the fastest to enter and exit. Powering down the supply is the most effective power saving action, but a power down / up sequence is more time demanding than clock gating or power down of clock sources.

This section describes how to manage the different power saving actions.

7.1.1 Control Inputs to Power Management

CC2538 has different configuration registers and an initiator (WFI instruction) for control of operational (i.e power) modes.

7.1.1.1 Clock Gating Registers

The description of the clock gating registers for each peripheral are found in the register section (see). There are three different types of registers:

- The **RCGCXX** register controls clocks in Active mode (Run mode)
- The **SCGCXX** register controls clocks in Sleep mode
- The **DCGSXX** register controls clocks in PM0 (Deep Sleep mode)

These register settings are don't care for Power Modes 1, 2 and 3 since the system clock source in these modes is powered down.

7.1.1.2 Deep Sleep Selector (SYSCTRL Register)

SLEEPDEEP is a bit in **SYSCTRL** register described in System Control (SYSCTRL). When the **SLEEPDEEP** bit is set the chip will enter power modes in accordance to **PMCTL** register setting when the operational mode initiator (WFI instruction) is asserted.

7.1.1.3 Power Mode Configuration Register (PMCTL Register)

PMCTL is a two bit register that configures which power mode (PM0, 1, 2 and 3) the chip shall enter when **SLEEPDEEP** is set and operational mode is initiated (using the WFI instruction).

7.1.1.4 WFI - Operational Mode Initiator (Wait For Interrupt)

The register settings described in the above sections will not take effect until WFI is asserted. WFI is a special assembly instruction to the CPU which enables the CPU to enter sleep mode. A flow diagram for an asserted WFI is shown in Figure 7-1. All operational modes other than Active (Run) mode are exited by an enabled interrupt. See Section 7.1.2.4 and Chapter 5 for description of interrupt handling. Also, while WFI is active, halting the CPU will exit the device from WFI. When run is initiated afterwards, CPU will commence at the immediate instruction following WFI.

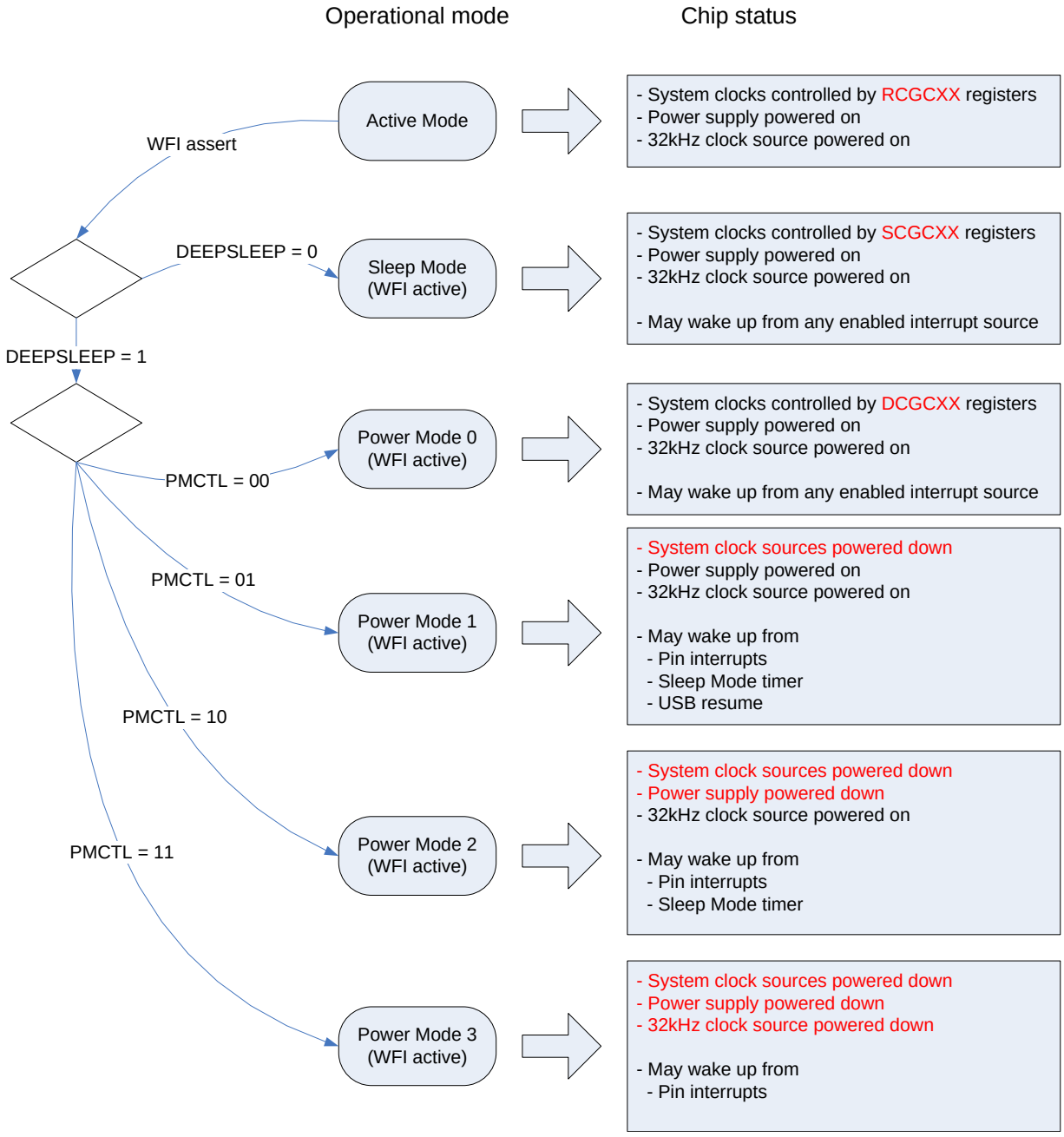


Figure 7-1. Flow Diagram for Operational Modes

7.1.2 Using Power Management

When using power management in an application required functionality, sequencing time and power saving must be considered. Table 7-1 shows a matrix of power saving, sequencing time and functional limitations.

Table 7-1. Power Management Matrix

Operational Mode	Power Consumption	Sequencing time	Functional limitations
Active	Clock gating with RCGC	None	None
Sleep	Clock gating with SCGC	Enter: immediate	CPU in sleep

Table 7-1. Power Management Matrix (continued)

Operational Mode	Power Consumption	Sequencing time	Functional limitations
PM0	Clock gating with DCGC	Enter: immediate	CPU in Deep sleep
PM1	Power down of: • System clock source Refer to CC2538 Data Sheet for typical current consumption.	Enter: 0.5 us Exit: 4 us	CPU in Deep sleep All peripherals inactive
PM2	Power down of: • System Clock source • Digital Power supply Refer to CC2538 Data Sheet for typical current consumption.	Enter: 136 us Exit: 136 us	CPU in Deep sleep (inactive) All peripherals inactive
PM3	Power down of: • System Clock source • Digital Power supply • 32 kHz Clock source Refer to CC2538 Data Sheet for typical current consumption.	Enter: 136 us Exit: 136 us	CPU in Deep sleep (inactive) All peripherals inactive Sleep Mode Timer inactive

NOTE: The sequencing time in Table 7-1 assumes that the selected system clock source is 16 MHz RCOSC. It is also possible to run the power down sequence on 32 MHz XOSC, but timing will be dependent on user configurations and calibration times see Section 7.1.2.3.2.

7.1.2.1 Sequencing when using Power Modes

When using power management in CC2538 it is important to understand the sequence of events and timing involved in the process. A simple flow diagram for power management is shown in Figure 7-2. As can be seen from the figure PM1, 2 and 3 are always entered from a state where the CPU is running on 16 MHz RCOSC.

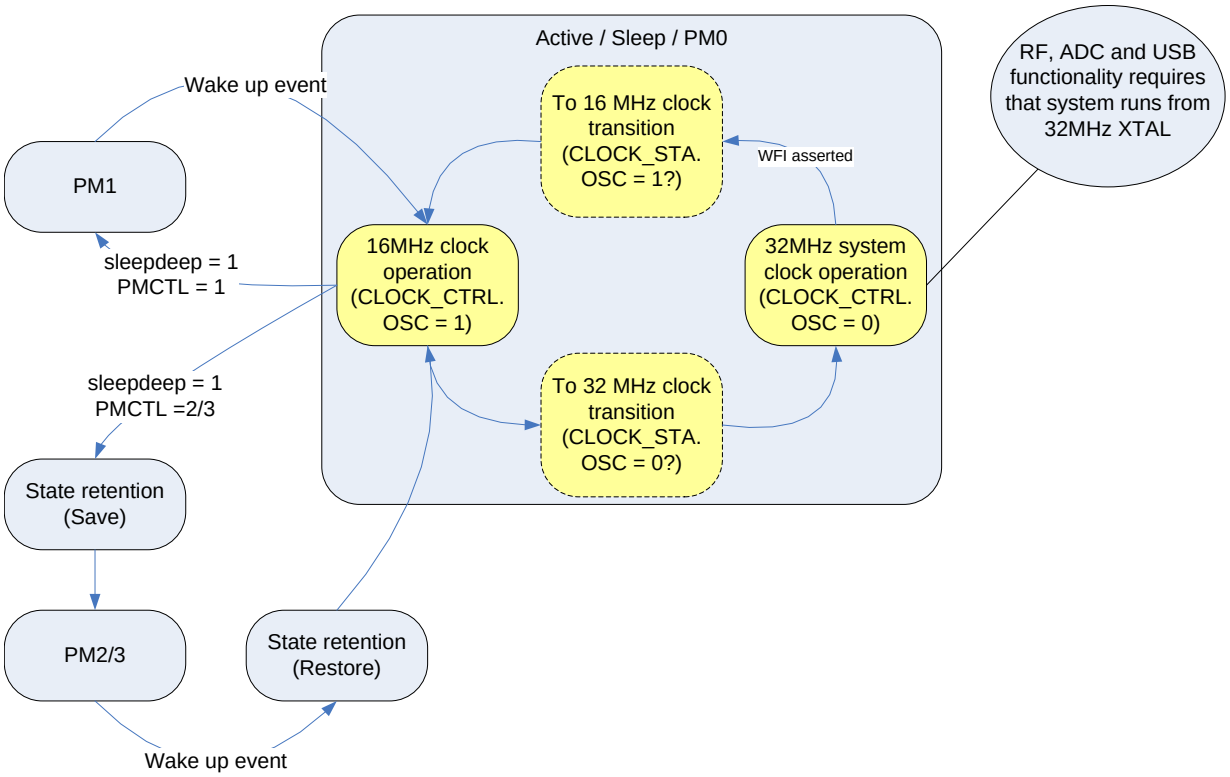


Figure 7-2. Simple Flow Diagram for Power Management

7.1.2.2 Time Considerations for Active, Sleep and PM0

As seen from Table 7-1 there is really no timing involved in transitions between Active (Run) mode, Sleep mode and PM0. The differences between these operating modes are limited to which of the configurable clock gating registers that is used:

- **RCDCXX** - Active
- **SCGCXX** - Sleep
- **DCGCXX** - PM0

NOTE: User configured clock gates may result in higher power consumption in Sleep / PM0 than in Active mode.

Transitions to Sleep and PM0 are initiated by asserting WFI.

- If system clock is running from 32 MHz XOSC when WFI is asserted, the chip will continue to run on 32 MHz XOSC after the wake up event with return to active mode.
- If system clock is running from 16 MHz RCOSC when WFI is asserted, the chip will continue to run on 16 MHz RCOSC after the wake up event with return to active mode.

7.1.2.3 Time Considerations for PM1, PM2 and PM3

When PM1, PM2 or PM3 is entered by asserting WFI the transition times are dependent of both user configurations and physical variations. WFI can be asserted in order to enter PM1, PM2 or PM3 at any point in time and the chip will eventually enter the requested power mode. However, in order to have good control over the timing it is important to understand the sequencing from Active mode to PM1, PM2 or PM3. As seen in Figure 7-2 the power down sequence to PM1, PM2 and PM3 from Active mode always

starts via 16 MHz system clock source. It is not required by the user to manually change the clock source from 32 MHz to 16 MHz before asserting WFI. If the chip is running on 32 MHz system clock assertion of WFI will trigger automatic controlled switching to 16 MHz clock if PM1, 2 or 3 is to be entered. In order to have good control over the timing it is recommended to enter PM1, 2 and 3 when the device is running on 16 MHz RCOSC, please see below.

7.1.2.3.1 Enter Power Mode when Running on 16 MHz System Clock

If the selected system clock is 16 MHz RCOSC when asserting WFI the start of the power down sequence is deterministic and immediate as shown in Figure 7-2. In order to guarantee that the chip is currently running on 16 MHz RCOSC when entering power modes follow this procedure:

- Configure power mode (**SLEEPDEEP** = 1 and **PMCTL** ≠ 00)
  - Does not need to be set here (can be set earlier) since the settings of SLEEPDEEP and **PMCTL** does not take effect before WFI is asserted.
- Read the **SOURCE_CHANGE** bit in **CLOCK_STA** to '0'
  - If it reads '1' a transition between system clocks is active. Wait until it reads '0'
- Read the **OSC** bit in **CLOCK_STA** to '1'.
  - This ensures that the system is running from the 16 MHz clock source.
- Assert WFI

7.1.2.3.2 Enter Power Mode when Running on 32 MHz System Clock

When running on 32 MHz system clock assertion of WFI will automatically start a transition to 16 MHz system clock before the power down sequence starts. During the time from WFI is asserted until the power down sequence have started (moved outside the Active mode in Figure 7-2) the system will run on settings selected by **DCGCXX** clock gate registers.

CAUTION

Any event on enabled interrupts in the transition period between 32 MHz and 16 MHz clock source will stop the chip from entering the selected power mode unless WFI is asserted again or not deasserted by the interrupt handler. The clock source will be reverted to the setting it had before, i.e. the chip will have the same status as before the WFI was asserted.

Figure 7-3 show an example of a sequence when CC2538 is in Active mode running on 32 MHz system clock, until the chip has entered PM1, PM2 or PM3.

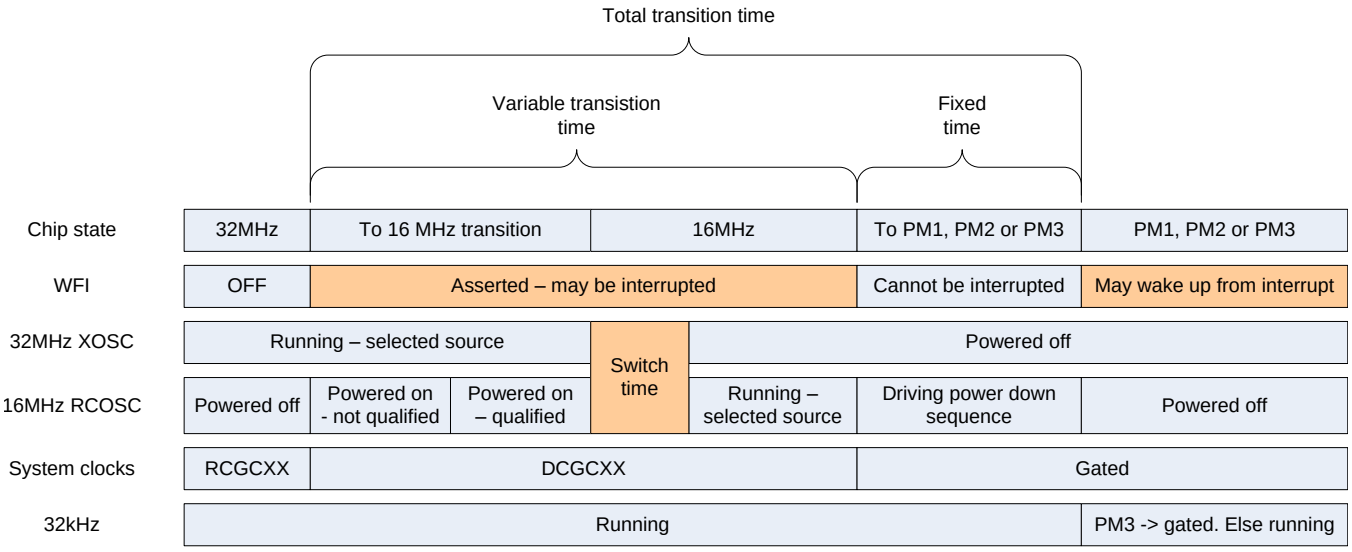


Figure 7-3. Timing Example for Transition from 32 MHz to PM's

As seen from Figure 7-3 the total transition time is made up from a fixed time component and a variable time component. The fixed time is independent of user configurations and is 0.5 us for PM1 and 30 us for PM2 and PM3. The variable time depend on:

- Startup and calibration of 16 MHz clock
- Calibration of 32 kHz clock
- User configuration, such as which system clock source is selected

7.1.2.3.2.1 16 MHz RCOSC Startup Time

The typical startup time of 16 MHz RCOSC is 10us. In order to remove the startup time from the total transition time **OSC_PD** bit in **CLOCK_CTRL** can be set to '0' minimum 10us before WFI is asserted. 16 MHz RCOSC will then be powered on and running when WFI is asserted and the switching between 32 MHz and 16 MHz can occur immediately.

7.1.2.3.2.2 16 MHz RCOSC Calibration Time

Calibration of 16 MHz RCOSC clock source is started immediately when 32 MHz XOSC is the selected clock source. The calibration takes approx 50us. Thus if WFI is asserted before the calibration of 16 MHz is finished after a switch to 32 MHz clock source is completed the remaining time of the calibration will add to the variable time shown in Figure 7-3. If 16 MHz calibration is ongoing while WFI is asserted the 16 MHz will not be powered down after the calibration is finished so the startup time does not come in addition. It is not possible to disable the 16 MHz calibration.

7.1.2.3.2.3 32 kHz RCOSC Calibration Time

Calibration of 32 kHz RCOSC will only occur when 32 kHz RCOSC is selected as the low frequency clock source by the **OSC32K** bit in **CLOCK_CTRL**. Calibration of 32k RCOSC can be disabled by **OSC32K_CALDIS** in **CLOCK_CTRL**. Calibration of 32 kHz RCOSC is started immediately when 32 MHz XOSC is the selected clock source. The calibration takes approx 2 ms. Thus if WFI is asserted before the calibration of 32 kHz RCOSC is finished after a switch to 32 MHz clock source is completed the remaining time of calibration will add to the variable time shown in Figure 7-3.

7.1.2.3.2.4 Clock Source Switch Time Before Power Down

After calibration is finished both clock sources are running and a switch from 32 MHz to 16 MHz can start and will occur immediately (~0.3us).

7.1.2.4 Exit from Power Modes

Exit from PM1, PM2 or PM3 occurs due to an event enabled for wake up. The chip can wake up on a event on any of the GPIO pins, USB and Sleep Timer. The interrupt event wake up is controlled using the **IWE** register and the specific registers for each peripheral. The chip state after wake up is the same state as the chip had when WFI was asserted and the chip entered power mode. Thus:

- If WFI was asserted when running on 16 MHz system clock the chip will run on 16 MHz after the wake up sequence is finished.
- If WFI was asserted when running on 32 MHz system clock the chip will start up on 16 MHz after the wake up event and automatically change to 32 MHz clock source as soon as the 32 MHz clock source is qualified by hardware.

If WFI was asserted on 16 MHz system clock source the wake up times are in accordance with Table 7-1. Similar to the transition time for the power down sequence there is a variable and fixed time for power up when WFI was asserted on 32 MHz system clock source. The variable time depend on:

- Startup and qualification of 32 MHz
- User configuration

CAUTION

These modules lose their state in PM2 and PM3;

- SRAM address space: 0x2000 0000 - 0x2000 3FFF
- AES and PKA
- I^2C
- General Purpose Timer 3
- USB
- RX and TX FIFOs

7.1.2.4.1 32 MHz XOSC Startup Time

The 32 MHz XOSC startup time is depending on the actual crystal being used. Please refer to the crystal manufacturer data sheet for details. If WFI was asserted when running from 32 MHz a wake up event will power up both 16 MHz and 32 MHz oscillators in parallel so startup of 32 MHz is ongoing while 16 MHz clock is driving the power up sequence. From PM1 this is of little importance, but when waking up from PM2 and PM3 the required time for restore that is driven by 16 MHz clock (approx 30us) will be saved.

7.1.2.4.2 32 MHz Qualification Time

In the CC2538 an additional module for detection of 32 MHz XOSC stability is available. Enabling this feature adds approx 20 us to the 32 MHz XOSC startup time, see 7.2.1 for details.

7.1.3 Power Consumption

The actual power consumption when using the different power modes depends on the time from WFI is asserted to a wake up event occurs. As shown in Table 7-1 the current consumption for Active, Sleep and PM0 is defined by clock gating configured by **R/S/DCGCXX** registers. For PM1 the system clock source is powered down immediately (when running on 16 MHz) and the power down current is constant from WFI is asserted until a wake up event occurs as shown in Figure 7-4.

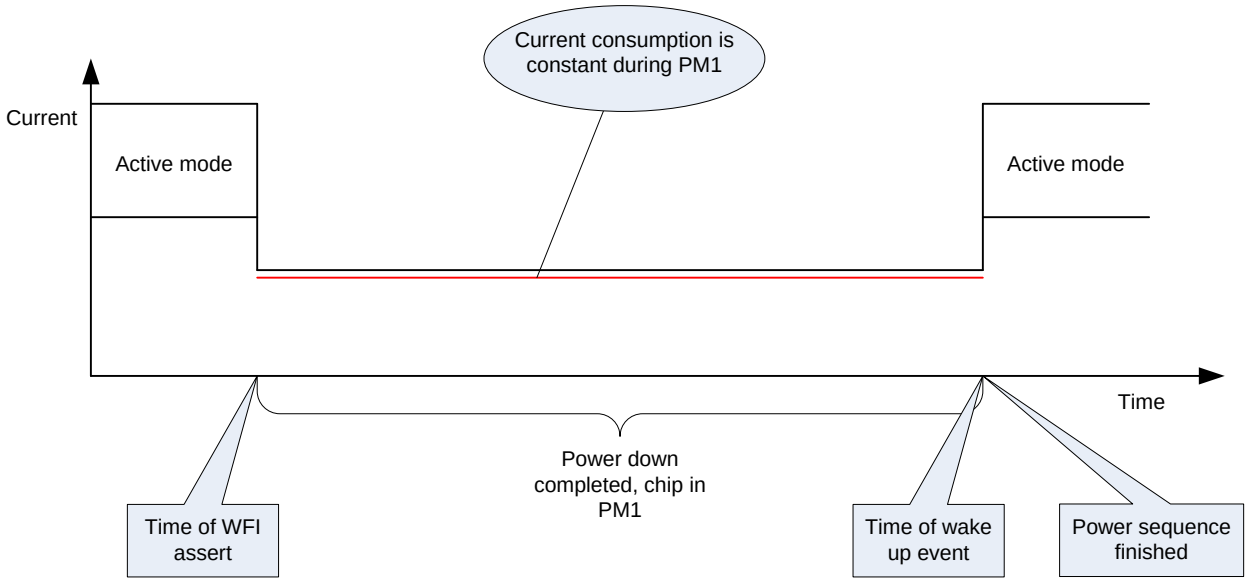


Figure 7-4. Simplified Figure of Current Consumption in PM1

For PM2 and PM3 the power consumption will be higher than for PM1 during power down sequence and power up sequence while it will be lower after the power down sequence has completed as shown in Figure 7-5. The power down sequence handles the chip state retention.

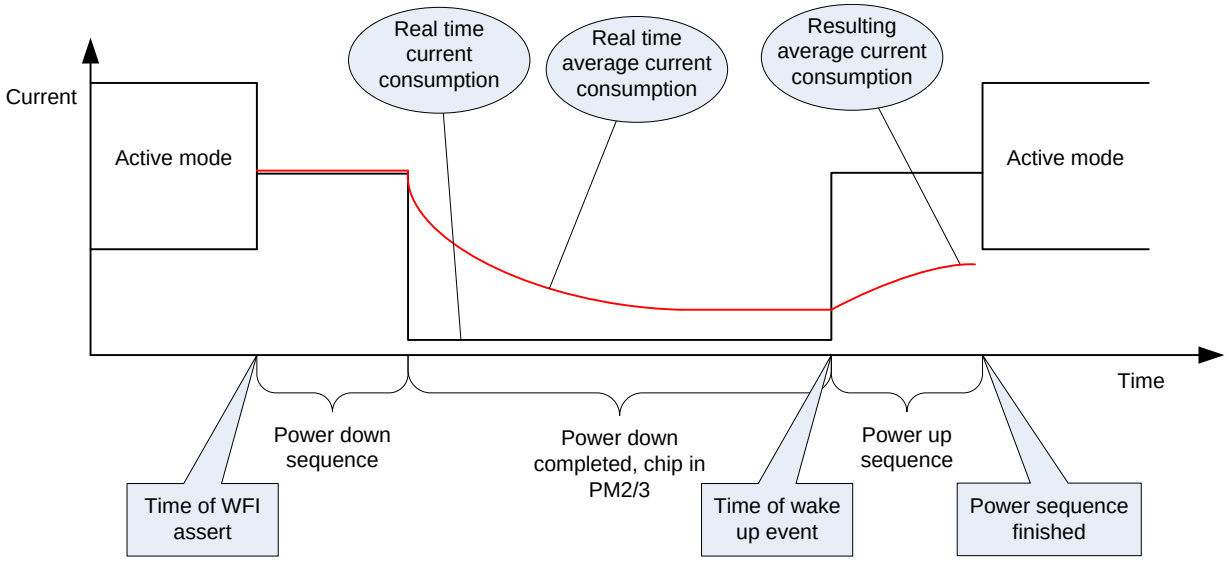


Figure 7-5. Simplified Figure of Current Consumption in PM2 and PM3

As seen from Figure 7-5 the average current consumption for PM2 and PM3 is time dependent. The average consumption from WFI assertion to power sequence is finished can be found from Table 7-1.

- For sleep times shorter than 3 ms the lowest possible power down mode is PM0. This is due to the Power-down guard time of the 32 MHz crystal oscillator.
- PM3 is the most effective power saving mode but can only wake up from pin interrupts (Sleep Mode Timer is disabled)

7.2 Oscillators and Clocks

The device has one internal system clock, or main clock. The source for the system clock can be either the 16 MHz RC oscillator or the 32 MHz crystal oscillator. There is also one 32 kHz clock source that can either be an RC oscillator or a crystal oscillator. Clock control is performed using the **CLOCK_CTRL** register.

The **CLOCK_STA** register is a read-only register used for getting the current clock status.

The choice of oscillator allows a trade-off between high accuracy in the case of the crystal oscillator and low power consumption when the RC oscillator is used. The 32 MHz crystal oscillator must be used for RF transceiver, ADC and USB operation.

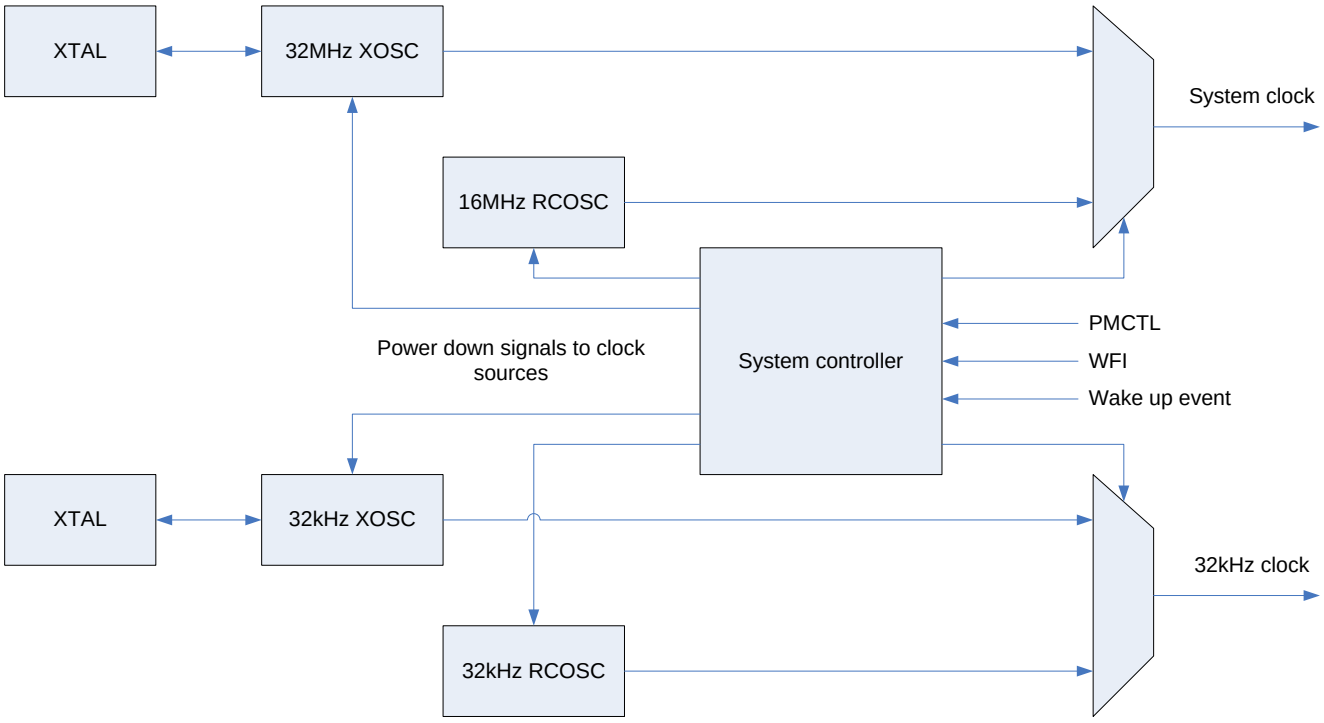


Figure 7-6. Block Diagram Oscillators and Clocks

7.2.1 High Frequency Oscillators

Figure 7-6 gives an overview of the clock system with available clock sources on the CC2538 device.

Two high-frequency oscillators are present in the device:

- 32 MHz crystal oscillator
- 16 MHz RC oscillator

The 32 MHz crystal oscillator start-up time may be too long for some applications; therefore, the device can run on the 16 MHz RC oscillator until the crystal oscillator is stable. The 16 MHz RC oscillator consumes less power than the crystal oscillator, but because it is not as accurate as the crystal oscillator it cannot be used for RF transceiver operation.

Some applications need to reduce the total time from a power mode until the chip is running on 32 MHz XOSC. When WFI is asserted while running on 32 MHz XOSC the 32 MHz XOSC will be automatically started after power mode. If WFI is asserted while running on the 16 MHz RC oscillator the **CLOCK_CTRL.OSC_PD** bit needs to be set to 0 in order to ensure that the 32 MHz XOSC is started as quick as possible after power mode. When these settings are used before the power down sequence, the XOSC will be powered up immediately when a wake up interrupt is present and thus the 32 MHz XOSC qualification will be done in parallel with the restore sequence.

NOTE: After coming up from PM1, PM2, or PM3, the CPU must wait for **CLOCK_STA.OSC** to be 0 before operations requiring the system to run on the 32 MHz XOSC (such as the radio) are started.

In the CC2538, an additional module for detection of 32 MHz XOSC stability is available. This amplitude detector can be useful in environments with significant noise on the power supply to ensure that the clock source is not used until the clock signal is stable. This module can be enabled by setting the **CLOCK_CTRL.AMP_DET** bit, this adds around 20us to the 32 MHz XOSC startup time.

7.2.1.1 16 MHz RCOSC Calibration

The 16 MHz RC oscillator is calibrated once after the 32 MHz XOSC has been selected and is stable (i.e., when the **CLOCK_CTRL.OSC** bit switches from '1' to '0'). Please see CC2538 Data Sheet for typical calibration time. It is not possible to disable the 16 MHz RCOSC calibration.

7.2.2 Low Frequency Oscillators

As can be seen from Figure 7-6 two low-frequency oscillators are present in the device:

- 32 kHz crystal oscillator
- 32 kHz RC oscillator

The 32 kHz XOSC is designed to operate at 32.768 kHz and provide a stable clock signal for systems requiring time accuracy. The 32 kHz RCOSC must be used to reduce cost and power consumption compared to the 32 kHz XOSC solution. The two 32 kHz oscillators cannot be operated simultaneously.

By default, after a reset, the 32 kHz RCOSC is enabled and selected as the 32 kHz clock source. The RCOSC consumes less power, but is less accurate compared to the 32 kHz XOSC. The chosen 32 kHz clock source drives the sleep timer, generates the tick for the watchdog timer, and is used as a strobe in MAC timer to calculate the sleep timer sleep time. The **CLOCK_CTRL.OSC32K** register bit selects the oscillator to be used as the 32 kHz clock source.

After having switched to the 32 kHz XOSC and when coming up from PM3 with the 32 kHz XOSC enabled, the oscillator requires up to 400 ms to stabilize on the correct frequency. The sleep timer, watchdog timer, and clock-loss detector must not be used before the 32 kHz XOSC is stable.

7.2.2.1 32 kHz RCOSC Calibration

The **CLOCK_CTRL.OSC32K** register bit can be written at any time, but does not take effect before the 16 MHz RCOSC is the active system clock source. When system clock is changed from the 16 MHz RCOSC to the 32 MHz XOSC (**CLOCL_CTRL.OSC** from 1 to 0), calibration of the 32 kHz RCOSC starts up and is performed once if the 32 kHz RCOSC is selected. During calibration, a divided version of the 32 MHz XOSC is used. The result of the calibration is that the 32 kHz RSOSC is running at 32.753 kHz. The 32 kHz RCOSC calibration may take up to 2 ms to complete. Calibration can be disabled by setting the **CLOCK_CTRL.OSC32K_CALDIS** bit to 1. At the end of the calibration, an extra pulse may occur on the 32 kHz clock source, which causes the sleep timer to be incremented by 1.

7.3 System Clock Gating

To decrease the total power consumption of the device system clocks to different peripherals can be gated. The clock gate controller handles the system clock gates as shown in Table 7-2.

SYS_DIV is the frequency setting for the system clock and all module core clocks not supported by **IO_DIV**. **IO_DIV** is the frequency setting for baud rate clocks for the SSI and UART modules. There is only one **IO_DIV** in the system, which is used by all SSI and UART instances.

The **RCGCxx**, **SCGCxx**, and **DCGCxx** registers are used to gate clocks (i.e., enable or disable) for the different peripherals in the respective operating mode. The **RCGCxx** registers configure the different peripheral clocks in active mode. The **SCGCxx** registers configure the different peripheral clocks in sleep mode. The **DCGCxx** registers configure the different peripheral clocks in PM0. Disabling (i.e., gating the clocks) peripherals that are not used will reduce the overall power consumption.

Table 7-2. Clock Gate Matrix

Chip State	Clock Gate Register	System Clock PCLK	Peripheral Core Clocks	UART/SSI System Clock		UART/SSI Core Clock	
Active Mode	RCGCxx = 0 (SCGCxx and DCGCxx = NA)	Running – SYS_DIV freq	gated – not running	gated – not running		gated – not running	
	RCGCxx = 1	Running – SYS_DIV freq	Running – SYS_DIV freq	Running – SYS_DIV freq		Running – CS PIOSC	
						0	1
						SYS_DIV	IO_DIV
Sleep Mode	SCGCxx = 0 (RCGCxx and DCGCxx = NA)	Running – SYS_DIV freq	gated – not running	gated – not running		gated – not running	
	SCGCxx = 1	Running – SYS_DIV freq	Running – SYS_DIV freq	Running – SYS_DIV freq		Running – CS PIOSC	
						0	1
						SYS_DIV	IO_DIV
PM0	DCGCxx = 0 (RCGCxx and SCGCxx = NA)	Running – SYS_DIV freq	gated – not running	gated – not running		gated – not running	
	DCGCxx = 1	Running – SYS_DIV freq	Running – SYS_DIV freq	Running – CS DSEN		Running – CS PIOSC	
				0	1	0	1
				SYS_DIV	Same as CCLK	SYS_DIV	IO_DIV
PM1/2/3	gated – not running	gated – not running	gated – not running	gated – not running		gated – not running	

7.3.1 Clocks for UART and SSI

The UARTs and SSIs are supplied with system clock and a programmable core clock. The UART core clock must not exceed 5/3 of system clock. It is SW responsibility to ensure a correct system clock – core clock ratio.

During PM0 the UART and SSI modules may be operational requiring a higher clock frequency than the system clock. During PM0 the UART/SSI system and core clocks may therefore both be supported with a clock with the IO_DIV frequency as listed in Table 7-2.

The clocks for the UART and SSI peripherals are configured using the **RCGCSSI/SCGCSSI/DCGCSSI** and **RCGCUART/SCGCUART/DCGCUART** registers, respectively.

7.3.2 Clocks for GPT, I^2C, and SEC

The clocks for the general purpose timers, I^2C, and security peripherals are configured using the **RCGCGPT/SCGCGPT/DCGCGPT**, **RCGCI2C/SCGCI2C/DCGCI2C**, and **RCGCSEC/SCGCSEC/DCGCSEC** registers, respectively.

7.4 Reset

The device has six reset sources. The following events generate a reset:

- Forcing the RESET_N input pin low
- A POR condition
- A brownout reset (BOD) condition
- Watchdog timer reset condition

- Clock-loss reset condition
- ICEPICK warm reset

The initial conditions after a reset are as follows:

- I/O pins are configured as inputs with pull-up.
- CPU program counter is loaded with 0x0000 0000 and program execution starts at this address.
- All peripheral registers are initialized to their reset values, see register descriptions for reset values.
- Watchdog Timer is disabled.
- Clock-loss detector is disabled.

During reset, the GPIO pins are configured as inputs with pull-up.

In the CC2538, a system reset can be generated immediately in software by writing the **NVIC_APINT_SYSRESETREQ** bit to '1' in the **NVIC_APINT** register (see Application Interrupt and Reset Control (APINT) for the register description). This performs a software reset of the entire device. The processor and all peripherals are reset and all device registers are returned to their default values (with the exception of the reset cause register, **CLOCK_STA.RST**).

7.4.1 Reset of Peripherals

The individual peripherals can be reset using separate registers:

- General Purpose Timers 0, 1, 2 and 3 are controlled using **SRGCP** register.
- SSI[1:0] are controlled using **SRSSI** register.
- UART[1:0] are controlled using **SRUART** register.
- I2C is controlled using **SRI2C** register.
- Security module (AES and PKA) are controlled using **SRSEC** register.

7.4.2 Power-On Reset and Brownout Detector

The device includes a POR, providing correct initialization during device power on. It also includes a BOD operating on the regulated 1.8-V digital power supply only. The BOD protects the memory contents during supply voltage variations which cause the regulated 1.8-V power to drop below the minimum level required by digital logic, flash memory, and SRAM.

When power is initially applied, the POR and BOD hold the device in the reset state until the supply voltage rises above the POR and BOD voltages.

If the supply voltage is lowered to below 1.4 V during PM2/PM3, at temperatures of 70°C or higher, and then brought back up to good operating voltage before active mode is re-entered, registers and RAM contents that are saved in PM2/PM3 may become altered. Hence, care should be taken in the design of the system power supply to ensure that this does not occur. The voltage can be periodically supervised accurately by entering active mode, as a BOD reset is triggered if the supply voltage is below approximately 1.7 V.

The cause of the last reset can be read from the **CLOCK_STA.RST** register bits. A BOD reset is read as a POR reset.

7.4.3 Clock Loss Detector

The clock-loss detector can be used in safety-critical systems to detect that one of the clock sources (16/32 MHz system or 32 kHz) has stopped. When the clock-loss detector is enabled, the two clocks monitor each other continuously. If one of the clocks stops toggling, a clock-loss detector reset is generated within a certain maximum time-out period. The time-out depends on which clock stops. If the 32 kHz clock stops, the time-out period is 0.5 ms. If the 32 MHz clock stops, the time-out period is 0.25 ms. If the 16 MHz clock stops, the time-out period is 0.50 ms. When the system comes up again from reset, software can detect the cause of the reset by reading **CLOCK_STA.RST**. The clock-loss detector is enabled/disabled with the **CLD.EN** bit. Clock loss detection does not have to be disabled when used during power modes as the feature is automatic turned on/off in accordance to the current chip mode.

7.5 Emulator in Power Modes

In the case where an emulator is attached and the device tries to enter PM1/2/3 and EMUOVR = 0xFF, the device will instead enter PM0. The debug logic will override the internal sleep mode configuration, such that the system clock and power supplies remain on, thus leaving the emulator connected. The logic of the application code can now be debugged, although the actual current consumption will be slightly higher as the device is now in PM0 as opposed to PM1/2/3.

In the case where an emulator is attached and the device tries to enter PM0 or Sleep, the device will behave as if no emulator was attached and thus entering PM0 or Sleep respectively. Since clocks and power remains on in these two modes, no internal override logic is required.

7.6 Chip State Retention

In power modes PM2 and PM3, power is removed from most of the internal circuitry. A chip state retention feature ensures that most of the chip state is retained during a power mode cycle. RF configuration registers and MAC timer registers are also retained. Most of the chip state is included in this feature with the following exceptions:

- SRAM, address space: 0x2000 0000 - 0x2000 3FFF
- AES
- PKA
- I^2C
- USB
- General Purpose Timer 3
- RX and TX FIFOs

State retention is automatic and does not require enabling from user.

7.6.1 CRC Check on State Retention

During entry/exit to/from PM2 and PM3 the device state is stored during power down and read back during power up are CRC checked. A feature to automatically reset the CC2538 in the case of a CRC error can be enabled by setting bit **CRC_REN_RF** and **CRC_REN_USB** to '1' in the **SRCRC** register.

NOTE: Both **CRC_REN_USB** and **CRC_REN_RF** must be set in order to enable Reset on CRC failure.

7.7 System Control Registers

7.7.1 SYS_CTRL Registers

7.7.1.1 SYS_CTRL Registers Mapping Summary

This section provides information on the SYS_CTRL module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 7-3. SYS_CTRL Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
SYS_CTRL_CLOCK_CTRL	RW	32	0x0103 0109	0x00	0x400D 2000
SYS_CTRL_CLOCK_STA	RO	32	0x0101 0109	0x04	0x400D 2004

Table 7-3. SYS_CTRL Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
SYS_CTRL_RCGCGPT	RW	32	0x0000 0000	0x08	0x400D 2008
SYS_CTRL_SCGCGPT	RW	32	0x0000 0000	0x0C	0x400D 200C
SYS_CTRL_DCGCGPT	RW	32	0x0000 0000	0x10	0x400D 2010
SYS_CTRL_SRGPT	RW	32	0x0000 0000	0x14	0x400D 2014
SYS_CTRL_RCGCSSI	RW	32	0x0000 0000	0x18	0x400D 2018
SYS_CTRL_SCGCSSI	RW	32	0x0000 0000	0x1C	0x400D 201C
SYS_CTRL_DCGCSSI	RW	32	0x0000 0000	0x20	0x400D 2020
SYS_CTRL_SRSSI	RW	32	0x0000 0000	0x24	0x400D 2024
SYS_CTRL_RCGCUART	RW	32	0x0000 0000	0x28	0x400D 2028
SYS_CTRL_SCGCUART	RW	32	0x0000 0000	0x2C	0x400D 202C
SYS_CTRL_DCGCUART	RW	32	0x0000 0000	0x30	0x400D 2030
SYS_CTRL_SRUART	RW	32	0x0000 0000	0x34	0x400D 2034
SYS_CTRL_RCGCI2C	RW	32	0x0000 0000	0x38	0x400D 2038
SYS_CTRL_SCGCI2C	RW	32	0x0000 0000	0x3C	0x400D 203C
SYS_CTRL_DCGCI2C	RW	32	0x0000 0000	0x40	0x400D 2040
SYS_CTRL_SRI2C	RW	32	0x0000 0000	0x44	0x400D 2044
SYS_CTRL_RCGCSEC	RW	32	0x0000 0000	0x48	0x400D 2048
SYS_CTRL_SCGCSEC	RW	32	0x0000 0000	0x4C	0x400D 204C
SYS_CTRL_DCGCSEC	RW	32	0x0000 0000	0x50	0x400D 2050
SYS_CTRL_SRSEC	RW	32	0x0000 0000	0x54	0x400D 2054
SYS_CTRL_PMCTL	RW	32	0x0000 0000	0x58	0x400D 2058
SYS_CTRL_SRCRC	RW	32	0x0000 0000	0x5C	0x400D 205C
SYS_CTRL_PWRDBG	RW	32	0x0000 0004	0x74	0x400D 2074
SYS_CTRL_CLD	RW	32	0x0000 0100	0x80	0x400D 2080
SYS_CTRL_IWE	RW	32	0x0000 003F	0x94	0x400D 2094
SYS_CTRL_I_MAP	RW	32	0x0000 0000	0x98	0x400D 2098
SYS_CTRL_RCGCRFC	RW	32	0x0000 0000	0xA8	0x400D 20A8

Table 7-3. SYS_CTRL Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
SYS_CTRL_SCGCRFC	RW	32	0x0000 0000	0xAC	0x400D 20AC
SYS_CTRL_DCGCRFC	RW	32	0x0000 0000	0xB0	0x400D 20B0
SYS_CTRL_EMUOVR	RW	32	0x0000 00FF	0xB4	0x400D 20B4

7.7.1.2 SYS_CTRL Register Descriptions

SYS_CTRL_CLOCK_CTRL

Address offset	0x00		
Physical Address	0x400D 2000	**Instance**	SYS_CTRL
Description	The clock control register handels clock settings in the CC2538. The settings in CLOCK_CTRL do not always reflect the current chip status which is found in CLOCK_STA register.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED						OSC32K_CALDIS	OSC32K	RESERVED		AMP_DET	RESERVED			OSC_PD	OSC	RESERVED					IO_DIV			RESERVED		RESERVED		RESERVED			SYS_DIV

Bits	Field Name	Description	Type	Reset
31:26	RESERVED	This bit field is reserved.	RO	0x00
25	OSC32K_CALDIS	Disable calibration 32-kHz RC oscillator. 0: Enable calibration 1: Disable calibration	RW	0
24	OSC32K	32-kHz clock oscillator selection 0: 32-kHz crystal oscillator 1: 32-kHz RC oscillator	RW	1
23:22	RESERVED	This bit field is reserved.	RO	0x0
21	AMP_DET	Amplitude detector of XOSC during power up 0: No action 1: Delay qualification of XOSC until amplitude is greater than the threshold.	RW	0
20:18	RESERVED	This bit field is reserved.	RO	0x0
17	OSC_PD	0: Power up both oscillators 1: Power down oscillator not selected by OSC bit (hardware-controlled when selected).	RW	1
16	OSC	System clock oscillator selection 0: 32-MHz crystal oscillator 1: 16-MHz HF-RC oscillator	RW	1
15:11	RESERVED	This bit field is reserved.	RO	0x00

Bits	Field Name	Description	Type	Reset
10:8	IO_DIV	I/O clock rate setting Cannot be higher than OSC setting 000: 32 MHz 001: 16 MHz 010: 8 MHz 011: 4 MHz 100: 2 MHz 101: 1 MHz 110: 0.5 MHz 111: 0.25 MHz	RW	0x1
7:6	RESERVED	This bit field is reserved.	RO	0x0
5	RESERVED	This bit field is reserved.	RW	0
4:3	RESERVED	This bit field is reserved.	RW	0x1
2:0	SYS_DIV	System clock rate setting Cannot be higher than OSC setting 000: 32 MHz 001: 16 MHz 010: 8 MHz 011: 4 MHz 100: 2 MHz 101: 1 MHz 110: 0.5 MHz 111: 0.25 MHz	RW	0x1

SYS_CTRL_CLOCK_STA

Address offset	0x04		
Physical Address	0x400D 2004	**Instance**	SYS_CTRL
Description	Clock status register This register reflects the current chip status.		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED					SYNC_32K	OSC32K_CALDIS	OSC32K	RST		RESERVED	SOURCE_CHANGE	XOSC_STB	HSOSC_STB	OSC_PD	OSC	RESERVED					IO_DIV			RESERVED					SYS_DIV		

Bits	Field Name	Description	Type	Reset
31:27	RESERVED	This bit field is reserved.	RO	0x00
26	SYNC_32K	32-kHz clock source synced to undivided system clock (16 or 32 MHz).	RO	0
25	OSC32K_CALDIS	Disable calibration 32-kHz RC oscillator. 0: Calibration enabled 1: Calibration disabled	RO	0
24	OSC32K	Current 32-kHz clock oscillator selected. 0: 32-kHz crystal oscillator 1: 32-kHz RC oscillator	RO	1
23:22	RST	Returns last source of reset 00: POR 01: External reset 10: WDT 11: CLD or software reset	RO	0x0
21	RESERVED	This bit field is reserved.	RO	0
20	SOURCE_CHANGE	0: System clock is not requested to change. 1: A change of system clock source has been initiated and is not finished. Same as when OSC bit in CLOCK_STA and CLOCK_CTRL register are not equal	RO	0

Bits	Field Name	Description	Type	Reset
19	XOSC_STB	XOSC stable status 0: XOSC is not powered up or not yet stable. 1: XOSC is powered up and stable.	RO	0
18	HSOSC_STB	HSOSC stable status 0: HSOSC is not powered up or not yet stable. 1: HSOSC is powered up and stable.	RO	0
17	OSC_PD	0: Both oscillators powered up and stable and OSC_PD_CMD = 0. 1: Oscillator not selected by CLOCK_CTRL.OSC bit is powered down.	RO	0
16	OSC	Current clock source selected 0: 32-MHz crystal oscillator 1: 16-MHz HF-RC oscillator	RO	1
15:11	RESERVED	This bit field is reserved.	RO	0x00
10:8	IO_DIV	Returns current functional frequency for IO_CLK (may differ from setting in the CLOCK_CTRL register) 000: 32 MHz 001: 16 MHz 010: 8 MHz 011: 4 MHz 100: 2 MHz 101: 1 MHz 110: 0.5 MHz 111: 0.25 MHz	RO	0x1
7:3	RESERVED	This bit field is reserved.	RO	0x0
2:0	SYS_DIV	Returns current functional frequency for system clock (may differ from setting in the CLOCK_CTRL register) 000: 32 MHz 001: 16 MHz 010: 8 MHz 011: 4 MHz 100: 2 MHz 101: 1 MHz 110: 0.5 MHz 111: 0.25 MHz	RO	0x1

SYS_CTRL_RCGCGPT

Address offset	0x08		
Physical Address	0x400D 2008	**Instance**	SYS_CTRL
Description	This register defines the module clocks for GPT[3:0] when the CPU is in active (run) mode. This register setting is don't care for PM1-3, because the system clock is powered down in these modes.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												GPT3	GPT2	GPT1	GPT0

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3	GPT3	0: Clock for GPT3 is gated. 1: Clock for GPT3 is enabled.	RW	0
2	GPT2	0: Clock for GPT2 is gated. 1: Clock for GPT2 is enabled.	RW	0
1	GPT1	0: Clock for GPT1 is gated. 1: Clock for GPT1 is enabled.	RW	0
0	GPT0	0: Clock for GPT0 is gated. 1: Clock for GPT0 is enabled.	RW	0

SYS_CTRL_SCGCGPT

Address offset	0x0C		
Physical Address	0x400D 200C	**Instance**	SYS_CTRL
Description	This register defines the module clocks for GPT[3:0] when the CPU is in sleep mode. This register setting is don't care for PM1-3, because the system clock is powered down in these modes.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												GPT3	GPT2	GPT1	GPT0

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3	GPT3	0: Clock for GPT3 is gated. 1: Clock for GPT3 is enabled.	RW	0
2	GPT2	0: Clock for GPT2 is gated. 1: Clock for GPT2 is enabled.	RW	0
1	GPT1	0: Clock for GPT1 is gated. 1: Clock for GPT1 is enabled.	RW	0
0	GPT0	0: Clock for GPT0 is gated. 1: Clock for GPT0 is enabled.	RW	0

SYS_CTRL_DCGCGPT

Address offset	0x10		
Physical Address	0x400D 2010	**Instance**	SYS_CTRL
Description	This register defines the module clocks for GPT[3:0] when the CPU is in PM0. This register setting is don't care for PM1-3, because the system clock is powered down in these modes.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												GPT3	GPT2	GPT1	GPT0

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3	GPT3	0: Clock for GPT3 is gated. 1: Clock for GPT3 is enabled.	RW	0
2	GPT2	0: Clock for GPT2 is gated. 1: Clock for GPT2 is enabled.	RW	0
1	GPT1	0: Clock for GPT1 is gated. 1: Clock for GPT1 is enabled.	RW	0
0	GPT0	0: Clock for GPT0 is gated. 1: Clock for GPT0 is enabled.	RW	0

SYS_CTRL_SRGPT

Address offset	0x14		
Physical Address	0x400D 2014	**Instance**	SYS_CTRL
Description	This register controls the reset for GPT[3:0].		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												GPT3	GPT2	GPT1	GPT0

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3	GPT3	0: GPT3 module is not reset 1: GPT3 module is reset	RW	0
2	GPT2	0: GPT2 module is not reset 1: GPT2 module is reset	RW	0
1	GPT1	0: GPT1 module is not reset 1: GPT1 module is reset	RW	0
0	GPT0	0: GPT0 module is not reset 1: GPT0 module is reset	RW	0

SYS_CTRL_RCGCSSI

Address offset	0x18		
Physical Address	0x400D 2018	**Instance**	SYS_CTRL
Description	This register defines the module clocks for SSI[1:0] when the CPU is in active (run) mode. This register setting is don't care for PM1-3, because the system clock is powered down in these modes.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																														SSI1	SSI0

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	RO	0x0000 0000
1	SSI1	0: Clock for SSI1 is gated. 1: Clock for SSI1 is enabled.	RW	0
0	SSI0	0: Clock for SSI0 is gated. 1: Clock for SSI0 is enabled.	RW	0

SYS_CTRL_SCGCSSI

Address offset	0x1C		
Physical Address	0x400D 201C	**Instance**	SYS_CTRL
Description	This register defines the module clocks for SSI[1:0] when the CPU is insSleep mode. This register setting is don't care for PM1-3, because the system clock is powered down in these modes.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																														SSI1	SSI0

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	RO	0x0000 0000
1	SSI1	0: Clock for SSI1 is gated. 1: Clock for SSI1 is enabled.	RW	0
0	SSI0	0: Clock for SSI0 is gated. 1: Clock for SSI0 is enabled.	RW	0

SYS_CTRL_DCGCSSI

Address offset	0x20		
Physical Address	0x400D 2020	**Instance**	SYS_CTRL
Description	This register defines the module clocks for SSI[1:0] when the CPU is in PM0. This register setting is don't care for PM1-3, because the system clock is powered down in these modes.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																														SSI1	SSI0

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	RO	0x0000 0000
1	SSI1	0: Clock for SSI1 is gated. 1: Clock for SSI1 is enabled.	RW	0
0	SSI0	0: Clock for SSI0 is gated. 1: Clock for SSI0 is enabled.	RW	0

SYS_CTRL_SRSSI

Address offset	0x24		
Physical Address	0x400D 2024	**Instance**	SYS_CTRL
Description	This register controls the reset for SSI[1:0].		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																														SSI1	SSI0

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	RO	0x0000 0000
1	SSI1	0: SSI1 module is not reset 1: SSI1 module is reset	RW	0
0	SSI0	0: SSI0 module is not reset 1: SSI0 module is reset	RW	0

SYS_CTRL_RCGCUART

Address offset	0x28		
Physical Address	0x400D 2028	**Instance**	SYS_CTRL
Description	This register defines the module clocks for UART[1:0] when the CPU is in active (run) mode. This register setting is don't care for PM1-3, because the system clock is powered down in these modes.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																														UART1	UART0

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	RO	0x0000 0000
1	UART1	0: Clock for UART1 is gated. 1: Clock for UART1 is enabled.	RW	0
0	UART0	0: Clock for UART0 is gated. 1: Clock for UART0 is enabled.	RW	0

SYS_CTRL_SCGCUART

Address offset	0x2C		
Physical Address	0x400D 202C	**Instance**	SYS_CTRL
Description	This register defines the module clocks for UART[1:0] when the CPU is in sleep mode. This register setting is don't care for PM1-3, because the system clock is powered down in these modes.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																														UART1	UART0

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	RO	0x0000 0000
1	UART1	0: Clock for UART1 is gated. 1: Clock for UART1 is enabled.	RW	0
0	UART0	0: Clock for UART0 is gated. 1: Clock for UART0 is enabled.	RW	0

SYS_CTRL_DCGCUART

Address offset	0x30		
Physical Address	0x400D 2030	**Instance**	SYS_CTRL
Description	This register defines the module clocks for UART[1:0] when the CPU is in PM0. This register setting is don't care for PM1-3, because the system clock is powered down in these modes.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																														UART1	UART0

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	RO	0x0000 0000
1	UART1	0: Clock for UART1 is gated. 1: Clock for UART1 is enabled.	RW	0
0	UART0	0: Clock for UART0 is gated. 1: Clock for UART0 is enabled.	RW	0

SYS_CTRL_SRUART

Address offset	0x34		
Physical Address	0x400D 2034	**Instance**	SYS_CTRL
Description	This register controls the reset for UART[1:0].		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																														UART1	UART0

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	RO	0x0000 0000
1	UART1	0: UART1 module is not reset 1: UART1 module is reset	RW	0

Bits	Field Name	Description	Type	Reset
0	UART0	0: UART0 module is not reset 1: UART0 module is reset	RW	0

SYS_CTRL_RCGCI2C

Address offset 0x38

Physical Address 0x400D 2038 **Instance** SYS_CTRL

Description This register defines the module clocks for I2C when the CPU is in active (run) mode. This register setting is don't care for PM1-3, because the system clock is powered down in these modes.

Type RW

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																															I2C0

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	RO	0x0000 0000
0	I2C0	0: Clock for I2C0 is gated. 1: Clock for I2C0 is enabled.	RW	0

SYS_CTRL_SCGCI2C

Address offset 0x3C

Physical Address 0x400D 203C **Instance** SYS_CTRL

Description This register defines the module clocks for I2C when the CPU is in sleep mode. This register setting is don't care for PM1-3, because the system clock is powered down in these modes.

Type RW

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																															I2C0

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	RO	0x0000 0000
0	I2C0	0: Clock for I2C0 is gated. 1: Clock for I2C0 is enabled.	RW	0

SYS_CTRL_DCGCI2C

Address offset 0x40

Physical Address 0x400D 2040 **Instance** SYS_CTRL

Description This register defines the module clocks for I2C when the CPU is in PM0. This register setting is don't care for PM1-3, because the system clock is powered down in these modes.

Type RW

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																															I2C0

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	RO	0x0000 0000
0	I2C0	0: Clock for I2C0 is gated. 1: Clock for I2C0 is enabled.	RW	0

SYS_CTRL_SRI2C

Address offset	0x44		
Physical Address	0x400D 2044	**Instance**	SYS_CTRL
Description	This register controls the reset for I2C.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																															I2C0

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	RO	0x0000 0000
0	I2C0	0: I2C0 module is not reset 1: I2C0 module is reset	RW	0

SYS_CTRL_RCGCSEC

Address offset	0x48		
Physical Address	0x400D 2048	**Instance**	SYS_CTRL
Description	This register defines the module clocks for the security module when the CPU is in active (run) mode. This register setting is don't care for PM1-3, because the system clock is powered down in these modes.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	RO	0x0000 0000
1	AES	0: Clock for AES is gated. 1: Clock for AES is enabled.	RW	0
0	PKA	0: Clock for PKA is gated. 1: Clock for PKA is enabled.	RW	0

SYS_CTRL_SCGCSEC

Address offset	0x4C		
Physical Address	0x400D 204C	**Instance**	SYS_CTRL
Description	This register defines the module clocks for the security module when the CPU is in sleep mode. This register setting is don't care for PM1-3, because the system clock is powered down in these modes.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	RO	0x0000 0000
1	AES	0: Clock for AES is gated. 1: Clock for AES is enabled.	RW	0
0	PKA	0: Clock for PKA is gated. 1: Clock for PKA is enabled.	RW	0

SYS_CTRL_DCGCSEC

Address offset	0x50		
Physical Address	0x400D 2050	**Instance**	SYS_CTRL
Description	This register defines the module clocks for the security module when the CPU is in PM0. This register setting is don't care for PM1-3, because the system clock is powered down in these modes.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	RO	0x0000 0000
1	AES	0: Clock for AES is gated. 1: Clock for AES is enabled.	RW	0
0	PKA	0: Clock for PKA is gated. 1: Clock for PKA is enabled.	RW	0

SYS_CTRL_SRSEC

Address offset	0x54		
Physical Address	0x400D 2054	**Instance**	SYS_CTRL
Description	This register controls the reset for the security module.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	RO	0x0000 0000
1	AES	0: AES module is not reset 1: AES module is reset	RW	0
0	PKA	0: PKA module is not reset 1: PKA module is reset	RW	0

SYS_CTRL_PMCTL

Address offset	0x58		
Physical Address	0x400D 2058	**Instance**	SYS_CTRL
Description	This register controls the power mode. Note: The Corresponding PM is not entered before the WFI instruction is asserted. To enter PM1-3 the DEEPSLEEP bit in SYSCTRL must be 1.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	RO	0x0000 0000
1:0	PM	00: No action 01: PM1 10: PM2 11: PM3	RW	0x0

SYS_CTRL_SRCRC

Address offset	0x5C		
Physical Address	0x400D 205C	**Instance**	SYS_CTRL
Description	This register controls CRC on state retention.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:13	RESERVED	This bit field is reserved.	RO	0x0 0000
12	RESERVED	This bit field is reserved.	RO	0
11:10	RESERVED	This bit field is reserved.	RW W0toClr[11:10]	0x0
9	RESERVED	This bit field is reserved.	RO	0
8	CRC_REN_USB	1: Enable reset of chip if CRC fails. 0: Disable reset feature of chip due to CRC.	RW	0
7:5	RESERVED	This bit field is reserved.	RO	0x0
4	RESERVED	This bit field is reserved.	RO	0
3:2	RESERVED	This bit field is reserved.	RW W0toClr[3:2]	0x0
1	RESERVED	This bit field is reserved.	RO	0
0	CRC_REN_RF	1: Enable reset of chip if CRC fails. 0: Disable reset feature of chip due to CRC.	RW	0

SYS_CTRL_PWRDBG

Address offset	0x74		
Physical Address	0x400D 2074	**Instance**	SYS_CTRL
Description	Power debug register		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3	FORCE_WARM_RESET	0: No action 1: When written high, the chip is reset in the same manner as a CLD event and is readable from the RST field in the CLOCK_STA register.	RW	0

Bits	Field Name	Description	Type	Reset
2	RESERVED	This bit field is reserved.	RW	1
1	RESERVED	This bit field is reserved.	RW	0
0	RESERVED	This bit field is reserved.	RW	0

SYS_CTRL_CLD

Address offset	0x80		
Physical Address	0x400D 2080	**Instance**	SYS_CTRL
Description	This register controls the clock loss detection feature.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:9	RESERVED	This bit field is reserved.	RO	0x00 0000
8	VALID	0: CLD status in always-on domain is not equal to status in the EN register. 1: CLD status in always-on domain and EN register are equal.	RO	1
7:1	RESERVED	This bit field is reserved.	RO	0x00
0	EN	0: CLD is disabled. 1: CLD is enabled. Writing to this register shall be ignored if VALID = 0	RW	0

SYS_CTRL_IWE

Address offset	0x94		
Physical Address	0x400D 2094	**Instance**	SYS_CTRL
Description	This register controls interrupt wake-up.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																										SM_TIMER_IWE	USB_IWE	PORT_D_IWE	PORT_C_IWE	PORT_B_IWE	PORT_A_IWE

Bits	Field Name	Description	Type	Reset
31:6	RESERVED	This bit field is reserved.	RO	0x000 0000
5	SM_TIMER_IWE	1: Enable SM Timer wake-up interrupt. 0: Disable SM Timer wake-up interrupt.	RW	1
4	USB_IWE	1: Enable USB wake-up interrupt. 0: Disable USB wake-up interrupt.	RW	1
3	PORT_D_IWE	1: Enable port D wake-up interrupt. 0: Disable port D wake-up interrupt.	RW	1
2	PORT_C_IWE	1: Enable port C wake-up interrupt. 0: Disable port C wake-up interrupt.	RW	1
1	PORT_B_IWE	1: Enable port B wake-up interrupt. 0: Disable port B wake-up interrupt.	RW	1
0	PORT_A_IWE	1: Enable port A wake-up interrupt. 0: Disable port A wake-up interrupt.	RW	1

SYS_CTRL_I_MAP

Address offset	0x98		
Physical Address	0x400D 2098	**Instance**	SYS_CTRL
Description	This register selects which interrupt map to be used.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	RO	0x0000 0000
0	ALTMAP	1: Select alternate interrupt map. 0: Select regular interrupt map. (See the ASD document for details.)	RW	0

SYS_CTRL_RCGCRFC

Address offset	0xA8		
Physical Address	0x400D 20A8	**Instance**	SYS_CTRL
Description	This register defines the module clocks for RF CORE when the CPU is in active (run) mode. This register setting is don't care for PM1-3, because the system clock is powered down in these modes.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																															RFC0

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	RO	0x0000 0000
0	RFC0	0: Clock for RF CORE is gated. 1: Clock for RF CORE is enabled.	RW	0

SYS_CTRL_SCGCRFC

Address offset	0xAC		
Physical Address	0x400D 20AC	**Instance**	SYS_CTRL
Description	This register defines the module clocks for RF CORE when the CPU is in sleep mode. This register setting is don't care for PM1-3, because the system clock is powered down in these modes.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																															RFC0

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	RO	0x0000 0000
0	RFC0	0: Clock for RF CORE is gated. 1: Clock for RF CORE is enabled.	RW	0

SYS_CTRL_DCGCRFC

Address offset	0xB0		
Physical Address	0x400D 20B0	**Instance**	SYS_CTRL
Description	This register defines the module clocks for RF CORE when the CPU is in PM0. This register setting is don't care for PM1-3, because the system clock is powered down in these modes.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																															RFC0

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	RO	0x0000 0000
0	RFC0	0: Clock for RF CORE is gated. 1: Clock for RF CORE is enabled.	RW	0

SYS_CTRL_EMUOVR

Address offset	0xB4		
Physical Address	0x400D 20B4	**Instance**	SYS_CTRL
Description	This register defines the emulator override controls for power mode and peripheral clock gate.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								ICEPICK_FORCE_CLOCK_CG	ICEPICK_FORCE_POWER_CG	ICEPICK_INHIBIT_SLEEP_CG	ICEMELTER_WKUP_CG	ICEPICK_FORCE_CLOCK_PM	ICEPICK_FORCE_POWER_PM	ICEPICK_INHIBIT_SLEEP_PM	ICEMELTER_WKUP_PM

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	ICEPICK_FORCE_CLOCK_CG	ICEPick 'Force Active' clock gate override bit. 'Force Active' is an ICEPick command. 1 --> In non-sleep power mode, peripherals clocks are forced to follow RCG* register settings. It forces CM3 clocks on. 0 --> Does not affect the peripheral clock settings.	RW	1
6	ICEPICK_FORCE_POWER_CG	ICEPick 'Force Power' clock gate override bit. 'Force Power' is an ICEPick command. 1 --> In non-sleep power mode, peripherals clocks are forced to follow RCG* register settings. It forces CM3 clocks on. 0 --> Does not affect the peripheral clock settings.	RW	1
5	ICEPICK_INHIBIT_SLEEP_CG	ICEPick 'Inhibit Sleep' clock gate override bit. 'Inhibit Sleep' is an ICEPick command. 1 --> In non-sleep power mode, peripherals clocks are forced to follow RCG* register settings. It forces CM3 clocks on. 0 --> Does not affect the peripheral clock settings.	RW	1
4	ICEMELTER_WKUP_CG	ICEMelter 'WAKEUPEMU' clock gate override bit. 1 --> In non-sleep power mode, peripherals clocks are forced to follow RCG* register settings. It forces CM3 clocks on. 0 --> Does not affect the peripheral clock settings	RW	1

Bits	Field Name	Description	Type	Reset
3	ICEPICK_FORCE_CLOCK_PM	ICEPick 'Force Active' power mode override bit. 'Force Active' is an ICEPick command. 1 --> Prohibit the system to go into any power down modes. Keeps the emulator attached. 0 --> Does not override any power mode settings from SYSREGS and does not prohibit system to go into any power down modes.	RW	1
2	ICEPICK_FORCE_POWER_PM	ICEPick 'Force Power' power mode override bit. 'Force Power' is an ICEPick command. 1 --> Prohibit the system to go into any power down modes. Keeps the emulator attached. 0 --> Does not override any power mode settings from SYSREGS and does not prohibit system to go into any power down modes.	RW	1
1	ICEPICK_INHIBIT_SLEEP_PM	ICEPick 'Inhibit Sleep' power mode override bit. 'Inhibit Sleep' is an ICEPick command. 1 --> Prohibit the system to go into any power down modes. Keeps the emulator attached. 0 --> Does not override any power mode settings from SYSREGS and does not prohibit system to go into any power down modes.	RW	1
0	ICEMELTER_WKUP_PM	ICEMelter 'WAKEUPEMU' power mode override bit. 1 --> Prohibit the system to go into any power down modes. Keeps the emulator attached. 0 --> Does not override any power mode settings from SYSREGS and does not prohibit system to go into any power down modes.	RW	1

Internal Memory

This chapter describes the Flash, ROM and SRAM of the SoC.

Topic **Page**

8.1 Introduction

The CC2538 family contains flash memory up to 512 kB for storage of program code. The flash memory is programmable from the user software and through the debug interface.

The flash controller handles writing and erasing the embedded flash memory. The embedded flash memory consists of up to 256 pages of 2048 bytes each.

The flash controller supports program/erase functionality with the following timings:

- Flash-page erase timing: minimum 20 ms
- Flash-chip erase timing: minimum 20 ms
- Flash-write timing (4 bytes): minimum 20 µs

8.2 Flash Memory Organization

The flash memory is divided into 2048-byte flash pages. A flash page is the smallest erasable unit in the memory, whereas a 32-bit word is the smallest writable unit that can be written to the flash.

When performing write operations, the flash memory is word-addressable using a 17-bit address written to the address registers FADDR.

When performing page-erase operations, the flash memory page to be erased is addressed through the register bits FADDR [16:9].

8.2.1 Information Page

The information block is used to store configuration settings and is not possible to write or erase this page.

NOTE: While reading the information page, prefetch must be disabled.

8.2.2 Lock Bit Page

The lock bits are placed in the uppermost available page which is called the lock bit page. When a page is locked, a write or erase operation to the page is aborted. The uppermost available page number depends on the flash size option (Table 8-2). For example, for the 512 KB option, the Lock Bit Page is page number 255. The Lock Bit Page is not accessible unless **FLASH_CTRL_FCTL .UPPER_PAGE_ACCESS** is set**.**

The following sections describe the procedures for flash write and flash page-erase in detail.

8.3 Flash Write

The flash is programmed serially with a sequence of one or more 32-bit words (4 bytes), starting at the start address (set by FADDR). In general, a page must be erased before writing can begin. The page-erase operation sets all bits in the page to 1. The mass erase command (through the debug interface) erases all pages in the flash. Erase is the only way to set bits in the flash to 1. When writing a word to the flash, the 0-bits are programmed to 0 and the 1-bits are ignored (leaves the bit in the flash unchanged). Thus, bits are erased to 1 and can be written to 0. It is possible to write multiple times to a word. This is described in Section 8.3.2.

8.3.1 Flash Write Procedure

The flash-write sequence algorithm is as follows:

1. Set FADDR to the start address.
2. Set FCTL.WRITE to 1. This starts the write-sequence state machine.
3. Write the 32-bit data to FWDATA (since the last time FCTL.FULL became 0, if not first iteration). FCTL.FULL goes high after writing to FWDATA

4. Wait until FCTL.FULL goes low. (The flash controller has started programming the 4 bytes written in step 3 and is ready to buffer the next 4 bytes).
5. Optional status check step:
   - If the 4 bytes were not written fast enough in step 3, the operation has timed out and FCTL.BUSY (and FCTL.WRITE) are 0 at this stage.
   - If the 4 bytes could not be written to the flash due to the page being locked, FCTL.BUSY (and FCTL.WRITE) are 0 and FCTL.ABORT is 1.
6. If this was the last 4 bytes then quit, otherwise go to step 3.

The write operation is performed using one of two methods:

- Using DMA transfer (preferred method)
- Using CPU

The CPU cannot access the flash, for example, to read program code, while a flash-write operation is in progress. Therefore, the program code executing the flash write must be executed from ROM or SRAM.

When a flash-write operation is executed from ROM or SRAM, the CPU continues to execute code from the next instruction after initiation of the flash-write operation (FCTL.WRITE = 1). It is important to note that while writing to FLASH, the system must not enter power modes 1, 2 or 3, and the system clock source (XOSC/RCOSC) should not be changed.

8.3.2 Writing Multiple Times to a Word

The following rules apply when writing multiple times to a 32-bit word between erase:

- The same address cannot be programmed (either '0' or '1') more than eight times before the next erase.
- The same bit-cell can be programmed '0' more than two times before the next erase.
- Writing a '1' to a bit does not change the state of the bit

The state of any bit of a 32-bit flash word is nondeterministic if these limitations are violated.

This makes it possible to write up to 4 new bits to a 32-bit word 8 times. One example write sequence to a word is shown in Table 8-1. Here b_n represents the 4 new bits written to the word for each update. This technique is useful to maximize the lifetime of the flash for data-logging applications.

Table 8-1. Example Write Sequence

Step	Value Written	FLASH Contents After Writing	Comment
1	(page erase)	0xFFFFFFFF	The erase sets all bits to 1.
2	0xFFFFFFFb_0	0xFFFFFFFb_0	Only the bits written 0 are set to 0, whereas all bits written 1 are ignored.
3	0xFFFFFFb_1F	0xFFFFFFb_1b_0	Only the bits written 0 are set to 0, whereas all bits written 1 are ignored.
4	0xFFFFFb_2FF	0xFFFFF$b_2b_1b_0$	Only the bits written 0 are set to 0, whereas all bits written 1 are ignored.
5	0xFFFFb_3FFF	0xFFFF$b_3b_2b_1b_0$	Only the bits written 0 are set to 0, whereas all bits written 1 are ignored.
6	0xFFFb_4FFFF	0xFFF$b_4b_3b_2b_1b_0$	Only the bits written 0 are set to 0, whereas all bits written 1 are ignored.
7	0xFFb_5FFFFF	0xFF$b_5b_4b_3b_2b_1b_0$	Only the bits written 0 are set to 0, whereas all bits written 1 are ignored.
8	0xFb_6FFFFFF	0xF$b_6b_5b_4b_3b_2b_1b_0$	Only the bits written 0 are set to 0, whereas all bits written 1 are ignored.
9	0xb_7FFFFFFF	0x$b_7b_6b_5b_4b_3b_2b_1b_0$	Only the bits written 0 are set to 0, whereas all bits written 1 are ignored.

8.3.3 DMA Flash Write

When using DMA write operations, make sure the transfer mode of the DMA is basic with arbitration size = 1.The data to be written into flash is stored in the SRAM memory. The DMA Flash channel is configured to read the data to be written from the memory source address and write this data to the flash write-data register (FWDATA) fixed destination address. Thus, the flash controller triggers a DMA transfer when the flash write-data register, FWDATA, is ready to receive new data. The DMA channel should be configured to perform basic mode, 32-bit transfers with the source address set to start-of-data block and destination address to fixed FWDATA. High priority should also be ensured for the DMA channel, so it is not interrupted in the write process. If interrupted for more than 20 µs, the write operation may time out, and the write bit, FCTL.WRITE, is set to 0.

When the DMA channel is enabled, starting a flash write by setting FCTL.WRITE to 1 triggers the first DMA transfer (DMA and flash controller handle the reset of the transfer).

Figure 8-1 shows an example of how a DMA channel is configured and how a DMA transfer is initiated to write a block of data from a location in SRAM to flash memory.

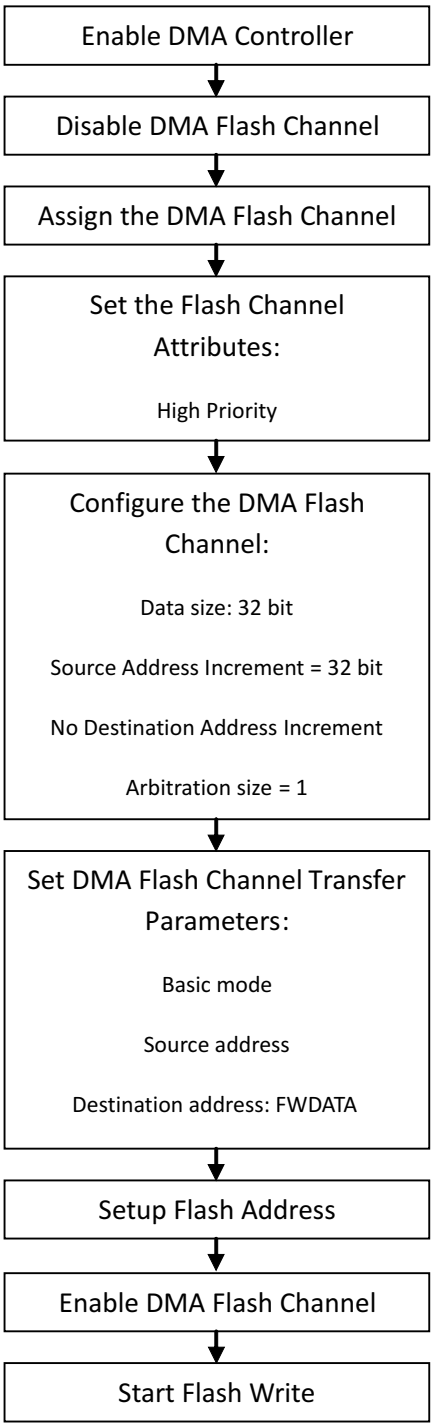


Figure 8-1. Flash Write Using DMA

8.3.4 CPU Flash Write

To write to the flash using the CPU, a program executing from SRAM must implement the steps outlined in the procedure described in Section 8.3.1. Disable interrupts to ensure the operation does not time out.

8.4 Flash DMA Trigger

The flash DMA trigger is activated when flash data written to the FWDATA register has been written to the specified location in the flash memory, thus indicating that the flash controller is ready to accept new data to be written to FWDATA. In order to start the first transfer, one must set the FCTL.WRITE bit to 1. The DMA and the flash controller then handle all transfers automatically for the defined block of data (LEN in DMA configuration). It is further important that the DMA is enabled prior to setting the FCTL.WRITE bit, and that the DMA has high priority so the transfer is not interrupted. If interrupted for more than 20 µs, the write operation times out and FCTL.WRITE bit is cleared.

8.5 Flash Page Erase

The flash page-erase operation sets all bits in the page to 1. Please note that when erasing a flash page (not an information page), the offset address from the flash base address must be written to **FLASH_CTRL_FADDR**. If an information page is to be erased, the offset address from the information page base address must be written. Also note that if a page erase is initiated simultaneously with a page write, that is, FCTL.WRITE is set to 1, the page erase is performed before the page-write operation starts. The FCTL.BUSY bit can be polled to see when the page erase has completed.

NOTE: If a flash page-erase operation is performed from within flash memory and the Watchdog Timer is enabled, a Watchdog Timer interval must be selected that is longer than 20 ms, the duration of the flash page-erase operation, so that the CPU can clear the Watchdog Timer.

8.5.1 Performing Flash Erase from Flash Memory

Note that while executing program code from within flash memory, when a flash erase is initiated the CPU stalls, and program execution resumes from the next instruction when the flash controller has completed the operation.

The following code example of how to erase one flash page in the CC2538 is given for use with the IAR compiler:

```
unsigned char erase_page_num = 3;   /* page number to erase, here: flash page #3 */

/* Erase one flash page */
while (FLASH_CTRL_FCTL & 0x80);
FLASH_CTRL_FADDR = erase_page_num << 9;
FLASH_CTRL_FCTL |= 0x01;
while (FLASH_CTRL_FCTL & 0x80);
  /* optional: wait until flash write has completed (~20 ms) */
```

8.6 Flash Lock Bit Page and Customer Configuration Area (CCA)

The user can disable erasing and writing to certain pages with a page lock bit that is available to each individual flash page. The lock bits are placed in the uppermost available page (the CCA) together with the boot loader configuration bytes. The uppermost available page number depends on the flash size option. For example, for the 512 KB option, the Lock Bit Page number or CCA is page number 255. When a page n is locked, i.e. LOCK_PAGE_N[n] = 0, a write or erase operation to the page is aborted. The debug lock bit (LOCK_DEBUG_N) is used to lock the debug interface.

The following table shows how the page lock bits and the debug lock bit are laid out in the upper 32 bytes of the CCA page. The table also shows the layout of the boot loader configuration options. Note that some of the page lock bits are unused for certain flash size options.

Table 8-2. Flash Size Configuration

Flash Size in KB	Flash Size [2:0]	Main Memory Pages Address Map
128	001	0x00200000 to 0x0021FFFF
256	010	0x00200000 to 0x0023FFFF

Table 8-2. Flash Size Configuration (continued)

Flash Size in KB	Flash Size [2:0]	Main Memory Pages Address Map
512	100	0x00200000 to 0x0027FFFF

The debug lock bit is set, the access to the CM3 DAP is locked. On the release of the reset, certain location of the lock pages are automatically read once to get the debug lock information. The debug lock bit is propagated to the ICEPick to enable/disable the debug interface through JTAG.

Table 1-3 shows how the page lock bits and the debug lock bit are laid out in upper 32 bytes of the Lock Bit Page. Note that some of the page lock bits are unused for certain flash size options.

Table 8-3. Upper 32 Bytes of Lock Bit Page and CCA layout

Byte	Bits	Name	Description
2047	7	LOCK_DEBUG_N	Debug lock bit 0 - Debug access blocked 1 - Debug access allowed
2047	6:0	LOCK_PAGE_N[254:248]	Page Lock Bits 0 - Write/Erase blocked 1 - Write/Erase allowed
2046	7:0	LOCK_PAGE_N[247:240]	Page Lock Bits 0 - Write/Erase blocked 1 - Write/Erase allowed
2045	7:0	LOCK_PAGE_N[239:232]	Page Lock Bits 0 - Write/Erase blocked 1 - Write/Erase allowed
2044	7:0	LOCK_PAGE_N[231:224]	Page Lock Bits 0 - Write/Erase blocked 1 - Write/Erase allowed
2043	7:0	LOCK_PAGE_N[223:216]	Page Lock Bits 0 - Write/Erase blocked 1 - Write/Erase allowed
...	...	...	...
2017	7:0	LOCK_PAGE_N[15:8]	Page Lock Bits 0 - Write/Erase blocked 1 - Write/Erase allowed
2016	7:0	LOCK_PAGE_N[7:0]	Page Lock Bits 0 - Write/Erase blocked 1 - Write/Erase allowed
2015-2012	-	Application Entry Point	The address of the Vector Table present in the flash memory. Note: The address must be in little endian format
2011-2008	-	Image Valid	Must have the value of 0x00000000 to indicate valid image in falsh
2007	7:0	Bootloader backdoor	Back door configuration
2006	7:0	Reserved	Reserved for future use
2005	7:0	Reserved	Reserved for future use
2004	7:0	Reserved	Reserved for future use

The remaining bytes of the Lock Bit Page, i.e. bytes [0, 2003], can be used for program code/data.

8.6.1 Flash Image Valid Bits in Lock Bit Page

Two 32-bits fields are defined below the upper 32 bytes. These fields are used by the ROM based startup FW in order to determine if a valid application image is present in the flash memory.

If both these fields indicate that a valid application image is present the startup FW will branch to the address of the Reset ISR defined in the Vector Table of the application image in flash. If either indicates presence of an invalid image in the flash, the startup FW will enter the ROM based boot loader. The table below defines the location of these fields within the Lock Bit page and specifies the values they must hold in order to make the startup FW able to enter the application image in the flash memory.

Table 8-4. Fields at POR/ Reset

Bytes	Name	Description
2015 - 2012	Image Vector Table Address	The address of the vector table present in the flash memory. The address must be in little endian format
2011 - 2008	Image valid	Must have the value of 0x00000000 to indicate valid image in flash

The fields described in the table above are read by the startup FW at POR/Reset and if the ‘Image Vector Table Addr’- field holds an address within flash memory map address area and the

‘Image Valid’-field equals 0x00000000 the startup FW will execute as follows:

- The Vector Table offset register is loaded with the value of the ‘Image Vector Table Addr’.
- The main stack pointer is loaded with the initial stack pointer value found at vector table offset 0 in the vector table given by the ‘Image Vector Table Addr’-field.
- A branch is done to the address found at vector table offset 1 in the vector table given by the ‘Image Vector Table Addr’-field. The found address should be the address of the Reset ISR of the application image in flash.

8.6.2 ROM Boot Loader Backdoor Configuration in Lock Bit Page

Even in the presence of a valid image, ROM bootloader can be entered by enabling the bootloader backdoor. This can be done by enabling the boot loader backdoor. This backdoor functionality is configured by a 32-bit field in the Lock Bit page. The table below describes the layout of the 32-bits field.

Table 8-5. 32 Bitfield of the Lock bit page

Bytes	Name	Description
2007 - 2004	Bootloader backdoor	Byte 2007 - Backdoor configuration Byte 2006 - Reserved Byte 2005 - Reserved Byte 2004 - Reserved

The layout of byte 2007 is shown in the table below:

Table 8-6. Layout of Byte 2007

Bit	Field Element	Description
7-5	Reserved	Reserved. Should be all ones
4	Enable	Enable / Disable backdoor function 1 - Backdoor and boot loader enable 0 - Backdoor and boot loader disable
3	Level	Sets active level for selected pin on pad A 1 - Active High 0 - Active Low

Table 8-6. Layout of Byte 2007 (continued)

Bit	Field Element	Description
0 - 2	Pin Number	The number (0 - 7) of the pin on pad A that is used when backdoor is enabled

NOTE: If the Enable bit is set to 0 in the CCA area of the Lock Bit page, the CC2538 ROM boot loader ignores any received boot loader commands, even if no application image is present in the flash memory. This security feature enhancement makes it impossible for an attacker that has gained access to the ROM boot loader, to reveal any flash image contents by using the boot loader commands. If the boot loader is disabled, it ignores any received boot loader command.

In order to enter the boot loader after power up/reset even if a valid application image is present in flash the selected pin on pad A must externally be set to the configured active level no later than 10 us after reset.

The pin level is just read once. Eventual following toggling of the pin level will have no effect. Please note that within the 32-bit Boot Loader Backdoor field it is Lock Bit page byte 2007 that holds the backdoor configuration (the other three bytes are reserved). This is the MSB byte of the 32-bit field in little endian format.

For an empty flash, the ROM boot loader backdoor is by default enabled with an active level configured to high on PA7.

8.7 Flash Mass Erase

Flash mass erase erases all available pages of the flash memory without altering the information page. The Mass Erase command is initiated through ICEPick. It may take several clock cycles before the Mass Erase operation starts. This is because ongoing operations and pending higher priority operations must complete first. See table below for details about the priority.

NOTE: Caches are also invalidated on commencement of mass erase

Table 8-7. Flash Controller Priorities

Priority	Request	Comments	Which FSM initiates
1	Read Configurable bits	Issued only once on reset	Programming FSM
2	Read Debug lock bit	Issued only once on reset	Programming FSM
3	Page Erase	Erases page	Programming FSM
4	Write	Writes to a location in a page	Programming FSM
5	Mass Erase	Erases all main pages	Programming FSM
6	Change Cache Mode		
7	Read from Flash		Read FSM
8	Prefetch		Read FSM

The lock bit, which controls the access of the DAP/TAP through ICEpick, is read by default at power on reset. A value of 1 enables the access to all the secondary/slave DAP/TAP connected to the ICEPick while 0 disables all the access. To enable the access, the steps in the following section are necessary.

For reference the table below shows the mapping of TAP state with the potapstate[3:0] bus on the extended interface of the ICEPick.

Table 8-8. ICEpick TAP State

State	POTAPSTATE [3:0]
POTAPTLR	0000
SELECTDR	0001
CAPDR	0010
SHIFTDR	0011
EXIT1DR	0100
PAUSEDR	0101
EXIT2DR	0110
UPDTDR	0111
IDLE	1000
SLECTIR	1001
CAPIR	1010
SHIFTIR	1011
EXIT1IR	1100
PAUSEIR	1101
EXIT2IR	1110
UPDTIR	1111

8.7.1 Flash Mass Erase Procedure

This procedure will erase all of flash main pages.

Step 1: Initiate mass erase

- Scan “Public Connect Sequence (with 0x07 IR followed by 0x89 DR)” for ICEPick. Refer to ICEPick functional spec for details on “Public Connect Sequence”.
- Scan “Private Connect Sequence (with 0x1F IR followed by 0x01 DR)” for ICEPick.
- Do IR scan 001101 (0x0D) followed by IR scan 001110(0x0E) for ICEPick. Once the BIST is completed for all the RAM memories, it will issue the mass erase command to the flash.
- Monitor the status register for 0x0E IR (**Data register description for instruction 0x0E (READ ONLY)** see table below). Following 0x0E IR, the data register read (DR) reflects status of various activities related to the flash mass erase command.
- Look for bits [0:4] & [6] being high and bit [5] being low. When this condition is true, the debug unlock sequence is complete.
- After the status mentioned above is achieved, issue an ICEPick instruction 111110 (0x3E) to clear the debug unlock command.
- Any other sequence of the IRs for ICEPick, will be ignored and flash erase will not be executed.

Step 2 : Perform External Reset

- The debug interface is not unlocked unless an external reset is performed

Table 8-9. Data Register Description for Instruction 0x0E (READ ONLY)

Bits	Name	Type	Description
31:7	NA		Not used
6	flash_erase_run	R	This reflects the command issued to the flash controller for the mass erase 1 - Command issued 0 - Command is not issued

Table 8-9. Data Register Description for Instruction 0x0E (READ ONLY) (continued)

Bits	Name	Type	Description
5	flash_erase_busy	R	1 - Mass erase is in progress 0 - Mass erase command is completed or there is no mass erase command in progress. This bit only takes effect after bit 6 is high. After bit 6 goes high, wait for 10 TCK before checking this bit.
0:4	reserved	R	reserved

8.8 ROM Sub System

The internal ROM of the CC2538 is located at address 0x0000 0000 of the device memory map. The ROM contains the following components:

- Bootloader
- RAM BIST algorithms
- Security certificates
- Utility library

The ROM is preprogrammed with a serial bootloader (SPI, or UART). For applications that require in-field programmability, the royalty-free bootloader can act as an application loader and support in-field firmware updates. The bootloader executes automatically if no valid image has been written to the flash.

8.9 SRAM

The CC2538 provides a 16kB of single-cycle on-chip SRAM with full retention in all power modes. In addition, some variants offer an additional 16kB of single-cycle on-chip SRAM without retention in the lowest power modes. Because read-modify-write (RMW) operations are very time consuming, ARM has introduced bit-banding technology in the Cortex-M3 processor. With a bit-band-enabled processor, certain regions in the memory map (SRAM and peripheral space) can use address aliases to access individual bits in a single, atomic operation.

Data can also be transferred to and from the SRAM using the micro direct memory access controller (µDMA).

The internal SRAM of the CC2538 is located at 0x2000 0000 of the device memory map, and has the following features:

- Two seperate modules (only true for device variants with largest RAM sizes).
  - 16kB of regular leakage RAM
  - 16kB ultra low leakage RAM
- Dedicated 1kB of ultra low leakage RAM for RF Core and USB modules
- Each module supports byte access capability

When the device prepares to enter into the PM2 or PM3 mode, some of the configuration data and internal signals may be saved in the ultra low leakage SRAM by the processor. The following have their states stored in the SRAM upon entering PM2 or PM3, and are restored upon exiting these modes:

- Cortex M3 System Control
- System Registers
- Flash Control
- Calibration Values
- General Purpose Timer Modules 0-2 (Not GPTM 3)
- Sleep Timer
- MAC Timer

- GPIO Control
- Analog Control
- RF Core
- SSI Control
- UART Control
- Analog Comparator
- uDMA Control
- Key store RAM
- Crypto core's key store module registers

For more information concerning power modes, see Section 7.1.

NOTE: Note that the program stack must reside in the retention part of the SRAM otherwise the program will crash after waking up from PM2 or PM3 modes.

8.10 Flash Control Registers

8.10.1 FLASH_CTRL Registers

8.10.1.1 FLASH_CTRL Registers Mapping Summary

This section provides information on the FLASH_CTRL module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 8-10. FLASH_CTRL Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
FLASH_CTRL_FCTL	RW	32	0x0000 0004	0x08	0x400D 3008
FLASH_CTRL_FADDR	RW	32	0x0000 0000	0x0C	0x400D 300C
FLASH_CTRL_FWDATA	RW	32	0x0000 0000	0x10	0x400D 3010
FLASH_CTRL_DIECFG0	RO	32	0xB964 0580	0x14	0x400D 3014
FLASH_CTRL_DIECFG1	RO	32	0x0000 0000	0x18	0x400D 3018
FLASH_CTRL_DIECFG2	RO	32	0x0000 2000	0x1C	0x400D 301C

8.10.1.2 FLASH_CTRL Register Descriptions

FLASH_CTRL_FCTL

Address offset	0x08		
Physical Address	0x400D 3008	**Instance**	FLASH_CTRL
Description	Flash control This register provides control and monitoring functions for the flash module.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																						UPPER_PAGE_ACCESS	SEL_INFO_PAGE	BUSY	FULL	ABORT	RESERVED	CM		WRITE	ERASE

Bits	Field Name	Description	Type	Reset
31:10	RESERVED	This bit field is reserved.	RO	0x00 0000
9	UPPER_PAGE_ACCESS	Lock bit for lock bit page 0: Neither write nor erase not allowed 1: Both write and erase allowed	RW	0
8	SEL_INFO_PAGE	Flash erase or write operation on APB bus must assert this when accessing the information page	RW	0
7	BUSY	Set when the WRITE or ERASE bit is set; that is, when the flash controller is busy	RO	0
6	FULL	Write buffer full The CPU can write to FWDATA when this bit is 0 and WRITE is 1. This bit is cleared when BUSY is cleared.	RO	0
5	ABORT	Abort status This bit is set to 1 when a write sequence or page erase is aborted. An operation is aborted when the accessed page is locked. Cleared when a write or page erase is started. If a write operation times out (because the FWDATA register is not written fast enough), the ABORT bit is not set even if the page is locked. If a page erase and a write operation are started simultaneously, the ABORT bit reflects the status of the last write operation. For example, if the page is locked and the write times out, the ABORT bit is not set because only the write operation times out.	RO	0
4	RESERVED	This bit field is reserved.	RO	0
3:2	CM	Cache Mode Disabling the cache increases the power consumption and reduces performance. Prefetching improves performance at the expense of a potential increase in power consumption. Real-time mode provides predictable flash read access time, the execution time is equal to cache disabled mode, but the power consumption is lower. 00: Cache disabled 01: Cache enabled 10: Cache enabled, with prefetch 11: Real-time mode Note: The read value always represents the current cache mode. Writing a new cache mode starts a cache mode change request that does not take effect until the controller is ready. Writes to this register are ignored if there is a current cache change request in progress.	RW R*/W*	0x1
1	WRITE	Write bit Start a write sequence by setting this bit to 1. Cleared by hardware when the operation completes. Writes to this bit are ignored when FCTL.BUSY is 1. If FCTL.ERASE is set simultaneously with this bit, the erase operation is started first, then the write is started.	RW R/WH0	0
0	ERASE	Erase bit Start an erase operation by setting this bit to 1. Cleared by hardware when the operation completes. Writes to this bit are ignored when FCTL.BUSY is 1. If FCTL.WRITE is set simultaneously with this bit, the erase operation is started first, then the write is started.	RW R/WH0	0

FLASH_CTRL_FADDR

Address offset	0x0C		
Physical Address	0x400D 300C	**Instance**	FLASH_CTRL
Description	Flash address The register sets the address to be written in flash memory. See the bitfield descriptions for formatting information.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:17	RESERVED	This bit field is reserved.	RO	0x0000
16:0	FADDR	Bit number [16:9] selects one of 256 pages for page erase. Bit number [8:7] selects one of the 4 row in a given page Bit number [6:1] selects one of the 64-bit wide locations in a give row. Bit number [0] will select upper/lower 32-bits in a given 64-bit location - 64Kbytes --> Bits [16:14] will always be 0. - 128Kbytes --> Bits [16:15] will always be 0. - 256Kbytes --> Bit [16] will always be 0. - 384/512Kbytes --> All bits written and valid. Writes to this register will be ignored when any of FCTL.WRITE and FCTL.ERASE is set. FADDR should be written with byte addressable location of the Flash to be programmed. Read back value always reflects a 32-bit aligned address. When the register is read back, the value that was written to FADDR gets right shift by 2 to indicate 32-bit aligned address. In other words lower 2 bits are discarded while reading back the register. Out of range address results in roll over. There is no status signal generated by flash controller to indicate this. Firmware is responsible to managing the addresses correctly.	RW R/W*	0x0 0000

FLASH_CTRL_FWDATA

Address offset	0x10		
Physical Address	0x400D 3010	**Instance**	FLASH_CTRL
Description	Flash data This register contains the 32-bits of data to be written to the flash location selected in FADDR.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	FWDATA	32-bit flash write data Writes to this register are accepted only during a flash write sequence; that is, writes to this register after having written 1 to the FCTL.WRITE bit. New 32-bit data is written only if FCTL.FULL = 0.	RW R0/W	0x0000 0000

FLASH_CTRL_DIECFG0

Address offset	0x14		
Physical Address	0x400D 3014	**Instance**	FLASH_CTRL
Description	These settings are a function of the FLASH information page bit settings, which are programmed during production test, and are subject for specific configuration for multiple device flavors of cc2538.		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CHIPID																RESERVED					CLK_SEL_GATE_EN_N	SRAM_SIZE		FLASH_SIZE			USB_ENABLE	MASS_ERASE_ENABLE	LOCK_FWT_N	LOCK_IP_N		

Bits	Field Name	Description	Type	Reset
31:16	CHIPID	Register copy of configuration bits Three clock cycles after reset is released, this bit field is equal to the field with the same name in the information page.	RO	0xB964
15:11	RESERVED	This bit field is reserved.	RO	0x00
10	CLK_SEL_GATE_EN_N	Register copy of configuration bits Three clock cycles after reset is released, this bit is equal to the field with the same name in the information page.	RO	1
9:7	SRAM_SIZE	Register copy of configuration bits Three clock cycles after reset is released, this bit field is equal to the field with the same name in the information page.	RO	0x3
6:4	FLASH_SIZE	Register copy of configuration bits Three clock cycles after reset is released, this bit field is equal to the field with the same name in the information page.	RO	0x0
3	USB_ENABLE	Register copy of configuration bits Three clock cycles after reset is released, this bit is equal to the field with the same name in the information page.	RO	0
2	MASS_ERASE_ENABLE	Register copy of configuration bits Three clock cycles after reset is released, this bit is equal to the field with the same name in the information page.	RO	0
1	LOCK_FWT_N	Register copy of configuration bits Three clock cycles after reset is released, this bit is equal to the field with the same name in the information page.	RO	0
0	LOCK_IP_N	Register copy of configuration bits Three clock cycles after reset is released, this bit is equal to the field with the same name in the information page.	RO	0

FLASH_CTRL_DIECFG1

Address offset	0x18		
Physical Address	0x400D 3018	**Instance**	FLASH_CTRL
Description	These settings are a function of the FLASH information page bit settings, which are programmed during production test, and are subject for specific configuration for multiple device flavors of cc2538.		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED							I2C_EN	RESERVED						UART1_EN	UART0_EN	RESERVED						SSI1_EN	SSI0_EN	RESERVED				GPTM3_EN	GPTM2_EN	GPTM1_EN	GPTM0_EN

Bits	Field Name	Description	Type	Reset
31:25	RESERVED	This bit field is reserved.	RO	0x00
24	I2C_EN	1: I2C is enabled. 0: I2C is permanently disabled.	RO	0
23:18	RESERVED	This bit field is reserved.	RO	0x00
17	UART1_EN	1: UART1 is enabled. 0: UART1 is permanently disabled.	RO	0

Bits	Field Name	Description	Type	Reset
16	UART0_EN	1: UART0 is enabled. 0: UART0 is permanently disabled.	RO	0
15:10	RESERVED	This bit field is reserved.	RO	0x00
9	SSI1_EN	1: SSI1 is enabled. 0: SSI1 is permanently disabled.	RO	0
8	SSI0_EN	1: SSI0 is enabled. 0: SSI0 is permanently disabled.	RO	0
7:4	RESERVED	This bit field is reserved.	RO	0x0
3	GPTM3_EN	1: GPTM3 is enabled. 0: GPTM3 is permanently disabled.	RO	0
2	GPTM2_EN	1: GPTM2 is enabled. 0: GPTM2 is permanently disabled.	RO	0
1	GPTM1_EN	1: GPTM1 is enabled. 0: GPTM1 is permanently disabled.	RO	0
0	GPTM0_EN	1: GPTM0 is enabled. 0: GPTM0 is permanently disabled.	RO	0

FLASH_CTRL_DIECFG2

Address offset	0x1C		
Physical Address	0x400D 301C	**Instance**	FLASH_CTRL
Description	These settings are a function of the FLASH information page bit settings, which are programmed during production test, and are subject for specific configuration for multiple device flavors of cc2538. The DIE_*_REVISION registers are an exeception to this, as they are hardwired and are not part of the FLASH information page.		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																DIE_MAJOR_REVISION				DIE_MINOR_REVISION				RESERVED					RF_CORE_EN	AES_EN	PKA_EN

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15:12	DIE_MAJOR_REVISION	Indicates the major revision (all layer change) number for the cc2538 0x0 - PG1.0 0x2 - PG2.0	RO	0x2
11:8	DIE_MINOR_REVISION	Indicates the minor revision (metla layer only) number for the cc2538 0x0 - PG1.0 or PG2.0	RO	0x0
7:3	RESERVED	This bit field is reserved.	RO	0x00
2	RF_CORE_EN	1: RF_CORE is enabled. 0: RF_CORE is permanently disabled.	RO	0
1	AES_EN	1: AES is enabled. 0: AES is permanently disabled.	RO	0
0	PKA_EN	1: PKA is enabled. 0: PKA is permanently disabled.	RO	0

General-Purpose Inputs/Outputs

This chapter describes the input/output (I/O) control and the general-purpose inputs/outputs (GPIOs).

Topic **Page**

9.1 I/O Control

The I/O control module connects with all on-chip peripheral external I/Os and routes the signals through a muxing matrix after which they are sent to the GPIO block and to the I/O pads. Signals are driven and received from the various peripherals on the chip. The I/O control module allows these peripheral signals to be routed to or from any of the 32 GPIO pads:

- UART0, UART1
- SSI0 and SSI1
- I^2C
- General Purpose Timers 0, 1, 2 and 3.

One exception is for the GPIO signals, each GPIO signal is assigned to a single GPIO pad.

9.1.1 I/O Muxing

There are 32 selection registers (**IOC_Pxx_SEL**) used to control the mux settings for signals driven out to the chip pads and separate selection registers (for example, **IOC_UARTRXD_UART0**) to direct signals being received from the GPIO pads. The number of input selection registers is equal to the number of signals being muxed to on-chip peripherals.

Table 9-1. Pheripheral Signal Select Values (Same for All IOC_Pxx_SEL Registers)

Select Value (bits 4:0)	Peripheral Output Signal to Pad
0x0	UART0 TXD
0x1	UART1 RTS
0x2	UART1 TXD
0x3	SSI0 TXD
0x4	SSI0 CLK OUT
0x5	SSI0 FSS OUT
0x6	SSI0 TX_SER OUT
0x7	SSI1 TXD
0x8	SSI1 CLK OUT
0x9	SSI1 FSS OUT
0xA	SSI1 TX_SER OUT
0xB	I2C SDA
0xC	I2C SCL
0xD	GPT0CP1
0xE	GPT0CP2
0xF	GPT1CP1
0x10	GPT1CP2
0x11	GPT2CP1
0x12	GPT2CP2
0x13	GPT3CP1
0x14	GPT3CP2

For example to select UART1 TXD to drive PA1 write 0x2 to the **IOC_PA1_SEL** register. To connect PA3 to UART1 RXD write 0x3 to **IOC_UARTRXD_UART1** register. See the GPIO section (Section 9.2.2.3) below to see how to get the PA1 output from the I/O Controller through the GPIO block to the GPIO pad.

9.2 GPIO

9.2.1 General-Purpose Inputs/Outputs

The GPIO module is composed of four physical GPIO blocks, each corresponding to an individual GPIO port (port A , port B , port C , port D). The GPIO module supports 32 programmable I/O pins. The device is configured for individual access to each of these ports. Additionally, each GPIO block allows for single read and write operations for individual GPIO bits.

The GPIO module has the following features:

- Up to 32 GPIOs, depending on configuration
- Highly flexible pin muxing allows use as GPIO or one of several peripheral functions
- Fast toggle capable of a change every two clock cycles
- Programmable control for GPIO interrupts
  - Interrupt generation masking
  - Edge-triggered on rising, falling, or both
  - Level-sensitive on high or low values
- Bit masking in read and write operations through address lines
- Programmable control for GPIO pad configuration
  - Weak pullup or pulldown resistors
  - Digital input enables

9.2.2 Functional Description

Each GPIO port is a separate hardware instantiation of the same physical block (see Figure 9-1). The device contains four ports, and thus four of these physical GPIO blocks. Each port supports 8 I/O pins. All GPIO pins can function as I/O signals for the on-chip peripheral modules. For information on how to setup GPIO pins to be used for alternate hardware (i.e., peripheral) functions, see Section 9.2.2.3.

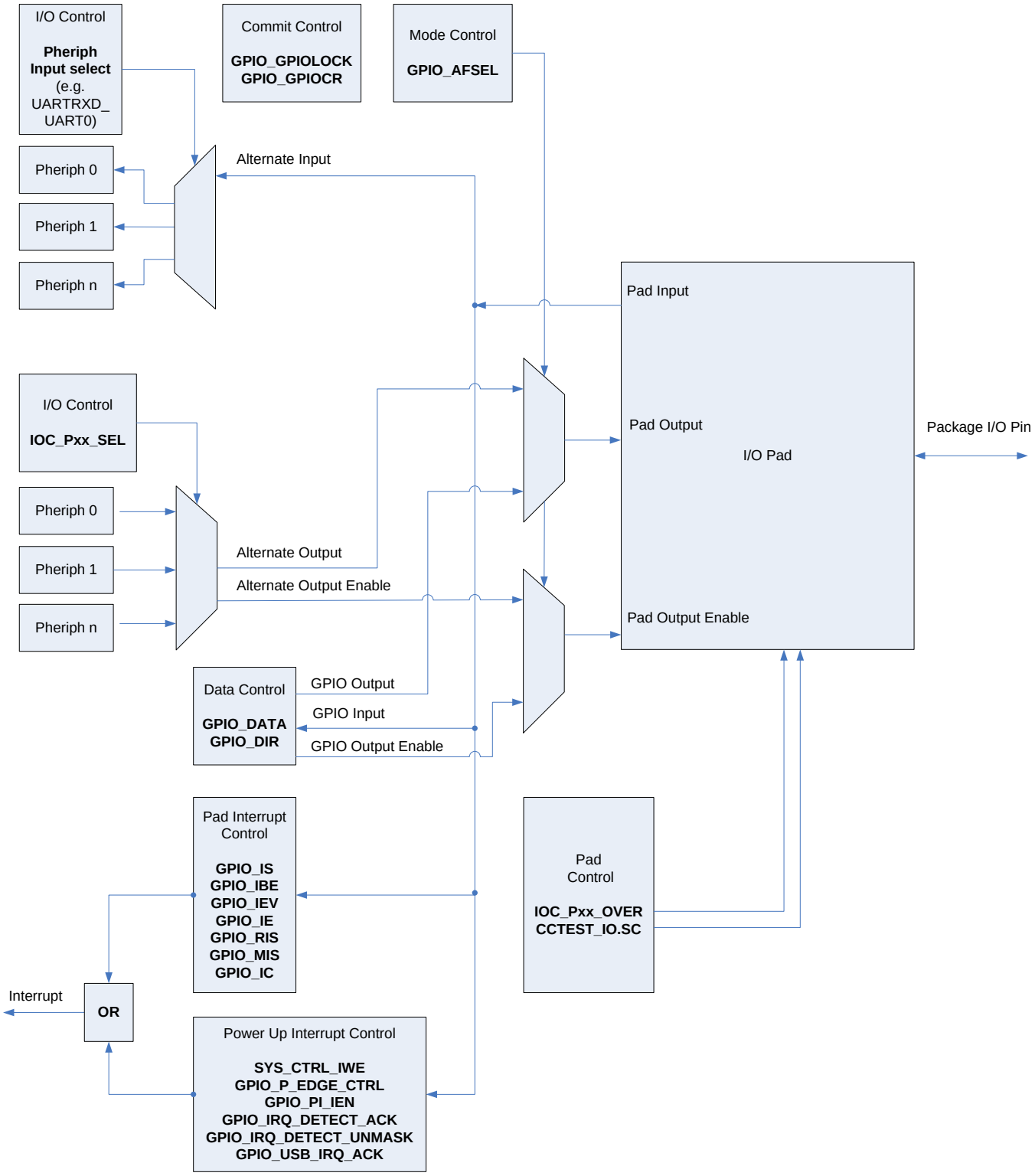


Figure 9-1. Digital I/O Pads (The Diagram Shows One of 32 Possible I/O Pins)

9.2.2.1 Data Control

The data control registers let software configure the operational modes of the GPIOs. The data direction register configures the GPIO as an input or an output while the data register either captures incoming data or drives it out to the pads. The data control register is only valid when the AF register selects GPIO not the alternate function.

9.2.2.1.1 Data Direction Operation

The **GPIO Direction (GPIO_DIR)** register is used to configure each individual pin as an input or output. When the data direction bit is cleared, the GPIO is configured as an input, and the corresponding data register bit captures and stores the value on the GPIO port. When the data direction bit is set, the GPIO is configured as an output, and the corresponding data register bit is driven out on the GPIO port.

9.2.2.1.2 Data Register Operation

To aid in the efficiency of software, the GPIO ports allow for the modification of individual bits in the **GPIO Data (GPIO_DATA)** register by using bits [9:2] of the address bus as a mask. In this manner, software drivers can modify individual GPIO pins in a single instruction without affecting the state of the other pins. This method, which is also known as address bus filtering mechanism, is more efficient than the conventional method of performing a read-modify-write operation to set or clear an individual GPIO pin. To implement this feature, the **GPIO_DATA** register covers 256 locations in the memory map.

During a write, if the address bit associated with that data bit is set, the value of the **GPIO_DATA** register is altered. If the address bit is cleared, the data bit is left unchanged.

For example, writing a value of 0xEB to the address DATA + 0x098 has the results shown in Figure 9-2, where **u** indicates that data is unchanged by the write.

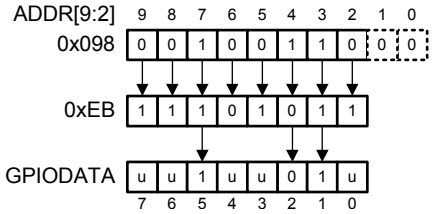


Figure 9-2. GPIODATA Write Example

During a read, if the address bit associated with the data bit is set, the value is read. If the address bit associated with the data bit is cleared, the data bit is read as a 0, regardless of its actual value. For example, reading address DATA + 0x0C4 yields as shown in Figure 9-3.

Figure 9-3. GPIODATA Read Example

9.2.2.2 Interrupt Control

The interrupt capabilities of each GPIO port are controlled by a set of seven registers. These registers are used to select the source of the interrupt, its polarity, and the edge properties. There are four GPIO interrupts, one for each port (A, B, C, and D). When one or more GPIO inputs cause an interrupt, a single interrupt output is sent to the interrupt controller (INTC) for the entire GPIO port. For edge-triggered interrupts, software must clear the interrupt to enable any further interrupts. For a level-sensitive interrupt, the external source must hold the level constant for the interrupt to be recognized by the INTC.

Three registers define the edge or sense that causes interrupts:

- **GPIO Interrupt Sense (GPIO_IS)** register
- **GPIO Interrupt Both Edges (GPIO_IBE)** register
- **GPIO Interrupt Event (GPIO_IEV)** register

Interrupts are enabled and disabled through the **GPIO Interrupt Enable (GPIO_IE)** register.

When an interrupt condition occurs, the state of the interrupt signal can be viewed in two locations: the **GPIO Raw Interrupt Status (GPIO_RIS)** and **GPIO Masked Interrupt Status (GPIO_MIS)** registers. As the name implies, the **GPIO_MIS** register only shows interrupt conditions that are allowed to be passed to the INTC. The **GPIO_RIS** register indicates that a GPIO pin meets the conditions for an interrupt, but has not necessarily been sent to the INTC.

Interrupts are cleared by writing a 1 to the appropriate bit of the **GPIO Interrupt Clear (GPIO_IC)** register.

When programming the interrupt control registers (**GPIO_IS**, **GPIO_IBE**, or **GPIO_IEV**), the interrupts must be masked (**GPIO_IM** cleared). Writing any value to an interrupt control register can generate a spurious interrupt if the corresponding bits are enabled.

9.2.2.2.1 Power-Up Interrupt

Each GPIO pin can be setup to trigger a power-up interrupt that wakes up the device from the various power modes. In addition to the GPIO pins both USB and Sleep Timer can be used to trigger power-up interrupts. The power-up interrupt is enabled by writing the corresponding bits in the **SYS_CTRL_IWE** register. In order to use power-up interrupt on a GPIO port, USB or Sleep Timer perform the following:

- Setup GPIO pin(s) as input (if a GPIO port shall be used as power-up interrupt source)
- Enable wakeup interrupt for the desired port using the **SYS_CTRL_IWE** register
- Set the power-up interrupt type for the specified pin(s), using the **GPIO_P_EDGE_CTRL** register for a GPIO.
- Clear any pending GPIO power-up interrupts for the specified pin(s) by writing 1 to the corresponding **IC** register bit
- Enable power-up interrupt for the specified port, using **GPIO_PI_IEN** register
- The device can now be put to sleep (all power modes) and will wake up on the configured interrupt

9.2.2.3 Mode Control

The GPIO pins can be controlled by software or hardware (i.e. peripherals). Software control is the default for most signals and corresponds to the GPIO mode, where the **GPIO_DATA** register is used to read or write the corresponding pins. When hardware control is enabled by the **GPIO Alternate Function Select (GPIO_AFSEL)** register (see Section 9.1), the pin state is controlled by its alternate function (i.e., the peripheral).

Flexible pin muxing options are provided through the I/O controller module which selects one of several peripheral functions for each GPIO. For information on the configuration options, see Section 9.1.1.

NOTE: Only port A can be used as input to the ADC. If any pin on port A is to be used as an ADC input, the appropriate register, **IOC_PAx_OVER**, must be set to analog (that is, bit 0 must be set to 1).

9.2.2.4 Commit Control

The commit control registers provide a layer of protection against accidental programming of critical hardware peripherals. Writes to protected bits of the GPIO Alternate Function Select (**AFSEL**) register are not committed to storage unless the GPIO Lock (**GPIO_GPIOLOCK**) register has been unlocked and the appropriate bits of the GPIO Commit (**GPIO_GPIOCR**) register have been set to 1.

NOTE: The commit register, **GPIO_GPIOCR**, is set to all 1's at reset. This means any operations to **GPIO_AFSEL** register, by default, are permitted, **not** protected. There is no need to first unlock and program the commit register to allow writing to the **GPIO_AFSEL** register. If the application code would like to prevent accidental writing to the **GPIO_AFSEL** register, then it is first necessary to unlock the commit register, as described above, and write the desired protection bits.

9.2.2.5 Pad Control

The pad control registers let software configure the GPIO pads based on the application requirements. There is a override configuration register (**IOC_Pxx_OVER**) for each pad that can be used to drive pullup and pulldown enables for each pad and to enable the pads to drive outputs. These registers are used to set the corresponding bits for each pad when output enable (OE), pullup enable (PUE), pulldown enable (PDE), or analog (ANA) is not driven from hardware (that is, the peripheral). If the bits are set on a pad that is configured for a peripheral that does drive OE, PUE, PDE or ANA, the register is used as an override for that signal if it is set active. Figure 9-4 shows the override logic. The inputs on the left side show the various possibilities for inputs from peripherals. Any peripheral I/O can be mapped to any pad so the appropriate pad override register must be configured properly. The **IOC_Pxx_OVER** registers can also be used to control the OE, PUE, PDE, and ANA settings when the pad is configured as software-controlled GPIO.

Drive strength control for GPIO pins in output mode are controlled using the **SC** bit in the **CCTEST_IO** register. This register selects output drive strength enhancement to account for low I/O supply voltage on the DVDD pin (this to ensure the same drive strength at lower voltages as at higher). Set the **SC** bit to 0 to have minimum drive strength enhancement. Use this when DVDD is equal to or greater than 2.6 V. Set the **SC** bit to 1 to have maximum drive strength enhancement when DVDD is less than 2.6 V.

NOTE: PC0 through PC3 are bidirectional high-drive (20 mA) pad cells. They do not support on-die pullup or pulldown resistors or analog connectivity.

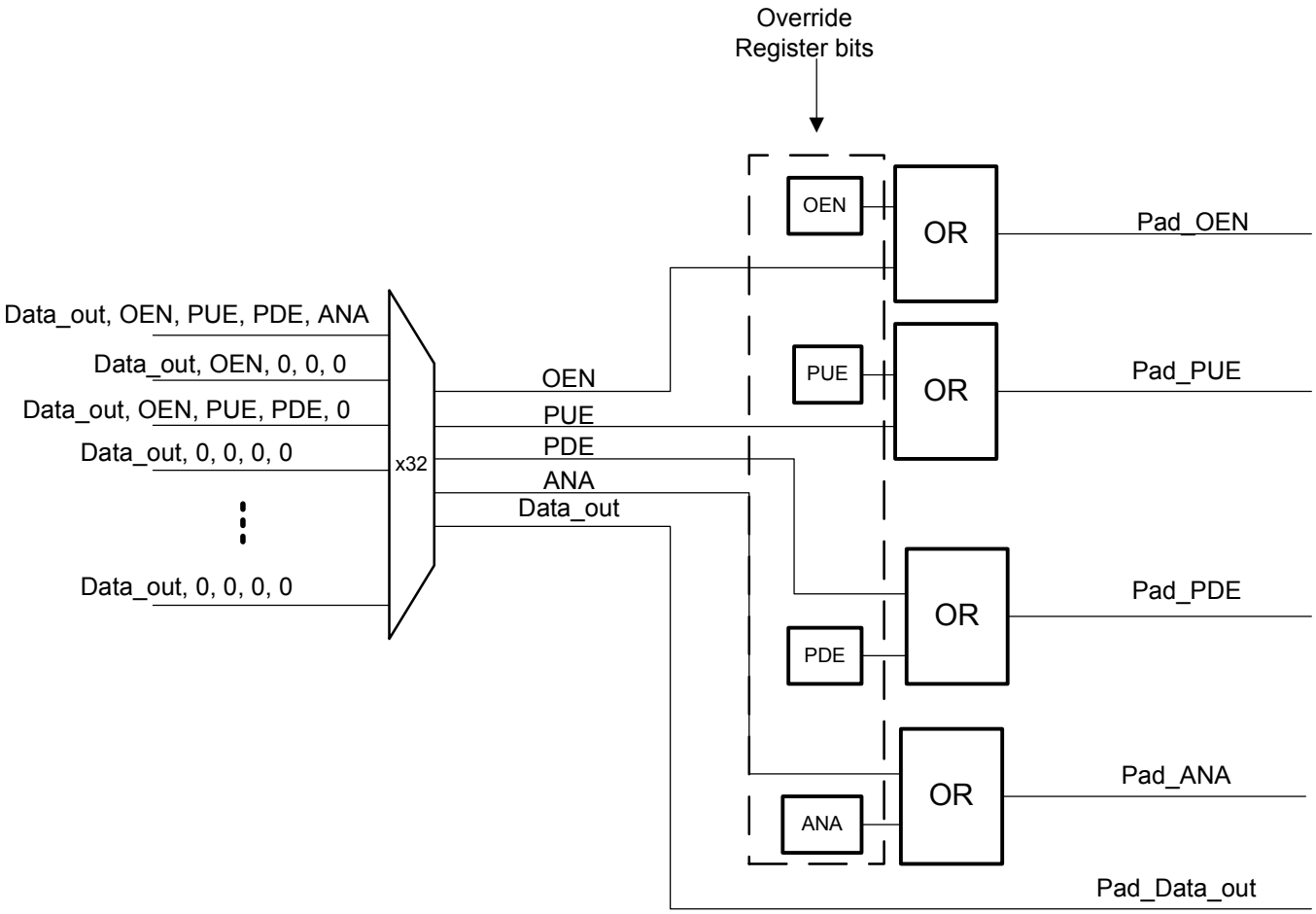


Figure 9-4. PAD Configuration Override Registers

On reset all bits in the **IOC_Pxx_OVER** registers are set to 0, except for the PUE bits. This means that at reset all GPIO pads are set as inputs and are pulled up.

9.2.3 Configuration

For example to set PA1 to be used as GPIO output write bit 1 to 1 in register Port A **GPIO_DIR** register (0x400D 9400). To make PA1 a software controlled GPIO write bit 1 to 0 in Port A **GPIO_AFSEL** register (0x400D 9420)

CAUTION

All GPIO pins are configured as input GPIOs with pullup enabled (except PC0-3) by default (**GPIO_AFSEL** = 0, **IOC_Pxx_OVER** = 0x4. A POR or asserting **RST** puts the pins back to their default state.

9.2.4 Radio Test Output Signals

By using the **CCTEST_OBSSELx** registers (**CCTEST_OBSSEL0–CCTEST_OBSSEL7**) the user can output different observation signals from the RF Core to GPIO pins. These signals can be useful for debugging of low-level protocols or control of external PA, LNA, or switches. The control registers **CCTEST_OBSSEL0–CCTEST_OBSSEL7** can be used to override the standard GPIO behavior and output RF Core signals (rfc_obs_sig0, rfc_obs_sig1, and rfc_obs_sig2) on the pins PC[0:7]. For a list of available signals, see the respective **RFCORE_XREG_RFC_OBS_CTRLx** registers in Section 23.16.2. To have the RF Core observe signal "Power Amplifier power-down signal" muxed to PC0 do the following:

- Configure PC0 as output without pull-up.
- Set **RFCORE_XREG_RFC_OBS_CTRL0** to 0x28 (Power Amplifier power-down signal)
- Set **CCTEST_OBSSEL0** (controlling PC0) to 0x80

NOTE: Writing the **CCTEST_OBSSELx.EN** bit to 1 overrides the standard configuration of the GPIO.

9.2.5 Power-Down Signal MUX (PMUX)

The **GPIO_PMUX** register can be used to output the 32-kHz clock and/or the digital regulator status. The selected 32-kHz clock source can be output on either PA0 or PB7 pins. The enable bit CKOEN enables the clock output, and the pin is selected using the CKOPIN (see the **GPIO_PMUX** register description, GPIO_PMUX, for details). When CKOEN is set, all other configurations for the selected pin are overridden. The clock is output in all power modes; however, in PM3 the clock stops. Furthermore, the digital regulator status can be output on either PB0 or PB1 pins. When the DCEN bit is set, the status of the digital regulator is output. DCPIN selects the PB pin (see the **GPIO_PMUX** register description, GPIO_PMUX, for details). When DCEN is set, all other configurations for the selected pin are overridden. The selected pin outputs 1 when the 1.8-V on-chip digital regulator is powered up (chip has regulated power). The selected pin outputs 0 when the 1.8-V on-chip digital regulator is powered down, i.e., in PM2 and PM3.

9.3 I/O Control and GPIO Registers

9.3.1 GPIO Registers

9.3.1.1 GPIO Registers Mapping Summary

This section provides information on the GPIO module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

9.3.1.1.1 GPIO Common Registers Mapping

This section provides information on the GPIO module instance within this product. Each of the registers within the Module Instance is described separately below.

Table 9-2. GPIO Common Registers Mapping Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset
GPIO_DATA	RW	32	0x0000 0000	0x000
GPIO_DIR	RW	32	0x0000 0000	0x400
GPIO_IS	RW	32	0x0000 0000	0x404
GPIO_IBE	RW	32	0x0000 0000	0x408
GPIO_IEV	RW	32	0x0000 0000	0x40C
GPIO_IE	RW	32	0x0000 0000	0x410
GPIO_RIS	RO	32	0x0000 0000	0x414
GPIO_MIS	RO	32	0x0000 0000	0x418

Table 9-2. GPIO Common Registers Mapping Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset
GPIO_IC	RW	32	0x0000 0000	0x41C
GPIO_AFSEL	RW	32	0x0000 0000	0x420
GPIO_GPIOLOCK	RW	32	0x0000 0001	0x520
GPIO_GPIOCR	RW	32	0x0000 00FF	0x524
GPIO_PMUX	RW	32	0x0000 0000	0x700
GPIO_P_EDGE_CTRL	RW	32	0x0000 0000	0x704
GPIO_PI_IEN	RW	32	0x0000 0000	0x710
GPIO_IRQ_DETECT_ACK	RW	32	0x0000 0000	0x718
GPIO_USB_IRQ_ACK	RW	32	0x0000 0000	0x71C
GPIO_IRQ_DETECT_UNMASK	RW	32	0x0000 0000	0x720

9.3.1.1.2 GPIO Instances Register Mapping Summary

9.3.1.1.2.1 GPIO_A Register Summary

Table 9-3. GPIO_A Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
GPIO_DATA	RW	32	0x0000 0000	0x000	0x400D 9000
GPIO_DIR	RW	32	0x0000 0000	0x400	0x400D 9400
GPIO_IS	RW	32	0x0000 0000	0x404	0x400D 9404
GPIO_IBE	RW	32	0x0000 0000	0x408	0x400D 9408
GPIO_IEV	RW	32	0x0000 0000	0x40C	0x400D 940C
GPIO_IE	RW	32	0x0000 0000	0x410	0x400D 9410
GPIO_RIS	RO	32	0x0000 0000	0x414	0x400D 9414
GPIO_MIS	RO	32	0x0000 0000	0x418	0x400D 9418
GPIO_IC	RW	32	0x0000 0000	0x41C	0x400D 941C
GPIO_AFSEL	RW	32	0x0000 0000	0x420	0x400D 9420
GPIO_GPIOLOCK	RW	32	0x0000 0001	0x520	0x400D 9520
GPIO_GPIOCR	RW	32	0x0000 00FF	0x524	0x400D 9524
GPIO_PMUX	RW	32	0x0000 0000	0x700	0x400D 9700
GPIO_P_EDGE_CTRL	RW	32	0x0000 0000	0x704	0x400D 9704
GPIO_PI_IEN	RW	32	0x0000 0000	0x710	0x400D 9710
GPIO_IRQ_DETECT_ACK	RW	32	0x0000 0000	0x718	0x400D 9718
GPIO_USB_IRQ_ACK	RW	32	0x0000 0000	0x71C	0x400D 971C
GPIO_IRQ_DETECT_UNMASK	RW	32	0x0000 0000	0x720	0x400D 9720

9.3.1.1.2.2 GPIO_B Register Summary

Table 9-4. GPIO_B Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
GPIO_DATA	RW	32	0x0000 0000	0x000	0x400D A000
GPIO_DIR	RW	32	0x0000 0000	0x400	0x400D A400
GPIO_IS	RW	32	0x0000 0000	0x404	0x400D A404
GPIO_IBE	RW	32	0x0000 0000	0x408	0x400D A408
GPIO_IEV	RW	32	0x0000 0000	0x40C	0x400D A40C
GPIO_IE	RW	32	0x0000 0000	0x410	0x400D A410
GPIO_RIS	RO	32	0x0000 0000	0x414	0x400D A414
GPIO_MIS	RO	32	0x0000 0000	0x418	0x400D A418
GPIO_IC	RW	32	0x0000 0000	0x41C	0x400D A41C
GPIO_AFSEL	RW	32	0x0000 0000	0x420	0x400D A420
GPIO_GPIOLOCK	RW	32	0x0000 0001	0x520	0x400D A520
GPIO_GPIOCR	RW	32	0x0000 00FF	0x524	0x400D A524
GPIO_PMUX	RW	32	0x0000 0000	0x700	0x400D A700
GPIO_P_EDGE_CTRL	RW	32	0x0000 0000	0x704	0x400D A704
GPIO_PI_IEN	RW	32	0x0000 0000	0x710	0x400D A710
GPIO_IRQ_DETECT_ACK	RW	32	0x0000 0000	0x718	0x400D A718
GPIO_USB_IRQ_ACK	RW	32	0x0000 0000	0x71C	0x400D A71C
GPIO_IRQ_DETECT_UNMASK	RW	32	0x0000 0000	0x720	0x400D A720

9.3.1.1.2.3 GPIO_C Register Summary

Table 9-5. GPIO_C Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
GPIO_DATA	RW	32	0x0000 0000	0x000	0x400D B000
GPIO_DIR	RW	32	0x0000 0000	0x400	0x400D B400
GPIO_IS	RW	32	0x0000 0000	0x404	0x400D B404
GPIO_IBE	RW	32	0x0000 0000	0x408	0x400D B408
GPIO_IEV	RW	32	0x0000 0000	0x40C	0x400D B40C
GPIO_IE	RW	32	0x0000 0000	0x410	0x400D B410
GPIO_RIS	RO	32	0x0000 0000	0x414	0x400D B414
GPIO_MIS	RO	32	0x0000 0000	0x418	0x400D B418
GPIO_IC	RW	32	0x0000 0000	0x41C	0x400D B41C
GPIO_AFSEL	RW	32	0x0000 0000	0x420	0x400D B420
GPIO_GPIOLOCK	RW	32	0x0000 0001	0x520	0x400D B520
GPIO_GPIOCR	RW	32	0x0000 00FF	0x524	0x400D B524
GPIO_PMUX	RW	32	0x0000 0000	0x700	0x400D B700

Table 9-5. GPIO_C Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
GPIO_P_EDGE_CTRL	RW	32	0x0000 0000	0x704	0x400D B704
GPIO_PI_IEN	RW	32	0x0000 0000	0x710	0x400D B710
GPIO_IRQ_DETECT_ACK	RW	32	0x0000 0000	0x718	0x400D B718
GPIO_USB_IRQ_ACK	RW	32	0x0000 0000	0x71C	0x400D B71C
GPIO_IRQ_DETECT_UNMASK	RW	32	0x0000 0000	0x720	0x400D B720

9.3.1.1.2.4 GPIO_D Register Summary

Table 9-6. GPIO_D Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
GPIO_DATA	RW	32	0x0000 0000	0x000	0x400D C000
GPIO_DIR	RW	32	0x0000 0000	0x400	0x400D C400
GPIO_IS	RW	32	0x0000 0000	0x404	0x400D C404
GPIO_IBE	RW	32	0x0000 0000	0x408	0x400D C408
GPIO_IEV	RW	32	0x0000 0000	0x40C	0x400D C40C
GPIO_IE	RW	32	0x0000 0000	0x410	0x400D C410
GPIO_RIS	RO	32	0x0000 0000	0x414	0x400D C414
GPIO_MIS	RO	32	0x0000 0000	0x418	0x400D C418
GPIO_IC	RW	32	0x0000 0000	0x41C	0x400D C41C
GPIO_AFSEL	RW	32	0x0000 0000	0x420	0x400D C420
GPIO_GPIOLOCK	RW	32	0x0000 0001	0x520	0x400D C520
GPIO_GPIOCR	RW	32	0x0000 00FF	0x524	0x400D C524
GPIO_PMUX	RW	32	0x0000 0000	0x700	0x400D C700
GPIO_P_EDGE_CTRL	RW	32	0x0000 0000	0x704	0x400D C704
GPIO_PI_IEN	RW	32	0x0000 0000	0x710	0x400D C710
GPIO_IRQ_DETECT_ACK	RW	32	0x0000 0000	0x718	0x400D C718
GPIO_USB_IRQ_ACK	RW	32	0x0000 0000	0x71C	0x400D C71C
GPIO_IRQ_DETECT_UNMASK	RW	32	0x0000 0000	0x720	0x400D C720

9.3.1.2 GPIO Common Register Descriptions

GPIO_DATA

Address offset	0x000		
Physical Address	0x400D C000 0x400D A000 0x400D 9000 0x400D B000	**Instance**	GPIO_D GPIO_B GPIO_A GPIO_C
Description	This is the data register. In software control mode, values written in the GPIODATA register are transferred onto the GPOUT pins if the respective pins have been configured as outputs through the GPIODIR register. A read from GPIODATA returns the last bit value written if the respective pins are configured as output, or it returns the value on the corresponding input GPIN bit when these are configured as inputs.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	DATA	Input and output data	RW	0x00

GPIO_DIR

Address offset	0x400		
Physical Address	0x400D C400 0x400D A400 0x400D 9400 0x400D B400	**Instance**	GPIO_D GPIO_B GPIO_A GPIO_C
Description	The DIR register is the data direction register. All bits are cleared by a reset; therefore, the GPIO pins are input by default.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	DIR	Bits set: Corresponding pin is output Bits cleared: Corresponding pin is input	RW	0x00

GPIO_IS

Address offset	0x404		
Physical Address	0x400D C404 0x400D A404 0x400D 9404 0x400D B404	**Instance**	GPIO_D GPIO_B GPIO_A GPIO_C
Description	The IS register is the interrupt sense register.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	IS	Bits set: Level on corresponding pin is detected Bits cleared: Edge on corresponding pin is detected	RW	0x00

GPIO_IBE

Address offset	0x408		
Physical Address	0x400D C408 0x400D A408 0x400D 9408 0x400D B408	**Instance**	GPIO_D GPIO_B GPIO_A GPIO_C
Description	The IBE register is the interrupt both-edges register. When the corresponding bit in IS is set to detect edges, bits set to high in IBE configure the corresponding pin to detect both rising and falling edges, regardless of the corresponding bit in the IEV (interrupt event register). Clearing a bit configures the pin to be controlled by IEV.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	IBE	Bits set: Both edges on corresponding pin trigger an interrupt Bits cleared: Interrupt generation event is controlled by GPIOIEV Single edge: Determined by corresponding bit in GPIOIEV register	RW	0x00

GPIO_IEV

Address offset	0x40C		
Physical Address	0x400D C40C 0x400D A40C 0x400D 940C 0x400D B40C	**Instance**	GPIO_D GPIO_B GPIO_A GPIO_C
Description	The IEV register is the interrupt event register. Bits set to high in IEV configure the corresponding pin to detect rising edges or high levels, depending on the corresponding bit value in IS. Clearing a bit configures the pin to detect falling edges or low levels, depending on the corresponding bit value in IS.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	IEV	Bits set: Rising edges or high levels on corresponding pin trigger interrupts Bits cleared: Falling edges or low levels on corresponding pin trigger interrupts	RW	0x00

GPIO_IE

Address offset	0x410		
Physical Address	0x400D C410 0x400D A410 0x400D 9410 0x400D B410	**Instance**	GPIO_D GPIO_B GPIO_A GPIO_C
Description	The IE register is the interrupt mask register. Bits set to high in IE allow the corresponding pins to trigger their individual interrupts and the combined GPIOINTR line. Clearing a bit disables interrupt triggering on that pin.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	IE	Bits set: Corresponding pin is not masked Bits cleared: Corresponding pin is masked	RW	0x00

GPIO_RIS

Address offset 0x414

Physical Address 0x400D C414 **Instance** GPIO_D
0x400D A414 GPIO_B
0x400D 9414 GPIO_A
0x400D B414 GPIO_C

Description The RIS register is the raw interrupt status register. Bits read high in RIS reflect the status of interrupts trigger conditions detected (raw, before masking), indicating that all the requirements are met, before they are finally allowed to trigger by IE. Bits read as 0 indicate that corresponding input pins have not initiated an interrupt.

Type RO

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	RIS	Reflects the status of interrupts trigger conditions detected on pins (raw, before masking) Bits set: Requirements met by corresponding pins Bits clear: Requirements not met	RO	0x00

GPIO_MIS

Address offset 0x418

Physical Address 0x400D C418 **Instance** GPIO_D
0x400D A418 GPIO_B
0x400D 9418 GPIO_A
0x400D B418 GPIO_C

Description The MIS register is the masked interrupt status register. Bits read high in MIS reflect the status of input lines triggering an interrupt. Bits read as low indicate that either no interrupt has been generated, or the interrupt is masked. MIS is the state of the interrupt after masking.

Type RO

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	MIS	Masked value of interrupt due to corresponding pin Bits clear: GPIO line interrupt not active Bits set: GPIO line asserting interrupt	RO	0x00

GPIO_IC

Address offset 0x41C

Physical Address 0x400D C41C **Instance** GPIO_D
0x400D A41C GPIO_B
0x400D 941C GPIO_A
0x400D B41C GPIO_C

Description The IC register is the interrupt clear register. Writing 1 to a bit in this register clears the corresponding interrupt edge detection logic register. Writing 0 has no effect.

Type RW

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	IC	Bit written as 1: Clears edge detection logic Bit written as 0: Has no effect	WO	0x00

GPIO_AFSEL

Address offset	0x420		
Physical Address	0x400D C420 0x400D A420 0x400D 9420 0x400D B420	**Instance**	GPIO_D GPIO_B GPIO_A GPIO_C
Description	The AFSEL register is the mode control select register. Writing 1 to any bit in this register selects the hardware (peripheral) control for the corresponding GPIO line. All bits are cleared by a reset, therefore no GPIO line is set to hardware control by default.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	AFSEL	Bit set: Enables hardware (peripheral) control mode Bit cleared: Enables software control mode	RW	0x00

GPIO_GPIOLOCK

Address offset	0x520		
Physical Address	0x400D C520 0x400D A520 0x400D 9520 0x400D B520	**Instance**	GPIO_D GPIO_B GPIO_A GPIO_C
Description	A write of the value 0x4C4F434B to the GPIOLOCK register unlocks the GPIO commit register (GPIOCR) for write access. A write of any other value reapplies the lock, preventing any register updates. Any write to the commit register (GPIOCR) causes the lock register to be locked.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	LOCK	A read of this register returns the following values: Locked: 0x00000001 Unlocked: 0x00000000	RW	0x0000 0001

GPIO_GPIOCR

Address offset	0x524		
Physical Address	0x400D C524 0x400D A524 0x400D 9524 0x400D B524	**Instance**	GPIO_D GPIO_B GPIO_A GPIO_C
Description	The GPIOCR register is the commit register. The value of the GPIOCR register determines which bits of the AFSEL register is committed when a write to the AFSEL register is performed. If a bit in the GPIOCR register is 0, the data being written to the corresponding bit in the AFSEL register is not committed and retains its previous value. If a bit in the GPIOCR register is set to 1, the data being written to the corresponding bit of the AFSEL register is committed to the register and will reflect the new value. The contents of the GPIOCR register can only be modified if the GPIOLOCK register is unlocked. Writes to the GPIOCR register will be ignored if the GPIOLOCK register is locked. Any write to the commit register causes the lock register to be locked.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	CR	On a bit-wise basis, any bit set allows the corresponding GPIOAFSEL bit to be set to its alternate function.	RW	0xFF

GPIO_PMUX

Address offset	0x700		
Physical Address	0x400D C700 0x400D A700 0x400D 9700 0x400D B700	**Instance**	GPIO_D GPIO_B GPIO_A GPIO_C
Description	The PMUX register can be used to output external decouple control and clock_32k on I/O pins. Decouple control can be output on specific PB pins and clock_32k can be output on a specific PA or PB pin. These features override the current setting of the selected pin when enabled. The pin is set to output, pull-up and -down disabled, and analog mode disabled.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								CKOEN	RESERVED		CKOPIN	DCEN	RESERVED		DCPIN

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	CKOEN	Clock out enable When this bit is set, the 32-kHz clock is routed to either PA[0] or PB[7] pins. PMUX.CKOPIN selects the pin to use. This overrides the current configuration setting for this pin. The pullup or pulldown is disabled and the direction is set to output for this pin.	RW	0
6:5	RESERVED	This bit field is reserved.	RO	0x0
4	CKOPIN	Decouple control pin select This control only has relevance when CKOEN is set. When 0, PA[0] becomes the 32-kHz clock output. When 1, PB[7] becomes the 32-kHz clock output.	RW	0
3	DCEN	Decouple control enable When this bit is set, the on-die digital regulator status is routed to either PB[1] or PB[0] pins. PMUX.DCPIN selects the pin to use. This overrides the current configuration setting for this pin. The pullup or pulldown is disabled and the direction is set to output for this pin.	RW	0

Bits	Field Name	Description	Type	Reset
2:1	RESERVED	This bit field is reserved.	RO	0x0
0	DCPIN	Decouple control pin select This control has relevance only when DCEN is set. When 0, PB[1] becomes the on-die digital regulator status (1 indicates the on-die digital regulator is active); when 1, PB[0] becomes the on-die digital regulator status. NOTE: PB[1] and PB[0] can also be controlled with other override features. In priority order for PB[1]: When POR/BOD test mode is active, PB[1] becomes the active low brown-out detected indicator. When DCEN is set and DCPIN is not set, PB[1] becomes the on-dir digital regulator status. In priority order for PB[0]: When POR/BOD test mode is active, PB[0] becomes the power-on-reset indicator. When DCEN and DCPIN are set, PB[0] becomes the on-die digital regulator status.	RW	0

GPIO_P_EDGE_CTRL

Address offset	0x704		
Physical Address	0x400D C704 0x400D A704 0x400D 9704 0x400D B704	**Instance**	GPIO_D GPIO_B GPIO_A GPIO_C
Description	The port edge control register is used to control which edge of each port input causes that port to generate a power-up interrupt to the system.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PDIRC7	PDIRC6	PDIRC5	PDIRC4	PDIRC3	PDIRC2	PDIRC1	PDIRC0	PCIRC7	PCIRC6	PCIRC5	PCIRC4	PCIRC3	PCIRC2	PCIRC1	PCIRC0	PBIRC7	PBIRC6	PBIRC5	PBIRC4	PBIRC3	PBIRC2	PBIRC1	PBIRC0	PAIRC7	PAIRC6	PAIRC5	PAIRC4	PAIRC3	PAIRC2	PAIRC1	PAIRC0

Bits	Field Name	Description	Type	Reset
31	PDIRC7	Port D bit 7 interrupt request condition: 0: Rising 1: Falling edge	RW	0
30	PDIRC6	Port D bit 6 interrupt request condition: 0: Rising 1: Falling edge	RW	0
29	PDIRC5	Port D bit 5 interrupt request condition: 0: Rising 1: Falling edge	RW	0
28	PDIRC4	Port D bit 4 interrupt request condition: 0: Rising 1: Falling edge	RW	0
27	PDIRC3	Port D bit 3 interrupt request condition: 0: Rising 1: Falling edge	RW	0
26	PDIRC2	Port D bit 2 interrupt request condition: 0: Rising 1: Falling edge	RW	0
25	PDIRC1	Port D bit 1 interrupt request condition: 0: Rising 1: Falling edge	RW	0
24	PDIRC0	Port D bit 0 interrupt request condition: 0: Rising 1: Falling edge	RW	0

Bits	Field Name	Description	Type	Reset
23	PCIRC7	Port C bit 7 interrupt request condition: 0: Rising 1: Falling edge	RW	0
22	PCIRC6	Port C bit 6 interrupt request condition: 0: Rising 1: Falling edge	RW	0
21	PCIRC5	Port C bit 5 interrupt request condition: 0: Rising 1: Falling edge	RW	0
20	PCIRC4	Port C bit 4 interrupt request condition: 0: Rising 1: Falling edge	RW	0
19	PCIRC3	Port C bit 3 interrupt request condition: 0: Rising 1: Falling edge	RW	0
18	PCIRC2	Port C bit 2 interrupt request condition: 0: Rising 1: Falling edge	RW	0
17	PCIRC1	Port C bit 1 interrupt request condition: 0: Rising 1: Falling edge	RW	0
16	PCIRC0	Port C bit 0 interrupt request condition: 0: Rising 1: Falling edge	RW	0
15	PBIRC7	Port B bit 7 interrupt request condition: 0: Rising 1: Falling edge	RW	0
14	PBIRC6	Port B bit 6 interrupt request condition: 0: Rising 1: Falling edge	RW	0
13	PBIRC5	Port B bit 5 interrupt request condition: 0: Rising 1: Falling edge	RW	0
12	PBIRC4	Port B bit 4 interrupt request condition: 0: Rising 1: Falling edge	RW	0
11	PBIRC3	Port B bit 3 interrupt request condition: 0: Rising 1: Falling edge	RW	0
10	PBIRC2	Port B bit 2 interrupt request condition: 0: Rising 1: Falling edge	RW	0
9	PBIRC1	Port B bit 1 interrupt request condition: 0: Rising 1: Falling edge	RW	0
8	PBIRC0	Port B bit 0 interrupt request condition: 0: Rising 1: Falling edge	RW	0
7	PAIRC7	Port A bit 7 interrupt request condition: 0: Rising 1: Falling edge	RW	0
6	PAIRC6	Port A bit 6 interrupt request condition: 0: Rising 1: Falling edge	RW	0
5	PAIRC5	Port A bit 5 interrupt request condition: 0: Rising 1: Falling edge	RW	0
4	PAIRC4	Port A bit 4 interrupt request condition: 0: Rising 1: Falling edge	RW	0

Bits	Field Name	Description	Type	Reset
3	PAIRC3	Port A bit 3 interrupt request condition: 0: Rising 1: Falling edge	RW	0
2	PAIRC2	Port A bit 2 interrupt request condition: 0: Rising 1: Falling edge	RW	0
1	PAIRC1	Port A bit 1 interrupt request condition: 0: Rising 1: Falling edge	RW	0
0	PAIRC0	Port A bit 0 interrupt request condition: 0: Rising 1: Falling edge	RW	0

GPIO_PI_IEN

Address offset	0x710		
Physical Address	0x400D C710 0x400D A710 0x400D 9710 0x400D B710	**Instance**	GPIO_D GPIO_B GPIO_A GPIO_C
Description	The power-up interrupt enable register selects, for its corresponding port A-D pin, whether interrupts are enabled or disabled.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PDIEN7	PDIEN6	PDIEN5	PDIEN4	PDIEN3	PDIEN2	PDIEN1	PDIEN0	PCIEN7	PCIEN6	PCIEN5	PCIEN4	PCIEN3	PCIEN2	PCIEN1	PCIEN0	PBIEN7	PBIEN6	PBIEN5	PBIEN4	PBIEN3	PBIEN2	PBIEN1	PBIEN0	PAIEN7	PAIEN6	PAIEN5	PAIEN4	PAIEN3	PAIEN2	PAIEN1	PAIEN0

Bits	Field Name	Description	Type	Reset
31	PDIEN7	Port D bit 7 interrupt enable: 1: Enabled 2: Disabled	RW	0
30	PDIEN6	Port D bit 6 interrupt enable: 1: Enabled 2: Disabled	RW	0
29	PDIEN5	Port D bit 5 interrupt enable: 1: Enabled 2: Disabled	RW	0
28	PDIEN4	Port D bit 4 interrupt enable: 1: Enabled 2: Disabled	RW	0
27	PDIEN3	Port D bit 3 interrupt enable: 1: Enabled 2: Disabled	RW	0
26	PDIEN2	Port D bit 2 interrupt enable: 1: Enabled 2: Disabled	RW	0
25	PDIEN1	Port D bit 1 interrupt enable: 1: Enabled 2: Disabled	RW	0
24	PDIEN0	Port D bit 0 interrupt enable: 1: Enabled 2: Disabled	RW	0
23	PCIEN7	Port C bit 7 interrupt enable: 1: Enabled 2: Disabled	RW	0
22	PCIEN6	Port C bit 6 interrupt enable: 1: Enabled 2: Disabled	RW	0

Bits	Field Name	Description	Type	Reset
21	PCIEN5	Port C bit 5 interrupt enable: 1: Enabled 2: Disabled	RW	0
20	PCIEN4	Port C bit 4 interrupt enable: 1: Enabled 2: Disabled	RW	0
19	PCIEN3	Port C bit 3 interrupt enable: 1: Enabled 2: Disabled	RW	0
18	PCIEN2	Port C bit 2 interrupt enable: 1: Enabled 2: Disabled	RW	0
17	PCIEN1	Port C bit 1 interrupt enable: 1: Enabled 2: Disabled	RW	0
16	PCIEN0	Port C bit 0 interrupt enable: 1: Enabled 2: Disabled	RW	0
15	PBIEN7	Port B bit 7 interrupt enable: 1: Enabled 2: Disabled	RW	0
14	PBIEN6	Port B bit 6 interrupt enable: 1: Enabled 2: Disabled	RW	0
13	PBIEN5	Port B bit 5 interrupt enable: 1: Enabled 2: Disabled	RW	0
12	PBIEN4	Port B bit 4 interrupt enable: 1: Enabled 2: Disabled	RW	0
11	PBIEN3	Port B bit 3 interrupt enable: 1: Enabled 2: Disabled	RW	0
10	PBIEN2	Port B bit 2 interrupt enable: 1: Enabled 2: Disabled	RW	0
9	PBIEN1	Port B bit 1 interrupt enable: 1: Enabled 2: Disabled	RW	0
8	PBIEN0	Port B bit 0 interrupt enable: 1: Enabled 2: Disabled	RW	0
7	PAIEN7	Port A bit 7 interrupt enable: 1: Enabled 2: Disabled	RW	0
6	PAIEN6	Port A bit 6 interrupt enable: 1: Enabled 2: Disabled	RW	0
5	PAIEN5	Port A bit 5 interrupt enable: 1: Enabled 2: Disabled	RW	0
4	PAIEN4	Port A bit 4 interrupt enable: 1: Enabled 2: Disabled	RW	0
3	PAIEN3	Port A bit 3 interrupt enable: 1: Enabled 2: Disabled	RW	0
2	PAIEN2	Port A bit 2 interrupt enable: 1: Enabled 2: Disabled	RW	0

Bits	Field Name	Description	Type	Reset
1	PAIEN1	Port A bit 1 interrupt enable: 1: Enabled 2: Disabled	RW	0
0	PAIEN0	Port A bit 0 interrupt enable: 1: Enabled 2: Disabled	RW	0

GPIO_IRQ_DETECT_ACK

Address offset	0x718		
Physical Address	0x400D C718 0x400D A718 0x400D 9718 0x400D B718	**Instance**	GPIO_D GPIO_B GPIO_A GPIO_C
Description	If the IRQ detect ACK register is read, the value returned can be used to determine which enabled I/O port is responsible for creating a power-up interrupt to the system. Writing the IRQ detect ACK register is used to clear any number of individual port bits that may be signaling that an edge was detected as configured by the port edge control register and the interrupt control register. There is a self-clearing function to this register that generates a reset pulse to clear any interrupt which has its corresponding bit set to 1.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PDIACK7	PDIACK6	PDIACK5	PDIACK4	PDIACK3	PDIACK2	PDIACK1	PDIACK0	PCIACK7	PCIACK6	PCIACK5	PCIACK4	PCIACK3	PCIACK2	PCIACK1	PCIACK0	PBIACK7	PBIACK6	PBIACK5	PBIACK4	PBIACK3	PBIACK2	PBIACK1	PBIACK0	PAIACK7	PAIACK6	PAIACK5	PAIACK4	PAIACK3	PAIACK2	PAIACK1	PAIACK0

Bits	Field Name	Description	Type	Reset
31	PDIACK7	Port D bit 7 masked interrupt status: 1: Detected 0: Not detected	RW	0
30	PDIACK6	Port D bit 6 masked interrupt status: 1: Detected 0: Not detected	RW	0
29	PDIACK5	Port D bit 5 masked interrupt status: 1: Detected 0: Not detected	RW	0
28	PDIACK4	Port D bit 4 masked interrupt status: 1: Detected 0: Not detected	RW	0
27	PDIACK3	Port D bit 3 masked interrupt status: 1: Detected 0: Not detected	RW	0
26	PDIACK2	Port D bit 2 masked interrupt status: 1: Detected 0: Not detected	RW	0
25	PDIACK1	Port D bit 1 masked interrupt status: 1: Detected0: Not detected	RW	0
24	PDIACK0	Port D bit 0 masked interrupt status: 1: Detected 0: Not detected	RW	0
23	PCIACK7	Port C bit 7 masked interrupt status: 1: Detected 0: Not detected	RW	0
22	PCIACK6	Port C bit 6 masked interrupt status: 1: Detected 0: Not detected	RW	0
21	PCIACK5	Port C bit 5 masked interrupt status: 1: Detected 0: Not detected	RW	0

Bits	Field Name	Description	Type	Reset
20	PCIACK4	Port C bit 4 masked interrupt status: 1: Detected 0: Not detected	RW	0
19	PCIACK3	Port C bit 3 masked interrupt status: 1: Detected 0: Not detected	RW	0
18	PCIACK2	Port C bit 2 masked interrupt status: 1: Detected 0: Not detected	RW	0
17	PCIACK1	Port C bit 1 masked interrupt status: 1: Detected 0: Not detected	RW	0
16	PCIACK0	Port C bit 0 masked interrupt status: 1: Detected 0: Not detected	RW	0
15	PBIACK7	Port B bit 7 masked interrupt status: 1: Detected 0: Not detected	RW	0
14	PBIACK6	Port B bit 6 masked interrupt status: 1: Detected 0: Not detected	RW	0
13	PBIACK5	Port B bit 5 masked interrupt status: 1: Detected 0: Not detected	RW	0
12	PBIACK4	Port B bit 4 masked interrupt status: 1: Detected 0: Not detected	RW	0
11	PBIACK3	Port B bit 3 masked interrupt status: 1: Detected 0: Not detected	RW	0
10	PBIACK2	Port B bit 2 masked interrupt status: 1: Detected 0: Not detected	RW	0
9	PBIACK1	Port B bit 1 masked interrupt status: 1: Detected 0: Not detected	RW	0
8	PBIACK0	Port B bit 0 masked interrupt status: 1: Detected 0: Not detected	RW	0
7	PAIACK7	Port A bit 7 masked interrupt status: 1: Detected 0: Not detected	RW	0
6	PAIACK6	Port A bit 6 masked interrupt status: 1: Detected 0: Not detected	RW	0
5	PAIACK5	Port A bit 5 masked interrupt status: 1: Detected 0: Not detected	RW	0
4	PAIACK4	Port A bit 4 masked interrupt status: 1: Detected 0: Not detected	RW	0
3	PAIACK3	Port A bit 3 masked interrupt status: 1: Detected 0: Not detected	RW	0
2	PAIACK2	Port A bit 2 masked interrupt status: 1: Detected 0: Not detected	RW	0
1	PAIACK1	Port A bit 1 masked interrupt status: 1: Detected 0: Not detected	RW	0

Bits	Field Name	Description	Type	Reset
0	PAIACK0	Port A bit 0 masked interrupt status: 1: Detected 0: Not detected	RW	0

GPIO_USB_IRQ_ACK

Address offset	0x71C		
Physical Address	0x400D C71C 0x400D A71C 0x400D 971C 0x400D B71C	**Instance**	GPIO_D GPIO_B GPIO_A GPIO_C
Description	Same functionality as IRQ_DETECT_ACK, but for USB		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	RO	0x0000 0000
0	USBACK	USB masked interrupt status: 1: Detected 0: Not detected	RW	0

GPIO_IRQ_DETECT_UNMASK

Address offset	0x720		
Physical Address	0x400D C720 0x400D A720 0x400D 9720 0x400D B720	**Instance**	GPIO_D GPIO_B GPIO_A GPIO_C
Description	Same functionality as IRQ_DETECT_ACK, but this register handles masked interrupts		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PDIACK7	PDIACK6	PDIACK5	PDIACK4	PDIACK3	PDIACK2	PDIACK1	PDIACK0	PCIACK7	PCIACK6	PCIACK5	PCIACK4	PCIACK3	PCIACK2	PCIACK1	PCIACK0	PBIACK7	PBIACK6	PBIACK5	PBIACK4	PBIACK3	PBIACK2	PBIACK1	PBIACK0	PAIACK7	PAIACK6	PAIACK5	PAIACK4	PAIACK3	PAIACK2	PAIACK1	PAIACK0

Bits	Field Name	Description	Type	Reset
31	PDIACK7	Port D bit 7 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
30	PDIACK6	Port D bit 6 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
29	PDIACK5	Port D bit 5 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
28	PDIACK4	Port D bit 4 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
27	PDIACK3	Port D bit 3 unmasked interrupt status: 1: Detected 0: Undetected	RW	0

Bits	Field Name	Description	Type	Reset
26	PDIACK2	Port D bit 2 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
25	PDIACK1	Port D bit 1 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
24	PDIACK0	Port D bit 0 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
23	PCIACK7	Port C bit 7 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
22	PCIACK6	Port C bit 6 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
21	PCIACK5	Port C bit 5 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
20	PCIACK4	Port C bit 4 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
19	PCIACK3	Port C bit 3 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
18	PCIACK2	Port C bit 2 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
17	PCIACK1	Port C bit 1 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
16	PCIACK0	Port C bit 0 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
15	PBIACK7	Port B bit 7 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
14	PBIACK6	Port B bit 6 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
13	PBIACK5	Port B bit 5 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
12	PBIACK4	Port B bit 4 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
11	PBIACK3	Port B bit 3 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
10	PBIACK2	Port B bit 2 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
9	PBIACK1	Port B bit 1 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
8	PBIACK0	Port B bit 0 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
7	PAIACK7	Port A bit 7 unmasked interrupt status: 1: Detected 0: Undetected	RW	0

Bits	Field Name	Description	Type	Reset
6	PAIACK6	Port A bit 6 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
5	PAIACK5	Port A bit 5 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
4	PAIACK4	Port A bit 4 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
3	PAIACK3	Port A bit 3 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
2	PAIACK2	Port A bit 2 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
1	PAIACK1	Port A bit 1 unmasked interrupt status: 1: Detected 0: Undetected	RW	0
0	PAIACK0	Port A bit 0 unmasked interrupt status: 1: Detected 0: Undetected	RW	0

9.3.2 IOC Registers

9.3.2.1 IOC Registers Mapping Summary

This section provides information on the IOC module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 9-7. IOC Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
IOC_PA0_SEL	RW	32	0x0000 0000	0x000	0x400D 4000
IOC_PA1_SEL	RW	32	0x0000 0000	0x004	0x400D 4004
IOC_PA2_SEL	RW	32	0x0000 0000	0x008	0x400D 4008
IOC_PA3_SEL	RW	32	0x0000 0000	0x00C	0x400D 400C
IOC_PA4_SEL	RW	32	0x0000 0000	0x010	0x400D 4010
IOC_PA5_SEL	RW	32	0x0000 0000	0x014	0x400D 4014
IOC_PA6_SEL	RW	32	0x0000 0000	0x018	0x400D 4018
IOC_PA7_SEL	RW	32	0x0000 0000	0x01C	0x400D 401C
IOC_PB0_SEL	RW	32	0x0000 0000	0x020	0x400D 4020
IOC_PB1_SEL	RW	32	0x0000 0000	0x024	0x400D 4024
IOC_PB2_SEL	RW	32	0x0000 0000	0x028	0x400D 4028
IOC_PB3_SEL	RW	32	0x0000 0000	0x02C	0x400D 402C
IOC_PB4_SEL	RW	32	0x0000 0000	0x030	0x400D 4030
IOC_PB5_SEL	RW	32	0x0000 0000	0x034	0x400D 4034
IOC_PB6_SEL	RW	32	0x0000 0000	0x038	0x400D 4038
IOC_PB7_SEL	RW	32	0x0000 0000	0x03C	0x400D 403C
IOC_PC0_SEL	RW	32	0x0000 0000	0x040	0x400D 4040
IOC_PC1_SEL	RW	32	0x0000 0000	0x044	0x400D 4044

Table 9-7. IOC Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
IOC_PC2_SEL	RW	32	0x0000 0000	0x048	0x400D 4048
IOC_PC3_SEL	RW	32	0x0000 0000	0x04C	0x400D 404C
IOC_PC4_SEL	RW	32	0x0000 0000	0x050	0x400D 4050
IOC_PC5_SEL	RW	32	0x0000 0000	0x054	0x400D 4054
IOC_PC6_SEL	RW	32	0x0000 0000	0x058	0x400D 4058
IOC_PC7_SEL	RW	32	0x0000 0000	0x05C	0x400D 405C
IOC_PD0_SEL	RW	32	0x0000 0000	0x060	0x400D 4060
IOC_PD1_SEL	RW	32	0x0000 0000	0x064	0x400D 4064
IOC_PD2_SEL	RW	32	0x0000 0000	0x068	0x400D 4068
IOC_PD3_SEL	RW	32	0x0000 0000	0x06C	0x400D 406C
IOC_PD4_SEL	RW	32	0x0000 0000	0x070	0x400D 4070
IOC_PD5_SEL	RW	32	0x0000 0000	0x074	0x400D 4074
IOC_PD6_SEL	RW	32	0x0000 0000	0x078	0x400D 4078
IOC_PD7_SEL	RW	32	0x0000 0000	0x07C	0x400D 407C
IOC_PA0_OVER	RW	32	0x0000 0004	0x080	0x400D 4080
IOC_PA1_OVER	RW	32	0x0000 0004	0x084	0x400D 4084
IOC_PA2_OVER	RW	32	0x0000 0004	0x088	0x400D 4088
IOC_PA3_OVER	RW	32	0x0000 0004	0x08C	0x400D 408C
IOC_PA4_OVER	RW	32	0x0000 0004	0x090	0x400D 4090
IOC_PA5_OVER	RW	32	0x0000 0004	0x094	0x400D 4094
IOC_PA6_OVER	RW	32	0x0000 0004	0x098	0x400D 4098
IOC_PA7_OVER	RW	32	0x0000 0004	0x09C	0x400D 409C
IOC_PB0_OVER	RW	32	0x0000 0004	0x0A0	0x400D 40A0
IOC_PB1_OVER	RW	32	0x0000 0004	0x0A4	0x400D 40A4
IOC_PB2_OVER	RW	32	0x0000 0004	0x0A8	0x400D 40A8
IOC_PB3_OVER	RW	32	0x0000 0004	0x0AC	0x400D 40AC
IOC_PB4_OVER	RW	32	0x0000 0004	0x0B0	0x400D 40B0
IOC_PB5_OVER	RW	32	0x0000 0004	0x0B4	0x400D 40B4
IOC_PB6_OVER	RW	32	0x0000 0004	0x0B8	0x400D 40B8
IOC_PB7_OVER	RW	32	0x0000 0004	0x0BC	0x400D 40BC
IOC_PC0_OVER	RW	32	0x0000 0004	0x0C0	0x400D 40C0
IOC_PC1_OVER	RW	32	0x0000 0004	0x0C4	0x400D 40C4
IOC_PC2_OVER	RW	32	0x0000 0004	0x0C8	0x400D 40C8
IOC_PC3_OVER	RW	32	0x0000 0004	0x0CC	0x400D 40CC
IOC_PC4_OVER	RW	32	0x0000 0004	0x0D0	0x400D 40D0
IOC_PC5_OVER	RW	32	0x0000 0004	0x0D4	0x400D 40D4
IOC_PC6_OVER	RW	32	0x0000 0004	0x0D8	0x400D 40D8
IOC_PC7_OVER	RW	32	0x0000 0004	0x0DC	0x400D 40DC
IOC_PD0_OVER	RW	32	0x0000 0004	0x0E0	0x400D 40E0
IOC_PD1_OVER	RW	32	0x0000 0004	0x0E4	0x400D 40E4

Table 9-7. IOC Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
IOC_PD2_OVER	RW	32	0x0000 0004	0x0E8	0x400D 40E8
IOC_PD3_OVER	RW	32	0x0000 0004	0x0EC	0x400D 40EC
IOC_PD4_OVER	RW	32	0x0000 0004	0x0F0	0x400D 40F0
IOC_PD5_OVER	RW	32	0x0000 0004	0x0F4	0x400D 40F4
IOC_PD6_OVER	RW	32	0x0000 0004	0x0F8	0x400D 40F8
IOC_PD7_OVER	RW	32	0x0000 0004	0x0FC	0x400D 40FC
IOC_UARTRXD_UART0	RW	32	0x0000 0000	0x100	0x400D 4100
IOC_UARTCTS_UART1	RW	32	0x0000 0000	0x104	0x400D 4104
IOC_UARTRXD_UART1	RW	32	0x0000 0000	0x108	0x400D 4108
IOC_CLK_SSI_SSI0	RW	32	0x0000 0000	0x10C	0x400D 410C
IOC_SSIRXD_SSI0	RW	32	0x0000 0000	0x110	0x400D 4110
IOC_SSIFSSIN_SSI0	RW	32	0x0000 0000	0x114	0x400D 4114
IOC_CLK_SSIIN_SSI0	RW	32	0x0000 0000	0x118	0x400D 4118
IOC_CLK_SSI_SSI1	RW	32	0x0000 0000	0x11C	0x400D 411C
IOC_SSIRXD_SSI1	RW	32	0x0000 0000	0x120	0x400D 4120
IOC_SSIFSSIN_SSI1	RW	32	0x0000 0000	0x124	0x400D 4124
IOC_CLK_SSIIN_SSI1	RW	32	0x0000 0000	0x128	0x400D 4128
IOC_I2CMSSDA	RW	32	0x0000 0000	0x12C	0x400D 412C
IOC_I2CMSSCL	RW	32	0x0000 0000	0x130	0x400D 4130
IOC_GPT0OCP1	RW	32	0x0000 0000	0x134	0x400D 4134
IOC_GPT0OCP2	RW	32	0x0000 0000	0x138	0x400D 4138
IOC_GPT1OCP1	RW	32	0x0000 0000	0x13C	0x400D 413C
IOC_GPT1OCP2	RW	32	0x0000 0000	0x140	0x400D 4140
IOC_GPT2OCP1	RW	32	0x0000 0000	0x144	0x400D 4144
IOC_GPT2OCP2	RW	32	0x0000 0000	0x148	0x400D 4148
IOC_GPT3OCP1	RW	32	0x0000 0000	0x14C	0x400D 414C
IOC_GPT3OCP2	RW	32	0x0000 0000	0x150	0x400D 4150

9.3.2.2 IOC Register Descriptions

IOC_PA0_SEL

Address offset	0x000		
Physical Address	0x400D 4000	**Instance**	IOC
Description	Peripheral select control for PA0		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PA0_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PA0_SEL	Select one peripheral signal output for PA0.	RW	0x00

IOC_PA1_SEL

Address offset	0x004		
Physical Address	0x400D 4004	**Instance**	IOC
Description	Peripheral select control for PA1		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PA1_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PA1_SEL	Select one peripheral signal output for PA1.	RW	0x00

IOC_PA2_SEL

Address offset	0x008		
Physical Address	0x400D 4008	**Instance**	IOC
Description	Peripheral select control for PA2		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PA2_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PA2_SEL	Select one peripheral signal output for PA2.	RW	0x00

IOC_PA3_SEL

Address offset	0x00C		
Physical Address	0x400D 400C	**Instance**	IOC
Description	Peripheral select control for PA3		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PA3_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PA3_SEL	Select one peripheral signal output for PA3.	RW	0x00

IOC_PA4_SEL

Address offset	0x010		
Physical Address	0x400D 4010	**Instance**	IOC
Description	Peripheral select control for PA4		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PA4_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PA4_SEL	Select one peripheral signal output for PA4.	RW	0x00

IOC_PA5_SEL

Address offset	0x014		
Physical Address	0x400D 4014	**Instance**	IOC
Description	Peripheral select control for PA5		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PA5_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PA5_SEL	Select one peripheral signal output for PA5.	RW	0x00

IOC_PA6_SEL

Address offset	0x018		
Physical Address	0x400D 4018	**Instance**	IOC
Description	Peripheral select control for PA6		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PA6_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PA6_SEL	Select one peripheral signal output for PA6.	RW	0x00

IOC_PA7_SEL

Address offset	0x01C		
Physical Address	0x400D 401C	**Instance**	IOC
Description	Peripheral select control for PA7		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PA7_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PA7_SEL	Select one peripheral signal output for PA7.	RW	0x00

IOC_PB0_SEL

Address offset	0x020		
Physical Address	0x400D 4020	**Instance**	IOC
Description	Peripheral select control for PB0		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PB0_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PB0_SEL	Select one peripheral signal output for PB0.	RW	0x00

IOC_PB1_SEL

Address offset	0x024		
Physical Address	0x400D 4024	**Instance**	IOC
Description	Peripheral select control for PB1		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PB1_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PB1_SEL	Select one peripheral signal output for PB1.	RW	0x00

IOC_PB2_SEL

Address offset	0x028		
Physical Address	0x400D 4028	**Instance**	IOC
Description	Peripheral select control for PB2		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PB2_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PB2_SEL	Select one peripheral signal output for PB2.	RW	0x00

IOC_PB3_SEL

Address offset	0x02C		
Physical Address	0x400D 402C	**Instance**	IOC
Description	Peripheral select control for PB3		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PB3_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PB3_SEL	Select one peripheral signal output for PB3.	RW	0x00

IOC_PB4_SEL

Address offset	0x030		
Physical Address	0x400D 4030	**Instance**	IOC
Description	Peripheral select control for PB4		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PB4_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PB4_SEL	Select one peripheral signal output for PB4.	RW	0x00

IOC_PB5_SEL

Address offset	0x034		
Physical Address	0x400D 4034	**Instance**	IOC
Description	Peripheral select control for PB5		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PB5_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PB5_SEL	Select one peripheral signal output for PB5.	RW	0x00

IOC_PB6_SEL

Address offset	0x038		
Physical Address	0x400D 4038	**Instance**	IOC
Description	Peripheral select control for PB6		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PB6_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PB6_SEL	Select one peripheral signal output for PB6.	RW	0x00

IOC_PB7_SEL

Address offset	0x03C		
Physical Address	0x400D 403C	**Instance**	IOC
Description	Peripheral select control for PB7		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PB7_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PB7_SEL	Select one peripheral signal output for PB7.	RW	0x00

IOC_PC0_SEL

Address offset	0x040		
Physical Address	0x400D 4040	**Instance**	IOC
Description	Peripheral select control for PC0		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PC0_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PC0_SEL	Select one peripheral signal output for PC0.	RW	0x00

IOC_PC1_SEL

Address offset	0x044		
Physical Address	0x400D 4044	**Instance**	IOC
Description	Peripheral select control for PC1		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PC1_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PC1_SEL	Select one peripheral signal output for PC1.	RW	0x00

IOC_PC2_SEL

Address offset	0x048		
Physical Address	0x400D 4048	**Instance**	IOC
Description	Peripheral select control for PC2		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PC2_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PC2_SEL	Select one peripheral signal output for PC2.	RW	0x00

IOC_PC3_SEL

Address offset	0x04C		
Physical Address	0x400D 404C	**Instance**	IOC
Description	Peripheral select control for PC3		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PC3_SEL			

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PC3_SEL	Select one peripheral signal output for PC3.	RW	0x00

IOC_PC4_SEL

Address offset	0x050		
Physical Address	0x400D 4050	**Instance**	IOC
Description	Peripheral select control for PC4		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PC4_SEL			

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PC4_SEL	Select one peripheral signal output for PC4.	RW	0x00

IOC_PC5_SEL

Address offset	0x054		
Physical Address	0x400D 4054	**Instance**	IOC
Description	Peripheral select control for PC5		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PC5_SEL			

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PC5_SEL	Select one peripheral signal output for PC5.	RW	0x00

IOC_PC6_SEL

Address offset	0x058		
Physical Address	0x400D 4058	**Instance**	IOC
Description	Peripheral select control for PC6		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PC6_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PC6_SEL	Select one peripheral signal output for PC6.	RW	0x00

IOC_PC7_SEL

Address offset	0x05C		
Physical Address	0x400D 405C	**Instance**	IOC
Description	Peripheral select control for PC7		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PC7_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PC7_SEL	Select one peripheral signal output for PC7.	RW	0x00

IOC_PD0_SEL

Address offset	0x060		
Physical Address	0x400D 4060	**Instance**	IOC
Description	Peripheral select control for PD0		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PD0_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PD0_SEL	Select one peripheral signal output for PD0.	RW	0x00

IOC_PD1_SEL

Address offset	0x064		
Physical Address	0x400D 4064	**Instance**	IOC
Description	Peripheral select control for PD1		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PD1_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PD1_SEL	Select one peripheral signal output for PD1.	RW	0x00

IOC_PD2_SEL

Address offset	0x068		
Physical Address	0x400D 4068	**Instance**	IOC
Description	Peripheral select control for PD2		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PD2_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PD2_SEL	Select one peripheral signal output for PD2.	RW	0x00

IOC_PD3_SEL

Address offset	0x06C		
Physical Address	0x400D 406C	**Instance**	IOC
Description	Peripheral select control for PD3		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PD3_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PD3_SEL	Select one peripheral signal output for PD3.	RW	0x00

IOC_PD4_SEL

Address offset	0x070		
Physical Address	0x400D 4070	**Instance**	IOC
Description	Peripheral select control for PD4		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PD4_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PD4_SEL	Select one peripheral signal output for PD4.	RW	0x00

IOC_PD5_SEL

Address offset	0x074		
Physical Address	0x400D 4074	**Instance**	IOC
Description	Peripheral select control for PD5		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PD5_SEL							

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PD5_SEL	Select one peripheral signal output for PD5.	RW	0x00

IOC_PD6_SEL

Address offset	0x078		
Physical Address	0x400D 4078	**Instance**	IOC
Description	Peripheral select control for PD6		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PD6_SEL			

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PD6_SEL	Select one peripheral signal output for PD6.	RW	0x00

IOC_PD7_SEL

Address offset	0x07C		
Physical Address	0x400D 407C	**Instance**	IOC
Description	Peripheral select control for PD7		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PD7_SEL			

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	PD7_SEL	Select one peripheral signal output for PD7.	RW	0x00

IOC_PA0_OVER

Address offset	0x080		
Physical Address	0x400D 4080	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PA0_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PA0_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PA1_OVER

Address offset	0x084		
Physical Address	0x400D 4084	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PA1_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PA1_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PA2_OVER

Address offset	0x088		
Physical Address	0x400D 4088	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PA2_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PA2_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PA3_OVER

Address offset	0x08C		
Physical Address	0x400D 408C	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PA3_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PA3_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PA4_OVER

Address offset	0x090		
Physical Address	0x400D 4090	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PA4_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PA4_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PA5_OVER

Address offset	0x094		
Physical Address	0x400D 4094	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PA5_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PA5_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PA6_OVER

Address offset	0x098		
Physical Address	0x400D 4098	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PA6_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PA6_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PA7_OVER

Address offset	0x09C		
Physical Address	0x400D 409C	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PA7_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PA7_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PB0_OVER

Address offset	0x0A0		
Physical Address	0x400D 40A0	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PB0_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PB0_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PB1_OVER

Address offset	0x0A4		
Physical Address	0x400D 40A4	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PB1_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PB1_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PB2_OVER

Address offset	0x0A8		
Physical Address	0x400D 40A8	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PB2_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PB2_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PB3_OVER

Address offset	0x0AC		
Physical Address	0x400D 40AC	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PB3_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PB3_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PB4_OVER

Address offset	0x0B0		
Physical Address	0x400D 40B0	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PB4_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PB4_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PB5_OVER

Address offset	0x0B4		
Physical Address	0x400D 40B4	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PB5_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PB5_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PB6_OVER

Address offset	0x0B8		
Physical Address	0x400D 40B8	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PB6_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PB6_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PB7_OVER

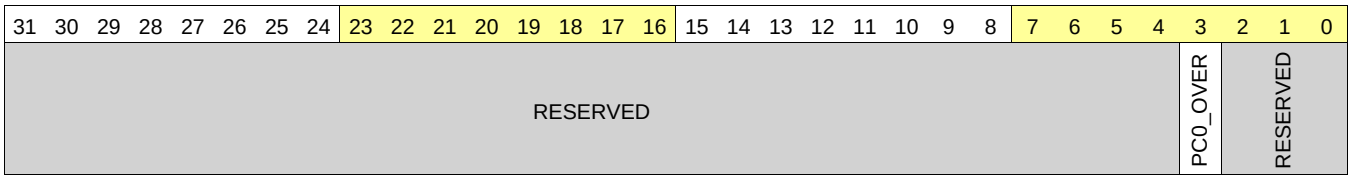
Address offset	0x0BC		
Physical Address	0x400D 40BC	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PB7_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PB7_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PC0_OVER

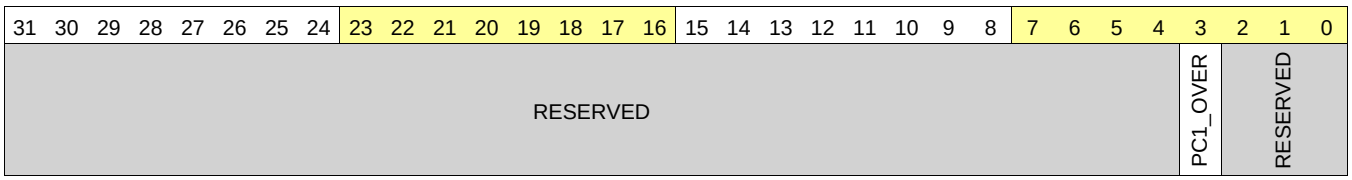
Address offset	0x0C0		
Physical Address	0x400D 40C0	**Instance**	IOC
Description	This is the overide configuration register for each pad. PC0 has high drive capability.		
Type	RW		



Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3	PC0_OVER	0: output disable 1: oe - output enable	RW	0
2:0	RESERVED	This bit field is reserved.	RW	0x4

IOC_PC1_OVER

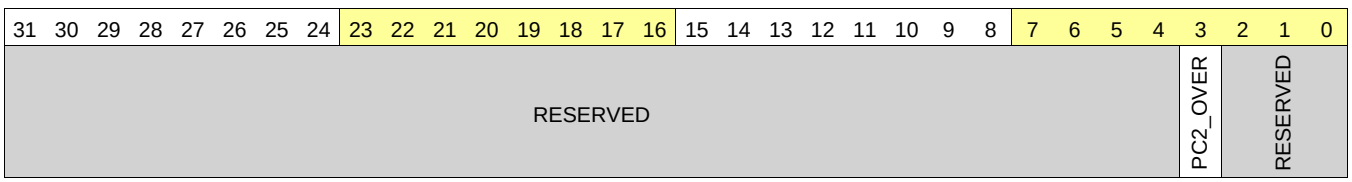
Address offset	0x0C4		
Physical Address	0x400D 40C4	**Instance**	IOC
Description	This is the overide configuration register for each pad. PC1 has high drive capability.		
Type	RW		



Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3	PC1_OVER	0: output disable 1: oe - output enable	RW	0
2:0	RESERVED	This bit field is reserved.	RW	0x4

IOC_PC2_OVER

Address offset	0x0C8		
Physical Address	0x400D 40C8	**Instance**	IOC
Description	This is the overide configuration register for each pad. PC2 has high drive capability.		
Type	RW		



Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3	PC2_OVER	0: output disable 1: oe - output enable	RW	0
2:0	RESERVED	This bit field is reserved.	RW	0x4

IOC_PC3_OVER

Address offset	0x0CC		
Physical Address	0x400D 40CC	**Instance**	IOC
Description	This is the overide configuration register for each pad. PC3 has high drive capability.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PC3_OVER	RESERVED		

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3	PC3_OVER	0: output disable 1: oe - output enable	RW	0
2:0	RESERVED	This bit field is reserved.	RW	0x4

IOC_PC4_OVER

Address offset	0x0D0		
Physical Address	0x400D 40D0	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PC4_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PC4_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PC5_OVER

Address offset	0x0D4		
Physical Address	0x400D 40D4	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PC5_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PC5_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PC6_OVER

Address offset	0x0D8		
Physical Address	0x400D 40D8	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PC6_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PC6_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PC7_OVER

Address offset	0x0DC		
Physical Address	0x400D 40DC	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PC7_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PC7_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PD0_OVER

Address offset	0x0E0		
Physical Address	0x400D 40E0	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PD0_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PD0_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PD1_OVER

Address offset	0x0E4		
Physical Address	0x400D 40E4	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PD1_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PD1_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PD2_OVER

Address offset	0x0E8		
Physical Address	0x400D 40E8	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PD2_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PD2_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PD3_OVER

Address offset	0x0EC		
Physical Address	0x400D 40EC	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PD3_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PD3_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PD4_OVER

Address offset	0x0F0		
Physical Address	0x400D 40F0	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PD4_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PD4_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PD5_OVER

Address offset	0x0F4		
Physical Address	0x400D 40F4	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PD5_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PD5_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PD6_OVER

Address offset	0x0F8		
Physical Address	0x400D 40F8	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PD6_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PD6_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_PD7_OVER

Address offset	0x0FC		
Physical Address	0x400D 40FC	**Instance**	IOC
Description	This is the overide configuration register for each pad.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												PD7_OVER			

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3:0	PD7_OVER	0x8: oe - output enable 0x4: pue - pullup enable 0x2: pde - pulldown enable 0x1: ana - analog enable	RW	0x4

IOC_UARTRXD_UART0

Address offset	0x100		
Physical Address	0x400D 4100	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the UART0 RX.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as UART0 RX 1: PA1 selected as UART0 RX ... 31: PD7 selected as UART0 RX	RW	0x00

IOC_UARTCTS_UART1

Address offset	0x104		
Physical Address	0x400D 4104	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the UART1 CTS.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as UART1 CTS 1: PA1 selected as UART1 CTS ... 31: PD7 selected as UART1 CTS	RW	0x00

IOC_UARTRXD_UART1

Address offset	0x108		
Physical Address	0x400D 4108	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the UART1 RX.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as UART1 RX 1: PA1 selected as UART1 RX ... 31: PD7 selected as UART1 RX	RW	0x00

IOC_CLK_SSI_SSI0

Address offset	0x10C		
Physical Address	0x400D 410C	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the SSI0 CLK.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as SSI0 CLK 1: PA1 selected as SSI0 CLK ... 31: PD7 selected as SSI0 CLK	RW	0x00

IOC_SSIRXD_SSI0

Address offset	0x110		
Physical Address	0x400D 4110	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the SSI0 RX.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as SSI0 RX 1: PA1 selected as SSI0 RX ... 31: PD7 selected as SSI0 RX	RW	0x00

IOC_SSIFSSIN_SSI0

Address offset	0x114		
Physical Address	0x400D 4114	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the SSI0 FSSIN.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as SSI0 FSSIN 1: PA1 selected as SSI0 FSSIN ... 31: PD7 selected as SSI0 FSSIN	RW	0x00

IOC_CLK_SSIIN_SSI0

Address offset	0x118		
Physical Address	0x400D 4118	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the SSI0 CLK_SSIN.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as SSI0 CLK_SSIN 1: PA1 selected as SSI0 CLK_SSIN ... 31: PD7 selected as SSI0 CLK_SSIN	RW	0x00

IOC_CLK_SSI_SSI1

Address offset	0x11C		
Physical Address	0x400D 411C	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the SSI1 CLK.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as SSI1 CLK 1: PA1 selected as SSI1 CLK ... 31: PD7 selected as SSI1 CLK	RW	0x00

IOC_SSIRXD_SSI1

Address offset	0x120		
Physical Address	0x400D 4120	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the SSI1 RX.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as SSI1 RX 1: PA1 selected as SSI1 RX ... 31: PD7 selected as SSI1 RX	RW	0x00

IOC_SSIFSSIN_SSI1

Address offset	0x124		
Physical Address	0x400D 4124	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the SSI1 FSSIN.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as SSI1 FSSIN 1: PA1 selected as SSI1 FSSIN ... 31: PD7 selected as SSI1 FSSIN	RW	0x00

IOC_CLK_SSIIN_SSI1

Address offset	0x128		
Physical Address	0x400D 4128	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the SSI1 CLK_SSIN.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as SSI1 CLK_SSIN 1: PA1 selected as SSI1 CLK_SSIN ... 31: PD7 selected as SSI1 CLK_SSIN	RW	0x00

IOC_I2CMSSDA

Address offset	0x12C		
Physical Address	0x400D 412C	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the I2C SDA.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as I2C SDA 1: PA1 selected as I2C SDA ... 31: PD7 selected as I2C SDA	RW	0x00

IOC_I2CMSSCL

Address offset	0x130		
Physical Address	0x400D 4130	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the I2C SCL.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as I2C SCL 1: PA1 selected as I2C SCL ... 31: PD7 selected as I2C SCL	RW	0x00

IOC_GPT0OCP1

Address offset	0x134		
Physical Address	0x400D 4134	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the GPT0OCP1.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as GPT0OCP1 1: PA1 selected as GPT0OCP1 ... 31: PD7 selected as GPT0OCP1	RW	0x00

IOC_GPT0OCP2

Address offset	0x138		
Physical Address	0x400D 4138	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the GPT0OCP2.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as GPT0OCP2 1: PA1 selected as GPT0OCP2 ... 31: PD7 selected as GPT0OCP2	RW	0x00

IOC_GPT1OCP1

Address offset	0x13C		
Physical Address	0x400D 413C	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the GPT1OCP1.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as GPT1OCP1 1: PA1 selected as GPT1OCP1 ... 31: PD7 selected as GPT1OCP1	RW	0x00

IOC_GPT1OCP2

Address offset	0x140		
Physical Address	0x400D 4140	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the GPT1OCP2.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as GPT1OCP2 1: PA1 selected as GPT1OCP2 ... 31: PD7 selected as GPT1OCP2	RW	0x00

IOC_GPT2OCP1

Address offset	0x144		
Physical Address	0x400D 4144	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the GPT2OCP1.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as GPT2OCP1 1: PA1 selected as GPT2OCP1 ... 31: PD7 selected as GPT2OCP1	RW	0x00

IOC_GPT2OCP2

Address offset	0x148		
Physical Address	0x400D 4148	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the GPT2OCP2.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as GPT2OCP2 1: PA1 selected as GPT2OCP2 ... 31: PD7 selected as GPT2OCP2	RW	0x00

IOC_GPT3OCP1

Address offset	0x14C		
Physical Address	0x400D 414C	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the GPT3OCP1.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as GPT3OCP1 1: PA1 selected as GPT3OCP1 ... 31: PD7 selected as GPT3OCP1	RW	0x00

IOC_GPT3OCP2

Address offset	0x150		
Physical Address	0x400D 4150	**Instance**	IOC
Description	Selects one of the 32 pins on the four 8-pin I/O-ports (port A, port B, port C, and port D) to be the GPT3OCP2.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RO	0x000 0000
4:0	INPUT_SEL	0: PA0 selected as GPT3OCP2 1: PA1 selected as GPT3OCP2 ... 31: PD7 selected as GPT3OCP2	RW	0x00

Micro Direct Memory Access

This chapter describes the micro direct memory access (µDMA).

Topic **Page**

10.1 µDMA Introduction

The microcontroller includes a direct memory access (DMA) controller, known as micro-DMA (µDMA). The µDMA controller provides a way to offload data transfer tasks from the Cortex™-M3 processor, allowing for more efficient use of the processor and the available bus bandwidth. The µDMA controller can perform transfers between memory and peripherals. It has dedicated channels for each supported on-chip module and can be programmed to automatically perform transfers between peripherals and memory as the peripheral is ready to transfer more data. The µDMA controller provides the following features:

- ARM® PrimeCell 32-channel configurable µDMA controller
- Support for memory-to-memory, memory-to-peripheral, and peripheral-to-memory in multiple transfer modes:
  - Basic for simple transfer scenarios
  - Ping-pong for continuous data flow
  - Scatter-gather for a programmable list of arbitrary transfers initiated from a single request
- Highly flexible and configurable channel operation:
  - Independently configured and operated channels
  - Dedicated channels for supported on-chip modules
  - Primary and secondary channel assignments
  - Flexible channel assignments
  - One channel each for receive and transmit paths for bidirectional modules
  - Dedicated channel for software-initiated transfers
  - Per-channel configurable priority scheme
  - Optional software-initiated requests for any channel
- Two levels of priority
- Design optimizations for improved bus access performance between µDMA controller and the processor core:
  - µDMA controller access is subordinate to core access
  - Peripheral bus segmentation
- Data sizes of 8, 16, and 32 bits
- Transfer size is programmable in binary steps from 1 to 1024.
- Source and destination address increment size of byte, half-word, word, or no increment
- Maskable peripheral requests
- Interrupt on transfer completion, with a separate interrupt per channel

10.2 Block Diagram

Figure 10-1 shows the µDMA block diagram.

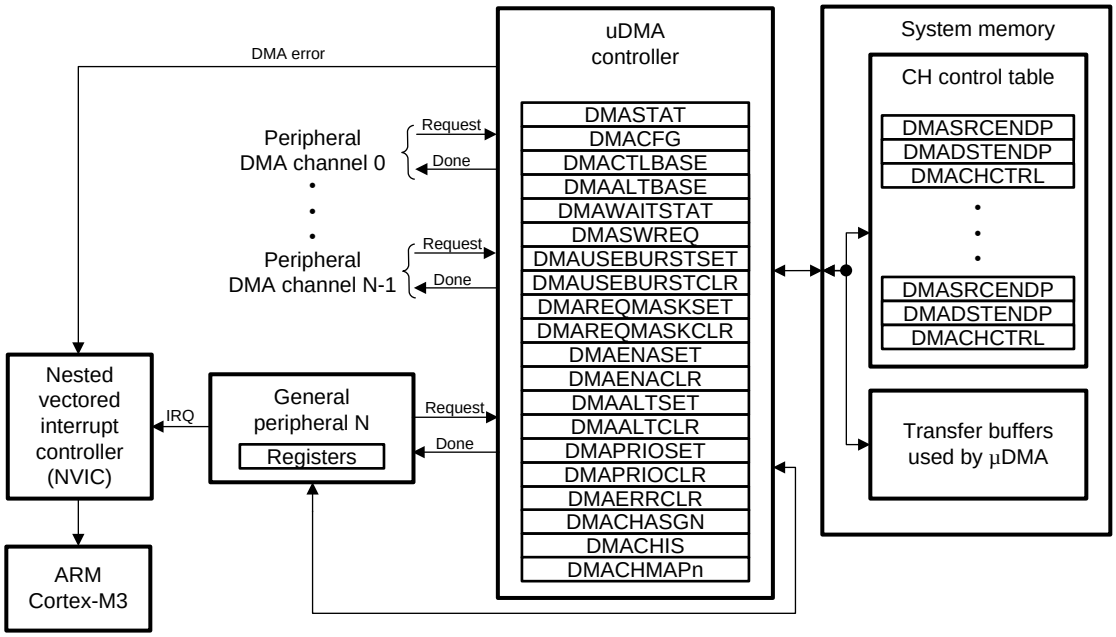


Figure 10-1. µDMA Block Diagram

10.3 Functional Description

The µDMA controller is a flexible and highly configurable DMA controller designed to work efficiently with the microcontroller Cortex-M3 processor core. It supports multiple data sizes and address increment schemes, multiple levels of priority among DMA channels, and several transfer modes to allow for sophisticated programmed data transfers. Since the CC2538 uses an AHB matrix, as long as the CPU and µDMA controller are targeting different slaves, the transfers will be concurrent. The only time the CPU will delay a µDMA transfer is if they are targeting the same slave device. In this instance, the µDMA will have a lower priority. Otherwise, bus masters behave as though they were the only master because the AHB matrix creates simultaneous connections for both masters.

The µDMA controller can transfer data to and from the on-chip SRAM. However, because the flash memory and ROM are on a separate internal bus, it is not possible to transfer data from the flash memory or ROM with the µDMA controller.

Each peripheral function that is supported has a dedicated channel on the µDMA controller that can be configured independently. The µDMA controller implements a unique configuration method using channel control structures that are maintained in system memory by the processor. While simple transfer modes are supported, it is also possible to build up sophisticated task lists in memory that allow the µDMA controller to perform arbitrary-sized transfers to and from arbitrary locations as part of a single transfer request. The µDMA controller also supports the use of ping-pong buffering to accommodate constant streaming of data to or from a peripheral.

Each channel also has a configurable arbitration size. The arbitration size is the number of items that are transferred in a burst before the µDMA controller requests channel priority. Using the arbitration size, it is possible to control exactly how many items are transferred to or from a peripheral each time it makes a µDMA service request.

10.3.1 Channel Assignments

µDMA channels 0–31 are assigned to peripherals according to Table 10-1. Each DMA channel has up to four possible assignments which are selected using the **DMA Channel Map Select n (CHMAPn)** registers with 4-bit assignment fields for each µDMA channel.

Table 10-1 shows the µDMA channel mapping. The Enc. column shows the encoding for the respective **CHMAPn** bit field. Encodings 0x4 – 0xF are all reserved. To support legacy software which uses the **DMA Channel Assignment (DMACHASGN)** register, Enc. 0 is equivalent to a **DMACHASGN** bit being clear, and Enc. 1 is equivalent to a **DMACHASGN** bit being set. If the **DMACHASGN** register is read, bit fields return 0 if the corresponding **CHMAPn** register field values are equal to 0; otherwise they return 1 if the corresponding **CHMAPn** register field values are not equal to 0. The Type column in the table indicates if a particular peripheral uses a single request (S), burst request (B), or either (SB).

NOTE: Channels noted in the table as "Available for software" may be assigned to peripherals in the future. However, they are currently available for software use. Channel 30 is dedicated for software use.

Table 10-1. µDMA Channel Assignments

Enc.	0		1		2		3		4	
Ch #	Peripheral	Type	Peripheral	Type	Peripheral	Type	Peripheral	Type	Peripheral	Type
0	Available for software	B	Available for software	B	Available for software	B	Available for software	B	-	B
1	Available for software	B	Available for software	B	Available for software	B	Available for software	B	ADC Combined	B
2	Available for software	B	GPTimer 3A	B	Available for software	B	Available for software	B	Flash	B
-	-	-	-	-	-	-	-	-	-	-
4	Available for software	B	GPTimer 2A	B	Available for software	B	Available for software	B	RF Core Trigger	B
5	Available for software	B	GPTimer 2B	B	Available for software	B	Available for software	B	Software	B
6	Available for software	B	GPTimer 2A	B	Available for software	B	Available for software	B	Available for software	B
7	Available for software	B	GPTimer 2B	B	Available for software	B	Available for software	B	Available for software	B
8	UART0 RX	SB	UART1 RX	SB	Available for software	SB	Available for software	SB	Available for software	B
9	UART0 TX	SB	UART1 TX	SB	Available for software	SB	Available for software	SB	Available for software	B
10	SSI0 RX	SB	SSI1 RX	SB	Available for software	SB	Available for software	SB	Available for software	B
11	SSI0 TX	SB	SSI1 TX	SB	Available for software	SB	Available for software	SB	Available for software	B
12	Available for software	B	Available for software	B	Available for software	B	Available for software	B	Available for software	B
13	Available for software	B	Available for software	B	Available for software	B	Available for software	B	Available for software	B
14	ADC Ch0	B	GPTimer 2A	B	Available for software	B	Available for software	B	Available for software	B
15	ADC Ch1	B	GPTimer 2B	B	Available for software	B	Available for software	B	Available for software	B
16	ADC Ch2	B	Available for software	B	Available for software	B	Available for software	B	Available for software	B
17	ADC Ch3	B	Available for software	B	Available for software	B	Available for software	B	Available for software	B
18	GPTimer 0A	B	GPTimer 1A	B	Available for software	B	Available for software	B	Available for software	B
19	GPTimer 0B	B	GPTimer 1B	B	Available for software	B	Available for software	B	Available for software	B
20	GPTimer 1A	B	Available for software	B	Available for software	B	Available for software	B	Available for software	B
21	GPTimer 1B	B	Available for software	B	Available for software	B	Available for software	B	Available for software	B
22	UART1 RX	S B	Available for software	B	Available for software	B	Available for software	B	Available for software	B
23	UART1 TX	S B	Available for software	B	Available for software	B	Available for software	B	Available for software	B
24	SSI1 RX	S B	ADC Ch4	B	Available for softwarere	B	Available for software	B	Available for software	B
25	SSI1 TX	S B	ADC Ch5	B	Available for software	B	Available for software	B	Available for software	B

Table 10-1. µDMA Channel Assignments (continued)

Enc.	0		1		2		3		4	
Ch #	Peripheral	Type	Peripheral	Type	Peripheral	Type	Peripheral	Type	Peripheral	Type
26	Available for software	B	ADC Ch6	B	Available for software	B	Available for software	B	Available for software	B
27	Available for software	B	ADC Ch7	B	Available for software	B	Available for software	B	Available for software	B
28	Available for software	B	Available for software	B	Available for software	B	Available for software	B	Available for software	B
29	Available for software	B	Available for software	B	Available for software	B	Available for software	B	MAC Timer 2 Trigger 1	B
30	Available for software	B	Available for software	B	Available for software	B	Available for software	B	MAC Timer 2 Trigger 2	B
31	Reserved	B	Reserved	B	Reserved	B	Reserved	B	Reserved	B

10.3.2 Priority

The µDMA controller assigns priority to each channel based on the channel number and the priority level bit for the channel. Channel 0 has the highest priority, and as the channel number increases, the priority of a channel decreases. Each channel has a priority level bit to provide two levels of priority: default priority and high priority. If the priority level bit is set, then that channel has a higher priority than all other channels at default priority. If multiple channels are set for high priority, then the channel number is used to determine relative priority among all the high-priority channels.

The priority bit for a channel can be set using the **DMA Channel Priority Set (UDMA_PRIOSET)** register and cleared with the **DMA Channel Priority Clear (UDMA_PRIOCLR)** register.

10.3.3 Arbitration Size

When a µDMA channel requests a transfer, the µDMA controller arbitrates among all the channels making a request and services the µDMA channel with the highest priority. Once a transfer begins, it continues for a selectable number of transfers before re-arbitrating among the requesting channels again. The arbitration size can be configured for each channel, ranging from 1 to 1024 item transfers. After the µDMA controller transfers the number of items specified by the arbitration size, it then checks among all the channels making a request and services the channel with the highest priority.

If a lower-priority µDMA channel uses a large arbitration size, the latency for higher-priority channels is increased because the µDMA controller completes the lower-priority burst before checking for higher-priority requests. Therefore, lower-priority channels must not use a large arbitration size for best response on high-priority channels.

The arbitration size can also be thought of as a burst size. It is the maximum number of items that are transferred at any one time in a burst. Here, the term arbitration refers to determination of µDMA channel priority, not arbitration for the bus. When the µDMA controller arbitrates for the bus, the processor always takes priority. Furthermore, the µDMA controller is delayed whenever the processor must perform a bus transaction on the same bus, even in the middle of a burst transfer.

10.3.4 Request Types

The µDMA controller responds to two types of requests from a peripheral: single or burst. Each peripheral may support either or both types of requests. A single request means that the peripheral is ready to transfer one item, while a burst request means that the peripheral is ready to transfer multiple items.

The µDMA controller responds differently depending on whether the peripheral is making a single request or a burst request. If both are asserted, and the µDMA channel has been set up for a burst transfer, then the burst request takes precedence. See Table 10-2, which lists how each peripheral supports the two request types.

Table 10-2. Request Type Support

Peripheral	Single Request Signal	Burst Request Signal
ADC	None. FIFO not empty	Sequencer **IE** bit. FIFO half full
General-purpose timer	Raw interrupt pulse	None
GPIO	Raw interrupt pulse	None
SSI TX	TX FIFO not full	TX FIFO level (fixed at 4)
SSI RX	RX FIFO not empty	RX FIFO level (fixed at 4)
UART TX	TX FIFO not full	TX FIFO level (configurable)
UART RX	RX FIFO not empty	RX FIFO level (configurable)

10.3.4.1 Single Request

When a single request is detected (but not a burst request), the µDMA controller transfers one item and then stops to wait for another request.

10.3.4.2 Burst Request

When a burst request is detected, the µDMA controller transfers the number of items that is the lesser of the arbitration size or the number of items remaining in the transfer. Therefore, the arbitration size should be the same as the number of data items that the peripheral can accommodate when making a burst request. For example, the UART generates a burst request based on the FIFO trigger level. In this case, the arbitration size should be set to the amount of data that the FIFO can transfer when the trigger level is reached. A burst transfer runs to completion once it starts, and cannot be interrupted, even by a higher-priority channel. Burst transfers complete in a shorter time than the same number of non-burst transfers.

It may be desirable to use only burst transfers and not allow single transfers. For example, perhaps the nature of the data is such that it only makes sense when transferred together as a single unit rather than one piece at a time. The single request can be disabled by using the **DMA Channel Useburst Set (UDMA USEBURSTSET)** register. By setting the bit for a channel in this register, the µDMA controller responds only to burst requests for that channel.

10.3.5 Channel Configuration

The µDMA controller uses an area of system memory to store a set of channel control structures in a table. The control table may have one or two entries for each µDMA channel. Each entry in the table structure contains source and destination pointers, transfer size, and transfer mode. The control table can be located anywhere in system memory, but it must be contiguous and aligned on a 1024-byte boundary.

Table 10-3 shows the layout in memory of the channel control table. Each channel may have one or two control structures in the control table: a primary control structure and an optional alternate control structure. The table is organized so that all of the primary entries are in the first half of the table, and all the alternate structures are in the second half of the table. The primary entry is used for simple transfer modes where transfers can be reconfigured and restarted after each transfer completes. In this case, the alternate control structures are not used and therefore only the first half of the table must be allocated in memory; the second half of the control table is not necessary, and that memory can be used for something else. If a more complex transfer mode, such as ping-pong or scatter-gather, is used, then the alternate control structure is also used and memory space should be allocated for the entire table.

Any unused memory in the control table may be used by the application. This includes the control structures for any channels that are unused by the application as well as the unused control word for each channel.

Table 10-3. Control Structure Memory Map

Offset	Channel
0x0	0, Primary
0x10	1, Primary
...	...

Table 10-3. Control Structure Memory Map (continued)

Offset	Channel
0x1F0	31, Primary
0x200	0, Alternate
0x210	1, Alternate
...	...
0x3F0	31, Alternate

Table 10-4 shows an individual control structure entry in the control table. Each entry is aligned on a 16-byte boundary. The entry contains four long words: the source end pointer, the destination end pointer, the control word, and an unused entry. The end pointers point to the ending address of the transfer and are inclusive. If the source or destination is nonincrementing (as for a peripheral register), then the pointer should point to the transfer address.

Table 10-4. Channel Control Structure

Offset	Description
0x000	Source end pointer
0x004	Destination end pointer
0x008	Control word
0x00C	Unused entry

The control word contains the following fields:

- Source and destination data sizes
- Source and destination address increment size
- Number of transfers before bus arbitration
- Total number of items to transfer
- Useburst flag
- Transfer mode

The control word and each field are described in detail in Section 10.5.1. The µDMA controller updates the transfer size and transfer mode fields as the transfer is performed. At the end of a transfer, the transfer size indicates 0, and the transfer mode indicates stopped. Because the control word is modified by the µDMA controller, it must be reconfigured before each new transfer. The source and destination end pointers are not modified, so they can be left unchanged if the source or destination addresses remain the same.

Before starting a transfer, a µDMA channel must be enabled by setting the appropriate bit in the **DMA Channel Enable Set (UDMA_ENASET)** register. A channel can be disabled by setting the channel bit in the **DMA Channel Enable Clear (UDMA_ENACLR)** register. At the end of a complete µDMA transfer, the controller automatically disables the channel.

10.3.6 Transfer Modes

The µDMA controller supports several transfer modes. Two of the modes support simple one-time transfers. Several complex modes support a continuous flow of data.

10.3.6.1 Stop Mode

While stop mode is not actually a transfer mode, stop is a valid value for the mode field of the control word. When the mode field has this value, the µDMA controller does not perform any transfers and disables the channel if it is enabled. At the end of a transfer, the µDMA controller updates the control word to set the mode to stop.

10.3.6.2 Basic Mode

In basic mode, the µDMA controller performs transfers as long as there are more items to transfer, and a transfer request is present. This mode is used with peripherals that assert a µDMA request signal whenever the peripheral is ready for a data transfer. Basic mode should not be used in any situation where the request is momentary even though the entire transfer should be completed. For example, a software-initiated transfer creates a momentary request, and in basic mode, only the number of transfers specified by the **ARBSIZE** field in the **DMA Channel Control Word (UDMA_CHCTL)** register is transferred on a software request, even if there is more data to transfer.

When all of the items have been transferred using basic mode, the µDMA controller sets the mode for that channel to stop.

10.3.6.3 Auto Mode

Auto mode is similar to basic mode, except that when a transfer request is received, the transfer completes, even if the µDMA request is removed. This mode is suitable for software-triggered transfers. Generally, auto mode is not used with a peripheral.

When all the items have been transferred using auto mode, the µDMA controller sets the mode for that channel to stop.

10.3.6.4 Ping-Pong

Ping-pong mode is used to support a continuous data flow to or from a peripheral. To use ping-pong mode, both the primary and alternate data structures must be implemented. Both structures are set up by the processor for data transfer between memory and a peripheral. The transfer is started using the primary control structure. When the transfer using the primary control structure completes, the µDMA controller reads the alternate control structure for that channel to continue the transfer. Each time this happens, an interrupt is generated, and the processor can reload the control structure for the just-completed transfer. Data flow can continue indefinitely this way, using the primary and alternate control structures to switch back and forth between buffers as the data flows to or from the peripheral.

Figure 10-2 shows an example operation in ping-pong mode.

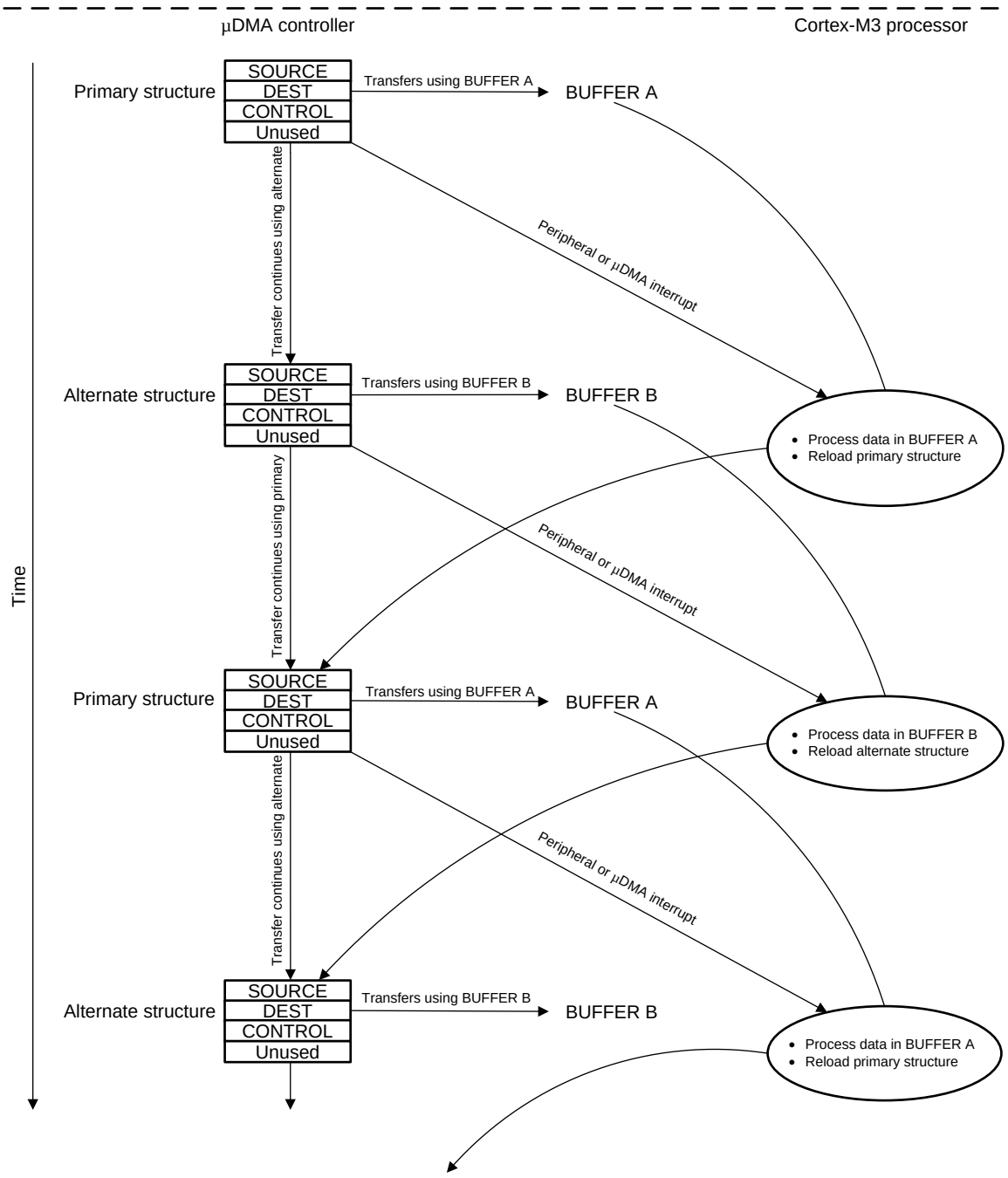


Figure 10-2. Example of Ping-Pong µDMA Transaction

10.3.6.5 Memory Scatter-Gather

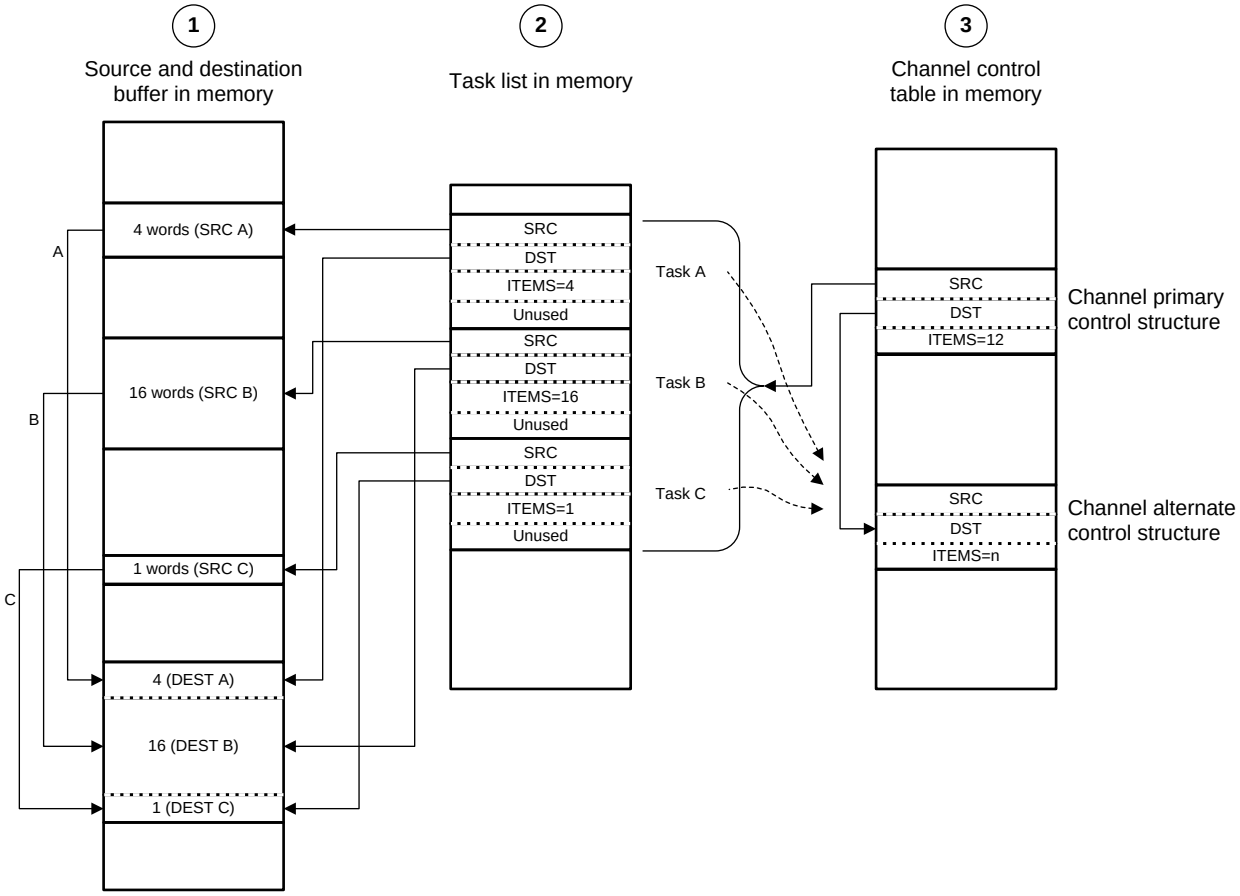
Memory scatter-gather mode is a complex mode used when data must be transferred to or from varied locations in memory instead of a set of contiguous locations in a memory buffer. For example, a gather µDMA operation could be used to selectively read the payload of several stored packets of a communication protocol and store them together in sequence in a memory buffer.

In memory scatter-gather mode, the primary control structure is used to program the alternate control structure from a table in memory. The table is set up by the processor software and contains a list of control structures, each containing the source and destination end pointers, and the control word for a specific transfer. The mode of each control word must be set to memory scatter-gather mode. Each entry in the table is copied in turn to the alternate structure where it is then executed. The µDMA controller alternates between using the primary control structure to copy the next transfer instruction from the list and then executing the new transfer instruction. The end of the list is marked by programming the control word for the last entry to use auto transfer mode. Once the last transfer is performed using auto mode, the µDMA controller stops. A completion interrupt is generated only after the last transfer. It is possible to loop the list by having the last entry copy the primary control structure to point back to the beginning of the list (or to a new list). It is also possible to trigger a set of other channels to perform a transfer, either directly, by programming a write to the software trigger for another channel, or indirectly, by causing a peripheral action that results in a µDMA request.

By programming the µDMA controller using this method, a set of arbitrary transfers can be performed based on a single µDMA request.

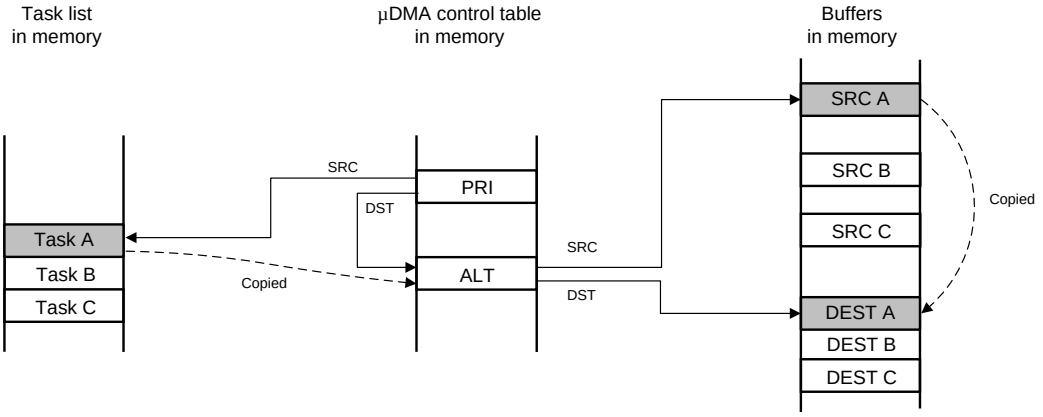
Figure 10-3 shows an example of operation in memory scatter-gather mode. This example shows a gather operation, where data in three separate buffers in memory is copied together into one buffer. Figure 10-3 shows how the application sets up a µDMA task list in memory that is used by the controller to perform three sets of copy operations from different locations in memory. The primary control structure for the channel that is used for the operation is configured to copy from the task list to the alternate control structure.

Figure 10-4 shows the sequence as the µDMA controller performs the three sets of copy operations. First, using the primary control structure, the µDMA controller loads the alternate control structure with task A. It then performs the copy operation specified by task A, copying the data from the source buffer A to the destination buffer. Next, the µDMA controller again uses the primary control structure to load task B into the alternate control structure, and then performs the B operation with the alternate control structure. The process is repeated for task C.



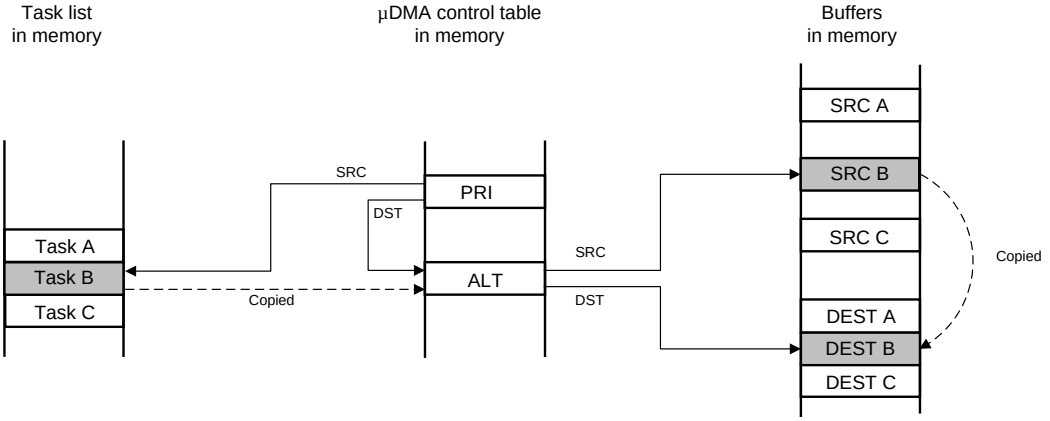
NOTES:
1. Application has a need to copy data items from three separate locations in memory into one combined buffer.
2. Application sets up µDMA "task list" in memory, which contains the pointers and control configuration for three µDMA copy "tasks."
3. Application sets up the channel primary control structure to copy each task configuration, one at a time, to the alternate control structure, where it is executed by the µDMA controller.

Figure 10-3. Memory Scatter-Gather, Setup and Configuration



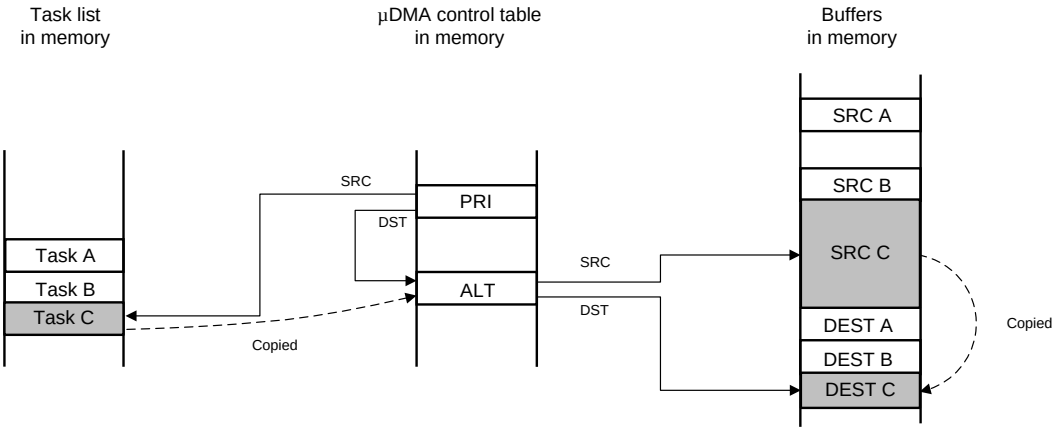
Using the primary control structure of the channel, the µDMA controller copies task A configuration to the alternate control structure of the channel.

Then, using the alternate control structure of the channel, the µDMA controller copies data from source buffer A to the destination buffer.



Using the primary control structure of the channel, the µDMA controller copies task B configuration to the alternate control structure of the channel.

Then, using the alternate control structure of the channel, the µDMA controller copies data from source buffer B to the destination buffer.



Using the primary control structure of the channel, the µDMA controller copies task C configuration to the alternate control structure of the channel.

Then, using the alternate control structure of the channel, the µDMA controller copies data from source buffer C to destination buffer.

Figure 10-4. Memory Scatter-Gather, µDMA Copy Sequence

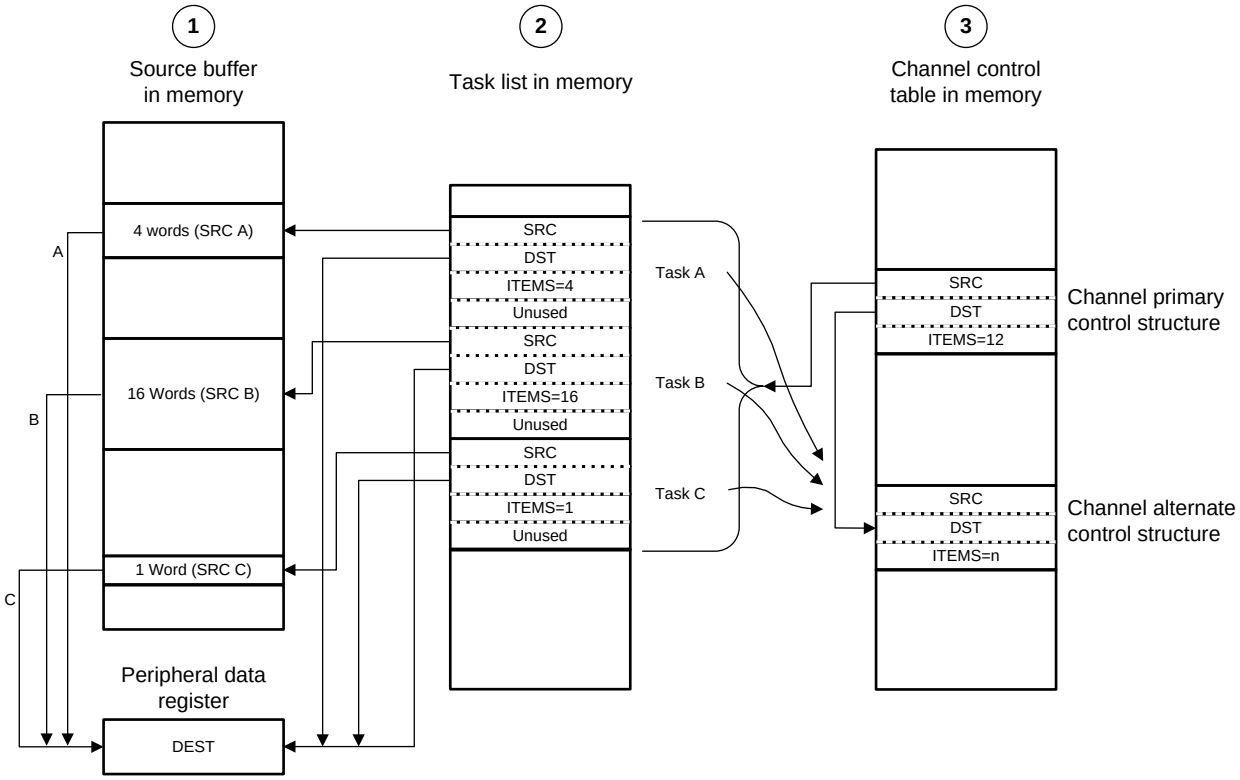
10.3.6.6 Peripheral Scatter-Gather

Peripheral scatter-gather mode is very similar to memory scatter-gather mode, except that the transfers are controlled by a peripheral making a µDMA request. When the µDMA controller detects a request from the peripheral, the µDMA controller uses the primary control structure to copy one entry from the list to the alternate control structure and then performs the transfer. At the end of this transfer, the next transfer is started only if the peripheral again asserts a µDMA request. The µDMA controller continues to perform transfers from the list only when the peripheral makes a request, until the last transfer completes. A completion interrupt is generated only after the last transfer.

By using this method, the µDMA controller can transfer data to or from a peripheral from a set of arbitrary locations whenever the peripheral is ready to transfer data.

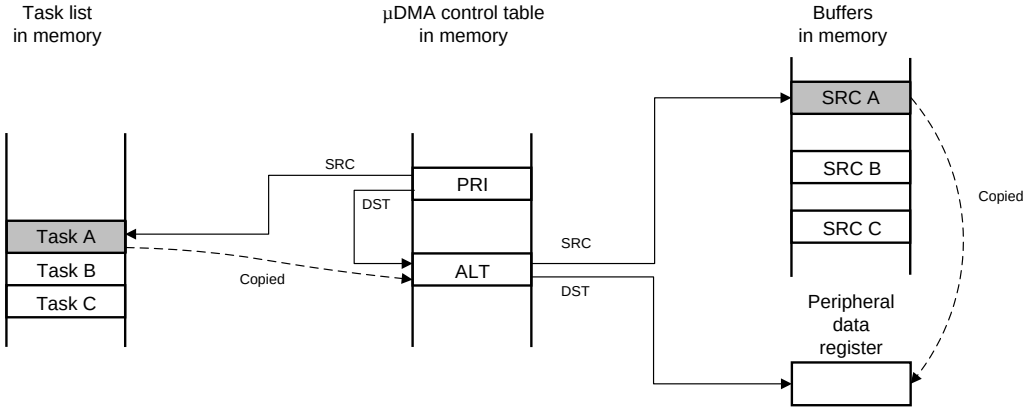
Figure 10-5 shows an example of operation in peripheral scatter-gather mode. This example shows a gather operation, where data from three separate buffers in memory is copied to a single peripheral data register. Figure 10-5 shows how the application sets up a µDMA task list in memory that is used by the controller to perform three sets of copy operations from different locations in memory. The primary control structure for the channel that is used for the operation is configured to copy from the task list to the alternate control structure.

Figure 10-6 shows the sequence as the µDMA controller performs the three sets of copy operations. First, using the primary control structure, the µDMA controller loads the alternate control structure with task A. It then performs the copy operation specified by task A, copying the data from the source buffer A to the peripheral data register. Next, the µDMA controller again uses the primary control structure to load task B into the alternate control structure, and then performs the B operation with the alternate control structure. The process is repeated for task C.



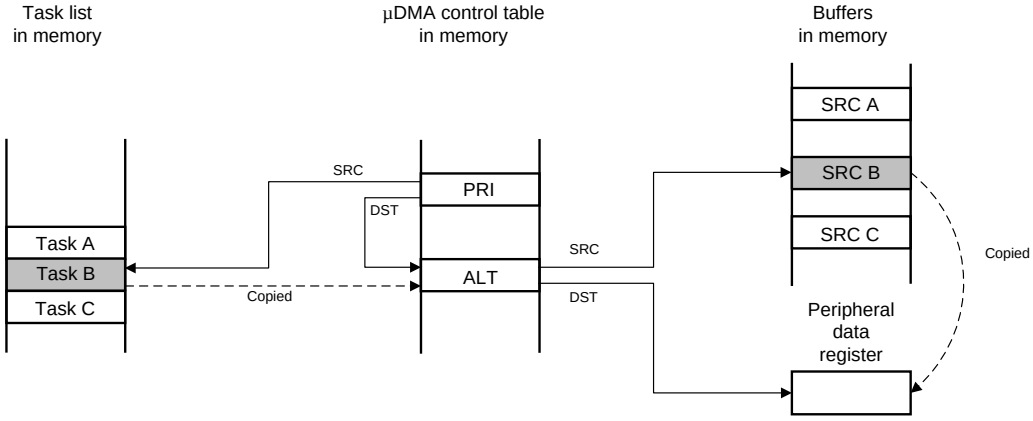
NOTES:
1. Application has a need to copy data items from three separate locations in memory into a peripheral data register.
2. Application sets up µDMA "task list" in memory, which contains the pointers and control configuration for three µDMA copy "tasks."
3. Application sets up the channel primary control structure to copy each task configuration, one at a time, to the alternate control structure, where it is executed by the µDMA controller.

Figure 10-5. Peripheral Scatter-Gather, Setup and Configuration



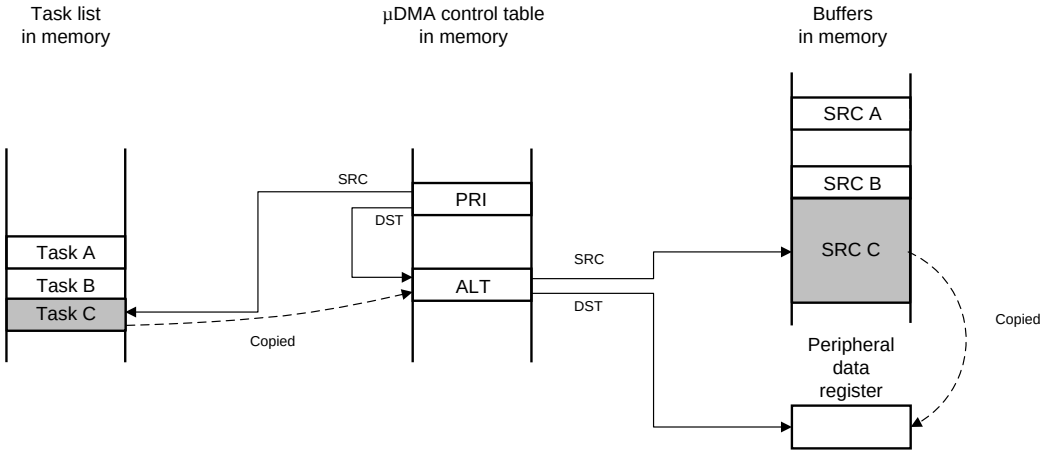
Using the primary control structure of the channel, the µDMA controller copies task A configuration to the alternate control structure of the channel.

Then, using the alternate control structure of the channel, the µDMA controller copies data from source buffer A to the peripheral data register.



Using the primary control structure of the channel, the µDMA controller copies task B configuration to the alternate control structure of the channel.

Then, using the alternate control structure of the channel, the µDMA controller copies data from source buffer B to the peripheral data register.



Using the primary control structure of the channel, the µDMA controller copies task C configuration to the alternate control structure of the channel.

Then, using the alternate control structure of the channel, the µDMA controller copies data from source buffer C to the peripheral data register.

Figure 10-6. Peripheral Scatter-Gather, µDMA Copy Sequence

10.3.7 Transfer Size and Increment

The µDMA controller supports transfer data sizes of 8, 16, or 32 bits. The source and destination data size must be the same for any given transfer. The source and destination address can be auto-incremented by bytes, half-words, or words, or can be set to no increment. The source and destination address increment values can be set independently, and it is not necessary for the address increment to match the data size as long as the increment is the same or larger than the data size. For example, it is possible to perform a transfer using 8-bit data size, but using an address increment of full words (4 bytes). The data to be transferred must be aligned in memory according to the data size (8, 16, or 32 bits).

Table 10-5 shows the configuration to read from a peripheral that supplies 8-bit data.

Table 10-5. µDMA Read Example: 8-Bit Peripheral

Field	Configuration
Source data size	8 bits
Destination data size	8 bits
Source address increment	No increment
Destination address increment	Byte
Source end pointer	Peripheral read FIFO register
Destination end pointer	End of the data buffer in memory

10.3.8 Peripheral Interface

Each peripheral that supports µDMA has a single request and/or burst request signal that is asserted when the peripheral is ready to transfer data (see Table 10-2). The request signal can be disabled or enabled using the **DMA Channel Request Mask Set (UDMA_REQMASKSET)** and **DMA Channel Request Mask Clear (UDMA_REQMASKCLR)** registers. The µDMA request signal is disabled, or masked, when the channel request mask bit is set. When the request is not masked, the µDMA channel is configured correctly and enabled, and the peripheral asserts the request signal, the µDMA controller begins the transfer.

NOTE: When using µDMA to transfer data to and from a peripheral, the peripheral must disable all interrupts to the NVIC.

When a µDMA transfer is complete, the µDMA controller generates an interrupt; for more information, see Section 10.3.10, *Interrupts and Errors*.

For more information on how a specific peripheral interacts with the µDMA controller, see the DMA Operation section in the chapter that discusses that peripheral.

10.3.9 Software Request

Any DMA channel can be used for software transfers. When programming the DMA using the **DMA Channel Software Request (UDMA_SWREQ)** register of the µDMA, any channel may be set up to perform a DMA transfer. In these cases any channel, used for software, that is also tied to a specific peripheral, as described in the table below, will provide its dma_done/interrupt signal directly to the M3 CPU instead of sending it to the peripheral. The interrupt used is a combined interrupt, number 46 – software udma interrupt, for all software transfers.

It is possible to initiate a transfer on any channel using the **UDMA_SWREQ** register. If software uses a µDMA channel of the peripheral to initiate a request, then the completion interrupt occurs on the interrupt vector for the peripheral instead of the software interrupt vector. Any channel may be used for software requests as long as the corresponding peripheral is not using µDMA for data transfer.

10.3.10 Interrupts and Errors

When a µDMA transfer completes, the µDMA controller generates a completion interrupt on the interrupt vector of the peripheral. Therefore, if µDMA is used to transfer data for a peripheral and interrupts are used, then the interrupt handler for that peripheral must be designed to handle the µDMA transfer completion interrupt. If the transfer uses the software µDMA channel, then the completion interrupt occurs on the dedicated software µDMA interrupt vector (see Table 10-6).

When µDMA is enabled for a peripheral, the µDMA controller stops the normal transfer interrupts for a peripheral from reaching the interrupt controller (INTC) (the interrupts are still reported in the interrupt registers of the peripheral). Thus, when a large amount of data is transferred using µDMA, instead of receiving multiple interrupts from the peripheral as data flows, the INTC receives only one interrupt when the transfer is complete. Unmasked peripheral error interrupts continue to be sent to the INTC.

When a µDMA channel generates a completion interrupt, the **CHIS** bit corresponding to the peripheral channel is set in the **DMA Channel Interrupt Status (UDMA_CHIS)** register (see UDMA_CHIS). This register can be used by the interrupt handler code of the peripheral to determine if the interrupt was caused by the µDMA channel or an error event reported by the interrupt registers of the peripheral. The completion interrupt request from the µDMA controller is automatically cleared when the interrupt handler is activated.

If the µDMA controller encounters a bus or memory protection error as it tries to perform a data transfer, it disables the µDMA channel that caused the error and generates an interrupt on the µDMA error interrupt vector. The processor can read the **DMA Bus Error Clear (UDMA_ERRCLR)** register to determine if an error is pending. The **ERRCLR** bit is set if an error occurred. The error can be cleared by setting the **ERRCLR** bit to 1.

Table 10-6 shows the dedicated interrupt assignments for the µDMA controller.

Table 10-6. µDMA Interrupt Assignments

Interrupt	Assignment
46	µDMA software channel transfer
47	µDMA error

10.4 Initialization and Configuration

10.4.1 Module Initialization

Before the µDMA controller can be used, it must be enabled in the system control block and in the peripheral. The location of the channel control structure must also be programmed.

The following steps should be performed one time during system initialization:

1. Enable the µDMA controller by setting the **MASTEREN** bit of the **DMA Configuration (UDMA_CFG)** register.
2. Program the location of the channel control table by writing the base address of the table to the **DMA Channel Control Base Pointer (UDMA_CTLBASE)** register. The base address must be aligned on a 1024-byte boundary.

10.4.2 Configuring a Memory-to-Memory Transfer

µDMA channel 30 is dedicated for software-initiated transfers. However, any channel can be used for software-initiated, memory-to-memory transfer if the associated peripheral is not being used.

10.4.2.1 Configure the Channel Attributes

First, configure the channel attributes:

1. Program bit 30 of the **DMA Channel Priority Set (UDMA_PRIOSET)** or **DMA Channel Priority Clear (UDMA_PRIOCLR)** registers to set the channel to high priority or default priority.
2. Set bit 30 of the **DMA Channel Primary Alternate Clear (UDMA_ALTCLR)** register to select the primary channel control structure for this transfer.

3. Set bit 30 of the **DMA Channel Useburst Clear (UDMA_USEBURSTCLR)** register to allow the µDMA controller to respond to single and burst requests.
4. Set bit 30 of the **DMA Channel Request Mask Clear (UDMA_REQMASKCLR)** register to allow the µDMA controller to recognize requests for this channel.

10.4.2.2 Configure the Channel Control Structure

Now the channel control structure must be configured.

This example transfers 256 words from one memory buffer to another. Channel 30 is used for a software transfer, and the control structure for channel 30 is at offset 0x1E0 of the channel control table. The channel control structure for channel 30 is located at the offsets listed in Table 10-7.

Table 10-7. Channel Control Structure Offsets for Channel 30

Offset	Description
Control Table Base + 0x1E0	Channel 30 source end pointer
Control Table Base + 0x1E4	Channel 30 destination end pointer
Control Table Base + 0x1E8	Channel 30 control word

10.4.2.2.1 Configure the Source and Destination

The source and destination end pointers must be set to the last address for the transfer (inclusive).

1. Program the source end pointer at offset 0x1E0 to the address of the source buffer + 0x3FC.
2. Program the destination end pointer at offset 0x1E4 to the address of the destination buffer + 0x3FC.
3. Program the control word at offset 0x1E8 according to Table 10-8.

Table 10-8. Channel Control Word Configuration for Memory Transfer Example

Field in DMACHCTL	Bits	Value	Description
DSTINC	31:30	2	32-bit destination address increment
DSTSIZE	29:28	2	32-bit destination data size
SRCINC	27:26	2	32-bit source address increment
SRCSIZE	25:24	2	32-bit source data size
Reserved	23:18	0	Reserved
ARBSIZE	17:14	3	Arbitrates after 8 transfers
XFERSIZE	13:4	255	Transfer 256 items
NXTUSEBURST	3	0	N/A for this transfer type
XFERMODE	2:0	2	Use auto-request transfer mode

10.4.2.3 Start the Transfer

Now the channel is configured and is ready to start.

1. Enable the channel by setting bit 30 of the **DMA Channel Enable Set (UDMA_ENASET)** register.
2. Issue a transfer request by setting bit 30 of the **DMA Channel Software Request (UDMA_SWREQ)** register.

The µDMA transfer begins. If the interrupt is enabled, then the processor is notified by interrupt when the transfer completes. If needed, the status can be checked by reading bit 30 of the **UDMA_ENASET** register. This bit is automatically cleared when the transfer completes. The status can also be checked by reading the **XFERMODE** field of the channel control word at offset 0x1E8. This field is automatically cleared at the end of the transfer.

10.4.3 Configuring a Peripheral for Simple Transmit

This example configures the µDMA controller to transmit a buffer of data to a peripheral. The peripheral has a transmit FIFO with a trigger level of 4. The example peripheral uses µDMA channel 7.

10.4.3.1 Configure the Channel Attributes

First, configure the channel attributes:

1. Program bit 7 of the **DMA Channel Priority Set (UDMA_PRIOSET)** or **DMA Channel Priority Clear (UDMA_PRIOCLR)** registers to set the channel to high priority or default priority.
2. Set bit 7 of the **DMA Channel Primary Alternate Clear (UDMA_ALTCLR)** register to select the primary channel control structure for this transfer.
3. Set bit 7 of the **DMA Channel Useburst Clear (UDMA_USEBURSTCLR)** register to allow the µDMA controller to respond to single and burst requests.
4. Set bit 7 of the **DMA Channel Request Mask Clear (UDMA_REQMASKCLR)** register to allow the µDMA controller to recognize requests for this channel.

10.4.3.2 Configure the Channel Control Structure

This example transfers 64 bytes from a memory buffer to the transmit FIFO register of the peripheral using µDMA channel 7. The control structure for channel 7 is at offset 0x070 of the channel control table. The channel control structure for channel 7 is at the offsets listed in Table 10-9.

Table 10-9. Channel Control Structure Offsets for Channel 7

Offset	Description
Control Table Base + 0x070	Channel 7 source end pointer
Control Table Base + 0x074	Channel 7 destination end pointer
Control Table Base + 0x078	Channel 7 control word

10.4.3.2.1 Configure the Source and Destination

The source and destination end pointers must be set to the last address for the transfer (inclusive). Because the peripheral pointer does not change, it simply points to the data register of the peripheral.

1. Program the source end pointer at offset 0x070 to the address of the source buffer + 0x3F.
2. Program the destination end pointer at offset 0x074 to the address of the transmit FIFO register of the peripheral.
3. Program the control word at offset 0x078 according to Table 10-10.

Table 10-10. Channel Control Word Configuration for Peripheral Transmit Example

Field in DMACHCTL	Bits	Value	Description
DSTINC	31:30	3	Destination address does not increment
DSTSIZE	29:28	0	8-bit destination data size
SRCINC	27:26	0	8-bit source address increment
SRCSIZE	25:24	0	8-bit source data size
Reserved	23:18	0	Reserved
ARBSIZE	17:14	2	Arbitrates after 4 transfers
XFERSIZE	13:4	63	Transfer 64 items
NXTUSEBURST	3	0	N/A for this transfer type
XFERMODE	2:0	1	Use basic transfer mode

NOTE: In this example, it is not important if the peripheral makes a single request or a burst request. Because the peripheral has a FIFO that triggers at a level of 4, the arbitration size is set to 4. If the peripheral makes a burst request, then 4 bytes are transferred, which is what the FIFO can accommodate. If the peripheral makes a single request (if there is any space in the FIFO), then 1 byte at a time is transferred. If it is important to the application that only transfers are in bursts, then the Channel Useburst **SET[7]** bit should be set in the **DMA Channel Useburst Set (UDMA_USEBURSTSET)** register.

10.4.3.3 Start the Transfer

Now the channel is configured and is ready to start.

1. Enable the channel by setting bit 7 of the **DMA Channel Enable Set (UDMA_ENASET)** register.

The µDMA controller is now configured for transfer on channel 7. The controller makes transfers to the peripheral whenever the peripheral asserts a µDMA request. The transfers continue until the entire buffer of 64 bytes is transferred. When that happens, the µDMA controller disables the channel and sets the **XFERMODE** field of the channel control word to 0 (stopped). The status of the transfer can be checked by reading bit 7 of the **DMA Channel Enable Set (UDMA_ENASET)** register. This bit is automatically cleared when the transfer completes. The status can also be checked by reading the **XFERMODE** field of the channel control word at offset 0x078. This field is automatically cleared at the end of the transfer.

If peripheral interrupts are enabled, then the peripheral interrupt handler receives an interrupt when the entire transfer completes.

10.4.4 Configuring a Peripheral for Ping-Pong Receive

This example configures the µDMA controller to continuously receive 8-bit data from a peripheral into a pair of 64-byte buffers. The peripheral has a receive FIFO with a trigger level of 8. The example peripheral uses µDMA channel 8.

10.4.4.1 Configure the Channel Attributes

First, configure the channel attributes:

1. Program bit 8 of the **DMA Channel Priority Set (UDMA_PRIOSET)** or **DMA Channel Priority Clear (UDMA_PRIOCLR)** registers to set the channel to high priority or default priority.
2. Set bit 8 of the **DMA Channel Primary Alternate Clear (UDMA_ALTCLR)** register to select the primary channel control structure for this transfer.
3. Set bit 8 of the **DMA Channel Useburst Clear (UDMA_USEBURSTCLR)** register to allow the µDMA controller to respond to single and burst requests.
4. Set bit 8 of the **DMA Channel Request Mask Clear (UDMA_REQMASKCLR)** register to allow the µDMA controller to recognize requests for this channel.

10.4.4.2 Configure the Channel Control Structure

This example transfers bytes from the receive FIFO register of the peripheral into two 64-byte memory buffers. As data is received, when one buffer is full, the µDMA controller switches to use the other.

To use ping-pong buffering, the primary and alternate channel control structures must be used. The primary control structure for channel 8 is at offset 0x080 of the channel control table, and the alternate channel control structure is at offset 0x280. The channel control structures for channel 8 are at the offsets shown in Table 10-11.

Table 10-11. Primary and Alternate Channel Control Structure Offsets for Channel 8

Offset	Description
Control table base + 0x080	Channel 8 primary source end pointer
Control table base + 0x084	Channel 8 primary destination end pointer

Table 10-11. Primary and Alternate Channel Control Structure Offsets for Channel 8 (continued)

Offset	Description
Control table base + 0x088	Channel 8 primary control word
Control table base + 0x280	Channel 8 alternate source end pointer
Control table base + 0x284	Channel 8 alternate destination end pointer
Control table base + 0x288	Channel 8 alternate control word

10.4.4.2.1 Configure the Source and Destination

The source and destination end pointers must be set to the last address for the transfer (inclusive). Because the peripheral pointer does not change, it simply points to the data register of the peripheral. The primary and alternate sets of pointers must be configured.

1. Program the primary source end pointer at offset 0x080 to the address of the receive buffer of the peripheral.
2. Program the primary destination end pointer at offset 0x084 to the address of ping-pong buffer A + 0x3F.
3. Program the alternate source end pointer at offset 0x280 to the address of the receive buffer of the peripheral.
4. Program the alternate destination end pointer at offset 0x284 to the address of ping-pong buffer B + 0x3F.

The primary control word at offset 0x088 and the alternate control word at offset 0x288 are initially programmed the same way.

1. Program the primary channel control word at offset 0x088 according to Table 10-12.
2. Program the alternate channel control word at offset 0x288 according to Table 10-12.

Table 10-12. Channel Control Word Configuration for Peripheral Ping-Pong Receive Example

Field in DMACHCTL	Bits	Value	Description
DSTINC	31:30	0	8-bit destination address increment
DSTSIZE	29:28	0	8-bit destination data size
SRCINC	27:26	3	Source address does not increment
SRCSIZE	25:24	0	8-bit source data size
reserved	23:18	0	Reserved
ARBSIZE	17:14	3	Arbitrates after 8 transfers
XFERSIZE	13:4	63	Transfer 64 items
NXTUSEBURST	3	0	N/A for this transfer type
XFERMODE	2:0	3	Use ping-pong transfer mode

NOTE: In this example, it is not important if the peripheral makes a single request or a burst request. Because the peripheral has a FIFO that triggers at a level of 8, the arbitration size is set to 8. If the peripheral makes a burst request, then 8 bytes are transferred, which is what the FIFO can accommodate. If the peripheral makes a single request (if there is any data in the FIFO), then 1 byte at a time is transferred. If it is important to the application that only transfers are made in bursts, then the Channel Useburst **SET[8]** bit should be set in the **DMA Channel Useburst Set (DMAUSEBURSTSET)** register.

10.4.4.3 Configure the Peripheral Interrupt

An interrupt handler should be configured when using µDMA ping-pong mode, it is best to use an interrupt handler. However, ping-pong mode can be configured without interrupts by polling. The interrupt handler is triggered after each buffer completes.

1. Configure and enable an interrupt handler for the peripheral.

10.4.4.4 Enable the µDMA Channel

Now the channel is configured and is ready to start.

1. Enable the channel by setting bit 8 of the **DMA Channel Enable Set (DMAENASET)** register.

10.4.4.5 Process Interrupts

The µDMA controller is now configured and enabled for transfer on channel 8. When the peripheral asserts the µDMA request signal, the µDMA controller makes transfers into buffer A using the primary channel control structure. When the primary transfer to buffer A completes, it switches to the alternate channel control structure and makes transfers into buffer B. At the same time, the primary channel control word mode field is configured to indicate stopped, and an interrupt is triggered.

When an interrupt is triggered, the interrupt handler must determine which buffer is complete and process the data or set a flag that the data must be processed by noninterrupt buffer processing code. Then the next buffer transfer must be set up.

In the interrupt handler:

1. Read the primary channel control word at offset 0x088 and check the **XFERMODE** field. If the field is 0, this means buffer A is complete. If buffer A is complete, then:
   (a) Process the newly received data in buffer A or signal the buffer processing code that buffer A has data available.
   (b) Reprogram the primary channel control word at offset 0x88 according to Table 10-12.
2. Read the alternate channel control word at offset 0x288 and check the **XFERMODE** field. If the field is 0, this means buffer B is complete. If buffer B is complete, then:
   (a) Process the newly received data in buffer B or signal the buffer processing code that buffer B has data available.
   (b) Reprogram the alternate channel control word at offset 0x288 according to Table 10-12.

10.4.5 Configuring Channel Assignments

Channel assignments for each µDMA channel can be changed using the **CHMAPn** registers. Each 4-bit field represents a µDMA channel. If the bit is set, then the secondary function is used for the channel.

See Table 10-1 for channel assignments.

For example, to use UART1 Receive on channel 8, configure the **CH8SEL** bit in the **CHMAP1** register to be 0x1.

10.5 µDMA Registers

10.5.1 UDMA Registers

10.5.1.1 UDMA Registers Mapping Summary

This section provides information on the UDMA module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 10-13. UDMA Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
UDMA_STAT	RO	32	0x001F 0000	0x000	0x400F F000
UDMA_CFG	WO	32	0x0000 0000	0x004	0x400F F004

Table 10-13. UDMA Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
UDMA_CTLBASE	RW	32	0x0000 0000	0x008	0x400F F008
UDMA_ALTBASE	RO	32	0x0000 0200	0x00C	0x400F F00C
UDMA_WAITSTAT	RO	32	0x0FFF FFC0	0x010	0x400F F010
UDMA_SWREQ	WO	32	0x0000 0000	0x014	0x400F F014
UDMA_USEBURSTSET	RW	32	0x0000 0000	0x018	0x400F F018
UDMA_USEBURSTCLR	WO	32	0x0000 0000	0x01C	0x400F F01C
UDMA_REQMASKSET	RW	32	0x0000 0000	0x020	0x400F F020
UDMA_REQMASKCLR	WO	32	0x0000 0000	0x024	0x400F F024
UDMA_ENASET	RW	32	0x0000 0000	0x028	0x400F F028
UDMA_ENACLR	WO	32	0x0000 0000	0x02C	0x400F F02C
UDMA_ALTSET	RW	32	0x0000 0000	0x030	0x400F F030
UDMA_ALTCLR	WO	32	0x0000 0000	0x034	0x400F F034
UDMA_PRIOSET	RW	32	0x0000 0000	0x038	0x400F F038
UDMA_PRIOCLR	WO	32	0x0000 0000	0x03C	0x400F F03C
UDMA_ERRCLR	RW	32	0x0000 0000	0x04C	0x400F F04C
UDMA_CHASGN	RW	32	0x0000 0000	0x500	0x400F F500
UDMA_CHIS	RW	32	0x0000 0000	0x504	0x400F F504
UDMA_CHMAP0	RW	32	0x0000 0000	0x510	0x400F F510
UDMA_CHMAP1	RW	32	0x0000 0000	0x514	0x400F F514
UDMA_CHMAP2	RW	32	0x0000 0000	0x518	0x400F F518
UDMA_CHMAP3	RW	32	0x0000 0000	0x51C	0x400F F51C

10.5.1.2 UDMA Register Descriptions

UDMA_STAT

Address offset	0x000		
Physical Address	0x400F F000	**Instance**	UDMA
Description	DMA status The STAT register returns the status of the uDMA controller. This register cannot be read when the uDMA controller is in the reset state.		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED								DMACHANS								RESERVED								STATE				RESERVED			MASTEN

Bits	Field Name	Description	Type	Reset
31:21	RESERVED	This bit field is reserved.	RO	0x000
20:16	DMACHANS	Available uDMA channels minus 1 This field contains a value equal to the number of uDMA channels the uDMA controller is configured to use, minus one. The value of 0x1F corresponds to 32 uDMA channels.	RO	0x1F
15:8	RESERVED	This bit field is reserved.	RO	0x00
7:4	STATE	Control state machine status This field shows the current status of the control state-machine. Status can be one of the following: 0x0: Idle 0x1: Reading channel controller data 0x2: Reading source end pointer 0x3: Reading destination end pointer 0x4: Reading source data 0x5: Writing destination data 0x6: Waiting for uDMA request to clear 0x7: Writing channel controller data 0x8: Stalled 0x9: Done 0xA-0xF: Undefined	RO	0x0
3:1	RESERVED	This bit field is reserved.	RO	0x0
0	MASTEN	Master enable status 0: The uDMA controller is disabled. 1: The uDMA controller is enabled.	RO	0

UDMA_CFG

Address offset 0x004

Physical Address 0x400F F004 **Instance** UDMA

Description DMA configuration
The CFG register controls the configuration of the uDMA controller.

Type WO

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

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	WO	0x0000 0000
0	MASTEN	Controller master enable 0: Disables the uDMA controller. 1: Enables the uDMA controller.	WO	0

UDMA_CTLBASE

Address offset 0x008

Physical Address 0x400F F008 **Instance** UDMA

Description DMA channel control base pointer
The CTLBASE register must be configured so that the base pointer points to a location in system memory. The amount of system memory that must be assigned to the uDMA controller depends on the number of uDMA channels used and whether the alternate channel control data structure is used. See Section 10.2.5 for details about the Channel Control Table. The base address must be aligned on a 1024-byte boundary. This register cannot be read when the uDMA controller is in the reset state.

Type RW

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

Bits	Field Name	Description	Type	Reset
31:10	ADDR	Channel control base address This field contains the pointer to the base address of the channel control table. The base address must be 1024-byte alligned.	RW	0x00 0000
9:0	RESERVED	This bit field is reserved.	RO	0x000

UDMA_ALTBASE

Address offset 0x00C

Physical Address 0x400F F00C **Instance** UDMA

Description DMA alternate channel control base pointer
The ALTBASE register returns the base address of the alternate channel control data. This register removes the necessity for application software to calculate the base address of the alternate channel control structures. This register cannot be read when the uDMA controller is in the reset state.

Type RO

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

Bits	Field Name	Description	Type	Reset
31:0	ADDR	Alternate channel address pointer This field provides the base address of the alternate channel control structures.	RO	0x0000 0200

UDMA_WAITSTAT

Address offset 0x010

Physical Address 0x400F F010 **Instance** UDMA

Description DMA channel wait-on-request status
This read-only register indicates that the uDMA channel is waiting on a request. A peripheral can hold off the uDMA from performing a single request until the peripheral is ready for a burst request to enhance the uDMA performance. The use of this feature is dependent on the design of the peripheral and is not controllable by software in any way. This register cannot be read when the uDMA controller is in the reset state.

Type RO

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

Bits	Field Name	Description	Type	Reset
31:0	WAITREQ	Channel [n] wait status These bits provide the tchannel wait-on-request status. Bit 0 corresponds to channel 0. 1: The corresponding channel is waiting on a request. 0: The corresponding channel is not waiting on a request.	RO	0x0FFF FFC0

UDMA_SWREQ

Address offset 0x014

Physical Address 0x400F F014 **Instance** UDMA

Description DMA channel software request
Each bit of the SWREQ register represents the corresponding uDMA channel. Setting a bit generates a request for the specified uDMA channel.

Type WO

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

Bits	Field Name	Description	Type	Reset
31:0	SWREQ	Channel [n] software request These bits generate software requests. Bit 0 corresponds to channel 0. 1: Generate a software request for the corresponding channel 0: No request generated These bits are automatically cleared when the software request has been completed.	WO	0x0000 0000

UDMA_USEBURSTSET

Address offset	0x018		
Physical Address	0x400F F018	**Instance**	UDMA
Description	DMA channel useburst set Each bit of the USEBURSTSET register represents the corresponding uDMA channel. Setting a bit disables the channel single request input from generating requests, configuring the channel to only accept burst requests. Reading the register returns the status of USEBURST. If the amount of data to transfer is a multiple of the arbitration (burst) size, the corresponding SET[n] bit is cleared after completing the final transfer. If there are fewer items remaining to transfer than the arbitration (burst) size, the uDMA controller automatically clears the corresponding SET[n] bit, allowing the remaining items to transfer using single requests. To resume transfers using burst requests, the corresponding bit must be set again. A bit must not be set if the corresponding peripheral does not support the burst request model.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	SET	Channel [n] useburst set 0: uDMA channel [n] responds to single or burst requests. 1: uDMA channel [n] responds only to burst requests. Bit 0 corresponds to channel 0. This bit is automatically cleared as described above. A bit can also be manually cleared by setting the corresponding CLR[n] bit in the DMAUSEBURSTCLR register.	RW	0x0000 0000

UDMA_USEBURSTCLR

Address offset	0x01C		
Physical Address	0x400F F01C	**Instance**	UDMA
Description	DMA channel useburst clear Each bit of the USEBURSTCLR register represents the corresponding uDMA channel. Setting a bit clears the corresponding SET[n] bit in the USEBURSTSET register.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	CLR	Channel [n] useburst clear 0: No effect 1: Setting a bit clears the corresponding SET[n] bit in the DMAUSEBURSTSET register meaning that uDMA channel [n] responds to single and burst requests.	WO	0x0000 0000

UDMA_REQMASKSET

Address offset	0x020		
Physical Address	0x400F F020	**Instance**	UDMA
Description	DMA channel request mask set Each bit of the REQMASKSET register represents the corresponding uDMA channel. Setting a bit disables uDMA requests for the channel. Reading the register returns the request mask status. When a uDMA channel request is masked, that means the peripheral can no longer request uDMA transfers. The channel can then be used for software-initiated transfers.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	SET	Channel [n] request mask set 0: The peripheral associated with channel [n] is enabled to request uDMA transfers 1: The peripheral associated with channel [n] is not able to request uDMA transfers. Channel [n] may be used for software-initiated transfers. Bit 0 corresponds to channel 0. A bit can only be cleared by setting the corresponding CLR[n] bit in the DMAREQMASKCLR register.	RW	0x0000 0000

UDMA_REQMASKCLR

Address offset	0x024		
Physical Address	0x400F F024	**Instance**	UDMA
Description	DMA channel request mask clear Each bit of the REQMASKCLR register represents the corresponding uDMA channel. Setting a bit clears the corresponding SET[n] bit in the REQMASKSET register.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	CLR	Channel [n] request mask clear 0: No effect 1: Setting a bit clears the corresponding SET[n] bit in the DMAREQMASKSET register meaning that the peripheral associated with channel [n] is enabled to request uDMA transfers.	WO	0x0000 0000

UDMA_ENASET

Address offset	0x028		
Physical Address	0x400F F028	**Instance**	UDMA
Description	DMA channel enable set Each bit of the ENASET register represents the corresponding uDMA channel. Setting a bit enables the corresponding uDMA channel. Reading the register returns the enable status of the channels. If a channel is enabled but the request mask is set (REQMASKSET), then the channel can be used for software-initiated transfers.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	SET	Channel [n] enable set 0: uDMA channel [n] is disabled 1: uDMA channel [n] is enabled Bit 0 corresponds to channel 0. A bit can only be cleared by setting the corresponding CLR[n] bit in the DMAENACLR register.	RW	0x0000 0000

UDMA_ENACLR

Address offset 0x02C

Physical Address 0x400F F02C **Instance** UDMA

Description DMA channel enable clear
Each bit of the ENACLR register represents the corresponding uDMA channel. Setting a bit clears the corresponding SET[n] bit in the ENASET register.

Type WO

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

Bits	Field Name	Description	Type	Reset
31:0	CLR	Channel [n] enable clear 0: No effect 1: Setting a bit clears the corresponding SET[n] bit in the DMAENASET register meaning that channel [n] is disabled for uDMA transfers. Note: The controller disables a channel when it completes the uDMA cycle.	WO	0x0000 0000

UDMA_ALTSET

Address offset 0x030

Physical Address 0x400F F030 **Instance** UDMA

Description DMA channel primary alternate set
Each bit of the ALTSET register represents the corresponding uDMA channel. Setting a bit configures the uDMA channel to use the alternate control data structure. Reading the register returns the status of which control data structure is in use for the corresponding uDMA channel.

Type RW

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

Bits	Field Name	Description	Type	Reset
31:0	SET	Channel [n] alternate set 0: uDMA channel [n] is using the primary control structure 1: uDMA channel [n] is using the alternate control structure Bit 0 corresponds to channel 0. A bit can only be cleared by setting the corresponding CLR[n] bit in the DMAALTCLR register. Note: For Ping-Pong and Scatter-Gather cycle types, the uDMA controller automatically sets these bits to select the alternate channel control data structure.	RW	0x0000 0000

UDMA_ALTCLR

Address offset 0x034

Physical Address 0x400F F034 **Instance** UDMA

Description DMA channel primary alternate clear
Each bit of the ALTCLR register represents the corresponding uDMA channel. Setting a bit clears the corresponding SET[n] bit in the ALTSET register.

Type WO

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

Bits	Field Name	Description	Type	Reset
31:0	CLR	Channel [n] alternate clear 0: No effect 1: Setting a bit clears the corresponding SET[n] bit in the DMAALTSET register meaning that channel [n] is using the primary control structure. Note: For Ping-Pong and Scatter-Gather cycle types, the uDMA controller automatically sets these bits to select the alternate channel control data structure.	WO	0x0000 0000

UDMA_PRIOSET

Address offset	0x038		
Physical Address	0x400F F038	**Instance**	UDMA
Description	DMA channel priority set Each bit of the PRIOSET register represents the corresponding uDMA channel. Setting a bit configures the uDMA channel to have a high priority level. Reading the register returns the status of the channel priority mask.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	SET	Channel [n] priority set 0: uDMA channel [n] is using the default priority level 1: uDMA channel [n] is using a high priority level Bit 0 corresponds to channel 0. A bit can only be cleared by setting the corresponding CLR[n] bit in the DMAPRIOCLR register.	RW	0x0000 0000

UDMA_PRIOCLR

Address offset	0x03C		
Physical Address	0x400F F03C	**Instance**	UDMA
Description	DMA channel priority clear Each bit of the DMAPRIOCLR register represents the corresponding uDMA channel. Setting a bit clears the corresponding SET[n] bit in the PRIOSET register.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	CLR	Channel [n] priority clear 0: No effect 1: Setting a bit clears the corresponding SET[n] bit in the DMAPRIOSET register meaning that channel [n] is using the default priority level.	WO	0x0000 0000

UDMA_ERRCLR

Address offset	0x04C		
Physical Address	0x400F F04C	**Instance**	UDMA
Description	DMA bus error clear The ERRCLR register is used to read and clear the uDMA bus error status. The error status is set if the uDMA controller encountered a bus error while performing a transfer. If a bus error occurs on a channel, that channel is automatically disabled by the uDMA controller. The other channels are unaffected.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	RO	0x0000 0000
0	ERRCLR	uDMA bus error status 0: No bus error is pending 1: A bus error is pending This bit is cleared by writing 1 to it.	RW	0

UDMA_CHASGN

Address offset	0x500		
Physical Address	0x400F F500	**Instance**	UDMA
Description	DMA channel assignment Each bit of the CHASGN register represents the corresponding uDMA channel. Setting a bit selects the secondary channel assignment as specified in the section "Channel Assignments"		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	CHASGN	Channel [n] assignment select 0: Use the primary channel assignment 1: Use the secondary channel assignment	RW	0x0000 0000

UDMA_CHIS

Address offset	0x504		
Physical Address	0x400F F504	**Instance**	UDMA
Description	DMA channel interrupt status Each bit of the CHIS register represents the corresponding uDMA channel. A bit is set when that uDMA channel causes a completion interrupt. The bits are cleared by writing 1.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	CHIS	Channel [n] interrupt status 0: The corresponding uDMA channel has not caused an interrupt. 1: The corresponding uDMA channel has caused an interrupt. This bit is cleared by writing 1 to it.	RW	0x0000 0000

UDMA_CHMAP0

Address offset	0x510		
Physical Address	0x400F F510	**Instance**	UDMA
Description	DMA channel map select 0 Each 4-bit field of the CHMAP0 register configures the uDMA channel assignment as specified in the uDMA channel assignment table in the "Channel Assignments" section.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH7SEL				CH6SEL				CH5SEL				CH4SEL				CH3SEL				CH2SEL				CH1SEL				CH0SEL			

Bits	Field Name	Description	Type	Reset
31:28	CH7SEL	uDMA channel 7 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
27:24	CH6SEL	uDMA channel 6 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
23:20	CH5SEL	uDMA channel 5 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
19:16	CH4SEL	uDMA channel 4 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
15:12	CH3SEL	uDMA channel 3 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
11:8	CH2SEL	uDMA channel 2 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
7:4	CH1SEL	uDMA channel 1 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
3:0	CH0SEL	uDMA channel 0 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0

UDMA_CHMAP1

Address offset	0x514		
Physical Address	0x400F F514	**Instance**	UDMA
Description	DMA channel map select 1 Each 4-bit field of the CHMAP1 register configures the uDMA channel assignment as specified in the uDMA channel assignment table in the "Channel Assignments" section.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH15SEL				CH14SEL				CH13SEL				CH12SEL				CH11SEL				CH10SEL				CH9SEL				CH8SEL			

Bits	Field Name	Description	Type	Reset
31:28	CH15SEL	uDMA channel 15 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
27:24	CH14SEL	uDMA channel 14 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
23:20	CH13SEL	uDMA channel 13 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0

Bits	Field Name	Description	Type	Reset
19:16	CH12SEL	uDMA channel 12 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
15:12	CH11SEL	uDMA channel 11 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
11:8	CH10SEL	uDMA channel 10 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
7:4	CH9SEL	uDMA channel 9 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
3:0	CH8SEL	uDMA channel 8 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0

UDMA_CHMAP2

Address offset	0x518		
Physical Address	0x400F F518	**Instance**	UDMA
Description	DMA channel map select 2 Each 4-bit field of the CHMAP2 register configures the uDMA channel assignment as specified in the uDMA channel assignment table in the "Channel Assignments" section.		
Type	RW		

31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
CH23SEL	CH22SEL	CH21SEL	CH20SEL	CH19SEL	CH18SEL	CH17SEL	CH16SEL

Bits	Field Name	Description	Type	Reset
31:28	CH23SEL	uDMA channel 23 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
27:24	CH22SEL	uDMA channel 22 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
23:20	CH21SEL	uDMA channel 21 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
19:16	CH20SEL	uDMA channel 20 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
15:12	CH19SEL	uDMA channel 19 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
11:8	CH18SEL	uDMA channel 18 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
7:4	CH17SEL	uDMA channel 17 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
3:0	CH16SEL	uDMA channel 16 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0

UDMA_CHMAP3

Address offset	0x51C		
Physical Address	0x400F F51C	**Instance**	UDMA
Description	DMA channel map select 3 Each 4-bit field of the CHMAP3 register configures the uDMA channel assignment as specified in the uDMA channel assignment table in the "Channel Assignments" section.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH31SEL				CH30SEL				CH29SEL				CH28SEL				CH27SEL				CH26SEL				CH25SEL				CH24SEL			

Bits	Field Name	Description	Type	Reset
31:28	CH31SEL	uDMA channel 31 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
27:24	CH30SEL	uDMA channel 30 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
23:20	CH29SEL	uDMA channel 29 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
19:16	CH28SEL	uDMA channel 28 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
15:12	CH27SEL	uDMA channel 27 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
11:8	CH26SEL	uDMA channel 26 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
7:4	CH25SEL	uDMA channel 25 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0
3:0	CH24SEL	uDMA channel 24 source select See section titled "Channel Assignments" in Micro Direct Memory Access chapter.	RW	0x0

General-Purpose Timers

This chapter describes the general-purpose timers.

Topic **Page**

11.1 General-Purpose Timers

Programmable timers can be used to count or time external events that drive the timer input pins. The CC2538 General-Purpose Timer Module (GPTM) contains GPTM blocks. Each GPTM block provides two 16-bit timers (referred to as timer A and timer B) that can be configured to operate independently as timers, or concatenated to operate as one 32-bit timer.

The GPT is one timing resource available on the CC2538 microcontroller. Other timer resources include the System Timer (SysTick) (see Section 3.2.1), MAC timer, sleep timer, and watchdog timer, see the respective chapters for reference.

The GPTM contains four GPTM blocks with the following functional options:

- Operating modes:
  - 16- or 32-bit programmable one-shot timer
  - 16- or 32-bit programmable periodic timer
  - 16-bit general-purpose timer with an 8-bit prescaler
- Count up or down
- Daisy-chaining of timer modules to allow a single timer to initiate multiple timing events
- Timer synchronization allows selected timers to start counting on the same clock cycle
- User-enabled stalling when the microcontroller asserts CPU Halt flag during debug
- Ability to determine the elapsed time between the assertion of the timer interrupt and entry into the interrupt service routine.

11.2 Block Diagram

Figure 11-1 shows the GPTM module block diagram.

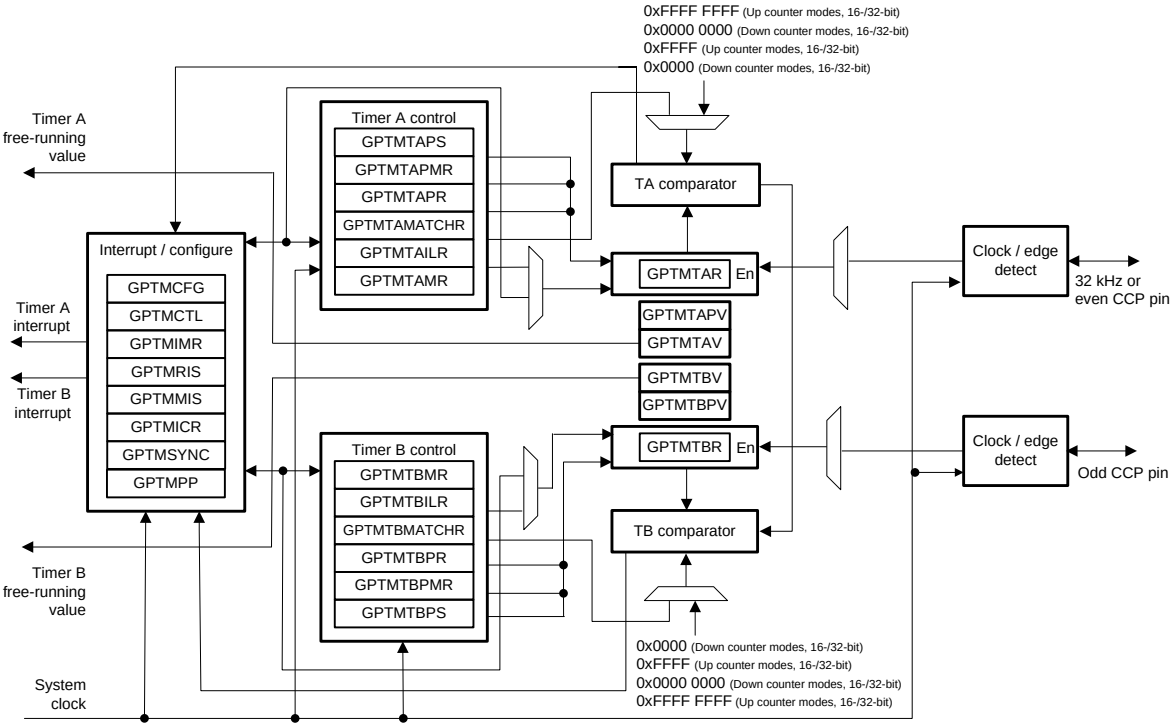


Figure 11-1. GPTM Module Block Diagram

11.3 Functional Description

The main components of each GPTM block are two free-running up and down counters (referred to as timer A and timer B), two match registers, two prescaler match registers, two shadow registers, and two load and initialization registers and their associated control functions. The exact function of each GPTM is controlled by software and configured through the register interface. Timer A and timer B can be used individually, in which case they have a 16-bit counting range. In addition, timer A and timer B can be concatenated to provide a 32-bit counting range. The prescaler can be used only when the timers are used individually.

Table 11-1 lists the available modes for each GPTM block. When counting down in one-shot or periodic modes, the prescaler acts as a true prescaler and contains the least-significant bits (LSBs) of the count. When counting up in one-shot or periodic modes, the prescaler acts as a timer extension and holds the most-significant bits (MSBs) of the count. In input edge count, input edge time, and PWM mode, the prescaler always acts as a timer extension, regardless of the count direction.

Table 11-1. General-Purpose Timer Capabilities

Mode	Timer Use	Count Direction	Counter Size	Prescaler Size[1]	Prescaller Behavior (Count Direction)
One-Shot	Individual	Up or Down	16-bit	8-bit	Timer Extension (Up), Prescaler (Down)
	Concatenated	Up or Down	32-bit	-	N/A
Periodic	Individual	Up or Down	16-bit	8-bit	Timer Extension (Up), Prescaler (Down)
	Concatenated	Up or Down	32-bit	-	N/A
-	-	-	-	-	-
Edge Count	Individual	Up or Down	16-bit	8-bit	Timer Extension (Both)
Edge Time	Individual	Up or Down	16-bit	8-bit	Timer Extension (Both)
PWM	Individual	Down	16-bit	8-bit	Timer Extension

(1) The prescaler is available only when the timers are used individually.

Software configures the GPTM using the **GPTM Configuration (GPTIMER_CFG)** register (see GPTIMER_CFG), the **GPTM Timer A Mode (GPTIMER_TAMR)** register (see GPTIMER_TAMR), and the **GPTM Timer B Mode (GPTIMER_TBMR)** register (see GPTIMER_TBMR). When in one of the concatentated modes, timer A and timer B can operate in one mode only. However, when configured in an individual mode, timer A and timer B can be independently configured in any combination of the individual modes.

11.3.1 GPTM Reset Conditions

After reset has been applied to the GPTM, the module is in an inactive state, and all control registers are cleared and in their default states. Counters Timer A and Timer B are initialized to all 1s, along with their corresponding load registers: the **GPTM Timer A Interval Load (GPTIMER_TAILR)** register (see GPTIMER_TAILR) and the **GPTM Timer B Interval Load (GPTIMER_TBILR)** register (see GPTIMER_TBILR). The prescale counters are initialized to 0x00:

- The **GPTM Timer A Prescale (GPTIMER_TAPR)** register (see GPTIMER_TAPR) and the **GPTM Timer B Prescale (GPTIMER_TBPR)** register (see GPTIMER_TBPR)
- The **GPTM Timer A Prescale Snapshot (GPTIMER_TAPS)** register (see GPTIMER_TAPS) and the **GPTM Timer B Prescale Snapshot (GPTIMER_TBPS)** register (see GPTIMER_TBPS)
- The **GPTM Timer A Prescale Value (GPTIMER_TAPV)** register (see GPTIMER_TAPV) and the **GPTM Timer B Prescale Value (GPTIMER_TBPV)** register (see GPTIMER_TBPV).

11.3.2 Timer Modes

This section describes the operation of the various timer modes. When using timer A and timer B in concatenated mode, only the timer A control and status bits must be used; there is no need to use timer B control and status bits. The GPTM is placed into individual or split mode by writing a value of 0x4 to the **GPTM Configuration (GPTIMER_CFG)** register (see GPTIMER_CFG). In the following sections, the variable "**n**" is used in bit field and register names to imply either a timer A function or a timer B function. Throughout this section, the time-out event in down-count mode is 0x0; in up-count mode the time-out event is the value in the **GPTM Interval Load (GPTIMER_TnILR)** and the optional **GPTM Timer n Prescale (GPTIMER_TnPR)** registers.

11.3.2.1 One-Shot or Periodic Timer Mode

The selection of one-shot or periodic mode is determined by the value written to the **TnMR** field of the **GPTM Timer n Mode (GPTIMER_TnMR)** register (see GPTIMER_TAMR). The timer is configured to count up or down using the **TnCDIR** bit in the **GPTIMER_TnMR** register.

When software sets the **TnEN** bit in the **GPTM Control (GPTIMER_CTL)** register (see GPTIMER_CFG), the timer begins counting up from 0x0 or down from its preloaded value. Alternatively, if the **TnWOT** bit is set in the **GPTIMER_TnMR** register, once the **TnEN** bit is set, the timer waits for a trigger to begin counting (see Section 11.3.3, *Wait-for-Trigger Mode*).

When the timer is counting down and it reaches the time-out event (0x0), the timer reloads its start value from the **GPTIMER_TnILR** and **GPTIMER_TnPR** registers on the next cycle. When the timer is counting up and it reaches the time-out event (the value in the **GPTIMER_TnILR** and the optional **GPTIMER_TnPR** registers), the timer reloads with 0x0. If configured to be a one-shot timer, the timer stops counting and clears the **TnEN** bit in the **GPTIMER_CTL** register. If configured as a periodic timer, the timer starts counting again on the next cycle. In periodic, snap-shot mode (the **TnMR** field is 0x2 and the **TnSNAPS** bit is set in the **GPTIMER_TnMR** register), the actual free-running value of the timer at the time-out event is loaded into the **GPTIMER_TnR** register. In this manner, software can determine the time elapsed from the interrupt assertion to the ISR entry by examing the snapshot values and the current value of the free-running timer, which is stored in the **GPTIMER_TnV** register. Snapshot mode is not available when the timer is configured in one-shot mode.

In addition to reloading the count value, the GPTM generates interrupts and triggers when it reaches the time-out event. The GPTM sets the **TnTORIS** bit in the **GPTM Raw Interrupt Status (GPTIMER_RIS)** register (see GPTIMER_RIS), and holds it until it is cleared by writing the **GPTM Interrupt Clear (GPTIMER_ICR)** register (see GPTIMER_ICR). If the time-out interrupt is enabled in the **GPTM Interrupt Mask (GPTIMER_IMR)** register (see GPTIMER_IMR), the GPTM also sets the **TnTOMIS** bit in the **GPTM Masked Interrupt Status (GPTIMER_MIS)** register (see GPTIMER_MIS). By setting the **TnMIE** bit in the **GPTIMER_TnMR** register, an interrupt condition can also be generated when the timer value equals the value loaded into the **GPTM Timer n Match (GPTIMER_TnMATCHR)** and **GPTM Timer n Prescale Match (GPTIMER_TnPMR)** registers. This interrupt has the same status, masking, and clearing functions as the time-out interrupt, but uses the match interrupt bits instead (for example, the raw interrupt status is monitored through **TnMRIS** bit in the **GPTM Raw Interrupt Status [GPTIMER_RIS]** register). The interrupt status bits are not updated by the hardware unless the **TnMIE** bit in the **GPTIMER_TnMR** register is set, which is different than the behavior for the time-out interrupt.

If software updates the **GPTIMER_TnILR** or **GPTIMER_TnPR** register while the counter is counting down, the counter loads the new value on the next clock cycle and continues counting from the new value if the **TnILD** bit in the **GPTIMER_TnMR** register is clear. If the **TnILD** bit is set, the counter loads the new value after the next time-out. If software updates the **GPTIMER_TnILR** or the **GPTIMER_TnPR** register while the counter is counting up, the time-out event is changed on the next cycle to the new value. If software updates the **GPTM Timer n Value (GPTIMER_TnV)** register while the counter is counting up or down, the counter loads the new value on the next clock cycle and continues counting from the new value. If software updates the **GPTIMER_TnMATCHR** or the **GPTIMER_TnPMR** register while the counter is counting, the match registers reflect the new values on the next clock cycle if the **TnMRSU** bit in the **GPTIMER_TnMR** register is clear. If the **TnMRSU** bit is set, the new value does not take effect until the next time-out.

If the **TnSTALL** bit in the **GPTIMER_CTL** register is set, the timer freezes counting while the processor is halted by the debugger. The timer resumes counting when the processor resumes execution.

Table 11-2 shows a variety of configurations for a 16-bit free-running timer while using the prescaler. All values assume a 16-MHz clock with Tc =62.5 ns (clock period). The prescaler can only be used when a 16 or 32-bit timer is configured in 16-bit mode.

Table 11-2. 16-Bit Timer With Prescaler Configurations

Prescale (8-bit value)	# of Timer Clocks (Tc) (1)	Maximum Time	Units
00000000	1	4.1	ms
00000001	2	8.2	ms
00000010	3	12.3	ms
----------	–	–	–
11111101	254	1040.4	ms
11111110	255	1045.5	ms
11111111	256	1049.6	ms

(1) Tc is the clock period.

11.3.2.2 Input Edge-Count Mode

NOTE: For rising-edge detection, the input signal must be High for at least two system clock periods following the rising edge. Similarly, for falling-edge detection, the input signal must be Low for at least two system clock periods following the falling edge. Based on this criteria, the maximum input frequency for edge detection is 1/4 of the system frequency.

In Edge-Count mode, the timer is configured as a 16-bit or 32-bit up- or down-counter including the optional prescaler with the upper count value stored in the **GPTM Timer n Prescale (GPTIMER_TnPR)** register and the lower bits in the **GPTIMER_TnR** register. In this mode, the timer is capable of capturing three types of events: rising edge, falling edge, or both. To place the timer in Edge-Count mode, the **TnCMR** bit of the **GPTIMER_TnMR** register must be cleared. The type of edge that the timer counts is determined by the **TnEVENT** fields of the **GPTIMER_CTL** register. During initialization in down-count mode, the **GPTIMER_TnMATCHR** and **GPTIMER_TnPMR** registers are configured so that the difference between the value in the **GPTIMER_TnILR** and **GPTIMER_TnPR** registers and the **GPTIMER_TnMATCHR** and **GPTIMER_TnPMR** registers equals the number of edge events that must be counted. In up-count mode, the timer counts from 0x0 to the value in the **GPTIMER_TnMATCHR** and **GPTIMER_TnPMR** registers. Table 11-3 shows the values that are loaded into the timer registers when the timer is enabled.

Table 11-3. Counter Values When the Timer is Enabled in Input Edge-Count Mode

Register	Count Down Mode	CountUp Mode
GPTIMER_TnR	**GPTM_TnILR**	0x0
GPTIMER_TnV	**GPTM_TnILR**	0x0
GPTIMER_TnPV	**GPTM_TnPR**	0x0

When software writes the TnEN bit in the **GPTM Control (GPTIMER_CTL)** register, the timer is enabled for event capture. Each input event on the CCP pin decrements or increments the counter by 1 until the event count matches **GPTIMER_TnMATCHR** and **GPTIMER_TnPMR**. When the counts match, the GPTM asserts the **CnMRIS** bit in the **GPTM Raw Interrupt Status (GPTIMER_RIS)** register, and holds it until it is cleared by writing the **GPTM Interrupt Clear (GPTMICR)** register. If the capture mode match interrupt is enabled in the **GPTM Interrupt Mask (GPTIMER_IMR)** register, the GPTM also sets the **CnMMIS** bit in the **GPTM Masked Interrupt Status (GPTIMER_MIS)** register. In this mode, the **GPTIMER_TnR** register holds the count of the input events while the **GPTIMER_TnV** and **GPTIMER_TnPV** registers hold the free-running timer value and the free-running prescaler value.

In addition to generating interrupts, an ADC and/or a µDMA trigger can be generated. The ADC trigger is enabled by setting the **TnOTE** bit in **GPTMCTL**.The µDMA trigger is enabled by configuring and enabling the appropriate µDMA channel.

After the match value is reached in down-count mode, the counter is then reloaded using the value in **GPTIMER_TnILR** and **GPTIMER_TnPR** registers, and stopped because the GPTM automatically clears the **TnEN** bit in the **GPTIMER_CTL** register. Once the event count has been reached, all further events are ignored until **TnEN** is re-enabled by software. In up-count mode, the timer is reloaded with 0x0 and continues counting.

Figure 11-2 shows how Input Edge-Count mode works. In this case, the timer start value is set to **GPTIMER_TnILR** =0x000A and the match value is set to **GPTIMER_TnMATCHR** =0x0006 so that four edge events are counted. The counter is configured to detect both edges of the input signal.

Note that the last two edges are not counted because the timer automatically clears the **TnEN** bit after the current count matches the value in the **GPTIMER_TnMATCHR** register.

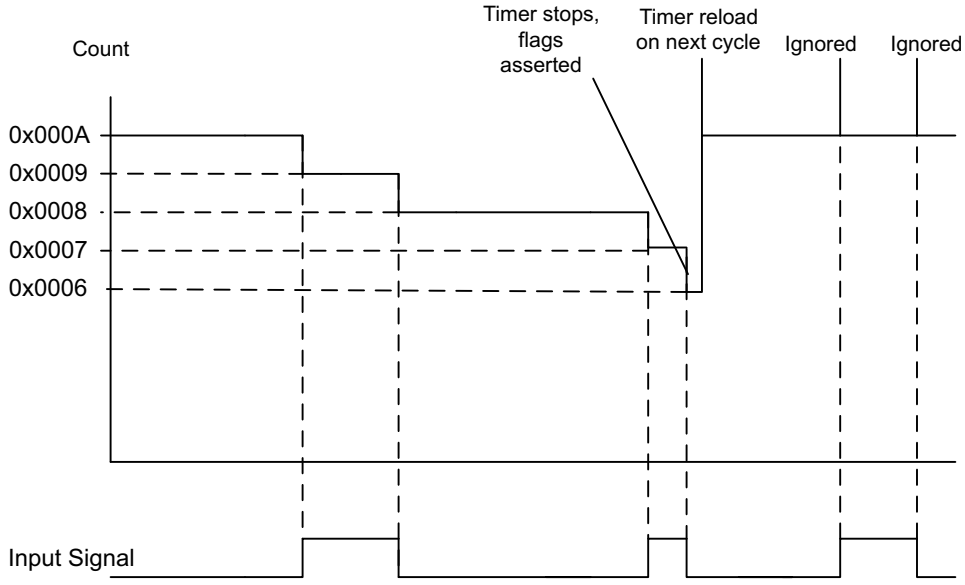


Figure 11-2. Input Edge-Count Mode Example, Counting Down

11.3.2.3 Input Edge-Time Mode

NOTE: For rising-edge detection, the input signal must be High for at least two system clock periods following the rising edge. Similarly, for falling edge detection, the input signal must be Low for at least two system clock periods following the falling edge. Based on this criteria, the maximum input frequency for edge detection is 1/4 of the system frequency.

In Edge-Time mode, the timer is configured as a 16-bit or 32-bit up- or down-counter including the optional prescaler with the upper timer value stored in the **GPTIMER_TnPR** register and the lower bits in the **GPTIMER_TnILR** register. In this mode, the timer is initialized to the value loaded in the **GPTIMER_TnILR** and **GPTIMER_TnPR** registers when counting down and 0x0 when counting up. The timer is capable of capturing three types of events: rising edge, falling edge, or both. The timer is placed into Edge-Time mode by setting the TnCMR bit in the **GPTIMER_TnMR** register, and the type of event that the timer captures is determined by the **TnEVENT** fields of the **GPTIMER_CTL** register. Table 11-4 shows the values that are loaded into the timer registers when the timer is enabled.

Table 11-4. Counter Values When the Timer is Enabled in Input Event-Count Mode

Register	Count Down Mode	CountUp Mode
GPTIMER_TnR	**GPTM_TnILR**	0x0
GPTIMER_TnV	**GPTM_TnILR**	0x0
GPTIMER_TnPV	**GPTM_TnPR**	0x0

When software writes the **TnEN** bit in the **GPTIMER_CTL** register, the timer is enabled for event capture. When the selected input event is detected, the current timer counter value is captured in the **GPTIMER_TnR** register and is available to be read by the microcontroller. The GPTM then asserts the **CnERIS** bit in the **GPTM Raw Interrupt Status (GPTIMER_RIS)** register, and holds it until it is cleared by writing the **GPTM Interrupt Clear (GPTIMER_ICR)** register. If the capture mode event interrupt is enabled in the **GPTM Interrupt Mask (GPTIMER_IMR)** register, the GPTM also sets the **CnEMIS** bit in the **GPTM Masked Interrupt Status (GPTIMER_MIS)** register. In this mode, the **GPTIMER_TnR** register holds the time at which the selected input event occurred while the **GPTIMER_TnV** and **GPTIMER_TnPV** registers hold the free-running timer value and the free-running prescaler value. These registers can be read to determine the time that elapsed between the interrupt assertion and the entry into the ISR.

In addition to generating interrupts, an ADC and/or a µDMA trigger can be generated. The ADC trigger is enabled by setting the **TnOTE** bit in **GPTIMER_CTL**.The µDMA trigger is enabled by configuring and enabling the appropriate µDMA channel.

After an event has been captured, the timer does not stop counting. It continues to count until the TnEN bit is cleared. When the timer reaches the timeout value, it is reloaded with 0x0 in up-count mode and the value from the **GPTIMER_TnILR** and **GPTIMER_TnPR** registers in down-count mode.

Figure 11-3 shows how input edge timing mode works. In the diagram, it is assumed that the start value of the timer is the default value of 0xFFFF, and the timer is configured to capture rising edge events.

Each time a rising edge event is detected, the current count value is loaded into the **GPTIMER_TnR** register, and is held there until another rising edge is detected (at which point the new count value is loaded into the **GPTIMER_TnR** register).

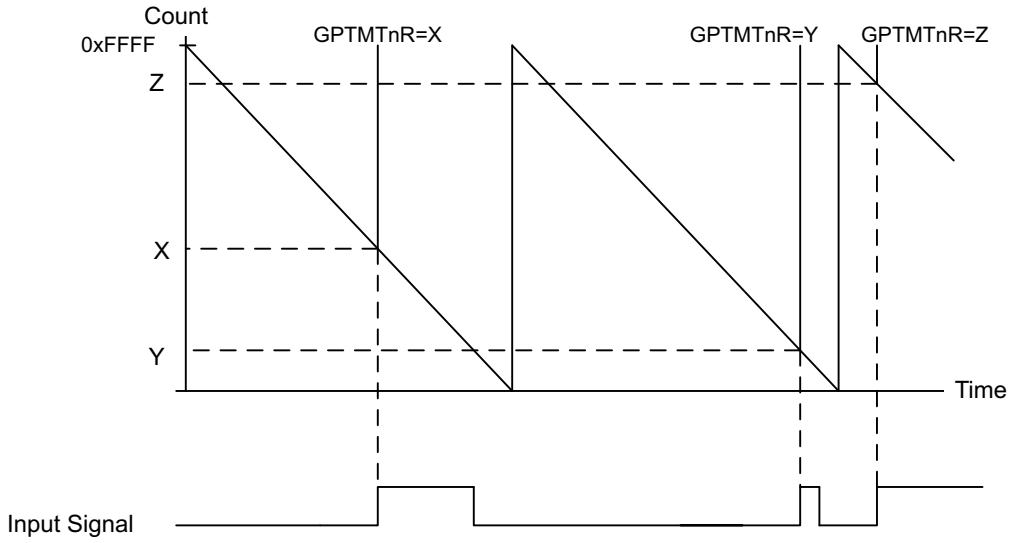


Figure 11-3. Input Edge-Time Mode Example

NOTE: When operating in Edge-time mode, the counter uses a modulo 224 count if prescalar is enabled or 216, if not. If there is a possibility the edge could take longer than the count, then another timer can be implemented to ensure detection of the missed edge.

11.3.2.4 PWM Mode

The GPTM supports a simple PWM generation mode. In PWM mode, the timer is configured as a 16-bit down-counter with a start value (and thus period) defined by the **GPTIMER_TnILR** and **GPTIMER_TnPR** registers. In this mode, the PWM frequency and period are synchronous events and therefore guaranteed to be glitch free. PWM mode is enabled with the **GPTIMER_TnMR** register by setting the **TnAMS** bit to 0x1, the **TnCMR** bit to 0x0, and the TnMR field to 0x1 or 0x2. Table 11-5 shows the values that are loaded into the timer registers when the timer is enabled.

Table 11-5. Counter Values When the Timer is Enabled in PWM Mode

Register	Count Down Mode	CountUp Mode
GPTIMER_TnR	**GPTM_TnILR**	Not Avaliable
GPTIMER_TnV	**GPTM_TnILR**	Not Avaliable
GPTIMER_TnPV	**GPTM_TnPR**	Not Avaliable

When software writes the **TnEN** bit in the **GPTIMER_CTL** register, the counter begins counting down until it reaches the 0x0 state. Alternatively, if the **TnWOT** bit is set in the **GPTIMER_TnMR** register, once the **TnEN** bit is set, the timer waits for a trigger to begin counting. On the next counter cycle in periodic mode, the counter reloads its start value from the **GPTIMER_TnILR** and **GPTIMER_TnPR** registers and continues counting until disabled by software clearing the **TnEN** bit in the **GPTIMER_CTL** register. The timer is capable of generating interrupts based on three types of events: rising edge, falling edge, or both. The event is configured by the **TnEVENT** field of the **GPTIMER_CTL** register, and the interrupt is enabled by setting the **TnPWMIE** bit in the **GPTIMER_TnMR** register. When the event occurs, the **CnERIS** bit is set in the **GPTM Raw Interrupt Status (GPTIMER_RIS)** register, and holds it until it is cleared by writing the **GPTM Interrupt Clear (GPTIMER_ICR)** register . If the capture mode event interrupt is enabled in the **GPTM Interrupt Mask (GPTIMER_IMR)** register , the GPTM also sets the **CnEMIS** bit in the **GPTM Masked Interrupt Status (GPTIMER_MIS)** register. Note that the interrupt status bits are not updated unless the **TnPWMIE** bit is set.

In this mode, the **GPTIMER_TnR** and **GPTIMER_TnV** registers always have the same value, as do the **GPTIMER_PnPS** and the **GPTIMER_TnPV** registers.

The output PWM signal asserts when the counter is at the value of the **GPTIMER_TnILR** and **GPTIMER_TnPR** registers (its start state), and is deasserted when the counter value equals the value in the **GPTIMER_TnMATCHR** and **GPTIMER_TnPMR** registers. Software has the capability of inverting the output PWM signal by setting the **TnPWML** bit in the **GPTIMER_CTL** register. Inverting the output PWM will not affect the edge detection interrupt. Thus, if a positive-edge interrupt trigger has been set, the event-trigger interrupt will be asserted when the PWM inversion generates a positive edge.

Figure 11-4 shows how to generate an output PWM with a 1-ms period and a 66% duty cycle assuming a 50-MHz input clock and TnPWML =0 (duty cycle would be 33% for the **TnPWML** =1 configuration). For this example, the start value is **GPTIMER_TnILR**=0xC350 and the match value is **GPTIMER_TnMATCHR**=0x411A.

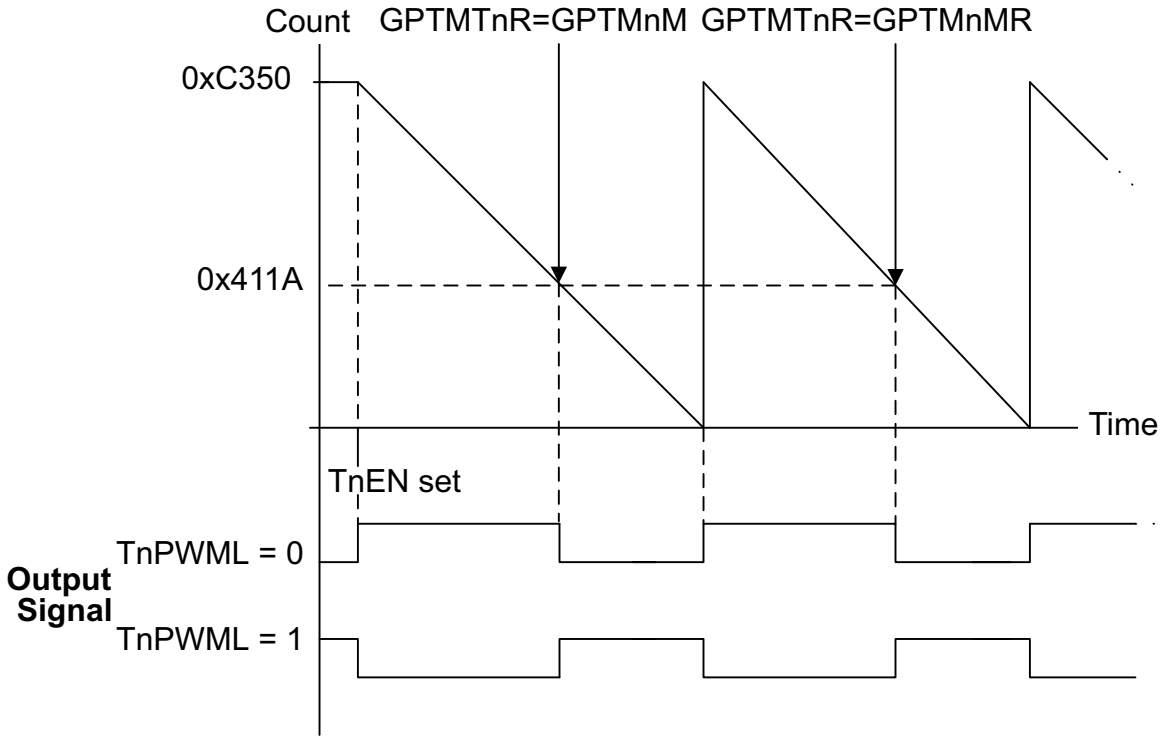


Figure 11-4. 16-bit PWM Mode Example

When synchronizing the timers using the **GPTIMER_SYNC** register, the timer must be properly configured to avoid glitches on the CCP outputs. Both the PLO and the MRSU bits must be set in the **GPTIMER_TnMR** register. Figure 11-5 shows how the CCP output operates when the **PLO** and **MRSU** bits are set and the **GPTIMER_TnMATCHR** value is greater than the **GPTIMER_TnILR** value.

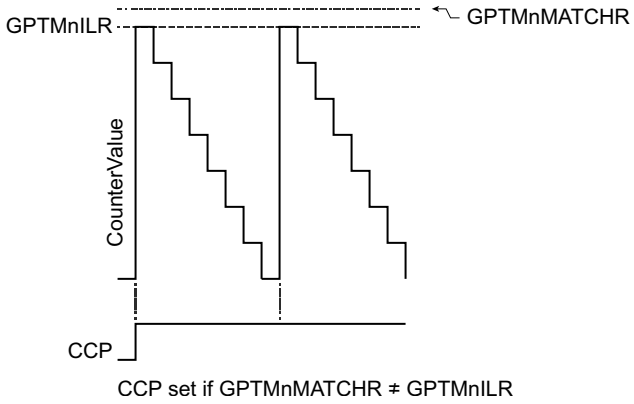


Figure 11-5. CCP Output, GPTIMER_TnMATCHR > GPTIMER_TnILR

Figure 11-6 shows how the CCP output operates when the **PLO** and **MRSU** bits are set and the **GPTIMER_TnMATCHR** value is the same as the **GPTIMER_TnILR** value. In this situation, if the **PLO** bit is 0, the CCP signal goes high when the **GPTIMER_TnILR** value is loaded and the match would be essentially ignored.

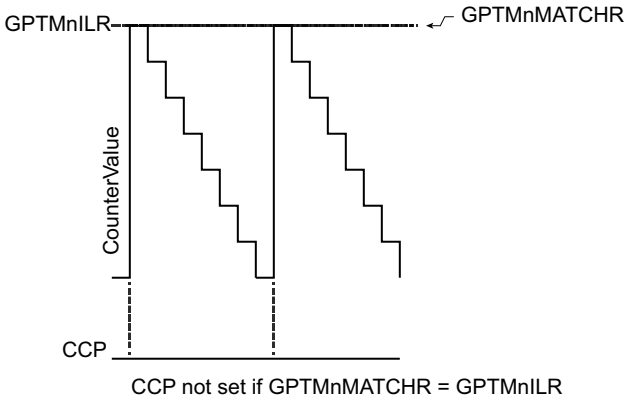


Figure 11-6. CCP Output, GPTIMER_TnMATCHR = GPTIMER_TnILR

Figure 11-7 shows how the CCP output operates when the **PLO** and **MRSU** bits are set and the **GPTIMER_TnILR** is greater than the **GPTIMER_TnMATCHR** value.

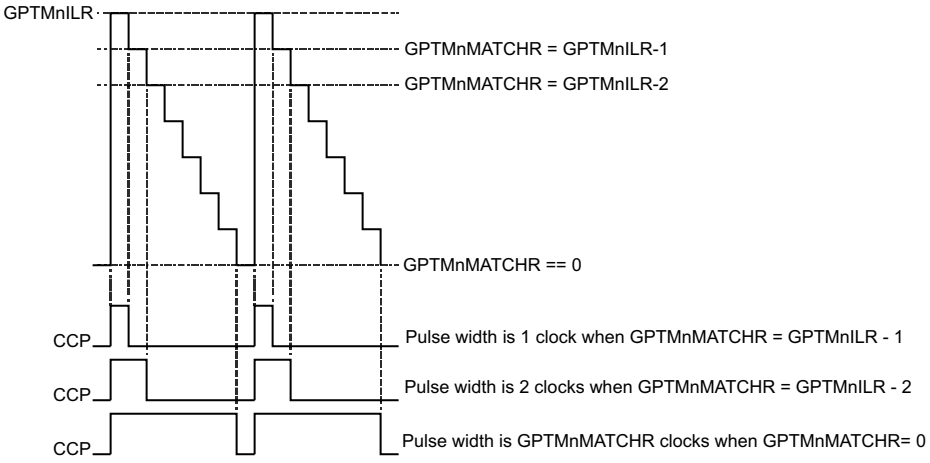


Figure 11-7. CCP Output, GPTIMER_TnILR > GPTIMER_TnMATCHR

11.3.3 Wait-for-Trigger Mode

Wait-for-trigger mode allows daisy-chaining of the timer modules such that once configured, a single timer can initiate multiple timing events using the timer triggers. Wait-for-trigger mode is enabled by setting the **TnWOT** bit in the **GPTIMER_TnMR** register. When the **TnWOT** bit is set, timer N+1 does not begin counting until the timer in the previous position in the daisy-chain (timer N) reaches its time-out event. The daisy-chain is configured such that GPTM1 always follows GPTM0, GPTM2 follows GPTM1, and so on. If timer A is configured as a 32-bit (16 or 32-bit mode) timer (controlled by the **GPTMCFG** field in the **GPTIMER_CFG** register), it triggers timer A in the next module. If timer A is configured as a 16-bit (16/32-bit mode) timer, it triggers timer B in the same module, and timer B triggers timer A in the next module. Care must be taken that the **TAWOT** bit is never set in GPTM0. Figure 11-8 shows how the **GPTMCFG** bit affects the daisy-chain. This function is valid for one-shot, periodic, and PWM modes.

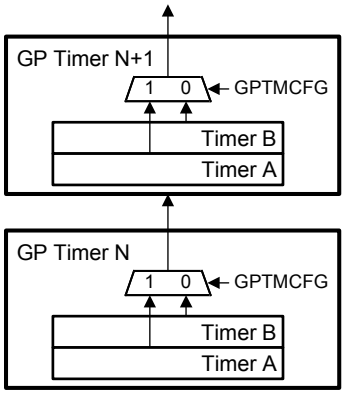


Figure 11-8. Timer Daisy-Chain

11.3.4 Synchronizing GP Timer Blocks

The **GPTM Synchronizer Control (GPTIMER_SYNC)** register in the GPTM0 block can be used to synchronize selected timers to begin counting at the same time. To be able to do so, the timers have to be started first. Setting a bit in the **GPTIMER_SYNC** register causes the associated timer to perform the actions of a time-out event. An interrupt is not generated when the timers are synchronized. If a timer is being used in concatenated mode, only the bit for timer A must be set in the **GPTIMER_SYNC** register. The register despcription shows which timers can be synchronized.

Table 11-6 shows the actions for the time-out event performed when the timers are synchronized in the various timer modes.

Table 11-6. Time-out Actions for GPTM Modes

Mode	Count Direction	Time-out Action
16-bit and 32-bit one-shot (concatenated timers)	–	N/A
16- bit and 32-bit periodic (concatenated timers)	Down	Count value = ILR
	Up	Count Value = 0
16-bit and 32- bit one-shot (individual and split timers)	–	N/A
16-bit and 32- bit periodic (individual and split timers)	Down	Count value = ILR
	Up	Count value = 0
16-bit and 32-bit edge-count (individual and split timers)	Down	Count value = ILR
	Up	Count Value = 0
16-bit and 32-bit edge-time (ndividual and split timers)	Down	Count value = ILR
	Up	Count Value = 0
16-bit PWM	Down	Count value = ILR

11.3.5 Accessing Concatenated 16- and 32-Bit GPTM Register Values

The GPTM is placed into concatenated mode by writing a 0x0 or a 0x1 to the **GPTMCFG** bit field in the **GPTM Configuration (GPTIMER_CFG)** register. In both configurations, certain 16- and 32-bit GPTM registers are concatenated to form pseudo 32-bit registers. These registers include:

- **GPTM Timer A Interval Load (GPTIMER_TAILR)** register [15:0], see GPTIMER_TAILR
- **GPTM Timer B Interval Load (GPTIMER_TBILR)** register [15:0], see GPTIMER_TBILR
- **GPTM Timer A (GPTIMER_TAR)** register [15:0], see GPTIMER_TAR
- **GPTM Timer B (GPTIMER_TBR)** register [15:0], see GPTIMER_TBR
- **GPTM Timer A Value (GPTIMER_TAV)** register [15:0], see GPTIMER_TAV
- **GPTM Timer B Value (GPTIMER_TBV)** register [15:0], see GPTIMER_TBV

- **GPTM Timer A Match (GPTIMER_TAMATCHR)** register [15:0], see GPTIMER_TAMATCHR
- **GPTM Timer B Match (GPTIMER_TBMATCHR)** register [15:0], see GPTIMER_TBMATCHR

In the 32-bit modes, the GPTM translates a 32-bit write access to **GPTIMER_TAILR** into a write access to both **GPTIMER_TAILR** and **GPTIMER_TBILR**. The resulting word ordering for such a write operation is: `GPTMTBILR[15:0]:GPTMTAILR[15:0]`. Likewise, a 32-bit read access to **GPTIMER_TAR** returns the value: `GPTMTBR[15:0]:GPTMTAR[15:0]`. A 32-bit read access to **GPTIMER_TAV** returns the value:`GPTMTBV[15:0]:GPTMTAV[15:0]`

11.4 Initialization and Configuration

To use a GPTM, the appropriate **TIMERn** bit must be set in the **SYS_CTRL_RCGC_GPT** register (see SYS_CTRL_RCGCGPT) . Configure the **Pxx_SEL** fields in the **IOC_Pxx_SEL** register to assign the timer signals to the appropriate pins (see Chapter 9).

This section shows module initialization and configuration examples for each of the supported timer modes.

11.4.1 One-Shot and Periodic Timer Modes

The GPTM is configured for one-shot and periodic modes by the following sequence:

1. Ensure the timer is disabled (the **TnEN** bit in the **GPTIMER_CTL** register is cleared) before making any changes.
2. Write the **GPTM Configuration Register (GPTIMER_CFG)** with a value of 0x0000 0000.
3. Configure the **TnMR** field in the **GPTM Timer n Mode Register (GPTIMER_TnMR)**:
   (a) Write a value of 0x1 for one-shot mode.
   (b) Write a value of 0x2 for periodic mode.
4. Optionally, configure the **TnSNAPS**, **TnWOT**, **TnMTE**, and **TnCDIR** bits in the **GPTIMER_TnMR** register to select whether to capture the value of the free-running timer at time-out, use an external trigger to start counting, configure an additional trigger or interrupt, and count up or down.
5. Load the start value into the **GPTM Timer n Interval Load Register (GPTIMER_TnILR)**.
6. If interrupts are required, set the appropriate bits in the **GPTM Interrupt Mask Register (GPTIMER_IMR)**.
7. Set the **TnEN** bit in the **GPTIMER_CTL** register to enable the timer and start counting.
8. Poll the **GPTMRIS** register or wait for the interrupt to be generated (if enabled). In both cases, the status flags are cleared by writing a 1 to the appropriate bit of the **GPTM Interrupt Clear Register (GPTIMER_ICR)**.

In one-shot mode, the timer stops counting after the time-out event. To re-enable the timer, repeat the sequence. A timer configured in periodic mode reloads the timer and continues counting after the time-out event.

11.4.2 Input Edge-Count Mode

A timer is configured to Input Edge-Count mode by the following sequence:

1. Ensure the timer is disabled (the **TAEN** bit is cleared) before making any changes.
2. Write the **GPTM Configuration (GPTIMER_CFG)** register with a value of 0x0000.0004.
3. In the **GPTM Timer Mode (GPTIMER_TnMR)** register, write the **TnCMR** field to 0x0 and the **TnMR** field to 0x3.
4. Configure the type of event(s) that the timer captures by writing the **TnEVENT** field of the **GPTM Control (GPTIMER_CTL)** register.
5. If a prescaler is to be used, write the prescale value to the **GPTM Timer n Prescale Register (GPTIMER_TnPR)**.
6. Load the timer start value into the **GPTM Timer n Interval Load (GPTIMER_TnILR)** register.
7. Load the event count into the **GPTM Timer n Match (GPTIMER_TnMATCHR)** register.

8. If interrupts are required, set the **CnMIM** bit in the **GPTM Interrupt Mask (GPTIMER_IMR)** register.
9. Set the **TnEN** bit in the **GPTIMER_CTL** register to enable the timer and begin waiting for edge events.
10. Poll the **CnMRIS** bit in the **GPTMRIS** register or wait for the interrupt to be generated (if enabled). In both cases, the status flags are cleared by writing a 1 to the **CnMCINT** bit of the **GPTM Interrupt Clear (GPTIMER_ICR)** register.

When counting down in Input Edge-Count Mode, the timer stops after the programmed number of edge events has been detected. To re-enable the timer, ensure that the **TnEN** bit is cleared and repeat #4 through #9.

11.4.3 Input Edge-Timing Mode

A timer is configured to Input Edge-Timing mode by the following sequence:

1. Ensure the timer is disabled (the **TAEN** bit is cleared) before making any changes.
2. Write the **GPTM Configuration (GPTIMER_CFG)** register with a value of 0x0000.0004.
3. In the **GPTM Timer Mode (GPTIMER_TnMR)** register, write the **TnCMR** field to 0x1 and the **TnMR** field to 0x3.
4. Configure the type of event(s) that the timer captures by writing the **TnEVENT** field of the **GPTM Control (GPTIMER_CTL)** register.
5. If a prescaler is to be used, write the prescale value to the **GPTM Timer n Prescale Register (GPTIMER_TnPR)**.
6. Load the timer start value into the **GPTM Timer n Interval Load (GPTIMER_TnILR)** register.
7. If interrupts are required, set the **CnMIM** bit in the **GPTM Interrupt Mask (GPTIMER_IMR)** register.
8. Set the **TnEN** bit in the **GPTIMER_CTL** register to enable the timer and start counting.
9. Poll the **CnMRIS** bit in the **GPTMRIS** register or wait for the interrupt to be generated (if enabled). In both cases, the status flags are cleared by writing a 1 to the **CnMCINT** bit of the **GPTM Interrupt Clear (GPTIMER_ICR)** register.

In Input Edge Timing mode, the timer continues running after an edge event has been detected, but the timer interval can be changed at any time by writing the GPTMTnILR register. The change takes effect at the next cycle after the write.

11.4.4 PWM Mode

A timer is configured to PWM mode using the following sequence:

1. Ensure the timer is disabled (the **TnEN** bit is cleared) before making any changes.
2. Write the **GPTM Configuration (GPTIMER_CFG)** register with a value of 0x0000.0004.
3. In the **GPTM Timer Mode (GPTIMER_TnMR)** register, write the **TnCMR** field to 0x1 and the **TnMR** field to 0x2.
4. Configure the output state of the PWM signal (whether or not it is inverted) in the **TnPWML** field of the **GPTM Control (GPTIMER_CTL)** register.
5. If a prescaler is to be used, write the prescale value to the **GPTM Timer n Prescale Register (GPTIMER_TnPR)**.
6. If PWM interrupts are used, configure the interrupt condition in the **TnEVENT** field in the **GPTIMER_CTL** register and enable the interrupts by setting the **TnPWMIE** bit in the **GPTIMER_TnMR** register.
7. Load the timer start value into the **GPTM Timer n Interval Load (GPTIMER_TnILR)** register.
8. Load the **GPTM Timer n Match (GPTIMER_TnMATCHR)** register with the match value.
9. Set the **TnEN** bit in the **GPTM Control (GPTIMER_CTL)** register to enable the timer and begin generation of the output PWM signal.

In PWM Timing mode, the timer continues running after the PWM signal has been generated. The PWM period can be adjusted at any time by writing the **GPTMTnILR** register, and the change takes effect at the next cycle after the write.

11.5 General-Purpose Timer Registers

11.5.1 GPTIMER Registers

11.5.1.1 GPTIMER Registers Mapping Summary

This section provides information on the GPTIMER module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

11.5.1.1.1 GPTIMER Common Registers Mapping

This section provides information on the GPTIMER module instance within this product. Each of the registers within the Module Instance is described separately below.

Table 11-7. GPTIMER Common Registers Mapping Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset
GPTIMER_CFG	RW	32	0x0000 0000	0x000
GPTIMER_TAMR	RW	32	0x0000 0000	0x004
GPTIMER_TBMR	RW	32	0x0000 0000	0x008
GPTIMER_CTL	RW	32	0x0000 0000	0x00C
GPTIMER_SYNC	RW	32	0x0000 0000	0x010
GPTIMER_IMR	RW	32	0x0000 0000	0x018
GPTIMER_RIS	RO	32	0x0000 0000	0x01C
GPTIMER_MIS	RO	32	0x0000 0000	0x020
GPTIMER_ICR	RW	32	0x0000 0000	0x024
GPTIMER_TAILR	RW	32	0xFFFF FFFF	0x028
GPTIMER_TBILR	RW	32	0x0000 FFFF	0x02C
GPTIMER_TAMATCHR	RW	32	0xFFFF FFFF	0x030
GPTIMER_TBMATCHR	RW	32	0x0000 FFFF	0x034
GPTIMER_TAPR	RW	32	0x0000 0000	0x038
GPTIMER_TBPR	RW	32	0x0000 0000	0x03C
GPTIMER_TAPMR	RW	32	0x0000 0000	0x040
GPTIMER_TBPMR	RW	32	0x0000 0000	0x044
GPTIMER_TAR	RO	32	0xFFFF FFFF	0x048
GPTIMER_TBR	RO	32	0x0000 FFFF	0x04C
GPTIMER_TAV	RW	32	0xFFFF FFFF	0x050
GPTIMER_TBV	RW	32	0x0000 FFFF	0x054
GPTIMER_TAPS	RO	32	0x0000 0000	0x05C
GPTIMER_TBPS	RO	32	0x0000 0000	0x060
GPTIMER_TAPV	RO	32	0x0000 0000	0x064
GPTIMER_TBPV	RO	32	0x0000 0000	0x068
GPTIMER_PP	RO	32	0x0000 0000	0xFC0

11.5.1.1.2 GPTIMER Instances Register Mapping Summary

11.5.1.1.2.1 GPTIMER0 Register Summary

Table 11-8. GPTIMER0 Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
GPTIMER_CFG	RW	32	0x0000 0000	0x000	0x4003 0000
GPTIMER_TAMR	RW	32	0x0000 0000	0x004	0x4003 0004
GPTIMER_TBMR	RW	32	0x0000 0000	0x008	0x4003 0008
GPTIMER_CTL	RW	32	0x0000 0000	0x00C	0x4003 000C
GPTIMER_SYNC	RW	32	0x0000 0000	0x010	0x4003 0010
GPTIMER_IMR	RW	32	0x0000 0000	0x018	0x4003 0018
GPTIMER_RIS	RO	32	0x0000 0000	0x01C	0x4003 001C
GPTIMER_MIS	RO	32	0x0000 0000	0x020	0x4003 0020
GPTIMER_ICR	RW	32	0x0000 0000	0x024	0x4003 0024
GPTIMER_TAILR	RW	32	0xFFFF FFFF	0x028	0x4003 0028
GPTIMER_TBILR	RW	32	0x0000 FFFF	0x02C	0x4003 002C
GPTIMER_TAMATCHR	RW	32	0xFFFF FFFF	0x030	0x4003 0030
GPTIMER_TBMATCHR	RW	32	0x0000 FFFF	0x034	0x4003 0034
GPTIMER_TAPR	RW	32	0x0000 0000	0x038	0x4003 0038
GPTIMER_TBPR	RW	32	0x0000 0000	0x03C	0x4003 003C
GPTIMER_TAPMR	RW	32	0x0000 0000	0x040	0x4003 0040
GPTIMER_TBPMR	RW	32	0x0000 0000	0x044	0x4003 0044
GPTIMER_TAR	RO	32	0xFFFF FFFF	0x048	0x4003 0048
GPTIMER_TBR	RO	32	0x0000 FFFF	0x04C	0x4003 004C
GPTIMER_TAV	RW	32	0xFFFF FFFF	0x050	0x4003 0050
GPTIMER_TBV	RW	32	0x0000 FFFF	0x054	0x4003 0054
GPTIMER_TAPS	RO	32	0x0000 0000	0x05C	0x4003 005C
GPTIMER_TBPS	RO	32	0x0000 0000	0x060	0x4003 0060
GPTIMER_TAPV	RO	32	0x0000 0000	0x064	0x4003 0064
GPTIMER_TBPV	RO	32	0x0000 0000	0x068	0x4003 0068
GPTIMER_PP	RO	32	0x0000 0000	0xFC0	0x4003 0FC0

11.5.1.1.2.2 GPTIMER1 Register Summary

Table 11-9. GPTIMER1 Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
GPTIMER_CFG	RW	32	0x0000 0000	0x000	0x4003 1000
GPTIMER_TAMR	RW	32	0x0000 0000	0x004	0x4003 1004
GPTIMER_TBMR	RW	32	0x0000 0000	0x008	0x4003 1008
GPTIMER_CTL	RW	32	0x0000 0000	0x00C	0x4003 100C
GPTIMER_SYNC	RW	32	0x0000 0000	0x010	0x4003 1010
GPTIMER_IMR	RW	32	0x0000 0000	0x018	0x4003 1018
GPTIMER_RIS	RO	32	0x0000 0000	0x01C	0x4003 101C

Table 11-9. GPTIMER1 Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
GPTIMER_MIS	RO	32	0x0000 0000	0x020	0x4003 1020
GPTIMER_ICR	RW	32	0x0000 0000	0x024	0x4003 1024
GPTIMER_TAILR	RW	32	0xFFFF FFFF	0x028	0x4003 1028
GPTIMER_TBILR	RW	32	0x0000 FFFF	0x02C	0x4003 102C
GPTIMER_TAMATCHR	RW	32	0xFFFF FFFF	0x030	0x4003 1030
GPTIMER_TBMATCHR	RW	32	0x0000 FFFF	0x034	0x4003 1034
GPTIMER_TAPR	RW	32	0x0000 0000	0x038	0x4003 1038
GPTIMER_TBPR	RW	32	0x0000 0000	0x03C	0x4003 103C
GPTIMER_TAPMR	RW	32	0x0000 0000	0x040	0x4003 1040
GPTIMER_TBPMR	RW	32	0x0000 0000	0x044	0x4003 1044
GPTIMER_TAR	RO	32	0xFFFF FFFF	0x048	0x4003 1048
GPTIMER_TBR	RO	32	0x0000 FFFF	0x04C	0x4003 104C
GPTIMER_TAV	RW	32	0xFFFF FFFF	0x050	0x4003 1050
GPTIMER_TBV	RW	32	0x0000 FFFF	0x054	0x4003 1054
GPTIMER_TAPS	RO	32	0x0000 0000	0x05C	0x4003 105C
GPTIMER_TBPS	RO	32	0x0000 0000	0x060	0x4003 1060
GPTIMER_TAPV	RO	32	0x0000 0000	0x064	0x4003 1064
GPTIMER_TBPV	RO	32	0x0000 0000	0x068	0x4003 1068
GPTIMER_PP	RO	32	0x0000 0000	0xFC0	0x4003 1FC0

11.5.1.1.2.3 GPTIMER2 Register Summary

Table 11-10. GPTIMER2 Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
GPTIMER_CFG	RW	32	0x0000 0000	0x000	0x4003 2000
GPTIMER_TAMR	RW	32	0x0000 0000	0x004	0x4003 2004
GPTIMER_TBMR	RW	32	0x0000 0000	0x008	0x4003 2008
GPTIMER_CTL	RW	32	0x0000 0000	0x00C	0x4003 200C
GPTIMER_SYNC	RW	32	0x0000 0000	0x010	0x4003 2010
GPTIMER_IMR	RW	32	0x0000 0000	0x018	0x4003 2018
GPTIMER_RIS	RO	32	0x0000 0000	0x01C	0x4003 201C
GPTIMER_MIS	RO	32	0x0000 0000	0x020	0x4003 2020
GPTIMER_ICR	RW	32	0x0000 0000	0x024	0x4003 2024
GPTIMER_TAILR	RW	32	0xFFFF FFFF	0x028	0x4003 2028
GPTIMER_TBILR	RW	32	0x0000 FFFF	0x02C	0x4003 202C
GPTIMER_TAMATCHR	RW	32	0xFFFF FFFF	0x030	0x4003 2030
GPTIMER_TBMATCHR	RW	32	0x0000 FFFF	0x034	0x4003 2034

Table 11-10. GPTIMER2 Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
GPTIMER_TAPR	RW	32	0x0000 0000	0x038	0x4003 2038
GPTIMER_TBPR	RW	32	0x0000 0000	0x03C	0x4003 203C
GPTIMER_TAPMR	RW	32	0x0000 0000	0x040	0x4003 2040
GPTIMER_TBPMR	RW	32	0x0000 0000	0x044	0x4003 2044
GPTIMER_TAR	RO	32	0xFFFF FFFF	0x048	0x4003 2048
GPTIMER_TBR	RO	32	0x0000 FFFF	0x04C	0x4003 204C
GPTIMER_TAV	RW	32	0xFFFF FFFF	0x050	0x4003 2050
GPTIMER_TBV	RW	32	0x0000 FFFF	0x054	0x4003 2054
GPTIMER_TAPS	RO	32	0x0000 0000	0x05C	0x4003 205C
GPTIMER_TBPS	RO	32	0x0000 0000	0x060	0x4003 2060
GPTIMER_TAPV	RO	32	0x0000 0000	0x064	0x4003 2064
GPTIMER_TBPV	RO	32	0x0000 0000	0x068	0x4003 2068
GPTIMER_PP	RO	32	0x0000 0000	0xFC0	0x4003 2FC0

11.5.1.1.2.4 GPTIMER3 Register Summary

Table 11-11. GPTIMER3 Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
GPTIMER_CFG	RW	32	0x0000 0000	0x000	0x4003 3000
GPTIMER_TAMR	RW	32	0x0000 0000	0x004	0x4003 3004
GPTIMER_TBMR	RW	32	0x0000 0000	0x008	0x4003 3008
GPTIMER_CTL	RW	32	0x0000 0000	0x00C	0x4003 300C
GPTIMER_SYNC	RW	32	0x0000 0000	0x010	0x4003 3010
GPTIMER_IMR	RW	32	0x0000 0000	0x018	0x4003 3018
GPTIMER_RIS	RO	32	0x0000 0000	0x01C	0x4003 301C
GPTIMER_MIS	RO	32	0x0000 0000	0x020	0x4003 3020
GPTIMER_ICR	RW	32	0x0000 0000	0x024	0x4003 3024
GPTIMER_TAILR	RW	32	0xFFFF FFFF	0x028	0x4003 3028
GPTIMER_TBILR	RW	32	0x0000 FFFF	0x02C	0x4003 302C
GPTIMER_TAMATCHR	RW	32	0xFFFF FFFF	0x030	0x4003 3030
GPTIMER_TBMATCHR	RW	32	0x0000 FFFF	0x034	0x4003 3034
GPTIMER_TAPR	RW	32	0x0000 0000	0x038	0x4003 3038
GPTIMER_TBPR	RW	32	0x0000 0000	0x03C	0x4003 303C
GPTIMER_TAPMR	RW	32	0x0000 0000	0x040	0x4003 3040
GPTIMER_TBPMR	RW	32	0x0000 0000	0x044	0x4003 3044
GPTIMER_TAR	RO	32	0xFFFF FFFF	0x048	0x4003 3048
GPTIMER_TBR	RO	32	0x0000 FFFF	0x04C	0x4003 304C
GPTIMER_TAV	RW	32	0xFFFF FFFF	0x050	0x4003 3050

Table 11-11. GPTIMER3 Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
GPTIMER_TBV	RW	32	0x0000 FFFF	0x054	0x4003 3054
GPTIMER_TAPS	RO	32	0x0000 0000	0x05C	0x4003 305C
GPTIMER_TBPS	RO	32	0x0000 0000	0x060	0x4003 3060
GPTIMER_TAPV	RO	32	0x0000 0000	0x064	0x4003 3064
GPTIMER_TBPV	RO	32	0x0000 0000	0x068	0x4003 3068
GPTIMER_PP	RO	32	0x0000 0000	0xFC0	0x4003 3FC0

11.5.1.2 GPTIMER Common Register Descriptions

GPTIMER_CFG

Address offset	0x000		
Physical Address	0x4003 0000 0x4003 2000 0x4003 1000 0x4003 3000	**Instance**	GPTIMER0 GPTIMER2 GPTIMER1 GPTIMER3
Description	GPTM configuration This register configures the global operation of the GPTM. The value written to this register determines whether the GPTM is in 32-bit mode (concatenated timers) or in 16-bit mode (individual, split timers).		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:3	RESERVED	This bit field is reserved.	RO	0x0000 0000
2:0	GPTMCFG	GPTM configuration The GPTMCFG values are defined as follows: 0x0: 32-bit timer configuration. 0x1: 32-bit real-time clock 0x2: Reserved 0x3: Reserved 0x4: 16-bit timer configuration. The function is controlled by bits [1:0] of GPTMTAMR and GPTMTBMR. 0x5-0x7: Reserved	RW	0x0

GPTIMER_TAMR

Address offset	0x004		
Physical Address	0x4003 0004 0x4003 2004 0x4003 1004 0x4003 3004	**Instance**	GPTIMER0 GPTIMER2 GPTIMER1 GPTIMER3
Description	GPTM Timer A mode This register configures the GPTM based on the configuration selected in the CFG register. This register controls the modes for Timer A when it is used individually. When Timer A and Timer B are concatenated, this register controls the modes for both Timer A and Timer B, and the contents of TBMR are ignored.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																				TAPLO	TAMRSU	TAPWMIE	TAILD	TASNAPS	TAWOT	TAMIE	TACDIR	TAAMS	TACMR	TAMR	

Bits	Field Name	Description	Type	Reset
31:12	RESERVED	This bit field is reserved.	RO	0x0 0000
11	TAPLO	Legacy PWM operation 0: Legacy operation 1: CCP is set to 1 on time-out.	RW	0
10	TAMRSU	Timer A match register update mode 0: Update GPTMAMATCHR and GPTMAPR if used on the next cycle. 1: Update GPTMAMATCHR and GPTMAPR if used on the next time-out. If the timer is disabled (TAEN is clear) when this bit is set, GPTMTAMATCHR and GPTMTAPR are updated when the timer is enabled. If the timer is stalled (TASTALL is set), GPTMTAMATCHR and GPTMTAPR are updated according to the configuration of this bit.	RW	0
9	TAPWMIE	GPTM Timer A PWM interrupt enable This bit enables interrupts in PWM mode on rising, falling, or both edges of the CCP output. 0: Interrupt is disabled. 1: Interrupt is enabled. This bit is valid only in PWM mode.	RW	0
8	TAILD	GPTM Timer A PWM interval load write 0: Update the GPTMTAR register with the value in the GPTMTAILR register on the next cycle. If the prescaler is used, update the GPTMTAPS register with the value in the GPTMTAPR register on the next cycle. 1: Update the GPTMTAR register with the value in the GPTMTAILR register on the next cycle. If the prescaler is used, update the GPTMTAPS register with the value in the GPTMTAPR register on the next time-out.	RW	0
7	TASNAPS	GPTM Timer A snap-shot mode 0: Snap-shot mode is disabled. 1: If Timer A is configured in periodic mode, the actual free-running value of Timer A is loaded at the time-out event into the GPTM Timer A (GPTMTAR) register.	RW	0
6	TAWOT	GPTM Timer A wait-on-trigger 0: Timer A begins counting as soon as it is enabled. 1: If Timer A is enabled (TAEN is set in the GPTMCTL register), Timer A does not begin counting until it receives a trigger from the Timer in the previous position in the daisy-chain. This bit must be clear for GP Timer module 0, Timer A.	RW	0
5	TAMIE	GPTM Timer A match interrupt enable 0: The match interrupt is disabled. 1: An interrupt is generated when the match value in the GPTMTAMATCHR register is reached in the one-shot and periodic modes.	RW	0
4	TACDIR	GPTM Timer A count direction 0: The timer counts down. 1: The timer counts up. When counting up, the timer starts from a value of 0x0.	RW	0
3	TAAMS	GPTM Timer A alternate mode 0: Capture mode is enabled. 1: PWM mode is enabled. Note: To enable PWM mode, the TACM bit must be cleared and the TAMR field must be configured to 0x2.	RW	0
2	TACMR	GPTM Timer A capture mode 0: Edge-count mode 1: Edge-time mode	RW	0

Bits	Field Name	Description	Type	Reset
1:0	TAMR	GPTM Timer A mode 0x0: Reserved 0x1: One-shot mode 0x2: Periodic mode 0x3: Capture mode The timer mode is based on the timer configuration defined by bits [2:0] in the GPTMCFG register.	RW	0x0

GPTIMER_TBMR

Address offset 0x008

Physical Address 0x4003 0008 (GPTIMER0), 0x4003 2008 (GPTIMER2), 0x4003 1008 (GPTIMER1), 0x4003 3008 (GPTIMER3)

Instance

Description GPTM Timer B mode
This register configures the GPTM based on the configuration selected in the CFG register.
This register controls the modes for Timer B when it is used individually. When Timer A and Timer B are concatenated, this register is ignored and TBMR controls the modes for both Timer A and Timer B.

Type RW

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																				TBPLO	TBMRSU	TBPWMIE	TBILD	TBSNAPS	TBWOT	TBMIE	TBCDIR	TBAMS	TBCMR	TBMR	

Bits	Field Name	Description	Type	Reset
31:12	RESERVED	This bit field is reserved.	RO	0x0 0000
11	TBPLO	Legacy PWM operation 0: Legacy operation 1: CCP is set to 1 on time-out.	RW	0
10	TBMRSU	Timer B match register update mode 0: Update the GPTMBMATCHR and the GPTMBPR, if used on the next cycle. 1: Update the GPTMBMATCHR and the GPTMBPR, if used on the next time-out. If the timer is disabled (TAEN is clear) when this bit is set, GPTMTBMATCHR and GPTMTBPR are updated when the timer is enabled. If the timer is stalled (TBSTALL is set), GPTMTBMATCHR and GPTMTBPR are updated according to the configuration of this bit.	RW	0
9	TBPWMIE	GPTM Timer B PWM interrupt enable This bit enables interrupts in PWM mode on rising, falling, or both edges of the CCP output. 0: Interrupt is disabled. 1: Interrupt is enabled. This bit is valid only in PWM mode.	RW	0
8	TBILD	GPTM Timer B PWM interval load write 0: Update the GPTMTBR register with the value in the GPTMTBILR register on the next cycle. If the prescaler is used, update the GPTMTBPS register with the value in the GPTMTBPR register on the next cycle. 1: Update the GPTMTBR register with the value in the GPTMTBILR register on the next cycle. If the prescaler is used, update the GPTMTBPS register with the value in the GPTMTBPR register on the next time-out.	RW	0
7	TBSNAPS	GPTM Timer B snap-shot mode 0: Snap-shot mode is disabled. 1: If Timer B is configured in the periodic mode, the actual free-running value of Timer A is loaded into the GPTM Timer B (GPTMTBR) register at the time-out event.	RW	0

Bits	Field Name	Description	Type	Reset
6	TBWOT	GPTM Timer B wait-on-trigger 0: Timer B begins counting as soon as it is enabled. 1: If Timer B is enabled (TBEN is set in the GPTMCTL register), Timer B does not begin counting until it receives a trigger from the timer in the previous position in the daisy-chain.	RW	0
5	TBMIE	GPTM Timer B match interrupt enable 0: The match interrupt is disabled. 1: An interrupt is generated when the match value in the GPTMTBMATCHR register is reached in the one-shot and periodic modes.	RW	0
4	TBCDIR	GPTM Timer B count direction 0: The timer counts down. 1: The timer counts up. When counting up, the timer starts from a value of 0x0.	RW	0
3	TBAMS	GPTM Timer B alternate mode 0: Capture mode is enabled. 1: PWM mode is enabled. Note: To enable PWM mode, the TBCM bit must be cleared and the TBMR field must be configured to 0x2.	RW	0
2	TBCMR	GPTM Timer B capture mode 0: Edge-count mode 1: Edge-time mode	RW	0
1:0	TBMR	GPTM Timer B mode 0x0: Reserved 0x1: One-shot timer mode 0x2: Periodic timer mode 0x3: Capture mode The timer mode is based on the timer configuration defined by bits [2:0] in the GPTMCFG register.	RW	0x0

GPTIMER_CTL

Address offset 0x00C

Physical Address	**Instance**
0x4003 000C	GPTIMER0
0x4003 200C	GPTIMER2
0x4003 100C	GPTIMER1
0x4003 300C	GPTIMER3

Description GPTM control
This register is used alongside the CFG and TnMR registers to fine-tune the timer configuration, and to enable other features such as timer stall.

Type RW

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																TBPWML	TBOTE	RESERVED	TBEVENT			TBSTALL	TBEN	RESERVED	TAPWML	TAOTE	RESERVED	TAEVENT		TASTALL	TAEN

Bits	Field Name	Description	Type	Reset
31:15	RESERVED	This bit field is reserved.	RO	0x0 0000
14	TBPWML	GPTM Timer B PWM output level 0: Output is unaffected. 1: Output is inverted.	RW	0
13	TBOTE	GPTM Timer B output trigger enable 0: The ADC trigger of output Timer B is disabled. 1: The ADC trigger of output Timer B is enabled.	RW	0
12	RESERVED	This bit field is reserved.	RO	0

Bits	Field Name	Description	Type	Reset
11:10	TBEVENT	GPTM Timer B event mode 0x0: Positive edge 0x1: Negative edge 0x2: Reserved 0x3: Both edges	RW	0x0
9	TBSTALL	GPTM Timer B stall enable 0: Timer B continues counting while the processor is halted by the debugger. 1: Timer B freezes counting while the processor is halted by the debugger.	RW	0
8	TBEN	GPTM Timer B enable 0: Timer B is disabled. 1: Timer B is enabled and begins counting or the capture logic is enabled based on the GPTMCFG register.	RW	0
7	RESERVED	This bit field is reserved.	RO	0
6	TAPWML	GPTM Timer A PWM output level 0: Output is unaffected. 1: Output is inverted.	RW	0
5	TAOTE	GPTM Timer A output trigger enable 0: The ADC trigger of output Timer A is disabled. 1: The ADC trigger of output Timer A is enabled.	RW	0
4	RESERVED	This bit field is reserved.	RW	0
3:2	TAEVENT	GPTM Timer A event mode 0x0: Positive edge 0x1: Negative edge 0x2: Reserved 0x3: Both edges	RW	0x0
1	TASTALL	GPTM Timer A stall enable 0: Timer A continues counting while the processor is halted by the debugger. 1: Timer A freezes counting while the processor is halted by the debugger.	RW	0
0	TAEN	GPTM Timer A enable 0: Timer A is disabled. 1: Timer A is enabled and begins counting or the capture logic is enabled based on the GPTMCFG register.	RW	0

GPTIMER_SYNC

Address offset	0x010		
Physical Address	0x4003 0010 0x4003 2010 0x4003 1010 0x4003 3010	**Instance**	GPTIMER0 GPTIMER2 GPTIMER1 GPTIMER3
Description	GPTM synchronize Note: This register is implemented on GPTM 0 base address only. This register does however, allow software to synchronize a number of timers.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								SYNC3		SYNC2		SYNC1		SYNC0	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:6	SYNC3	Synchronize GPTM3 0x0: GPTM3 is not affected. 0x1: A time-out event for Timer A of GPTM3 is triggered. 0x2: A time-out event for Timer B of GPTM3 is triggered. 0x3: A time-out event for Timer A and Timer B of GPTM3 is triggered.	RW	0x0

Bits	Field Name	Description	Type	Reset
5:4	SYNC2	Synchronize GPTM2 0x0: GPTM2 is not affected. 0x1: A time-out event for Timer A of GPTM2 is triggered. 0x2: A time-out event for Timer B of GPTM2 is triggered. 0x3: A time-out event for Timer A and Timer B of GPTM2 is triggered.	RW	0x0
3:2	SYNC1	Synchronize GPTM1 0x0: GPTM1 is not affected. 0x1: A time-out event for Timer A of GPTM1 is triggered. 0x2: A time-out event for Timer B of GPTM1 is triggered. 0x3: A time-out event for Timer A and Timer B of GPTM1 is triggered.	RW	0x0
1:0	SYNC0	Synchronize GPTM0 0x0: GPTM0 is not affected. 0x1: A time-out event for Timer A of GPTM0 is triggered. 0x2: A time-out event for Timer B of GPTM0 is triggered. 0x3: A time-out event for Timer A and Timer B of GPTM0 is triggered.	RW	0x0

GPTIMER_IMR

Address offset 0x018

Physical Address 0x4003 0018, 0x4003 2018, 0x4003 1018, 0x4003 3018

Instance GPTIMER0, GPTIMER2, GPTIMER1, GPTIMER3

Description GPTM interrupt mask
This register allows software to enable and disable GPTM controller-level interrupts. Setting a bit enables the corresponding interrupt, while clearing a bit disables it.

Type RW

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																				TBMIM	CBEIM	CBMIM	TBTOIM	RESERVED			TAMIM	RESERVED	CAEIM	CAMIM	TATOIM

Bits	Field Name	Description	Type	Reset
31:12	RESERVED	This bit field is reserved.	RO	0x0 0000
11	TBMIM	GPTM Timer B match interrupt mask 0: Interrupt is disabled. 1: Interrupt is enabled.	RW	0
10	CBEIM	GPTM Timer B capture event interrupt mask 0: Interrupt is disabled. 1: Interrupt is enabled.	RW	0
9	CBMIM	GPTM Timer B capture match interrupt mask 0: Interrupt is disabled. 1: Interrupt is enabled.	RW	0
8	TBTOIM	GPTM Timer B time-out interrupt mask 0: Interrupt is disabled. 1: Interrupt is enabled.	RW	0
7:5	RESERVED	This bit field is reserved.	RO	0x0
4	TAMIM	GPTM Timer A match interrupt mask 0: Interrupt is disabled. 1: Interrupt is enabled.	RW	0
3	RESERVED	This bit field is reserved.	RW	0
2	CAEIM	GPTM Timer A capture event interrupt mask 0: Interrupt is disabled. 1: Interrupt is enabled.	RW	0

Bits	Field Name	Description	Type	Reset
1	CAMIM	GPTM Timer A capture match interrupt mask 0: Interrupt is disabled. 1: Interrupt is enabled.	RW	0
0	TATOIM	GPTM Timer A time-out interrupt mask 0: Interrupt is disabled. 1: Interrupt is enabled.	RW	0

GPTIMER_RIS

Address offset 0x01C

Physical Address 0x4003 001C, 0x4003 201C, 0x4003 101C, 0x4003 301C

Instance GPTIMER0, GPTIMER2, GPTIMER1, GPTIMER3

Description GPTM raw interrupt status
This register shows the state of the GPTM internal interrupt signal. These bits are set whether or not the interrupt is masked in the IMR register. Each bit can be cleared by writing 1 to its corresponding bit in ICR.

Type RO

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																				TBMRIS	CBERIS	CBMRIS	TBTORIS	RESERVED			TAMRIS	RESERVED	CAERIS	CAMRIS	TATORIS

Bits	Field Name	Description	Type	Reset
31:12	RESERVED	This bit field is reserved.	RO	0x0 0000
11	TBMRIS	GPTM Timer B match raw interrupt	RO	0
10	CBERIS	GPTM Timer B capture event raw interrupt	RO	0
9	CBMRIS	GPTM Timer B capture match raw interrupt	RO	0
8	TBTORIS	GPTM Timer B time-out raw interrupt	RO	0
7:5	RESERVED	This bit field is reserved.	RO	0x0
4	TAMRIS	GPTM Timer A match raw interrupt	RO	0
3	RESERVED	This bit field is reserved.	RO	0
2	CAERIS	GPTM Timer A capture event raw interrupt	RO	0
1	CAMRIS	GPTM Timer A capture match raw interrupt	RO	0
0	TATORIS	GPTM Timer A time-out raw interrupt	RO	0

GPTIMER_MIS

Address offset 0x020

Physical Address 0x4003 0020, 0x4003 2020, 0x4003 1020, 0x4003 3020

Instance GPTIMER0, GPTIMER2, GPTIMER1, GPTIMER3

Description GPTM masked interrupt status
This register shows the state of the GPTM controller-level interrupt. If an interrupt is unmasked in IMR, and there is an event that causes the interrupt to be asserted, the corresponding bit is set in this register. All bits are cleared by writing 1 to the corresponding bit in ICR.

Type RO

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																				TBMMIS	CBEMIS	CBMMIS	TBTOMIS	RESERVED			TAMRIS	RESERVED	CAEMIS	CAMMIS	TATOMIS

Bits	Field Name	Description	Type	Reset
31:12	RESERVED	This bit field is reserved.	RO	0x0 0000
11	TBMMIS	GPTM Timer B match masked interrupt	RO	0
10	CBEMIS	GPTM Timer B capture event masked interrupt	RO	0
9	CBMMIS	GPTM Timer B capture match masked interrupt	RO	0
8	TBTOMIS	GPTM Timer B time-out masked interrupt	RO	0
7:5	RESERVED	This bit field is reserved.	RO	0x0
4	TAMRIS	GPTM Timer A match raw interrupt	RO	0
3	RESERVED	This bit field is reserved.	RO	0
2	CAEMIS	GPTM Timer A capture event raw interrupt	RO	0
1	CAMMIS	GPTM Timer A capture match raw interrupt	RO	0
0	TATOMIS	GPTM Timer A time-out raw interrupt	RO	0

GPTIMER_ICR

Address offset	0x024		
Physical Address	0x4003 0024 0x4003 2024 0x4003 1024 0x4003 3024	**Instance**	GPTIMER0 GPTIMER2 GPTIMER1 GPTIMER3
Description	GPTM interrupt clear This register is used to clear the status bits in the RIS and MIS registers. Writing 1 to a bit clears the corresponding bit in the RIS and MIS registers.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED															WUECINT	RESERVED				TBMCINT	CBECINT	CBMCINT	TBTOCINT	RESERVED			TAMCINT	RESERVED	CAECINT	CAMCINT	TATOCINT

Bits	Field Name	Description	Type	Reset
31:17	RESERVED	This bit field is reserved.	RO	0x0000
16	WUECINT	GPTM write update error interrupt clear	RW	0
15:12	RESERVED	This bit field is reserved.	RO	0x0
11	TBMCINT	GPTM Timer B match interrupt clear	RW	0
10	CBECINT	GPTM Timer B capture event Interrupt clear	RW	0
9	CBMCINT	GPTM Timer B capture match interrupt clear	RW	0
8	TBTOCINT	GPTM Timer B time-out interrupt clear	RW	0
7:5	RESERVED	This bit field is reserved.	RO	0x0
4	TAMCINT	GPTM Timer A match interrupt clear	RW	0
3	RESERVED	This bit field is reserved.	RW	0
2	CAECINT	GPTM Timer A capture event Interrupt clear	RW	0
1	CAMCINT	GPTM Timer A capture match interrupt clear	RW	0
0	TATOCINT	GPTM Timer A time-out interrupt clear	RW	0

GPTIMER_TAILR

Address offset	0x028		
Physical Address	0x4003 0028 0x4003 2028 0x4003 1028 0x4003 3028	**Instance**	GPTIMER0 GPTIMER2 GPTIMER1 GPTIMER3
Description	GPTM Timer A interval load When the Timer is counting down, this register is used to load the starting count value into the Timer. When the Timer is counting up, this register sets the upper bound for the timeout event. When a GPTM is configured to one of the 32-bit modes, TAILR appears as a 32-bit register (the upper 16-bits correspond to the contents of the GPTM Timer B Interval Load (TBILR) register). In a 16-bit mode, the upper 16 bits of this register read as 0s and have no effect on the state of TBILR.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	TAILR	GPTM A interval load register	RW	0xFFFF FFFF

GPTIMER_TBILR

Address offset	0x02C		
Physical Address	0x4003 002C 0x4003 202C 0x4003 102C 0x4003 302C	**Instance**	GPTIMER0 GPTIMER2 GPTIMER1 GPTIMER3
Description	GPTM Timer B interval load When the Timer is counting down, this register is used to load the starting count value into the Timer. When the Timer is counting up, this register sets the upper bound for the time-out event. When a GPTM is configured to one of the 32-bit modes, the contents of bits [15:0] in this register are loaded into the upper 16 bits of the TAILR register. Reads from this register return the current value of Timer B and writes are ignored. In a 16-bit mode, bits [15:0] are used for the load value. Bits [31:16] are reserved in both cases.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15:0	TBILR	GPTM B interval load register	RW	0xFFFF

GPTIMER_TAMATCHR

Address offset	0x030		
Physical Address	0x4003 0030 0x4003 2030 0x4003 1030 0x4003 3030	**Instance**	GPTIMER0 GPTIMER2 GPTIMER1 GPTIMER3
Description	GPTM Timer A match This register is loaded with a match value. Interrupts can be generated when the Timer value is equal to the value in this register in one-shot or periodic mode. When a GPTM is configured to one of the 32-bit modes, TAMATCHR appears as a 32-bit register (the upper 16-bits correspond to the contents of the GPTM Timer B match (GPTMTBMATCHR) register). In a 16-bit mode, the upper 16 bits of this register read as 0s and have no effect on the state of TBMATCHR.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	TAMR	GPTM Timer A match register	RW	0xFFFF FFFF

GPTIMER_TBMATCHR

Address offset 0x034

Physical Address 0x4003 0034, 0x4003 2034, 0x4003 1034, 0x4003 3034

Instance GPTIMER0, GPTIMER2, GPTIMER1, GPTIMER3

Description PTM Timer B match
This register is loaded with a match value. Interrupts can be generated when the Timer value is equal to the value in this register in one-shot or periodic mode.
When a GPTM is configured to one of the 32-bit modes, the contents of bits [15:0] in this register are loaded into the upper 16 bits of the TAMATCHR register. Reads from this register return the current match value of Timer B and writes are ignored. In a 16-bit mode, bits [15:0] are used for the match value. Bits [31:16] are reserved in both cases.

Type RW

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

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15:0	TBMR	GPTM Timer B match register	RW	0xFFFF

GPTIMER_TAPR

Address offset 0x038

Physical Address 0x4003 0038, 0x4003 2038, 0x4003 1038, 0x4003 3038

Instance GPTIMER0, GPTIMER2, GPTIMER1, GPTIMER3

Description GPTM Timer A prescale
This register allows software to extend the range of the 16-bit Timers in periodic and one-shot modes.

Type RW

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	TAPSR	GPTM Timer A prescale	RW	0x00

GPTIMER_TBPR

Address offset 0x03C

Physical Address 0x4003 003C, 0x4003 203C, 0x4003 103C, 0x4003 303C

Instance GPTIMER0, GPTIMER2, GPTIMER1, GPTIMER3

Description GPTM Timer B prescale
This register allows software to extend the range of the 16-bit Timers in periodic and one-shot modes.

Type RW

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	TBPSR	GPTM Timer B prescale	RW	0x00

GPTIMER_TAPMR

Address offset 0x040

Physical Address 0x4003 0040, 0x4003 2040, 0x4003 1040, 0x4003 3040

Instance GPTIMER0, GPTIMER2, GPTIMER1, GPTIMER3

Description GPTM Timer A prescale match
This register effectively extends the range of TAMATCHR to 24 bits when operating in 16-bit, one-shot or periodic mode.

Type RW

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	TAPSR	GPTM Timer A prescale match	RW	0x00

GPTIMER_TBPMR

Address offset 0x044

Physical Address 0x4003 0044, 0x4003 2044, 0x4003 1044, 0x4003 3044

Instance GPTIMER0, GPTIMER2, GPTIMER1, GPTIMER3

Description GPTM Timer B prescale match
This register effectively extends the range ofMTBMATCHR to 24 bits when operating in 16-bit, one-shot or periodic mode.

Type RW

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	TBPSR	GPTM Timer B prescale match	RW	0x00

GPTIMER_TAR

Address offset	0x048		
Physical Address	0x4003 0048 0x4003 2048 0x4003 1048 0x4003 3048	**Instance**	GPTIMER0 GPTIMER2 GPTIMER1 GPTIMER3
Description	GPTM Timer A This register shows the current value of the Timer A counter. When a GPTM is configured to one of the 32-bit modes, TAR appears as a 32-bit register (the upper 16-bits correspond to the contents of the GPTM Timer B (TBR) register). In the16-bit Input edge count, input edge time, and PWM modes, bits [15:0] contain the value of the counter and bits 23:16 contain the value of the prescaler, which is the upper 8 bits of the count. Bits [31:24] always read as 0. To read the value of the prescaler in 16-bit, one-shot and periodic modes, read bits [23:16] in the TAV register.		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:0	TAR	GPTM Timer A register	RO	0xFFFF FFFF

GPTIMER_TBR

Address offset	0x04C		
Physical Address	0x4003 004C 0x4003 204C 0x4003 104C 0x4003 304C	**Instance**	GPTIMER0 GPTIMER2 GPTIMER1 GPTIMER3
Description	GPTM Timer B This register shows the current value of the Timer B counter. When a GPTM is configured to one of the 32-bit modes, the contents of bits [15:0] in this register are loaded into the upper 16 bits of the TAR register. Reads from this register return the current value of Timer B. In a 16-bit mode, bits 15:0 contain the value of the counter and bits [23:16] contain the value of the prescaler in Input edge count, input edge time, and PWM modes, which is the upper 8 bits of the count. Bits [31:24] always read as 0. To read the value of the prescaler in 16-bit, one-shot and periodic modes, read bits [23:16] in the TBV register.		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15:0	TBR	GPTM Timer B register	RO	0xFFFF

GPTIMER_TAV

Address offset	0x050		
Physical Address	0x4003 0050 0x4003 2050 0x4003 1050 0x4003 3050	**Instance**	GPTIMER0 GPTIMER2 GPTIMER1 GPTIMER3

Description

GPTM Timer A value

When read, this register shows the current, free-running value of Timer A in all modes. Software can use this value to determine the time elapsed between an interrupt and the ISR entry when using the snapshot feature with the periodic operating mode. When written, the value written into this register is loaded into the TAR register on the next clock cycle.

When a GPTM is configured to one of the 32-bit modes, TAV appears as a 32-bit register (the upper 16-bits correspond to the contents of the GPTM Timer B Value (TBV) register). In a 16-bit mode, bits [15:0] contain the value of the counter and bits [23:16] contain the current, free-running value of the prescaler, which is the upper 8 bits of the count in input edge count, input edge time, PWM and one-shot or periodic up count modes. In one-shot or periodic down count modes, the prescaler stored in [23:16] is a true prescaler, meaning bits [23:16] count down before decrementing the value in bits [15:0]. The prescaler its [31:24] always read as 0.

Type RW

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

Bits	Field Name	Description	Type	Reset
31:0	TAV	GPTM Timer A register	RW	0xFFFF FFFF

GPTIMER_TBV

Address offset	0x054		
Physical Address	0x4003 0054 0x4003 2054 0x4003 1054 0x4003 3054	**Instance**	GPTIMER0 GPTIMER2 GPTIMER1 GPTIMER3

Description

GPTM Timer B value

When read, this register shows the current, free-running value of Timer B in all modes. Software can use this value to determine the time elapsed between an interrupt and the ISR entry. When written, the value written into this register is loaded into the TBR register on the next clock cycle.

When a GPTM is configured to one of the 32-bit modes, the contents of bits 15:0 in this register are loaded into the upper 16 bits of the TAV register. Reads from this register return the current free-running value of Timer B. In a 16-bit mode, bits [15:0] contain the value of the counter and bits [23:16] contain the current, free-running value of the prescaler, which is the upper 8 bits of the count in input edge count, input edge time, PWM and one-shot or periodic up count modes. In one-shot or periodic down count modes, the prescaler stored in [23:16] is a true prescaler, meaning bits [23:16] count down before decrementing the value in bits [15:0]. The prescaler its [31:24] always read as 0.

Type RW

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

Bits	Field Name	Description	Type	Reset
31:24	RESERVED	This bit field is reserved.	RO	0x00
23:16	PRE	GPTM Timer B prescale register (16-bit mode)	RO	0x00
15:0	TBV	GPTM Timer B register	RW	0xFFFF

GPTIMER_TAPS

Address offset	0x05C		
Physical Address	0x4003 005C 0x4003 205C 0x4003 105C 0x4003 305C	**Instance**	GPTIMER0 GPTIMER2 GPTIMER1 GPTIMER3
Description	GPTM Timer A prescale snapshot For the 32-bit wide GPTM, this register shows the current value of the Timer A prescaler in the 32-bit modes. This register is ununsed in 16-bit GPTM mode.		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15:0	PSS	GPTM Timer A prescaler	RO	0x0000

GPTIMER_TBPS

Address offset	0x060		
Physical Address	0x4003 0060 0x4003 2060 0x4003 1060 0x4003 3060	**Instance**	GPTIMER0 GPTIMER2 GPTIMER1 GPTIMER3
Description	GPTM Timer B prescale snapshot For the 32-bit wide GPTM, this register shows the current value of the Timer B prescaler in the 32-bit modes. This register is ununsed in 16-bit GPTM mode.		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15:0	PSS	GPTM Timer B prescaler	RO	0x0000

GPTIMER_TAPV

Address offset	0x064		
Physical Address	0x4003 0064 0x4003 2064 0x4003 1064 0x4003 3064	**Instance**	GPTIMER0 GPTIMER2 GPTIMER1 GPTIMER3
Description	GPTM Timer A prescale value For the 32-bit wide GPTM, this register shows the current free-running value of the Timer A prescaler in the 32-bit modes. Software can use this value in conjunction with the TAV register to determine the time elapsed between an interrupt and the ISR entry. This register is ununsed in 16- or 32-bit GPTM mode.		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15:0	PSV	GPTM Timer A prescaler value	RO	0x0000

GPTIMER_TBPV

Address offset	0x068		
Physical Address	0x4003 0068 0x4003 2068 0x4003 1068 0x4003 3068	**Instance**	GPTIMER0 GPTIMER2 GPTIMER1 GPTIMER3
Description	GPTM Timer B prescale value For the 32-bit wide GPTM, this register shows the current free-running value of the Timer B prescaler in the 32-bit modes. Software can use this value in conjunction with the TBV register to determine the time elapsed between an interrupt and the ISR entry. This register is ununsed in 16- or 32-bit GPTM mode.		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15:0	PSV	GPTM Timer B prescaler value	RO	0x0000

GPTIMER_PP

Address offset	0xFC0		
Physical Address	0x4003 0FC0 0x4003 2FC0 0x4003 1FC0 0x4003 3FC0	**Instance**	GPTIMER0 GPTIMER2 GPTIMER1 GPTIMER3
Description	GPTM peripheral properties The PP register provides information regarding the properties of the general-purpose Timer module.		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED	ALTCLK	SYNCNT	CHAIN	SIZE			

Bits	Field Name	Description	Type	Reset
31:7	RESERVED	This bit field is reserved.	RO	0x000 0000
6	ALTCLK	Alternate clock source 0: Timer is not capable of using an alternate clock. 1: Timer is capable of using an alternate clock.	RO	0
5	SYNCNT	Synchronized start 0: Timer is not capable of synchronizing the count value with other timers. 1: Timer is capable of synchronizing the count value with other timers.	RO	0
4	CHAIN	Chain with other timers 0: Timer is not capable of chaining with previously numbered Timers. 1: Timer is capable of chaining with previously numbered timers.	RO	0
3:0	SIZE	Timer size 0: Timer A and Timer B are 16 bits wide with 8-bit prescale. 1: Timer A and Timer B are 32 bits wide with 16-bit prescale.	RO	0x0

MAC Timer

The MAC timer is mainly used:

- To provide timing for 802.15.4 CSMA-CA algorithms and for general timekeeping in the 802.15.4 MAC layer on CC2538 devices
- For general radio timekeeping when running the radio in proprietary mode on the CC2538 device

When the MAC timer is used with the sleep timer, the timing function is provided even when the system enters low-power modes PM1 and PM2. The timer runs at a speed according to the system clock. If the MAC timer is to be used with the sleep timer, the system clock source must be the 32-MHz crystal whenever the MAC timer is running, and an external 32-kHz XOSC should be used for accurate results.

The main features of the MAC timer are:

- 16-bit timer up-counter providing, for example, a symbol/frame period of 16 µs/320 µs
- Adjustable period with accuracy of 31.25 ns
- 2 × 16-bit timer compare function
- 24-bit overflow count
- 2 × 24-bit overflow compare function
- Start-of-frame-delimiter capture function
- Timer start/stop synchronous with 32-kHz clock and timekeeping maintained by the sleep timer
- Interrupts generated on compare and overflow
- DMA trigger capability
- Ability to adjust timer value while counting by introducing delay

Topic **Page**

12.1 Timer Operation

This section describes the operation of the timer.

12.1.1 General

After a reset, the timer is in the timer idle mode, where it is stopped. The timer starts running when the **RUN** bit in the **RFCORE_SFR_MTCTRL** register is set to 1. The timer then enters the timer run mode. Either the entry is immediate, or it is performed synchronously with the 32-kHz clock. For a description of the synchronous start and stop mode, see Section 12.4, *Timer Start and Stop Synchronization*.

Once the timer is running in run mode, it can be stopped by setting the **RUN** bit in the **RFCORE_SFR_MTCTRL** register to 0. The timer then enters the timer idle mode. The stopping of the timer is performed either immediately or synchronously with the 32-kHz clock.

12.1.2 Up Counter

The MAC timer contains a 16-bit timer, which increments on each clock cycle. The counter value can be read from the registers **RFCORE_SFR_MTM0** and **RFCORE_SFR_MTM1** with the **MTMSEL** bit in the **RFCORE_SFR_MTMSEL** register set to 000. The register content in the **RFCORE_SFR_MTM1** register is latched when the **RFCORE_SFR_MTM0** register is read, meaning that the **RFCORE_SFR_MTM0** register must always be read first.

When the timer is idle, writing to registers **RFCORE_SFR_MTM1** and RFCORE_SFR_**MTM0** with the **MTMSEL** bit in the **RFCORE_SFR_MTMSEL** register set to 000 can modify the counter. The **RFCORE_SFR_MTM0** register must be written first.

12.1.3 Timer Overflow

At the same time as the timer counts to a value that is equal to the set timer period, a timer overflow occurs. When the timer overflow occurs, the timer is set to 0x0000. If the overflow interrupt mask bit (the **MACTIMER_PERM** bit in the **RFCORE_SFR_MTIRQM** register) is set to 1, an interrupt request is generated. The interrupt flag bit (the **MACTIMER_PERF** bit in the **RFCORE_SFR_MTIRQF** register) is set to 1, regardless of the interrupt mask value.

12.1.4 Timer Delta Increment

The timer period may be adjusted once during a timer period by writing a timer delta value. When the timer is running and a timer delta value is written to multiplexed registers **RFCORE_SFR_MTM1** and **RFCORE_SFR_MTM0** with the **MTMSEL** bit in the **RFCORE_SFR_MTMSEL** register set to 000, the 16-bit timer halts at its current value and a delta counter starts counting. The **RFCORE_SFR_MTM0** register must be written before **RFCORE_SFR_MTM1**. The delta counter starts counting from the delta value written, down to 0. When the delta counter reaches zero, the 16-bit timer starts counting again.

The delta counter decrements at the same rate as the timer. When the delta counter reaches 0, it does not start counting again until a delta value is written again. In this way, a timer period may be increased by the delta value to make adjustments to the timer overflow events over time.

12.1.5 Timer Compare

A timer compare occurs at the same time as the timer counts to a value that is equal to one of the 16-bit compare values set. When a timer compare occurs, the interrupt flag (the **MACTIMER_COMPARE1F** bit in the **RFCORE_SFR_MTIRQF** register or the bit **MACTIMER_COMPARE2F** in the **RFCORE_SFR_MTIRQF** register) is set to 1, depending of which compare value is reached. An interrupt request is also generated if the corresponding interrupt mask in the **MACTIMER_COMPARE1M** bit in the **RFCORE_SFR_MTIRQM** register or the **MACTIMER_COMPARE2M** bit in the **RFCORE_SFR_MTIRQM** register is set to 1.

12.1.6 Overflow Count

At each timer overflow, the 24-bit overflow counter is incremented by 1. The overflow counter value is read through registers **RFCORE_SFR_MTMOVF2**, **RFCORE_SFR_MTMOVF1**, and **RFCORE_SFR_MTMOVF0** with the **MTMOVEFSEL** bit in the **RFCORE_SFR_MTMSEL** register set to 000. The registers are latched as in the following description.

To achieve a unique timestamp, where the timer and overflow counters are latched at the same time, read the **RFCORE_SFR_MTM0** register with the **MTMSEL** bit in the **RFCORE_SFR_MTMSEL** register set to 000 and the **LATCH_MODE** bit in the **RFCORE_SFR_MTCTRL** register set to 1. This read returns the low byte of the timer value, and also latches the high byte of the timer and the entire overflow counter, so the rest of the timestamp is ready to be read.

To read just the overflow counter without reading timer first, read the **RFCORE_SFR_MTMOVF0** register with the **MTMOVFSEL** bit in the **RFCORE_SFR_MTMSEL** register set to 000 and the **LATCH_MODE** bit in the **RFCORE_SFR_MTCTRL** register set to 0. This read returns the low byte of the overflow counter, and latches the two most-significant bytes (MSBytes) of the overflow counter so that the values are ready to be read.

12.1.7 Overflow-Count Update

To update the overflow count value, write to registers **RFCORE_SFR_MTMOVF2**, **RFCORE_SFR_MTMOVF1**, and **RFCORE_SFR_MTMOVF0** with the **MTMOVFSEL** bit in the **RFCORE_SFR_MTMSEL** register set to 000. Always write the least-significant byte (LSBytes) first, and always write all three bytes. The write takes effect once the high byte is written.

12.1.8 Overflow-Count Overflow

At the same time as the overflow counter counts to a value that is equal to the overflow period setting, an overflow period event occurs. When the period event occurs, the overflow counter is set to 0x00 0000. If the overflow interrupt mask bit (the **MACTIMER_OVF_PERM** bit in the **RFCORE_SFR_MTIRQM** register) is 1, an interrupt request is generated. The interrupt flag bit (the **MACTIMER_OVF_PERF** bit in the **RFCORE_SFR_MTIRQF** register is set to 1, regardless of the interrupt mask value.

12.1.9 Overflow-Count Compare

Two compare values may be set for the overflow counter. The compare values are set by writing to **RFCORE_SFR_MTMOVF2**, **RFCORE_SFR_MTMOVF1**, and **RFCORE_SFR_MTMOVF0** with the **TMTMOVFSEL** bit in the **RFCORE_SFR_MTMSEL** register set to 011 or 100. At the same time as the overflow counter counts to a value equal to one of the overflow count compare values, an overflow count compare event occurs. If the corresponding overflow compare interrupt mask bit (the **MACTIMER_OVF_COMPARE1M** bit in the **RFCORE_SFR_MTIRQM** register or the **MACTIMER_OVF_COMPARE2M** bit in the **RFCORE_SFR_MTIRQM** register) is set to 1, an interrupt request is generated. The interrupt flags bit (the **MACTIMER_OVF_COMPARE1F** bit in the **RFCORE_SFR_MTIRQF** register and the **MACTIMER_OVF_COMPARE2F** bit in the **RFCORE_SFR_MTIRQF** register are set to 1, regardless of the interrupt mask value.

12.1.10 Capture Input

The MAC timer has a timer capture function, which captures the time when the start-of-frame delimiter (SFD) status in the radio goes high.

When the capture event occurs, the current timer value is captured in the capture register. The capture value can be read from registers **RFCORE_SFR_MTM1** and **RFCORE_SFR_MTM0** if the **MTMSEL** bit in the **MTMSEL** register is set to 001. The value of the overflow count is also captured at the time of the capture event and can be read from registers **RFCORE_SFR_MTMOVF2**, **RFCORE_SFR_MTMOVF1**, and **RFCORE_SFR_MTMOVF0** if the **MTMOVFSEL** bit in the **RFCORE_SFR_MTMSEL** register is set to 001.

12.2 Interrupts

The timer has six individually maskable interrupt sources. These are the following:

- Timer overflow
- Timer compare 1
- Timer compare 2
- Overflow-count overflow
- Overflow-count compare 1
- Overflow-count compare 2

The interrupt flags are given in the **RFCORE_SFR_MTIRQF** registers. The interrupt flag bits are set only by hardware and are cleared only by writing to the SFR register.

Each interrupt source can be masked by its corresponding mask bit in the **RFCORE_SFR_MTIRQM** register. An interrupt is generated when the corresponding mask bit is set; otherwise, the interrupt is not generated. The interrupt flag bit is set, however, regardless of the state of the interrupt mask bit.

12.3 Event Outputs (DMA Trigger and Radio Events)

The MAC timer has two event outputs, **MT_EVENT1** and **MT_EVENMT**. These can be used as DMA triggers, as inputs to the radio, for conditions in conditional instructions in the CSP on CC2538, or for timing TX or RX in CC2538 when running the radio in proprietary mode. The event outputs can be configured individually to any of the following events:

- Timer overflow
- Timer compare 1
- Timer compare 2
- Overflow-count overflow
- Overflow-count compare 1
- Overflow-count compare 2

The DMA triggers are configured using the **MACTIMER_EVENT1_CFG** bit in the **RFCORE_SFR_MTCSPCFG** register and the **MACTIMER_EVENMT_CFG** bit in the **RFCORE_SFR_MTCSPCFG** register.

12.4 Timer Start and Stop Synchronization

This section describes the synchronized timer start and stop.

12.4.1 General

The timer can be started and stopped synchronously with the 32-kHz clock rising edge. Note that this event is derived from a 32-kHz clock signal, but is synchronous with the 32-MHz system clock and thus has a period approximately equal to that of the 32-kHz clock period. Do not attempt synchronous starting and stopping unless both the 32-kHz clock and 32-MHz XOSC are running and stable.

At the time of a synchronous start, the timer is reloaded with new calculated values for the timer and overflow count such that it appears that the timer has not been stopped.

12.4.2 Timer Synchronous Stop

After the timer starts running (that is, enters timer run mode), it is stopped synchronously by writing 0 to the **RUN** bit in the **RFCORE_SFR_MTCTRL** register when the **SYNC** bit in the **RFCORE_SFR_MTCTRL** register is 1. After the **RUN** bit in the **RFCORE_SFR_MTCTRL** register is set to 0, the timer continues running until the 32-kHz clock rising edge is sampled as 1. When this occurs, the timer stops, the current sleep timer value is stored, and the **STATE** bit in the **RFCORE_SFR_MTCTRL** register goes from 1 to 0.

12.4.3 Timer Synchronous Start

When the timer is in idle mode, it is started synchronously by writing 1 to the **RUN** bit in the **RFCORE_SFR_MTCTRL** register when the **SYNC** bit in the **RFCORE_SFR_MTCTRL** register is 1. After the **RUN** bit in the **RFCORE_SFR_MTCTRL** register is set to 1, the timer remains in idle mode until the 32-kHz clock rising edge is detected. When this occurs, the timer first calculates new values for the 16-bit timer value and for the 24-bit timer overflow count, based on the current and stored sleep timer values and the current 16-bit timer values. The new MAC timer and overflow count values are loaded into the timer, and the timer enters the run mode. When the **STATE** bit in the **RFCORE_SFR_MTCTRL** register is set to 1, this bit indicates that the module is running. This synchronous start process takes 86 clock cycles from the time when the 32-kHz clock rising edge is sampled high. The synchronous start-and-stop function requires that the system clock frequency is selected to be 32 MHz. If the 16-MHz clock is selected, an offset is added to the new calculated value.

If a synchronous start is done without a previous synchronous stop, the timer is loaded with unpredictable values. To avoid these unpredictable values, do the first start of the timer asynchronously, then enable synchronous mode for subsequent stops and starts.

The method for calculating the new MAC timer value and overflow-count value is given as follows. Because the MAC timer and sleep timer clocks are asynchronous with a noninteger clock ratio, there is a maximum error of ±1 in the calculated timer value compared to the ideal timer value, not considering clock inaccuracies.

Calculation of New Timer Value and Overflow Count Value

N_c = Current sleep timer value

N_{ST} = Stored sleep timer value

K_{ck} = Clock ratio = 976.5625[1]

stw = Sleep timer width = 24

P_T = Mac timer period

P_{OVF} = Overflow period

O_{ST} = Stored overflow-count value

O_{TICK} = Overflow tics while sleeping

t_{ST} = Stored timer value

T_{OH} = Overhead = 86

$N_t = N_c - N_{ST}$

$N_t \leq 0 \rightarrow N_d = 2^{stw} + N_t$; $N_t > 0 \rightarrow N_d = N_t$

$C = N_d \times K_{ck} + T_{ST} + T_{OH}$ (rounded to nearest integer value)

$T = C \bmod P_T$

MAC timer value = T

$$O_{TICK} = \frac{(C - T)}{P_T}$$

$O = (O_{TICK} + O_{ST}) \bmod P_{OVF}$

MACtimerOverflowCount = O

[1] Clock ratio of the MAC timer clock frequency (32 MHz) and sleep timer clock frequency (32 kHz)

For a given MAC timer period value, P_T, there is a maximum duration between the MAC timer synchronous stop and start for which the timer value is correctly updated after starting. The maximum value is given in terms of the number of sleep timer clock periods; that is, 32-kHz clock periods, $t_{ST(max)}$.

$$t_{ST(max)} \leq \frac{(2^{24} - 1) \times P_T + T_{OH}}{K_{ck}}$$

12.5 MAC Timer Registers

12.5.1 RFCORE_SFR Registers

12.5.1.1 RFCORE_SFR Registers Mapping Summary

This section provides information on the RFCORE_SFR module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 12-1. RFCORE_SFR Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
RFCORE_SFR_MTCSPCFG	RW	32	0x0000 0000	0x00	0x4008 8800
RFCORE_SFR_MTCTRL	RW	32	0x0000 0002	0x04	0x4008 8804

Table 12-1. RFCORE_SFR Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
RFCORE_SFR_MTIRQM	RW	32	0x0000 0000	0x08	0x4008 8808
RFCORE_SFR_MTIRQF	RW	32	0x0000 0000	0x0C	0x4008 880C
RFCORE_SFR_MTMSEL	RW	32	0x0000 0000	0x10	0x4008 8810
RFCORE_SFR_MTM0	RW	32	0x0000 0000	0x14	0x4008 8814
RFCORE_SFR_MTM1	RW	32	0x0000 0000	0x18	0x4008 8818
RFCORE_SFR_MTMOVF2	RW	32	0x0000 0000	0x1C	0x4008 881C
RFCORE_SFR_MTMOVF1	RW	32	0x0000 0000	0x20	0x4008 8820
RFCORE_SFR_MTMOVF0	RW	32	0x0000 0000	0x24	0x4008 8824

12.5.1.2 RFCORE_SFR Register Descriptions

RFCORE_SFR_MTCSPCFG

Address offset	0x00		
Physical Address	0x4008 8800	**Instance**	RFCORE_SFR
Description	MAC Timer event configuration		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED	MACTIMER_EVENMT_CFG			RESERVED	MACTIMER_EVENT1_CFG		

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RW	0x00 0000
7	RESERVED	This bit field is reserved.	RO	0
6:4	MACTIMER_EVENMT_CFG	Selects the event that triggers an MT_EVENT2 pulse 000: MT_per_event 001: MT_cmp1_event 010: MT_cmp2_event 011: MTovf_per_event 100: MTovf_cmp1_event 101: MTovf_cmp2_event 110: Reserved 111: No event	RW	0x0
3	RESERVED	This bit field is reserved.	RO	0

Bits	Field Name	Description	Type	Reset
2:0	MACTIMER_EVENT1_CFG	Selects the event that triggers an MT_EVENT1 pulse 000: MT_per_event 001: MT_cmp1_event 010: MT_cmp2_event 011: MTovf_per_event 100: MTovf_cmp1_event 101: MTovf_cmp2_event 110: Reserved 111: No event	RW	0x0

RFCORE_SFR_MTCTRL

Address offset	0x04		
Physical Address	0x4008 8804	**Instance**	RFCORE_SFR
Description	MAC Timer control register		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED				LATCH_MODE	STATE	SYNC	RUN

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RW	0x00 0000
7:4	RESERVED	This bit field is reserved.	RO	0x0
3	LATCH_MODE	0: Reading MTM0 with MTMSEL.MTMSEL = 000 latches the high byte of the timer, making it ready to be read from MTM1. Reading MTMOVF0 with MTMSEL.MTMOVFSEL = 000 latches the two most-significant bytes of the overflow counter, making it possible to read these from MTMOVF1 and MTMOVF2. 1: Reading MTM0 with MTMSEL.MTMSEL = 000 latches the high byte of the timer and the entire overflow counter at once, making it possible to read the values from MTM1, MTMOVF0, MTMOVF1, and MTMOVF2.	RW	0
2	STATE	State of MAC Timer 0: Timer idle 1: Timer running	RO	0
1	SYNC	0: Starting and stopping of timer is immediate; that is, synchronous with clk_rf_32m. 1: Starting and stopping of timer occurs at the first positive edge of the 32-kHz clock. For more details regarding timer start and stop, see Section 22.4.	RW	1
0	RUN	Write 1 to start timer, write 0 to stop timer. When read, it returns the last written value.	RW	0

RFCORE_SFR_MTIRQM

Address offset	0x08		
Physical Address	0x4008 8808	**Instance**	RFCORE_SFR
Description	MAC Timer interrupt mask		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED		MACTIMER_OVF_COMPARE2M	MACTIMER_OVF_COMPARE1M	MACTIMER_OVF_PERM	MACTIMER_COMPARE2M	MACTIMER_COMPARE1M	MACTIMER_PERM

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RW	0x00 0000
7:6	RESERVED	This bit field is reserved.	RO	0x0
5	MACTIMER_OVF_COMPARE2M	Enables the MACTIMER_OVF_COMPARE2 interrupt	RW	0
4	MACTIMER_OVF_COMPARE1M	Enables the MACTIMER_OVF_COMPARE1 interrupt	RW	0
3	MACTIMER_OVF_PERM	Enables the MACTIMER_OVF_PER interrupt	RW	0
2	MACTIMER_COMPARE2M	Enables the MACTIMER_COMPARE2 interrupt	RW	0
1	MACTIMER_COMPARE1M	Enables the MACTIMER_COMPARE1 interrupt	RW	0
0	MACTIMER_PERM	Enables the MACTIMER_PER interrupt	RW	0

RFCORE_SFR_MTIRQF

Address offset	0x0C		
Physical Address	0x4008 880C	**Instance**	RFCORE_SFR
Description	MAC Timer interrupt flags		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED		MACTIMER_OVF_COMPARE2F	MACTIMER_OVF_COMPARE1F	MACTIMER_OVF_PERF	MACTIMER_COMPARE2F	MACTIMER_COMPARE1F	MACTIMER_PERF

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RW	0x00 0000
7:6	RESERVED	This bit field is reserved.	RO	0x0
5	MACTIMER_OVF_COMPARE2F	Set when the MAC Timer overflow counter counts to the value set at MTovf_cmp2	RW	0
4	MACTIMER_OVF_COMPARE1F	Set when the MAC Timer overflow counter counts to the value set at Timer 2 MTovf_cmp1	RW	0

Bits	Field Name	Description	Type	Reset
3	MACTIMER_OVF_PERF	Set when the MAC Timer overflow counter would have counted to a value equal to MTovf_per, but instead wraps to 0	RW	0
2	MACTIMER_COMPARE2F	Set when the MAC Timer counter counts to the value set at MT_cmp2	RW	0
1	MACTIMER_COMPARE1F	Set when the MAC Timer counter counts to the value set at MT_cmp1	RW	0
0	MACTIMER_PERF	Set when the MAC Timer counter would have counted to a value equal to MT_per, but instead wraps to 0	RW	0

RFCORE_SFR_MTMSEL

Address offset	0x10		
Physical Address	0x4008 8810	**Instance**	RFCORE_SFR
Description	MAC Timer multiplex select		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	RESERVED	This bit field is reserved.	RO	0
6:4	MTMOVFSEL	The value of this register selects the internal registers that are modified or read when accessing MTMOVF0, MTMOVF1, and MTMOVF2. 000: MTovf (overflow counter) 001: MTovf_cap (overflow capture) 010: MTovf_per (overflow period) 011: MTovf_cmp1 (overflow compare 1) 100: MTovf_cmp2 (overflow compare 2) 101 to 111: Reserved	RW	0x0
3	RESERVED	This bit field is reserved.	RO	0
2:0	MTMSEL	The value of this register selects the internal registers that are modified or read when accessing MTM0 and MTM1. 000: MTtim (timer count value) 001: MT_cap (timer capture) 010: MT_per (timer period) 011: MT_cmp1 (timer compare 1) 100: MT_cmp2 (timer compare 2) 101 to 111: Reserved MTM0	RW	0x0

RFCORE_SFR_MTM0

Address offset	0x14		
Physical Address	0x4008 8814	**Instance**	RFCORE_SFR
Description	MAC Timer multiplexed register 0		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								MTM0							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RW	0x00 0000
7:0	MTM0	Indirectly returns and modifies bits [7:0] of an internal register depending on the value of MTMSEL.MTMSEL. When reading the MTM0 register with MTMSEL.MTMSEL set to 000 and MTCTRL.LATCH_MODE set to 0, the timer (MTtim) value is latched. When reading the MTM0 register with MTMSEL.MTMSEL set to 000 and MTCTRL.LATCH_MODE set to 1, the timer (MTtim) and overflow counter (MTovf) values are latched.	RW	0x00

RFCORE_SFR_MTM1

Address offset	0x18		
Physical Address	0x4008 8818	**Instance**	RFCORE_SFR
Description	MAC Timer multiplexed register 1		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								MTM1							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	MTM1	Indirectly returns and modifies bits [15:8] of an internal register, depending on the value of MTMSEL.MTMSEL. When reading the MTM0 register with MTMSEL.MTMSEL set to 000, the timer (MTtim) value is latched. Reading this register with MTMSEL.MTMSEL set to 000 returns the latched value of MTtim[15:8].	RW	0x00

RFCORE_SFR_MTMOVF2

Address offset	0x1C		
Physical Address	0x4008 881C	**Instance**	RFCORE_SFR
Description	MAC Timer multiplexed overflow register 2		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								MTMOVF2							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RW	0x00 0000
7:0	MTMOVF2	Indirectly returns and modifies bits [23:16] of an internal register, depending on the value of MTMSEL.MTMOVFSEL. Reading this register with MTMSEL.MTMOVFSEL set to 000 returns the latched value of MTovf[23:16].	RW	0x00

RFCORE_SFR_MTMOVF1

Address offset	0x20		
Physical Address	0x4008 8820	**Instance**	RFCORE_SFR
Description	MAC Timer multiplexed overflow register 1		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								MTMOVF1							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RW	0x00 0000
7:0	MTMOVF1	Indirectly returns and modifies bits [15:8] of an internal register, depending on the value of MTMSEL.MTMSEL. Reading this register with MTMSEL.MTMOVFSEL set to 000 returns the latched value of MTovf[15:8].	RW	0x00

RFCORE_SFR_MTMOVF0

Address offset	0x24		
Physical Address	0x4008 8824	**Instance**	RFCORE_SFR
Description	MAC Timer multiplexed overflow register 0		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								MTMOVF0							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RW	0x00 0000
7:0	MTMOVF0	Indirectly returns and modifies bits [7:0] of an internal register, depending on the value of MTMSEL.MTMOVFSEL. When reading the MTMOVF0 register with MTMSEL.MTMOVFSEL set to 000 and MTCTRL.LATCH_MODE set to 0, the overflow counter value (MTovf) is latched. When reading the MTM0 register with MTMSEL.MTMOVFSEL set to 000 and MTCTRL.LATCH_MODE set to 1, the overflow counter value (MTovf) is latched.	RW	0x00

Sleep Timer

The sleep timer is used to set the period during which the system enters and exits low-power modes PM1 and PM2. The sleep timer is also used to maintain timing in MAC timer when entering power mode PM1 or PM2.

The main features of the sleep timer are the following:

- 32-bit timer up-counter operating at 32-kHz clock rate
- 32-bit compare with interrupt and DMA trigger
- 32-bit capture
- Synchronization with GPTimer2 (see Section 11.3.4)

Topic **Page**

13.1 General

The sleep timer is a 32-bit timer running on the 32-kHz clock (Clk32k). The timer starts running immediately after a reset and continues to run uninterrupted.

The current value of the timer can be read from registers **SMWDTHROSC_ST3, SMWDTHROSC_ST2**, **SMWDTHROSC_ST1**, and **SMWDTHROSC_ST0**. When **SMWDTHROSC_ST0** is read, the current value of the 32-bit counter is latched. Thus, the **SMWDTHROSC_ST0** register must be read before **SMWDTHROSC_ST1**, **SMWDTHROSC_ST2**, and **SMWDTHROSC_ST3** to read a correct sleep timer count value.

The sleep timer runs when operating in all power modes except PM3. The value of the sleep timer is not preserved in PM3. When returning from PM1 or PM2 (where the system clock is shut down), the sleep timer value in **SMWDTHROSC_ST3**, **SMWDTHROSC_ST2**, **SMWDTHROSC_ST1**, and **SMWDTHROSC_ST0** is not up-to-date until a positive edge on the 32-kHz clock has been detected after the system clock restarted. To ensure an updated value is read, wait for a positive transition on the 32-kHz clock by polling the **SYS_CTRL_CLOCK_STA.SYNC_32K** bit, before reading the sleep timer value.

NOTE: A supply voltage below 2 V while in PM2 might affect sleep interval.

13.2 Timer Compare

A timer compare event occurs when the timer value is equal to the 32-bit compare value and there is a positive edge on the 32-kHz clock. The compare value is set by writing to registers **SMWDTHROSC_ST3**, **SMWDTHROSC_ST2**, **SMWDTHROSC_ST1**, and **SMWDTHROSC_ST0**. Writing to **SMWDTHROSC_ST0** while **SMWDTHROSC_STLOAD.STLOAD** is 1 initiates loading of the new compare value (that is, the most-recent values written to the **SMWDTHROSC_ST3**, **SMWDTHROSC_ST2**, **SMWDTHROSC_ST1**, and **SMWDTHROSC_ST0** registers). This means that when writing a compare value, **SMWDTHROSC_ST3**, **SMWDTHROSC_ST2**, and **SMWDTHROSC_ST1** must be written before **SMWDTHROSC_ST0**. **SMWDTHROSC_STLOAD.STLOAD** is 0 during the load, and software must not start a new load until **SMWDTHROSC_STLOAD.STLOAD** has flipped back to 1.

When setting a new compare value, the value must be at least 5 more than the current sleep timer value. Otherwise, the timer compare event may be lost.

The interrupt enable bit for the ST interrupt is **EN1.INT[0]** , and the interrupt flag is **PEND1.INT[0]** . When a timer compare event occurs, the interrupt flag **PEND1.INT[0]** is asserted.

In PM1 and PM2, the sleep timer compare event may be used to wake up the device and return to active operation in active mode. The default value of the compare value after reset is 0xFFFF FFFF.

13.3 Timer Capture

The timer capture occurs when the interrupt flag for a selected I/O pin is set and the 32-kHz clock has detected this . Sleep timer capture is enabled by setting **SMWDTHROSC_ST3STCC.PORT[1:0]** and **SMWDTHROSC_ST3STCC.PIN[2:0]** to the I/O pin that will trigger the capture. When the **VALID** bit in the **SMWDTHROSC_STCS** register goes high, the capture value in **SMWDTHROSC_STCV3**, **SMWDTHROSC_STCV2**, **SMWDTHROSC_STCV1**, and **SMWDTHROSC_STCV0** can be read. The captured value is one more than the value at the instant for the event on the I/O pin. Software should therefore subtract 1 from the captured value if absolute timing is required. To enable a new capture, follow these steps:

1. Clear the **VALID** in the **SMWDTHROSC_STCS** register.
2. Wait until the **SYNC_32K** bit in the **SYS_CTRL_CLOCK_STA** register is low.
3. Wait until the **SYNC_32K** bit in the **SYS_CTRL_CLOCK_STA.** register is high.
4. Clear the pin interrupt flag on the **PEND1** Interrupt 32–63 Set Pending (PEND1).

Figure 13-1 shows this sequence, using a rising edge input as an example. Failure to follow the procedure may cause the capture functionality to stop working until a chip reset.

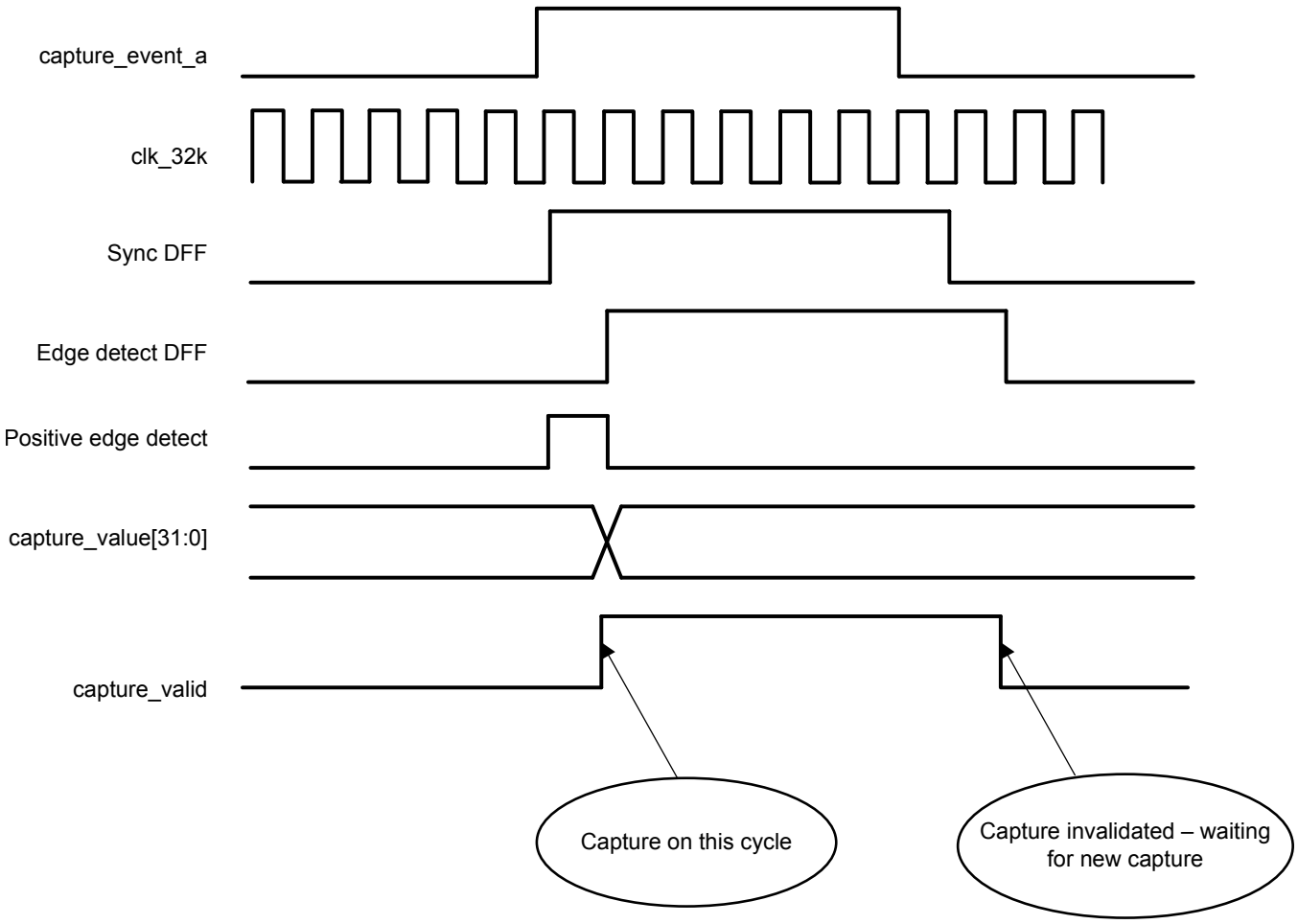


Figure 13-1. Sleep timer Capture

It is not possible to switch the input-capture pin while capture is enabled. Capture must be disabled before a new input-capture pin can be selected. To disable capture, follow these steps (the procedure disables interrupts for up to half of a 32-kHz cycle, or 15.26 µs):

1. Disable interrupts
2. Wait until the **SYNC_32K** bit in the **SYS_CTRL_CLOCK_STA** register is high.
3. Set the **PORT[1:0]** bit field in the **SMWDTHROSC_STCC** register to 3; this disables capture.

13.4 Sleep Timer Registers

13.4.1 SMWDTHROSC Registers

13.4.1.1 SMWDTHROSC Registers Mapping Summary

This section provides information on the SMWDTHROSC module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 13-1. SMWDTHROSC Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
SMWDTHROSC_ST0	RW	32	0x0000 0000	0x40	0x400D 5040
SMWDTHROSC_ST1	RW	32	0x0000 0000	0x44	0x400D 5044
SMWDTHROSC_ST2	RW	32	0x0000 0000	0x48	0x400D 5048
SMWDTHROSC_ST3	RW	32	0x0000 0000	0x4C	0x400D 504C
SMWDTHROSC_STLOAD	RO	32	0x0000 0001	0x50	0x400D 5050
SMWDTHROSC_STCC	RW	32	0x0000 0038	0x54	0x400D 5054
SMWDTHROSC_STCS	RW	32	0x0000 0000	0x58	0x400D 5058
SMWDTHROSC_STCV0	RO	32	0x0000 0000	0x5C	0x400D 505C
SMWDTHROSC_STCV1	RO	32	0x0000 0000	0x60	0x400D 5060
SMWDTHROSC_STCV2	RO	32	0x0000 0000	0x64	0x400D 5064
SMWDTHROSC_STCV3	RO	32	0x0000 0000	0x68	0x400D 5068

13.4.1.2 SMWDTHROSC Register Descriptions

SMWDTHROSC_ST0

Address offset	0x40		
Physical Address	0x400D 5040	**Instance**	SMWDTHROSC
Description	Sleep Timer 0 count and compare		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								ST0							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	ST0	Sleep Timer count and compare value. When read, this register returns the low bits [7:0] of the Sleep Timer count. When writing this register sets the low bits [7:0] of the compare value.	RW	0x00

SMWDTHROSC_ST1

Address offset	0x44		
Physical Address	0x400D 5044	**Instance**	SMWDTHROSC
Description	Sleep Timer 1 count and compare		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								ST1							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	ST1	Sleep Timer count and compare value When read, this register returns the middle bits [15:8] of the Sleep Timer count. When writing this register sets the middle bits [15:8] of the compare value. The value read is latched at the time of reading register ST0. The value written is latched when ST0 is written.	RW	0x00

SMWDTHROSC_ST2

Address offset	0x48		
Physical Address	0x400D 5048	**Instance**	SMWDTHROSC
Description	Sleep Timer 2 count and compare		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								ST2							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	ST2	Sleep Timer count and compare value When read, this register returns the high bits [23:16] of the Sleep Timer count. When writing this register sets the high bits [23:16] of the compare value. The value read is latched at the time of reading register ST0. The value written is latched when ST0 is written.	RW	0x00

SMWDTHROSC_ST3

Address offset	0x4C		
Physical Address	0x400D 504C	**Instance**	SMWDTHROSC
Description	Sleep Timer 3 count and compare		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								ST3							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	ST3	Sleep Timer count and compare value When read, this register returns the high bits [31:24] of the Sleep Timer count. When writing this register sets the high bits [31:24] of the compare value. The value read is latched at the time of reading register ST0. The value written is latched when ST0 is written.	RW	0x00

SMWDTHROSC_STLOAD

Address offset	0x50		
Physical Address	0x400D 5050	**Instance**	SMWDTHROSC
Description	Sleep Timer load status		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:1	RESERVED	This bit field is reserved.	RO	0x00
0	STLOAD	Status signal for when STx registers have been uploaded to 32-kHz counter. 1: Load is complete 0: Load is busy and STx regs are blocked for writing	RO	1

SMWDTHROSC_STCC

Address offset	0x54		
Physical Address	0x400D 5054	**Instance**	SMWDTHROSC
Description	Sleep Timer Capture control		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:6	RESERVED	This bit field is reserved.	RO	0x000 0000
5:3	PORT	Port select Valid settings are 0-3, all others inhibit any capture from occurring 000: Port A selected 001: Port B selected 010: Port C selected 011: Port D selected	RW	0x7
2:0	PIN	Pin select Valid settings are 1-7 when either port A, B, C, or D is selected.	RW	0x0

SMWDTHROSC_STCS

Address offset	0x58		
Physical Address	0x400D 5058	**Instance**	SMWDTHROSC
Description	Sleep Timer Capture status		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:1	RESERVED	This bit field is reserved.	RO	0x00
0	VALID	Capture valid flag Set to 1 when capture value in STCV has been updated Clear explicitly to allow new capture	RW	0

SMWDTHROSC_STCV0

Address offset	0x5C		
Physical Address	0x400D 505C	**Instance**	SMWDTHROSC
Description	Sleep Timer Capture value byte 0		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								STCV0							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	STCV0	Bits [7:0] of Sleep Timer capture value	RO	0x00

SMWDTHROSC_STCV1

Address offset	0x60		
Physical Address	0x400D 5060	**Instance**	SMWDTHROSC
Description	Sleep Timer Capture value byte 1		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								STCV1							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	STCV1	Bits [15:8] of Sleep Timer capture value	RO	0x00

SMWDTHROSC_STCV2

Address offset	0x64		
Physical Address	0x400D 5064	**Instance**	SMWDTHROSC
Description	Sleep Timer Capture value byte 2		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								STCV2							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	STCV2	Bits [23:16] of Sleep Timer capture value	RO	0x00

SMWDTHROSC_STCV3

Address offset	0x68		
Physical Address	0x400D 5068	**Instance**	SMWDTHROSC
Description	Sleep Timer Capture value byte 3		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								STCV3							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	STCV3	Bits [32:24] of Sleep Timer capture value	RO	0x00

Watchdog Timer

The watchdog timer (WDT) module is intended as a recovery method in situations that may subject the CPU to a software upset. The WDT resets the system when software fails to clear the WDT within the selected time interval. The watchdog can be used in applications that are subject to electrical noise, power glitches, electrostatic discharge, and so forth, or where high reliability is required. The WDT requires that both the 32-kHz watchdog clock and the 32-MHz system clock are running when not in power-down mode.

The Watchdog Timer has four selectable timer intervals.

The operation of the WDT is controlled by the **SMWDTHROSC_WDCTL** register. The WDT consists of a 15-bit counter clocked by the 32-kHz clock source. The contents of the 15-bit counter are not user-accessible. The contents of the 15-bit counter are retained during all power modes, and the WDT continues counting when entering active mode again.

Topic **Page**

14.1 Watchdog Timer

The WDT is disabled after a system reset. To start the WDT in watchdog mode, set the **EN** bit in the **SMWDTHROSC_WDCTL** register to 1, and the **MODE** bit in the **SMWDTHROSC_WDCTL** register to 0. The watchdog timer counter then starts incrementing from 0. When the timer is enabled in watchdog mode, it is not possible to disable the timer. Therefore, setting the **EN** bit in the **SMWDTHROSC_WDCTL** register to 0 has no effect if the WDT is already operating in watchdog mode.

The WDT operates with a watchdog timer clock frequency of 32.768 kHz (when the 32-kHz XOSC is used, and this makes T_{WDT} = 0.0305 ms). This clock frequency gives time-out periods equal to 1.9 ms, 15.625 ms, 0.25 s, and 1 s, corresponding to the count value settings 64, 512, 8192, and 32,768, respectively.

If the counter reaches the selected timer interval value, the WDT generates a reset signal for the system. If a watchdog clear sequence is performed before the counter reaches the selected timer interval value, the counter is reset to 0 and continues incrementing its value. The watchdog clear sequence consists of writing 0xA to the **CLR[3:0]** bit in the **SMWDTHROSC_WDCTL** register, followed by writing 0x5 to the same register bits within one watchdog clock period (T_{WDT}). If this complete sequence is not performed before the end of the watchdog period, the watchdog timer generates a reset signal for the system.

14.2 Watchdog Timer Registers

14.2.1 SMWDTHROSC Registers

14.2.1.1 SMWDTHROSC Registers Mapping Summary

This section provides information on the SMWDTHROSC module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 14-1. SMWDTHROSC Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
SMWDTHROSC_WDCTL	RW	32	0x0000 0000	0x00	0x400D 5000

14.2.1.2 SMWDTHROSC Register Descriptions

SMWDTHROSC_WDCTL

Address offset	0x00		
Physical Address	0x400D 5000	**Instance**	SMWDTHROSC
Description	Watchdog Timer Control		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								CLR				EN	RESERVED	INT	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:4	CLR	Clear timer When 0xA followed by 0x5 is written to these bits, the timer is loaded with 0x0000. Note that 0x5 must be written within one watchdog clock period Twdt after 0xA was written for the clearing to take effect (ensured). If 0x5 is written between Twdt and 2Twdt after 0xA was written, the clearing may take effect, but there is no guarantee. If 0x5 is written > 2Twdt after 0xA was written, the timer will not be cleared. If a value other than 0x5 is written after 0xA has been written, the clear sequence is aborted. If 0xA is written, this starts a new clear sequence. Writing to these bits when EN = 0 has no effect.	RW	0x0
3	EN	Enable timer When 1 is written to this bit the timer is enabled and starts incrementing. The interval setting specified by INT[1:0] is used. Writing 0 to this bit have no effect.	RW	0
2	RESERVED	This bit field is reserved.	RW	0
1:0	INT	Timer interval select These bits select the timer interval as follows: 00: Twdt x 32768 01: Twdt x 8192 10: Twdt x 512 11: Twdt x 64 Writing these bits when EN = 1 has no effect.	RW	0x0

ADC

The analog-to-digital converter (ADC) supports 14-bit analog-to-digital conversion with up to 12 effective number of bits (ENOBs). The ADC includes an analog multiplexer with up to eight individually configurable channels and a reference voltage generator. Conversion results can be written to memory through direct memory access (DMA). Several modes of operation are available.

Topic **Page**

15.1 ADC Introduction

The main features of the ADC are as follows:

- Selectable decimation rates which also set the effective resolution (7 to 12 bits)
- Eight individual input channels, single-ended or differential
- Reference voltage selectable as internal, external single-ended, external differential, or AVDD5
- Interrupt request generation
- DMA triggers at end of conversions
- Temperature sensor input
- Battery measurement capability

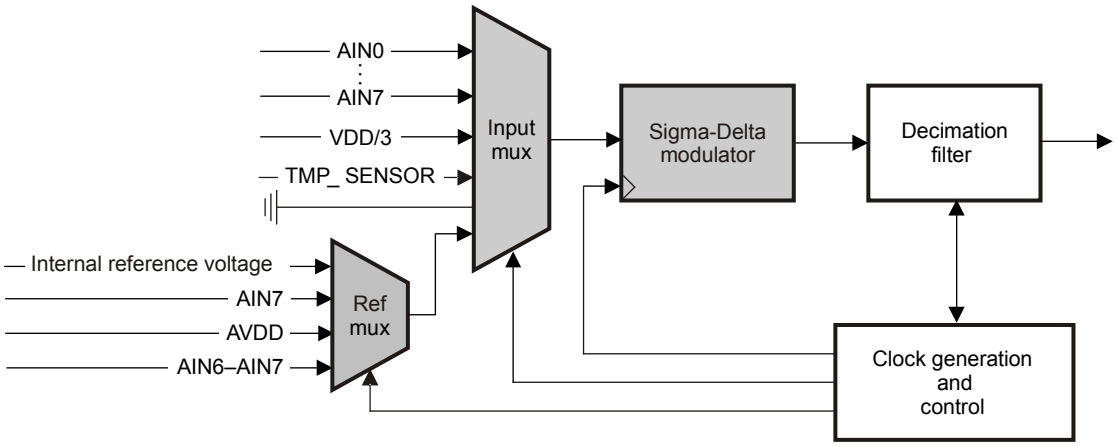


Figure 15-1. ADC Block Diagram

15.2 ADC Operation

This section describes the general setup and operation of the ADC and describes the use of the ADC control and status registers accessed by the central processing unit (CPU).

15.2.1 ADC Inputs

The signals on the PA pins can be used as ADC inputs. In the following discussion, these port pins are referred to as the AIN0–AIN7 pins. The input pins AIN0–AIN7 are connected to the ADC.

Configure the pins PA0–PA7 for use as ADC inputs, as analog inputs with no pullup or pulldown enabled (for details see Chapter 9).

It is possible to configure the inputs as single-ended or differential inputs. When differential inputs are selected, the differential inputs consist of the input pairs AIN0–AIN1, AIN2–AIN3, AIN4–AIN5, and AIN6–AIN7. Note that no negative supply can be applied to these pins, nor a supply higher than VDD (unregulated power). The difference between the pins of each pair is converted in differential mode.

In addition to input pins AIN0–AIN7, the output of an on-chip temperature sensor can be selected as an input to the ADC for temperature measurements. To do so, the **ADCTM** bit in the **CCTEST_TR0** register must be set as described in the register descriptions in , *SOC_ADC Registers*, and the **ATEST_CTRL** bit in the **RFCORE_XREG_ATEST** must be set as described in the register descriptions in , *RFCORE_XREG Registers*.

NOTE: PD[5:4] are not usable when temperature sensor is selected.

It is also possible to select a voltage corresponding to AVDD5/3 as an ADC input. This input allows the implementation of, for example, a battery monitor in applications where this feature is required. The reference in this case must not depend on the battery voltage; for instance, the AVDD5 voltage is not a valid reference.

The single-ended inputs AIN0—AIN7 are represented by channel numbers 0 to 7. Channel numbers 8 through 11 represent the differential inputs consisting of AIN0 and AIN1, AIN2 and AIN3, AIN4 and AIN5, and AIN6 and AIN7. Channel numbers 12 through 15 represent GND (12), temperature sensor (14), and AVDD5/3 (15), while channel 13 is reserved. These values are used in the **SCH** bit of the **SOC_ADC_ADCCON2** register and the **SCH** bit of the **SOC_ADC_ADCCON3** register.

The ADC input is a switched capacitance stage that draws current during the conversion. As an example, the equivalent input impedance of a typical device was found to be 176 kΩ when used with an input voltage of 3 V, a 512× decimation rate, and the internal reference.

15.2.2 ADC Conversion Sequences

The ADC can perform a sequence of conversions and move the results to memory (through DMA) without any interaction from the CPU.

The **SOC_ADC_ADCCON2.SCH** register bits are used to define an ADC conversion sequence from the ADC inputs. If **SOC_ADC_ADCCON2.SCH** is set to a value less than 8, the conversion sequence contains a conversion from each channel from 0 up to and including the channel number programmed in **SOC_ADC_ADCCON2.SCH**. When **SOC_ADC_ADCCON2.SCH** is set to a value between 8 and 12, the sequence consists of differential inputs, starting at channel 8 and ending at the programmed channel. When **SOC_ADC_ADCCON2.SCH** is set to a value greater than or equal to 12, the sequence consists of the selected channel only.

The ADC also considers the I/O configuration in the I/O Control module, so when a channel is included in the conversion sequence, but is not activated in the **IOC_PAx_SEL** registers, the channel is skipped. For the difference channels, to perform the conversion, active inputs are required (both inputs must be active). For extra conversions, the **IOC_PAx_SEL** registers are not checked.

15.2.3 Single ADC Conversion

In addition to this sequence of conversions, the ADC can be programmed to perform a single conversion from any channel. Such a conversion is triggered by writing to the **SOC_ADC_ADCCON3** register. The conversion starts immediately unless a conversion sequence is already in progress, in which case the single conversion is performed as soon as that sequence finishes.

15.2.4 ADC Operating Modes

This section describes the operating modes and initialization of conversions.

The ADC has three control registers: **SOC_ADC_ADCCON1**, **SOC_ADC_ADCCON2**, and **SOC_ADC_ADCCON3**. These registers are used to configure the ADC and to report status.

The **EOC** bit in the **SOC_ADC_ADCCON1** register is a status bit that is set high when a conversion ends and is cleared when ADCH is read.

The **ST** bit in the **SOC_ADC_ADCCON1** register is used to start a sequence of conversions. A sequence starts when the **ST** bit is set high, the **STSEL** bit in the **ADCCON1** register is 11, and no conversion is currently running. When the sequence completes, this bit is automatically cleared.

The **STSEL** bits in the **ADCCON1** register select the event that starts a new sequence of conversions. The options which can be selected are end of previous sequence, GP Timer 0 compare event from timer A or B, or **ST** bit in the **SOC_ADC_ADCCON1** register is 1.

The **SOC_ADC_ADCCON2** register controls how the sequence of conversions is performed.

The **SREF** bit in the **SOC_ADC_ADCCON2** register is used to select the reference voltage. The reference voltage should change only when no conversion is running.

The **SREF** bits in the **SOC_ADC_ADCCON2** register select the decimation rate, thereby also setting the resolution and time required to complete a conversion, and hence the sample rate. The decimation rate should change only when no conversion is running.

The last channel of a sequence is selected with the **SCH** bits in the **SOC_ADC_ADCCON2** register as described previously (see Section 15.2.2, *ADC Conversion Sequences*.

The **SOC_ADC_ADCCON3** register controls the channel number, reference voltage, and decimation rate for a single conversion. The single conversion occurs immediately after the **SOC_ADC_ADCCON3** register is written to, or if a conversion sequence is ongoing, immediately after the sequence endes. The coding of the register bits is identical to that of the **SOC_ADC_ADCCON2** register.

15.2.5 ADC Conversion Results

The digital conversion result is represented in 2s-complement form. Expect a positive result for single-ended configurations. This is because the result is the difference between the input signal and ground, which is always positively signed ($V_{CONV} = V_{INP} – V_{INN}$, where $V_{INN} = 0$ V). The maximum value is reached when the input signal is equal to VREF, the selected voltage reference. For differential configurations, the difference between two pins is converted, and this difference can be negatively signed. For example, with a decimation rate of 512 using only the 12 MSBs of the digital conversion result register, the maximum value of 2047 is reached when the analog input (V_{CONV}) is equal to V_{REF}, and minimum value of –2048 is reached when the analog input is equal to $–V_{REF}$.

The digital conversion result is available in the **SOC_ADC_ADCH** and **SOC_ADC_ADCL** registers when the **EOC** bit in the **SOC_ADC_ADCCON1** register is set to 1. The conversion result always resides in the MSB section of the combined **SOC_ADC_ADCH** and **SOC_ADC_ADCL** registers.

When the **SCH** bits in the **SOC_ADC_ADCCON2** register are read, they indicate the channel on which conversion is ongoing. The results in **SOC_ADC_ADCL** and **SOC_ADC_ADCH** normally apply to the previous conversion. If the conversion sequence has ended, the **SCH** bit in the **SOC_ADC_ADCCON2** register has a value of one more than the last channel number, but if the channel number last written to the **SCH** bit in the **SOC_ADC_ADCCON2** register was 12 or more, the same value is read back.

15.2.6 ADC Reference Voltage

The positive reference voltage for analog-to-digital conversions is selectable as either an internally generated voltage, the AVDD5 pin, an external voltage applied to the AIN7 input pin, or a differential voltage applied to the AIN6 and AIN7 inputs.

The accuracy of the conversion results depend on the stability and noise properties of the reference voltage. Offset from the wanted voltage introduces a gain error in the ADC proportional to the ratio of the wanted voltage and the actual voltage. To ensure the specified signal-to-noise ratio (SNR) is achieved, the noise on the reference must not exceed the quantization noise of the ADC.

15.2.7 ADC Conversion Timing

Use the ADC only be with the 32-MHz XOSC; do not implement system clock division. The actual ADC sampling frequency of 4 MHz is generated by fixed internal division. The time required to perform a conversion depends on the selected decimation rate. In general, the conversion time is given by:

T_{CONV} = (decimation rate + 16) × 0.25 µs.

15.2.8 ADC Interrupts

The ADC generates an interrupt when a single conversion triggered by writing to the **SOC_ADC_ADCCON3** register completes. No interrupt is generated when a conversion from the sequence completes.

15.2.9 ADC DMA Triggers

The ADC generates a DMA trigger every time a conversion from the sequence has completed. When a single conversion completes, no DMA trigger is generated. There is one DMA trigger for each of the eight ADC channels, DMA channels 14-17 (index 0) and 24-27 (index 0). In addition there is one DMA trigger for all eight ADC channels combined, DMA channel 1 (index 4). Please see Table 10-1 for DMA channel assignments. The DMA trigger is active when a new sample is ready from the conversion for the channel. Since the ADC has only one interrupt signal to the CPU (#14), DMA transfers on all of the ADC channels will end with an interrupt to the CPU on the same interrupt line. The processor will jump to an interrupt handler that will need to determine which of the assigned channels actually completed the transfer by interrogating the DMA status registers. See Chapter 10 for details on DMA.

15.3 Analog-to-Digital Converter Registers

15.3.1 SOC_ADC Registers

15.3.1.1 SOC_ADC Registers Mapping Summary

This section provides information on the SOC_ADC module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 15-1. SOC_ADC Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
SOC_ADC_ADCCON1	RW	32	0x0000 0033	0x00	0x400D 7000
SOC_ADC_ADCCON2	RW	32	0x0000 0010	0x04	0x400D 7004
SOC_ADC_ADCCON3	RW	32	0x0000 0000	0x08	0x400D 7008
SOC_ADC_ADCL	RO	32	0x0000 0000	0x0C	0x400D 700C
SOC_ADC_ADCH	RO	32	0x0000 0000	0x10	0x400D 7010

15.3.1.2 SOC_ADC Register Descriptions

SOC_ADC_ADCCON1

Address offset	0x00		
Physical Address	0x400D 7000	**Instance**	SOC_ADC
Description	This register controls the ADC.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								EOC	ST	STSEL		RCTRL		RESERVED	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	EOC	End of conversion. Cleared when ADCH has been read. If a new conversion is completed before the previous data has been read, the EOC bit remains high. 0: Conversion not complete 1: Conversion completed	RW R/HO	0
6	ST	Start conversion Read as 1 until conversion completes 0: No conversion in progress. 1: Start a conversion sequence if ADCCON1.STSEL = 11 and no sequence is running.	RW R/W1/HO	0
5:4	STSEL	Start select Selects the event that starts a new conversion sequence 00: Not implemented 01: Full speed. Do not wait for triggers 10: Timer 1 channel 0 compare event 11: ADCCON1.ST = 1	RW	0x3

Bits	Field Name	Description	Type	Reset
3:2	RCTRL	Controls the 16-bit random-number generator (see User Guide Chapter 16) When 01 is written, the setting automatically returns to 00 when the operation completes. 00: Normal operation (13x unrolling) 01: Clock the LFSR once (13x unrolling) 10: Reserved 11: Stopped. The random-number generator is turned off.	RW	0x0
1:0	RESERVED	This bit field is reserved.	RW	0x3

SOC_ADC_ADCCON2

Address offset	0x04		
Physical Address	0x400D 7004	**Instance**	SOC_ADC
Description	This register controls the ADC.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								SREF		SDIV		SCH			

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:6	SREF	Selects reference voltage used for the sequence of conversions 00: Internal reference 01: External reference on AIN7 pin 10: AVDD5 pin 11: External reference on AIN6-AIN7 differential input	RW	0x0
5:4	SDIV	Sets the decimation rate for channels included in the sequence of conversions. The decimation rate also determines the resolution and time required to complete a conversion. 00: 64 decimation rate (7 bits ENOB setting) 01: 128 decimation rate (9 bits ENOB setting) 10: 256 decimation rate (10 bits ENOB setting) 11: 512 decimation rate (12 bits ENOB setting)	RW	0x1
3:0	SCH	Sequence channel select Selects the end of the sequence A sequence can either be from AIN0 to AIN7 (SCH <= 7) or from differential input AIN0-AIN1 to AIN6-AIN7 (8 <= SCH <= 11). For other settings, only one conversions is performed. When read, these bits indicate the channel number on which a conversion is ongoing: 0000: AIN0 0001: AIN1 0010: AIN2 0011: AIN3 0100: AIN4 0101: AIN5 0110: AIN6 0111: AIN7 1000: AIN0-AIN1 1001: AIN2-AIN3 1010: AIN4-AIN5 1011: AIN6-AIN7 1100: GND 1101: Reserved 1110: Temperature sensor 1111: VDD/3	RW	0x0

SOC_ADC_ADCCON3

Address offset	0x08		
Physical Address	0x400D 7008	**Instance**	SOC_ADC
Description	This register controls the ADC.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								EREF		EDIV		ECH			

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:6	EREF	Selects reference voltage used for the extra conversion 00: Internal reference 01: External reference on AIN7 pin 10: AVDD5 pin 11: External reference on AIN6-AIN7 differential input	RW	0x0
5:4	EDIV	Sets the decimation rate used for the extra conversion The decimation rate also determines the resolution and the time required to complete the conversion. 00: 64 decimation rate (7 bits ENOB) 01: 128 decimation rate (9 bits ENOB) 10: 256 decimation rate (10 bits ENOB) 11: 512 decimation rate (12 bits ENOB)	RW	0x0
3:0	ECH	Single channel select. Selects the channel number of the single conversion that is triggered by writing to ADCCON3. 0000: AIN0 0001: AIN1 0010: AIN2 0011: AIN3 0100: AIN4 0101: AIN5 0110: AIN6 0111: AIN7 1000: AIN0-AIN1 1001: AIN2-AIN3 1010: AIN4-AIN5 1011: AIN6-AIN7 1100: GND 1101: Reserved 1110: Temperature sensor 1111: VDD/3	RW	0x0

SOC_ADC_ADCL

Address offset	0x0C		
Physical Address	0x400D 700C	**Instance**	SOC_ADC
Description	This register contains the least-significant part of ADC conversion result.		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:2	ADC	Least-significant part of ADC conversion result	RO	0x00
1:0	RESERVED	This bit field is reserved.	RO	0x0

SOC_ADC_ADCH

Address offset	0x10		
Physical Address	0x400D 7010	**Instance**	SOC_ADC
Description	This register contains the most-significant part of ADC conversion result.		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	ADC	Most-significant part of ADC conversion result	RO	0x00

Random Number Generator

This chapter provides more information about the random number generator (RNG) and its usage.

Topic **Page**

16.1 Introduction

The RNG has the following features:

- Generates pseudo-random bytes which can be read by the central-processing unit (CPU) or used directly by the command strobe processor
- Calculates CRC16 of bytes that are written to the **SOC_ADC_RNDH** register
- Seeded by value written to the **SOC_ADC_RNDL** register

The random-number generator is a 16-bit linear-feedback shift register (LFSR) with polynomial $X^{16} + X^{15} + X^2 + 1$ (that is, CRC16). It uses different levels of unrolling depending on the operation it performs. The basic version (no unrolling) is shown in Figure 16-1.

The random-number generator is turned off when the **RCTRL** bits in the **SOC_ADC_ADCCON1** register are set to 11.

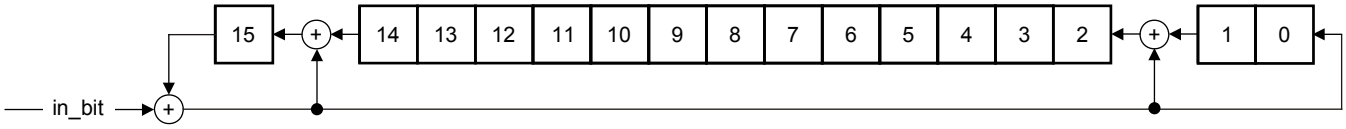


Figure 16-1. Basic Structure of the RNG

16.2 Random-Number-Generator Operation

The operation of the RNG is controlled by the **RCTRL** bits in the **SOC_ADC_ADCCON1** register. The current value of the 16-bit shift register in the LFSR can be read from the **SOC_ADC_RNDH** and **SOC_ADC_RNDL** registers.

16.2.1 Pseudo-random Sequence Generation

The default operation (the **RCTRL** bits in the **SOC_ADC_ADCCON1** register are set to 00) is to clock the LFSR once (13× unrolling; where clocking with 13× unrolling means performing an operation equivalent to doing 13 shifts with feedback) each time the command strobe processor (Section 23.14) reads the random value. This leads to the availability of a fresh pseudo-random byte from the LSB end of the LFSR.

Another way to update the LFSR is to set the **RCTRL** bits in the **SOC_ADC_ADCCON1** register to 01. This clocks the LFSR once (13× unrolling), and the **RCTRL** bits in the **SOC_ADC_ADCCON1** register automatically clear when the operation completes.

16.2.2 Seeding

Writing to the RNDL register twice can seed the LFSR. Each time the **SOC_ADC_RNDL** register is written, the 8 LSBs of the LFSR are copied to the 8 MSBs and the 8 LSBs are replaced with the new data byte that was written to the **SOC_ADC_RNDL** register.

For the CC2538, when a random value is required, writing the **SOC_ADC_RNDL** register with random bits from the IF_ADC in the RF receive path seeds the LFSR. To use this seeding method, first power on the radio. The radio must be placed in the infinite RX state to avoid possible sync detect in the RX state. The random bits from the IF_ADC are read from the LSB position of the RF register **RFCORE_XREG_RFRND**. These bits should be concatenated over time to form the bytes needed for the RNG seed. For a description of the randomness of these numbers, see Section 23.12. This cannot be done while the radio is in use for normal tasks.

NOTE: A seed value of 0x0000 or 0x8003 always leads to an unchanged value in the LFSR after clocking, as no values are pushed in through the in_bit (see Figure 16-1); hence, do not use either of these seed values for random-number generation.

16.2.3 CRC16

The LFSR can also calculate the CRC value of a sequence of bytes. Writing to the **SOC_ADC_RNDH** register triggers a CRC calculation. The new byte is processed from the MSB end and an 8× unrolling is used, so that a new byte can be written to **SOC_ADC_RNDH** every clock cycle.

NOTE: To properly seed the LFSR, write to the **SOC_ADC_RNDL** register before the CRC calculations start. Usually, the seed value for CRC calculations is 0x0000 or 0xFFFF.

16.3 Random Number Generator Registers

16.3.1 SOC_ADC Registers

16.3.1.1 SOC_ADC Registers Mapping Summary

This section provides information on the SOC_ADC module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 16-1. SOC_ADC Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
SOC_ADC_ADCCON1	RW	32	0x0000 0033	0x00	0x400D 7000
SOC_ADC_RNDL	RW	32	0x0000 00FF	0x14	0x400D 7014
SOC_ADC_RNDH	RW	32	0x0000 00FF	0x18	0x400D 7018

16.3.1.2 SOC_ADC Register Descriptions

SOC_ADC_ADCCON1

Address offset	0x00		
Physical Address	0x400D 7000	**Instance**	SOC_ADC
Description	This register controls the ADC.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								EOC	ST	STSEL		RCTRL		RESERVED	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	EOC	End of conversion. Cleared when ADCH has been read. If a new conversion is completed before the previous data has been read, the EOC bit remains high. 0: Conversion not complete 1: Conversion completed	RW R/HO	0
6	ST	Start conversion Read as 1 until conversion completes 0: No conversion in progress. 1: Start a conversion sequence if ADCCON1.STSEL = 11 and no sequence is running.	RW R/W1/HO	0

Bits	Field Name	Description	Type	Reset
5:4	STSEL	Start select Selects the event that starts a new conversion sequence 00: Not implemented 01: Full speed. Do not wait for triggers 10: Timer 1 channel 0 compare event 11: ADCCON1.ST = 1	RW	0x3
3:2	RCTRL	Controls the 16-bit random-number generator (see User Guide Chapter 16) When 01 is written, the setting automatically returns to 00 when the operation completes. 00: Normal operation (13x unrolling) 01: Clock the LFSR once (13x unrolling) 10: Reserved 11: Stopped. The random-number generator is turned off.	RW	0x0
1:0	RESERVED	This bit field is reserved.	RW	0x3

SOC_ADC_RNDL

Address offset	0x14		
Physical Address	0x400D 7014	**Instance**	SOC_ADC
Description	This registers contains random-number-generator data; low byte.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	RNDL	Random value/seed or CRC result, low byte When used for random-number generation, writing to this register twice seeds the random-number generator. Writing to this register copies the 8 LSBs of the LFSR to the 8 MSBs and replaces the 8 LSBs with the data value written. The value returned when reading from this register is the 8 LSBs of the LFSR. When used for random-number generation, reading this register returns the 8 LSBs of the random number. When used for CRC calculations, reading this register returns the 8 LSBs of the CRC result.	RW	0xFF

SOC_ADC_RNDH

Address offset	0x18		
Physical Address	0x400D 7018	**Instance**	SOC_ADC
Description	This register contains random-number-generator data; high byte.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	RNDH	Random value or CRC result/input data, high byte When written, a CRC16 calculation is triggered, and the data value written is processed starting with the MSB. The value returned when reading from this register is the 8 MSBs of the LFSR. When used for random-number generation, reading this register returns the 8 MSBs of the random number. When used for CRC calculations, reading this register returns the 8 MSBs of the CRC result.	RW	0xFF

Analog Comparator

The analog comparator in the CC2538 device has the following features:

- Low-power operation
- Wake-up source

Topic **Page**

17.1 Introduction

The analog comparator is connected to the I/O pins as follows:

- The positive input pin is connected to PA5.
- The negative input pin is connected to PA4.
- The output can be read from the **OUTPUT** bits in the **SOC_ADC_CMPCTL** register.

The comparator pins must be configured as analog pins by setting the corresponding bits, see Chapter 9. The **EN** bit in the **SOC_ADC_CMPCTL** register is used to enable and disable the comparator. The output from the comparator is connected internally to the edge detector that controls PA5. This makes it possible to associate an I/O interrupt with a rising/falling edge on the comparator output. When enabled, the comparator remains active in all power modes. Thus, for example, it is possible to wake up from power mode 2 or 3 on a rising or falling edge on the comparator output. The comparator output overrides the PA5 pin data input when the comparator is turned on.

To use the comparator as an interrupt source, bit 5 of Port A should be configured to accept interrupts as well as the edge type using appropriate fields in the GPIO_PI_IEN and GPIO_P_EDGE_CTRL registers respectively. Also, any pending interrupts on the pin should be cleared and global CPU interrupts enabled to get the interrupt.

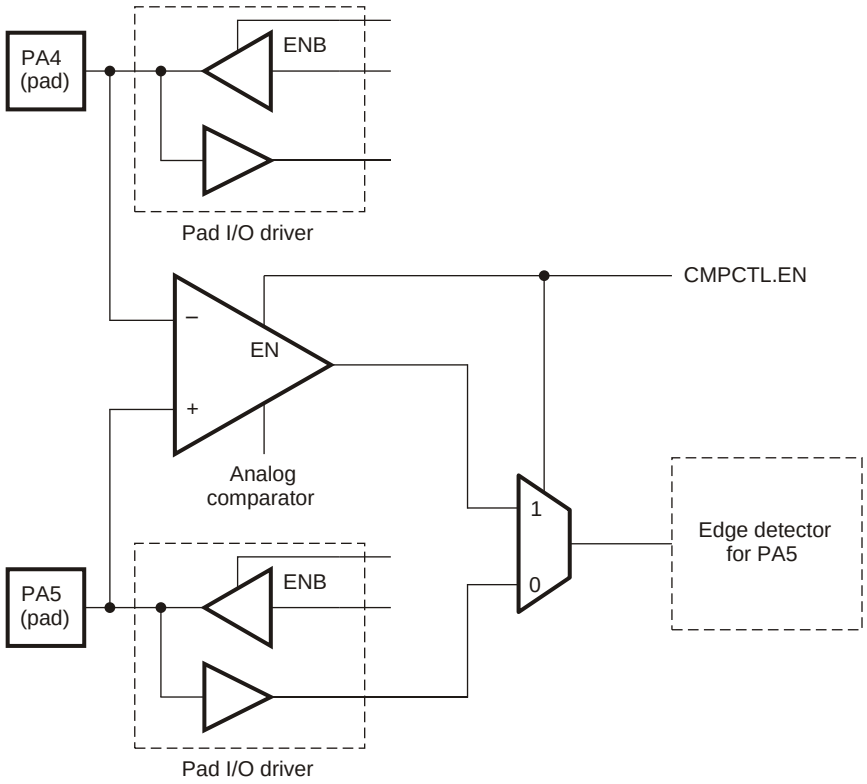


Figure 17-1. Analog Comparator

17.2 Analog Comparator Registers

17.2.1 SOC_ADC Registers

17.2.1.1 SOC_ADC Registers Mapping Summary

This section provides information on the SOC_ADC module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 17-1. SOC_ADC Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
SOC_ADC_CMPCTL	RW	32	0x0000 0001	0x24	0x400D 7024

17.2.1.2 SOC_ADC Register Descriptions

SOC_ADC_CMPCTL

Address offset	0x24		
Physical Address	0x400D 7024	**Instance**	SOC_ADC
Description	Analog comparator control and status register.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:2	RESERVED	This bit field is reserved.	RO	0x00
1	EN	Comparator enable	RW	0
0	OUTPUT	Comparator output	RO	1

Universal Asynchronous Receivers and Transmitters

This chapter describes the universal asynchronous receivers and transmitters (UARTs).

Topic **Page**

18.1 Universal Asynchronous Receivers and Transmitters

The CC2538 controller includes two UARTs with the following features:

- Programmable baud-rate generator allowing speeds up to 2 Mbps for regular speed (divide by 16) and 4 Mbps for high speed (divide by 8)
- Separate 16 × 8 bits FIFOs to transmit (TX) and receive (RX) first in first out buffers (FIFOs) to reduce CPU interrupt service loading
- Programmable FIFO length, including 1-byte deep operation providing conventional double-buffered interface
- FIFO trigger levels of 2/16,4/16,8/16,12/16, and 14/16
- Standard asynchronous communication bits for start, stop, and parity
- Line-break generation and detection
- Fully programmable serial interface characteristics:
  - 5, 6, 7, or 8 data bits
  - Even, odd, stick, or no-parity bit generation/detection
  - 1 or 2 stop-bit generation
- Full modem handshake support (on UART1)
- Standard FIFO-level and end-of-transmission interrupts
- Efficient transfers using micro direct memory access controller (µDMA):
  - Seperate channels for transmit and receive
  - Receive single request asserted when data is in the FIFO; burst request asserted at programmed FIFO level
  - Transmit single request asserted when there is space in the FIFO; burst request asserted at programmed FIFO level
- LIN protocol support
- UART0 does not support RTS or CTS signaling
- UART1 does support RTS or CTS signaling
- IrDA support available.

18.2 Block Diagram

Figure 18-1 shows the UART module block diagram.

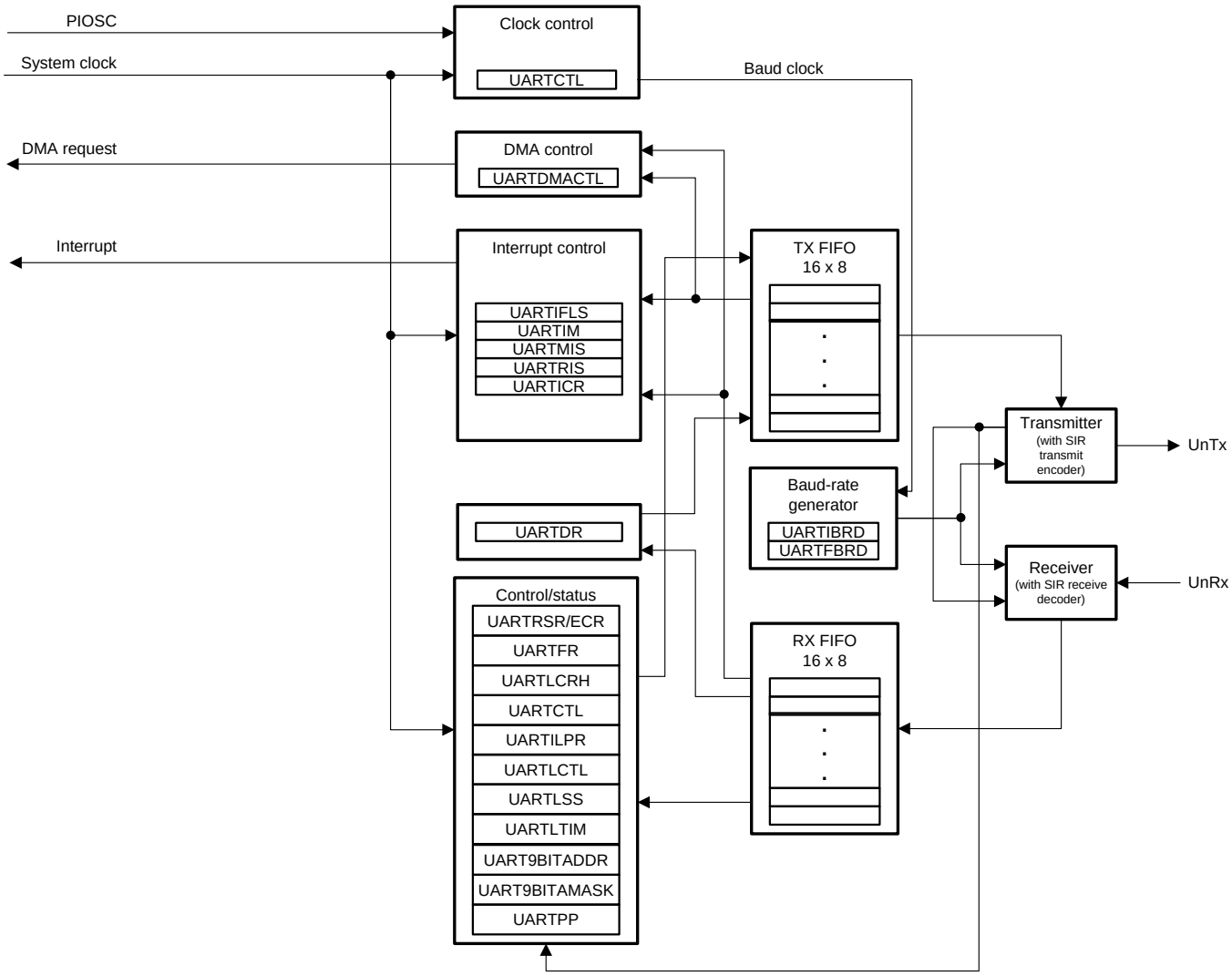


Figure 18-1. UART Module Block Diagram

18.3 Signal Description

Table 18-1 lists the external signals of the UART module and describes the function of each. The UART signals are set in the GPIO module through the **Pxx_SEL** registers. For more information on configuring GPIOs, see Chapter 9.

Table 18-1. Signals for UART (64LQFP)

Pin Name	Pin Number	Pin Type[1]	Buffer Type[2]	Description
U0Rx	Assigned through GPIO configuration	I	TTL	UART module 0 receive
U0Tx		O	TTL	UART module 0 transmit
U1Rx		I	TTL	UART module 1 receive
U1Tx		O	TTL	UART module 1 transmit

(1) I = Input; O = Output; I/O = Bidirectional

(2) TTL indicates the pin has TTL-compatible voltage levels.

18.4 Functional Description

Each CC2538 UART performs the functions of parallel-to-serial and serial-to-parallel conversions. The CC2538 UART is similar in functionality to a 16C550 UART, but is not register compatible.

The UART is configured for transmit and/or receive through the **TXE** and **RXE** bits of the **UART Control (UART_CTL)** register (see UART_CTL). Transmit and receive are both enabled out of reset. Before any control registers are programmed, the UART must be disabled by clearing the **UARTEN** bit in the **UART_CTL** register. If the UART is disabled during a TX or RX operation, the current transaction is completed before the UART stops.

18.4.1 Transmit and Receive Logic

The transmit logic performs parallel-to-serial conversion on the data read from the TX FIFO. The control logic outputs the serial bit stream beginning with a start-bit and followed by the data bits (LSB first), parity bit, and the stop-bits according to the programmed configuration in the control registers. See Figure 18-2 for details.

The receive logic performs serial-to-parallel conversion on the received bit stream after a valid start pulse is detected. Overrun, parity, frame error checking, and line-break detection are also performed, and 4 status bits accompany the data that is written to the RX FIFO.

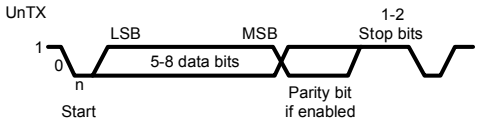


Figure 18-2. UART Character Frame

18.4.2 Baud-Rate Generation

The baud-rate divisor is a 22-bit number consisting of a 16-bit integer and a 6-bit fractional part. The number formed by these two values is used by the baud-rate generator to determine the bit period. Having a fractional baud-rate divider allows the UART to generate all the standard baud rates.

The 16-bit integer is loaded through the **UART Integer Baud-Rate Divisor (UART_IBRD)** register (see UART_IBRD) and the 6-bit fractional part is loaded with the **UART Fractional Baud-Rate Divisor (UART_FBRD)** register (see UART_FBRD). The baud-rate divisor (BRD) has the following relationship to the system clock (where BRDI is the integer part of the BRD and BRDF is the fractional part, separated by a decimal place.)

BRD = BRDI + BRDF = UARTSysClk / (ClkDiv × Baud Rate)

where **UARTSysClk** is the system clock connected to the UART, and **ClkDiv** is either 16 (if the **HSE** bit in the **UART_CTL** register is clear) or 8 (if the **HSE** bit is set).

The 6-bit fractional number (that is to be loaded into the **DIVFRAC** bit field in the **UART_FBRD** register) can be calculated by taking the fractional part of the baud-rate divisor, multiplying it by 64, and adding 0.5 to account for rounding errors:

UARTFBRD[DIVFRAC] = integer(BRDF × 64 + 0.5)

The UART generates an internal baud-rate reference clock at 8x or 16x the baud rate (referred to as **Baud8** and **Baud16**, depending on the setting of the **HSE** bit (bit 5 in the **UART_CTL** register). This reference clock is divided by 8 or 16 to generate the transmit clock, and is used for error detection during receive operations.

Along with the **UART Line Control, High Byte (UART_LCRH)** register (see LCRH), the **UART_IBRD** and **UART_FBRD** registers form an internal 30-bit register. This internal register is updated only when a write operation to **UART_LCRH** is performed, so a write to the **UART_LCRH** register must follow any changes to the baud-rate divisor for the changes to take effect.

To update the baud-rate registers, there are four possible sequences:

- **UART_IBRD** write, **UART_FBRD** write, and **UART_LCRH** write
- **UART_FBRD** write, **UART_IBRD** write, and **UART_LCRH** write

- **UART_IBRD** write and **UART_LCRH** write
- **UART_FBRD** write and **UART_LCRH** write

18.4.3 Data Transmission

Data received or transmitted is stored in two 16-byte FIFOs, though the RX FIFO has an extra 4 bits per character for status information. For transmission, data is written into the TX FIFO. If the UART is enabled, it causes a data frame to start transmitting with the parameters indicated in the **UART_LCRH** register. Data continues to transmit until no data is left in the TX FIFO. The **BUSY** bit in the **UART Flag (UART_FR)** register (see UART_FR) is asserted as soon as data is written to the TX FIFO (that is, if the FIFO is not empty) and remains asserted while data is transmitting. The **BUSY** bit is negated only when the TX FIFO is empty, and the last character has transmitted from the shift register, including the stop-bits. The UART can indicate that it is busy even though the UART may no longer be enabled.

When the receiver is idle (the **UnRx** signal is continuously 1), and the data input goes low (a start-bit was received), the receive counter begins running and data is sampled on the eighth cycle of **Baud16** or fourth cycle of **Baud8** depending on the setting of the **HSE** bit (bit 5) in **UART_CTL** (described in Section 18.4.1, *Transmit and Receive Logic*).

The start-bit is valid and recognized if the **UnRx** signal is still low on the eighth cycle of **Baud16** (**HSE** clear) or the fourth cycle of Baud 8 (**HSE** set), otherwise it is ignored. After a valid start bit is detected, successive data bits are sampled on every sixteenth cycle of **Baud16** or eighth cycle of **Baud8** (that is, one bit period later) according to the programmed length of the data characters and value of the **HSE** bit in **UART_CTL**. The parity bit is then checked if parity mode is enabled. Data length and parity are defined in the **UART_LCRH** register.

Lastly, a valid stop-bit is confirmed if the **UnRx** signal is high, otherwise a framing error has occurred. When a full word is received, the data is stored in the receive FIFO with any error bits associated with that word.

18.4.4 Modem Handshake Support

This section describes how to configure and use the modem flow control signals for UART1 when connected as a data terminal equipment (DTE) or as a data communications equipment (DCE). In general, a modem is a DCE and a computing device that connects to a modem is the DTE.

18.4.4.1 Signaling

The status signals provided by UART1 differ based on whether the UART is used as a DTE or DCE. When used as a DTE, the modem flow control signals are defined as:

- **U1CTS** is Clear To Send
- **U1RTS** is Request To Send

When used as a DCE, the modem flow control signals are defined as:

- **U1CTS** is Request To Send
- **U1RTS** is Clear To Send

18.4.4.2 Flow Control

UART1 optionally uses hardware flow control. UART0 does not support hardware flow control. The following section describes the UART1 hardware flow control and different methods.

18.4.4.2.1 Hardware Flow Control (RTS and CTS)

Hardware flow control between two devices is accomplished by connecting the **U1RTS** output to the Clear-To-Send input on the receiving device, and connecting the Request-To-Send output on the receiving device to the **U1CTS** input.

The **U1CTS** input controls the transmitter. The transmitter can transmit data only when the **U1CTS** input is asserted. The **U1RTS** output signal indicates the state of the receive FIFO. **U1CTS** remains asserted until the preprogrammed watermark level is reached, indicating that the RX FIFO has no space to store additional characters.

The **UART_CTL** register bits 15 (**CTSEN**) and 14 (**RTSEN**) specify the flow control mode as shown in Table 18-2.

Table 18-2. Flow Control Mode

CTSEN	RTSEN	Description
1	1	U1RTS and U1CTS flow control enabled
1	0	Only U1CTS flow control enabled
0	1	Only U1RTS flow control enabled
0	0	Both U1RTS and U1CTS flow control disabled

18.4.5 LIN Support

The UART module offers hardware support for the LIN protocol as either a master or a slave. The LIN mode is enabled by setting the **LIN** bit in the **UART_CTL** register. A LIN message is identified by the use of a sync break at the beginning of the message. The sync break is a transmission of a series of 0s. The sync break is followed by the sync data field (0x55). Figure 18-3 shows the structure of a LIN message.

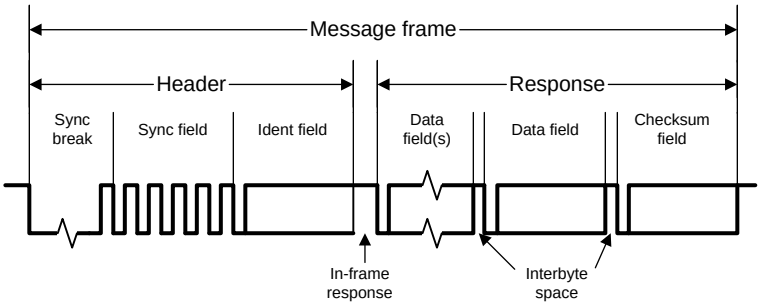


Figure 18-3. LIN Message

The UART should be configured as followed to operate in LIN mode:

1. Configure the UART for 1 start-bit, 8 data bits, no parity, and 1 stop-bit. Enable the TX FIFO.
2. Set the **LIN** bit in the **UART_CTL** register.

When preparing to send a LIN message, the TX FIFO should contain the sync data (0x55) at FIFO location 0 and the identifier data at location 1, followed by the data to be transmitted, and with the checksum in the final FIFO entry.

18.4.5.1 LIN Master

By setting the **MASTER** bit in the **UART_LCTL** register, the UART is enabled as the LIN master. The length of the sync break is programmable using the **BLEN** field in the **UART_LCTL** register and can be 13 to 16 bits (baud clock cycles).

18.4.5.2 LIN Slave

The LIN UART slave is required to adjust its baud rate to that of the LIN master. In slave mode, the LIN UART recognizes the sync break, which must be at least 13 bits in long. A timer is provided to capture timing data on the first and fifth falling edges of the sync field so that the baud rate can be adjusted to match the master.

After detecting a sync break, the UART waits for the synchronization field. The first falling edge generates an interrupt using the **LME1RIS** bit in the **UART_RIS** register, and the timer value is captured and stored in the **UARTLSS** register (T1). On the fifth falling edge, a second interrupt is generated using the **LME5RIS** bit in the **UART_RIS** register, and the timer value is captured again (T2). The actual baud rate can be calculated using (T2 – T1) / 8, and the local baud rate should be adjusted as needed. Figure 18-4 shows the synchronization field.

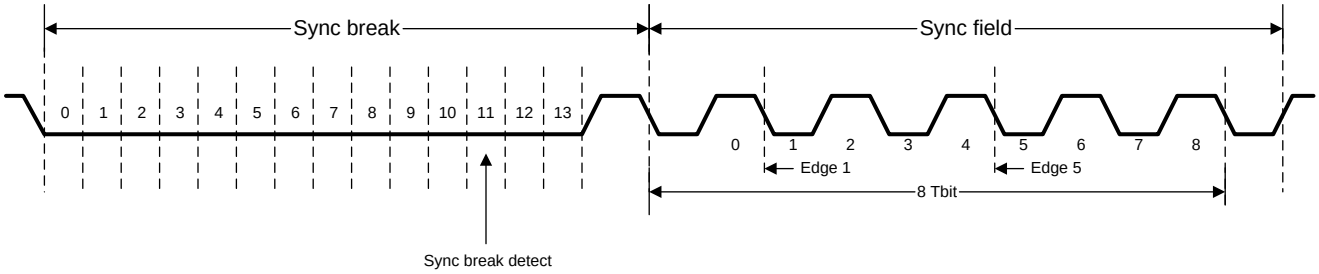


Figure 18-4. LIN Synchronization Field

18.4.6 9-Bit UART Mode

The UART provides a 9-bit mode that is selectable by using the **NINEBITEN** bit of the **UART_CTL** register. This feature is useful in a multidrop configuration of the UART where a single master connected to multiple slaves can communicate with a particular slave through its address or address range along with a qualifier for an address byte. All the slaves check for the address qualifier in the place of the parity bit. If the address qualifier is set, the slaves compare the byte received with the preprogrammed address. If the address matches then it receives or sends further data. If the address does not match, it drops the address byte and any subsequent data bytes. If the UART is in 9-bit mode then the receiver operates with no parity mode. The address can be predefined to match with the received byte and it can be configured through software at **UART_NINBITADDR** register. The matching could extend to an address range using the address mask **UART_NINEBITAMASK** that is ANDed with **UART_NINEBITADDR** to form the range. By default, the **UART_NINEBITAMASK** is 0xFF, meaning that only the specified address is matched. If a match is not found, the rest of the data bytes with the ninth bit cleared are dropped. If a match is found, then an interrupt is generated to the nested vector interrupt controller (NVIC) for further action. The subsequent data bytes with the cleared ninth bit are stored in the FIFO. Software can mask this interrupt in case µDMA and/or FIFO operations are enabled for this instance and processor intervention is not required. All the send transactions with 9-bit mode are data bytes and the ninth bit is cleared. Software can override the ninth bit to be set (to indicate address) by overriding the parity settings to sticky parity with odd parity enabled for particular byte. To match the transmission time with correct parity settings, the address byte can be transmitted as a single than a burst transfer. The TX FIFO does not hold the address or data bit, hence software should take care to enable address bit appropriately.

18.4.7 FIFO Operation

The UART has two 16-entry FIFOs; one for transmit and one for receive. Both FIFOs are accessed through the **UART Data (UART_DR)** register (see UART_DR). Read operations of the **UART_DR** register return a 12-bit value consisting of 8 data bits and 4 error flags while write operations place 8-bit data in the TX FIFO.

Out of reset, both FIFOs are disabled and act as 1-byte-deep holding registers. The FIFOs are enabled by setting the **FEN** bit in the **UART_LCRH** register (UART_LCRH).

FIFO status can be monitored through the **UART Flag (UART_FR)** register (see UART_FR) and the **UART Receive Status (UART_RSR)** register. Hardware monitors empty, full, and overrun conditions. The **UART_FR** register contains empty and full flags (**TXFE**, **TXFF**, **RXFE**, and **RXFF** bits), and the **UART_RSR** register shows overrun status through the **OE** bit. If the FIFOs are disabled, the empty and full flags are set according to the status of the 1-byte-deep holding registers.

The trigger points at which the FIFOs generate interrupts is controlled through the **UART Interrupt FIFO Level Select (UART_IFLS)** register (see UART_IFLS). Both FIFOs can be individually configured to trigger interrupts at different levels. Available configurations include ⅛, ¼, ½, ¾, and ⅞. For example, if the ¼ option is selected for the receive FIFO, the UART generates a receive interrupt after 4 data bytes are received. Out of reset, both FIFOs are configured to trigger an interrupt at the ½ mark.

18.4.8 Interrupts

The UART can generate interrupts when the following conditions are observed:

- Overrun Error
- Break Error
- Parity Error
- Framing Error
- Receive Time-out
- Transmit (when condition defined in the **TXIFLSEL** bit in the **UART_IFLS** register is met, or if the **EOT** bit in **UART_CTL** is set, when the last bit of all transmitted data leaves the serializer)
- Receive (when condition defined in the **RXIFLSEL** bit in the **UART_IFLS** register is met)

All of the interrupt events are ORed together before being sent to the interrupt controller (INTC), so the UART can only generate a single interrupt request to the controller at any given time. Software can service multiple interrupt events in a single interrupt service routine (ISR) by reading the **UART Masked Interrupt Status (UART_MIS)** register (see UART_MIS).

The interrupt events that can trigger a controller-level interrupt are defined in the **UART Interrupt Mask (UART_IM)** register (see UART_IM) by setting the corresponding **IM** bits. If interrupts are not used, the raw interrupt status is always visible through the **UART Raw Interrupt Status (UART_RIS)** register (see UART_RIS).

Interrupts are always cleared (for the **UARTMIS** and **UARTRIS** registers) by setting the corresponding bit in the **UART Interrupt Clear (UART_ICR)** register to 1 (see UART_ICR).

The receive time-out interrupt is asserted when the RX FIFO is not empty, and no further data is received over a 32-bit period. The receive time-out interrupt is cleared either when the FIFO becomes empty through reading all the data (or by reading the holding register), or when the corresponding bit in the **UART_ICR** register is set to 1.

18.4.9 Loopback Operation

The UART can be placed into an internal loopback mode for diagnostic or debug work by setting the **LBE** bit in the **UART_CTL** register (see UART_CTL). In loopback mode, data transmitted on the **UnTx** output is received on the **UnRx** input. Set the **LBE** bit before the UART is enabled.

18.5 Initialization and Configuration

To enable and initialize the UART, the following steps are necessary:

1. Enable the UART module using the **SYS_CTRL_RCGCUART** register (see SYS_CTRL_RCGCUART).
2. Set the GPIO pin configuration through the **Pxx_SEL** IOC_PA0_SEL registers for the desired output pins using the appropriate signal select value.
3. To enable IO pads to drive outputs, the corresponding **IPxx_OVER** bits in **IOC_Pxx_OVER** register has to be configured to 0x8 (OE - Output Enable) (see Section 9.2.2.5).
4. Connect the appropriate input signals to the UART module through the following registers:
   - **IOC_UARTRXD_UART0** IOC_UARTRXD_UART0
   - **IOC_UARTRXD_UART1** IOC_UARTRXD_UART1
   - **IOC_UARTCTS_UART1** IOC_UARTCTS_UART1
5. For more information on pin connections, see Section 9.1.1, *I/O Muxing*, of Chapter 9, *General Purpose Input/Outputs (GPIO)*.

To use the UART, the peripheral clock must be enabled by setting the appropriate bit in the **SYS_CTRL_RCGCUART** register (SYS_CTRL_RCGCUART). In addition, the clock to the appropriate GPIO module must be enabled through the **SYS_CTRL_RCGCUART** register (see SYS_CTRL_RCGCUART register in the system control module (SCM).

This section discusses the steps that are required to use a UART module. For this example, the UART clock is assumed to be 20 MHz, and the desired UART configuration is:

- 115,200 baud rate
- Data length of 8 bits
- One stop-bit
- No parity
- FIFOs disabled
- No interrupts

The first thing to consider when programming the UART is the baud-rate divisor (BRD), because the **UART_IBRD** and **UART_FBRD** registers must be written before the **UART_LCRH** register. Using the equation described in Section 18.4.2, the BRD can be calculated:

BRD = 20,000,000 / (16 × 115,200) = 10.8507

which means that the **DIVINT** field of the **UART_IBRD** register (see UART_IBRD) should be set to 10 decimal or 0xA. The value to be loaded into the **UART_FBRD** register (see UART_FBRD) is calculated by the equation:

UARTFBRD[DIVFRAC] = integer(0.8507 × 64 + 0.5) = 54

With the BRD values available, the UART configuration is written to the module in the following order:

1. Disable the UART by clearing the **UARTEN** bit in the **UART_CTL** register.
2. Write the integer portion of the BRD to the **UART_IBRD** register.
3. Write the fractional portion of the BRD to the **UART_FBRD** register.
4. Write the desired serial parameters to the **UART_LCRH** register (in this case, a value of 0x0000 0060).
5. Enable the UART by setting the **UARTEN** bit in the **UART_CTL** register.

18.6 UART Registers

18.6.1 UART Registers

18.6.1.1 UART Registers Mapping Summary

This section provides information on the UART module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

18.6.1.1.1 UART Common Registers Mapping

This section provides information on the UART module instance within this product. Each of the registers within the Module Instance is described separately below.

Table 18-3. UART Common Registers Mapping Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset
UART_DR	RW	32	0x0000 0000	0x000
UART_RSR	RO	32	0x0000 0000	0x004
UART_ECR	WO	32	0x0000 0000	0x004
UART_FR	RO	32	0x0000 0090	0x018
UART_ILPR	RW	32	0x0000 0000	0x020
UART_IBRD	RW	32	0x0000 0000	0x024

Table 18-3. UART Common Registers Mapping Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset
UART_FBRD	RW	32	0x0000 0000	0x028
UART_LCRH	RW	32	0x0000 0000	0x02C
UART_CTL	RW	32	0x0000 0300	0x030
UART_IFLS	RW	32	0x0000 0012	0x034
UART_IM	RW	32	0x0000 0000	0x038
UART_RIS	RO	32	0x0000 000X	0x03C
UART_MIS	RO	32	0x0000 0000	0x040
UART_ICR	WO	32	0x0000 0000	0x044
UART_DMACTL	RW	32	0x0000 0000	0x048
UART_LCTL	RW	32	0x0000 0000	0x090
UART_LSS	RO	32	0x0000 0000	0x094
UART_LTIM	RO	32	0x0000 0000	0x098
UART_NINEBITADDR	RW	32	0x0000 0000	0x0A4
UART_NINEBITAMASK	RW	32	0x0000 00FF	0x0A8
UART_PP	RO	32	0x0000 0003	0xFC0
UART_CC	RW	32	0x0000 0000	0xFC8

18.6.1.1.2 UART Instances Register Mapping Summary

18.6.1.1.2.1 UART0 Register Summary

Table 18-4. UART0 Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
UART_DR	RW	32	0x0000 0000	0x000	0x4000 C000
UART_RSR	RO	32	0x0000 0000	0x004	0x4000 C004
UART_ECR	WO	32	0x0000 0000	0x004	0x4000 C004
UART_FR	RO	32	0x0000 0090	0x018	0x4000 C018
UART_ILPR	RW	32	0x0000 0000	0x020	0x4000 C020
UART_IBRD	RW	32	0x0000 0000	0x024	0x4000 C024
UART_FBRD	RW	32	0x0000 0000	0x028	0x4000 C028
UART_LCRH	RW	32	0x0000 0000	0x02C	0x4000 C02C
UART_CTL	RW	32	0x0000 0300	0x030	0x4000 C030
UART_IFLS	RW	32	0x0000 0012	0x034	0x4000 C034
UART_IM	RW	32	0x0000 0000	0x038	0x4000 C038
UART_RIS	RO	32	0x0000 000X	0x03C	0x4000 C03C
UART_MIS	RO	32	0x0000 0000	0x040	0x4000 C040
UART_ICR	WO	32	0x0000 0000	0x044	0x4000 C044
UART_DMACTL	RW	32	0x0000 0000	0x048	0x4000 C048
UART_LCTL	RW	32	0x0000 0000	0x090	0x4000 C090
UART_LSS	RO	32	0x0000 0000	0x094	0x4000 C094
UART_LTIM	RO	32	0x0000 0000	0x098	0x4000 C098
UART_NINEBITADDR	RW	32	0x0000 0000	0x0A4	0x4000 C0A4

Table 18-4. UART0 Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
UART_NINEBITAMASK	RW	32	0x0000 00FF	0x0A8	0x4000 C0A8
UART_PP	RO	32	0x0000 0003	0xFC0	0x4000 CFC0
UART_CC	RW	32	0x0000 0000	0xFC8	0x4000 CFC8

18.6.1.1.2.2 UART1 Register Summary

Table 18-5. UART1 Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
UART_DR	RW	32	0x0000 0000	0x000	0x4000 D000
UART_RSR	RO	32	0x0000 0000	0x004	0x4000 D004
UART_ECR	WO	32	0x0000 0000	0x004	0x4000 D004
UART_FR	RO	32	0x0000 0090	0x018	0x4000 D018
UART_ILPR	RW	32	0x0000 0000	0x020	0x4000 D020
UART_IBRD	RW	32	0x0000 0000	0x024	0x4000 D024
UART_FBRD	RW	32	0x0000 0000	0x028	0x4000 D028
UART_LCRH	RW	32	0x0000 0000	0x02C	0x4000 D02C
UART_CTL	RW	32	0x0000 0300	0x030	0x4000 D030
UART_IFLS	RW	32	0x0000 0012	0x034	0x4000 D034
UART_IM	RW	32	0x0000 0000	0x038	0x4000 D038
UART_RIS	RO	32	0x0000 000X	0x03C	0x4000 D03C
UART_MIS	RO	32	0x0000 0000	0x040	0x4000 D040
UART_ICR	WO	32	0x0000 0000	0x044	0x4000 D044
UART_DMACTL	RW	32	0x0000 0000	0x048	0x4000 D048
UART_LCTL	RW	32	0x0000 0000	0x090	0x4000 D090
UART_LSS	RO	32	0x0000 0000	0x094	0x4000 D094
UART_LTIM	RO	32	0x0000 0000	0x098	0x4000 D098
UART_NINEBITADDR	RW	32	0x0000 0000	0x0A4	0x4000 D0A4
UART_NINEBITAMASK	RW	32	0x0000 00FF	0x0A8	0x4000 D0A8
UART_PP	RO	32	0x0000 0003	0xFC0	0x4000 DFC0
UART_CC	RW	32	0x0000 0000	0xFC8	0x4000 DFC8

18.6.1.2 UART Common Register Descriptions

UART_DR

Address offset	0x000		
Physical Address	0x4000 D000 0x4000 C000	**Instance**	UART1 UART0
Description	UART data Important: This register is read-sensitive. See the register description for details. This register is the data register (the interface to the FIFOs). For transmitted data, if the FIFO is enabled, data written to this location is pushed onto the transmit FIFO. If the FIFO is disabled, data is stored in the transmitter holding register (the bottom word of the transmit FIFO). A write to this register initiates a transmission from the UART. For received data, if the FIFO is enabled, the data byte and the 4-bit status (break, frame, parity, and overrun) is pushed onto the 12-bit wide receive FIFO. If the FIFO is disabled, the data byte and status are stored in the receiving holding register (the bottom word of the receive FIFO). The received data can be retrieved by reading this register.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																				OE	BE	PE	FE	DATA							

Bits	Field Name	Description	Type	Reset
31:12	RESERVED	This bit field is reserved.	RO	0x0 0000
11	OE	UART overrun error 1: New data was received when the FIFO was full, resulting in data loss. 0: No data has been lost due to a FIFO overrun.	RO	0
10	BE	UART break error 1: A break condition has been detected, indicating that the receive data input was held low for longer than a full-word transmission time (defined as start, data, parity, and stop bits). 0: No break condition has occurred. In FIFO mode, this error is associated with the character at the top of the FIFO. When a break occurs, only the one 0 character is loaded into the FIFO. The next character is only enabled after the received data input goes to a 1 (marking state), and the next valid start bit is received.	RO	0
9	PE	UART parity error 1: The parity of the received data character does not match the parity defined by bits 2 and 7 of the UARTLCRH register 0: No parity error has occurred. In FIFO mode, this error is associated with the character at the top of the FIFO.	RO	0
8	FE	UART framing error 1: The received character does not have a valid stop bit (a valid stop bit is 1). 0: No framing error has occurred.	RO	0
7:0	DATA	Data transmitted or received Data that is to be transmitted via the UART is written to this field. When read, this field contains the data that was received by the UART.	RW	0x00

UART_RSR

Address offset	0x004		
Physical Address	0x4000 D004 0x4000 C004	**Instance**	UART1 UART0
Description	UART receive status and error clear The RSR/ECR register is the receive status register and error clear register. In addition to the DR register, receive status can also be read from the RSR register. If the status is read from this register, then the status information corresponds to the entry read from DR before reading RSR. The status information for overrun is set immediately when an overrun condition occurs. The RSR register cannot be written. Read-only status register		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												OE	BE	PE	FE

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3	OE	UART overrun error 1: New data was received when the FIFO was full, resulting in data loss. 0: No data has been lost due to a FIFO overrun. This bit is cleared by a write to UARTECR. The FIFO contents remain valid because no further data is written when the FIFO is full, only the contents of the shift register are overwritten. The CPU must read the data in order to empty the FIFO.	RO	0
2	BE	UART break error 1: A break condition has been detected, indicating that the receive data input was held low for longer than a full-word transmission time (defined as start, data, parity, and stop bits). 0: No break condition has occurred. This bit is cleared to 0 by a write to UARTECR. In FIFO mode, this error is associated with the character at the top of the FIFO. When a break occurs, only one 0 character is loaded into the FIFO. The next character is only enabled after the receive data input goes to a 1 (marking state) and the next valid start bit is received.	RO	0
1	PE	UART parity error 1: The parity of the received data character does not match the parity defined by bits 2 and 7 of the UARTLCRH register. 0: No parity error has occurred. This bit is cleared to 0 by a write to UARTECR.	RO	0
0	FE	UART framing error 1: The received character does not have a valid stop bit (a valid stop bit is 1). 0: No framing error has occurred. This bit is cleared to 0 by a write to UARTECR. In FIFO mode, this error is associated with the character at the top of the FIFO.	RO	0

UART_ECR

Address offset	0x004		
Physical Address	0x4000 D004 0x4000 C004	**Instance**	UART1 UART0
Description	UART receive status and error clear The RSR/ECR register is the receive status register/error clear register. A write of any value to the ECR register clears the framing, parity, break, and overrun errors. All the bits are cleared on reset. Write-only error clear register		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	WO	0x00 0000
7:0	DATA	Error clear A write to this register of any data clears the framing, parity, break, and overrun flags.	WO	0x00

UART_FR

Address offset	0x018		
Physical Address	0x4000 D018 0x4000 C018	**Instance**	UART1 UART0
Description	UART flag The FR register is the flag register. After reset, the TXFF, RXFF, and BUSY bits are 0, and TXFE and RXFE bits are 1. The CTS bit indicate the modem flow control. Note that the modem bits are only implemented on UART1 and are tied inactive on UART0. Due to this difference, the reset state of the UART0 FR register is 0x90, while UART1 FR register reset state 0x197 .		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								TXFE	RXFF	TXFF	RXFE	BUSY	RESERVED		CTS

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	TXFE	UART transmit FIFO empty The meaning of this bit depends on the state of the FEN bit in the UARTLCRH register. 1: If the FIFO is disabled (FEN is 0), the transmit holding register is empty. If the FIFO is enabled (FEN is 1), the transmit FIFO is empty. 0: The transmitter has data to transmit.	RO	1
6	RXFF	UART receive FIFO full The meaning of this bit depends on the state of the FEN bit in the UARTLCRH register. 1: If the FIFO is disabled (FEN is 0), the receive holding register is full. If the FIFO is enabled (FEN is 1), the receive FIFO is full. 0: The receiver can receive data.	RO	0
5	TXFF	UART transmit FIFO full The meaning of this bit depends on the state of the FEN bit in the UARTLCRH register. 1: If the FIFO is disabled (FEN is 0), the transmit holding register is full. If the FIFO is enabled (FEN is 1), the transmit FIFO is full. 0: The transmitter is not full.	RO	0
4	RXFE	UART receive FIFO empty The meaning of this bit depends on the state of the FEN bit in the UARTLCRH register. 1: If the FIFO is disabled (FEN is 0), the receive holding register is empty. If the FIFO is enabled (FEN is 1), the receive FIFO is empty. 0: The receiver is not empty.	RO	1
3	BUSY	UART busy 1: The UART is busy transmitting data. This bit remains set until the complete byte, including all stop bits, has been sent from the shift register. 0: The UART is not busy. This bit is set as soon as the transmit FIFO becomes non-empty (regardless of whether UART is enabled).	RO	0
2:1	RESERVED	This bit field is reserved.	RO	0x0
0	CTS	Clear to send (UART1 only, reserved for UART0). 1: The U1CTS signal is asserted. 0: The U1CTS signal is not asserted.	RO	0

UART_ILPR

Address offset	0x020		
Physical Address	0x4000 D020 0x4000 C020	**Instance**	UART1 UART0
Description	UART IrDA low-power register The ILPR register stores the 8-bit low-power counter divisor value used to derive the low-power SIR pulse width clock by dividing down the system clock (SysClk). All the bits are cleared when reset. The internal IrLPBaud16 clock is generated by dividing down SysClk according to the low-power divisor value written to ILPR. The duration of SIR pulses generated when low-power mode is enabled is three times the period of the IrLPBaud16 clock. The low-power divisor value is calculated as follows: ILPDVSR = SysClk / FIrLPBaud16 where FIrLPBaud16 is nominally 1.8432 MHz The divisor must be programmed such that FIrLPBaud16 is in the range 1.42 MHz to 2.12 MHz, resulting in a low-power pulse duration of 1.41-2.11 us (three times the period of IrLPBaud16). The minimum frequency of IrLPBaud16 ensures that pulses less than one period of IrLPBaud16 are rejected, but pulses greater than 1.4 us are accepted as valid pulses. Note: Zero is an illegal value. Programming a zero value results in no IrLPBaud16 pulses being generated.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	ILPDVSR	IrDA low-power divisor This field contains the 8-bit low-power divisor value.	RW	0x00

UART_IBRD

Address offset	0x024		
Physical Address	0x4000 D024 0x4000 C024	**Instance**	UART1 UART0
Description	UART integer baud-rate divisor The IBRD register is the integer part of the baud-rate divisor value. All the bits are cleared on reset. The minimum possible divide ratio is 1 (when IBRD = 0), in which case the FBRD register is ignored. When changing the IBRD register, the new value does not take effect until transmission or reception of the current character is complete. Any changes to the baud-rate divisor must be followed by a write to the LCRH register.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15:0	DIVINT	Integer baud-rate divisor	RW	0x0000

UART_FBRD

Address offset	0x028		
Physical Address	0x4000 D028 0x4000 C028	**Instance**	UART1 UART0
Description	UART fractional baud-rate divisor The FBRD register is the fractional part of the baud-rate divisor value. All the bits are cleared on reset. When changing the FBRD register, the new value does not take effect until transmission or reception of the current character is complete. Any changes to the baud-rate divisor must be followed by a write to the LCRH register.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:6	RESERVED	This bit field is reserved.	RO	0x000 0000
5:0	DIVFRAC	Fractional baud-rate divisor	RW	0x00

UART_LCRH

Address offset	0x02C		
Physical Address	0x4000 D02C 0x4000 C02C	**Instance**	UART1 UART0
Description	UART line control The LCRH register is the line control register. Serial parameters such as data length, parity, and stop bit selection are implemented in this register. When updating the baud-rate divisor (IBRD and/or IFRD), the LCRH register must also be written. The write strobe for the baud-rate divisor registers is tied to the LCRH register.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								SPS	WLEN			FEN	STP2	EPS	PEN	BRK

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	SPS	UART stick parity select When bits 1, 2, and 7 of UARTLCRH are set, the parity bit is transmitted and checked as a 0. When bits 1 and 7 are set and 2 is cleared, the parity bit is transmitted and checked as a 1. When this bit is cleared, stick parity is disabled.	RW	0
6:5	WLEN	UART word length The bits indicate the number of data bits transmitted or received in a frame as follows: 0x0: 5 bits (default) 0x1: 6 bits 0x2: 7 bits 0x3: 8 bits	RW	0x0
4	FEN	UART enable FIFOs 1: The transmit and receive FIFObuffers are enabled (FIFOmode). 0: The FIFOs are disabled (Character mode). The FIFOs become 1-byte-deep holding registers.	RW	0
3	STP2	UART two stop bits select 1: Two stop bits are transmitted at the end of a frame. The receive logic does not check for two stop bits being received. 0: One stop bit is transmitted at the end of a frame.	RW	0
2	EPS	UART even parity select 1: Even parity generation and checking is performed during transmission and reception, which checks for an even number of 1s in data and parity bits. 0: Odd parity is performed, which checks for an odd number of 1s. This bit has no effect when parity is disabled by the PEN bit.	RW	0
1	PEN	UART parity enable 1: Parity checking and generation is enabled. 0: Parity is disabled and no parity bit is added to the data frame.	RW	0
0	BRK	UART send break 1: A low level is continually output on the UnTx signal, after completing transmission of the current character. For the proper execution of the break command, software must set this bit for at least two frames (character periods). 0: Normal use	RW	0

UART_CTL

Address offset	0x030		
Physical Address	0x4000 D030 0x4000 C030	**Instance**	UART1 UART0

Description

UART control
The CTL register is the control register. All the bits are cleared on reset except for the transmit enable (TXE) and receive enable (RXE) bits, which are set.
To enable the UART module, the UARTEN bit must be set. If software requires a configuration change in the module, the UARTEN bit must be cleared before the configuration changes are written. If the UART is disabled during a transmit or receive operation, the current transaction is completed before the UART stopping.
Note: The UARTCTL register should not be changed while the UART is enabled or else the results are unpredictable. The following sequence is recommended for making changes to the UARTCTL register:
1. Disable the UART.
2. Wait for the end of transmission or reception of the current character.
3. Flush the transmit FIFO by clearing bit 4 (FEN) in the line control register (UARTLCRH).
4. Reprogram the control register.
5. Enable the UART.

Type RW

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																CTSEN	RTSEN	RESERVED				RXE	TXE	LBE	LIN	HSE	EOT	RESERVED	SIRLP	SIREN	UARTEN

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15	CTSEN	U1CTS Hardware Flow control enable 1: When U1CTS input is asserted, UART1 can transmit data. 0: U1CTS does not control UART1 data transmission. Note: Only used for UART1. This bit is reserved RO for UART0.	RW	0
14	RTSEN	U1RTS Hardware Flow control enable 1: U1RTS indicates the state of UART1 receive FIFO. U1RTS remains asserted until the preprogrammed watermark level is reached, indicating that the UART1 RXFIFO has no space to store additional characters. 0: U1RTS does not indicate state of UART1 RX FIFO. Note: Only used for UART1. This bit is reserved RO for UART0.	RW	0
13:10	RESERVED	This bit field is reserved.	RO	0x0
9	RXE	UART receive enable 1: The receive section of the UART is enabled. 0: The receive section of the UART is disabled. If the UART is disabled in the middle of a receive, it completes the current character before stopping. Note: To enable reception, the UARTEN bit must also be set.	RW	1
8	TXE	UART transmit enable 1: The transmit section of the UART is enabled. 0: The transmit section of the UART is disabled. If the UART is disabled in the middle of a transmission, it completes the current character before stopping. Note: To enable transmission, the UARTEN bit must also be set.	RW	1
7	LBE	UART loop back enable 1: The UnTx path is fed through the UnRx path. 0: Normal operation	RW	0
6	LIN	LIN mode enable 1: The UART operates in LIN mode. 0: Normal operation	RW	0
5	HSE	High-speed enable 0: The UART is clocked using the system clock divided by 16. 1: The UART is clocked using the system clock divided by 8. Note: System clock used is also dependent on the baud-rate divisor configuration (See Universal Asynchronous Receivers/Transmitters - Baud-Rate Generation).	RW	0

Bits	Field Name	Description	Type	Reset
4	EOT	End of transmission This bit determines the behavior of the TXRIS bit in the UARTRIS register. 1: The TXRIS bit is set only after all transmitted data, including stop bits, have cleared the serializer. 0: The TXRIS bit is set when the transmit FIFO condition specified in UARTIFLS is met.	RW	0
3	RESERVED	This bit field is reserved.	RW	0
2	SIRLP	UART SIR low-power mode This bit selects the IrDA encoding mode. 1: The UART operates in SIR Low-Power mode. Low-level bits are transmitted with a pulse width which is 3 times the period of the IrLPBaud16 input signal, regardless of the selected bit rate. 0: Low-level bits are transmitted as an active high pulse with a width of 3/16th of the bit period. Setting this bit uses less power, but might reduce transmission distances.	RW	0
1	SIREN	UART SIR enable 1: The IrDA SIR block is enabled, and the UART transmits and receives data using SIR protocol. 0: Normal operation.	RW	0
0	UARTEN	UART enable 1: The UART is enabled. 0: The UART is disabled. If the UART is disabled in the middle of transmission or reception, it completes the current character before stopping.	RW	0

UART_IFLS

Address offset 0x034

Physical Address 0x4000 D034 **Instance** UART1
0x4000 C034 UART0

Description UART interrupt FIFO level select
The IFLS register is the interrupt FIFO level select register. This register can be used to define the FIFO level at which the TXRIS and RXRIS bits in the RIS register are triggered.
The interrupts are generated based on a transition through a level rather than being based on the level. That is, the interrupts are generated when the fill level progresses through the trigger level. For example, if the receive trigger level is set to the half-way mark, the interrupt is triggered as the module is receiving the 9th character.
Out of reset, the TXIFLSEL and RXIFLSEL bits are configured so that the FIFOs trigger an interrupt at the half-way mark.

Type RW

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

Bits	Field Name	Description	Type	Reset
31:6	RESERVED	This bit field is reserved.	RO	0x000 0000
5:3	RXIFLSEL	UART receive interrupt FIFO level select The trigger points for the receive interrupt are as follows: 0x0: RX FIFO >= 1/8 full 0x1: RX FIFO >= 1/4 full 0x2: RX FIFO >= 1/2 full (default) 0x3: RX FIFO >= 3/4 full 0x4: RX FIFO >= 7/8 full 0x5-0x7: Reserved	RW	0x2

Bits	Field Name	Description	Type	Reset
2:0	TXIFLSEL	UART Transmit Interrupt FIFO Level Select The trigger points for the transmit interrupt are as follows: 0x0: TX FIFO <= 7/8 empty 0x1: TX FIFO <= 3/4 empty 0x2: TX FIFO <= 1/2 empty (default) 0x3: TX FIFO <= 1/4 empty 0x4: TX FIFO <= 1/8 empty 0x5-0x7: Reserved Note: If the EOT bit in UARTCTL is set, the transmit interrupt is generated once the FIFO is completely empty and all data including stop bits have left the transmit serializer. In this case, the setting of TXIFLSEL is ignored.	RW	0x2

UART_IM

Address offset 0x038

Physical Address 0x4000 D038 **Instance** UART1
0x4000 C038 UART0

Description UART interrupt mask
The IM register is the interrupt mask set/clear register.
On a read, this register gives the current value of the mask on the relevant interrupt. Setting a bit allows the corresponding raw interrupt signal to be routed to the interrupt controller. Clearing a bit prevents the raw interrupt signal from being sent to the interrupt controller.

Type RW

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																LME5IM	LME1IM	LMSBIM	NINEBITIM	RESERVED	OEIM	BEIM	PEIM	FEIM	RTIM	TXIM	RXIM	RESERVED			

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15	LME5IM	LIN mode edge 5 interrupt mask 1: An interrupt is sent to the interrupt controller when the LME5RIS bit in the UARTRIS register is set. 0: The LME5RIS interrupt is suppressed and not sent to the interrupt controller.	RW	0
14	LME1IM	LIN mode edge 1 interrupt mask 1: An interrupt is sent to the interrupt controller when the LME1RIS bit in the UARTRIS register is set. 0: The LME1RIS interrupt is suppressed and not sent to the interrupt controller.	RW	0
13	LMSBIM	LIN mode sync break interrupt mask 1: An interrupt is sent to the interrupt controller when the LMSBRIS bit in the UARTRIS register is set. 0: The LMSBRIS interrupt is suppressed and not sent to the interrupt controller.	RW	0
12	NINEBITIM	9-bit mode interrupt mask 1: An interrupt is sent to the interrupt controller when the 9BITRIS bit in the UARTRIS register is set. 0: The 9BITRIS interrupt is suppressed and not sent to the interrupt controller.	RW	0
11	RESERVED	This bit field is reserved.	RO	0
10	OEIM	UART overrun error interrupt mask 1: An interrupt is sent to the interrupt controller when the OERIS bit in the UARTRIS register is set. 0: The OERIS interrupt is suppressed and not sent to the interrupt controller.	RW	0

Bits	Field Name	Description	Type	Reset
9	BEIM	UART break error interrupt mask 1: An interrupt is sent to the interrupt controller when the BERIS bit in the UARTRIS register is set. 0: The BERIS interrupt is suppressed and not sent to the interrupt controller.	RW	0
8	PEIM	UART parity error interrupt mask 1: An interrupt is sent to the interrupt controller when the PERIS bit in the UARTRIS register is set. 0: The PERIS interrupt is suppressed and not sent to the interrupt controller.	RW	0
7	FEIM	UART framing error interrupt mask 1: An interrupt is sent to the interrupt controller when the FERIS bit in the UARTRIS register is set. 0: The FERIS interrupt is suppressed and not sent to the interrupt controller.	RW	0
6	RTIM	UART receive time-out interrupt mask 1: An interrupt is sent to the interrupt controller when the RTRIS bit in the UARTRIS register is set. 0: The RTRIS interrupt is suppressed and not sent to the interrupt controller.	RW	0
5	TXIM	UART transmit interrupt mask 1: An interrupt is sent to the interrupt controller when the TXRIS bit in the UARTRIS register is set. 0: The TXRIS interrupt is suppressed and not sent to the interrupt controller.	RW	0
4	RXIM	UART receive interrupt mask 1: An interrupt is sent to the interrupt controller when the RXRIS bit in the UARTRIS register is set. 0: The RXRIS interrupt is suppressed and not sent to the interrupt controller.	RW	0
3:0	RESERVED	This bit field is reserved.	RO	0x0

UART_RIS

Address offset 0x03C

Physical Address 0x4000 D03C **Instance** UART1
0x4000 C03C UART0

Description UART raw interrupt status
The RIS register is the raw interrupt status register. On a read, this register gives the current raw status value of the corresponding interrupt. A write has no effect. Note that the HW modem flow control bits are only implemented on UART1 and are tied inactive on UART0.

Type RO

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																LME5RIS	LME1RIS	LMSBRIS	NINEBITRIS	RESERVED	OERIS	BERIS	PERIS	FERIS	RTRIS	TXRIS	RXRIS	RESERVED			

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15	LME5RIS	LIN mode edge 5 raw interrupt status 1: The timer value at the 5th falling edge of the LIN sync field has been captured. 0: No interrupt This bit is cleared by writing 1 to the LME5IC bit in the UARTICR register.	RO	0

Bits	Field Name	Description	Type	Reset
14	LME1RIS	LIN mode edge 1 raw interrupt status 1: The timer value at the 1st falling edge of the LIN Sync Field has been captured. 0: No interrupt This bit is cleared by writing 1 to the LME1IC bit in the UARTICR register.	RO	0
13	LMSBRIS	LIN mode sync break raw interrupt status 1: A LIN sync break has been detected. 0: No interrupt This bit is cleared by writing 1 to the LMSBIC bit in the UARTICR register.	RO	0
12	NINEBITRIS	9-mit mode raw interrupt status 1: A receive address match has occurred. 0: No interrupt This bit is cleared by writing 1 to the 9BITIC bit in the UARTICR register.	RO	0
11	RESERVED	This bit field is reserved.	RO	0
10	OERIS	UART overrun error raw interrupt status 1: An overrun error has occurred. 0: No interrupt This bit is cleared by writing 1 to the OEIC bit in the UARTICR register.	RO	0
9	BERIS	UART break error raw interrupt status 1: A break error has occurred. 0: No interrupt This bit is cleared by writing 1 to the BEIC bit in the UARTICR register.	RO	0
8	PERIS	UART parity error raw interrupt status 1: A parity error has occurred. 0: No interrupt This bit is cleared by writing 1 to the PEIC bit in the UARTICR register.	RO	0
7	FERIS	UART framing error raw interrupt status 1: A framing error has occurred. 0: No interrupt This bit is cleared by writing 1 to the FEIC bit in the UARTICR register.	RO	0
6	RTRIS	UART receive time-out raw interrupt status 1: A receive time out has occurred. 0: No interrupt This bit is cleared by writing 1 to the RTIC bit in the UARTICR register.	RO	0
5	TXRIS	UART transmit raw interrupt status 1: If the EOT bit in the UARTCTL register is clear, the transmit FIFO level has passed through the condition defined in the UARTIFLS register. If the EOT bit is set, the last bit of all transmitted data and flags has left the serializer. 0: No interrupt This bit is cleared by writing 1 to the TXIC bit in the UARTICR register.	RO	0
4	RXRIS	UART receive raw interrupt status 1: The receive FIFO level has passed through the condition defined in the UARTIFLS register. 0: No interrupt This bit is cleared by writing 1 to the RXIC bit in the UARTICR register.	RO	0
3:0	RESERVED	This bit field is reserved.	RO	0xX

UART_MIS

Address offset	0x040		
Physical Address	0x4000 D040 0x4000 C040	**Instance**	UART1 UART0
Description	UART masked interrupt status The MIS register is the masked interrupt status register. On a read, this register gives the current masked status value of the corresponding interrupt. A write has no effect.		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																LME5MIS	LME1MIS	LMSBMIS	NINEBITMIS	RESERVED	OEMIS	BEMIS	PEMIS	FEMIS	RTMIS	TXMIS	RXMIS	RESERVED			

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15	LME5MIS	LIN mode edge 5 masked interrupt status 1: An unmasked interrupt was signaled due to the 5th falling edge of the LIN sync field. 0: An interrupt has not occurred or is masked. This bit is cleared by writing 1 to the LME5IC bit in the UARTICR register.	RO	0
14	LME1MIS	LIN mode edge 1 masked interrupt status 1: An unmasked interrupt was signaled due to the 1st falling edge of the LIN sync field. 0: An interrupt has not occurred or is masked. This bit is cleared by writing 1 to the LME1IC bit in the UARTICR register.	RO	0
13	LMSBMIS	LIN mode sync break masked interrupt status 1: An unmasked interrupt was signaled due to the receipt of a LIN sync break. 0: An interrupt has not occurred or is masked. This bit is cleared by writing 1 to the LMSBIC bit in the UARTICR register.	RO	0
12	NINEBITMIS	9-bit mode masked interrupt status 1: An unmasked interrupt was signaled due to a receive address match. 0: An interrupt has not occurred or is masked. This bit is cleared by writing 1 to the 9BITIC bit in the UARTICR register.	RO	0
11	RESERVED	This bit field is reserved.	RO	0
10	OEMIS	UART overrun error masked interrupt status 1: An unmasked interrupt was signaled due to an overrun error. 0: An interrupt has not occurred or is masked. This bit is cleared by writing 1 to the OEIC bit in the UARTICR register.	RO	0
9	BEMIS	UART break error masked interrupt status 1: An unmasked interrupt was signaled due to a break error. 0: An interrupt has not occurred or is masked. This bit is cleared by writing 1 to the BEIC bit in the UARTICR register.	RO	0
8	PEMIS	UART parity error masked interrupt status 1: An unmasked interrupt was signaled due to a parity error. 0: An interrupt has not occurred or is masked. This bit is cleared by writing 1 to the PEIC bit in the UARTICR register.	RO	0
7	FEMIS	UART framing error masked interrupt status 1: An unmasked interrupt was signaled due to a framing error. 0: An interrupt has not occurred or is masked. This bit is cleared by writing 1 to the FEIC bit in the UARTICR register.	RO	0

Bits	Field Name	Description	Type	Reset
6	RTMIS	UART receive time-out masked interrupt status 1: An unmasked interrupt was signaled due to a receive time out. 0: An interrupt has not occurred or is masked. This bit is cleared by writing 1 to the RTIC bit in the UARTICR register.	RO	0
5	TXMIS	UART transmit masked interrupt status 1: An unmasked interrupt was signaled due to passing through the specified transmit FIFO level (if the EOT bit is clear) or due to the transmission of the last data bit (if the EOT bit is set). 0: An interrupt has not occurred or is masked. This bit is cleared by writing 1 to the TXIC bit in the UARTICR register.	RO	0
4	RXMIS	UART receive masked interrupt status 1: An unmasked interrupt was signaled due to passing through the specified receive FIFO level. 0: An interrupt has not occurred or is masked. This bit is cleared by writing 1 to the RXIC bit in the UARTICR register.	RO	0
3:0	RESERVED	This bit field is reserved.	RO	0x0

UART_ICR

Address offset 0x044

Physical Address 0x4000 D044 **Instance** UART1
0x4000 C044 UART0

Description UART interrupt clear
The ICR register is the interrupt clear register. On a write of 1, the corresponding interrupt (both raw interrupt and masked interrupt, if enabled) is cleared. A write of 0 has no effect.

Type WO

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																LME5IC	LME1IC	LMSBIC	NINEBITIC	RESERVED	OEIC	BEIC	PEIC	FEIC	RTIC	TXIC	RXIC	RESERVED			

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	WO	0x0000
15	LME5IC	LIN mode edge 5 interrupt clear Writing 1 to this bit clears the LME5RIS bit in the UARTRIS register and the LME5MIS bit in the UARTMIS register.	WO	0
14	LME1IC	LIN mode edge 1 interrupt clear Writing 1 to this bit clears the LME1RIS bit in the UARTRIS register and the LME1MIS bit in the UARTMIS register.	WO	0
13	LMSBIC	LIN mode sync break interrupt clear Writing 1 to this bit clears the LMSBRIS bit in the UARTRIS register and the LMSBMIS bit in the UARTMIS register.	WO	0
12	NINEBITIC	9-bit mode interrupt clear Writing 1 to this bit clears the 9BITRIS bit in the UARTRIS register and the 9BITMIS bit in the UARTMIS register.	WO	0
11	RESERVED	This bit field is reserved.	WO	0
10	OEIC	Overrun error interrupt clear Writing 1 to this bit clears the OERIS bit in the UARTRIS register and the OEMIS bit in the UARTMIS register.	WO	0
9	BEIC	Break error interrupt clear Writing 1 to this bit clears the BERIS bit in the UARTRIS register and the BEMIS bit in the UARTMIS register.	WO	0
8	PEIC	Parity error interrupt clear Writing 1 to this bit clears the PERIS bit in the UARTRIS register and the PEMIS bit in the UARTMIS register.	WO	0

Bits	Field Name	Description	Type	Reset
7	FEIC	Framing error interrupt clear Writing 1 to this bit clears the FERIS bit in the UARTRIS register and the FEMIS bit in the UARTMIS register.	WO	0
6	RTIC	Receive time-out interrupt clear Writing 1 to this bit clears the RTRIS bit in the UARTRIS register and the RTMIS bit in the UARTMIS register.	WO	0
5	TXIC	Transmit interrupt clear Writing 1 to this bit clears the TXRIS bit in the UARTRIS register and the TXMIS bit in the UARTMIS register.	WO	0
4	RXIC	Receive interrupt clear Writing 1 to this bit clears the RXRIS bit in the UARTRIS register and the RXMIS bit in the UARTMIS register.	WO	0
3:0	RESERVED	This bit field is reserved.	WO	0x0

UART_DMACTL

Address offset 0x048

Physical Address 0x4000 D048 **Instance** UART1
0x4000 C048 UART0

Description UART DMA control
The DMACTL register is the DMA control register.

Type RW

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																													DMAERR	TXDMAE	RXDMAE

Bits	Field Name	Description	Type	Reset
31:3	RESERVED	This bit field is reserved.	RO	0x0000 0000
2	DMAERR	DMA on error 1: uDMA receive requests are automatically disabled when a receive error occurs. 0: uDMA receive requests are unaffected when a receive error occurs.	RW	0
1	TXDMAE	Transmit DMA enable 1: uDMA for the transmit FIFO is enabled. 0: uDMA for the transmit FIFO is disabled.	RW	0
0	RXDMAE	Receive DMA enable 1: uDMA for the receive FIFO is enabled. 0: uDMA for the receive FIFO is disabled.	RW	0

UART_LCTL

Address offset 0x090

Physical Address 0x4000 D090 **Instance** UART1
0x4000 C090 UART0

Description UART LIN control
The LCTL register is the configures the operation of the UART when in LIN mode.

Type RW

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

Bits	Field Name	Description	Type	Reset
31:6	RESERVED	This bit field is reserved.	RO	0x000 0000
5:4	BLEN	Sync break length 0x3: Sync break length is 16T bits 0x2: Sync break length is 15T bits 0x1: Sync break length is 14T bits 0x0: Sync break length is 13T bits (default)	RW	0x0
3:1	RESERVED	This bit field is reserved.	RO	0x0
0	MASTER	LIN master enable 1: The UART operates as a LIN master. 0: The UART operates as a LIN slave.	RW	0

UART_LSS

Address offset 0x094

Physical Address 0x4000 D094 **Instance** UART1
0x4000 C094 UART0

Description LIN snap shot
The LSS register captures the free-running timer value when either the sync edge 1 or the sync edge 5 is detected in LIN mode.

Type RO

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

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15:0	TSS	Timer snap shot This field contains the value of the free-running timer when either the sync edge 5 or the sync edge 1 was detected.	RO	0x0000

UART_LTIM

Address offset 0x098

Physical Address 0x4000 D098 **Instance** UART1
0x4000 C098 UART0

Description UART LIN timer
The LTIM register contains the current timer value for the free-running timer that is used to calculate the baud rate when in LIN slave mode. The value in this register is used along with the value in the UART LIN snap shot (LSS) register to adjust the baud rate to match that of the master.

Type RO

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

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15:0	TIMER	Timer value This field contains the value of the free-running timer.	RO	0x0000

UART_NINEBITADDR

Address offset	0x0A4		
Physical Address	0x4000 D0A4 0x4000 C0A4	**Instance**	UART1 UART0
Description	UART 9-bit self address The NINEBITADDR register is used to write the specific address that should be matched with the receiving byte when the 9-bit address mask (NINEBITAMASK) is set to 0xFF. This register is used in conjunction with NINEBITAMASK to form a match for address-byte received.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15	NINEBITEN	Enable 9-bit mode 1: 9-bit mode is enabled. 0: 9-bit mode is disabled.	RW	0
14:8	RESERVED	This bit field is reserved.	RO	0x00
7:0	ADDR	Self address for 9-bit mode This field contains the address that should be matched when UART9BITAMASK is 0xFF.	RW	0x00

UART_NINEBITAMASK

Address offset	0x0A8		
Physical Address	0x4000 D0A8 0x4000 C0A8	**Instance**	UART1 UART0
Description	UART 9-bit self address mask The NINEBITAMASK register is used to enable the address mask for 9-bit mode. The lower address bits are masked to create a range of address to be matched with the received address byte.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15:8	RANGE	Self address range for 9-bit mode Writing to the RANGE field does not have any effect; reading it reflects the ANDed output of the ADDR field in the UART9BITADDR register and the MASK field.	RO	0x00
7:0	MASK	Self Address Mask for 9-Bit Mode This field contains the address mask that creates a range of addresses that should be matched.	RW	0xFF

UART_PP

Address offset	0xFC0		
Physical Address	0x4000 DFC0 0x4000 CFC0	**Instance**	UART1 UART0
Description	UART peripheral properties The PP register provides information regarding the properties of the UART module.		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	RO	0x0000 0000
1	NB	9-bit support 1: The UART module provides support for the transmission of 9-bit data for RS-485 support. 0: The UART module does not provide support for the transmission of 9-bit data for RS-485 support.	RO	1
0	RESERVED	This bit field is reserved.	RO	1

UART_CC

Address offset	0xFC8		
Physical Address	0x4000 DFC8 0x4000 CFC8	**Instance**	UART1 UART0
Description	UART clock configuration The CC register controls the baud and system clocks sources for the UART module. For more information, see the section called "Baud-Rate Generation". Note: If the PIOSC is used for the UART baud clock, the system clock frequency must be at least 9 MHz in run mode.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:3	RESERVED	This bit field is reserved.	RO	0x0000 0000
2:0	CS	UART baud and system clock source The following bits determine the clock source that generates the baud and system clocks for the UART. bit0 (PIOSC): 1: The UART baud clock is determined by the IO DIV setting in the system controller. 0: The UART baud clock is determined by the SYS DIV setting in the system controller. bit1: Unused bit2: (DSEN) Only meaningful when the system is in deep sleep mode. This bit is a don't care when not in sleep mode. 1: The UART system clock is running on the same clock as the baud clock, as per PIOSC setting above. 0: The UART system clock is determined by the SYS DIV setting in the system controller.	RW	0x0

Synchronous Serial Interface

This chapter describes the synchronous serial interface (SSI).

Topic **Page**

19.1 Synchronous Serial Interface

The CC2538 microcontroller includes two SSI modules. Each SSI is a master or slave interface for synchronous serial communication with peripheral devices that have either Freescale SPI, MICROWIRE, or Texas Instruments SSIs. The CC2538 SSI modules have the following features:

- Programmable interface operation for Freescale SPI, MICROWIRE, or TI SSIs
- Configurable as a master or as a slave on the interface
- Programmable clock bit rate and prescaler
- Separate transmit (TX) and receive (RX) first in first out buffers (FIFOs), each 16 bits wide and 8 locations deep
- Programmable data frame size from 4 to 16 bits
- Internal loopback test mode for diagnostic and debug testing
- Standard FIFO-based interrupts and end-of-transmission interrupt
- Efficient transfers using micro direct memory access controller (µDMA):
  - Separate channels for transmit and receive
  - Receive single request asserted when data is in the FIFO; burst request asserted when FIFO contains four entries
  - Transmit single request asserted when there is space in the FIFO; burst request asserted when FIFO contains four entries

19.2 Block Diagram

Figure 19-1 shows the SSI block diagram.

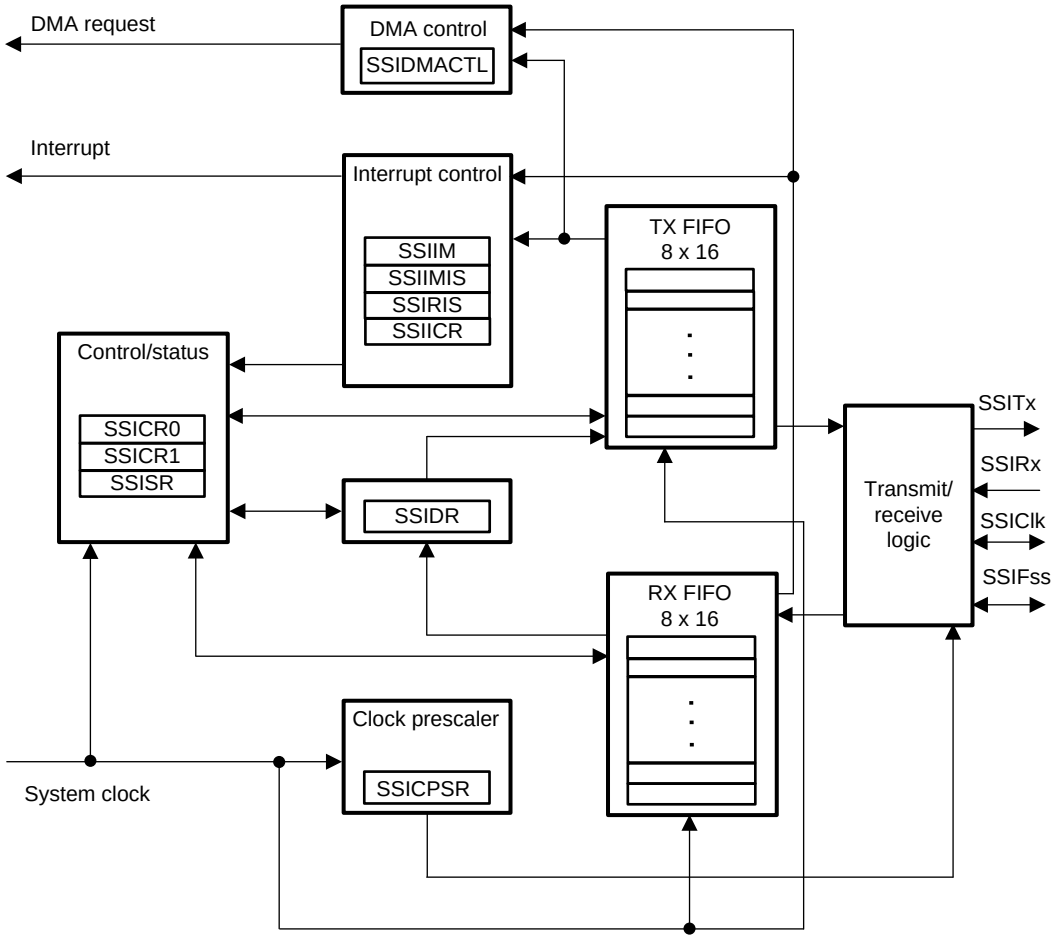


Figure 19-1. SSI Module Block Diagram

19.3 Signal Description

Table 19-1 lists the external signals of the SSI module and describes the function of each. The SSI signals are set in the GPIO module through the **Pxx_SEL** registers. For more information on configuring GPIOs, see Chapter 9.

Table 19-1. Signals for SSI (64LQFP)

Pin Name	Pin Number	Pin Type[1]	Buffer Type[2]	Description
SSI0Clk	Assigned through GPIO configuration	I/O	TTL	SSI module 0 clock
SSI0Fss		I/O	TTL	SSI module 0 frame
SSI0Rx		I	TTL	SSI module 0 receive
SSI0Tx		O	TTL	SSI module 0 transmit
SSI1Clk		I/O	TTL	SSI module 1 clock
SSI1Fss		I/O	TTL	SSI module 1 frame
SSI1Rx		I	TTL	SSI module 1 receive
SSI1Tx		O	TTL	SSI module 1 transmit

(1) I = Input; O = Output; I/O = Bidirectional

(2) TTL indicates the pin has TTL-compatible voltage levels.

19.4 Functional Description

The SSI performs serial-to-parallel conversion on data received from a peripheral device. The CPU accesses data, control, and status information. The transmit and receive paths are buffered with internal FIFO memories allowing idependent storage of up to eight 16-bit values in both transmit and receive modes. The SSI also supports the µDMA interface. The TX and RX FIFOs can be programmed as destination or source addresses in the µDMA module. µDMA operation is enabled by setting the appropriate bits in the **SSI_DMACTL** register.

19.4.1 Bit Rate Generation

The SSI includes a programmable bit rate clock divider and prescaler to generate the serial output clock. Bit rates are supported to 2 MHz and higher, although maximum bit rate is determined by peripheral devices.

The serial bit rate is derived by dividing down the input clock (SysClk). First, the clock is divided by an even prescale value **CPSDVSR** from 2 to 254, which is programmed in the **SSI Clock Prescale (SSI_CPSR)** register (see SSI_CPSR). The clock is further divided by a value from 1 to 256, which is 1 + **SCR**, where **SCR** is the value programmed in the **SSI Control 0 (SSI_CR0)** register (see SSI_CR0).

The frequency of the output clock **SSIClk** is defined by:

SSIClk = SysClk / (CPSDVSR × (1 + SCR))

NOTE: The PIOSC is used as the source for the **SSIClk** when the **CS** field in the **SSI Clock Configuration (SSI_CC)** register is configured to 0x1. For master mode, the system clock or the PIOSC must be at least two times faster than the **SSIClk**. For slave mode, the system clock or the PIOSC must be at least six times faster than the **SSIClk**.

19.4.2 FIFO Operation

19.4.2.1 Transmit FIFO

The common TX FIFO is a 16-bit-wide, 8-location-deep, first-in first-out memory buffer. The CPU writes data to the FIFO by writing the **SSI Data (SSI_DR)** register (see SSI_DR), and data is stored in the FIFO until it is read out by the transmission logic.

When configured as a master or a slave, parallel data is written into the TX FIFO before serial conversion and transmission to the attached slave or master, respectively, through the **SSITx** pin.

In slave mode, the SSI transmits data each time the master initiates a transaction. If the TX FIFO is empty and the master initiates, the slave transmits the eighth most-recent value in the transmit FIFO. If less than eight values are written to the TX FIFO since the SSI module clock was enabled using the **SSI** bit in the **SYS_CTRL_RCGCSSI** register, then 0 is transmitted. Take care to ensure that valid data is in the FIFO as needed. The SSI can be configured to generate an interrupt or a µDMA request when the FIFO is empty.

19.4.2.2 Receive FIFO

The common RX FIFO is a 16-bit-wide, 8-location-deep, first-in first-out memory buffer. Received data from the serial interface is stored in the buffer until read out by the CPU, which accesses the read FIFO by reading the **SSI_DR** register.

When configured as a master or slave, serial data received through the **SSIRx** pin is registered before parallel loading into the attached slave or master RX FIFO, respectively.

19.4.3 Interrupts

The SSI can generate interrupts when the following conditions are observed:

- TX FIFO service (when the TX FIFO is half full or less)
- RX FIFO service (when the RX FIFO is half full or more)
- RX FIFO time-out
- RX FIFO overrun
- End of transmission

All of the interrupt events are ORed together before being sent to the interrupt controller (INTC), so the SSI generates a single interrupt request to the controller regardless of the number of active interrupts. Each of the four individual maskable interrupts can be masked by clearing the appropriate bit in the **SSI Interrupt Mask (SSI_IM)** register (see SSI_IM). Setting the appropriate mask bit enables the interrupt.

The individual outputs, along with a combined interrupt output, allow use of either a global interrupt service routine or modular device drivers to handle interrupts. The transmit and receive dynamic dataflow interrupts are separated from the status interrupts so that data can be read or written in response to the FIFO trigger levels. The status of the individual interrupt sources can be read from the **SSI Raw Interrupt Status (SSI_RIS)** and **SSI Masked Interrupt Status (SSI_MIS)** registers (see SSI_RIS and SSI_MIS, respectively).

The RX FIFO has a time-out period that is 32 periods at the rate of **SSIClk** (whether or not **SSIClk** is currently active) and is started when the RX FIFO goes from empty to not empty. If the RX FIFO is emptied before 32 clocks pass, the time-out period is reset. As a result, the interrupt service routine (ISR) should clear the RX FIFO time-out interrupt just after reading out the RX FIFO by setting the **RTIC** bit in the **SSI Interrupt Clear (SSI_ICR)** register to 1.

NOTE: The interrupt should not be cleared so late that the ISR returns before the interrupt is actually cleared, or the ISR may be re-activated unnecessarily.

The end-of-transmission (EOT) interrupt indicates that the data has transmitted completely. This interrupt can indicate when it is safe to turn off the SSI module clock or enter sleep mode. In addition, because transmitted data and received data complete at exactly the same time, the interrupt can also indicate that read data is ready immediately, without waiting for the RX FIFO time-out period to complete.

19.4.4 Frame Formats

Each data frame is between 4 and 16 bits long, depending on the size of data programmed, and is transmitted starting with the most-significant bit (MSB). Three basic frame types can be selected:

- Texas Instruments synchronous serial
- Freescale SPI

- MICROWIRE

For all three formats, the serial clock (**SSIClk**) is held inactive while the SSI is idle, and **SSIClk** transitions at the programmed frequency only during active transmission or reception of data. The IDLE state of **SSIClk** provides a receive time-out indication that occurs when the RX FIFO still contains data after a time-out period.

For Freescale SPI and MICROWIRE frame formats, the serial frame (**SSIFss**) pin is active low, and is asserted (pulled down) during the entire transmission of the frame.

For Texas Instruments synchronous serial frame format, the **SSIFss** pin is pulsed for one serial clock period starting at its rising edge, before the transmission of each frame. For this frame format, both the SSI and the off-chip slave device drive their output data on the rising edge of **SSIClk** and latch data from the other device on the falling edge.

Unlike the full-duplex transmission of the other two frame formats, the MICROWIRE format uses a special master-slave messaging technique that operates at half-duplex. In this mode, when a frame begins an 8-bit control message is transmitted to the off-chip slave. During this transmit, no incoming data is received by the SSI. After the message is sent, the off-chip slave decodes it and, after waiting one serial clock after the last bit of the 8-bit control message is sent, responds with the requested data. The returned data can be 4 to 16 bits long, making the total frame length anywhere from 13 to 25 bits.

19.4.4.1 Texas Instruments Synchronous Serial Frame Format

Figure 19-2 shows the Texas Instruments synchronous serial frame format for a single transmitted frame.

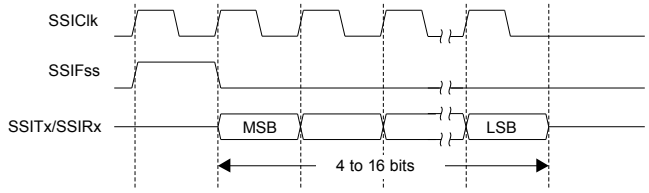


Figure 19-2. TI Synchronous Serial Frame Format (Single Transfer)

In this mode, **SSIClk** and **SSIFss** are forced low, and the transmit data line **SSITx** is tristated whenever the SSI is idle. Once the bottom entry of the TX FIFO contains data, **SSIFss** is pulsed high for one **SSIClk** period. The transmitted value is also transferred from the TX FIFO to the serial shift register of the transmit logic. On the next rising edge of **SSIClk**, the MSB of the 4- to 16-bit data frame is shifted out on the **SSITx** pin. Likewise, the MSB of the received data is shifted onto the **SSIRx** pin by the off-chip serial slave device.

Both the SSI and the off-chip serial slave device then clock each data bit into their serial shifter on each falling edge of **SSIClk**. The received data is transferred from the serial shifter to the RX FIFO on the first rising edge of **SSIClk** after the least significant bit (LSB) is latched.

Figure 19-3 shows the Texas Instruments synchronous serial frame format when back-to-back frames are transmitted.

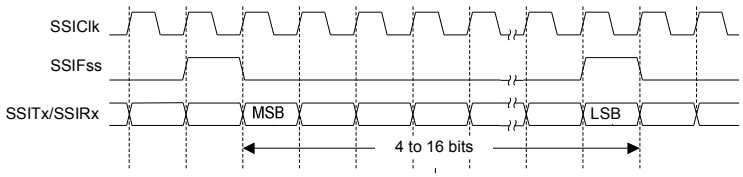


Figure 19-3. TI Synchronous Serial Frame Format (Continuous Transfer)

19.4.4.2 Freescale SPI Frame Format

The Freescale SPI interface is a 4-wire interface where the **SSIFss** signal behaves as a slave select. The main feature of the Freescale SPI format is that the inactive state and phase of the **SSIClk** signal are programmable through the **SPO** and **SPH** bits in the **SCR0** control register.

19.4.4.2.1 SPO Clock Polarity Bit

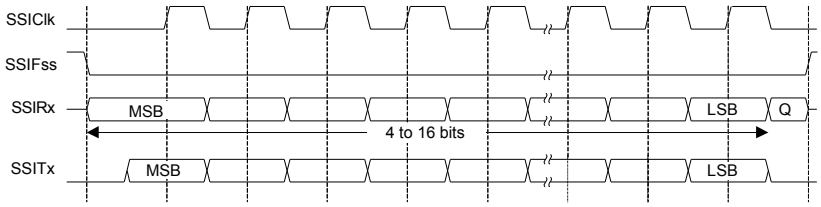
When the **SPO** clock polarity control bit is clear, it produces a steady state low value on the **SSIClk** pin. If the **SPO** bit is set, a steady state high value is placed on the **SSIClk** pin when data is not being transferred.

19.4.4.2.2 SPH Phase Control Bit

The **SPH** phase control bit selects the clock edge that captures data and allows it to change state. The state of this bit has the most impact on the first bit transmitted by either allowing or not allowing a clock transition before the first data capture edge. When the **SPH** phase control bit is clear, data is captured on the first clock edge transition. If the **SPH** bit is set, data is captured on the second clock edge transition.

19.4.4.3 Freescale SPI Frame Format With SPO = 0 and SPH = 0

Figure 19-4 and Figure 19-5 show single and continuous transmission signal sequences for Freescale SPI format with **SPO** = 0 and **SPH** = 0, respectively.



Note: Q is undefined.

Figure 19-4. Freescale SPI Format (Single Transfer) With SPO = 0 and SPH = 0

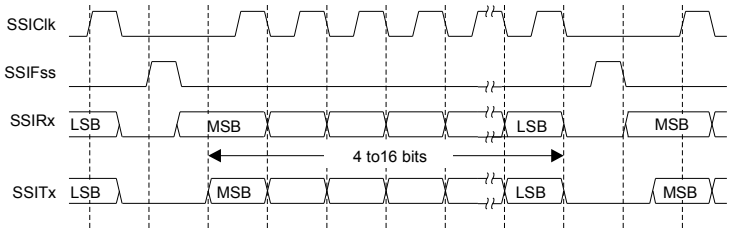


Figure 19-5. Freescale SPI Format (Continuous Transfer) With SPO = 0 and SPH = 0

In this configuration, during idle periods:

- **SSIClk** is forced low.
- **SSIFss** is forced high.
- The transmit data line **SSITx** is arbitrarily forced low.
- When the SSI is configured as a master, it enables the **SSIClk** pad.
- When the SSI is configured as a slave, it disables the **SSIClk** pad.

If the SSI is enabled and valid data is in the TX FIFO, the start of transmission is signified by the **SSIFss** master signal being driven low, causing enabling of slave data onto the **SSIRx** input line of the master. The master **SSITx** output pad is enabled.

One-half **SSIClk** period later, valid master data is transferred to the **SSITx** pin. Once both the master and slave data are set, the **SSIClk** master clock pin goes high after an additional one-half **SSIClk** period.

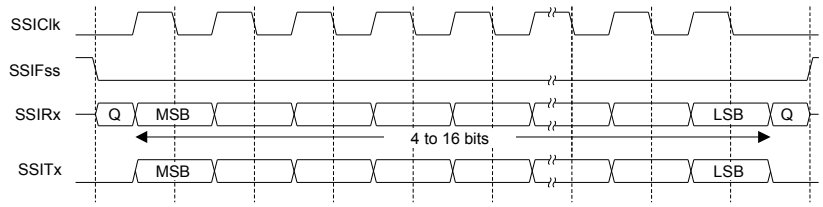
The data is now captured on the rising edges and propagated on the falling edges of the **SSIClk** signal.

In the case of a single-word transmission, after all bits of the data word are transferred, the **SSIFss** line is returned to its idle high state one **SSIClk** period after the last bit is captured.

However, in the case of continuous back-to-back transmissions, the **SSIFss** signal must pulse high between each data word transfer because the slave select pin freezes the data in its serial peripheral register and does not allow it to be altered if the **SPH** bit is clear. Therefore, the master device must raise the **SSIFss** pin of the slave device between each data transfer to enable the serial peripheral data write. When the continuous transfer completes, the **SSIFss** pin is returned to its IDLE state one **SSIClk** period after the last bit is captured.

19.4.4.4 Freescale SPI Frame Format With SPO = 0 and SPH = 1

Figure 19-6 shows the transfer signal sequence for Freescale SPI format with **SPO** = 0 and **SPH** = 1, which covers both single and continuous transfers.



Note: Q is undefined.

Figure 19-6. Freescale SPI Frame Format With SPO = 0 and SPH = 1

In this configuration, during idle periods:

- **SSIClk** is forced low.
- **SSIFss** is forced high.
- The transmit data line **SSITx** is arbitrarily forced low.
- When the SSI is configured as a master, it enables the **SSIClk** pad.
- When the SSI is configured as a slave, it disables the **SSIClk** pad.

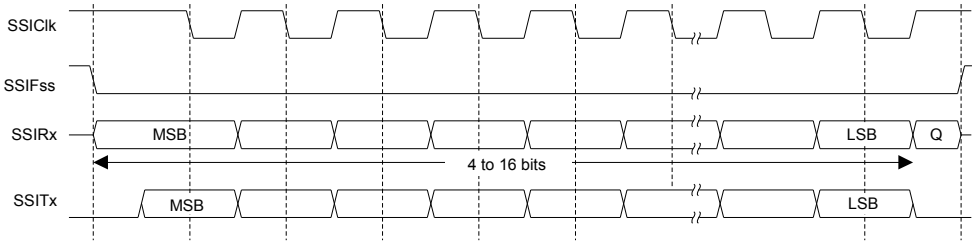
If the SSI is enabled and valid data is in the TX FIFO, the start of transmission is signified by the **SSIFss** master signal going low. The master **SSITx** output is enabled. After an additional one-half **SSIClk** period, both master and slave valid data are enabled onto their respective transmission lines. At the same time, **SSIClk** is enabled with a rising-edge transition. Data is then captured on the falling edges and propagated on the rising edges of the **SSIClk** signal.

In the case of a single-word transfer, after all bits are transferred, the **SSIFss** line is returned to its idle high state one **SSIClk** period after the last bit is captured.

For continuous back-to-back transfers, the **SSIFss** pin is held low between successive data words, and termination is the same as that of the single word transfer.

19.4.4.5 Freescale SPI Frame Format With SPO = 1 and SPH = 0

Figure 19-7 and Figure 19-8 shows single and continuous transmission signal sequences, respectively, for Freescale SPI format with **SPO** = 1 and **SPH** = 0.



Note: Q is undefined.

Figure 19-7. Freescale SPI Frame Format (Single Transfer) With SPO = 1 and SPH = 0

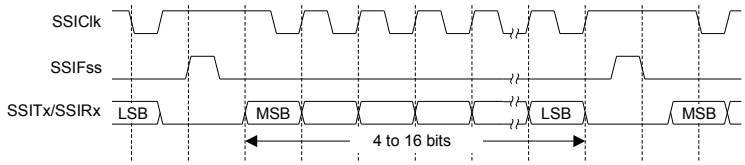


Figure 19-8. Freescale SPI Frame Format (Continuous Transfer) With SPO = 1 and SPH = 0

In this configuration, during idle periods:

- **SSIClk** is forced high.
- **SSIFss** is forced high.
- The transmit data line **SSITx** is arbitrarily forced low.
- When the SSI is configured as a master, it enables the **SSIClk** pad.
- When the SSI is configured as a slave, it disables the **SSIClk** pad.

If the SSI is enabled and valid data is in the transmit FIFO, the start of transmission is signified by the **SSIFss** master signal going low, causing slave data to be immediately transferred onto the **SSIRx** line of the master. The master **SSITx** output pad is enabled.

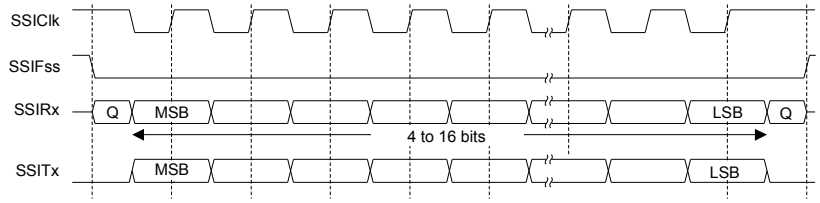
One-half period later, valid master data is transferred to the **SSITx** line. Once both the master and slave data have been set, the **SSIClk** master clock pin becomes low after one additional half **SSIClk** period, meaning that data is captured on the falling edges and propagated on the rising edges of the **SSIClk** signal.

In the case of a single word transmission, after all bits of the data word are transferred, the **SSIFss** line is returned to its idle high state one **SSIClk** period after the last bit is captured.

However, in the case of continuous back-to-back transmissions, the **SSIFss** signal must pulse high between each data word transfer because the slave select pin freezes the data in its serial peripheral register and does not allow it to be altered if the **SPH** bit is clear. Therefore, the master device must raise the **SSIFss** pin of the slave device between each data transfer to enable the serial peripheral data write. On completion of the continuous transfer, the **SSIFss** pin is returned to its idle state one **SSIClk** period after the last bit is captured.

19.4.4.6 Freescale SPI Frame Format With SPO = 1 and SPH = 1

The transfer signal sequence for Freescale SPI format with **SPO** = 1 and **SPH** = 1 is shown in Figure 19-9, which covers both single and continuous transfers.



Note: Q is undefined.

Figure 19-9. Freescale SPI Frame Format With SPO = 1 and SPH = 1

In this configuration, during idle periods:

- **SSIClk** is forced high.
- **SSIFss** is forced high.
- The transmit data line **SSITx** is arbitrarily forced low.
- When the SSI is configured as a master, it enables the **SSIClk** pad.
- When the SSI is configured as a slave, it disables the **SSIClk** pad.

If the SSI is enabled and valid data is in the TX FIFO, the start of transmission is signified by the **SSIFss** master signal going low. The master **SSITx** output pad is enabled. After an additional one-half **SSIClk** period, both master and slave data are enabled onto their respective transmission lines. At the same time, **SSIClk** is enabled with a falling edge transition. Data is then captured on the rising edges and propagated on the falling edges of the **SSIClk** signal.

In the case of a single word transmission, after all bits are transferred the **SSIFss** line is returned to its idle high state one **SSIClk** period after the last bit is captured.

For continuous back-to-back transmissions, the **SSIFss** pin remains in its active low state until the final bit of the last word is captured and then returns to its idle state as previously described.

For continuous back-to-back transfers, the **SSIFss** pin is held low between successive data words, and termination is the same as that of the single-word transfer.

19.4.4.7 MICROWIRE Frame Format

Figure 19-10 shows the MICROWIRE frame format for a single frame. Figure 19-11 shows the same format when back-to-back frames are transmitted.

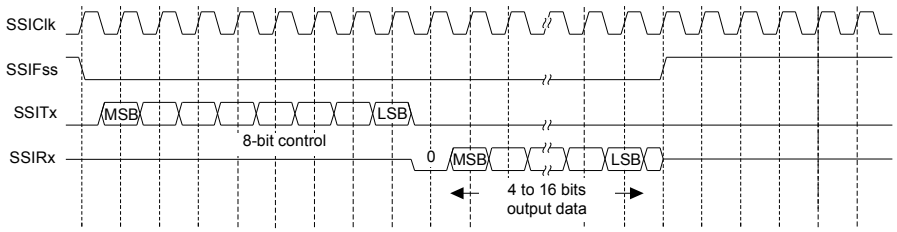


Figure 19-10. MICROWIRE Frame Format (Single Frame)

MICROWIRE format is very similar to SPI format, except that transmission is half-duplex instead of full-duplex and uses a master-slave message passing technique. Each serial transmission begins with an 8-bit control word that is transmitted from the SSI to the off-chip slave device. During this transmission, the SSI does not receive incoming data. After the message is sent, the off-chip slave decodes it and, after waiting one serial clock after the last bit of the 8-bit control message is sent, responds with the required data. The returned data is 4 to 16 bits long, making the total frame length anywhere from 13 to 25 bits.

In this configuration, during idle periods:

- **SSIClk** is forced low.
- **SSIFss** is forced high.
- The transmit data line **SSITx** is arbitrarily forced low.

A transmission is triggered by writing a control byte to the transmit FIFO. The falling edge of **SSIFss** causes the value contained in the bottom entry of the TX FIFO to be transferred to the serial shift register of the transmit logic and the MSB of the 8-bit control frame to be shifted out onto the **SSITx** pin. **SSIFss** remains low for the duration of the frame transmission. The **SSIRx** pin remains 3-stated during this transmission.

The off-chip serial slave device latches each control bit into its serial shifter on each rising edge of **SSIClk**. After the last bit is latched by the slave device, the control byte is decoded during a one clock wait-state, and the slave responds by transmitting data back to the SSI. Each bit is driven onto the **SSIRx** line on the falling edge of **SSIClk**. The SSI in turn latches each bit on the rising edge of **SSIClk**. At the end of the frame, for single transfers, the **SSIFss** signal is pulled high one clock period after the last bit is latched in the receive serial shifter, thus transferring the data to the RX FIFO.

NOTE: The off-chip slave device can 3-state the receive line either on the falling edge of **SSIClk** after the LSB has been latched by the receive shifter or when the **SSIFss** pin goes High.

For continuous transfers, data transmission begins and ends in the same manner as a single transfer. However, the **SSIFss** line is continuously asserted (held low) and transmission of data occurs back-to-back. The control byte of the next frame follows directly after the LSB of the received data from the current frame. Each received value is transferred from the receive shifter on the falling edge of **SSIClk**, after the LSB of the frame is latched into the SSI.

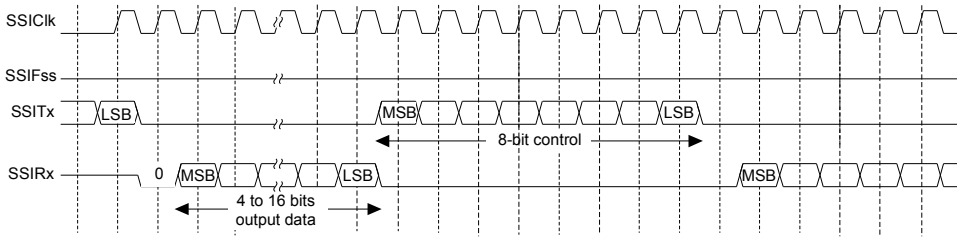


Figure 19-11. MICROWIRE Frame Format (Continuous Transfer)

In the MICROWIRE mode, the SSI slave samples the first bit of receive data on the rising edge of **SSIClk** after **SSIFss** has gone low. Masters that drive a free-running **SSIClk** must ensure that the **SSIFss** signal has sufficient setup and hold margins with respect to the rising edge of **SSIClk**.

Figure 19-12 shows these setup and hold time requirements. With respect to the **SSIClk** rising edge on which the first bit of receive data is to be sampled by the SSI slave, **SSIFss** must have a setup of at least two times the period of **SSIClk** on which the SSI operates. With respect to the **SSIClk** rising edge previous to this edge, **SSIFss** must have a hold of at least one **SSIClk** period.

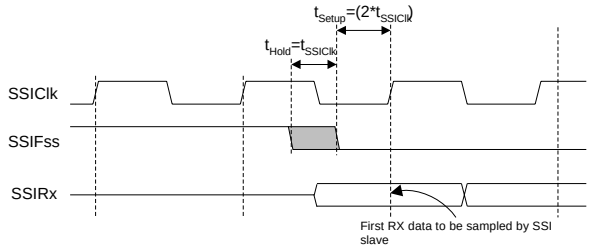


Figure 19-12. MICROWIRE Frame Format, SSIFss Input Setup and Hold Requirements

19.5 DMA Operation

The SSI peripheral provides an interface to the µDMA controller with separate channels for transmit and receive. The µDMA operation of the SSI is enabled through the **SSI DMA Control (SSIDMACTL)** register. When µDMA operation is enabled, the SSI asserts a µDMA request on the receive or transmit channel when the associated FIFO can transfer data. For the receive channel, a single transfer request is asserted whenever any data is in the RX FIFO. A burst transfer request is asserted whenever the amount of data in the RX FIFO is four or more items. For the transmit channel, a single transfer request is asserted whenever at least one empty location is in the TX FIFO. The burst request is asserted whenever the TX FIFO has four or more empty slots. The single and burst µDMA transfer requests are handled automatically by the µDMA controller depending how the µDMA channel is configured. To enable µDMA operation for the receive channel, the **RXDMAE** bit of the **DMA Control (SSIDMACTL)** register must be set. To enable µDMA operation for the transmit channel, the **TXDMAE** bit of the **SSIDMACTL** register must be set. If the µDMA is enabled, then the µDMA controller triggers an interrupt when a transfer is complete. The interrupt occurs on the SSI interrupt vector. Therefore, if interrupts are used for SSI operation and the µDMA is enabled, the SSI interrupt handler must be designed to handle the µDMA completion interrupt.

For more details about programming the µDMA controller, see Chapter 10.

19.6 Initialization and Configuration

To enable and initialize the SSI, perform the following steps:

1. Enable the SSI module using the **SYS_CTRL_RCGCSSI** register (see SYS_CTRL_RCGCSSI).
2. Set the GPIO pin configuration through the **Pxx_SEL** (IOC_PA0_SEL) registers for the desired output pins using the appropriate signal select value.
3. Connect the appropriate input signals to the SSI module through the following registers:
   - **IOC_CLK_SSI_SSI0** IOC_CLK_SSI_SSI0
   - **IOC_SSIRXD_SSI0** IOC_SSIRXD_SSI0
   - **IOC_SSIFSSIN_SSI0** IOC_SSIFSSIN_SSI0
   - **IOC_CLK_SSIN_SSI0** IOC_CLK_SSIIN_SSI0
   - **IOC_CLK_SSI_SSI1** IOC_CLK_SSI_SSI1
   - **IOC_SSIRXD_SSI1** IOC_SSIRXD_SSI1
   - **IOC_SSIFSSIN_SSI1** IOC_SSIFSSIN_SSI1
   - **IOC_CLK_SSIN_SSI1** IOC_CLK_SSIIN_SSI1
4. For more information on pin connections, see Section 9.1.1, *I/O Muxing*, of Chapter 9, *General Purpose Input/Outputs (GPIO)*.

For each of the frame formats, the SSI is configured using the following steps:

1. Ensure that the **SSE** bit in the **SSI_CR1** register is clear before making any configuration changes.
2. Select whether the SSI is a master or slave:
   (a) For master operations, set the **SSI_CR1** register to 0x0000 0000.
   (b) For slave mode (output enabled), set the **SSI_CR1** register to 0x0000 0004.
   (c) For slave mode (output disabled), set the **SSI_CR1** register to 0x0000 000C.
3. Configure the SSI clock source by writing to the **SSI_CC** register (see SSI_CC)
4. Configure the clock prescale divisor by writing the **SSI_CPSR** register.
5. Write the **SSI_CR0** register with the following configuration:
   - Serial clock rate (**SCR**)
   - Desired clock phase and polarity, if using Freescale SPI mode (**SPH** and **SPO**)
   - The protocol mode: Freescale SPI, TI SSF, MICROWIRE (**FRF**)
   - The data size (**DSS**)
6. Optionally, configure the µDMA channel (see Chapter 10) and enable the DMA options in the **SSI_DMACTL** register.
7. Enable the SSI by setting the **SSE** bit in the **SSI_CR1** register.

As an example, assume that the SSI configuration is required to operate with the following parameters:

- Master operation
- Freescale SPI mode (SPO = 1, SPH = 1)
- 1 Mbps bit rate
- 8 data bits

Assuming the system clock is 20 MHz, the bit rate calculation is:

```
SSIClk = SysClk / (CPSDVSR × (1 + SCR))
1x106  = 20x106 / (CPSDVSR × (1 + SCR))
```

In this case, if **CPSDVSR** = 0x2, **SCR** must be 0x9.

The configuration sequence i as follows:

1. Ensure that the **SSE** bit in the **SSI_CR1** register is clear.
2. Write the **SSI_CR1** register with a value of 0x0000.0000.
3. Write the **SSI_CPSR** register with a value of 0x0000.0002.
4. Write the **SSICR0** register with a value of 0x0000.09C7.
5. The SSI is then enabled by setting the **SSE** bit in the **SSI_CR1** register.

19.7 SSI Registers

19.7.1 SSI Registers

19.7.1.1 SSI Registers Mapping Summary

This section provides information on the SSI module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

19.7.1.1.1 SSI Common Registers Mapping

This section provides information on the SSI module instance within this product. Each of the registers within the Module Instance is described separately below.

Table 19-2. SSI Common Registers Mapping Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset
SSI_CR0	RW	32	0x0000 0000	0x000
SSI_CR1	RW	32	0x0000 0000	0x004
SSI_DR	RW	32	0x0000 0000	0x008
SSI_SR	RO	32	0x0000 0003	0x00C
SSI_CPSR	RW	32	0x0000 0000	0x010
SSI_IM	RW	32	0x0000 0000	0x014
SSI_RIS	RO	32	0x0000 0008	0x018
SSI_MIS	RO	32	0x0000 0000	0x01C
SSI_ICR	RW	32	0x0000 0000	0x020
SSI_DMACTL	RW	32	0x0000 0000	0x024
SSI_CC	RW	32	0x0000 0000	0xFC8

19.7.1.1.2 SSI Instances Register Mapping Summary

19.7.1.1.2.1 SSI0 Register Summary

Table 19-3. SSI0 Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
SSI_CR0	RW	32	0x0000 0000	0x000	0x4000 8000
SSI_CR1	RW	32	0x0000 0000	0x004	0x4000 8004
SSI_DR	RW	32	0x0000 0000	0x008	0x4000 8008
SSI_SR	RO	32	0x0000 0003	0x00C	0x4000 800C
SSI_CPSR	RW	32	0x0000 0000	0x010	0x4000 8010
SSI_IM	RW	32	0x0000 0000	0x014	0x4000 8014
SSI_RIS	RO	32	0x0000 0008	0x018	0x4000 8018
SSI_MIS	RO	32	0x0000 0000	0x01C	0x4000 801C
SSI_ICR	RW	32	0x0000 0000	0x020	0x4000 8020
SSI_DMACTL	RW	32	0x0000 0000	0x024	0x4000 8024
SSI_CC	RW	32	0x0000 0000	0xFC8	0x4000 8FC8

19.7.1.1.2.2 SSI1 Register Summary

Table 19-4. SSI1 Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
SSI_CR0	RW	32	0x0000 0000	0x000	0x4000 9000
SSI_CR1	RW	32	0x0000 0000	0x004	0x4000 9004
SSI_DR	RW	32	0x0000 0000	0x008	0x4000 9008
SSI_SR	RO	32	0x0000 0003	0x00C	0x4000 900C
SSI_CPSR	RW	32	0x0000 0000	0x010	0x4000 9010
SSI_IM	RW	32	0x0000 0000	0x014	0x4000 9014
SSI_RIS	RO	32	0x0000 0008	0x018	0x4000 9018
SSI_MIS	RO	32	0x0000 0000	0x01C	0x4000 901C
SSI_ICR	RW	32	0x0000 0000	0x020	0x4000 9020
SSI_DMACTL	RW	32	0x0000 0000	0x024	0x4000 9024
SSI_CC	RW	32	0x0000 0000	0xFC8	0x4000 9FC8

19.7.1.2 SSI Common Register Descriptions

SSI_CR0

Address offset	0x000		
Physical Address	0x4000 8000 0x4000 9000	**Instance**	SSI0 SSI1
Description	The CR0 register contains bit fields that control various functions within the SSI module. Functionality such as protocol mode, clock rate, and data size are configured in this register.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																SCR								SPH	SPO	FRF		DSS			

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15:8	SCR	SSI serial clock rate (R/W) Reset value: 0x0 The value SCR is used to generate the transmit and receive bit rate of the SSI. Where the bit rate is: BR = FSSICLK/(CPSDVR * (1 + SCR)) where CPSDVR is an even value from 2-254, programmed in the SSICPSR register and SCR is a value from 0-255.	RW	0x00
7	SPH	SSI serial clock phase (R/W) Reset value: 0x0 This bit is only applicable to the Motorola SPI Format.	RW	0
6	SPO	SSI serial clock phase (R/W) Reset value: 0x0 This bit is only applicable to the Motorola SPI Format.	RW	0
5:4	FRF	SSI frame format select (R/W) Reset value: 0x0 00: Motorola SPI frame format 01: TI synchronous serial frame format 10: National Microwire frame format 11: Reserved	RW	0x0

Bits	Field Name	Description	Type	Reset
3:0	DSS	SSI data size select (R/W) Reset value: 0x0 0000-0010: Reserved 0011: 4-bit data 0100: 5-bit data 0101: 6-bit data 0110: 7-bit data 0111: 8-bit data 1000: 9-bit data 1001: 10-bit data 1010: 11-bit data 1011: 12-bit data 1100: 13-bit data 1101: 14-bit data 1110: 15-bit data 1111: 16-bit data	RW	0x0

SSI_CR1

Address offset	0x004		
Physical Address	0x4000 8004 0x4000 9004	**Instance**	SSI0 SSI1
Description	The CR1 register contains bit fields that control various functions within the SSI module. Master and slave mode functionality is controlled by this register.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																RESERVED												SOD	MS	SSE	LBM

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RW	0x0000
15:4	RESERVED	This bit field is reserved.	RW	0x000
3	SOD	SSI slave mode output disable (R/W) Reset value: 0x0 This bit is relevant only in the slave mode (MS = 1). In multiple-slave systems, it is possible for the SSI master to broadcast a message to all slaves in the system while ensuring that only one slave drives data onto the serial output line. In such systems, the RXD lines from multiple slaves could be tied together. To operate in such a system, the SOD bit can be set if the SSI slave is not suppose to drive the SSITXD line. 0: SSI can drive SSITXD in slave output mode 1: SSI must not drive the SSITXD output in slave mode	RW	0
2	MS	SSI master and slave select (R/W) Reset value: 0x0 This bit can be modified only when the SSI is disabled (SSE = 0). 0: Device configured as a master (default) 1: Device configured as a slave	RW	0
1	SSE	SSI synchronous serial port enable (R/W) Reset value: 0x0 0: SSI operation is disabled. 1: SSI operation is enabled.	RW	0
0	LBM	SSI loop-back mode (R/W) Reset value: 0x0 0: Normal serial port operation is enabled. 1: The output of the transmit serial shifter is connected to the input of the receive serial shift register internally.	RW	0

SSI_DR

Address offset	0x008		
Physical Address	0x4000 8008 0x4000 9008	**Instance**	SSI0 SSI1
Description	The DR register is 16 bits wide. When the SSIDR register is read, the entry in the receive FIFO that is pointed to by the current FIFO read pointer is accessed. When a data value is removed by the SSI receive logic from the incoming data frame, it is placed into the entry in the receive FIFO pointed to by the current FIFO write pointer. When the DR register is written to, the entry in the transmit FIFO that is pointed to by the write pointer is written to. Data values are removed from the transmit FIFO one value at a time by the transmit logic. Each data value is loaded into the transmit serial shifter, then serially shifted out onto the SSITx pin at the programmed bit rate. When a data size of less than 16 bits is selected, the user must right-justify data written to the transmit FIFO. The transmit logic ignores the unused bits. Received data less than 16 bits is automatically right-justified in the receive buffer. When the SSI is programmed for MICROWIRE frame format, the default size for transmit data is eight bits (the most significant byte is ignored). The receive data size is controlled by the programmer. The transmit FIFO and the receive FIFO are not cleared even when the SSE bit in the SSICR1 register is cleared, allowing the software to fill the transmit FIFO before enabling the SSI.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15:0	DATA	SSI receive/transmit data register (R/W) Reset value: 0xXXXX A read operation reads the receive FIFO. A write operation writes the transmit FIFO. Software must right-justify data when the SSI is programmed for a data size that is less than 16 bits. Unused bits at the top are ignored by the transmit logic. The receive logic automatically right-justified the data.	RW	0x0000

SSI_SR

Address offset	0x00C		
Physical Address	0x4000 800C 0x4000 900C	**Instance**	SSI0 SSI1
Description	The SR register contains bits that indicate the FIFO fill status and the SSI busy status.		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																RESERVED											BSY	RFF	RNE	TNF	TFE

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15:5	RESERVED	This bit field is reserved.	RO	0x000
4	BSY	SSI busy bit (RO) Reset value: 0x0 0: SSI is idle. 1: SSI is currently transmitting and/or receiving a frame or the transmit FIFO is not empty.	RO	0
3	RFF	SSI receive FIFO full (RO) Reset value: 0x0 0: Receive FIFO is not full. 1: Receive FIFO is full.	RO	0
2	RNE	SSI receive FIFO not empty (RO) Reset value: 0x0 0: Receive FIFO is empty. 1: Receive FIFO is not empty.	RO	0

Bits	Field Name	Description	Type	Reset
1	TNF	SSI transmit FIFO not full (RO) Reset value: 0x1 0: Transmit FIFO is full. 1: Transmit FIFO is not full.	RO	1
0	TFE	SSI transmit FIFO empty (RO) Reset value: 0x1 0: Transmit FIFO is not empty. 1: Transmit FIFO is empty.	RO	1

SSI_CPSR

Address offset 0x010

Physical Address 0x4000 8010 **Instance** SSI0
0x4000 9010 SSI1

Description The CPSR register specifies the division factor which is used to derive the SSIClk from the system clock. The clock is further divided by a value from 1 to 256, which is 1 + SCR. SCR is programmed in the SSICR0 register. The frequency of the SSIClk is defined by:
SSIClk = SysClk / (CPSDVSR x (1 + SCR))
The value programmed into this register must be an even number between 2 and 254. The least-significant bit of the programmed number is hard-coded to zero. If an odd number is written to this register, data read back from this register has the least-significant bit as zero.

Type RW

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

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15:8	RESERVED	This bit field is reserved.	RO	0x00
7:0	CPSDVSR	SSI clock prescale divisor (R/W) Reset value: 0x0 This value must be an even number from 2 to 254, depending on the frequency of SSICLK. The LSB always returns zero on reads.	RW	0x00

SSI_IM

Address offset 0x014

Physical Address 0x4000 8014 **Instance** SSI0
0x4000 9014 SSI1

Description The IM register is the interrupt mask set or clear register. It is a read/write register and all bits are cleared on reset. On a read, this register gives the current value of the mask on the corresponding interrupt. Setting a bit sets the mask, preventing the interrupt from being signaled to the interrupt controller. Clearing a bit clears the corresponding mask, enabling the interrupt to be sent to the interrupt controller.

Type RW

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												TXIM	RXIM	RTIM	RORIM

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	RO	0x000 0000
3	TXIM	SSI transmit FIFO interrupt mask (R/W) Reset value: 0x0 0: TX FIFO half empty or condition interrupt is masked. 1: TX FIFO half empty or less condition interrupt is not masked.	RW	0

Bits	Field Name	Description	Type	Reset
2	RXIM	SSI receive FIFO interrupt mask (R/W) Reset value: 0x0 0: RX FIFO half empty or condition interrupt is masked. 1: RX FIFO half empty or less condition interrupt is not masked.	RW	0
1	RTIM	SSI receive time-out interrupt mask (R/W) Reset value: 0x0 0: RX FIFO time-out interrupt is masked. 1: RX FIFO time-out interrupt is not masked	RW	0
0	RORIM	SSI receive overrun interrupt mask (R/W) Reset value: 0x0 0: RX FIFO Overrun interrupt is masked. 1: RX FIFO Overrun interrupt is not masked	RW	0

SSI_RIS

Address offset	0x018		
Physical Address	0x4000 8018 0x4000 9018	**Instance**	SSI0 SSI1
Description	The RIS register is the raw interrupt status register. On a read, this register gives the current raw status value of the corresponding interrupt before masking. A write has no effect.		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																RESERVED												TXRIS	RXRIS	RTRIS	RORRIS

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15:4	RESERVED	This bit field is reserved.	RO	0x000
3	TXRIS	SSI SSITXINTR raw state (RO) Reset value: 0x1 Gives the raw interrupt state (before masking) of SSITXINTR	RO	1
2	RXRIS	SSI SSIRXINTR raw state (RO) Reset value: 0x0 Gives the raw interrupt state (before masking) of SSIRXINTR	RO	0
1	RTRIS	SSI SSIRTINTR raw state (RO) Reset value: 0x0 Gives the raw interrupt state (before masking) of SSIRTINTR	RO	0
0	RORRIS	SSI SSIRORINTR raw state (RO) Reset value: 0x0 Gives the raw interrupt state (before masking) of SSIRORINTR	RO	0

SSI_MIS

Address offset	0x01C		
Physical Address	0x4000 801C 0x4000 901C	**Instance**	SSI0 SSI1
Description	The MIS register is the masked interrupt status register. On a read, this register gives the current masked status value of the corresponding interrupt. A write has no effect.		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																RESERVED												TXMIS	RXMIS	RTMIS	RORMIS

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15:4	RESERVED	This bit field is reserved.	RO	0x000
3	TXMIS	SSI SSITXINTR masked state (RO) Reset value: 0x0 Gives the interrupt state (after masking) of SSITXINTR	RO	0
2	RXMIS	SSI SSIRXINTR masked state (RO) Reset value: 0x0 Gives the interrupt state (after masking) of SSIRXINTR	RO	0
1	RTMIS	SSI SSIRTINTR masked state (RO) Reset value: 0x0 Gives the interrupt state (after masking) of SSIRTINTR	RO	0
0	RORMIS	SSI SSIRORINTR masked state (RO) Reset value: 0x0 Gives the interrupt state (after masking) of SSIRORINTR	RO	0

SSI_ICR

Address offset	0x020		
Physical Address	0x4000 8020 0x4000 9020	**Instance**	SSI0 SSI1
Description	The ICR register is the interrupt clear register. On a write of 1, the corresponding interrupt is cleared. A write of 0 has no effect.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15:2	RESERVED	This bit field is reserved.	RO	0x0000
1	RTIC	SSI receive time-out interrupt clear (W1C) Reset value: 0x0 0: No effect on interrupt 1: Clears interrupt	RW W1C	0
0	RORIC	SSI receive overrun interrupt clear (W1C) Reset value: 0x0 0: No effect on interrupt 1: Clears interrupt	RW W1C	0

SSI_DMACTL

Address offset	0x024		
Physical Address	0x4000 8024 0x4000 9024	**Instance**	SSI0 SSI1
Description	The DMACTL register is the uDMA control register.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	RO	0x0000 0000
1	TXDMAE	Transmit DMA enable 0: uDMA for the transmit FIFO is disabled. 1: uDMA for the transmit FIFO is enabled.	RW	0
0	RXDMAE	Receive DMA enable 0: uDMA for the receive FIFO is disabled. 1: uDMA for the receive FIFO is enabled.	RW	0

SSI_CC

Address offset 0xFC8

Physical Address 0x4000 8FC8 **Instance** SSI0
0x4000 9FC8 SSI1

Description SSI clock configuration
The CC register controls the baud clock and system clocks sources for the SSI module.
Note: If the PIOSC is used for the SSI baud clock, the system clock frequency must be at least 16 MHz in run mode.

Type RW

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

Bits	Field Name	Description	Type	Reset
31:3	RESERVED	This bit field is reserved.	RO	0x0000 0000
2:0	CS	SSI baud and system clock source The following bits determine the clock source that generates the baud and system clocks for the SSI. bit0 (PIOSC): 1: The SSI baud clock is determined by the IO DIV setting in the system controller. 0: The SSI baud clock is determined by the SYS DIV setting in the system controller. bit1: Unused bit2: (DSEN) Only meaningful when the system is in deep sleep mode. This bit is a don't care when not in sleep mode. 1: The SSI system clock is running on the same clock as the baud clock, as per PIOSC setting above. 0: The SSI system clock is determined by the SYS DIV setting in the system controller.	RW	0x0

Inter-Integrated Circuit Interface

This chapter describes the inter-integrated circuit (I^2C) interface.

Topic **Page**

20.1 Inter-Integrated Circuit Interface

The I²C bus provides bidirectional data transfer through a 2-wire design (a serial data line SDA and a serial clock line SCL), and interfaces to external I²C devices such as serial memory (RAMs and ROMs), networking devices, LCDs, tone generators, and so on. The I²C bus may also be used for system testing and diagnostic purposes in product development and manufacture. The CC2538 device includes one I²C module, providing the ability to interact (both transmit and receive) with other I²C devices on the bus.

The C2538 includes one I²C module with the following features:

- Devices on the I²C bus can be designated as either a master or a slave:
  - Supports both transmitting and receiving data as either a master or a slave
  - Supports simultaneous master and slave operation
- Four I²C modes:
  - Master transmit
  - Master receive
  - Slave transmit
  - Slave receive
- Two transmission speeds: Standard (100 Kbps) and fast (400 Kbps)
- Master and slave interrupt generation:
  - Master generates interrupts when a transmit or receive operation completes (or aborts due to an error)
  - Slave generates interrupts when data has been transferred or requested by a master or when a Start or Stop condition is detected
- Master with arbitration and clock synchronization, multimaster support, and 7-bit addressing mode

20.2 Block Diagram

Figure 20-1 shows the I²C block diagram.

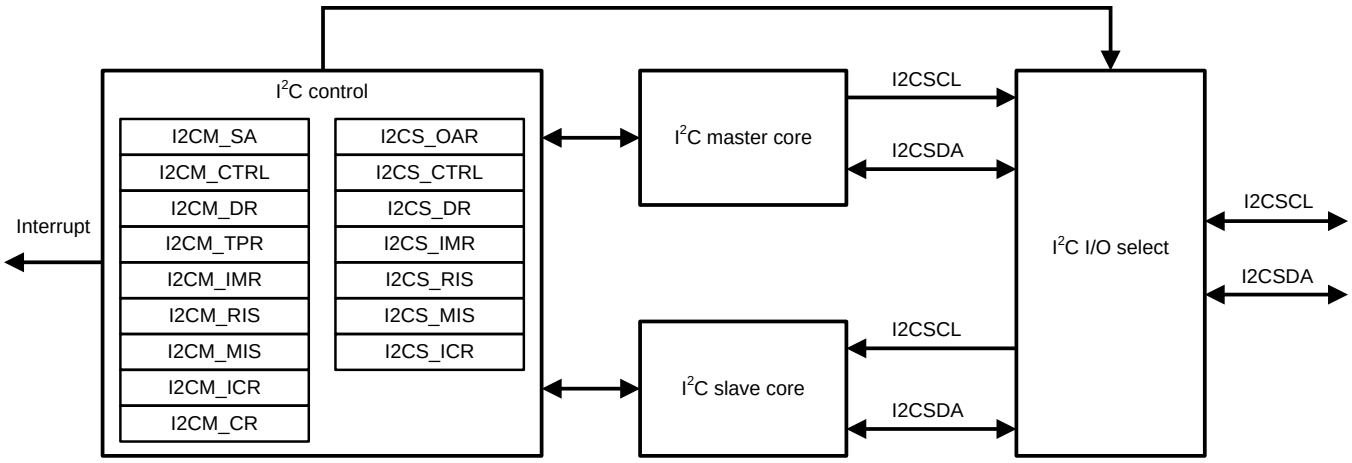


Figure 20-1. I²C Block Diagram

20.3 Functional Description

I²C module is comprised of both master and slave functions. For proper operation, the SDA pins must be configured as open-drain signals. For proper operation, the SDA pin must be configured as an open-drain signal. Figure 20-2 shows a typical I²C bus configuration.

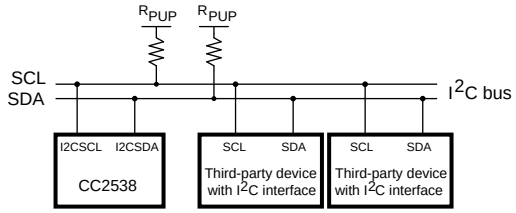


Figure 20-2. I²C Bus Configuration

20.3.1 I²C Bus Functional Overview

The I²C bus uses only two signals: SDA and SCL, named **I2CSDA** and **I2CSCL** on the cc2538 controller. SDA is the bidirectional serial data line and SCL is the bidirectional serial clock line. The bus is considered idle when both lines are high.

Every transaction on the I²C bus is nine bits long, consisting of eight data bits and a single acknowledge bit. The number of bytes per transfer (defined as the time between a valid Start and Stop condition, described in Section 20.3.1.1) is unrestricted, an acknowledge bit must follow each byte, and data must be transferred by the MSB first. When a receiver cannot receive another complete byte, it can hold the clock line SCL low and force the transmitter into a wait-state. The data transfer continues when the receiver releases the clock SCL.

20.3.1.1 Start and Stop Conditions

The protocol of the I²C bus defines two states to begin and end a transaction: Start and Stop. A high-to-low transition on the SDA line while the SCL is high is defined as a Start condition, and a low-to-high transition on the SDA line while SCL is high is defined as a Stop condition. The bus is considered busy after a Start condition and free after a Stop condition. See Figure 20-3.

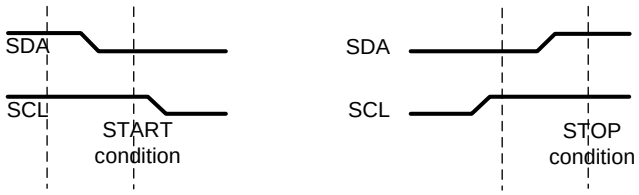


Figure 20-3. Start and Stop Conditions

The STOP bit determines if the cycle stops at the end of the data cycle or continues on to a Repeated Start condition. To generate a single transmit cycle, the **I²C Master Slave Address (I2CM_SA)** register is written with the desired address, the **R/S** bit is cleared, and the Control register is written with **ACK** = X (0 or 1), **STOP** = 1, **START** = 1, and **RUN** = 1 to perform the operation and stop. When the operation is completed (or aborted due an error), the interrupt pin becomes active and the data is readable from the **I²C Master Data (I2CM_DR)** register. When the I²C module operates in master receiver mode, the **ACK** bit is normally set, thus causing the I²C bus controller to transmit an acknowledge automatically after each byte. When the I²C bus controller requires no further data transmission from the slave transmitter, the **ACK** bit must be cleared.

When operating in slave mode, two bits in the **I²C Slave Raw Interrupt Status (I2CS_RIS)** register indicate detection of Start and Stop conditions on the bus, while two bits in the **I²C Slave Masked Interrupt Status (I2CS_MIS)** register allow Start and Stop conditions to be promoted to controller interrupts (when interrupts are enabled).

20.3.1.2 Data Format With 7-Bit Address

Data transfers follow the format shown in Figure 20-4. After the Start condition, a slave address is transmitted. This address is 7 bits long followed by an eighth bit, which is a data direction bit (the **R/S** bit in the **I2CM_SA** register). If the **R/S** bit is clear, it indicates a transmit operation (send), and if it is set, it indicates a request for data (receive). A data transfer is always terminated by a Stop condition generated by the master; however, a master can initiate communications with another device on the bus by generating a Repeated Start condition and addressing another slave without first generating a Stop condition. Various combinations of receive and transmit formats are then possible within a single transfer.

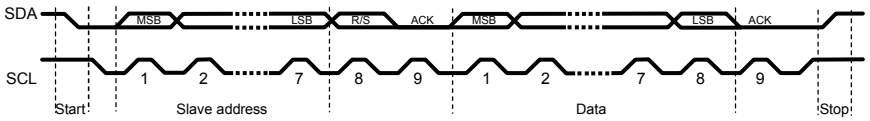


Figure 20-4. Complete Data Transfer With a 7-Bit Address

The first seven bits of the first byte make up the slave address (see Figure 20-5). The eighth bit determines the direction of the message. A 0 in the R/S position of the first byte means that the master transmits (sends) data to the selected slave, and a 1 in this position means that the master receives data from the slave.

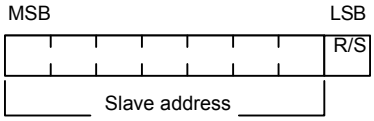


Figure 20-5. R/S Bit in First Byte

20.3.1.3 Data Validity

The SDA line must contain stable data during the high period of the clock, and the data line can change only when SCL is low (see Figure 20-6).

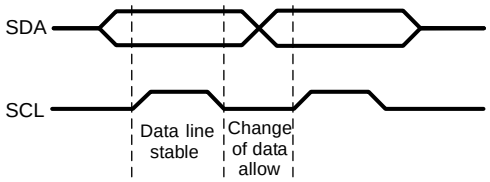


Figure 20-6. Data Validity During Bit Transfer on the I²C Bus

20.3.1.4 Acknowledge

All bus transactions have a required acknowledge clock cycle that is generated by the master. During the acknowledge cycle, the transmitter master or slave) releases the SDA line. To acknowledge the transaction, the receiver must pull down SDA during the acknowledge clock cycle. The data transmitted by the receiver during the acknowledge cycle must comply with the data validity requirements described in Section 20.3.1.3.

When a slave receiver does not acknowledge the slave address, the slave must leave SDA high so that the master can generate a Stop condition and abort the current transfer. If the master device is acting as a receiver during a transfer, it is responsible for acknowledging each transfer made by the slave. Because the master controls the number of bytes in the transfer, it signals the end of data to the slave transmitter by not generating an acknowledge on the last data byte. The slave transmitter must then release SDA to allow the master to generate a Stop or a Repeated Start condition.

20.3.1.5 Arbitration

A master may start a transfer only if the bus is idle. Two or more masters can generate a Start condition within minimum hold time of the Start condition. In these situations, an arbitration scheme occurs on the SDA line, while SCL is high. During arbitration, the first of the competing master devices to place 1 (high) on SDA while another master transmits 0 (low) switches off its data output stage and retires until the bus is idle again.

Arbitration can occur over several bits. The first stage of arbitration is a comparison of address bits; if both masters are trying to address the same device, arbitration continues to the comparison of data bits.

20.3.2 Available Speed Modes

The I^2C bus can run in either standard mode (100 kbps) or fast mode (400 kbps). The selected mode should match the speed of the other I^2C devices on the bus.

20.3.2.1 Standard and Fast Modes

Standard and fast modes are selected using a value in the **I^2C Master Timer Period (I2CM_TPR)** register that results in an SCL frequency of 100 kbps for standard mode or 400 kbps for fast mode.

The I^2C clock rate is determined by the parameters CLK_PRD, TIMER_PRD, SCL_LP, and SCL_HP where:

CLK_PRD is the system clock period.

SCL_LP is the low phase of SCL (fixed at 6).

SCL_HP is the high phase of SCL (fixed at 4).

TIMER_PRD is the programmed value in the **I2CM_TPR** register (see I2CM_TPR).

The I^2C clock period is calculated as follows:

SCL_PERIOD = 2 × (1 + TIMER_PRD) × (SCL_LP + SCL_HP) × CLK_PRD

For example:

CLK_PRD = 50 ns

TIMER_PRD = 2

SCL_LP = 6

SCL_HP = 4

yields a SCL frequency of:

1 / SCL_PERIOD = 333 kHz

Table 20-1 lists examples of the timer periods used to generate both standard and fast mode SCL frequencies based on various system clock frequencies.

Table 20-1. Examples of I^2C Master Timer Period versus Speed Mode

System Clock (MHz)	Timer Period	Standard Mode (kpbs)	Timer Period	Fast Mode (kbps)
4	0x01	100	–	–
8	0x03	100	0x01	200
16	0x07	100	0x01	400
32	0x13	80	0x03	400

20.3.3 Interrupts

The I^2C can generate interrupts when the following conditions are observed:

- Master transaction completed
- Master arbitration lost

- Master transaction error
- Master bus time-out
- Slave transaction received
- Slave transaction requested
- Stop condition on bus detected
- Start condition on bus detected

The I²C master and I²C slave modules have separate interrupt signals. While both modules can generate interrupts for multiple conditions, only a single interrupt signal is sent to the interrupt controller (INTC).

20.3.3.1 I²C Master Interrupts

The I²C master module generates an interrupt when a transaction completes (either transmit or receive), when arbitration is lost, or when an error occurs during a transaction. To enable the I²C master interrupt, software must set the **IM** bit in the **I²C Master Interrupt Mask (I2CM_IMR)** register. When an interrupt condition is met, software must check the **ERROR** and **ARBLST** bits in the **I²C Master Control and Status (I2CM_STAT)** register to verify that an error did not occur during the last transaction and to ensure that arbitration has not been lost. An error condition is asserted if the last transaction was not acknowledged by the slave. If an error is not detected and the master has not lost arbitration, the application can proceed with the transfer. The interrupt is cleared by setting the **IC** bit in the **I²C Master Interrupt Clear (I2CM_ICR)** register to 1.

If the application does not require the use of interrupts, the raw interrupt status is always visible through the **I²C Master Raw Interrupt Status (I2CM_RIS)** register.

20.3.3.2 I²C Slave Interrupts

The slave module can generate an interrupt when data is received or requested. This interrupt is enabled by setting the **DATAIM** bit in the **I²C Slave Interrupt Mask (I2CS_IMR)** register. Software determines whether the module should write (transmit) or read (receive) data from the **I²C Slave Data (I2CS_DR)** register, by checking the **RREQ** and **TREQ** bits of the **I²C Slave Control and Status (I2CS_STAT)** register. If the slave module is in receive mode and the first byte of a transfer is received, the **FBR** bit and **RREQ** bits are set. The interrupt is cleared by setting the **DATAIC** bit in the **I²C Slave Interrupt Clear (I2CS_ICR)** register.

In addition, the slave module can generate an interrupt when a Start and Stop condition is detected. These interrupts are enabled by setting the **STARTIM** and **STOPIM** bits of the **I²C Slave Interrupt Mask (I2CS_IMR)** register and cleared by setting the **STOPIC** and **STARTIC** bits of the **I²C Slave Interrupt Clear (I2CS_ICR)** register to 1.

If the application does not require the use of interrupts, the raw interrupt status is always visible through the **I²C Slave Raw Interrupt Status (I2CS_RIS)** register.

20.3.4 Loopback Operation

The I²C modules can be placed into an internal loopback mode for diagnostic or debug work by setting the **LPBK** bit in the **I²C Master Configuration (I2CM_CR)** register. In loopback mode, the SDA and SCL signals from the master and slave modules are tied together.

20.3.5 Command Sequence Flow Charts

This section details the steps required to perform the various I²C transfer types in both master and slave mode. In order to do that, the SDA and SCL signal ports must be configured by setting IOC_Pxx_SEL, IOC_I2CMSSDA and IOC_I2CMSSCL registers properly.

20.3.5.1 I²C Master Command Sequences

Figure 20-7 through Figure 20-12 show the command sequences available for the I²C master.

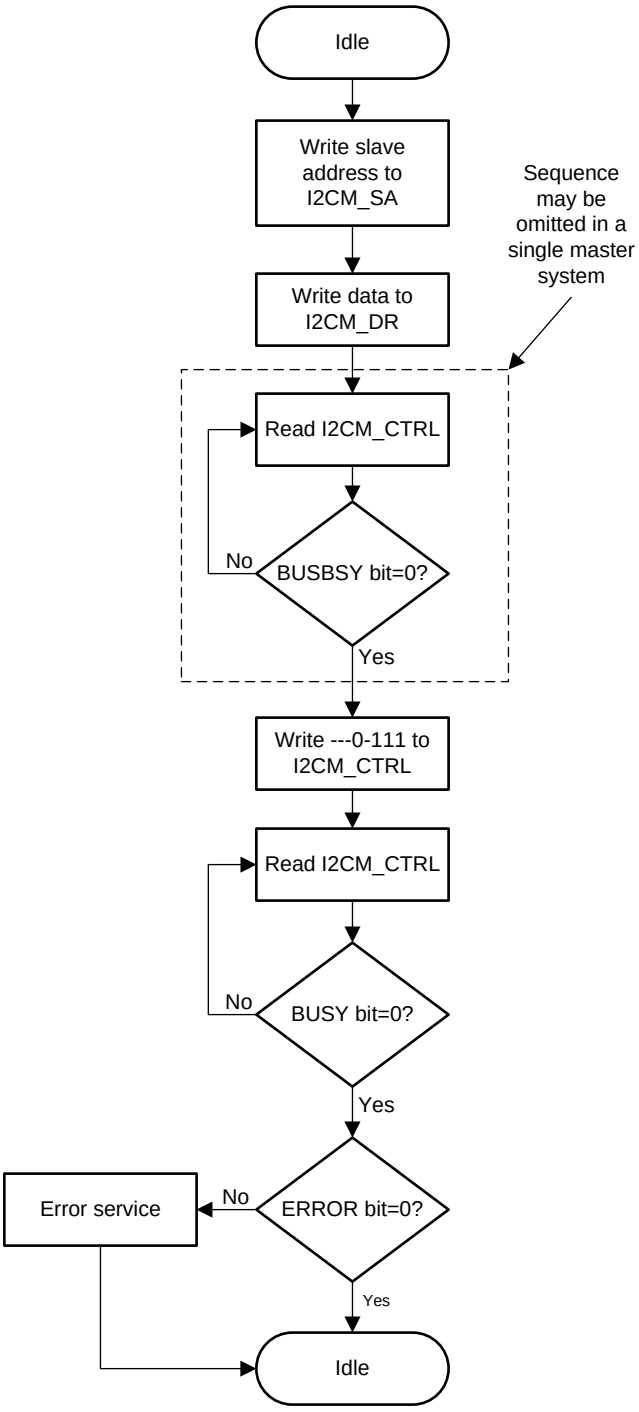


Figure 20-7. Master Single TRANSMIT

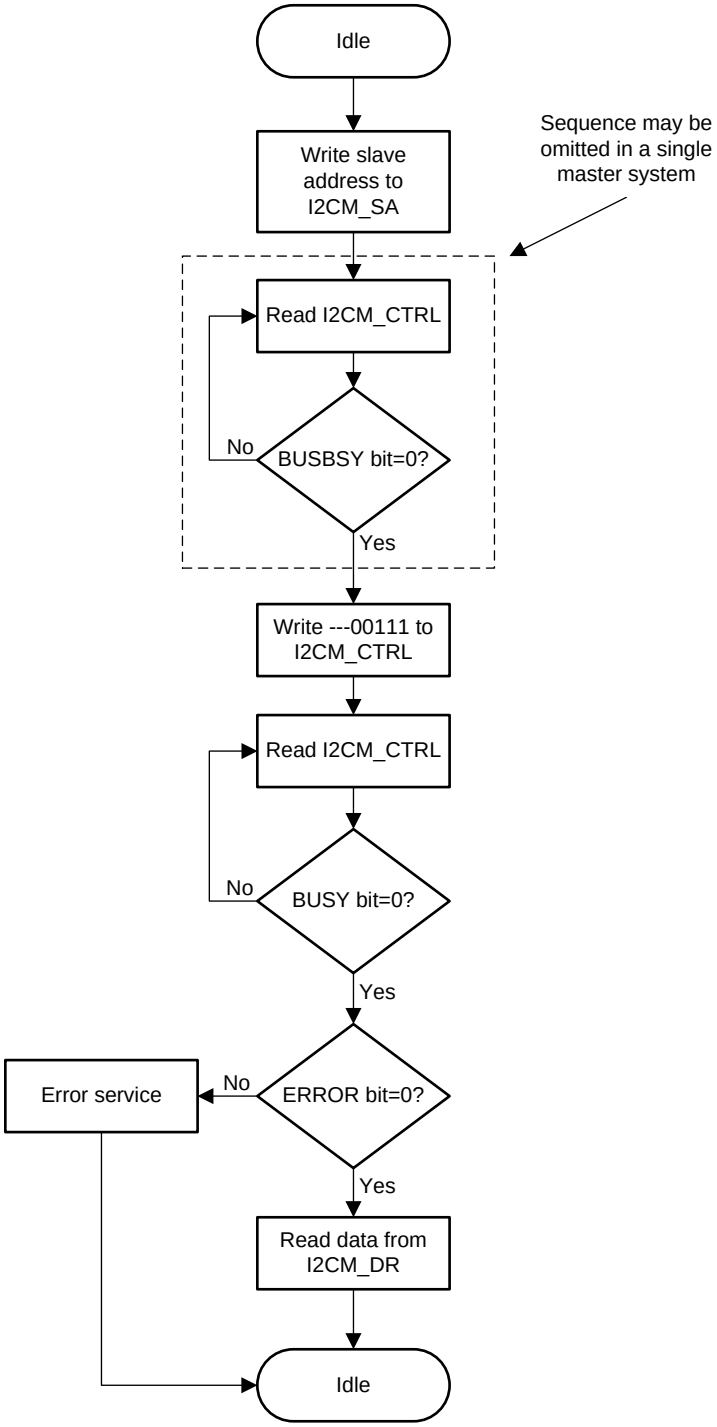


Figure 20-8. Master Single RECEIVE

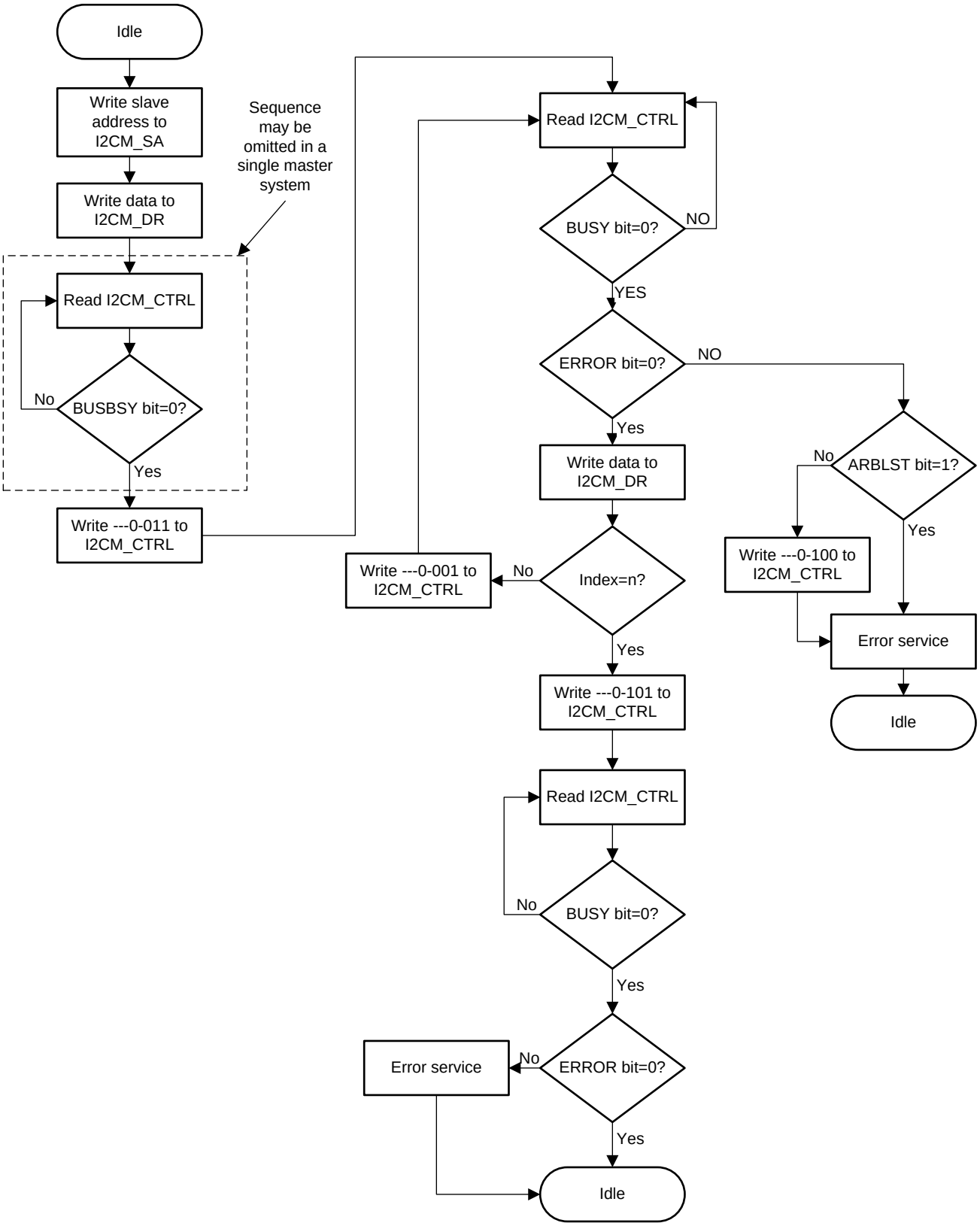


Figure 20-9. Master TRANSMIT With Repeated Start Condition

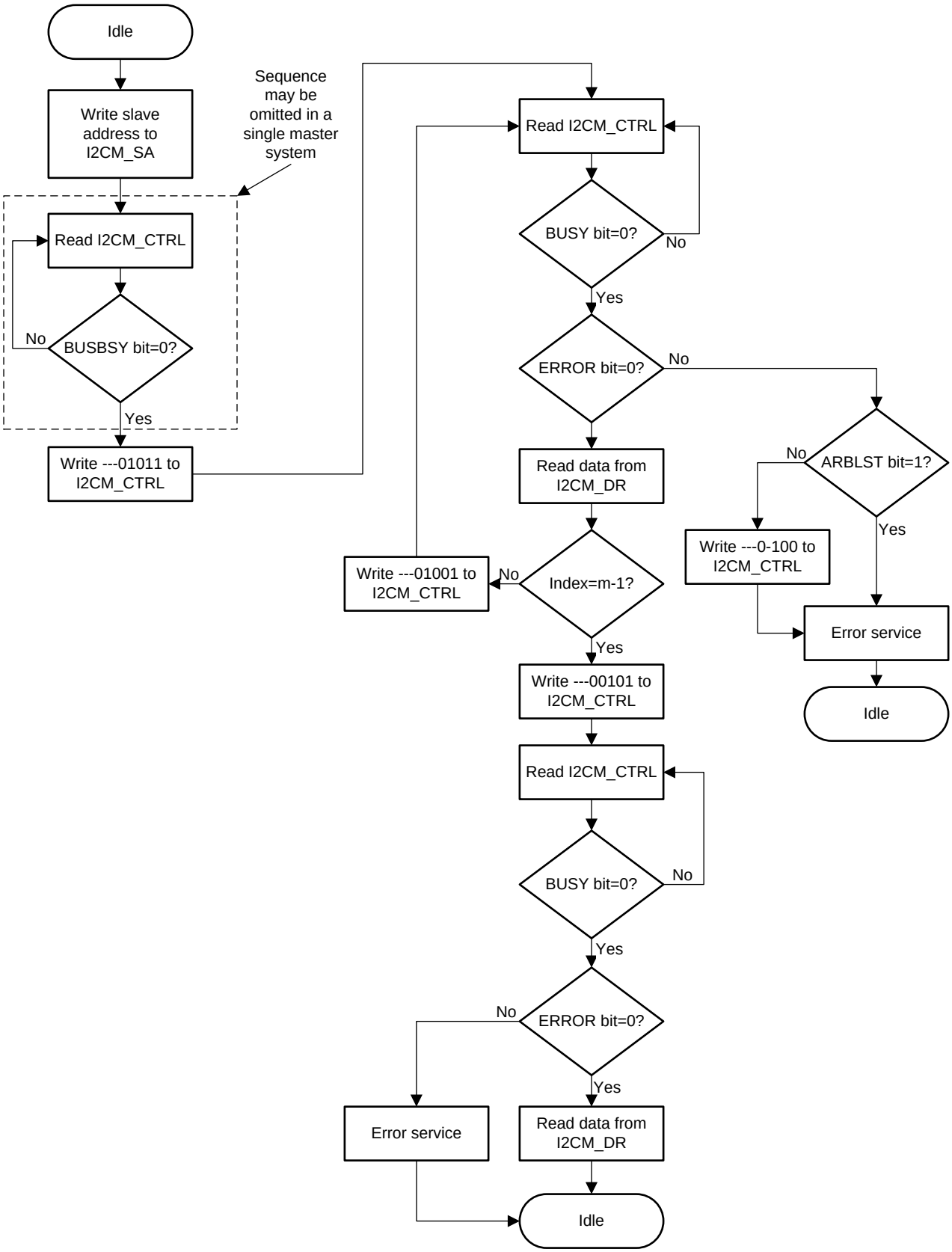


Figure 20-10. Master RECEIVE With Repeated Start Condition

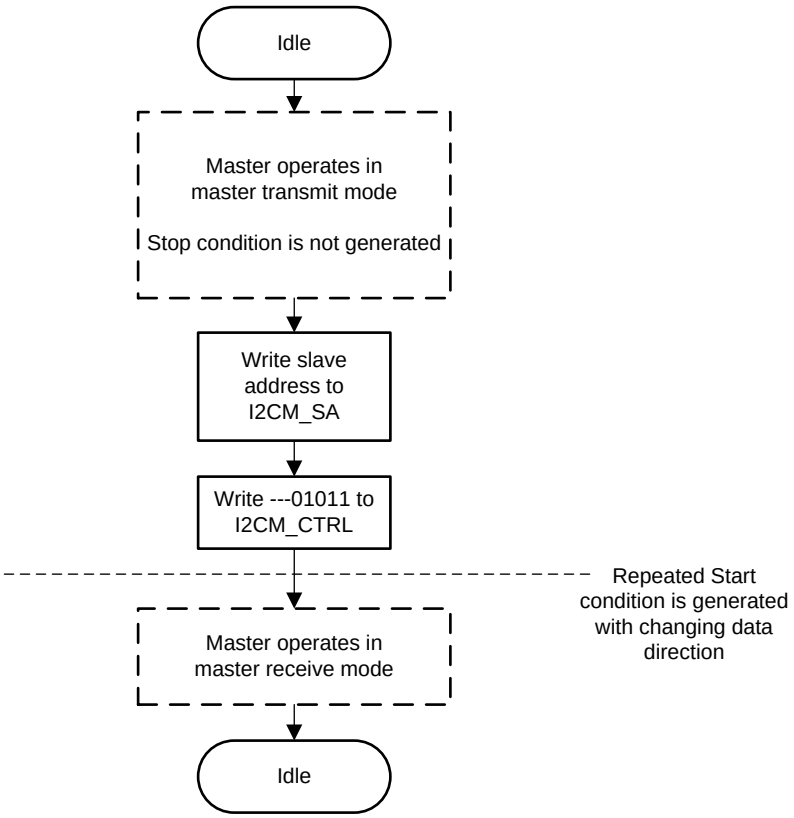


Figure 20-11. Master RECEIVE With Repeated Start After TRANSMIT With Repeated Start Condition

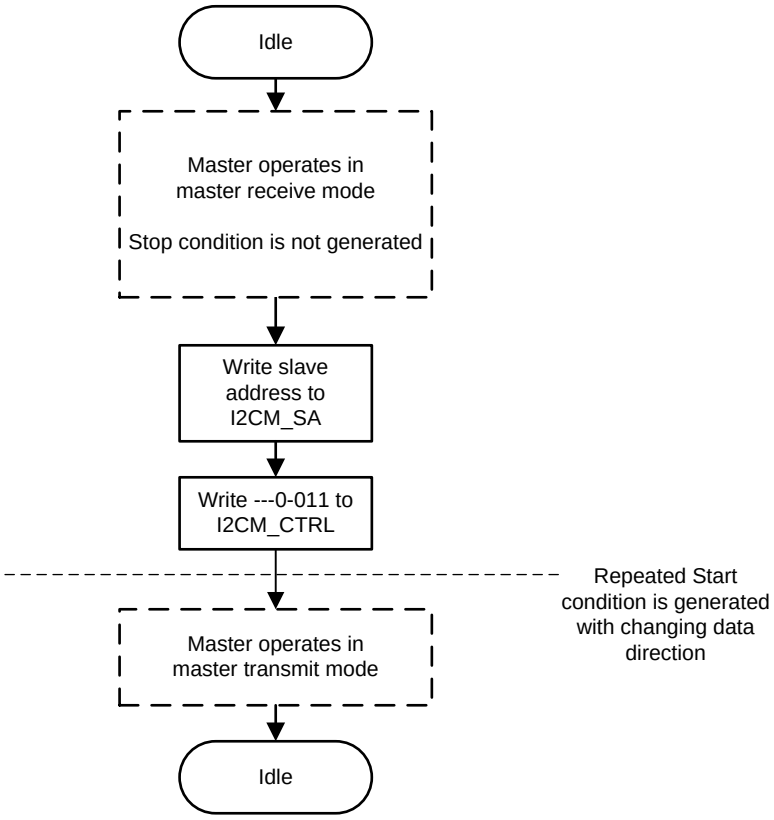


Figure 20-12. Master TRANSMIT With Repeated Start After RECEIVE With Repeated Start Condition

20.3.5.2 I²C Slave Command Sequences

Figure 20-13 shows the command sequence available for the I²C slave.

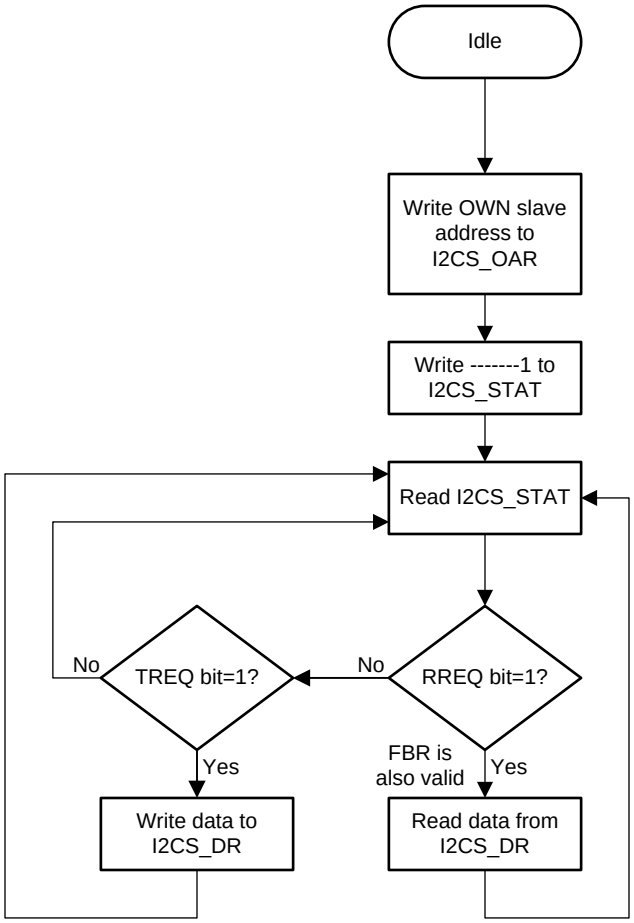


Figure 20-13. Slave Command Sequence

20.4 Initialization and Configuration

The following example shows how to configure the I²C module to transmit a single byte as a master. This assumes the system clock to the I²C module is 16 MHz.

1. Enable the I²C clock using the **SYS_CTRL_RCGCI2C** register in the system control module (SCM) (see SYS_CTRL_RCGCI2C).
2. In the GPIO module, enable the appropriate pins for their alternate function using the **IOC_Pxx_SEL** register (see IOC_PA0_SEL). Also, route the desired port pin to the I2C input signals using registers IOC_I2CMSSDA and IOC_I2CMSSCL
3. Initialize the I²C master by writing the **I2CM_CR** register with a value of 0x0000 0010.
4. Set the desired SCL clock speed of 100 Kbps by writing the **I2CM_TPR** register with the correct value. The value written to the **I2CM_TPR** register represents the number of system clock periods in one SCL clock period. The TPR value is determined by the following equation:

   TPR = (System Clock / (2 × (SCL_LP + SCL_HP) × SCL_CLK)) – 1;

   TPR = (16 MHz / (2 × (6 + 4) × 100000)) – 1;

   TPR = 7

   Write the **I2CM_TPR** register with the value of 0x0000 0007.
5. Specify the slave address of the master and that the next operation is a transmit by writing the **I2CM_SA** register with a value of 0x0000 0076. This sets the slave address to 0x3B.
6. Place data (byte) to be transmitted in the data register by writing the **I2CM_DR** register with the desired data.

7. Initiate a single-byte transmit of the data from master to slave by writing the **I2CM_CTRL** register with a value of 0x0000 0007 (Stop, Start, Run).
8. Wait until the transmission completes by polling the **BUSY** bit of the **I2CM_STAT** register until it has been cleared.
9. Check the **ERROR** bit in the **I2CM_STAT** register to confirm the transmit was acknowledged.

20.5 I²C Registers

20.5.1 I2CM Registers

20.5.1.1 I2CM Registers Mapping Summary

This section provides information on the I2CM module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 20-2. I2CM Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
I2CM_SA	RW	32	0x0000 0000	0x00	0x4002 0000
I2CM_CTRL	WO	32	0x0000 0000	0x04	0x4002 0004
I2CM_STAT	RO	32	0x0000 0000	0x04	0x4002 0004
I2CM_DR	RW	32	0x0000 0000	0x08	0x4002 0008
I2CM_TPR	RW	32	0x0000 0001	0x0C	0x4002 000C
I2CM_IMR	RW	32	0x0000 0000	0x10	0x4002 0010
I2CM_RIS	RO	32	0x0000 0000	0x14	0x4002 0014
I2CM_MIS	RO	32	0x0000 0000	0x18	0x4002 0018
I2CM_ICR	WO	32	0x0000 0000	0x1C	0x4002 001C
I2CM_CR	RW	32	0x0000 0000	0x20	0x4002 0020

20.5.1.2 I2CM Register Descriptions

I2CM_SA

Address offset	0x00		
Physical Address	0x4002 0000	**Instance**	I2C_M0
Description	I2C master slave address This register consists of eight bits, seven address bits (A6-A0), and a receive and send bit, which determines if the next operation is a receive (high) or transmit (low).		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:1	SA	I2C slave address	RW	0x00
0	RS	Receive and send The R/S bit specifies if the next operation is a receive (high) or transmit (low). 0: Transmit 1: Receive	RW	0

I2CM_CTRL

Address offset 0x04

Physical Address 0x4002 0004 **Instance** I2C_M0

Description I2C master control and status
This register accesses status bits when read and control bits when written. When read, the status register indicates the state of the I2C bus controller. When written, the control register configures the I2C controller operation.
The START bit generates the START or REPEATED START condition. The STOP bit determines if the cycle stops at the end of the data cycle or continues on to a repeated START condition. To generate a single transmit cycle, the I2C master slave address (I2CMSA) register is written with the desired address, the R/S bit is cleared, and this register is written with ACK = X (0 or 1), STOP = 1, START = 1, and RUN = 1 to perform the operation and stop. When the operation is completed (or aborted due an error), an interrupt becomes active and the data may be read from the I2CMDR register. When the I2C module operates in master receiver mode, the ACK bit is normally set, causing the I2C bus controller to automatically transmit an acknowledge after each byte. This bit must be cleared when the I2C bus controller requires no further data to be transmitted from the slave transmitter.

Type WO

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												ACK	STOP	START	RUN

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	WO	0x000 0000
3	ACK	Data acknowledge enable 0: The received data byte is not acknowledged automatically by the master. 1: The received data byte is acknowledged automatically by the master.	WO	0
2	STOP	Generate STOP 0: The controller does not generate the STOP condition. 1: The controller generates the STOP condition.	WO	0
1	START	Generate START 0: The controller does not generate the START condition. 1: The controller generates the START condition.	WO	0
0	RUN	I2C master enable 0: The master is disabled. 1: The master is enabled to transmit or receive data. When the BUSY bit is set, the other status bits are not valid.	WO	0

I2CM_STAT

Address offset	0x04		
Physical Address	0x4002 0004	**Instance**	I2C_M0
Description	I2C master control and status This register accesses status bits when read and control bits when written. When read, the status register indicates the state of the I2C bus controller. When written, the control register configures the I2C controller operation. The START bit generates the START or REPEATED START condition. The STOP bit determines if the cycle stops at the end of the data cycle or continues on to a repeated START condition. To generate a single transmit cycle, the I2C master slave address (I2CMSA) register is written with the desired address, the R/S bit is cleared, and this register is written with ACK = X (0 or 1), STOP = 1, START = 1, and RUN = 1 to perform the operation and stop. When the operation is completed (or aborted due an error), an interrupt becomes active and the data may be read from the I2CMDR register. When the I2C module operates in master receiver mode, the ACK bit is normally set, causing the I2C bus controller to automatically transmit an acknowledge after each byte. This bit must be cleared when the I2C bus controller requires no further data to be transmitted from the slave transmitter.		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								BUSBSY	IDLE	ARBLST	DATACK	ADRACK	ERROR	BUSY	

Bits	Field Name	Description	Type	Reset
31:7	RESERVED	This bit field is reserved.	RO	0x000 0000
6	BUSBSY	Bus busy 0: The I2C bus is idle. 1: The I2C bus is busy. The bit changes based on the START and STOP conditions.	RO	0
5	IDLE	I2C idle 0: The I2C controller is not idle. 1: The I2C controller is idle.	RO	0
4	ARBLST	Arbitration lost 0: The I2C controller won arbitration. 1: The I2C controller lost arbitration.	RO	0
3	DATACK	Acknowledge data 0: The transmited data was acknowledged. 1: The transmited data was not acknowledged.	RO	0
2	ADRACK	Acknowledge address 0: The transmited address was acknowledged. 1: The transmited address was not acknowledged.	RO	0
1	ERROR	Error 0: No error was detected on the last operation. 1: An error occurred on the last operation.	RO	0
0	BUSY	I2C busy 0: The controller is idle. 1: The controller is busy. When the BUSY bit is set, the other status bits are not valid.	RO	0

I2CM_DR

Address offset	0x08		
Physical Address	0x4002 0008	**Instance**	I2C_M0
Description	I2C master data This register contains the data to be transmitted when in the master transmit state and the data received when in the master receive state.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	DATA	Data transferred Data transferred during transaction	RW	0x00

I2CM_TPR

Address offset	0x0C		
Physical Address	0x4002 000C	**Instance**	I2C_M0
Description	I2C master timer period This register specifies the period of the SCL clock.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:7	RESERVED	This bit field is reserved.	RO	0x000 0000
6:0	TPR	SCL clock period This field specifies the period of the SCL clock. SCL_PRD = 2 * (1+TPR)*(SCL_LP + SCL_HP)*CLK_PRD where: SCL_PRD is the SCL line period (I2C clock). TPR is the timer period register value (range of 1 to 127) SCL_LP is the SCL low period (fixed at 6). SCL_HP is the SCL high period (fixed at 4). CLK_PRD is the system clock period in ns.	RW	0x01

I2CM_IMR

Address offset	0x10		
Physical Address	0x4002 0010	**Instance**	I2C_M0
Description	I2C master interrupt mask This register controls whether a raw interrupt is promoted to a controller interrupt.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	RO	0x0000 0000
0	IM	Interrupt mask 1: The master interrupt is sent to the interrupt controller when the RIS bit in the I2CMRIS register is set. 0: The RIS interrupt is suppressed and not sent to the interrupt controller.	RW	0

I2CM_RIS

Address offset	0x14		
Physical Address	0x4002 0014	**Instance**	I2C_M0
Description	I2C master raw interrupt status This register specifies whether an interrupt is pending.		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	RO	0x0000 0000
0	RIS	Raw interrupt status 1: A master interrupt is pending. 0: No interrupt This bit is cleared by writing 1 to the IC bit in the I2CMICR register.	RO	0

I2CM_MIS

Address offset	0x18		
Physical Address	0x4002 0018	**Instance**	I2C_M0
Description	I2C master masked interrupt status This register specifies whether an interrupt was signaled.		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	RO	0x0000 0000
0	MIS	Masked interrupt status 1: An unmasked master interrupt is pending. 0: An interrupt has not occurred or is masked. This bit is cleared by writing 1 to the IC bit in the I2CMICR register.	RO	0

I2CM_ICR

Address offset	0x1C		
Physical Address	0x4002 001C	**Instance**	I2C_M0
Description	I2C master interrupt clear This register clears the raw and masked interrupts.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	WO	0x0000 0000
0	IC	Interrupt clear Writing 1 to this bit clears the RIS bit in the I2CMRIS register and the MIS bit in the I2CMMIS register. Reading this register returns no meaningful data.	WO	0

I2CM_CR

Address offset	0x20		
Physical Address	0x4002 0020	**Instance**	I2C_M0
Description	I2C master configuration This register configures the mode (master or slave) and sets the interface for test mode loopback.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								SFE	MFE	RESERVED					LPBK

Bits	Field Name	Description	Type	Reset
31:6	RESERVED	This bit field is reserved.	RO	0x000 0000
5	SFE	I2C slave function enable 1: Slave mode is enabled. 0: Slave mode is disabled.	RW	0
4	MFE	I2C master function enable 1: Master mode is enabled. 0: Master mode is disabled.	RW	0
3:1	RESERVED	This bit field is reserved.	RO	0x0
0	LPBK	I2C loopback 1: The controller in a test mode loopback configuration. 0: Normal operation	RW	0

20.5.2 I2CS Registers

20.5.2.1 I2CS Registers Mapping Summary

This section provides information on the I2CS module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 20-3. I2CS Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
I2CS_OAR	RW	32	0x0000 0000	0x00	0x4002 0800
I2CS_STAT	RO	32	0x0000 0000	0x04	0x4002 0804
I2CS_CTRL	WO	32	0x0000 0000	0x04	0x4002 0804
I2CS_DR	RW	32	0x0000 0000	0x08	0x4002 0808
I2CS_IMR	RW	32	0x0000 0000	0x0C	0x4002 080C
I2CS_RIS	RO	32	0x0000 0000	0x10	0x4002 0810
I2CS_MIS	RO	32	0x0000 0000	0x14	0x4002 0814
I2CS_ICR	WO	32	0x0000 0000	0x18	0x4002 0818

20.5.2.2 I2CS Register Descriptions

I2CS_OAR

Address offset	0x00		
Physical Address	0x4002 0800	**Instance**	I2C_S0
Description	I2C slave own address This register consists of seven address bits that identify the CC2538 I2C device on the I2C bus.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:7	RESERVED	This bit field is reserved.	RO	0x000 0000
6:0	OAR	I2C slave own address This field specifies bits A6 through A0 of the slave address.	RW	0x00

I2CS_STAT

Address offset 0x04

Physical Address 0x4002 0804 **Instance** I2C_S0

Description I2C slave control and status
This register functions as a control register when written, and a status register when read.

Type RO

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																													FBR	TREQ	RREQ

Bits	Field Name	Description	Type	Reset
31:3	RESERVED	This bit field is reserved.	RO	0x0000 0000
2	FBR	First byte received 1: The first byte following the slave's own address has been received. 0: The first byte has not been received. This bit is only valid when the RREQ bit is set and is automatically cleared when data has been read from the I2CSDR register. Note: This bit is not used for slave transmit operations.	RO	0
1	TREQ	Transmit request 1: The I2C controller has been addressed as a slave transmitter and is using clock stretching to delay the master until data has been written to the I2CSDR register. 0: No outstanding transmit request.	RO	0
0	RREQ	Receive request 1: The I2C controller has outstanding receive data from the I2C master and is using clock stretching to delay the master until data has been read from the I2CSDR register. 0: No outstanding receive data	RO	0

I2CS_CTRL

Address offset 0x04

Physical Address 0x4002 0804 **Instance** I2C_S0

Description I2C slave control and status
This register functions as a control register when written, and a status register when read.

Type WO

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

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	WO	0x0000 0000
0	DA	Device active 0: Disables the I2C slave operation 1: Enables the I2C slave operation	WO	0

I2CS_DR

Address offset	0x08		
Physical Address	0x4002 0808	**Instance**	I2C_S0
Description	I2C slave data This register contains the data to be transmitted when in the slave transmit state, and the data received when in the slave receive state.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	DATA	Data for transfer This field contains the data for transfer during a slave receive or transmit operation.	RW	0x00

I2CS_IMR

Address offset	0x0C		
Physical Address	0x4002 080C	**Instance**	I2C_S0
Description	I2C slave interrupt mask This register controls whether a raw interrupt is promoted to a controller interrupt.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																													STOPIM	STARTIM	DATAIM

Bits	Field Name	Description	Type	Reset
31:3	RESERVED	This bit field is reserved.	RO	0x0000 0000
2	STOPIM	Stop condition interrupt mask 1: The STOP condition interrupt is sent to the interrupt controller when the STOPRIS bit in the I2CSRIS register is set. 0: The STOPRIS interrupt is supressed and not sent to the interrupt controller.	RO	0
1	STARTIM	Start condition interrupt mask 1: The START condition interrupt is sent to the interrupt controller when the STARTRIS bit in the I2CSRIS register is set. 0: The STARTRIS interrupt is supressed and not sent to the interrupt controller.	RO	0
0	DATAIM	Data interrupt mask 1: The data received or data requested interrupt is sent to the interrupt controller when the DATARIS bit in the I2CSRIS register is set. 0: The DATARIS interrupt is surpressed and not sent to the interrupt controller.	RW	0

I2CS_RIS

Address offset	0x10		
Physical Address	0x4002 0810	**Instance**	I2C_S0
Description	I2C slave raw interrupt status This register specifies whether an interrupt is pending.		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																													STOPRIS	STARTRIS	DATARIS

Bits	Field Name	Description	Type	Reset
31:3	RESERVED	This bit field is reserved.	RO	0x0000 0000
2	STOPRIS	Stop condition raw interrupt status 1: A STOP condition interrupt is pending. 0: No interrupt This bit is cleared by writing 1 to the STOPIC bit in the I2CSICR register.	RO	0
1	STARTRIS	Start condition raw interrupt status 1: A START condition interrupt is pending. 0: No interrupt This bit is cleared by writing 1 to the STARTIC bit in the I2CSICR register.	RO	0
0	DATARIS	Data raw interrupt status 1: A data received or data requested interrupt is pending. 0: No interrupt This bit is cleared by writing 1 to the DATAIC bit in the I2CSICR register.	RO	0

I2CS_MIS

Address offset	0x14		
Physical Address	0x4002 0814	**Instance**	I2C_S0
Description	I2C slave masked interrupt status This register specifies whether an interrupt was signaled.		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																													STOPMIS	STARTMIS	DATAMIS

Bits	Field Name	Description	Type	Reset
31:3	RESERVED	This bit field is reserved.	RO	0x0000 0000
2	STOPMIS	Stop condition masked interrupt status 1: An unmasked STOP condition interrupt is pending. 0: An interrupt has not occurred or is masked. This bit is cleared by writing 1 to the STOPIC bit in the I2CSICR register.	RO	0
1	STARTMIS	Start condition masked interrupt status 1: An unmasked START condition interrupt is pending. 0: An interrupt has not occurred or is masked. This bit is cleared by writing 1 to the STARTIC bit in the I2CSICR register.	RO	0
0	DATAMIS	Data masked interrupt status 1: An unmasked data received or data requested interrupt is pending. 0: An interrupt has not occurred or is masked. This bit is cleared by writing 1 to the DATAIC bit in the I2CSICR register.	RO	0

I2CS_ICR

Address offset	0x18		
Physical Address	0x4002 0818	**Instance**	I2C_S0
Description	I2C slave interrupt clear This register clears the raw interrupt. A read of this register returns no meaningful data.		
Type	WO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																													STOPIC	STARTIC	DATAIC

Bits	Field Name	Description	Type	Reset
31:3	RESERVED	This bit field is reserved.	WO	0x0000 0000
2	STOPIC	Stop condition interrupt clear Writing 1 to this bit clears the STOPRIS bit in the I2CSRIS register and the STOPMIS bit in the I2CSMIS register. A read of this register returns no meaningful data.	WO	0
1	STARTIC	Start condition interrupt vlear Writing 1 to this bit clears the STARTRIS bit in the I2CSRIS register and the STARTMIS bit in the I2CSMIS register. A read of this register returns no meaningful data.	WO	0
0	DATAIC	Data interrupt clear Writing 1 to this bit clears the DATARIS bit in the I2CSRIS register and the DATAMIS bit in the I2CSMIS register. A read of this register returns no meaningful data.	WO	0

USB Controller

This section focuses on describing the function of the USB controller, and it is assumed that the reader has a good understanding of USB and is familiar with the terms and concepts used (for details, see the Universal Serial Bus Specification Rev. 2.0).

Standard USB nomenclature is used regarding IN and OUT; that is, IN is always into the host (PC) and OUT is out of the host.

Topic **Page**

21.1 USB Introduction

Appropriate response to USB interrupts and loading and unloading of packets into and from endpoint FIFOs is the responsibility of the firmware. The firmware must reply correctly to all standard requests from the USB host and work according to the protocol implemented in the driver on the PC.

The USB controller has the following features:

- Complies with USB 2.0 Specification and earlier, and supports the full-speed (12 Mbps) USB device.
- Five IN and five OUT endpoints for bulk, interrupt or isochronous transfers, in addition to endpoint 0 for control transfers.
- 1-KB SRAM FIFO available for storing USB packets.
- Support for remote wake up and packet sizes between 8 and 256 bytes with restrictions specified in section 21.7.1.
- Support for double-buffering of USB packets.
- Supports suspend and resume signaling.

Figure 21-1 shows a block diagram of the USB controller. The USB PHY is the physical interface with input and output drivers. The USB SIE is the serial-interface engine, which controls the packet transfer to and from the endpoints. The USB controller is connected to the rest of the system through the AHB matrix and is mapped into system memory.

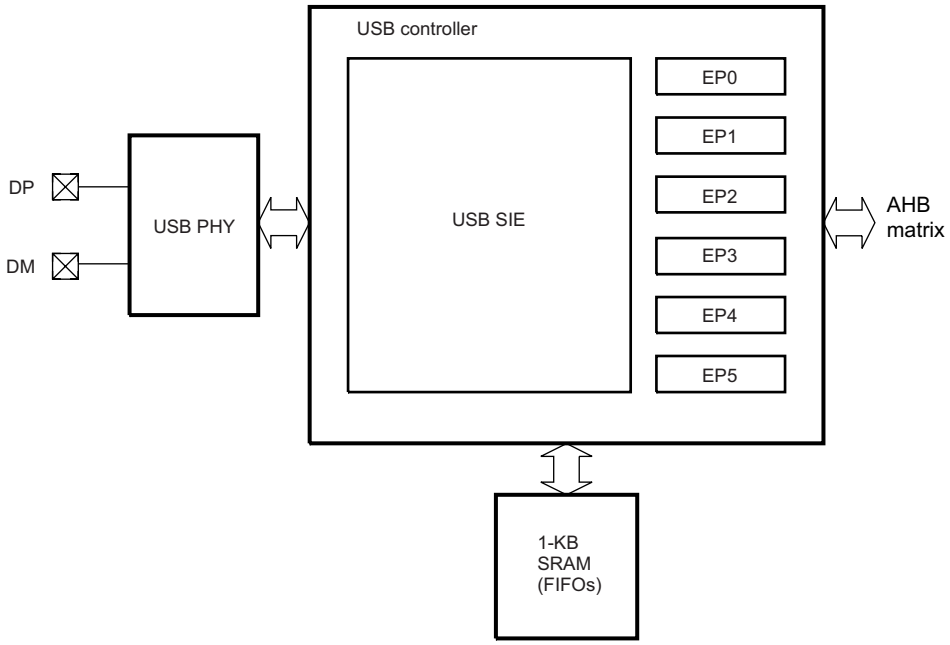


Figure 21-1. USB Controller Block Diagram

21.2 USB Enable

The USB is enabled by setting the **USBEN** bit in the **USB_CTRL** register to 1. Setting the **USBEN** bit in the **USB_CTRL** register to 0 resets the USB controller.

21.3 48-MHz USB PLL

The 48-MHz internal USB phase-locked loop (PLL) must be powered up and stable for the USB controller to operate correctly. It is important that the crystal oscillator is selected as source and is stable before the USB PLL is enabled. The USB PLL is enabled by setting the **PLLEN** bit in the **USB_CTRL** register and waiting for the **PLLLOCKED** bit in the **USB_CTRL** register (status flag) to go high. When the PLL has locked, it is safe to use the USB controller.

NOTE: The PLL must be disabled before exiting active mode and re-enabled after entering active mode.

21.4 USB Interrupts

Three interrupt flag registers have associated interrupt-enable mask registers.

Table 21-1. USB Interrupt Flags and Associated Interrupt-Enable Mask Registers

Interrupt Flag	Description	Associated Interrupt-Enable Mask Register
USB_CIF	Contains interrupt flags for common USB interrupts	**USB_CIE**
USB_IIF	Contains interrupt flags for endpoint 0 and all the IN endpoints	**USB_IIE**
USB_OIF	Contains interrupt flags for all OUT endpoints	**USB_OIE**

The USB controller uses interrupt 140 (44 is an alternate) for USB interrupts. To generate an interrupt request, the associated bit in **ENn** must be set to 1 together with the desired interrupt enable bits from the **USB_CIE**, **USB_IIE**, and **USB_OIE** registers. When an interrupt request is generated, the CPU starts executing the interrupt service routine (ISR) if there are no higher-priority interrupts pending. The ISR should read all the interrupt flag registers and take action depending on the status of the flags. The interrupt flag registers are cleared when they are read, thus saving the status of the individual interrupt flags in memory (typically in a local variable on the stack) and allowing multiple accesses to them.

Figure 21-2 is a flow chart for the USB interrupt service routine.

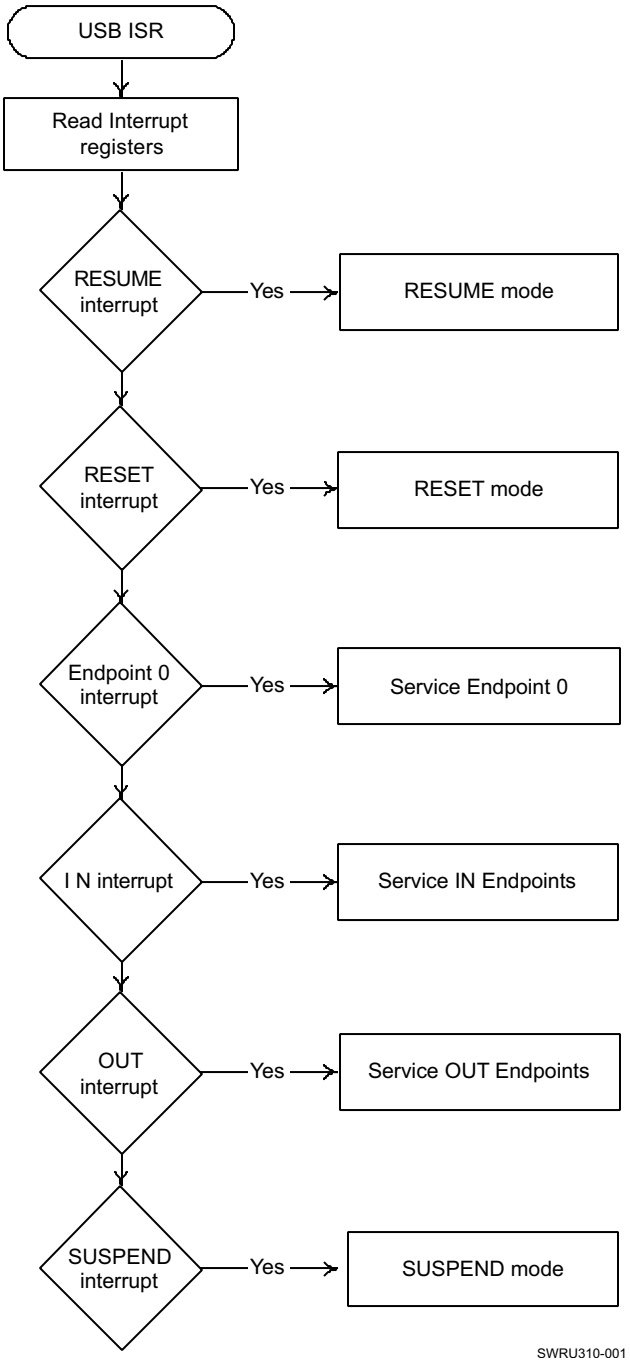


Figure 21-2. USB Interrupt Service Routine

21.5 USB Reset

When reset signaling is detected on the bus, the **RSTIF** bit in the **USB_CIF** register is asserted. Firmware should take appropriate action when a USB reset occurs. A USB reset should place the device in the default state, where it only responds to address 0 (the default address). One or more resets normally occur during the enumeration phase, immediately after the USB cable is connected.

The USB controller performs the following actions when a USB reset occurs:

- Sets the **USB_ADDR** register to 0.

- Sets the **USB_INDEX** register to 0.
- Flushes all endpoint FIFOs.
- Clears **USB_MAXI**, **USB_CS0**, **USB_CSIL**, **USB_CSIH**, **USB_MAXO**, **USB_CSOL**, **USB_CSOH**, **USB_CNT0**, **USB_CNTL**, and **USB_CNTH** registers.
- Enables all interrupts, except SOF and suspend.
- Generates an interrupt request (if the **RSTIE** bit in the **USB_CIE** register are set to 1).
  Firmware should close all pipes and wait for a new enumeration phase when USB reset is detected.

21.6 USB Index Register

Index is a 4-bit register that determines which endpoint control/status registers are accessed at addresses 10h to 17h. Each IN endpoint and each OUT endpoint have their own set of control/status registers. Only one set of IN control/status and one set of OUT control/status registers appear in the memory map at any one time. Before accessing an endpoint's control/status registers, the endpoint number should be written to the Index register to ensure that the correct control/status registers appear in the memory map.

21.7 USB Suspend and Resume

The USB controller asserts the **USB_CIF.SUSPENDIF** bit and enters suspend mode when the USB has been continuously idle for 3 ms, provided that the **USB_POW.SUSPENDEN** bit is set.

While in suspend mode, only limited current is sourced from the USB host or hub. For details, see the USB 2.0 Specification. To meet the suspend-current requirement, the device should enter PM1 when suspend is detected. The device should not enter PM2 or PM3, because this resets the USB controller. Before entering PM1, the 48-MHz USB PLL must turn off; this is done by resetting **USB_CTRL.PLLEN** bit and waiting for **USB_CTRL.PLLLOCKED** bit to clear.

Any valid non idle signaling on the USB will wake up the system (if **SYS_CTRL_IWE.USB_IW** is set) and generate a resume interrupt if the USB resume interrupt is enabled. The USB resume event will set **GPIO_USB_IRQ_ACK.USBACK** bit and **USB_CIF.RESUMEIF** bit.

Note that the only USB register that can be accessed before the USB PLL is enabled is the **USB_CTRL** register. To read **USB_CIF** when the chip wakes up, re-enable the PLL by setting the **USB_CTRL.PLLEN** bit and wait until the **USB_CTRL.PLLLOCKED** bit is set.

The USB resume interrupt enable is shared with the interrupt enable port D pin 7. This is the **GPIO_PI_IEN.PD7IEN** bit. Consequently, a USB resume interrupt generates a Port D interrupt.

A USB reset also wakes up the system from suspend. A USB resume interrupt request is generated if the interrupt is enabled, but **USB_CIF.RSTIF** is set instead of the **USB_CIF.RESUMEIF**.

21.7.1 USB Suspend and Resume Procedure

The procedures described below must be performed to be able to enter PM1 during USB suspend, and subsequently wake up when resume signaling is detected. The resume mechanism detects a falling edge on the USB D+ data line to generate an internal rising edge resume interrupt request, which is logically OR'ed with the GPIO port D pin 7 interrupt request.

Upon detection of the USB suspend interrupt (**USB_CIF.SUSPENDIF** being set), the software should configure the device for a USB resume by performing the following sequence:

- Disable the USB PLL
- Set the **GPIO_USB_IRQ_ACK.USBACK** bit (clears any pending events)
- Set the **GPIO_IRQ_DETECT_ACK.PDIACK7** bit (clears any pending events)
- Set **GPIO_PI_IEN.PDIEN7** bit
- Set **SYS_CTRL_IWE.USB_IWE** bit (enables wake up on USB)

- Clear the GPIO D interrupt pending flag in NVIC
- Enable the GPIO D interrupt in NVIC
- Enter PM1

The GPIO port D ISR must perform the following operations when the resume interrupt occurs. The following code assumes no other Port D interrupts are in use. If other Port D events are active, the resume ISR would need to check for and handle these appropriately:

- If **GPIO_USB_IRQ_ACK** is non-zero (USB resume event has occurred)
  - Set **GPIO_USB_IRQ_ACK.USBACK** bit
  - Disable the GPIO D interrupt in NVIC
  - Clear **SYS_CTRL_IWE.USB_IWE** bit
- Clear the GPIO D interrupt pending flag in NVIC

NOTE: If other interrupts can occur during USB suspend, for example to maintain a radio link, PM1 must be re-entered after servicing these, provided that the USB wake-up interrupt has not occurred in the meantime.

21.8 Endpoint 0

Endpoint 0 (EP0) is a bidirectional control endpoint. During the enumeration phase, all communication is performed across EP0. Before the **USB_ADDR** register has been set to a value other than 0, the USB controller is only able to communicate through EP0. Setting the **USB_ADDR** register to a value from 1 to 127 brings the USB function out of the default state in the enumeration phase and into the address state. All configured endpoints are then available for the application. The EP0 FIFO is only used as either IN or OUT, and double-buffering is not provided for EP0. The maximum packet size for EP0 is fixed at 32 bytes. EP0 is controlled through the **USB_CS0_CSIL** register by setting the **USB_INDEX** register to 0. The **CNT0_CNTL** register contains the number of bytes received.

The software is required to handle all the Standard Device Requests that may be received via Endpoint 0. These are described in Universal Serial Bus Specification, Revision 2.0. The protocol for these device requests involves different numbers and types of transaction per transfer. To accommodate this, the CPU needs to take a state machine approach to command decoding and handling.

The Standard Device Requests are the Zero Data Requests, Write Requests and Read Requests. Following subsections look at the sequence of events that the software must perform to process the different types of device request.

21.8.1 Zero Data Requests

Zero data requests have all their information included in the 8-byte command and require no additional data to be transferred. The sequence of events will begin when the software receives an Endpoint 0 interrupt. The **USB_CS0.OUTPKTRDY** bit will also have been set. The 8-byte command should then be read from the Endpoint 0 FIFO, decoded and acted upon accordingly. The **USB_CS0_CSIL** register should then be written to set the **USB_CS0.CLROUTPKTRDY** (indicating that the command has been read from the FIFO) and to set the **USB_CS0.DATAEND** bit (indicating that no further data is expected for this request).

When the host moves to the status stage of the request, a second Endpoint 0 interrupt will be generated to indicate that the request has completed. No further action is required from the software: the second interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when it has been decoded, the **USB_CS0_CSIL** register should be written to set the **USB_CS0.CLROUTPKTRDY** bit and to set the **USB_CS0.SENDSTALL** bit. When the host moves to the status stage of the request, the USB controller will send a STALL to tell the host that the request was not executed. A second Endpoint 0 interrupt will be generated and the **USB_CS0.SENTSTALL** bit will be set.

If the host sends more data after the **USB_CS0.DATAEND** bit has been set, then the USB controller will send a STALL. An Endpoint 0 interrupt will be generated and the **USB_CS0.SENTSTALL** bit will be set.

21.8.2 Write Requests

Write requests involve an additional packet (or packets) of data being sent from the host after the 8-byte command. The sequence of events will begin when the software receives an Endpoint 0 interrupt. The **USB_CS0.CLROUTPKTRDY** bit will also have been set. The 8-byte command should then be read from the Endpoint 0 FIFO and decoded.

The **USB_CS0_CSIL** register should then be written to set the **USB_CS0.CLROUTPKTRDY** bit (indicating that the command has been read from the FIFO) but in this case the **USB_CS0.DATAEND** bit should not be set (indicating that more data is expected).

When a second Endpoint 0 interrupt is received, the **USB_CS0_CSIL** register should be read to check the endpoint status. The **USB_CS0.OUTPKTRDY** bit should be set to indicate that a data packet has been received. The **USB_CS0.FIFOCNT** register should then be read to determine the size of this data packet. The data packet can then be read from the Endpoint 0 FIFO. If the length of the data associated with the request is greater than the maximum packet size for Endpoint 0, further data packets will be sent. In this case, **USB_CS0_CSIL** should be written to set the **USB_CS0.CLROUTPKTRDY** bit, but the **USB_CS0.DATAEND** bit should not be set.

When all the expected data packets have been received, the **USB_CS0_CSIL** register should be written to set the **USB_CS0.CLROUTPKTRDY** bit and to set the **USB_CS0.DATAEND** bit (indicating that no more data is expected).

When the host moves to the status stage of the request, another Endpoint 0 interrupt will be generated to indicate that the request has completed. No further action is required from the software, the interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized, the **USB_CS0_CSIL** register should be written to set the **USB_CS0.CLROUTPKTRDY** bit and to set the **USB_CS0.SENDSTALL** bit. When the host sends more data, the USB controller will send a STALL to tell the host that the request was not executed. An Endpoint 0 interrupt will be generated and the **USB_CS0.SENTSTALL** bit will be set. If the host sends more data after the **USB_CS0.DATAEND** has been set, then the USB controller will send a STALL. An Endpoint 0 interrupt will be generated and the **USB_CS0.SENTSTALL** bit will be set.

21.8.3 Read Requests

Read requests have a packet (or packets) of data sent from the function to the host after the 8-byte command. The sequence of events will begin when the software receives an Endpoint 0 interrupt. The **USB_CS0.OUTPKTRDY** bit will also have been set. The 8-byte command should then be read from the Endpoint 0 FIFO and decoded. The **USB_CS0_CSIL** register should then be written to set the **USB_CS0.CLROUTPKTRDY** bit (indicating that the command has read from the FIFO).

The data to be sent to the host should then be written to the Endpoint 0 FIFO. If the data to be sent is greater than the maximum packet size for Endpoint 0, only the maximum packet size should be written to the FIFO. The **USB_CS0_CSIL** register should then be written to set the **USB_CS0.INPKTRDY** bit (indicating that there is a packet in the FIFO to be sent). When the packet has been sent to the host, another Endpoint 0 interrupt will be generated and the next data packet can be written to the FIFO.

When the last data packet has been written to the FIFO, the **USB_CS0_CSIL** register should be written to set the **USB_CS0.INPKTRDY** bit and to set the **USB_CS0.DATAEND** bit (indicating that there is no more data after this packet).

When the host moves to the status stage of the request, another Endpoint 0 interrupt will be generated to indicate that the request has completed. No further action is required from the software: the interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when it has been decoded, the **USB_CS0_CSIL** register should be written to set the **USB_CS0.CLROUTPKTRDY** bit and to set the **USB_CS0.SENDSTALL** bit. When the host requests data, the USB controller will send a STALL to tell the host that the request was not executed. An Endpoint 0 interrupt will be generated and the **USB_CS0.SENTSTALL** bit will be set.

If the host requests more data after **USB_CS0.DATAEND** has been set, then the USB controller will send a STALL. An Endpoint 0 interrupt will be generated and the **USB_CS0.SENTSTALL** bit will be set.

21.9 EndPoint 0 Interrupts

The following events can generate an EP0 interrupt request:

- A data packet was received (the **USB_CS0.OUTPKTRDY** bit is set).
- A data packet that was loaded into the EP0 FIFO was sent to the USB host. (The **USB_CS0.INPKTRDY** bit should be set when a new packet is ready to be transferred. Hardware clears this bit when the data packet is sent.)
- An IN transaction was completed (the interrupt is generated during the status stage of the transaction).
- A STALL was sent (the **USB_CS0.SENTSTALL** bit is set).
- A control transfer ends due to a premature end-of-control transfer (the **USB_CS0.SETUPEND** is set).

Any of these events causes assertion of the **EP0IF** bit in the **USB_IIF** register, regardless of the status of the EP0 interrupt mask bit, **EP0IE**, in the **USB_IIE** register. An interrupt request is generated only if **ENn** and the **EP0IE** bit in the **USB_IIE** register are both set to 1.

Whenever the Endpoint 0 service routine is entered, the firmware must first check to see if the current control transfer has been ended due to either a STALL condition or a premature end of control transfer. If the control transfer ends due to a STALL condition, the **USB_CS0.SENTSTALL** bit would be set. If the control transfer ends due to a premature end of control transfer, the **USB_CS0.SETUPEND** bit would be set. In either case, the firmware should abort processing the current control transfer and set the state to IDLE.

The Endpoint 0 control needs three modes IDLE, TX and RX, corresponding to the different phases of the control transfer and the states Endpoint 0 enters for the different phases of the transfer. The default mode on power-up or reset should be IDLE. **USB_CS0.OUTPKTRDY** becoming set when Endpoint 0 is in IDLE state indicates a new device request. Once the device request is unloaded from the FIFO, the USB controller decodes the descriptor to find whether there is a Data phase and, if so, the direction of the Data phase for the control transfer (in order to set the FIFO direction).

Depending on the direction of the Data phase, Endpoint 0 goes into either TX state or RX state. If there is no Data phase, Endpoint 0 remains in IDLE state to accept the next device request.

Figure 21-3 shows the states that Endpoint 0 enters for the different phases of the control transfer.

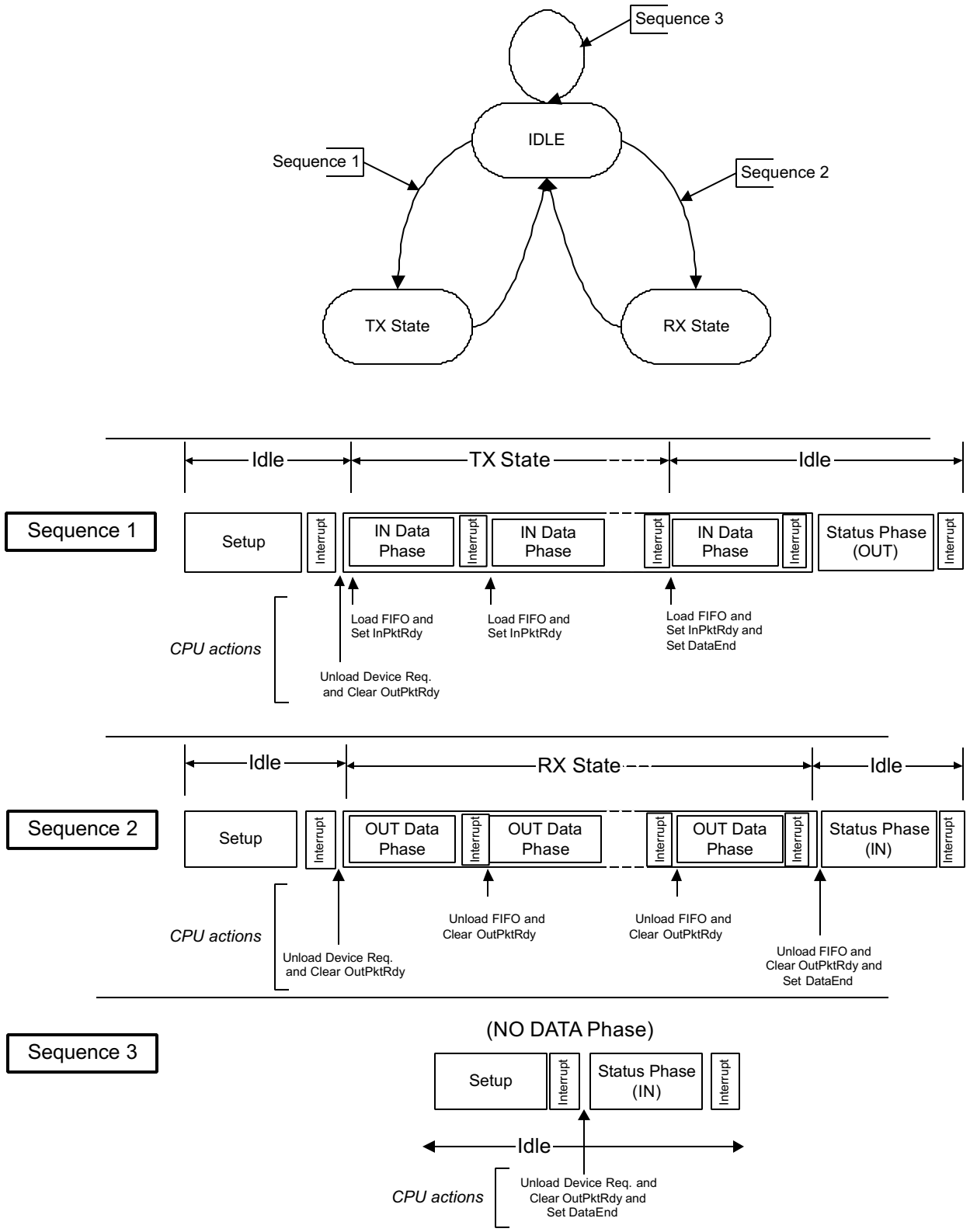


Figure 21-3. Endpoint 0 States

Figure 21-4 is a flow chart for the Endpoint 0 service routine.

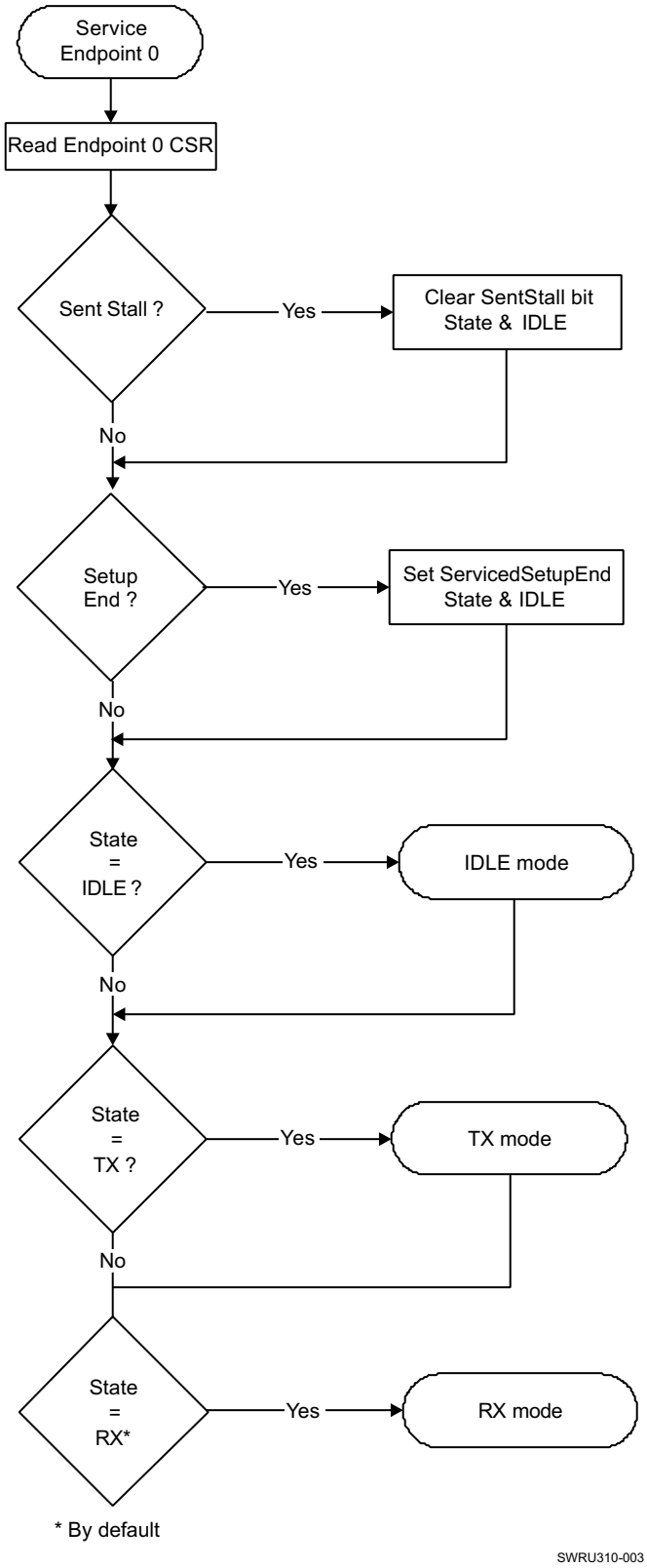


Figure 21-4. Endpoint 0 Service Routine

21.9.1 Error Conditions

When a protocol error occurs, the USB controller sends a STALL handshake. The **USB_CS0.SENTSTALL** bit is set, and an interrupt request is generated if the EP0 interrupt is properly enabled. A protocol error can be any of the following:

- An OUT token is received after the **USB_CS0.DATAEND** bit is set to complete the OUT data stage (the host tries to send more data than expected).
- An IN token is received after the **USB_CS0.DATAEND** bit is set to complete the IN data stage (the host tries to receive more data than expected).
- The USB host tries to send a packet that exceeds the maximum packet size during the OUT data stage.
- The size of the DATA1 packet received during the status stage is not 0.

The firmware can also terminate the current transaction by setting the **USB_CS0.SENDSTALL** bit is set. The USB controller then sends a STALL handshake in response to the next request from the USB host.

If assertion of the **USB_CS0.SENTSTALL** bit causes an EP0 interrupt, de-assertion of this bit should occur, and firmware should consider the transfer as aborted (and consequently free the memory buffers, and so forth).

If EP0 receives an unexpected token during the data stage, the **USB_CS0.SETUPEND** bit register is set, and an EP0 interrupt is generated (if enabled properly). EP0 then switches to the IDLE state. Firmware should then set the **USB_CS0.SETUPEND** bit and abort the current transfer. Asserting the **USB_CS0.OUTPKTRDY** bit indicates that another setup packet that firmware should process was received.

21.9.2 SETUP Transactions (IDLE State)

The control transfer consists of two or three stages of transactions (setup – data – status or setup – status). The first transaction is a setup transaction. A successful setup transaction comprises three sequential packets (a token packet, a data packet, and a handshake packet), where the data field (payload) of the data packet is 8 bytes long and is referred to as the setup packet. In the setup stage of a control transfer, EP0 is in the IDLE state. The USB controller rejects the data packet if the setup packet is not 8 bytes. Also, the USB controller examines the contents of the setup packet to determine whether or not there is a data stage in the control transfer. If there is a data stage, EP0 switches state to TX (IN transaction) or RX (OUT transaction) when the **USB_CS0.CLROUTPKTRDY** bit is set (if the **USB_CS0.DATAEND** bit is reset).

When a packet is received, the **USB_CS0.OUTPKTRDY** bit is set and an interrupt request is generated (EP0 interrupt) if the interrupt is enabled. Firmware should perform the following steps when a setup packet is received:

1. Unload the setup packet from the EP0 FIFO.
2. Examine the contents and perform the appropriate operations.
3. Set the **USB_CS0.CLROUTPKTRDY** bit. This denotes the end of the setup stage. If the control transfer has no data stage, also set the **USB_CS0.DATAEND** bit. If there is no data stage, the USB controller stays in the IDLE state.

Figure 21-5 is a flow chart for the handling of the SETUP phase of the control transfer.

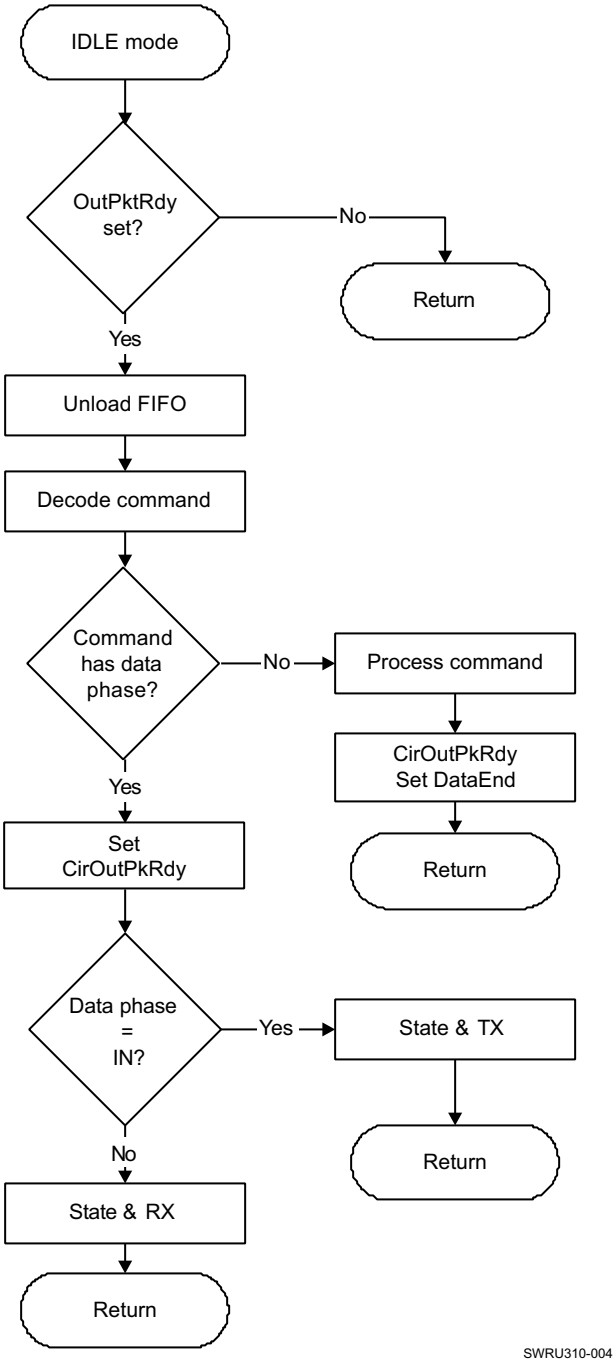


Figure 21-5. SETUP Phase of Control Transfer

Figure 21-6 is a flow chart of control transactions in the SETUP phase.

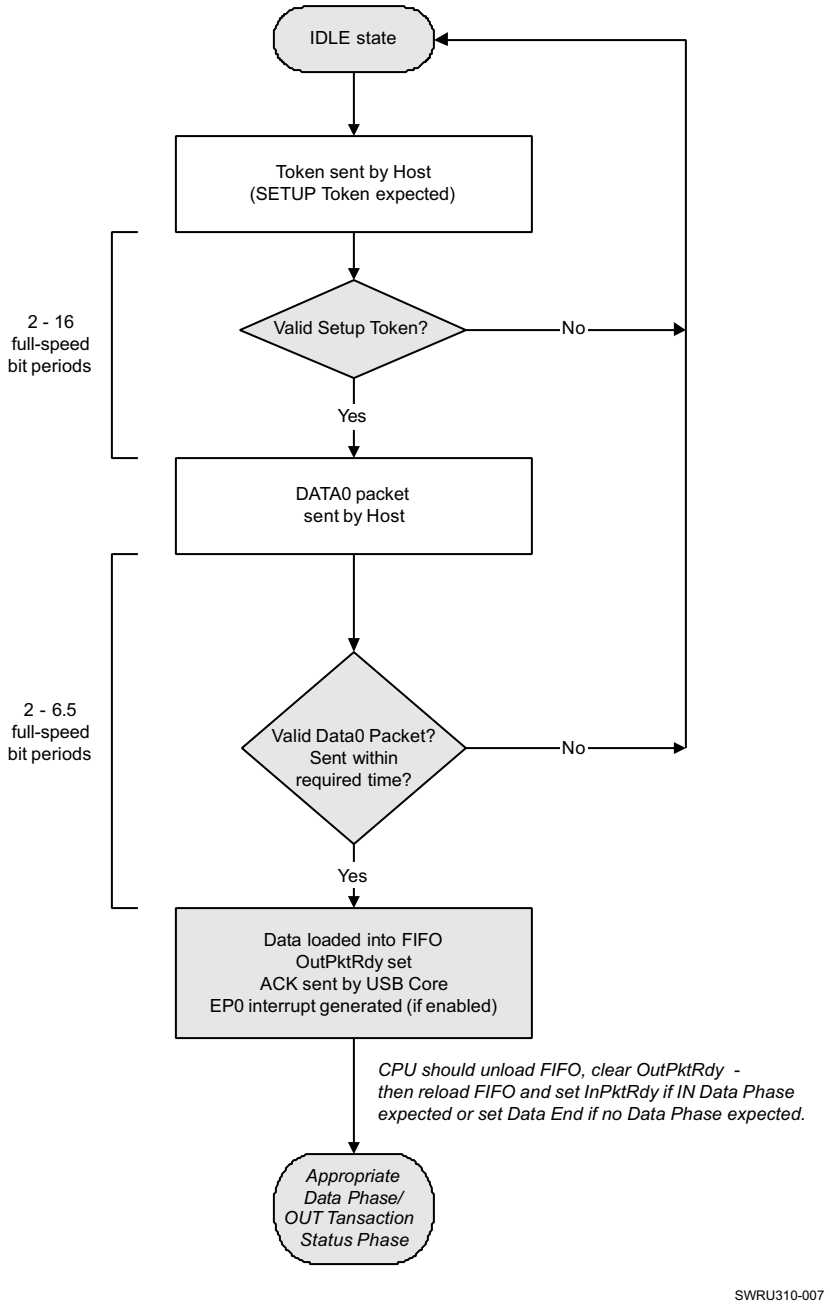


Figure 21-6. SETUP Phase Control Transactions

21.9.3 IN Transactions (TX State)

If the control transfer requires sending data to the host, the setup stage is followed by one or more IN transactions in the data stage. In this case, the USB controller is in the TX state and only accepts IN tokens. A successful IN transaction comprises two or three sequential packets (a token packet, a data packet, and a handshake packet). If more than 32 bytes (maximum packet size) are sent, split the data into a number of 32-byte packets followed by a residual packet. If the number of bytes to send is a multiple of 32, the residual packet is a zero-length data packet, because a packet size less than 32 bytes denotes the end of the transfer.

NOTE: For isochronous transfers there would not be a handshake packet from the host.

Firmware should load the EP0 FIFO with the first data packet and set the **USB_CS0.INPKTRDY** bit as soon as possible after the **USB_CS0.CLROUTPKTRDY** bit is set. The **USB_CS0.INPKTRDY** bit is cleared and an EP0 interrupt is generated when the data packet is sent. Firmware might then load more data packets as necessary. An EP0 interrupt is generated for each packet sent. Firmware must set the **USB_CS0.DATAEND** bit in addition to the **USB_CS0.INPTKRDY** bit when the last data packet is loaded. This starts the status stage of the control transfer.

EP0 switches to the IDLE state when the status stage completes. The status stage may fail if the **USB_CS0.SENDSTALL** bit is set. The **USB_CS0.SENTSTALL** bit is then asserted, and an EP0 interrupt is generated.

If the **USB_CS0.INPKTRDY** bit is not set when receiving an IN token, the USB controller replies with a NAK to indicate that the endpoint is working, but temporarily has no data to send. If the endpoint is in TX state, the interrupt indicates that the core has received an IN token and data from the FIFO has been sent. The firmware must respond to this either by placing more data in the FIFO if the host is still expecting more data or by setting the **USB_CS0.DATAEND** bit to indicate that the data phase is complete.

Three events can cause TX mode to be terminated before the expected amount of data has been sent:

- The host sends an invalid token causing a **USB_CS0.SETUPEND** to set.
- The firmware sends a packet containing less than the maximum packet size for Endpoint 0.
- The firmware sends an empty data packet.

Figure 21-7 is a flow chart for the IN data phase of a control transfer.

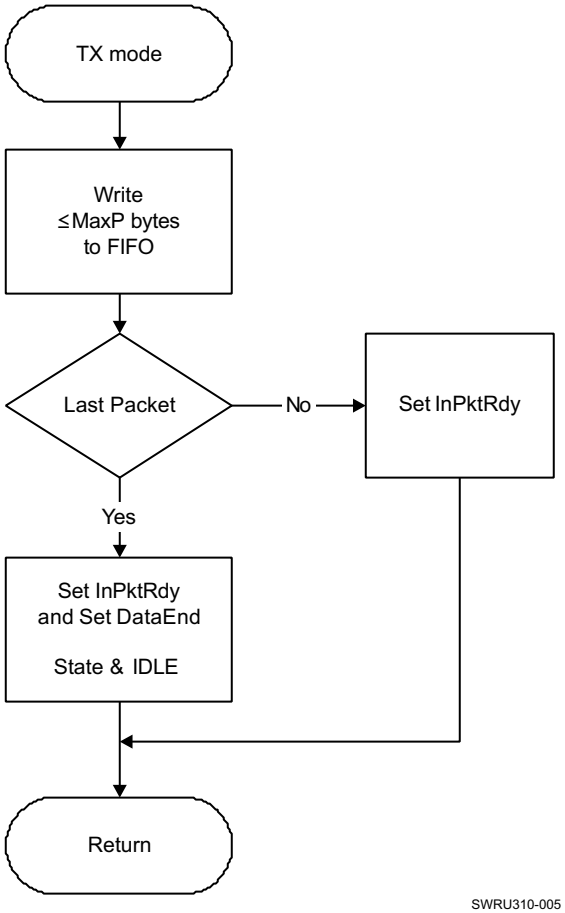


Figure 21-7. IN Data Phase for Control Transfer

Figure 21-8 is a flow chart of control transactions in the IN data phase; Figure 21-9 shows transactions after the status stage.

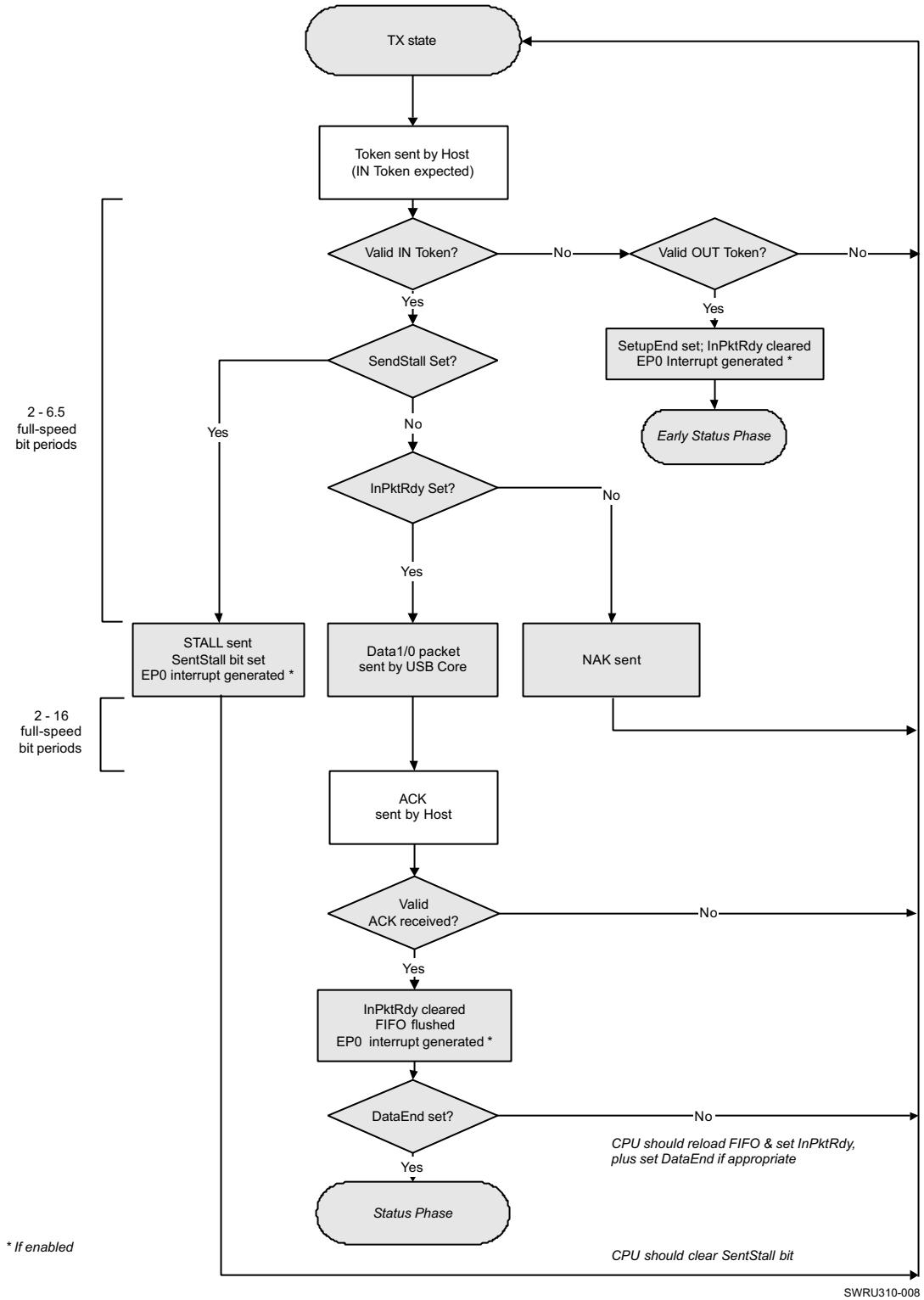


Figure 21-8. IN Phase Control Transactions

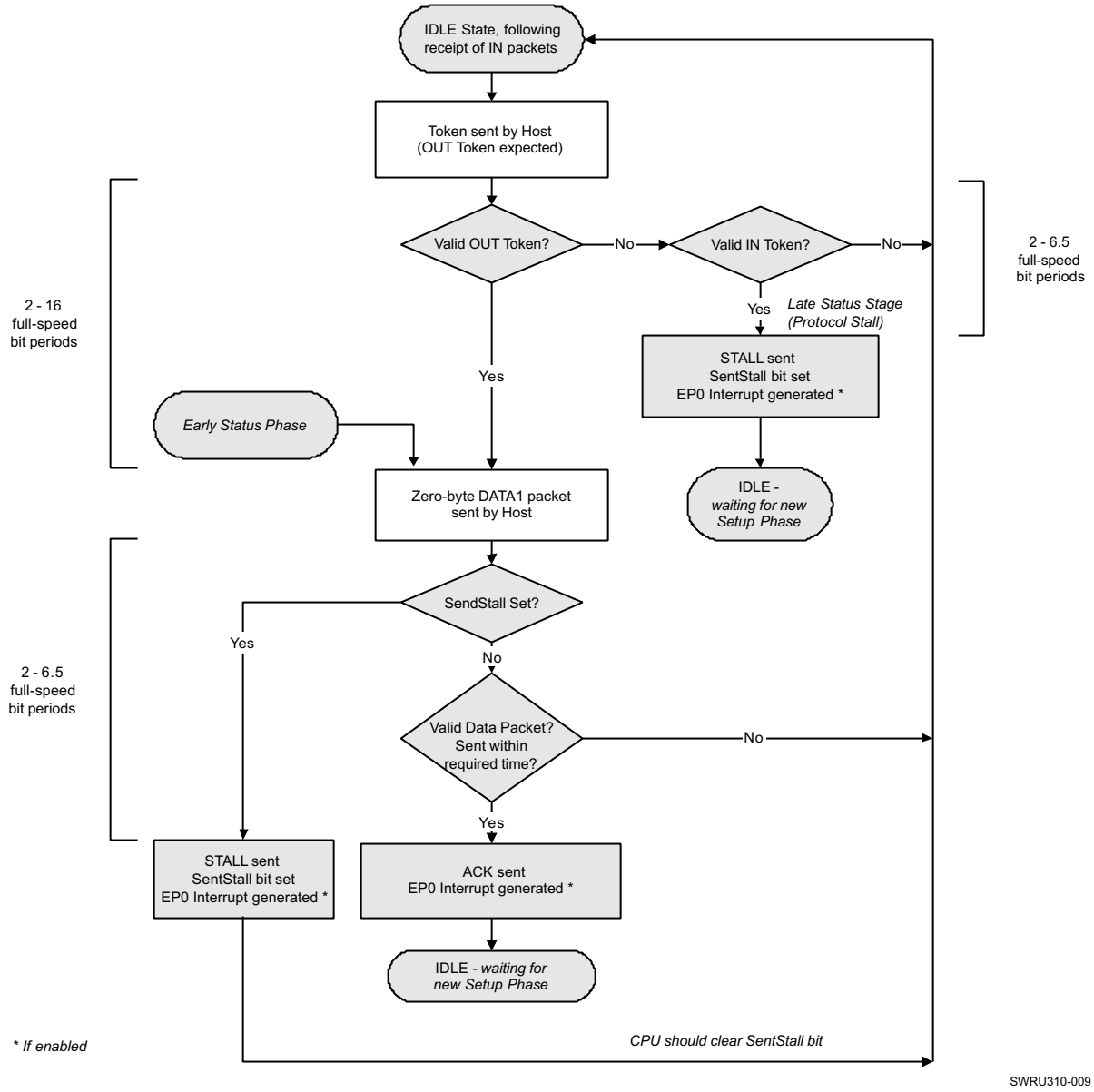


Figure 21-9. Control Transactions Following Status Stage (TX Mode)

21.9.4 OUT Transactions (RX State)

If the control transfer requires receiving data from the host, the setup stage is followed by one or more OUT transactions in the data stage. In this case, the USB controller is in the RX state and only accepts OUT tokens. A successful OUT transaction comprises two or three sequential packets (a token packet, a data packet, and a handshake packet). If more than 32 bytes (maximum packet size) are received, split the data into a number of 32-byte packets followed by a residual packet. If the number of bytes to receive is a multiple of 32, the residual packet is a zero-length data packet, because a data packet size less than 32 bytes denotes the end of the transfer.

NOTE: For isochronous transfers, there is no handshake packet from the device.

The **USB_CS0.OUTPKTRDY** bit is set and an EP0 interrupt is generated when a data packet is received. The firmware should set the **USB_CS0.CLROUTPKTRDY** bit when the data packet is unloaded from the EP0 FIFO. When the last data packet is received (packet size less than 32 bytes) firmware should also set the **USB_CS0.DATAEND** bit. This starts the status stage of the control transfer. The size of the data packet is kept in the **USB_CS0.FIFOCNT** registers. This value is only valid when the **USB_CS0.OUTPKTRDY** bit is set.

EP0 switches to the IDLE state when the status stage completes. The status stage may fail if the DATA1 packet received is not a zero-length data packet or if the **USB_CS0.SENDSTALL** bit is set. The **SENT_STALL** bit in the **USB_CS0_CSIL** register is then asserted and an EP0 interrupt is generated.

If the endpoint is in RX state, the interrupt indicates that a data packet has been received. The firmware must respond by unloading the received data from the FIFO. The firmware must then determine whether it has received all of the expected data. If it has, the firmware should set the **USB_CS0.DATAEND** bit and return Endpoint 0 to IDLE state. If more data is expected, the firmware should set the **USB_CS0.CLROUTPKTRDY** bit to indicate that it has read the data in the FIFO and leave the endpoint in RX state.

Three events can cause RX mode to be terminated before the expected amount of data has been received:

- The host sends an invalid token causing a **USB_CS0.SETUPEND** to set.
- The host sends a packet which contains less than the maximum packet size for Endpoint 0 ·
- The host sends an empty data packet.

Figure 21-10 is a flow chart for the OUT data phase of a control transfer.

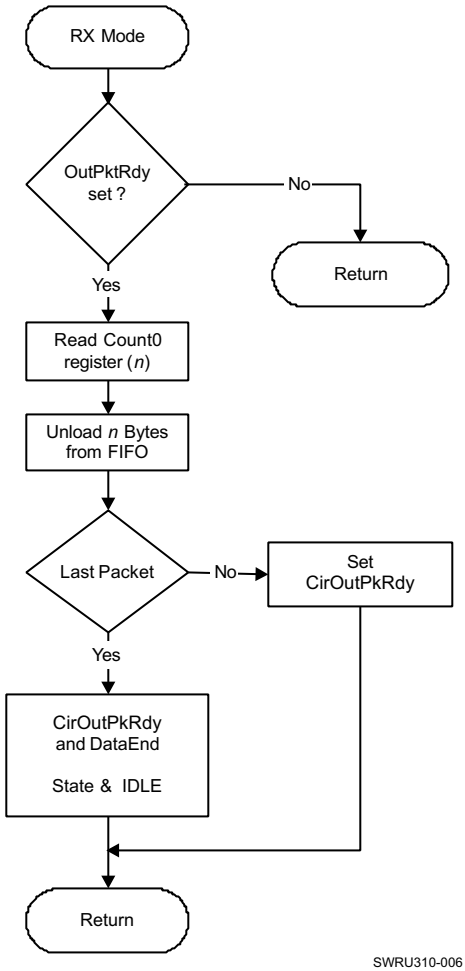


Figure 21-10. OUT Data Phase for Control Transfer

Figure 21-11 is a flow chart of control transactions in the OUT data phase; Figure 21-12 shows transactions after the status stage.

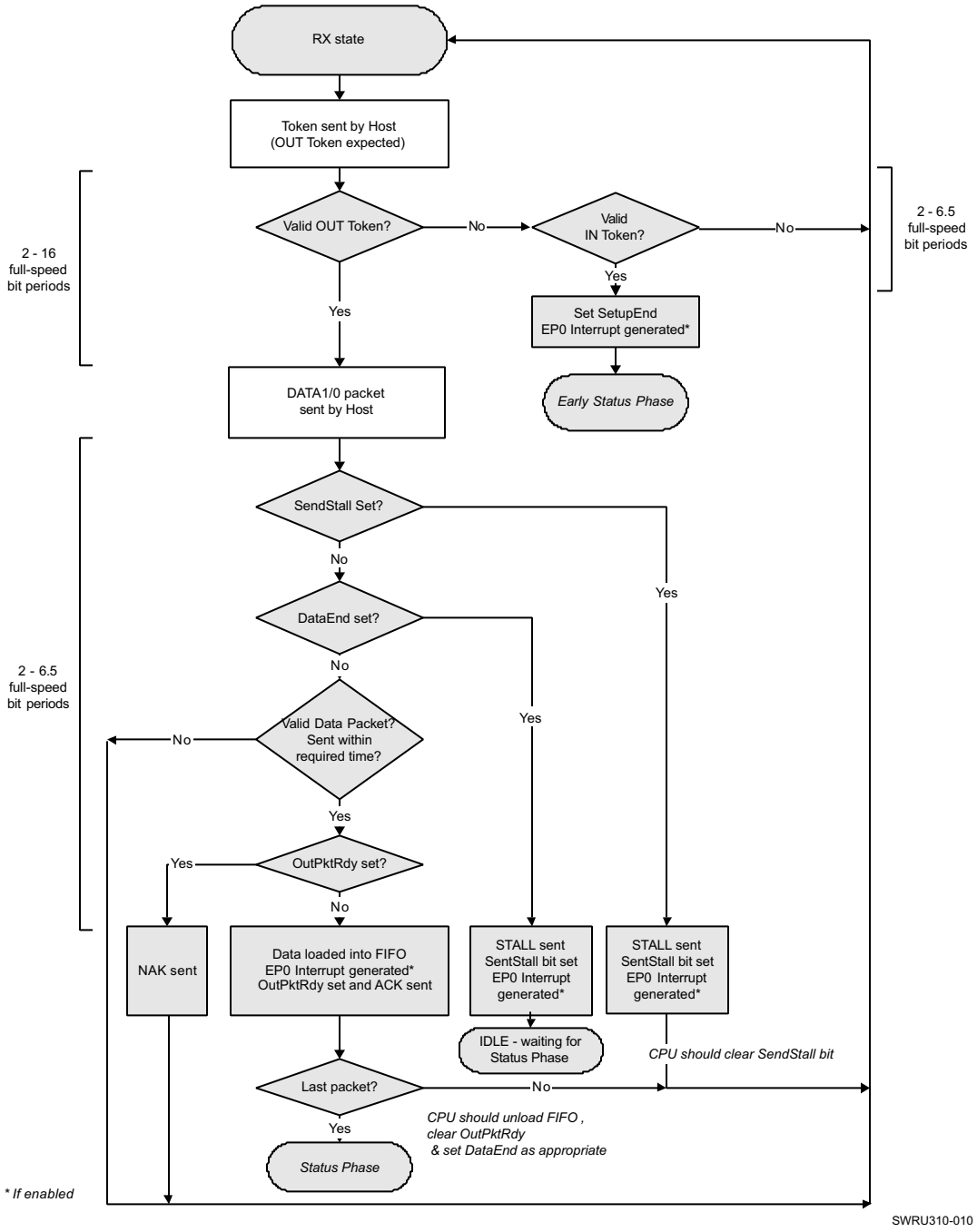


Figure 21-11. OUT Phase Control Transactions

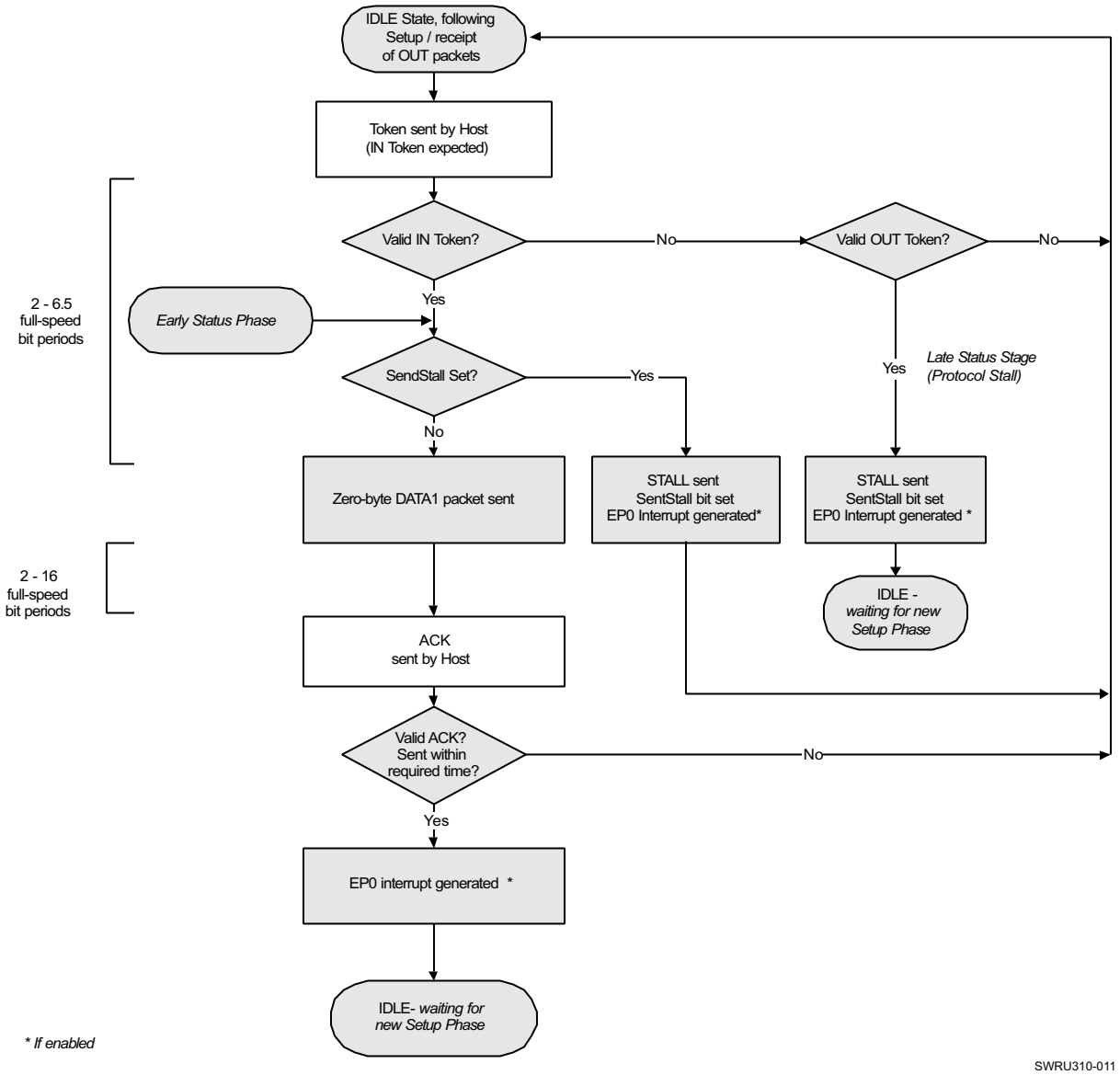


Figure 21-12. Control Transactions Following Status Stage (RX Mode)

21.10 Endpoints 1–5

Use each endpoint as an IN only, an OUT only, or an IN/OUT. For an IN/OUT endpoint, there are basically two endpoints, an IN endpoint and an OUT endpoint associated with the endpoint number. Configuration and control of IN endpoints is performed through the **USB_CS0_CSIL** and **USB_CSIH** registers. Use the **USB_CSOL** and **USB_CSOH** registers to configure and control OUT endpoints. Configure each IN and OUT endpoint as either an isochronous or bulk/interrupt endpoint. The USB controller handles bulk and interrupt endpoints identically, but they have different properties from a firmware perspective.

The **USB_INDEX** register must have the value of the endpoint number before the indexed endpoint registers are accessed.

21.10.1 FIFO Management

Each endpoint has a certain number of FIFO memory bytes available for incoming and outgoing data packets. Table 21-2 lists the FIFO size for endpoints 1–5. The firmware is responsible for setting the **USB_MAXI** and **USB_MAXO** registers correctly for each endpoint to prevent the overwriting of data.

When the IN and OUT endpoints of an endpoint number do not use double-buffering, the sum of **USB_MAXI** and **USB_MAXO** must not exceed the FIFO size for the endpoint. Figure 21-13 a) shows how the IN and OUT FIFO memory for an endpoint is organized with single-buffering.

When the IN or OUT endpoint of an endpoint number uses double-buffering, the sum of **USB_MAXI** and **USB_MAXO** must not exceed half the FIFO size for the endpoint. Figure 21-13 b) shows the IN and OUT FIFO memory for an endpoint that uses double-buffering.

To configure an endpoint as IN-only, set **USB_MAXO** to 0, and to configure an endpoint as OUT-only, set **USB_MAXI** to 0.

For unused endpoints, set **USB_MAXO** and **USB_MAXI** to 0.

Table 21-2. FIFO Sizes for EP1–EP5

EP Number	FIFO Size (in Bytes)
1	32
2	64
3	128
4	256
5	512

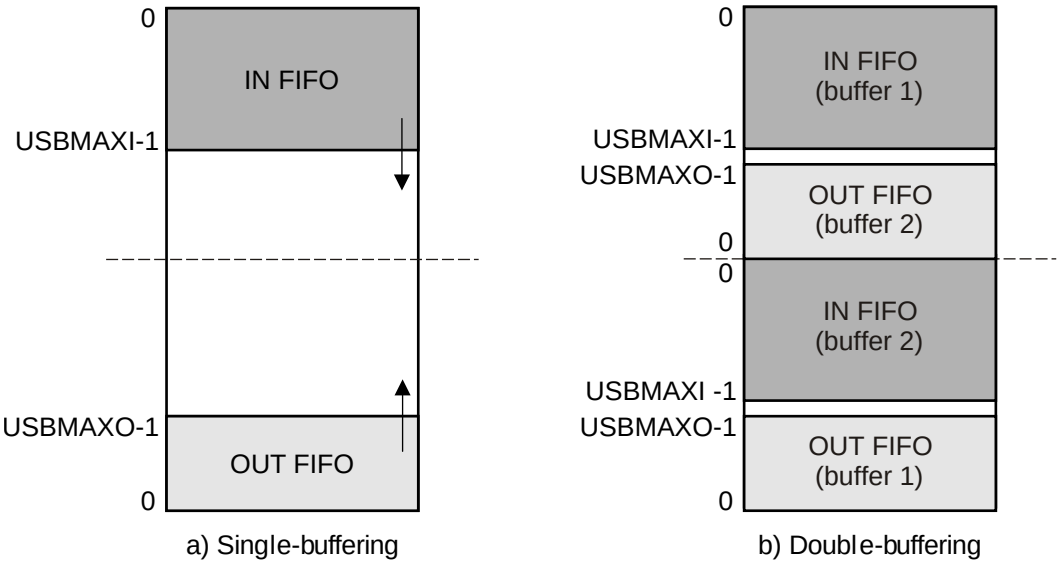


Figure 21-13. IN/OUT FIFOs

21.10.2 Double-Buffering

- For bulk endpoints, double--buffering can be used to increase data transfer rate, by enabling buffering of the next packet while the current is being transmitted.
- For interrupt endpoints, double-buffering is normally not used when there is only one packet per interval and the interval is multiple milliseconds. Using double-buffering in such cases would only result in increased data latency.
- For isochronous endpoints, double-buffering is required to avoid data underrun (since there can be little time between the isochronous packet in one USB frame and the packet in the next).

To enable double-buffering for an IN endpoint, set the **INDBLBUF** bit in the **USB_CSIH** register to 1. To enable double-buffering for an OUT endpoint, set the **OUTDBLBUF** bit in the **USB_CSOH** register to 1.

21.10.3 FIFO Access

The endpoint FIFOs are accessed by reading and writing to the registers USBF0–USBF5. Writing to a register inserts the byte written into the IN FIFO. Reading a register extracts the next byte in the OUT FIFO and returns the value of this byte.

A data packet can be read from the OUT FIFO when the **OUTPKTRDY** bit in the **USB_CSOL** register is set to 1. If enabled, an interrupt is generated when this occurs. The size of the data packet is kept in the **USB_CSIL.FIFOCNTL** register. This value is valid only when the **USB_CSOL.OUTPKTRDY** bit is set. When the data packet is read from the OUT FIFO, clear the **USB_CSOL.OUTPKT_RDY** bit. If double-buffering is enabled, there may be two data packets in the FIFO. If another data packet is ready when the **USB_CSOL.OUTPKTRDY** bit is cleared, this bit is asserted immediately, and an interrupt is generated (if enabled) to signal that a new data packet was received. The **USB_CSOL.FIFOFULL** bit is set when there are two data packets in the OUT FIFO.

The AutoClear feature is supported for OUT endpoints. When enabled, the **USB_CSOL.OUTPKTRDY** bit is cleared automatically when **MAXO** bytes are read from the OUT FIFO. The AutoClear feature is enabled by setting the **USB_CSOH.AUTOCLEAR** bit. The AutoClear feature can reduce the time the data packet occupies the OUT FIFO buffer and is typically used for bulk endpoints.

A complementary AutoSet feature is supported for IN endpoints. When enabled, the **USB_CS0.INPKTRDY** bit is set automatically when **MAXI** bytes are written to the IN FIFO. The AutoSet feature is enabled by setting the **USB_CSIH.AUTIOSET** bit. The AutoSet feature can reduce the overall time needed to send a data packet and is typically used for bulk endpoints.

21.10.4 Endpoint 1–5 Interrupts

The following events can generate an IN EPn interrupt request (where n indicates the endpoint number):

- A data packet that was loaded into the IN FIFO is sent to the USB host. (Set the **USB_CSIL.INPKTRDY** bit when a new packet is ready to be transferred. Hardware clears this bit when the data packet is sent.)
- A STALL was sent (the **USB_CSIL.SENTSTALL** bit set). Only bulk and interrupt endpoints can be stalled.
- The IN FIFO is flushed due to the **USB_CSIL.FLUSHPACKET** bit being set.

Any of these events causes assertion of the **INEPnIF** bit in the **USB_IIF** register, regardless of the status of the IN EPn interrupt mask bit **INEPnIE** in the **USB_IIE** register. The x in the register name refers to the endpoint number, 1–5.

The following events can generate an OUT EPn interrupt request:

- A data packet was received (the **USB_CSOL.OUTPKT_RDY** bit is set).
- A STALL was sent (the **USB_CSIL.SENTSTALL** bit is set). Only bulk and interrupt endpoints can be stalled.

Any of these events causes assertion of the **OUTEPnIF** bit in the **USB_OIF** register, regardless of the status of the OUT EPn interrupt mask bit **OUTEPnIE** in the **USB_OIE** register.

21.10.5 Bulk and Interrupt IN Endpoint

Interrupt IN transfers occur at regular intervals, whereas bulk IN transfers use available bandwidth not allocated to isochronous, interrupt, or control transfers.

Interrupt IN endpoints may set the **USB_CSIH.FORCEDATATOG** bit. When this bit is set, the data toggle bit is continuously toggled, regardless of whether an ACK was received or not. This feature is typically used by interrupt IN endpoints that are used to communicate rate feedback for isochronous endpoints.

Setting the **USB_CS0.SENDSTALL** bit can stall a bulk and interrupt IN endpoint. When the endpoint is stalled, the USB controller responds with a STALL handshake to IN tokens. The **USB_CSIL.SENTSTALL** bit is set, and an interrupt is generated, if enabled.

A bulk transfer longer than the maximum packet size is performed by splitting the transfer into a number of data packets of maximum size followed by a smaller data packet containing the remaining bytes. If the transfer length is a multiple of the maximum packet size, a zero-length data packet is sent last. This means that a packet with a size less than the maximum packet size denotes the end of the transfer. The AutoSet feature is useful in this case, because many data packets are of maximum size.

To use Bulk IN endpoint, the **USB_MAXI** register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the endpoint. In addition, the relevant interrupt enable bit in the **USB_IIE** register should be set to 1 (if an interrupt is required for this endpoint) and the **USB_CSIH** register should be set as shown below:

- **AUTISET** to 1 if AutoSet feature is required.
- **ISO** to 0 to enable Bulk protocol.
- **FORCE_DATA_TOG** to 0 to allow normal data toggle operation.

When a Bulk IN endpoint is first configured, the **USB_CSIL.CLRDATATOG** should be set. This will ensure that the data toggle (which is handled automatically by the USB controller) starts in the correct state. Also, if there are any data packets in the FIFO, they should be flushed by setting the **USB_CS0.FLUSHPACKET** bit. It may be necessary to set this bit twice in succession if double buffering is enabled.

When data is to be transferred over a Bulk IN pipe, a data packet needs to be loaded into the FIFO and **USB_CS0.INPKTRDY** bit should be set. When the packet has been sent, the this bit will be cleared by the USB controller and an interrupt generated so that the next packet can be loaded into the FIFO. If double packet buffering is enabled, then after the first packet has been loaded and the **USB_CSIL.INPKTRDY** bit set, this bit will immediately be cleared by the USB controller and an interrupt generated so that a second packet can be loaded into the FIFO. The software should operate in the same way, loading a packet when it receives an interrupt, regardless of whether double packet buffering is enabled or not. The packet size must not exceed the size specified in the **USB_MAXI** register. When a block of data larger than USB_MAXI is to be transferred, it must be sent as multiple packets. These packets should be USB_MAXI in size, except the last packet which holds the residue. The host may determine that all the data for a transfer has been sent by knowing the total amount of data that is expected. Alternatively it may infer that all the data have been sent when it receives a packet which is less than **USB_MAXI** in size. In the latter case, if the total size of the data block is a multiple of USB_MAXI, it will be necessary for the function to send a null packet after all the data has been sent. This is done by setting **USB_CSIL.INPKTRDY** when the next interrupt is received, without loading any data into the FIFO.

If the software wants to shut down the Bulk IN pipe, it should set the **USB_CSIL.SENDSTALL** bit. When the USB controller receives the next IN token, it will send a STALL to the host, set the **USB_CSIL.SENTSTALL** bit and generate an interrupt. When the software receives an interrupt with the **USB_CSIL.SENTSTALL** bit set, it should clear this bit and leave the **USB_CSIL.SENDSTALL** bit set until it is ready to re-enable the Bulk IN pipe. If the host failed to receive the STALL packet, it will send another IN token, so it is advisable to leave the **USB_CSIL.SENDSTALL** bit set until the software is ready to reenable the Bulk IN pipe. When a pipe is re-enabled, the data toggle sequence should be restarted by setting the **USB_CSLI.CLRDATATOG** bit.

An Interrupt IN endpoint is used to transfer periodic data from the function controller to the host. An Interrupt IN endpoint uses the same protocol as a Bulk IN endpoint and can be used the same way. Interrupt IN endpoints also support continuous toggle of the data toggle bit. This feature is enabled by setting the **USB_CSIH.FORCEDATATOG** bit. When this bit is set to 1, the USB controller will consider the packet as having been successfully sent and toggle the data bit for the endpoint, regardless of whether an ACK was received from the host.

Figure 21-14 is a flow chart of Bulk and Interrupt IN transactions.

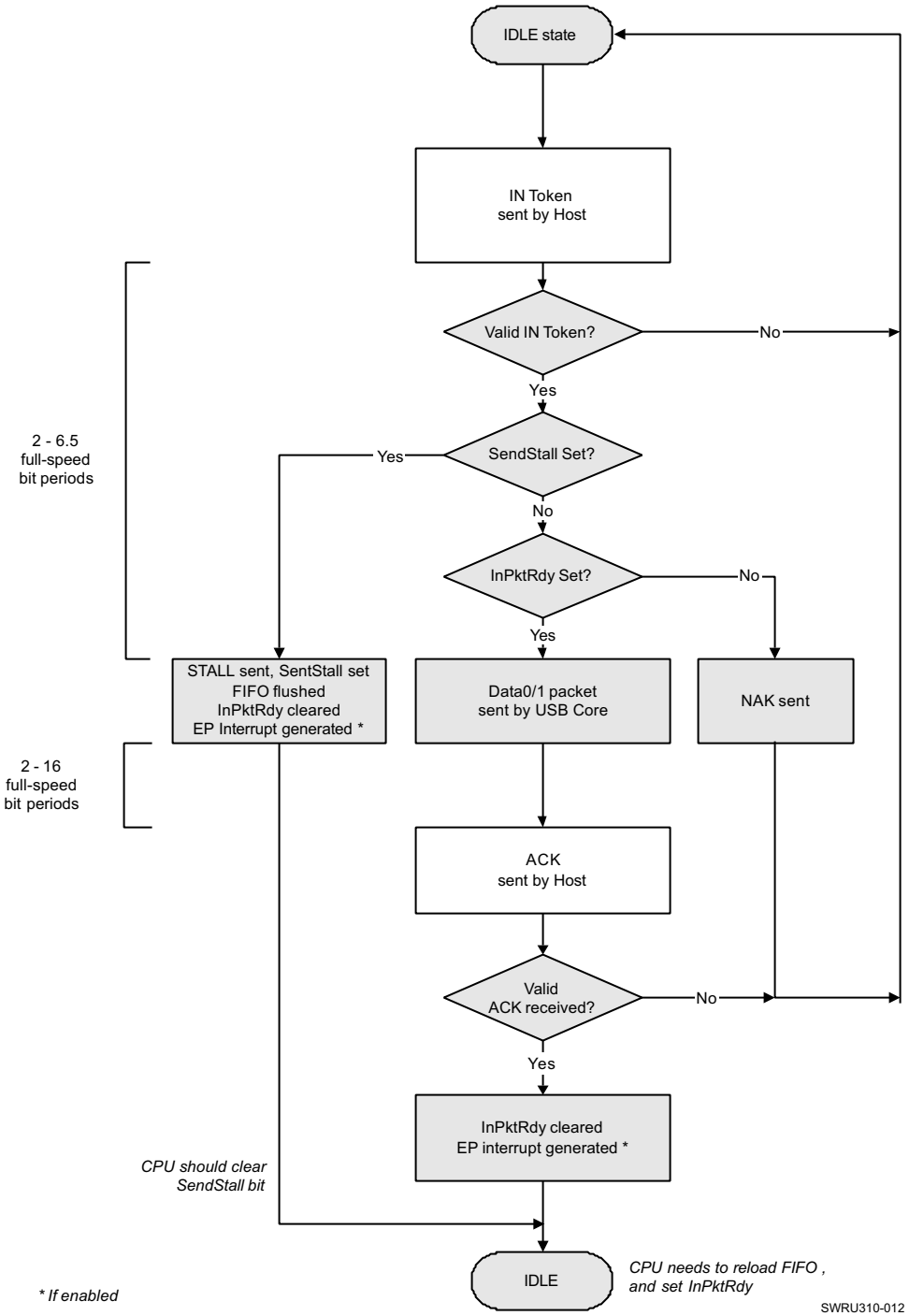


Figure 21-14. Bulk and Interrupt IN Transactions

21.10.6 Isochronous IN Endpoint

An isochronous IN endpoint is used to transfer periodic data from the USB controller to the host (one data packet every USB frame). If there is no data packet loaded in the IN FIFO when the USB host requests data, the USB controller sends a zero-length data packet, and the **USB_CSIL.UNDERRUN** bit is set. Double-buffering requires that a data packet is loaded into the IN FIFO during the frame preceding the frame where it should be sent. If the first data packet is loaded before an IN token is received, the data packet is sent during the same frame as it was loaded and hence violates the double-buffering strategy. Thus, when double-buffering is used, set the **USB_POW.ISOWAITSOF** bit to avoid this. Setting this bit ensures that a loaded data packet is not sent until the next SOF token has been received.

The AutoSet feature typically is not used for isochronous endpoints, because the packet size increases or decreases from frame to frame.

To use an Isochronous IN endpoint, the **USB_MAXI** register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the endpoint. In addition, the relevant interrupt enable bit in the **USB_IIE** register should be set to 1 (if an interrupt is required for this endpoint) and the **USB_CSIH** register should be set as shown below:

- **AUTISET** to 1 if AutoSet feature is required.
- **ISO** to 1 to enable Isochronous protocol.
- **FORCEDATATOG** to 0 ignored in Isochronous mode.

An Isochronous endpoint does not support data retries, therefore to avoid data underrun, the data to be sent to the host must be loaded into the FIFO before the IN token is received. The host will send one IN token per frame, however the time within the frame can vary. If an IN token is received near the end of one frame and then at the start of the next frame, there will be little time to reload the FIFO. For this reason, double buffering is usually required for an Isochronous IN endpoint.

An interrupt is generated whenever a packet is sent to the host and the software may use this interrupt to load the next packet into the FIFO and set the **USB_CSIL.INPKTRDY** bit in the same way as for a Bulk IN endpoint. As the interrupt could occur almost any time within a frame, depending on when the host has scheduled the transaction, this may result in irregular timing of FIFO load requests. If the data source for the endpoint is coming from some external hardware, it may be more convenient to wait until the end of each frame before loading the FIFO as this will minimize the requirement for additional buffering. This can be done by using either the **USB_CIF.SOFIE** interrupt or the external **SOF_PULSE** signal from the USB controller to trigger the loading of the next data packet. The **SOF_PULSE** is generated once per frame when a SOF packet is received (the USB controller also maintains an external frame counter so it can still generate a SOF_PULSE when the SOF packet has been lost). The interrupts may still be used to set the **USB_CSIL.INPKTRDY** bit and to check for data overruns/underruns.

If the endpoint has no data in its FIFO when an IN token is received, it will send a null data packet to the host and set the **USB_CSIL.UNDERRUN** bit register. This is an indication that the software is not supplying data fast enough for the host. It is up to the application to determine how this error condition is handled.

Figure 21-15 is a flow chart of Isochronous IN transactions.

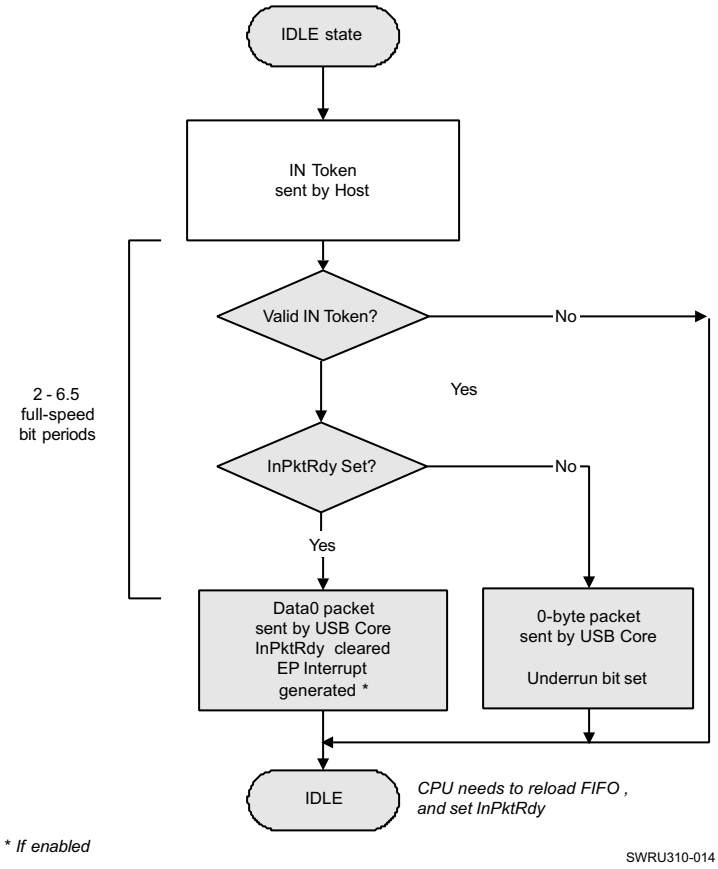


Figure 21-15. Isochronous IN Transactions

21.10.7 Bulk and Interrupt OUT Endpoint

Interrupt OUT transfers occur at regular intervals, whereas bulk OUT transfers use available bandwidth not allocated to isochronous, interrupt, or control transfers.

Setting the **USB_CSOL.SENDSTALL** bit can stall a bulk and interrupt OUT endpoint. When the endpoint is stalled, the USB controller responds with a STALL handshake when the host is done sending the data packet. The data packet is discarded and is not placed in the OUT FIFO. The USB controller asserts the **USB_CSOL.SENTSTALL** bit when the STALL handshake is sent and generates an interrupt request if the OUT endpoint interrupt is enabled.

As the AutoSet feature is useful for bulk IN endpoints, the AutoClear feature is useful for OUT endpoints, because many packets are of maximum size.

To use a Bulk OUT endpoint, the **USB_MAXO** register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the endpoint. In

addition, the relevant interrupt enable bit in the USB_OIE register should be set to 1 (if an interrupt is required for

this endpoint) and the **USB_CSOH** register should be set as shown below :

- **AUTOCLEAR** to 1 if AutoClear feature is required.
- **ISO** to 0 to enable Bulk protocol.

When a Bulk OUT endpoint is first configured, **USB_CSOL.CLRDATATOG** should be set. This will ensure that the data toggle (which is handled automatically by the USB controller) starts in the correct state. Also, if there are any data packets in the FIFO (indicated by the setting of **USB_CSOL.OUTPKTRDY** bit), they should be flushed by setting the **USB_CSOL.FLUSHPACKET** bit. It may be necessary to set this bit twice in succession if double buffering is enabled.

When a data packet is received by a Bulk OUT endpoint, the **USB_CSOL.OUTPKTRDY** bit is set and an interrupt is generated. The software should read **USB_CNTH.FIFOCNTH** register for the endpoint to determine the size of the data packet. The data packet should be read from the FIFO, then the **USB_CSOL.OUTPKTRDY** bit should be cleared. The packet sizes should not exceed the size specified in the **USB_MAXO** register (as this should be the value set in the wMaxPacketSize field of the endpoint descriptor sent to the host). When a block of data larger than **USB_MAXO** is to be sent to the function, it will be sent as multiple packets. All the packets will be **USB_MAXO** in size, except the last packet which will contain the residue. The software may use an application specific method of determining the total size of the block and hence when the last packet has been received. Alternatively it may infer that the entire block has been received when it receives a packet which is less than **USB_MAXO** in size.

If the software wants to shut down the Bulk OUT pipe, it should set the **USB_CSOL.SENDSTALL** bit. When the USB controller receives the next packet it will send a STALL to the host, set the **USB_CSOL.SENTSTALL** bit and generate an interrupt. When the software receives an interrupt with the this bit set, it should clear the it and leave the **USB_CSOL.SENDSTALL** bit set until it is ready to re-enable the Bulk OUT pipe. If the host failed to receive the STALL packet for some reason, it will send another packet, so it is advisable to leave the **USB_CSOL.SENDSTALL** bit set until the software is ready to reenable the Bulk OUT pipe. When a Bulk OUT pipe is re-enabled, the data toggle sequence should be restarted by setting the **USB_CSOL.CLRDATATOG** bit.

An Interrupt OUT endpoint is used to transfer periodic data from the host to a function controller. An Interrupt OUT endpoint uses almost the same protocol as a Bulk OUT endpoint and can be used the same way.

Figure 21-16 is a flow chart of Bulk and Interrupt OUT transactions.

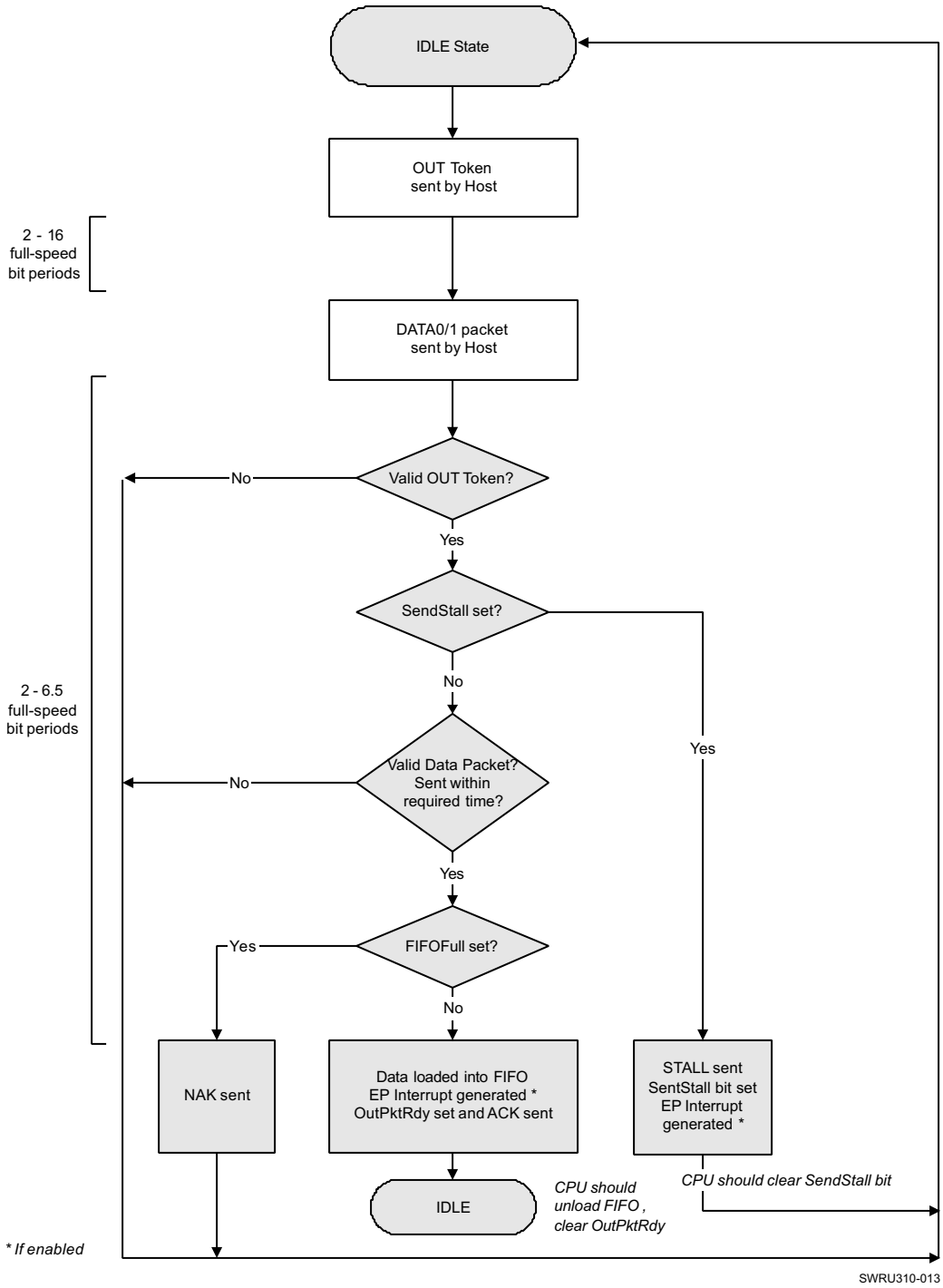


Figure 21-16. Bulk and Interrupt OUT Transactions

21.10.8 Isochronous OUT Endpoint

An isochronous OUT endpoint is used to transfer periodic data from the host to the USB controller (one data packet every USB frame). If there is no buffer available when receiving a data packet, the **USB_CSOL.OVERRUN** bit is set and the packet data is lost. Firmware can reduce this possibility by using double-buffering and using DMA to unload data packets effectively.

An isochronous data packet in the OUT FIFO may have bit errors. The hardware detects this condition and sets the **USB_CSOL.DATA_ERROR** bit. Firmware should therefore always check this bit when unloading a data packet.

The AutoClear feature typically is not used for isochronous endpoints, because the packet size increases or decreases from frame to frame.

To use an Isochronous OUT endpoint, the **USB_MAXO** register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the endpoint. In addition, the relevant interrupt enable bit in the **USB_OIE** register should be set to 1 (if an interrupt is required for this endpoint) and the **USB_CSOH** register should be set as shown below:

- **AUTOCLEAR** to 1 if AutoClear feature is required.
- **ISO** to 1 to enable Isochronous protocol.

An Isochronous endpoint does not support data retries so, if a data overrun is to be avoided, there must be space in the FIFO to accept a packet when it is received. The host will send one packet per frame, however the time within the frame can vary. If a packet is received near the end of one frame and another arrives at the start of the next frame, there will be little time to unload the FIFO. For this reason, double buffering is usually required for an Isochronous OUT endpoint. An interrupt is generated whenever a packet is received from the host and the software may use this interrupt to unload the packet from the FIFO and clear the **USB_CSOL.OUTPKTRDY** bit.

As the interrupt could occur almost any time within a frame, depending on when the host has scheduled the transaction, the timing of FIFO unload requests will probably be irregular. If the data sink for the endpoint is going to some external hardware, it may be better to minimize the requirement for additional buffering by waiting until the end of each frame before unloading the FIFO. This can be done by using either the **USB_CIF.SOFIE** interrupt or the external SOF_PULSE signal from the USB controller to trigger the unloading of the data packet. The SOF_PULSE is generated once per frame when a SOF packet is received (the USB controller also maintains an external frame counter so it can still generate a SOF_PULSE when the SOF packet has been lost). The interrupts may still be used to clear the **USB_CSOL.OUTPKTRDY** bit and to check for data overruns/underruns.

If there is no space in the FIFO to store a packet when it is received from the host, the **USB_CSOL.OVERRUN** bit will be set. This is an indication that the software is not unloading data fast enough for the host. It is up to the application to determine how this error condition is handled.

Figure 21-17 is a flow chart of Isochronous OUT transactions.

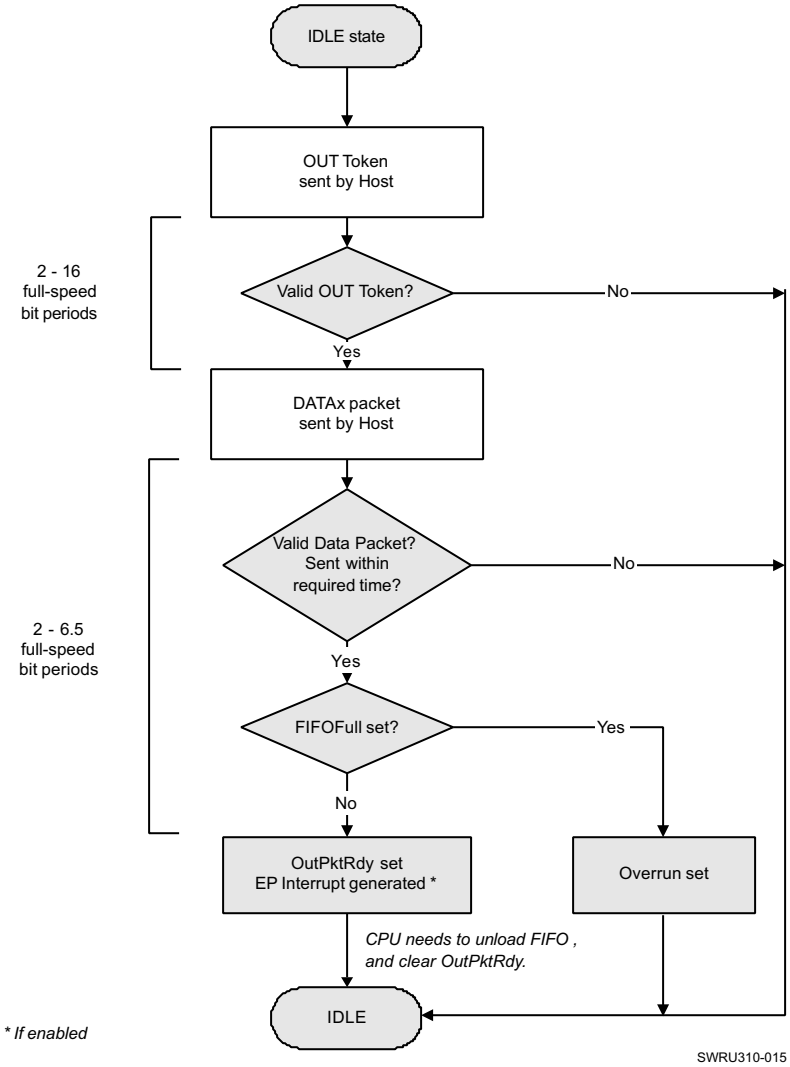


Figure 21-17. Isochronous OUT Transactions

21.11 DMA

Direct memory access (DMA) should be used to fill the IN endpoint FIFOs and empty the OUT endpoint FIFOs. Using DMA improves the read and write performance significantly compared to using the CPU; therefore, using DMA is highly recommended unless timing is not critical or only a few bytes are to be transferred.

Because there are no DMA triggers for the USB controller, firmware must trigger DMA transfers.

21.12 Remote Wake-Up

The USB controller can resume from suspend by signaling resume to the USB hub. Resume is performed by setting the **RESUME** bit in the **USB_POW** register to 1 for approximately 10 ms. According to the USB 2.0 Specification, the resume signaling must exist for at least 1ms and no more than 15ms. It is, however, recommended to keep the resume signaling for approximately 10ms. The USB descriptor must declare support for remote wakeup and the USB host must grant the device the privilege to perform remote wakeup (through a SET_FEATURE request).

NOTE: Two interrupts will be generated if both **USB_IWE** and I/O pad interrupts are enabled.

21.13 USB Registers Overview

This section describes all USB registers used for control and status for the USB. The USB registers reside at base address of 0x4008 9000. These registers can be divided into three groups: The common USB registers, the indexed endpoint registers, and the endpoint FIFO registers. The common USB registers provide control and status for the entire controller. The INDEX registers provide control and status information for the different endpoints and represent the currently selected endpoint. The endpoints are selected by writing the endpoint number to the **USB_INDEX** register. So to access the registers for IN Endpoint 1 and OUT Endpoint 1, 1 must first be written to the **USB_INDEX** register. The registers return to their reset values and the FIFOs are cleared when the chip enters PM2 or PM3.

21.14 USB Registers

21.14.1 USB Registers

21.14.1.1 USB Registers Mapping Summary

This section provides information on the USB module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 21-3. USB Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
USB_ADDR	RW	32	0x0000 0000	0x00	0x4008 9000
USB_POW	RW	32	0x0000 0000	0x04	0x4008 9004
USB_IIF	RO	32	0x0000 0000	0x08	0x4008 9008
USB_OIF	RO	32	0x0000 0000	0x10	0x4008 9010
USB_CIF	RO	32	0x0000 0000	0x18	0x4008 9018
USB_IIE	RW	32	0x0000 003F	0x1C	0x4008 901C
USB_OIE	RW	32	0x0000 003E	0x24	0x4008 9024
USB_CIE	RW	32	0x0000 0006	0x2C	0x4008 902C
USB_FRML	RO	32	0x0000 0000	0x30	0x4008 9030
USB_FRMH	RO	32	0x0000 0000	0x34	0x4008 9034
USB_INDEX	RW	32	0x0000 0000	0x38	0x4008 9038
USB_CTRL	RW	32	0x0000 0000	0x3C	0x4008 903C
USB_MAXI	RW	32	0x0000 0000	0x40	0x4008 9040
USB_CS0_CSIL	RW	32	0x0000 0000	0x44	0x4008 9044
USB_CSIH	RW	32	0x0000 0020	0x48	0x4008 9048
USB_MAXO	RW	32	0x0000 0000	0x4C	0x4008 904C
USB_CSOL	RW	32	0x0000 0000	0x50	0x4008 9050
USB_CSOH	RW	32	0x0000 0000	0x54	0x4008 9054
USB_CNT0_CNTL	RO	32	0x0000 0000	0x58	0x4008 9058
USB_CNTH	RO	32	0x0000 0000	0x5C	0x4008 905C
USB_F0	RW	32	0x0000 0000	0x80	0x4008 9080
USB_F1	RW	32	0x0000 0000	0x88	0x4008 9088

Table 21-3. USB Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
USB_F2	RW	32	0x0000 0000	0x90	0x4008 9090
USB_F3	RW	32	0x0000 0000	0x98	0x4008 9098
USB_F4	RW	32	0x0000 0000	0xA0	0x4008 90A0
USB_F5	RW	32	0x0000 0000	0xA8	0x4008 90A8

21.14.1.2 USB Register Descriptions

USB_ADDR

Address offset	0x00		
Physical Address	0x4008 9000	**Instance**	USB
Description	Function address		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	UPDATE	This bit is set by hardware when writing to this register, and is cleared by hardware when the new address becomes effective.	RO	0
6:0	USBADDR	Device address. The address shall be updated upon successful completion of the status stage of the SET_ADDRESS request.	RW	0x00

USB_POW

Address offset	0x04		
Physical Address	0x4008 9004	**Instance**	USB
Description	Power management and control register		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								ISOWAITSOF	RESERVED			RST	RESUME	SUSPEND	SUSPENDEN

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	ISOWAITSOF	For isochronous mode IN endpoints: When set, the USB controller will wait for an SOF token from the time USB_CSIL.INPKTRDY is set before sending the packet. If an IN token is received before an SOF token, then a zero length data packet will be sent.	RW	0
6:4	RESERVED	This bit field is reserved.	RO	0x0
3	RST	Indicates that reset signaling is present on the bus	RO	0

Bits	Field Name	Description	Type	Reset
2	RESUME	Drives resume signaling for remote wakeup According to the USB Specification, the resume signal must be held active for at least 1 ms and no more than 15 ms. It is recommended to keep this bit set for approximately 10 ms.	RW	0
1	SUSPEND	Indicates entry into suspend mode Suspend mode must be enabled by setting USB_POW.SUSPENDEN Software clears this bit by reading the USB_CIF register or by asserting USB_POW.RESUME	RO	0
0	SUSPENDEN	Enables detection of and entry into suspend mode.	RW	0

USB_IIF

Address offset	0x08		
Physical Address	0x4008 9008	**Instance**	USB
Description	Interrupt flags for endpoint 0 and IN endpoints 1-5		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED	INEP5IF	INEP4IF	INEP3IF	INEP2IF	INEP1IF	EP0IF	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:6	RESERVED	This bit field is reserved.	RO	0x0
5	INEP5IF	Interrupt flag for IN endpoint 5 Cleared by hardware when read	RO	0
4	INEP4IF	Interrupt flag for IN endpoint 4 Cleared by hardware when read	RO	0
3	INEP3IF	Interrupt flag for IN endpoint 3 Cleared by hardware when read	RO	0
2	INEP2IF	Interrupt flag for IN endpoint 2 Cleared by hardware when read	RO	0
1	INEP1IF	Interrupt flag for IN endpoint 1 Cleared by hardware when read	RO	0
0	EP0IF	Interrupt flag for endpoint 0 Cleared by hardware when read	RO	0

USB_OIF

Address offset	0x10		
Physical Address	0x4008 9010	**Instance**	USB
Description	Interrupt flags for OUT endpoints 1-5		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED	OUTEP5IF	OUTEP4IF	OUTEP3IF	OUTEP2IF	OUTEP1IF	RESERVED	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:6	RESERVED	This bit field is reserved.	RO	0x0
5	OUTEP5IF	Interrupt flag for OUT endpoint 5 Cleared by hardware when read	RO	0
4	OUTEP4IF	Interrupt flag for OUT endpoint 4 Cleared by hardware when read	RO	0
3	OUTEP3IF	Interrupt flag for OUT endpoint 3 Cleared by hardware when read	RO	0
2	OUTEP2IF	Interrupt flag for OUT endpoint 2 Cleared by hardware when read	RO	0
1	OUTEP1IF	Interrupt flag for OUT endpoint 1 Cleared by hardware when read	RO	0
0	RESERVED	This bit field is reserved.	RO	0

USB_CIF

Address offset	0x18		
Physical Address	0x4008 9018	**Instance**	USB
Description	Common USB interrupt flags		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED				SOFIF	RSTIF	RESUMEIF	SUSPENDIF

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:4	RESERVED	This bit field is reserved.	RO	0x0
3	SOFIF	Start-of-frame interrupt flag Cleared by hardware when read	RO	0
2	RSTIF	Reset interrupt flag Cleared by hardware when read	RO	0
1	RESUMEIF	Resume interrupt flag Cleared by hardware when read	RO	0
0	SUSPENDIF	Suspend interrupt flag Cleared by hardware when read	RO	0

USB_IIE

Address offset	0x1C		
Physical Address	0x4008 901C	**Instance**	USB
Description	Interrupt enable mask for IN endpoints 1-5 and endpoint 0		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED	INEP5IE	INEP4IE	INEP3IE	INEP2IE	INEP1IE	EP0IE	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:6	RESERVED	This bit field is reserved.	RO	0x0
5	INEP5IE	Interrupt enable for IN endpoint 5 0: Interrupt disabled 1: Interrupt enabled	RW	1
4	INEP4IE	Interrupt enable for IN endpoint 4 0: Interrupt disabled 1: Interrupt enabled	RW	1
3	INEP3IE	Interrupt enable for IN endpoint 3 0: Interrupt disabled 1: Interrupt enabled	RW	1
2	INEP2IE	Interrupt enable for IN endpoint 2 0: Interrupt disabled 1: Interrupt enabled	RW	1
1	INEP1IE	Interrupt enable for IN endpoint 1 0: Interrupt disabled 1: Interrupt enabled	RW	1
0	EP0IE	Interrupt enable for endpoint 0 0: Interrupt disabled 1: Interrupt enabled	RW	1

USB_OIE

Address offset	0x24		
Physical Address	0x4008 9024	**Instance**	USB
Description	Interrupt enable mask for OUT endpoints 1-5		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED	OUTEP5IE	OUTEP4IE	OUTEP3IE	OUTEP2IE	OUTEP1IE	RESERVED	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:6	RESERVED	This bit field is reserved.	RO	0x0
5	OUTEP5IE	Interrupt enable for OUT endpoint 5 0: Interrupt disabled 1: Interrupt enabled	RW	1
4	OUTEP4IE	Interrupt enable for OUT endpoint 4 0: Interrupt disabled 1: Interrupt enabled	RW	1
3	OUTEP3IE	Interrupt enable for OUT endpoint 3 0: Interrupt disabled 1: Interrupt enabled	RW	1
2	OUTEP2IE	Interrupt enable for OUT endpoint 2 0: Interrupt disabled 1: Interrupt enabled	RW	1
1	OUTEP1IE	Interrupt enable for OUT endpoint 1 0: Interrupt disabled 1: Interrupt enabled	RW	1
0	RESERVED	This bit field is reserved.	RO	0

USB_CIE

Address offset	0x2C		
Physical Address	0x4008 902C	**Instance**	USB
Description	Common USB interrupt enable mask		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED				SOFIE	RSTIE	RESUMEIE	SUSPENDIE

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:4	RESERVED	This bit field is reserved.	RO	0x0
3	SOFIE	Start-of-frame interrupt enable 0: Interrupt disabled 1: Interrupt enabled	RW	0
2	RSTIE	Reset interrupt enable 0: Interrupt disabled 1: Interrupt enabled	RW	1
1	RESUMEIE	Resume interrupt enable 0: Interrupt disabled 1: Interrupt enabled	RW	1
0	SUSPENDIE	Suspend interrupt enable 0: Interrupt disabled 1: Interrupt enabled	RW	0

USB_FRML

Address offset	0x30		
Physical Address	0x4008 9030	**Instance**	USB
Description	Frame number (low byte)		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	FRAMEL	Bits 7:0 of the 11-bit frame number The frame number is only updated upon successful reception of SOF tokens	RO	0x00

USB_FRMH

Address offset	0x34		
Physical Address	0x4008 9034	**Instance**	USB
Description	Frame number (high byte)		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:3	RESERVED	This bit field is reserved.	RO	0x00
2:0	FRAMEH	Bits 10:8 of the 11-bit frame number The frame number is only updated upon successful reception of SOF tokens	RO	0x0

USB_INDEX

Address offset	0x38		
Physical Address	0x4008 9038	**Instance**	USB
Description	Index register for selecting the endpoint status and control registers		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:4	RESERVED	This bit field is reserved.	RO	0x0
3:0	USBINDEX	Index of the currently selected endpoint The index is set to 0 to enable access to endpoint 0 control and status registers The index is set to 1, 2, 3, 4 or 5 to enable access to IN/OUT endpoint 1, 2, 3, 4 or 5 control and status registers, respectively	RW	0x0

USB_CTRL

Address offset	0x3C		
Physical Address	0x4008 903C	**Instance**	USB
Description	USB peripheral control register		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PLLLOCKED	RESERVED				RESERVED	PLLEN	USBEN

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	PLLLOCKED	PLL lock status. The PLL is locked when USB_CTRL.PLLLOCKED is 1.	RO	0
6:3	RESERVED	This bit field is reserved.	RO	0x0
2	RESERVED	This bit field is reserved.	RO	0
1	PLLEN	48 MHz USB PLL enable When this bit is set, the 48 MHz PLL is started. Software must avoid access to other USB registers before the PLL has locked; that is, USB_CTRL.PLLLOCKED is 1. This bit can be set only when USB_CTRL.USBEN is 1. The PLL must be disabled before entering PM1 when suspended, and must be re-enabled when resuming operation.	RW	0
0	USBEN	USB enable The USB controller is reset when this bit is cleared	RW	0

USB_MAXI

Address offset	0x40		
Physical Address	0x4008 9040	**Instance**	USB
Description	Indexed register: For USB_INDEX = 1-5: Maximum packet size for IN endpoint {1-5}		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	USBMAXI	Maximum packet size, in units of 8 bytes, for the selected IN endpoint The value of this register should match the wMaxPacketSize field in the standard endpoint descriptor for the endpoint. The value must not exceed the available memory.	RW	0x00

USB_CS0_CSIL

Address offset	0x44		
Physical Address	0x4008 9044	**Instance**	USB
Description	Indexed register: For USB_INDEX = 0: Endpoint 0 control and status For USB_INDEX = 1-5: IN endpoint {1-5} control and status (low byte)		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED	CLROUTPKTRDY_OR_CLRDATATOG	SENDSTALL_OR_SENTSTALL	SETUPEND_OR_SENDSTALL	DATAEND_OR_FLUSHPACKET	SENTSTALL_OR_UNDERRUN	INPKTRDY_OR_PKTPRESENT	OUTPKTRDY_OR_INPKTRDY

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	RESERVED	This bit field is reserved.	RW	0
6	CLROUTPKTRDY_OR_CLRDATATOG	USB_CS0.CLROUTPKTRDY [RW]: Software sets this bit to clear the USB_CS0.OUTPKTRDY bit. It is cleared automatically. USB_CSIL.CLRDATATOG [RW]: Software sets this bit to reset the IN endpoint data toggle to 0.	RW	0

Bits	Field Name	Description	Type	Reset
5	SENDSTALL_OR_SENTSTALL	USB_CS0.SENDSTALL [RW]: Software sets this bit to terminate the current transaction with a STALL handshake. The bit is cleared automatically when the STALL handshake has been transmitted. USB_CSIL.SENTSTALL [RW]: For bulk/interrupt mode IN endpoints: This bit is set when a STALL handshake is transmitted. The FIFO is flushed and the USB_CSIL.INPKTRDY bit cleared. Software should clear this bit.	RW	0
4	SETUPEND_OR_SENDSTALL	USB_CS0.SETUPEND [RO]: This bit is set when a control transaction ends before the USB_CS0.DATAEND bit has been set. An interrupt is generated and the FIFO flushed at this time. Software clears this bit by setting USB_CS0.CLRSETUPEND. CSIL.SENDSTALL [RW]: For bulk/interrupt mode IN endpoints: Software sets this bit to issue a STALL handshake. Software clears this bit to terminate the stall condition.	RO	0
3	DATAEND_OR_FLUSHPACKET	USB_CS0.DATAEND [RW]: This bit is used to signal the end of the data stage, and must be set: 1. When the last data packet is loaded and USB_CS0.INPKTRDY is set. 2. When the last data packet is unloaded and USB_CS0.CLROUTPKTRDY is set. 3. When USB_CS0.INPKTRDY is set to send a zero-length packet. The USB controller clears this bit automatically. USB_CSIL.FLUSHPACKET [RW]: Software sets this bit to flush the next packet to be transmitted from the IN endpoint FIFO. The FIFO pointer is reset and the USB_CSIL.INPKTRDY bit is cleared. Note: If the FIFO contains two packets, USB_CSIL.FLUSHPACKET will need to be set twice to completely clear the FIFO.	RW	0
2	SENTSTALL_OR_UNDERRUN	USB_CS0.SENTSTALL [RW]: This bit is set when a STALL handshake is sent. An interrupt is generated is generated when this bit is set. Software must clear this bit. USB_CSIL.UNDERRUN [RW]: In isochronous mode, this bit is set when a zero length data packet is sent after receiving an IN token with USB_CSIL.INPKTRDY not set. In bulk/interrupt mode, this bit is set when a NAK is returned in response to an IN token. Software should clear this bit.	RW	0
1	INPKTRDY_OR_PKTPRESENT	USB_CS0. INPKTRDY [RW]: Software sets this bit after loading a data packet into the endpoint 0 FIFO. It is cleared automatically when the data packet has been transmitted. An interrupt is generated when the bit is cleared. USB_CSIL.PKTPRESENT [RO]: This bit is set when there is at least one packet in the IN endpoint FIFO.	RW	0
0	OUTPKTRDY_OR_INPKTRDY	USB_CS0.OUTPKTRDY [RO]: Endpoint 0 data packet received An interrupt request (EP0) is generated if the interrupt is enabled. Software must read the endpoint 0 FIFO empty, and clear this bit by setting USB_CS0.CLROUTPKTRDY USB_CSIL.INPKTRDY [RW]: IN endpoint {1-5} packet transfer pending Software sets this bit after loading a data packet into the FIFO. It is cleared automatically when a data packet has been transmitted. An interrupt is generated (if enabled) when the bit is cleared. When using double-buffering, the bit is cleared immediately if the other FIFO is empty.	RO	0

USB_CSIH

Address offset	0x48		
Physical Address	0x4008 9048	**Instance**	USB
Description	Indexed register: For USB_INDEX = 1-5: IN endpoint {1-5} control and status (high byte)		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								AUTISET	ISO	RESERVED	RESERVED	FORCEDATATOG	RESERVED		INDBLBUF

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	AUTISET	If set by software, the USB_CSIL.INPKTRDY bit is automatically set when a data packet of maximum size (specified by USBMAXI) is loaded into the IN endpoint FIFO. If a packet of less than the maximum packet size is loaded, then USB_CSIL.INPKTRDY will have to be set manually.	RW	0
6	ISO	Selects IN endpoint type: 0: Bulk/interrupt 1: Isochronous	RW	0
5	RESERVED	This bit field is reserved.	RW	1
4	RESERVED	This bit field is reserved.	RW	0
3	FORCEDATATOG	Software sets this bit to force the IN endpoint's data toggle to switch after each data packet is sent regardless of whether an ACK was received. This can be used by interrupt IN endpoints which are used to communicate rate feedback for isochronous endpoints.	RW	0
2:1	RESERVED	This bit field is reserved.	RO	0x0
0	INDBLBUF	IN endpoint FIFO double-buffering enable: 0: Double buffering disabled 1: Double buffering enabled	RW	0

USB_MAXO

Address offset	0x4C		
Physical Address	0x4008 904C	**Instance**	USB
Description	Indexed register: For USB_INDEX = 1-5: Maximum packet size for OUT endpoint {1-5}		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	USBMAXO	Maximum packet size, in units of 8 bytes, for the selected OUT endpoint The value of this register should match the wMaxPacketSize field in the standard endpoint descriptor for the endpoint. The value must not exceed the available memory.	RW	0x00

USB_CSOL

Address offset	0x50		
Physical Address	0x4008 9050	**Instance**	USB
Description	Indexed register: For USB_INDEX = 1-5: OUT endpoint {1-5} control and status (low byte)		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								CLRDATATOG	SENTSTALL	SENDSTALL	FLUSHPACKET	DATAERROR	OVERRUN	FIFOFULL	OUTPKTRDY

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	CLRDATATOG	Software sets this bit to reset the endpoint data toggle to 0.	RW	0
6	SENTSTALL	This bit is set when a STALL handshake is transmitted. An interrupt is generated when this bit is set. Software should clear this bit.	RW	0
5	SENDSTALL	For bulk/interrupt mode OUT endpoints: Software sets this bit to issue a STALL handshake. Software clears this bit to terminate the stall condition.	RW	0
4	FLUSHPACKET	Software sets this bit to flush the next packet to be read from the endpoint OUT FIFO. Note: If the FIFO contains two packets, USB_CSOL.FLUSHPACKET will need to be set twice to completely clear the FIFO.	RW	0
3	DATAERROR	For isochronous mode OUT endpoints: This bit is set when USB_CSOL.OUTPKTRDY is set if the data packet has a CRC or bit-stuff error. It is cleared automatically when USB_CSOL.OUTPKTRDY is cleared.	RO	0
2	OVERRUN	For isochronous mode OUT endpoints: This bit is set when an OUT packet cannot be loaded into the OUT endpoint FIFO. Firmware should clear this bit.	RW	0
1	FIFOFULL	This bit is set when no more packets can be loaded into the OUT endpoint FIFO.	RO	0
0	OUTPKTRDY	This bit is set when a data packet has been received. Software should clear this bit when the packet has been unloaded from the OUT endpoint FIFO. An interrupt is generated when the bit is set.	RW	0

USB_CSOH

Address offset	0x54		
Physical Address	0x4008 9054	**Instance**	USB
Description	Indexed register: For USB_INDEX = 1-5: OUT endpoint {1-5} control and status (high byte)		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								AUTOCLEAR	ISO	RESERVED	RESERVED	RESERVED			OUTDBLBUF

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	AUTOCLEAR	If software sets this bit, the USB_CSOL.OUTPKTRDY bit will be automatically cleared when a packet of maximum size (specified by USB_MAXO) has been unloaded from the OUT FIFO. When packets of less than the maximum packet size are unloaded, USB_CSOL.OUTPKTRDY will have to be cleared manually.	RW	0
6	ISO	Selects OUT endpoint type: 0: Bulk/interrupt 1: Isochronous	RW	0
5	RESERVED	This bit field is reserved.	RW	0
4	RESERVED	This bit field is reserved.	RW	0
3:1	RESERVED	This bit field is reserved.	RO	0x0
0	OUTDBLBUF	OUT endpoint FIFO double-buffering enable: 0: Double buffering disabled 1: Double buffering enabled	RW	0

USB_CNT0_CNTL

Address offset	0x58		
Physical Address	0x4008 9058	**Instance**	USB
Description	Indexed register: For USB_INDEX = 0: Number of received bytes in the endpoint 0 FIFO For USB_INDEX = 1-5: Number of received bytes in the OUT endpoint {1-5} FIFO (low byte)		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	FIFOCNT_OR_FIFOCNTL	USB_CS0.FIFOCNT (USBINDEX = 0) [RO]: Number of bytes received in the packet in the endpoint 0 FIFO Valid only when USB_CS0.OUTPKTRDY is set USB_CSIL.FIFOCNTL (USBINDEX = 1 to 5) [RW]: Bits 7:0 of the of the number of bytes received in the packet in the OUT endpoint {1-5} FIFO Valid only when USB_CSOL.OUTPKTRDY is set	RO	0x00

USB_CNTH

Address offset	0x5C		
Physical Address	0x4008 905C	**Instance**	USB
Description	Indexed register: For USB_INDEX = 1-5: Number of received in the OUT endpoint {1-5} FIFO (high byte)		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:3	RESERVED	This bit field is reserved.	RO	0x00
2:0	FIFOCNTH	Bits 10:8 of the of the number of bytes received in the packet in the OUT endpoint {1-5} FIFO Valid only when USB_CSOL.OUTPKTRDY is set	RO	0x0

USB_F0

Address offset	0x80		
Physical Address	0x4008 9080	**Instance**	USB
Description	Endpoint 0 FIFO		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								USBF0							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	USBF0	Endpoint 0 FIFO Reading this register unloads one byte from the endpoint 0 FIFO. Writing to this register loads one byte into the endpoint 0 FIFO. The FIFO memory for EP0 is used for incoming and outgoing data packets.	RW	0x00

USB_F1

Address offset	0x88		
Physical Address	0x4008 9088	**Instance**	USB
Description	IN/OUT endpoint 1 FIFO		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								USBF1							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	USBF1	Endpoint 1 FIFO register Reading this register unloads one byte from the EP1 OUT FIFO. Writing to this register loads one byte into the EP1 IN FIFO.	RW	0x00

USB_F2

Address offset	0x90		
Physical Address	0x4008 9090	**Instance**	USB
Description	IN/OUT endpoint 2 FIFO		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								USBF2							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	USBF2	Endpoint 2 FIFO register Reading this register unloads one byte from the EP2 OUT FIFO. Writing to this register loads one byte into the EP2 IN FIFO.	RW	0x00

USB_F3

Address offset	0x98		
Physical Address	0x4008 9098	**Instance**	USB
Description	IN/OUT endpoint 3 FIFO		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								USBF3							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	USBF3	Endpoint 3 FIFO register Reading this register unloads one byte from the EP3 OUT FIFO. Writing to this register loads one byte into the EP3 IN FIFO.	RW	0x00

USB_F4

Address offset	0xA0		
Physical Address	0x4008 90A0	**Instance**	USB
Description	IN/OUT endpoint 4 FIFO		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								USBF4							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	USBF4	Endpoint 4 FIFO register Reading this register unloads one byte from the EP4 OUT FIFO. Writing to this register loads one byte into the EP4 IN FIFO.	RW	0x00

USB_F5

Address offset	0xA8		
Physical Address	0x4008 90A8	**Instance**	USB
Description	IN/OUT endpoint 5 FIFO		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								USBF5							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	USBF5	Endpoint 5 FIFO register Reading this register unloads one byte from the EP5 OUT FIFO. Writing to this register loads one byte into the EP5 IN FIFO.	RW	0x00

Security Core

This chapter provides information on configuring the security engine of the CC2538 device.

Topic **Page**

22.1 PKA Engine

22.1.1 Terms and Conventions Used in this Manual

22.1.1.1 Acronyms

ACT2	Addition Chaining Table with 2 address bits (4 entries)
ACT4	Addition Chaining Table with 4 address bits (16 entries)
CRT	Chinese Remainder Theorem
ECC	Elliptic Curve Cryptography
LNME	Large Number Multiplier and Exponentiator
MMM	Montgomery Modular Multiplication
PE	Processing Element
PKA	Public Key Accelerator
PKCP	Public Key Co-Processor
RAM	Random Access Memory
ROM	Read Only Memory

22.1.1.2 Formulae and Nomenclature

This document contains formulas and nomenclature for different data types. The presentation of syntax is given below:

0x00 or 0h	Hexadecimal value
0b	Binary value
0d	Digital logic 0 or LOW
‘0’	Decimal value
‘1’	Digital logic 1 or HIGH
bit	Binary digit
8 bits	1 byte
16 bits	half word
32 bits	Dword
64 bits	dual-word
128 bits	quad-word
MOD	MODulo
REM	REMainder
A & B	A Logical AND B
A OR B	A Logical OR B
NOR	Logical NOR
NOT A	Logical NOT
A NOR B	A Logical NOR B
A ⊕ B	A logic eXclusive OR B or XOR
XNOR	logic eXclusive NOR
NAND	Logical NAND
DIV	Integer DIVision
||	Concatenation
[n:m]	Size of a register or signal in bits where n > m(1)

(1) [31:0] indicates a size of 32 bits with most significant bit 31 and least significant bit 0.

[11:3] indicates a size of 9 bits with most significant bit 11 and least significant bit 3.

22.1.2 Overview

22.1.2.1 Feature List

The *PKA Engine* provides the following basic operations in hardware:

- Large vector addition, subtraction and combined addition/subtraction
- Large vector shift right or left
- Large vector multiplication, division (with and without quotient)
- Large vector compare and copy
- Large vector modular Montgomery multiplication
- Large vector modular Montgomery exponentiation

The optional *LNME* module is needed to perform the latter two operations – they are not available on the *PKCP*-only version. The *PKCP* module performs the other operations.The *PKA Engine* provides the following complex operations:

- Large vector unsigned value modular exponentiation
- Large vector unsigned value modular exponentiation using the 'Chinese Remainders Theorem'
- ('CRT') method with pre-calculated Q inverse vector
- Modular inversion: given A and M, calculate B such that ((A x B) MOD M) = 1
- ECC point addition/doubling on elliptic curve y2=x3+*a*x+*b* (mod *p*) with '*p*' a prime number and '*a*'and '*b*' input values to the operation, adding two identical points automatically performs pointdoubling• ECC point multiplication on elliptic curve y2=x3+*a*x+*b* (mod *p*) with '*p*' a prime number and '*a*' and'*b*' input values to the operation – a version of the 'Montgomery ladder' algorithm is used to provideside channel attack resistanceThese operations are done under control of the *Sequencer*. Depending on the firmware, the *Sequencer* implements modular exponentiations and ECC point multiplications using only the *PKCP* or using both the *PKCP* and the *LNME*. In both cases, the *Sequencer* hides the fact that the actual exponentiation and ECC point multiplication is done using numbers in the Montgomery domain. For improved performance of modular exponentiation operations, the *Sequencer* employs exponent recoding techniques that use a table with pre-calculated odd powers.

NOTE: The modular Montgomery multiplication and exponentiation operations and the direct interface of the *LNME* are not described further in this document, as the *LNME's* functionality is meant to be used by the *Sequencer* only.

22.1.2.2 Performance

Table 22-1 shows the performance numbers. The percentages indicate relative performance for the maximum clock frequencies, using the 32-bit *PKCP*-only version as baseline.

Table 22-1. Performance

Configuration		16-bit PKCP only
Max clock freq.		250 MHz
RSA-1024 performance	# clocks(1)	12.6 x 106
	At max freq.	50.4 ms(37%)
RSA-2048 performance(2)	# clocks(1)	93.0 x 106
	At max freq.	372 ms(27%)

(1) Using 4 pre-calculated odd powers. Timing is slightly data-dependent (a few percent margin around the given values is to be expected – mostly due to actual exponent vector value).

(2) 50% of the exponent bits are '1', no use of CRT: straight modular exponentiation using 1024/2048-bit modulus and exponent vectors (timing includes the necessary pre- and post-calculations).

Table 22-1. Performance (continued)

Configuration		16-bit PKCP only
RSA-CRT-1024 performance[3]	# clocks[1]	3.89 x 106
	At max f.[4]	15.6 ms(92%)
RSA-CRT-2048 performance[3]	# clocks[1]	25.0 x 106
	At max f.[4]	100 ms(103%)
ModInv-1024 performance	# clocks	719,587
	At max freq.	2.88 ms
ECC-ADD-384 perf.[5]	# clocks	145,655
	At max freq.	0.59 ms
ECC-MUL-384 performance[6]	# clocks	15.5 x 106
	At max freq.	62.1 ms(38%)

(3) 50% of the exponent bits are '1', CRT is used: two modular exponentiations, each using 512/1024-bit modulus and exponent vectors (timing includes input value splitting and output value recombination).

(4) Percentages are relative to the 32-bit *PKCP* only RSA-1024/2048 performance.

(5) Addition of points on a prime field elliptic curve. Added points differ – if they were the same a point doubling would be performed automatically. Prime modulus has 384 significant bits.

(6) Multiplication of a point on a prime field elliptic curve by a scalar value. Prime modulus and scalar value both have 384 significant bits.

22.1.3 Functional Description

22.1.3.1 Module Architecture

22.1.3.2 PKA RAM

The vectors (big numbers) that are the input and output of the PKA operations are stored in the *PKA RAM*. Each vector consists of a sequence of (32-bit) words, stored 'little endian' in a contiguous block of memory with the least significant word at the lowest address of that memory block and bit [0] of the vector in bit [0] of that word.

NOTE: All input and output vectors must start at an even 32-bit word address (in other words, must be aligned to an 8 byte boundary in *PKA RAM*).

The *PKA RAM* size is 2K.

22.1.3.3 PKCP Operations

Table 22-2 lists the arguments and results for each *PKCP* operation.

Table 22-2. Summary of PKCP Vector Operations

Function	MathematicalOperation	Vector A	Vector B	Vector C	Vector D
Multiply	A x B → C	Multiplicand	Multiplier	Product	N/A
Add	A + B → C	Addend	Addend	Sum	N/A
Subtract	A – B → C	Minuend	Subtrahend	Difference	N/A
AddSub	A + C – B → D	Addend	Subtrahend	Addend	Result
Right Shift	A >> Shift → C	Input	N/A	Result	N/A
Left Shift	A << Shift → C	Input	N/A	Result	N/A
Divide	A mod B → C, A div B → D	Dividend	Divisor	Remainder	Quotient
Modulo	A mod B → C	Dividend	Divisor	Remainder	N/A
Compare	A = B, A < B, A > B	Input1	Input2	N/A	N/A

Table 22-2. Summary of PKCP Vector Operations (continued)

Function	MathematicalOperation	Vector A	Vector B	Vector C	Vector D
Copy	A → C	Input	N/A	Result	N/A

To obtain correct result, the input vectors must meet the requirements presented in Table 22-3. Note that:

- Input restrictions are not checked by the *PKCP*
- A_Len and B_Len indicate the size of vectors A and B in (32-bit) words
- Max_Len equals 64 (32-bit) words, i.e. the standard maximum vector size is 2048 bit

NOTE: Maximum vector sizes can be optionally extended to 4096 or 8192 bits (with Max_Len equal to 128 respectively 256).

Table 22-3. Restrictions On Input Vectors for PKCP Operations

Operational Restrictions	
Function	Requirements
Multiply	0 < A_Len, B_Len ≤ Max_Len
Add	0 < A_Len, B_Len ≤ Max_Len
Subtract	0 < A_Len, B_Len ≤ Max_LenResult must be positive (A ≥ B)
AddSub	0 < A_Len ≤ Max_Len (B and C operands have A_Len as length, B_Len ignored)Result must be positive ((A + C) ≥ B)
Right Shift	0 < A_Len ≤ Max_Len
Left Shift	0 < A_Len ≤ Max_Len
Divide, Modulo	1 < B_Len ≤ A_Len ≤ Max_LenMost significant 32-bit word of B operand cannot be zero
Compare	0 < A_Len ≤ Max_Len (B operand has A_Len as length, B_Len ignored)
Copy	0 < A_Len ≤ Max_Len

The host is responsible for allocating a block of contiguous memory in *PKA RAM* for the result vector(s). The table below indicates how much memory should be allocated for the result vector(s).

Table 22-4. PKCP Result Vector Memory Allocation

Result Vector Memory Allocation		
Function	Result Vector	Result Vector Length (in 32-bit words)
Multiply	C	A_Len + B_Len + 6 (the 6 'scratchpad' words should be discarded)
Add	C	Max(A_Len, B_Len) + 1
Subtract	C	Max(A_Len, B_Len)
AddSub	D	A_Len + 1
Right Shift	C	A_Len
Left Shift	C	A_Len + 1 (when Shift Value is non-zero) A_Len (when Shift Value is zero)
Divide	C	Remainder → B_Len + 1 (one 'scratchpad' word should be discarded)
Divide	D	Quotient → A_Len – B_Len + 1
Modulo	C	Remainder → B_Len + 1
Compare	None	Compare updates the **PKA_COMPARE** register
Copy	C	A_Len

Input vectors for an operation are always allowed to overlap in memory (partially or completely). The table below gives restrictions for the overlap of output and input vectors of the operations.

Table 22-5. PKCP Result Vector / Input Vector overlap restrictions

Result Vector / Input Vector overlap restrictions		
Function	Result Vector	Restrictions
Multiply	C	No overlap with A or B vectors allowed
Add, Subtract	C	May overlap with A and/or B vector, provided the start address of the C vector does not lie above the start address of the vector(s) with which it overlaps
AddSub	D	May overlap with A, B and/or C vector, provided the start address of the D vector does not lie above the start address of the vector(s) with which it overlaps
Right Shift, Left Shift	C	May overlap with A vector, provided the start address of the C vector does not lie above the start address of the A vector
Divide	C	No overlap with A, B or D vectors allowed
Divide	D	No overlap with A, B or C vectors allowed
Modulo	C	No overlap with A or B vectors allowed
Compare	None	Compare does not write a result vector
Copy	C	Same restrictions as for Right/Left Shift, copy of a vector to a lower address is always allowed even if source and destination overlap(1)

(1) The Copy operation can be used to fill memory by breaking the overlap restrictions, but requires TWO initial (32-bit) words to be set up: To zero a block of memory, set A vector pointer to the block start, set C vector pointer two words higher and A vector length to the block length minus two (words). Fill the first two words of the block with constant zero and perform a PKCP Copy operation to zero the remainder of the block.

22.1.3.4 Sequencer Operations

22.1.3.4.1 Modular Exponentiation Operations

The *Sequencer* controls modular exponentiation operations.

Table 22-6. Summary of ExpMod Operations

Function	Mathematical operation	Vector A	Vector B	Vector C	Vector D
ExpMod-ACT2 ExpMod-ACT4 ExpMod-variable	$C^A \bmod B \rightarrow D$	Exponent, length = A_Len	Modulus, length = B_Len	Base, length = B_Len	Result & Workspace

Table 22-6. Summary of ExpMod Operations (continued)

Function	Mathematical operation	Vector A	Vector B	Vector C	Vector D
ExpMod-CRT	See below	Exp P followed by Exp Q at next higher even word address(1), both A_Len long	Mod P + buffer word followed by Mod Q at next higher even word address(2), both B_Len long	Q inverse, length = B_Len	Input, Result(both 2xB-Len long)& Workspace

(1) If A_Len is even, Exp Q follows Exp P immediately – if A_Len is odd, there is one empty word between Exp Q and Exp P.

(2) If B_Len is even, there are two empty words between Mod P and Mod Q – if B_Len is odd, there is one empty (buffer) word between Mod Q and Mod P. Note that the words following Mod P and Mod Q may be zeroed by *Sequencer* firmware.

The ExpMod-CRT operation performs the following computation steps (1):

- X ← (Input mod Mod P)Exp P mod Mod P
- Y ← (Input mod Mod Q)Exp Q mod Mod Q
- Z ← ((((X – Y) mod Mod P) * Q inverse) mod Mod P) * Mod Q
- Result ← Y + Z

The ExpMod-ACT2, -ACT4 and -variable functions implement the same mathematical operation but with a differently sized table with pre-calculated ‘odd powers’. The ExpMod-ACT2 function uses a table with 2 entries whereas ExpMod-ACT4 uses a table with 8 entries. The ACT4 version gives better performance but needs more memory.

ExpMod-variable and ExpMod-CRT allow a variable amount (from 1 up to and including 16) of odd powers to be selected via the register normally used to specify the number of bits to shift for shift operations.

For a user of the *PKA Engine*, the exponentiation functions appear to be extensions of the set of *PKCP* functions as described in Section 22.1.3.3. Input and result vectors are passed just like this is done for basic *PKCP* operations. Table 10 shows the restrictions on the input and result vectors for the exponentiation operations.

(1) These steps implement Garner’s recombination algorithm after the basic exponentiations.

Table 22-7. Restrictions on Input Vectors for ExpMod Operations

Operational Restrictions	
Function	Requirements
ExpMod-ACT2ExpMod-ACT4ExpMod-variable	1) 0 < A_Len ≤ Max_Len 2) 1 < B_Len ≤ Max_Len 3) Modulus B must be odd (i.e. the least significant bit must be ONE) 4) Modulus B > 232 5) Base C < Modulus B 6) Vectors B and C must be followed by an empty 32-bit ‘buffer’ word
ExpMod-CRT	1) 0 < A_Len ≤ Max_Len2) 1 < B_Len ≤ Max_Len 3) Mod P and Mod Q must be odd (i.e. the least significant bits must be ONE) 4) Mod P > Mod Q > 232 5) Mod P and Mod Q must be co-prime (their GCD must be 1) 6) 0 < Exp P < (Mod P – 1) 7) 0 < Exp Q < (Mod Q – 1) 8) (Q inverse * Mod Q) ≡ 1 (modulo Mod P) 9) Input < (Mod P * Mod Q) 10) Mod P and Mod Q must be followed by an empty 32-bit ‘buffer’ word

Table 22-8 shows the required scratchpad sizes for the exponentiation operations. The ‘M_Len’ used in the table is the ‘real’ Modulus length (for Mod P in an ExpMod-CRT operation, for Modulus B in the other operations) in 32-bit words, i.e. without trailing zero words at the end. If the last word of the modulus vector as given is non-zero, ‘M_Len’ equals B_Len.

Table 22-8. ExpMod Result Vector/Scratchpad Area Memory Allocation

Result Vector/Scratchpad Area Memory Allocation (both starting at PKA_DPTR)		
Function	PKA engine type	Scratchpad area size (in 32-bit words),Result Vector is either M_Len or 2xM_Len 32-bit words long
ExpMod-ACT2	PKCP-only	5 x (M_Len + 2)
ExpMod-ACT4	PKCP-only	11 x (M_Len + 2)
ExpMod-variable	PKCP-only	(# odd powers + 3) x (M_Len + 2)
ExpMod-CRT	PKCP-only	(# odd powers + 3) x (M_Len + 2) + (M_Len + 2 – (M_Len MOD 2))

NOTE: During execution of an ExpMod-ACT2, -ACT4 or -variable operation, the last 34 bytes of the *PKA RAM* are used as general scratchpad for the *Sequencer's* program execution. The ExpMod-CRT operation requires the last 72 bytes of the *PKA RAM* as scratchpad. These (fixed location) areas may not overlap with any of the input vectors and/or the D vector scratchpad area, they can be used freely when executing basic *PKCP* operations.

Table 22-9. ExpMod Scratchpad Area / Input Vector Overlap Restrictions

Result Vector / Input Vector overlap restrictions		
Function	ResultVector	Restrictions
ExpMod-ACT2ExpMod-ACT4ExpMod-variable	D	Scratchpad area starting at D may not overlap with any of the other vectors, except that Base C may be co-located with result vector D to save space(i.e. **PKA_CPTR** = **PKA_DPTR** is allowed).
ExpMod-CRT	D	Scratchpad area starting at D may not overlap with any of the other vectors, this is also the location of the main Input vector (with length 2 x B_Len)

22.1.3.4.2 PKA RAM Size Needed for Exponentiation Operations

For exponentiation operations, the minimum size of the *PKA RAM* depends on the maximum modulus length and the number of odd-powers. In addition, a fixed number of bytes are needed as scratchpad for the *Sequencer* firmware during execution of the exponentiation – this scratchpad must be located at the end of the *PKA RAM*.

NOTE: The *Sequencer* firmware allocates the 34 or 72 bytes of general scratchpad at the end of a (virtual) 8K Byte *PKA RAM* – due to address duplication, smaller *PKA RAM* sizes of 1K, 2K and 4K Byte will also hold this area at their highest addresses.

It is possible to generate a *PKA Engine* with a non-standard *PKA RAM* size, but in this case the actual size must be provided to SafeNet to allow correct parameterization of the hardware and firmware.

Table 22-10 indicates the required sizes of the *PKA RAM* to support different modulus lengths and numbers of odd powers (assuming the exponent length is the same as the modulus length). Overlapping the Base (**PKA_CPTR**) vector with the start of the Scratchpad/result (**PKA_DPTR**) area saves memory space for non-CRT operations and is therefore indicated separately:

Table 22-10. Required PKA RAM Sizes for Exponentiations

Modulus size(non-CRT)	1 odd power		> 1 odd power(add to '1 odd power' sizes)
	PKA_CPTR = PKA_DPTR	PKA_CPTR ≠ PKA_DPTR	
1024 bits	808 bytes	944 bytes	+ 136 bytes per extra odd power
2048 bits	1576 bytes	1840 bytes	+ 264 bytes per extra odd power
Scratchpad:	+ 34 bytes (fixed, at end of PKA RAM)		
CRT Moduli	1 odd power		> 1 odd power(add to '1 odd power' sizes)
2x512 bits	696 bytes		+ 72 bytes per extra odd power
2x1024 bits	1336 bytes		+ 136 bytes per extra odd power
Scratchpad:	+ 72 bytes (fixed, at end of PKA RAM)		

Table 22-11 indicates the maximum number of odd powers that can be used for different standard *PKA RAM* sizes and *PKA Engine* types (non-CRT operations using **PKA_CPTR** = **PKA_DPTR**):

Table 22-11. Maximum Number of Odd Powers for 2K Byte PKA RAM Size

PKA engine type	Operation	Modulus and Exponent sizes	Maximum number of odd powers for 2K Byte PKA RAM Sizes
With LNME	Non-CRT	1024 bits	11
		2048 bits	4
		4096 bits	N/A
	CRT	2x512 bits	16
		2x1024 bits	7
		2x2048 bits	N/A
PKCP-only	Non-CRT	1024 bits	9
		2048 bits	2
		4096 bits	N/A
	CRT	2x512 bits	16
		2x1024 bits	5
		2x2048 bits	N/A

- Using more than 8 odd powers is not advisable as the speed advantage for each extra odd power decreases rapidly (and can even become negative for short Exponent vector lengths due to the extra pre-processing required).
- The maximum amount of odd powers is 16 (limited by firmware). All '16 odd powers' entries in the table above hit this limit
  - they are not limited by the PKA RAM size.

22.1.3.4.3 Modular Inversion Operation

Besides modular exponentiation, the *Sequencer* also controls modular inversion operations.

Table 22-12 summarizes the ModInv operation.

Table 22-12. Summary of ModInv Operation

Function	Mathematical operation	Vector A	Vector B	Vector C	Vector D
ModInv	A-1 mod B → D	NumToInvert, length = A_Len	Modulus,length = B_Len	Not used	Result & Workspace

For a user of the *PKA Engine*, the above function appears to be an extension of the set of basic *PKCP*functions as described in section 3.3, with the following exceptions:

- Vector D not only addresses the Result but also a Workspace;
- The PKA_SHIFT register field is used to return info on the operation's result, see Table 22-13.

Table 22-13. PKA_SHIFT Result Values for ModInv Operation

PKA_SHIFT result values	
Function	PKA_SHIFT register field value at conclusion
ModInv	0 → success; VectorD holds result7 → no inverse exists (GCD(A, B)≠1, i.e. A and B have common factors); result undefined31 → error, Modulus even; result undefinedOther values are reserved

Table 22-14 and Table 22-15 list restrictions on the input and result vectors for the ModInv operation:

Table 22-14. Operational Restrictions on Input Vectors for the ModInv Operation

Operational Restrictions	
Function	Requirements
ModInv	0 < A_Len ≤ Max_Len0 < B_Len ≤ Max_LenModulus B must be odd (i.e. the least significant bit must be ONE) Modulus B may not have value 1 (result is undefined, no error indicated)The highest word of the modulus vector, as indicated by B_Len, may not be zero.

Table 22-15. ModInv Scratchpad Area / Input Vector Overlap Restrictions

Result Vector / Input Vectors overlap restrictions		
Function	ResultVector	Restrictions
ModInv	D	Scratchpad area starting at D may not overlap with any of the other vectors

Table 22-16 shows the required scratchpad sizes for the ModInv operation:

Table 22-16. ModInv Result Vector/Scratchpad Area Memory Allocation

Result Vector/Scratchpad Area Memory Allocation(both starting at PKA_DPTR)ε(*n*) = 2 + (*n* MOD 2), i.e. 2 (for *n* even) or 3 (for *n* odd)	
Function	Scratchpad area size (in 32-bit words), Result Vector is B_Length 32-bit words long
ModInv	5 x (M + ε(M)), with M = Max(A_Length, B_Length)

NOTE: During execution of a ModInv operation, the last 34 bytes of the *PKA RAM* are used as general scratchpad for the *Sequencer's* program execution. This (fixed location) area may not overlap with any of the input vectors and/or the D vector scratchpad area during execution.

22.1.3.4.4 Modular Inversion With an Even Modulus

The ModInv operation requires the modulus to be odd. At first, this appears to make the operation useless in the case of RSA key generation where the private key exponent *d* is derived from a chosen public exponent *e* as follows:

d = ModInv(e, φ)

where φ = (*p*-1)(*q*-1) and *p* and *q* both primeNote that φ is even. However, since *e* must be odd (otherwise no inverse exists), *d* can be calculated as:

d = (1 + (φ x (e – ModInv(φ, e))) / e

So with four additional basic *PKCP* operations, ModInv can also be used to find inverse values in case the modulus is even.

22.1.3.4.5 Modular Inversion With a Prime Modulus

Modular inversion can be performed with a modular exponentiation using the modulus value minus two as exponent, provided that the modulus value is a prime. This is due to the fact that(AM) mod M = A ⇒(AM-1) mod M = 1 ⇒(AM-2) mod M = A-1 (mod M)

Under the constraint that M is a prime value.

22.1.3.4.6 ECC Operations

Besides modular exponentiation and modular inversion, the *Sequencer* also controls ECC operations.

Table 22-17. Summary of ECC Operations

Function	Mathematical operation	Vector A	Vector B	Vector C	Vector D
ECC-ADD	Point addition/doubling7on elliptic curve:y2=x3+*ax*+*b* (mod *p*)pntA + pntC → pntD	8pntA.x followedby pntA.yboth B_Len long(A_Len not used)	Curve parameter *p* followed8 by *a* (*b* is not needed)all B_Len long	pntC.x followed8by pntC.y both B_Len long	Result, i.e. pntD.x followed8 by pntD.y&Workspace
ECC-MUL	Point multiplication on elliptic curve:y2=x3+*ax*+*b* (mod *p*)k x pntC → pntD	Scalar kA_Len long	Curve parameter *p* followed8 by *a* and *b*all B_Len long.	8pntC.x followedby pntC.yboth B_Len long	Result, i.e. pntD.x followed8 by pntD.y& Workspace

7 If pntA = pntC, a point doubling operation is performed automatically.

8 All input components must be located on a 64-bit boundary and must have **ε** extra 'buffer' words (of 32 bits each) after their most significant word. **ε**

must be **3** (B_Len odd) or **2** (B_Len even). Each result component (i.e. pntD.x, pntD.y) will also be followed by **ε** buffer (zero) words.

For a user of the *PKA Engine*, the above functions appear to be extensions of the set of *PKCP* functions as described in section 3.3, with the following exceptions:

- Input and result vectors can now be composite (i.e. consist of two or three equal-sized sub-vectors);
- Vector D not only addresses the Result but also a Workspace;
- The PKA_SHIFT register is used to return info on the operation's result, see Table 22-18

Table 22-18. PKA_SHIFT Result Values for ECC Operations

PKA_SHIFT result values	
Function	PKA_SHIFT register field value at conclusion
ECC-ADD ECC-MUL	0 → success; VectorD holds result point7 → result is "point-at-infinity"; VectorD result point undefined31 → error, (*p* not odd, *p* too short, etc); VectorD result point undefinedOther values are reserved

Table 22-19 and Table 22-20 list restrictions on the input and result vectors for the ECC operations.

Table 22-19. Operational Restrictions on Input Vectors for ECC Operations

Operational Restrictions	
Function	Requirements
ECC-ADD	1 < B_Len ≤ 24 (maximum vector length is 768 bits)Modulus *p* must be a prime > 263Effective modulus size (in bits) must be a multiple of 32 (1)The highest word of the modulus vector, as indicated by B_Len, may not be zero.a < p and b < ppntA and pntC must be on the curve (this is not checked)Neither pntA nor pntC can be the "point-at-infinity", although ECC-ADD can return this point as a result
ECC-MUL	0 < A_Len ≤ 24 (maximum vector length is 768 bits)1 < B_Len ≤ 24 (maximum vector length is 768 bits)Modulus *p* must be a prime > 263Effective modulus size (in bits) must be a multiple of 32 (1)The highest word of the modulus vector, as indicated by B_Len, may not be zero.a < p and b < ppntC must be on the curve (this is not checked)pntC cannot be the "point-at-infinity", although ECC-MUL can return this point as a result

(1): A few modulus lengths not adhering to this rule will lead to incorrect results – the 'standard' modulus lengths of 112 and 521 bits will work on all engines, so these lengths are the exceptions to this rule.

Table 22-20. ECC Scratchpad Area / Input Vector Overlap Restrictions

Result Vector / Input Vector overlap restrictions		
Function	ResultVector	Restrictions
ECC-ADD ECC-MUL	D	Scratchpad area starting at D may not overlap with any of the other vectors

Table 22-21 shows the required scratchpad sizes for the ECC operations:

Table 22-21. ECC Result Vector/Scratchpad Area memory Allocation

Result Vector/Scratchpad Area Memory Allocation (both starting at PKA_DPTR)ε(*n*) = 2 + (*n* MOD 2), i.e. 2 (for *n* even) or 3 (for *n* odd)	
Function	Scratchpad area size (in 32-bit words),Result Vector is 2x(B_Length+ ε(B_Length)) 32-bit words long
ECC-ADD	2 x L + 5 x M, where L = B_Length + ε(B_Length) , M = B_Length+1 + ε(B_Length+1)
ECC-MUL	18 x L + Max(8, L), where L = (B_Length + ε(B_Length))

NOTE: During execution of an ECC-ADD or ECC-MUL operation, the last 72 bytes of the *PKA RAM* are used as a general scratchpad for the *Sequencer's* program execution. These (fixed location) areas must not overlap with any of the input vectors and/or the D vector scratchpad area during execution.

22.1.4 Performance

22.1.4.1 Basic PKCP Operations Performance

The following table contains information on the number of clocks needed to perform the basic *PKCP* operations – this does not include the time needed to write input vectors into– and read results from *PKA RAM*:

Table 22-22. Basic PKCP Operations Performance

Operation (vector lengths in bits)	**# Clocks for 16-bit PKCP**	**Scaling to other vector sizes with… (approximate)**
1K + 1K addition	195	Max (A_Len, B_Len)
1K – 1K subtract	194	Max (A_Len, B_Len)

Table 22-22. Basic PKCP Operations Performance (continued)

Operation (vector lengths in bits)	# Clocks for 16-bit PKCP	Scaling to other vector sizes with… (approximate)
1K x 1K multiply	4210	A_Len x B_Len
2K / 1K divide/modulo	19241[(1)]	(A_Len – B_Len) x B_Len
1K + 1K – 1K add/sub	259	A_Len
1K shift left/right	133	A_Len
1K copy	128	A_Len
1K = 1K compare	133[(2)]	A_Len

(1) Slight variability in timing due to possibility of correction cycles being needed.

(2) Data dependent – maximum time given is needed only when vectors have same value or differ in the Least Significant 16/32- bit word only.

22.1.4.2 ExpMod Performance

In general, the following can be said – using 1 odd power as baseline performance of 100%:

- 2 odd powers (as used by ExpMod-ACT2) delivers approximately 112% performance
- 4 odd powers delivers approximately 121% performance
- 8 odd powers (as used by ExpMod-ACT4) delivers approximately 125% performance

These figures assume the pre-processing takes negligible time (which is true for the longer exponent lengths) and the use of random exponent vectors with 50% ones. Using more than 8 odd powers does not provide significant speed gains and is therefore not advisable.

NOTE: In general, modular exponentiations should be performed with four (4) odd powers – this is a good compromise between needed PKA RAM size and performance.

Due to the known exponent value (with lower than normal number of ONEs), using one odd power for'Verify' operations is actually faster than using four odd powers. The speed difference depends upon the configuration, but will normally be between 5 and 10%

Table 22-23 to provide measured performance figures for standard versions of the *PKA Engine*. The number of (clock) cycles does not include the transfer of data between the host and the *PKA Engine* and may vary a little with the actual vector values used.

Table 22-23. Exponentiation Performance for 16-bit PKCP-only

Operation	SW overhead	HW operations	Key (bits)	Modulus (bits)	#Odd powers	Performance (# cycles)	Time (us) @ 250 MHz	Ops/sec @ 250 MHz
DSA sign 160	host + seq	Exp.Mod.	160	512	1	697,280	2,789	359
DSA sign 160	host + seq	Exp.Mod.	160	512	4	634,414	2,538	394
DSA verify 160	host + seq	2 x Exp.Mod.	160	512	1	1,394,560	5,578	179
DSA verify 160	host + seq	2 x Exp.Mod.	160	512	4	1,268,828	5,075	197
DH key neg 180	host + seq	Exp.Mod.	180	1024	1	2,526,169	10,105	99
DH key neg 180	host + seq	Exp.Mod.	180	1024	4	2,244,311	8,977	111
RSA sign/encr/decr 512	sequencer	Exp.Mod.	512	512	1	2,299,238	9,197	109
RSA sign/encr/decr 512	sequencer	Exp.Mod.	512	512	4	1,959,458	7,838	128

Table 22-23. Exponentiation Performance for 16-bit PKCP-only (continued)

RSA sign/encr/decr 1024	sequencer	Exp.Mod.	1024	1024	1	14,779,279	59,117	16.9
RSA sign/encr/decr 1024	sequencer	Exp.Mod.	1024	1024	4	12,501,019	50,004	20
RSA sign/encr/decr 2048	sequencer	Exp.Mod.	2048	2048	1	111,325,297	445,301	2.25
RSA sign/encr/decr 2048	sequencer	Exp.Mod.	2048	2048	4	92,931,362	371,725	2.69
RSA sign/encr/decr 4096	sequencer	Exp.Mod.	4096	4096	1	833,993,111	3,335,972	0.3
RSA sign/encr/decr 4096	sequencer	Exp.Mod.	4096	4096	4	700,049,874	2,800,199	0.36
RSA sign/decr CRT 512	sequencer	2 x Exp.Mod.	256	256	1	704,406	2,818	355
RSA sign/decr CRT 512	sequencer	2 x Exp.Mod.	256	256	4	608,979	2,436	411
RSA sign/decr CRT 1024	sequencer	2 x Exp.Mod.	512	512	1	4,643,494	18,574	54
RSA sign/decr CRT 1024	sequencer	2 x Exp.Mod.	512	512	4	3,882,581	15,530	64
RSA sign/decr CRT 2048	sequencer	2 x Exp.Mod.	1024	1024	1	30,055,168	120,221	8.32
RSA sign/decr CRT 2048	sequencer	2 x Exp.Mod.	1024	1024	4	24,992,042	99,968	10
RSA sign/decr CRT 4096	sequencer	2 x Exp.Mod.	2048	2048	1	225,147,137	900,589	1.11
RSA sign/decr CRT 4096	sequencer	2 x Exp.Mod.	2048	2048	4	186,614,169	746,457	1.34
RSA verify 512	sequencer	Exp.Mod.	17	512	1	58,911	236	4244
RSA verify 1024	sequencer	Exp.Mod.	17	1024	1	188,527	754	1326
RSA verify 2048	sequencer	Exp.Mod.	17	2048	1	696,421	2,786	359
RSA verify 4096	sequencer	Exp.Mod.	17	4096	1	2,617,449	10,470	96
RSA verify 512	sequencer	Exp.Mod.	3	512	1	16,771	67	14907
RSA verify 1024	sequencer	Exp.Mod.	3	1024	1	52,184	209	4791
RSA verify 2048	sequencer	Exp.Mod.	3	2048	1	187,471	750	1334
RSA verify 4096	sequencer	Exp.Mod.	3	4096	1	698,057	2,792	358

notes:

1. this performance overview provides only the most commonly used configurations, other configurations are possible

2. the appendix shows more details about RSA, DH and DSA operations

3. for DH & CRT, the sequencer performs all SW operations inside the PkA engine

4. for DSA, the sequencer doesn't perform all operations inside the PKA engine,

for example random number generation has to be done via the host processor and an RNG

5. for CRT two exponentiations are done sequentialy, the cyclecount mentioned is the total of the cycles needed for these two operations and includes pre/post-processing

6. SW overhead on the host is NOT included in the cycle count. Most of the times this can be ignored,

but not in cases where the HW operations take little time

9 With enough *PKA RAM*, it is possible to use two separate sets of vectors – use one to read results and preload new vectors while the other is in use by the *PKA Engine*. A command to start processing can then be given (and the areas swap functionality) immediately after a previous command finishes.

22.1.4.3 Modular Inversion Performance

Modular inversion is slightly dependent upon the data values used, a few percent variability is to be expected. Table 22-24 below gives an average performance value over five different simulations:

Table 22-24. ModInv Performance

Vector lengths in bits for modulus and input	16-bit *PKCP*			
	#clocks	Ops/sec 230 MHz	Ops/sec 240 MHz	Ops/sec 250 MHz
128	29,226	1085	1093	1088
256	75,483	417	420	422
512	220,647	139	140	141
1024	719,587	43	44	44
2048	2,594,516	11	12	12
4096	9,812,581	3	3	3

22.1.4.4 ECC Operation Performance

ECC-ADD (which can perform a point addition as well as a point doubling) is slightly dependent upon the data values used (due to a modular inversion used in the calculations), less than two percent variability is to be expected:

Table 22-25. ECC-ADD Performance

Vector lengths in bits for all input values & operation		16-bit *PKCP*			
		#clocks	Ops/sec230 MHz	Ops/sec240 MHz	Ops/sec250 MHz
128	Point addition	33,714	949	949	949
128	Point doubling	34,749	921	921	921
160	Point addition	43,266	739	740	740
160	Point doubling	46,246	692	692	692
192	Point addition	56,403	567	567	567
192	Point doubling	58,229	549	549	545
224	Point addition	68,969	464	463	464
224	Point doubling	70,169	456	456	456
256	Point addition	84,140	380	380	380

Table 22-25. ECC-ADD Performance (continued)

Vector lengths in bits for all input values & operation		16-bit *PKCP*			
		#clocks	Ops/sec230 MHz	Ops/sec240 MHz	Ops/sec250 MHz
256	Point doubling	82,935	386	386	386
320	Point addition	110,963	288	288	288
320	Point doubling	115,581	277	277	277
384	Point addition	145,655	220	220	220
384	Point doubling	154,698	207	207	207

For ECC-MUL, a slight variability (of less than two percent) can be expected due to the three modular inversions performed at the end of the algorithm:

Table 22-26. ECC-MUL Performance

Vector lengths in bits for all input values	Performance in clock cycles and ops/sec	16 bits *PKCP*
128	# clocks	1.36 x106
128	Ops/sec	23
160	# clocks	1.90 x106
160	Ops/sec	17
192	# clocks	3.00 x106
192	Ops/sec	11
224	# clocks	4.67 x106
224	Ops/sec	7
256	# clocks	5.51 x106
256	Ops/sec	6
320	# clocks	10.3 x106
320	Ops/sec	3
384	# clocks	15.5 x106
384	Ops/sec	2

22.1.5 Interfaces

22.1.5.1 Functional Interface

22.1.5.1.1 External Interface Address Map

22.1.5.1.2 PKA Engine Control Registers

The following sections describe the *PKCP* registers in detail. Since these only registers used to control the actual *PKA Engine* operation, they are labeled as '*PKA*' registers. Also, one register controlling the *Sequencer* is described, but this is only needed for the *Sequencer program RAM* option.

22.1.5.1.3 PKA Vector_A Address (PKA_APTR)

During execution of basic PKCP operations14, this register is double buffered and can be written with a new value for the next operation - when not written, the value will remain intact. During the execution of *Sequencer* controlled complex operations, this register may not be written and its value is undefined at conclusion of the operation. The driver software can not rely on the written value to remain intact.

Table 22-27. PKA_APTR

PKA_APTR, (Read/Write), 6-bit word offset in control register space: 0x00																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																					APTR										
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits [10:0], **APTR:**

This register specifies the location of Vector A within the *PKA RAM*. Vectors are identified through the location of their least-significant 32-bit word. Note that bit 0 must be zero to ensure that the vector starts at an 8-byte boundary.

Bits [31:11], **Reserved:** Set to zero on Write, ignore on Read.

22.1.5.1.4 PKA Vector_B Address (PKA_BPTR)

During execution of basic PKCP operations [(1)], this register is double buffered and can be written with a new value for the next operation - when not written, the value will remain intact. During the execution of *Sequencer* controlled complex operations, this register may not be written and its value is undefined at conclusion of the operation. The driver software can not rely on the written value to remain intact.

(1) These are the Add, Subtract, AddSub, Multiply, Divide, Modulo, Lshift, Rshift, Copy and Compare operations selected directly with bits in the PKA_FUNCTION register.

Table 22-28. PKA_BPTR

PKA_BPTR, (Read/Write), 6-bit word offset in control register space: 0x01																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																					BPTR										
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits [10:0], **BPTR:** This register specifies the location of Vector B within the *PKA RAM*. Vectors are identified through the location of their least-significant 32-bit word. Note that bit 0 must be zero to ensure that the vector starts at an 8-byte boundary.

Bits [31:11], **Reserved:** Set to zero on Write, ignore on Read.

22.1.5.1.5 PKA Vector_C Address (PKA_CPTR)

During execution of basic PKCP operations14, this register is double buffered and can be written with a new value for the next operation - when not written, the value will remain intact. During the execution of *Sequencer* controlled complex operations, this register may not be written and its value is undefined at conclusion of the operation. The driver software can not rely on the written value to remain intact.

Table 22-29. PKA_CPTR

PKA_CPTR, (Read/Write), 6-bit word offset in control register space: 0x02																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																					CPTR										
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits [10:0], **CPTR:** This register specifies the location of Vector C within the *PKA RAM*. Vectors are identified through the location of their least-significant 32-bit word. Note that bit 0 must be zero to ensure that the vector starts at an 8-byte boundary.

Bits [31:11], **Reserved:** Set to zero on Write, ignore on Read.

22.1.5.1.6 PKA Vector_D Address (PKA_DPTR)

During execution of basic *PKCP* operations (1), this register is double buffered and can be written with a new value for the next operation - when not written, the value will remain intact. During the execution of *Sequencer* controlled complex operations, this register may not be written and its value is undefined at conclusion of the operation. The driver software can not rely on the written value to remain intact.

(1) These are the Add, Subtract, AddSub, Multiply, Divide, Modulo, Lshift, Rshift, Copy and Compare operations selected directly with bits in the PKA_FUNCTION register.

Table 22-30. PKA_DPTR

PKA_DPTR, (Read/Write), 6-bit word offset in control register space: 0x03																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																					DPTR										
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits [10:0], **DPTR:** This register specifies the location of Vector D within the *PKA RAM*. Vectors are identified through the location of their least-significant 32-bit word. Note that bit 0 must be zero to ensure that the vector starts at an 8-byte boundary.

Bits [31:11], **Reserved:** Set to zero on Write, ignore on Read.

22.1.5.1.7 PKA Vector_A Length (PKA_ALENGTH)

During execution of basic PKCP operations15, this register is double buffered and can be written with a new value for the next operation - when not written, the value will remain intact. During the execution of *Sequencer* controlled complex operations, this register may not be written and its value is undefined at conclusion of the operation. The driver software can not rely on the written value to remain intact.

Table 22-31. PKA_ALENGTH

PKA_ALENGTH, (Read/Write), 6-bit word offset in control register space: 0x04																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																							ALENGTH								
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits [8:0], **ALENGTH:** This register specifies the length (in 32-bit words) of Vector A.

Bits [31:9], **Reserved:** Set to zero on Write, ignore on Read.

22.1.5.1.8 PKA Vector_B Length (PKA_BLENGTH)

During execution of basic PKCP operations (1), this register is double buffered and can be written with a new value for the next operation - when not written, the value will remain intact. During the execution of *Sequencer* controlled complex operations, this register may not be written and its value is undefined at conclusion of the operation. The driver software can not rely on the written value to remain intact.

(1) These are the Add, Subtract, AddSub, Multiply, Divide, Modulo, Lshift, Rshift, Copy and Compare operations selected directly with bits in the PKA_FUNCTION register.

Table 22-32. PKA_BLENGTH

PKA_BLENGTH, (Read/Write), 6-bit word offset in control register space: 0x05

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

Bits [8:0], **BLENGTH:** This register specifies the length (in 32-bit words) of Vector B.

Bits [31:9], **Reserved:** Set to zero on Write, ignore on Read.

22.1.5.1.9 PKA Bit Shift Value (PKA_SHIFT)

For basic *PKCP* operations, modifying the contents of this register is made impossible while the operation is being performed. For the ExpMod-variable and ExpMod-CRT operations, this register is used to indicate the number of odd powers to use (directly as a value in the range 1-16). For the ModInv and ECC operations, this register is used to hold a completion code.

Table 22-33. PKA_SHIFT

PKA_SHIFT, (Read/Write), 6-bit word offset in control register space: 0x06

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																											# bits to shift				
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits [4:0], **#bits_to_shift:** This register specifies the number of bits to shift the input vector (in the range 0-31) during a Rshift or Lshift operation.

Bits [31:5], **Reserved:** Set to zero on Write, ignore on Read.

22.1.5.1.10 PKA Function (PKA_FUNCTION)

This register contains the control bits to start basic *PKCP* as well as complex *Sequencer* operations. The'**Run**' bit can be used to poll for the completion of the operation. Modifying bits [11:0] is made impossible during the execution of a basic *PKCP* operation.During the execution of *Sequencer* controlled complex operations, this register is modified - the 'Run' and 'Stall result' bits are set to zero at the conclusion, but other bits are undefined.

Table 22-34. PKA_FUNCTION

PKA_FUNCTION, (Read/Write), 6-bit word offset in control register space: 0x07

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED							Stall result	RESERVED								Run	Sequencer Operations			Copy	Compare	Molulo	Divide	Lshift	Rshift	Subtract	Add	MS one	RESERVED	AddSub	Multiply
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit [0], **Multiply:** Perform multiply operation.

Bit [1], **AddSub:** Perform combined Add/Subtract operation. Bit [2], **Reserved:** Set to zero on Write, ignore on Read.

Bit [3], **MS one:** Loads the location of the Most Significant one bit within the result word indicated in the **PKA_MSW** register into bits [4:0] of the **PKA_DIVMSW** register – can only be used with basic *PKCP* operations [(1)], except for Divide, Modulo and Compare.

(1) These are the Add, Subtract, AddSub, Multiply, Divide, Modulo, Lshift, Rshift, Copy and Compare operations selected directly with bits in the PKA_FUNCTION register.

Bit [4], **Add:** Perform add operation.

Bit [5], **Subtract:** Perform subtract operation.

Bit [6], **RShift:** Perform right shift operation. Bit [7], **LShift:** Perform left shift operation. Bit [8], **Divide:** Perform divide operation.

Bit [9], **Modulo:** Perform modulo operation. Bit [10], **Compare:** Perform compare operation. Bit [11], **Copy:** Perform copy operation.

Bits [14:12], **Sequencer Operations:**

These bits select the complex *Sequencer* operation to perform:000b = None001b = ExpMod-CRT010b = ExpMod-ACT4[2]011b = ECC-ADD (if available in firmware, otherwise: reserved)100b = ExpMod-ACT2101b = ECC-MUL (if available in firmware, otherwise: reserved)110b = ExpMod-variable111b = ModInv (if available in firmware, otherwise: reserved)The encoding of these operations is determined by *Sequencer* firmware.Bit [15], **Run:** The host sets this bit to instruct the PKA module to begin processing the basic *PKCP* or complex *Sequencer* operation. This bit is reset low automatically when the operation is complete. The complement of this bit is output as **interrupts[1]**.After a reset, the **Run** bit is always set to 1. Depending on the option, *Program ROM* or *Program RAM*, the following applies:

Program ROM - The first Sequencer instruction sets the bit to 0. This is done immediately after the hardware reset is released.

Program RAM - The *Sequencer* needs to set the bit to 0. As a valid firmware may not have been loaded, the *Sequencer* is held in software reset after the hardware reset is released (the **Reset** bit in PKA_SEQ_CRTL is set to 1). After the FW image is loaded and the **Reset** bit is cleared, the *Sequencer* starts to execute the FW. The first instruction clears the **Run** bit.In both cases a few clock cycles are needed before the first instruction is executed and the **Run** bit state has been propagated.

Bits [23:16], **Reserved:** Set to zero on Write, ignore on Read.

Bit [24], Stall result: When written with a '1', updating of the PKA_COMPARE, PKA_MSW and PKA_DIVMSW registers, as well as resetting the Run bit is stalled beyond the point that a running operation is actually finished. Use this to allow software enough time to read results from a previous operation when the newly started operation is known to take only a short amount of time. If a result is waiting, the result registers is updated and the Run bit is reset in the clock cycle following writing the Stall Result bit back to '0'.

The **Stall result** function may only be used for basic *PKCP* operations. [3]

Bits [31:25], **Reserved:** Set to zero on Write, ignore on Read.

WARNING: continuously reading this register to poll the **Run** bit is NOT allowed when executing complex *Sequencer* operations (the *Sequencer* cannot access the *PKCP* when this is done).Leave at least one **sysclk** cycle between poll operations!

22.1.5.1.11 PKA Compare Result (PKA_COMPARE)

This register provides the result of a basic *PKCP* Compare operation. It is updated when the 'Run' bit in the **PKA_FUNCTION** register is reset at the end of that operation. Status after a complex *Sequencer* operation is unknown.

[2] ExpMod-ACT2 and ExpMod-ACT4 functions will be dropped in the future – recommended to use the equivalent ExpMod-variable commands.

[3] These are the Add, Subtract, AddSub, Multiply, Divide, Modulo, Lshift, Rshift, Copy and Compare operations selected directly with bits in the PKA_FUNCTION register.

Table 22-35. PKA_COMPARE

PKA_COMPARE, (Read Only), 6-bit word offset in control register space: 0x08																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																													A > B	A < B	A = B
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Bit [0], **A=B:** Vector_A is equal to Vector_B. Bit [1], **A<B:** Vector_A is less than Vector_B.

Bit [2], **A>B:** Vector_A is greater than Vector_B.

Bits [31:3], **Reserved:** Ignore on Read.

22.1.5.1.12 PKA Most-Significant-Word of Result Vector (PKA_MSW)

This register indicates the (word) address in the *PKA RAM* where the most significant non-zero 32-bit word of the result is stored. Should be ignored for Modulo operations. For basic *PKCP* operations, this register is updated when the '**Run**' bit in the **PKA_FUNCTION** register is reset at the end of the operation. For the complex *Sequencer* controlled operations, updating of the final value matching the actual result is done near the end of the operation – note that the result is only meaningful if no errors were detected and that for ECC operations, the PKA_MSW register will provide information for the x- coordinate of the result point only.

Table 22-36. PKA_MSW

PKA_MSW, (Read Only), 6-bit word offset in control register space: 0x09																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																Result is Zero	RESERVED				Msw_Address										
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits [10:0], **Msw_Address:** Address of the most significant non-zero 32-bit word of the result vector in PKA RAM.

Bits [14:11], **Reserved:** Ignore on Read.

Bit [15], Result Is Zero: The result vector is all zeroes, ignore the address returned in bits 0-10. Bits [31:16], Reserved: Ignore on Read.

22.1.5.1.13 PKA Most-Significant-Word of Divide Remainder (PKA_DIVMSW)

This register indicates the (32-bit word) address in the *PKA RAM* where the most significant non-zero32-bit word of the Remainder result for the basic Divide and Modulo operations is stored. Bits [4:0] are loaded with the bit number of the most significant non-zero bit in the most significant non-zero word when '**MS one**' control bit is set. For Divide, Modulo and '**MS one**' reporting, this register is updated when the '**Run**' bit in the **PKA_FUNCTION** register is reset at the end of the operation. For the complex *Sequencer* controlled operations, updating of bits [4:0] of this register with the actual result's most significant bit location is done near the end of the operation – note that the result is only meaningful if no errors were detected and that for ECC operations, the PKA_DIVMSW register will provide information for the x-coordinate of the result point only.

Table 22-37. PKA_DIVMSW

PKA_DIVMSW, (Read Only), 6-bit word offset in control register space: 0x0A																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																Result is Zero	RESERVED				Msw_Address										
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits [10:0], **Msw_Address:** Address of the most significant non-zero 32-bit word of the Remainder result vector.

Bits [14:11], **Reserved:** Ignore on Read.

Bit [15], Result Is Zero: The result vector is all zeroes, ignore the address returned in bits 0-10.

Bits [31:16], Reserved: Ignore on Read.

22.1.5.1.14 PKA Sequencer Control/Status Register (PKA_SEQ_CTRL)

The sequencer is interfaced with the 'outside world' through a single control/status register. With the exception of bit [31], the actual use of bits in the separate sub-fields of this register is determined by the *Sequencer* firmware. This register need only be accessed when the *Sequencer* program is stored in RAM. The reset value 'P' depends upon the option chosen for *Sequencer* program storage.

Table 22-38. PKA_SEQ_CTRL

PKA_SEQ_CTRL, (Read/Write), 6-bit word offset in control register space: 0x32																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	RESERVED															Sequencer status (read-only)								SW control/triggers (read/set-only)							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits [7:0], **SW control/triggers:** These bits can be used by software to trigger *Sequencer* operations. External logic can set these bits by writing a 1, can not reset them by writing a 0. The Sequencer can reset these bits by writing a 0, can not set them by writing a 1. Setting the '**Run**' bit in **PKA_FUNCTION** together with a non-zero '*Sequencer* operations' field automatically sets bit [0] here. *This field should always be written with zeroes and ignored when reading this register.*

Bits [15:8], **Sequencer status:** These read-only bits can be used by the *Sequencer* to communicate status to the outside world. Bit [8] is also used as *Sequencer* interrupt, with the complement of this bit OR-ed into the '**Run**' bit in **PKA_FUNCTION**. *This field should always be written with zeroes and ignored when reading this register.*

Bits [30:16], **Reserved:** Bits should be written with a '0' and should be ignored on a read.

The function (and reset state) of bit [31] depends upon the *Sequencer* program storage option chosen.

When using a Sequencer program ROM, the following applies:

Bit [31], **Reset:** Read/Write, reset value '0' (ZERO). Writing '1' will reset the *Sequencer*, write to '0' to restart operations again. As the reset value is '0', the *Sequencer* will automatically start operations executing from program ROM. *This bit should always be written with zero and ignored when reading this register.*

When using a Sequencer program RAM, the following applies:

Bit [31], **Reset:** Read/Write, reset value '1' (ONE). When '1', the *Sequencer* is held in a reset state and the **PKA_PROGRAM** area is accessible for loading the sequencer program (while the **PKA_DATA_RAM** is inaccessible), write to '0' to (re)start *Sequencer* operations and disable **PKA_PROGRAM** area accessibility (also enables the **PKA_DATA_RAM** accesses). *Resetting the Sequencer (in order to load other firmware) should only be done when the PKA Engine is not performing any operations (i.e. the* '**Run**' *bit in the* **PKA_FUNCTION** *register should be zero).*

22.1.5.1.15 PKA HW Options Register (PKA_OPTIONS)

This register provides the *Host* with a means to determine the hardware configuration implemented in this *PKA Engine* – focused on options that have an effect on software interacting with the module.

Table 22-39. PKA_OPTIONS

PKA_OPTIONS, (Read only), 6-bit word offset in control register space: 0x3D

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED								RESERVED		RESERVED										Int. masking	RESERVED									PKCP configuration	
?	?	?	?	?	?	?	?	0	0	?	?	?	?	?	?	0	0	0	0	?	0	?	?	?	1	0	0	0	?	0	1

Bits [1:0] – **PKCP configuration:** Value 1 indicates a *PKCP* with a 16x16 multiplier; other values reserved.

Bit [11] – Int. masking: Value 0 indicates that the main interrupt output (bit [1] of the 'interrupts' output bus) is the direct complement of the 'Run' bit in the 'PKA_CONTROL' register, value 1 indicates that interrupt masking logic is present for this output.

Reserved: Bits should be ignored on a read.

22.1.5.1.16 PKA Firmware Revision and Capabilities Register (PKA_SW_REV)

This register allows the *Host* access to the internal firmware revision number of the *PKA Engine* for software driver matching and diagnostic purposes. This register also contains a field that encodes the capabilities of the embedded firmware.The PKA_SW_REV register is written by the firmware within a few clock cycles after starting up that firmware. The hardware reset value is zero, indicating that the information has not been written yet.

Table 22-40. PKA_SW_REV

PKA_SW_REV, (Read only), 6-bit word offset in control register space: 0x3E

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Firmware capabilities				Major FW revision				Minor FW revision				FW patch level				RESERVED															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits [15:0], **Reserved:** Bits should be ignored on a read.

Bits [19:16], **FW patch level:** 4 bits binary encoding of the firmware patch level, initial release will carry value zero.

Patches are used to remove bugs without changing the functionality or interface of a module – see therelease notes delivered with the package.

Bits [23:20], **Minor FW revision:** 4 bits binary encoding of the minor firmware revision number. Bits [27:24], **Major FW revision:** 4 bits binary encoding of the major firmware revision number. Bits [31:28], **Firmware capabilities:** 4 bits binary encoding for the functionality implemented in the

firmware. Value 0 indicates basic ModExp with/without CRT. Value 1 adds Modular Inversion, value 2 adds Modular Inversion and ECC operations. Values 3 – 15 are reserved.

22.1.5.1.17 PKA HW Revision Register (PKA_REVISION)

This register allows the *Host* access to the hardware revision number of the *PKA Engine* for software driver matching and diagnostic purposes. It is always located at the highest address in the access space of the module and contains an encoding of the EIP number (with its complement as 'signature') for recognition of the hardware module.

Table 22-41. PKA_REVISION

PKA_REVISION, (Read only), 6-bit word offset in control register space: 0x3F																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED				Major HW revision				Minor HW revision				HW patch level				Complement of basic EIP number								Basic EIP number							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits [7:0], **Basic EIP number:** 8 bits binary encoding of the EIP number, PKA gives 0x1C.

Bits [15:8], **Complement of basic EIP number:** bit-by-bit logic complement of bits [7:0], PKA gives 0xE3.

Bits [19:16], **HW patch level:** 4 bits binary encoding of the hardware patch level, initial release will carry value zero.

Patches are used to remove bugs without changing the functionality or interface of a module – see the release notes delivered with the package.

Bits [23:20], **Minor HW revision:** 4 bits binary encoding of the minor hardware revision number.

Bits [27:24], **Major HW revision:** 4 bits binary encoding of the major hardware revision number.

Bits [31:28], **Reserved:** Bits should be ignored on a read.

22.1.5.1.18 PKA Vector Ram (PKA_RAM)

The *PKA RAM* is accessible in the top half of the *PKA Engine* address space. If the *Sequencer program* is stored in RAM, this *PKA RAM* starts in the address space at the same location of the *PKA program RAM*. The size of the *PKA RAM* can be chosen by the customer (with standard supported sizes of 1K,2K, 4K and 8K Byte):

Table 22-42. PKA_RAM

PKA_RAM																															
(Read/Write)																															
Word addresses in PKA Engine address space: (address space byte size / 8) .. ((address space byte size / 8) + (PKA RAM byte size / 4) – 1)																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits [31:0], **Data:** Vector Data. The last 32/48 bytes (8/12 words of 32 bits) of *PKA RAM* are used as *Sequencer* scratchpad during execution of ExpMod operations. This *PKA RAM* is not accessible when the *Sequencer program* is stored in RAM and the '**Reset**' control bit in the **PKA_SEQ_CTRL** register is '1' (see Section 22.1.5.1.14).

22.1.5.1.19 Sequencer Program RAM (PKA_PROGRAM)

This section is only applicable if the option is chosen to store the *Sequencer* program in RAM.The *Sequencer program RAM* is accessible in the top half of the *PKA Engine* address space. This RAM starts in the address space at the same location of the *PKA RAM*. The (functional) width of the *Sequencer program RAM* must be 24 bits, as that is the number of bits in *Sequencer* instructions. The customer can choose the number of words, although at least 1536 words must be available to store the full-function operation firmware.

Table 22-43. PKA_PROGRAM

PKA_PROGRAM
(Read/Write)
Word addresses in PKA Engine address space: (address space byte size / 8) .. ((address space byte size / 8) + Program RAM word size – 1)

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

Bits [23:0], **Sequencer instructions:** The program executed by the *Sequencer,* to be loaded with the firmware image before resetting the '**Reset**' control bit in the **PKA_SEQ_CTRL** register to '0' (see Section 22.1.5.1.14section 5.4.14). This *Sequencer program RAM* is only accessible while this **Reset** control bit is '1'.

Bits [31:24], **Reserved:** Bits should be written with a '0' and should be ignored on a read.

22.1.5.2 Operation Sequences

Figure 22-1 below shows several operation sequences that can be used to operate the *PKA Engine*. Interface bus transactions are described in a stylized way, as is the behavior of the main interrupt output (bit [1] of the '**interrupts**' output bus).

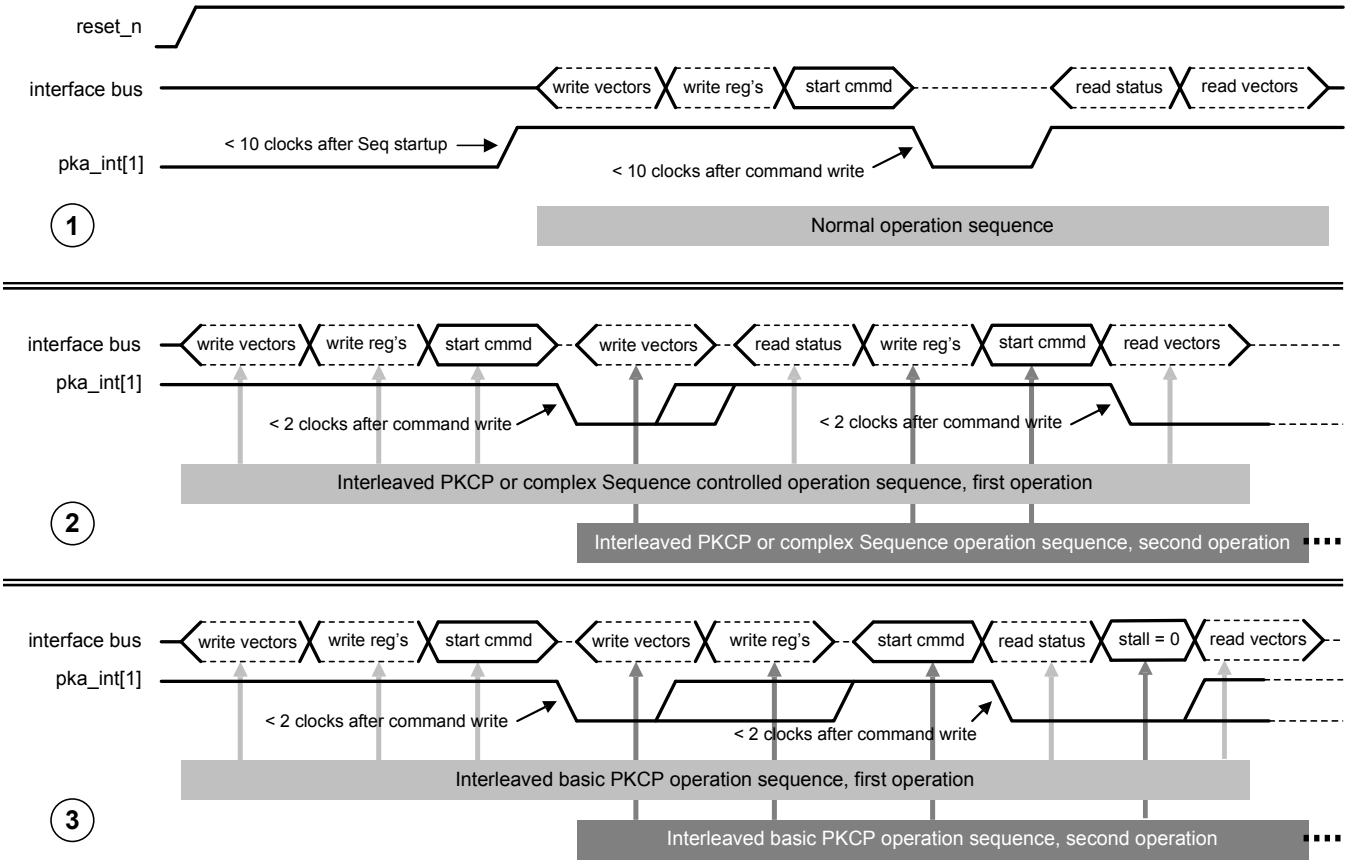


Figure 22-1. Operation Sequences and Main Interrupt

The left hand side of the top sequence (1) shows startup behavior:

The main interrupt is inactive (LOW) during module reset, going active (HIGH) within ten module clock cycles after starting up the *Sequencer program*. In case a program ROM is used, the *Sequencer* isstarted up immediately when the module reset is released. For program RAM equipped engines, the

Sequencer firmware must first be loaded and then the Sequencer taken out of reset (write a '0' to bit

[31] of the PKA_SEQ_CTRL register) to start the *Sequencer* program.

The right hand side of the top sequence (1) shows the normal operation sequence:

A normal operation sequence starts by writing input vectors in *PKA data RAM* and vector pointers and length values to *PKA Engine* control registers (in principle, this can be done in any order). The actualoperation is started with a write to the PKA_FUNCTION register, which results in dropping the maininterrupt output inactive within 2 clock cycles after setting the 'Run' bit. When the *PKA Engine* has finished execution the requested operation, the main interrupt is activated again and result status can be read from status registers (if needed), while the result vector(s) can be read from *PKA data RAM*.

The middle sequence (2) shows a more optimized interleaved operation sequence:

When enough *PKA data RAM* is available, separate areas in that RAM can be used for interleaving the input vector writes and result vector reads. In the example, the input vectors for the second operation arewritten while the first operation is in execution. Writing of the pointer and length registers and actual starting of the second command is done before the result vector(s) of the first command are read from the PKA data RAM. Writing the input vectors for a third operation can be done immediately following the reading of the first result, all during the time that the second operation is in execution.

The bottom sequence (3) shows a highly optimized basic PKCP operations sequence:

For basic *PKCP* operations, the vector pointer and length registers are double buffered – they may be written while an operation is in progress. This allows an even more optimized interleaving of busaccesses with writing the vectors and pointer/length registers for the second operation while the first operation is in execution. Upon the interrupt activation, only the command of the second operation need be written to start it up. The diagram shows that the status of the first operation is read *after* the second command is started up – to make sure that this status is not changed by an early finishing of the secondoperation, that second operation must be started with the '**Stall result**' bit (bit [24] of thePKA_FUNCTION register) written '1'. After reading the status, the '**Stall result**' bit *must* be reset to allow the status to be updated. If the first operation has no status to check, setting the '**Stall result**' bit is not needed.

22.1.6 Appendix A: RSA, ELGAMAL, DH, AND DSA Use Cases

22.1.6.1 A1: RSA Use Cases

The RSA (Rivest-Shamir-Adleman) algorithm can be used both for public key data encryption/decryption as well as public key signature generation/verification. Both require the same public key generation operations, which only have to be done once:

RSA public key generation (one time only):

- Generate random primes *p* and *q*, roughly the same size
- Compute $n = p \cdot q$ and $\varphi = (p-1) \cdot (q-1)$
- Select a random integer *e*, with $1 < e < \varphi$ and $gcd(e,\varphi) \equiv 1$
- Calculate *d*, such that $(e \cdot d) \bmod \varphi \equiv 1$
- The public key is the pair (*n*,*e*), the private key is *d*

If the Chinese Remainders Theorem ('CRT') is used:

- Calculate q*inv*, such that $q_{inv} \cdot q \bmod p \equiv 1$
- Calculate $dp = d \bmod (p-1)$ and $dq = d \bmod (q-1)$
- The private key is the quintuplet (*p*,*q*,*dp*,*dq*,*q inv*), all values of the same length (half the number ofbits of *n*)

  For RSA public key signature use, *e* can be chosen as a small number – normally 216+1 is used, but other values are also possible

RSA public key data encryption of a plaintext message *M* is done as follows:

- Represent message *M* as a value < *n* (may have to split if message is too long)
- Compute encrypted message $m = M^e \bmod n$ M, e and n are all of the same length, so this is a ModExp operation where the modulus and exponent lengths are the same

RSA private key data decryption of an encrypted message *m* without CRT is done as follows:

- Compute plaintext message value $M = m^d \bmod n$
- Extract actual message string from value *M* m, d and n are all of the same length, so this is a ModExp operation where the modulus and exponent lengths are the same

RSA private key data decryption of an encrypted message *m* with CRT is done as follows:

- Compute values $M1 = m^{dp} \bmod p$ and $M2 = m^{dq} \bmod q$
- Use Garner's recombination algorithm to reconstruct message value *M* from *M 1* and *M 2*, using *p*, *q*and *q inv*
- Extract actual message string from value *M*

  Both modular exponentiations and Garner's recombination algorithm are performed directly by theExpMod-CRT operation

An RSA private key signature of a plaintext message *M* is generated as follows without CRT:

- Add redundancy to the message (for instance, attach the hash of the message to the message string) and convert the redundant message to message value M, which must be < *n*
- Compute signature $s = M^d \bmod n$

M, d and n are all of the same length, so this is a ModExp operation where the modulus and exponent lengths are the same – adding redundancy to the message must be done by a host processor

An RSA private key signature of a plaintext message *M* is generated as follows with CRT:

- Add redundancy to the message (for instance, attach the hash of the message to the message string) and convert the redundant message to message value M, which must be < *n*
- Compute values *s 1* = *M dp* mod *p* and *s 2* = *M dq* mod *q*
- Use Garner's recombination algorithm to generate signature value *s* from *s 1* and *s 2*, using *p*, *q* and q*inv*

  Both modular exponentiations and Garner's recombination algorithm are performed directly by theExpMod-CRT operation – adding redundancy to the message must be done by a host processor

RSA public key signature verification of a signature *s* is done as follows:

- Compute redundant message value *M* = *s e* mod *n*
- Verify the redundancy of the message value *M* and reject the signature if the redundant part does notmatch the original plaintext message that is also contained in *M*
- Extract the original plaintext message from the redundant message value *M*

  s and n are of the same length, but e is normally chosen as a small (and fixed) value so this is a ModExp operation where the modulus and exponent lengths are different – checking redundancy and extracting the plaintext message must be done by a host processor

22.1.6.2 A2: Diffie-Hellman Use Cases

Basic Diffie-Hellman key agreement requires the following preparations (one-time setup):

- Select a prime *p*
- Calculate a 'generator' α for Z*p** with 2 ≤ α ≤ *p*–2
- Distribute the pair (*p*,α) to both parties wishing to perform a key agreement operation

  Both *p* and α are normally chosen as long vectors with the same amount of bits – the actual lengthmay vary according to security requirements

The actual (basic) Diffie-Hellman key exchange between parties 'A' and 'B' proceeds as follows:

- 'A' chooses a random secret *x* and sends the value α*x* mod *p* to 'B'
- 'B' chooses a random secret *y* and sends the value α*y* mod *p* to 'A'
- 'B' receives α*x* mod *p* from 'A' and calculates (α*x* mod *p*)*y* mod *p* ≡ α*xy* mod *p*
- 'A' receives α*y* mod *p* from 'B' and calculates (α*y* mod *p*)*x* mod *p* ≡ α*yx* mod *p*
- As α*xy* mod *p* equals α*yx* mod *p*, both 'A' and 'B' now have a 'shared secret' they can use as key fornon-public encryption and decryption

  Diffie-Hellman key exchanges normally use 180 bits values for *x* and *y* – each of the communicating parties must perform two ExpMod operations with an 180 bits exponent and base/modulus (*p* and α)vectors as distributed during setup

22.1.6.3 A3: ElGamal Use Cases

The ElGamal key agreement protocol is similar but non-symmetric and requires only one message to be sent.

ElGamal key agreement protocol requires the following preparations (one-time setup):

- 'B' selects a prime *p* and calculates a 'generator' α for Z*p** with 2 ≤ α ≤ *p*–2
- 'B' chooses a random secret *b* with 1 ≤ *b* ≤ *p*–2 and calculates α*b* mod *p*
- 'B' publishes the public key triplet (*p*,α,α*b* mod *p*) and keeps *b* private

The actual ElGamal key agreement between parties 'A' and 'B' proceeds as follows:

- 'A' chooses a random secret *x* with 1 ≤ *x* ≤ *p*–2 and sends the value α*x* mod *p* to 'B'
- 'A' calculates (α*b* mod *p*)*x* mod *p* ≡ α*bx* mod *p*
- 'B' receives α*x* mod *p* from 'A' and calculates (α*x* mod *p*)*b* mod *p* ≡ α*xb* mod *p*

- As α*bx* mod *p* equals α*xb* mod *p*, both ‘A’ and ‘B’ now have a ‘shared secret’ they can use as key fornon-public encryption and decryption

  ElGamal key exchanges normally use 180 bits values for *b* and *x* – all ExpMod operations are done with a 180 bits exponent and base/modulus vectors as distributed during setup. ‘A’ must execute two of these ExpMod operations in a row to calculate the shared secret while ‘B’ executes one ExpMod during setup and only has to perform one other ExpMod upon reception of the message from ‘A’ to arrive at the same shared secret

22.1.6.4 A4: DSA Use Cases

The Digital Signature Algorithm (‘DSA’) allows signing of a message using a private key and verifying the message’s authenticity using the matching public key. It requires the use of a hashing function, for which the Digital Signature Standard (‘DSS’) version explicitly specifies the SHA-1 algorithm.

The Digital Signature Algorithm setup needs to be done only once by the owner of the private key:

- ‘A’ chooses a prime number *q* with length of 160 bits
- ‘A’ chooses a value *t* and then another prime number *p* in the range 2511+64*t* < *p* < 2512+64*t* with theproperty that *q* divides *p*–1
- ‘A’ chooses a ‘generator’ α of the unique cyclic group of order *q* in *Zp**
- ‘A’ selects a random integer *a* in the range 1 ≤ *a* ≤ *q*–1 and computes y = α*a* mod *p*
- ‘A’ publishes the public key quadruple (*p*,*q*,α,y) and keeps private key *a*

The DSA sign operation on message *m* proceeds as follows:

- ‘A’ chooses a random secret integer *k* in the range 0 < *k* < *q*
- ‘A’ computes *r* = (α*k* mod *p*) mod *q*
- ‘A’ computes the multiplicative inverse *k*–1 mod *q* of *k*
- ‘A’ computes s = k–1 • (hash(m) + a•r) mod q
- The actual signature for message *m* is the pair (*r*,*s*)

  The second bullet is an ExpMod operation with a 160 bits exponent and modulus length rangingfrom 512 bits up to 1024 bits (depending on the value *t* chosen during setup), followed by a simple MOD operation. The next most expensive operation is the calculation of the multiplicative inverse of a 160 bits value (with a modulus of also 160 bits) – the PKA can perform the necessary computations under control of the host

The DSA signature (*r*,*s*) verification for message *m* is done as follows:

- ‘B’ verifies that 0 < *r* < *q* and 0 < *s* < *q.* If not, reject the signature immediately• ‘B’ computes multiplicative inverse *s*–1 mod *q* of *s*
- ‘B’ computes u*1* = (s–1 • hash(m)) mod q and u*2* = (r•s–1) mod q
- ‘B’ computes *v* = ((α*u 1* mod *p*) • (*y u 2* mod *p*)) mod *q*
- Reject the signature when *v* is not equal to *r*

  The fourth bullet requires two ExpMod operations with a 160 bits exponent and modulus lengthranging from 512 bits up to 1024 bits (depending on the value *t* chosen during setup), followed by a straightforward MUL and MOD operations. The next most expensive operation is the calculation of the multiplicative inverse of a 160 bits value (with a modulus of also 160 bits) in the second bullet – the PKA can perform the necessary computations under control of the host

22.2 AES and SHA Cryptoprocessor

22.2.1 Architecture Overview

22.2.1.1 Functional Description

22.2.1.1.1 Basic DMA Controller With AHB Master Interface

The basic DMA controller (DMAC) registers are mapped to the external register map. To start the operation, the Host must program the mode of the DMAC and parameters of the operation. These parameters involve direction (read, write or read-and-write), length (1-65535 bytes), external source address (for reading) and external destination address (for writing).

NOTE: Note: The internal destination is programmed via a dedicated algorithm selection register in master control module. Based on the setting of that register the burst size is provided to the DMAC.

22.2.1.1.2 Key Store

The local key storage module is directly connected to a 1 kbit memory. It is capable of storing up to eight AES keys. It has eight 128-bit entries of which two are needed to store a 192 or 256-bit key. The key size is programmed in the key store module.

Keys can only be written to the key store via DMA. Once a DMA operation for a key read is started, all received data is written to the key store module. Keys that are stored in the key store memory can only be transferred to the AES key registers and are not accessible for any other purpose.

22.2.1.1.3 AES Crypto Engine

The AES crypto core supports the following modes of operation:

1. ECB
2. CBC
3. CTR
4. GCM
5. CCM
6. CBC-MAC

Modes 1 through 5 require both reading and writing of data. CBC-MAC only requires reading of data from an external source. The modes of operation 4, 5 and 6 return an authentication result. This result can either be DMA-ed out with a separate DMA operation or read via the slave interface. For all modes there is an option to provide (part of) the data via the slave interface instead of using DMA.

The AES engine is forced to use keys from the key store module for its operations. A key is provided to the AES engine by triggering the key store module to read an AES key from the key store memory and write it to the AES engine.

The AES engine automatically pads and/or masks misaligned last data blocks with zeroes for AES CBC-MAC, GCM and CCM (including misaligned AAD data). For AES CTR mode, misaligned last data blocks are internally masked to support non-block size input data.

22.2.1.1.4 SHA-256 Hash Engine

The hash module supports basic SHA-256 operations. It only requires reading of data from an external source, this data is DMA-ed via the DMAC modules. The hash result can either be DMA- ed out or read via the slave interface. To allow keyed hash operations like HMAC, there is an

option to provide (part of) the data via the AHB slave interface instead of using DMA. Refer to section 6.3.3 for HMAC programming guidelines.

22.2.1.1.5 Master Control and Interrupts

The master control module synchronizes the DMA operations and the cryptographic module handshake signals. In this module, the crypto algorithm is selected and the DMA burst sizes are defined. Once the complete crypto operation is done, an interrupt is asserted.

NOTE: Note: For authentication/hash operations, the interrupt is only asserted if the authentication result is available.

The AES module also provides an interrupt to indicate that the input DMA transfer has completed. This interrupt is primarily used to determine the end of an AAD data DMA transfer (AES-CCM and AES-CCM), which is typically setup as separate input data transfer.

Interrupts are available as output signals that should be connected to the interrupt controller of the Host system, located outside of the AES module.

22.2.1.1.6 Debug Capabilities

The AES module provides a number of status registers that should be used to monitor operations of the engine. Those are:

- DMA status and port error status registers;
- Master control module status registers.
- Key store module status register;
- Status registers of hash and crypto modules;

22.2.1.1.7 Exception Handling

The AES module can detect AHB master bus errors and abort the DMA operation. The AES key store module can detect key load errors and does not store the "bad" key in that case. In both cases, the status register in the master control module indicates the error.

22.2.2 Hardware Description

22.2.2.1 Slave Bus

22.2.2.1.1 Functional Description

For registers access, only 32-bit single accesses are allowed.

22.2.2.1.2 Endianness

The AHB Slave handles only little endian transfers.

22.2.2.1.3 Performance

The AHB Slave Interface is optimized for easy integration rather than high throughput. As each transfer is checked for multiple error conditions depending on the address, size and type of the transfer, these checks are performed on registered signals to improve timing on the input signals. Therefore, one wait cycle must be inserted for each transfer. If an ERROR response occurs, h_ready_out must be taken low one cycle after reception of the address. This results in the following timing:

- Write transfers take 2 clock cycles
- Read transfers take 3 clock cycles

22.2.2.2 Master Bus

The module is configured by the DMA configuration register (refer to Section 22.2.3.3.3.3, *DMAC Master Run-Time Parameters*) and performs single 8-bit or 32-bit non-sequential single transfers by default. Transfer addresses and length parameters of the DMA transfer are byte aligned.

22.2.2.3 Interrupts

The AES module has two interrupt outputs; both are driven from the master control module and are controlled by the respective registers (see Section 22.2.3.4.3, *Software Reset*)

An interrupt is activated when an operation that uses DMA is finished – both DMA and the used internal module are in the idle state.

Another interrupt is activated when only the input DMA is finished and is intended for debugging purpose.

NOTE: Note: Interrupt outputs are not triggered for operations where DMA is not used.

22.2.3 Module Description

22.2.3.1 Introduction

This section describes all accessible registers, internal interfaces and detailed module functionality. The registers and functionality are discussed per sub-module.

22.2.3.2 Global and Detailed Memory Map

Table 15Global memory map per sub-module

Byte Address Module

0x000 DMAC_CH0_CTRL – 0x3FFDMA controller

0x400 – 0x4FFKey store module

0x500 – 0x5FFCrypto module (AES engine)

0x600 – 0x6FFHash module (SHA-256)

0x700 – 0x7FFControl module

Table 16Detailed memory map

Byte Address Register nameR/W DefaultRemarks

DMAC registers

0x000DMAC_CH0_CTRLR/W0x00000000Channel 0 control register

0x004DMAC_CH0_EXTADDRR/W0x00000000Channel 0 external address

0x00CDMAC_CH0_DMALENGTHR/W0x00000000Channel 0 DMA length

0x018DMAC_STATUSR0x00000000DMAC status

0x01CDMAC_SWRESW0x00000000DMAC software reset

0x020DMAC_CH1_CTRLR/W0x00000000Channel 1 control register

0x024DMAC_CH1_EXTADDRR/W0x00000000Channel 1 external address

0x02CDMAC_CH1_DMALENGTHR/W0x00000000Channel 1 DMA length

0x078DMAC_MST_RUNPARAMSR/W0x00006000Master run-time parameters

0x07CDMAC_PERSRR0x00000000Port error raw status register

0x0F8DMAC_OPTIONSR0x00000202DMAC options register

0x0FCDMAC_VERSIONR0x01012ED1DMAC version register

Key store registers

0x400KEY_STORE_WRITE_AREAR/W0x00000000Write area register

0x404KEY_STORE_WRITTEN_AREAR/W0x00000000Written area register

0x408KEY_STORE_SIZER/W0x00000001Key size register

0x40CKEY_STORE_READ_AREAR/W0x00000008Read area register

Table 22-44. Register Names and Detail

Byte Address	Register name	R/W	Default	Remarks	Link
AES registers					
0x500-0x50C	AES_KEY2	W	0x00000000	Clear/wipe AES_KEY2 register	AES_AES_KEY2_0
0x510-0x51C	AES_KEY3	W	0x00000000	Clear/wipe AES_KEY3 register	AES_AES_KEY3_0
0x540	AES_IV_0	R/W	0x00000000	AES IV (LSW)	AES_AES_IV_0
0x544	AES_IV_1	R/W	0x00000000	AES IV	AES_AES_IV_1
0x548	AES_IV_2	R/W	0x00000000	AES IV	AES_AES_IV_2
0x54C	AES_IV_3	R/W	0x00000000	AES IV (MSW)	AES_AES_IV_3
0x550	AES_CTRL	R/W	0x80000000	Input/output control and mode	Table 22-71
0x554	AES_C_LENGTH_0	W	0x00000000	Crypto data length (LSW)	Table 22-72
0x558	AES_C_LENGTH_1	W	0x00000000	Crypto data length (MSW)	Table 22-73
0x55C	AES_AUTH_LENGTH	W	0x00000000	AAD data length	Table 22-74
0x560	AES_DATA_IN_0	W	0x00000000	Data input (LSW)	AES_AES_DATA_IN_OUT_0
0x560	AES_DATA_OUT_0	R	0x00000000	Data output (LSW)	
0x564	AES_DATA_IN_1	W	0x00000000	Data input	
0x564	AES_DATA_OUT_1	R	0x00000000	Data output	
0x568	AES_DATA_IN_2	W	0x00000000	Data input	
0x568	AES_DATA_OUT_2	R	0x00000000	Data output	
0x56C	AES_DATA_IN_3	W	0x00000000	Data input (MSW)	
0x56C	AES_DATA_OUT_3	R	0x00000000	Data output (MSW)	
0x570	AES_TAG_OUT_0	W	0x00000000	Tag output (LSW)	
0x574	AES_TAG_OUT_1	R	0x00000000	Tag output	
0x578	AES_TAG_OUT_2	W	0x00000000	Tag output	
0x57C	AES_TAG_OUT_3	R	0x00000000	Tag output (MSW)	
	Hash registers				
0x600	HASH_DATA_IN_0	W	0x00000000	Data Input bits [31:0] (LSW)	
0x604	HASH_DATA_IN_1	W	0x00000000	Data Input bits [63:32]	
0x608	HASH_DATA_IN_2	W	0x00000000	Data Input bits [95:64]	
0x60C	HASH_DATA_IN_3	W	0x00000000	Data Input bits [127:96]	
0x610	HASH_DATA_IN_4	W	0x00000000	Data Input bits [159:128]	
0x614	HASH_DATA_IN_5	W	0x00000000	Data Input bits [191:160]	

Table 22-44. Register Names and Detail (continued)

0x618	HASH_DATA_IN_6	W	0x00000000	Data Input bits [223:192]	
0x61C	HASH_DATA_IN_7	W	0x00000000	Data Input bits [255:224]	
0x620	HASH_DATA_IN_8	W	0x00000000	Data Input bits [287:256]	
0x624	HASH_DATA_IN_9	W	0x00000000	Data Input bits [319:288]	
0x628	HASH_DATA_IN_10	W	0x00000000	Data Input bits [351:320]	
0x62C	HASH_DATA_IN_11	W	0x00000000	Data Input bits [383:352]	
0x630	HASH_DATA_IN_12	W	0x00000000	Data Input bits [415:384]	
0x634	HASH_DATA_IN_13	W	0x00000000	Data Input bits [447:416]	
0x638	HASH_DATA_IN_14	W	0x00000000	Data Input bits [479:448]	
0x63C	HASH_DATA_IN_15	W	0x00000000	Data Input bits [511:480] (MSW)	
0x640	HASH_IO_BUF_CTRL	W	0x00000000	Input/Output Buffer Control	
0x640	HASH_IO_BUF_STAT	R	0x00000004	Input/Output Buffer Status	
0x644	HASH_MODE_IN	W	0x00000000	Mode Input register	
0x648	HASH_LENGTH_IN_L	W	0x00000000	Length Input bits [31:0] (LSW)	
0x64C	HASH_LENGTH_IN_H	W	0x00000000	Length Input bits [63:32] (MSW)	
0x650	HASH_DIGEST_A	R/W	0x00000000	Hash Digest bits [31:0] (LSW)	
0x654	HASH_DIGEST_B	R/W	0x00000000	Hash Digest bits [63:32]	
0x658	HASH_DIGEST_C	R/W	0x00000000	Hash Digest bits [95:64]	
0x65C	HASH_DIGEST_D	R/W	0x00000000	Hash Digest bits [127:96]	
0x660	HASH_DIGEST_E	R/W	0x00000000	Hash Digest bits [159:128]	
0x664	HASH_DIGEST_F	R/W	0x00000000	Hash Digest bits [191:160]	
0x668	HASH_DIGEST_G	R/W	0x00000000	Hash Digest bits [223:192]	
0x66C	HASH_DIGEST_H	R/W	0x00000000	Hash Digest bits [255:224] (MSW)	
	Master control registers				
0x700	CTRL_ALG_SEL	R/W	0x00000000	Algorithm selection	
0x704	CTRL_PROT_EN	R/W	0x00000000	Enable privileged access on master	
0x740	CTRL_SW_RESET	W	0x00000000	Master control software reset	
0x780	CTRL_INT_CFG	R/W	0x00000000	Interrupt configuration register	
0x784	CTRL_INT_EN	R/W	0x00000000	Interrupt enabling register	

Table 22-44. Register Names and Detail (continued)

0x788	CTRL_INT_CLR	W	0x00000000	Interrupt clear register	
0x78C	CTRL_INT_SET	W	0x00000000	Interrupt set register	
0x790	CTRL_INT_STAT	R	0x00000000	Interrupt status register	
0x7F8	CTRL_OPTION	R	0x01013137	AES module Type 1 options register	
0x7FC	CTRL_VERSION	R	0x91108778	AES module version	

Unspecified addresses within the AES module address space is reserved and should not be written and ignored on a read.

22.2.3.3 DMA Controller

This section describes details of the DMA controller and its operation and programming registers.

22.2.3.3.1 Operation

22.2.3.3.1.1 Internal Design

Figure 22-2 shows the DMA Controller and its integration in the AES module.

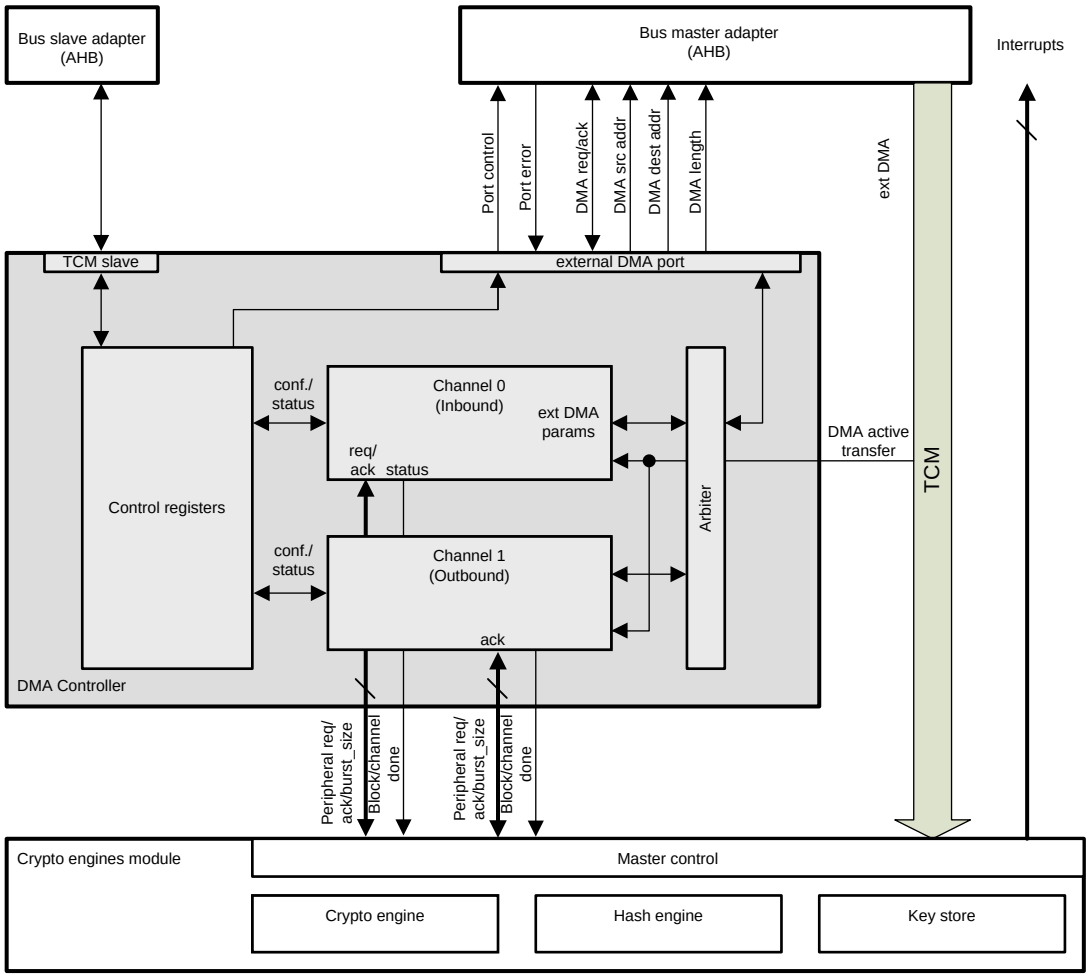


Figure 22-2. DMA Controller and Its Integration

The DMA controller (DMAC) of the AES module controls the data transfer requests to the AHB Master adapter which transfers data from and to the crypto engines (AES), hash engine and key store. The DMAC splits channel DMA operation into small DMA transfers. The size of small DMA transfers is determined by the target internal module and equals to block size of the cryptographic operation.

The DMAC has the following features:

- Two channels (one inbound and one outbound) that can be enabled at the same time;
- A maximum size of the DMA operation is controlled by 16-bit length register;
- An arbiter to schedule channel accesses to the external AHB port;
- Functionality to capture external bus errors;

22.2.3.3.1.2 Internal Operation Details

The DMAC operates in conjunction with the AHB master adapter that has two ports. One is an external AHB port used to perform read and write operations to the external AHB subsystem. It can address the complete 32-bit address range. The second one is an internal TCM port (master TCM) used to perform read and write operations to the internal modules of the crypto core's AES engine, Hash engine and Key store. Assignment of the internal modules for DMA operation must be selected in the master control module (refer toSection 22.2.3.4.1.1, *Algorithm Select*); therefore an internal address is not needed in the DMAC.

The data path from the TCM port of the AHB master module to the internal modules is located outside of the DMAC. The DMAC only observes the number of transferred words to determine when the requested DMA operation is finished for the corresponding channel.

The key store is a 32-bit block memory with a depth of 32 words, surrounded by control logic. When the AES module is configured to write keys to this module via DMA, the key store internally manages access to the key store RAM based on its register settings (including generation of the key store RAM addresses). The AES module supports only DMA write operations to the key store.

The hash engine has a 32-bit write interface for input data from the master TCM controller to be hashed and a 32-bit read interface to (optionally) read the result hash digest via the master TCM interface. The module internally collects the 32-bit data into a 512-bit input block (SHA-256 block size) and when a full block is received (indication from the DMAC) the hash calculation for the received block is started. When receiving the last word (aligned or misaligned), the DMAC and master controller generate the „last block" signal to the hash engine. The mode of the hash engine and the length of the message (for hash finalization) is programmed via the target interface.

The crypto engine has a 32-bit write interface for input data to be encrypted/decrypted and a 32-bit read interface for result data and tag. The write interface of the AES module collects 32-bit data into a 128-bit input block (AES block size) and when a full block is received, the AES calculation for the received block is started. When receiving the last word of the last block, the DMAC and master controller generate a „data done" signal to the crypto engine. The mode, length of the message, and optional parameters are programmed via the target interface.

On the TCM side, the key store module immediately accepts all data without delay cycles, while the hash and crypto modules operate on a data block boundary, processing of which takes a number of clock cycles. Special handshake signals are used between DMAC and hash/crypto modules

- A data input request is send to the DMA inbound channel (channel 0) when hash/crypto module can accept the next data block;
- A data output request is send to the DMA output channel (channel 1) when crypto module has the next block of data or tag is available after processing or hash module has a digest available;
- Both channels send an acknowledge when DMA operation has started, channel transfer done, when a block has been transmitted and the channel done, when all data are transmitted.

22.2.3.3.1.3 Supported DMA Operations

With each data request from the crypto/hash engine, the DMAC requests a transfer from the AHB master. The transfer size is at most the block size of the corresponding algorithm. This block size depends on the selected algorithm in the master control module.

A summary of the supported DMAC operations is shown in Table 22-45. The module refers to the selected module in the master control module. TAG enable indicates whether the TAG bit is set in the master control configuration register.

Table 22-45. Supported DMAC Operations

Module	Incoming data stream(for channel 0)		Outcoming data stream(for channel 1)	
	source	destination	source	destination
Key store	External memory location	Key store RAM	-	-
Crypto	RAM (Authentication data only)	AES	-(1)	-(1)
	External memory location	AES	AES	External memory location
	-(2)	-(2)	AES(TAG enabled)	External memory location
Hash	External memory location	SHA-256(TAG disabled)	-(1)	-(1)
	-(2)	-(2)	SHA-256(TAG enabled)	External memory location
	External memory location	SHA-256(TAG enabled)	SHA-256(TAG enabled)	External memory location

(1) TAG is transferred via the slave interface or transferred with a separate DMA.
(2) Data is transferred via another DMA, that has been executed before

22.2.3.3.2 Channels and Arbiter

The DMAC consists of two DMA channels: one is programmable to move input data and keys from the external memory to the AES module and another is programmable to move result data from the AES module to the external memory. The channel handling priority is programmable. Access to the channels of the AHB master port is handled by the arbiter module.

NOTE: Note: All channel control registers (DMAC_CHx_CTRL, DMAC_CHx_EXT_ADDR and DMAC_CHx_DMALENGTH) must be programmed by the Host to start new DMA operation.

22.2.3.3.2.1 Channel Control

This register is used for channel enabling and priority selection. When a channel is disabled, it becomes inactive only when all ongoing requests are finished.

Table 22-46. DMAC Channel Control Register

DMAC_CH0_CTRL, (Read/Write), 32-bit Address Offset: 0x000
DMAC_CH1_CTRL, (Read/Write), 32-bit Address Offset: 0x020

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

Bit	Name	Function
[0]	EN	Channel enable:
		0 – Disabled
		1 – Enable
		Note: Disabling an active channel will interrupt the DMA operation. The ongoing block transfer will be completed, but no new transfers will be requested.
[1]	PRIO	Channel priority:
		0 – Low

		1 – High
		If both channels have the same priority, access of the channels to the external port is arbitrated using the „round robin" scheme. If one channel has a „high" priority and another one „low", the channel with the „high" priority is served first, in case of simultaneous access requests.
[31:2]	Reserved	Should be written with zeroes and ignored on read

22.2.3.3.2.2 Channel External Address

Table 22-47. DMAC Channel External Address

DMAC_CH0_EXTADDR, (Read/Write), 32-bit Address Offset: 0x004
DMAC_CH1_EXTADDR, (Read/Write), 32-bit Address Offset: 0x024

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

Bit	Name	Function
[31:0]	ADDR	Channel external address value When read during operation, it holds the last updated external address after being sent to the master interface.

22.2.3.3.2.3 Channel DMA Length

Table 22-48. DMAC Channel Length

DMAC_CH0_DMALENGTH, (Read/Write), 32-bit Address Offset: 0x00C
DMAC_CH1_DMALENGTH, (Read/Write), 32-bit Address Offset: 0x02C

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

Bit	Name	Function
[15:0]	DMALEN	Channel DMA length in bytes
		During configuration, this register contains the DMA transfer length in bytes. During operation, it contains the last updated value of the DMA transfer length after being sent to the master interface.
		Note: Setting this register to a non-zero value starts the transfer if the channel is enabled. Therefore, this register must be written last when setting up a DMA channel!
[31:16]	Reserved	Should be written with zeroes and ignored on read

22.2.3.3.3 Control/Status Registers

22.2.3.3.3.1 DMAC Status

This register provides the actual state of each DMA channel. It also reports port errors in case these were received by the master interface module, during the data transfer.

Table 22-49. DMAC Status

DMAC_STATUS, (Read Only), 32-bit Address Offset: 0x018																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED														PORT_ERR		RESERVED														CH1_ACT	CH0_ACT
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	Name	Function
[0]	CH0_ACT	A „1" indicates that channel 0 is active (DMA transfer on-going).
[1]	CH1_ACT	A „1" indicates that channel 1 is active (DMA transfer on-going).
[16:2]	Reserved	Bits should be ignored on a read.
[17]	PORT_ERR	Reflects possible transfer errors on the AHB port.
[31:18]	Reserved	Bits should be ignored on a read.

22.2.3.3.3.2 DMAC Software Reset Register

Software reset is used to reset the DMAC to stop all transfers and clears the „port error status" register (section 4.3.3.4). After the software reset is performed, all the channels are disabled and no new requests are performed by the channels. The DMAC waits for the existing (active) requests to finish and accordingly sets the DMAC status registers.

Table 22-50. DMAC Software Reset

DMAC_SWRES, (Write Only), 16 bit Byte address: 0x01C																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																															SWRES
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	Name	Function
[0]	SWRES	Software reset enable0 = disabled
		1 = enabled (self-cleared to zero).
		Completion of the software reset must be checked via the DMAC_STATUS register.
[31:1]	Reserved	Bits should be written with a 0.

22.2.3.3.3.3 DMAC Master Run-Time Parameters

This register defines all the run-time parameters for the AHB master interface port. These parameters are required for the proper functioning of the AHB master adapter.

Table 22-51. DMAC Master Run-Time Parameters

DMAC_MST_RUNPARAMS, (Read/Write), 32-bit Address Offset: 0x078

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																Port 1 = AHB								Port 0 = TCM							
																		AHB_MST1_BURST_SIZE		AHB_MST1_IDLE_EN	AHB_MST1_INCR_EN	AHB_MST1_LOCK_EN	AHB_MST1_BIGEND	RESERVED							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0

Bits	Name	Function
[7:0]	Reserved	No Port controls are available for TCM port. Bits should be written with zeroes and ignored on read
[8]	AHB_MST1_BIGEND	Endianess for the AHB master
		0" – little endian
		1" – big endian
[9]	AHB_MST1_LOCK_EN	Locked transform on AHB
		0" – transfers are not locked
		1" – transfers are locked
[10]	AHB_MST1_INCR_EN	Burst length type of AHB transfer
		„0" – unspecified length burst transfers
		„1" – fixed length burst or single transfers
[11]	AHB_MST1_IDLE_EN	Idle insertion between consecutive burst transfers on AHB
		„0" – No Idle insertion
		„1" – Idle insertion
[15:12]	AHB_MST1_BURST_SIZE	Maximum burst size that can be performed on the AHB bus
		0010b = 4 bytes (default)
		0011b = 8 bytes
		0100b = 16 bytes
		0101b = 32 bytes
		0110b = 64 bytes
		Others = reserved
[31:16]	Reserved	Bits should be written with zeroes and ignored on read

The default configuration of this register configures fixed length transfers and a maximum burst size of 4 bytes. As a result, only non-sequential single transfers are performed on the AHB bus.

If the addresses and lengths are 32-bit aligned, the master does only NONSEQ and SINGLE type of transfers with a size of 4 bytes.

22.2.3.3.3.4 DMAC Port Error Raw Status Register

This register provides the actual status of individual port errors. It also indicates which channel is serviced by an external AHB port (which is frozen by a port error). A port error aborts operations on all serviced channels (channel enable bit is forced to zero) and prevents further transfers via that port until the error is cleared by writing to the DMAC_SWRES register.

Table 22-52. DMAC Port Error Raw Status

DMAC_PERSR, (Read Only), 16 bit Byte address: 0x07C

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Port 3 = N/A:								Port 2 = N/A:								Port 1 = AHB:								Port 0 = TCM:							
RESERVED								RESERVED								RESERVED			PORT1_AHB_ERROR	RESERVED		PORT1_CHANNEL	RESERVED	RESERVED							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	Name	Function
[7:0]	Reserved	Bits should be written with a 0 and ignored on a read.
[8]	Reserved	Bits should be written with a 0 and ignored on a read.
[11:9]	PORT1_CHANNEL	Indicates which channel has serviced last (channel 0 or channel 1) by AHB master port.
[12]	PORT1_AHB_ERROR	A "1" indicates that the AHB has detected an AHB bus error
[31:13]	Reserved	Bits should be written with a 0 and ignored on a read.

22.2.3.3.3.5 DMAC Options Register

These registers contain information regarding the different options configured in this DMAC.

Table 22-53. DMAC Options Register

DMAC_OPTIONS, (Read Only), 16 bit Byte address: 0x0F8

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

Bits	Name	Function
[2:0]	NR_OF_PORTS	Number of ports implemented, value in range 1-4.
[7:3]	Reserved	Bits should be ignored on a read.
[11:8]	NR_OF_CHANNELS	Number of channels implemented, value in the range 1-8.
[31:12]	Reserved	Bits should be ignored on a read.

22.2.3.3.3.6 DMAC Version Register

This register contains an indication/"signature" of the EIP type of this DMAC, as well as the hardware version/patch numbers.

Table 22-54. DMAC Version Register

DMAC_VERSION, (Read Only), 16 bit Byte address: 0x0FC																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED				HW_MAJOR_VERSION				HW_MINOR_VERSION				HW_PATCH_LEVEL				EIP_NUMBER_COMPL								EIP_NUMBER							
0	0	0	0	0	0	0	1	0	0	0	0	?	?	?	?	0	0	1	0	1	1	1	0	1	1	0	1	0	0	0	1

Bits	Name	Function
[7:0]	EIP_NUMBER	Binary encoding of the EIP-number of this DMA Controller (209)
[15:8]	EIP_NUMBER_COMPL	Bit-by-bit complement of the EIP_NUMBER field bits.
[19:16]	HW_PATCH_LEVEL	Patch level, starts at 0 at first delivery of this version.
[23:20]	HW_MINOR_VERSION	Minor version number
[27:24]	HW_MAJOR_VERSION	Major version number
[31:28]	Reserved	Bits should be ignored on a read.

22.2.3.4 Master Control and Select

22.2.3.4.1 Algorithm Select

This algorithm selection register configures the internal destination of the DMA controller.

22.2.3.4.1.1 Algorithm Select

Table 22-55. Master Control Algorithm Select (CTRL_ALG_SEL)

CTRL_ALG_SEL, (Read/Write), 32-bit Address Offset: 0x700

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TAG	RESERVED																												HASH (SHA-256)	AES	KEY-STORE
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[0]	KEY STORE	If set to one, selects the Key Store as destination for the DMA
		The maximum transfer size to DMA engine is set to 32 bytes (however transfers of 16,24 and 32 bytes are allowed)
[1]	AES	If set to one, selects the AES engine as source/destination for the DMA
		Both Read and Write maximum transfer size to DMA engine is set to 16 bytes
[2]	HASH (SHA-256)	If set to one, selects the Hash engine as destination for the DMA
		The maximum transfer size to DMA engine is set to 64 bytes for reading and 32 bytes for writing (the latter is only applicable if the hash result is written out via DMA)
[30:3]	Reserved	Bits should be written with zeroes and ignored on read
[31]	TAG	If this bit is cleared to zero, the DMA operation involves only data.
		If this bit is set, the DMA operation includes a TAG (Authentication Result / Digest).
		For SHA-256 operation, a DMA must be set up for both input data and TAG. For any other selected module, setting this bit only allows a DMA that reads the TAG. No data allowed to be transferred to or from the selected module via the DMA.

Table 22-56 summarizes the allowed bit combinations of the CTRL_ALG_SEL register.

Table 22-56. Valid Combinations for CTRL_ALG_SEL Flags

Operation	Flags				
	KEY STORE	AES	HASH(SHA-256)	TAG	
Hash data is loaded via the DMA, result digest is read via the slave interface	0	0	1	0	
Hash data is loaded via the DMA, result digest is read via the DMA interface	0	0	1	1	
Key store is loaded via the DMA	1	0	0	0	
AES data is loaded via the DMA and encrypted/decrypted data are read via the DMA (encryption/decryption)orAES data is loaded via the DMA and result tag is read via the slave interface (authentication-onlyoperations)	0	1	0	0	
AES data is loaded via the DMA, result tag is read via the DMA (authentication-only operations)	0	1	0	1	

22.2.3.4.2 Master PROT Enable

22.2.3.4.2.1 Master PROT -Privileged Access- Enable

This register enables the second bit (bit [1]) of the AHB HPROT bus of the AHB master interface when a read action of key(s) is performed on the AHB master interface for writing keys into the store module.

Table 22-57. Master Control Algorithm Select

CTRL_PROT_EN, (Read/Write), 32-bit Address Offset: 0x704																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																															PROT_EN
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[0]	PROT_EN	If this bit is cleared to zero m_h_prot[1] on the AHB mater interface always remains zero.
		If this bit is set to one, the m_h_prot[1] signal on the master AHB bus is asserted to„1" if a AHB read operation is performed, using DMA, with the key store module as destination.
[31:1]	Reserved	Bits should be written with zeroes and ignored on read

22.2.3.4.3 Software Reset

Table 22-58. Software Reset

CTRL_SW_RESET, (Read/Write), 32-bit Address Offset: 0x740																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																															SW_RESET
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[0]	SW_RESET	If this bit is set to „1", the following modules are reset: • Master control internal state is reset. That includes interrupt, error status register and result available interrupt generation FSM. • Key store module state is reset. That includes clearing the „written area" flags; therefore the keys must be reloaded to the key store module. Writing „0" has no effect. The bit is self cleared after executing the reset.
[31:1]	Reserved	Bits should be written with zeroes and ignored on read

Please refer to Section 22.2.5.5.1, *Soft Reset*, for more details on the soft reset procedure.

22.2.3.4.4 Interrupt

This section describes registers that are used to control the interrupt outputs.

22.2.3.4.4.1 Interrupt configuration

Table 22-59. Interrupt Configuration

CTRL_INT_CFG, (Read/Write), 32-bit Address Offset: 0x780

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

Bit	Name	Function
[0]	LEVEL	If this bit is zero, the interrupt output is a pulse
		If this bit is set to one, the interrupt is a level interrupt that must be cleared by writing the interrupt clear register
		This bit is applicable for both interrupt output signals.
[31:1]	Reserved	Bits should be written with zeroes and ignored on read

22.2.3.4.4.2 Interrupt Enable

Table 22-60. Interrupt Enable

CTRL_INT_EN, (Read/Write), 32-bit Address Offset: 0x784

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																INTERRUPTS															
RESERVED																														DMA_IN_DONE	RESULT_AV
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[0]	RESULT_AV	If this bit is set to zero the result available (irq_result_av) interrupt output is disabled
		and will remain zero
		If this bit is set to one, the result available interrupt output is enabled
[1]	DMA_IN_DONE	If this bit is set to zero the DMA input done (irq_dma_in_done) interrupt output is disabled and will remain zero
		If this bit is set to one, the DMA input done interrupt output is enabled

[31:2]	Reserved	Bits should be written with zeroes and ignored on read

22.2.3.4.4.3 Interrupt Clear

The interrupt clear register is available to clear an interrupt output in case of a level interrupt (refer to CTRL_INT_CFG) and to clear error status bits.

Table 22-61. Interrupt Clear

CTRL_INT_CLR, (Write Only), 32-bit Address Offset: 0x788

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERRORS																INTERRUPTS															
DMA_BUS_ERR	KEY_ST_WR_ERR	KEY_ST_RD_ERR	RESERVED																											DMA_IN_DONE	RESULT_AV
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[0]	RESULT_AV	If a one is written to this bit, the result available (irq_result_av) interrupt output is cleared, writing a zero has no effect.
		Note that clearing an interrupt only makes sense if the interrupt output is programmed as level (refer to CTRL_INT_CFG)
[1]	DMA_IN_DONE	If a one is written to this bit, the DMA in done (irq_dma_in_done) interrupt output is cleared, writing a zero has no effect.
		Note that clearing an interrupt only makes sense if the interrupt output is programmed as level (refer to CTRL_INT_CFG)
[28:2]	Reserved	Bits should be written with zeroes and ignored on read
[29]	KEY_ST_RD_ERR	If a one is written to this bit, the key store read error status is cleared, writing a zero has no effect.
[30]	KEY_ST_WR_ERR	If a one is written to this bit, the key store write error status is cleared, writing a zero has no effect.
[31]	DMA_BUS_ERR	If a one is written to this bit, the DMA bus error status is cleared, writing a zero has no effect

22.2.3.4.4.4 Interrupt Set

This register provides the software a way to test the interrupt connections. It should be used for debugging purposes only.

Table 22-62. Interrupt Set

CTRL_INT_SET, (Write Only), 32-bit Address Offset: 0x78C

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESEVED																INTERRUPT															
RESERVED																														DMA_IN_DONE	RESULT_AV
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[0]	RESULT_AV	If a one is written to this bit, the result available (irq_result_av) interrupt output is set to one, writing a zero has no effect.
		If the interrupt configuration register is programmed to pulse, clearing the result available (irq_result_av) interrupt is not needed. If it is programmed to level, clearing the interrupt output should be done by writing the interrupt clear register (CTRL_INT_CLR)
[1]	DMA_IN_DONE	If a one is written to this bit, the DMA data in done (irq_dma_in_done) interrupt output is set to one, writing a zero has no effect.
		If the interrupt configuration register is programmed to pulse, clearing the DMA data in done (irq_dma_in_done) interrupt is not needed. If it is programmed to level, clearing the interrupt output should be done by writing the interrupt clear register (CTRL_INT_CLR)
[31:1]	Reserved	Bits should be written with zeroes and ignored on read

22.2.3.4.4.5 Interrupt Status

Table 22-63. Interrupt Status

CTRL_INT_STAT, (Read Only), 32-bit Address Offset: 0x790

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERRORS																INTERRUPT															
DMA_BUS_ERR	KEY_ST_WR_ERR	KEY_ST_RD_ERR	RESERVED																											DMA_IN_DONE	RESULT_AV
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[0]	RESULT_AV	This read only bit returns the actual result available (irq_result_av) interrupt status of
		the result available interrupt output pin (irq_result_av).
[1]	DMA_IN_DONE	This read only bit returns the actual DMA data in done (irq_data_in_done) interrupt
		status of the DMA data in done interrupt output pin (irq_data_in_done).

[28:2]	Reserved	Bits should be written with zeroes and ignored on read
[29]	KEY_ST_RD_ERR	This bit will be set when a read error is detected during the read of a key from the key store, while copying it to the AES core. The value of this register is held until it is
		cleared via the CTRL_INT_CLR register.
		Note: This error is asserted if a key location is selected in the key store that is not
		available.
[30]	KEY_ST_WR_ERR	This bit is set when a write error is detected during the DMA write operation to the key store memory. The value of this register is held until it is cleared via the
		CTRL_INT_CLR register.
		Note: This error is asserted if a DMA operation does not cover a full key area or more areas are written than expected.
[31]	DMA_BUS_ERR	This bit is set when a DMA bus error is detected during a DMA operation. The value of this register is held until it is cleared via the CTRL_INT_CLR register.
		Note: This error is asserted if an error is detected on the AHB master interface during a DMA operation.

The error status bits in this register are asserted once they are detected. Typically, the values are valid after assertion of the irq_result_av output interrupt (DMA_BUS_ERR and KEY_ST_WR_ERR). The KEY_ST_RD_ERR bit is valid after triggering the key store module to read a key from the memory and providing it to the AES core.

22.2.3.4.5 Version and Configuration Registers

22.2.3.4.5.1 Type and Options Register

Table 22-64. Options Register

CTRL_OPTIONS, (Read Only), 32-bit Address Offset: 0x7F8

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Type								RESERVED							AHB interface	RESERVED							SHA-256	AES-CCM	AES-GCM	AES-256	AES-128	RESERVED	HASH	AES	KEY-STORE
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1	1	1	0	1	1	1

Bit	Name	Function
[0]	KEY STORE	KEY STORE is available
[1]	AES	AES core is available
[2]	HASH	HASH Core is available
[3]	Reserved	This bits should be ignored.
[4]	AES-128	AES core supports 128-bit keys
[5]	AES-256	AES core supports 256-bit keys
		Note: If both AES-128 and AES-256 are set to one, the AES core supports 192-bit keys as well.

[6]	AES-GCM	AES-GCM is available as a single operation
[7]	AES-CCM	AES-CCM is available as a single operation
[8]	SHA-256	The HASH core supports SHA-256
[15:19]	Reserved	These bits should be ignored.
[16]	AHB interface	AHB interface is available
		If this bit is zero, the AES module has a TCM interface
[23:17]	Reserved	These bits should be ignored.
[31:24]	Type	This field is 0x01 for the TYPE1 device

22.2.3.4.5.2 Version Register

Table 22-65. Version Register

CTRL_VERSION, (Read Only), 32-bit Address Offset: 0x7FC																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED				Major version number				Minor version number				Patch level				Bit-by-bit complement of EIP-number								EIP-number							
1	0	0	1	0	0	0	1	0	0	0	1	?	?	?	?	1	0	0	0	0	1	1	1	0	1	1	1	1	0	0	0

Bit	Name	Function
[7:0]	EIP-number	These bits encode the EIP number for the AES module, this field contains the value 120 (decimal) or 0x78.
[19:16]	Patch level	These bits encode the hardware patch level for this module – they start at value 0 on the first release.
[23:20]	Minor version number	These bits encode the minor version number for this module.
[27:24]	Major version number	These bits encode the major version number for this module.
[31:28]	Reserved	These bits should be ignored on a read.

22.2.3.5 AES Engine

This section describes the accessible registers for the integrated AES engine. In addition, the internal key registers, used by keys read from the store module, are discussed, as well as the internally generated intermediate keys and authentication values. These registers can be cleared via a Host access and are therefore available in this section.

22.2.3.5.1 Second Key / GHASH Key (internal, but clearable)

The following registers are not accessible via the Host for reading and writing. They are used to store internally calculated key information and intermediate results. However, when the Host performs a write to the any of the respective AES_KEY2_n or AES_KEY3_n addresses, respectively the whole 128-bit AES_KEY2_n or AES_KEY3_n register is cleared to zeroes.

The AES_GHASH_H_IN_n registers (required for GHASH, which is part of GCM) are mapped to the AES_KEY2_n registers. The (intermediate) authentication result for GCM and CCM is stored in the AES_KEY3_n register.

Table 22-66. AES_KEY

AES_KEY2_0 / AES_GHASH_H_IN_0(Write Only), 32-bit Address Offset: 0x500 AES_KEY2_1 / AES_GHASH_H_IN_1(Write Only), 32-bit Address Offset: 0x504 AES_KEY2_2 / AES_GHASH_H_IN_2(Write Only), 32-bit Address Offset: 0x508 AES_KEY2_3 / AES_GHASH_H_IN_3(Write Only), 32-bit Address Offset: 0x50C																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AES_KEY2/AES_GHASH_H[31:0] AES_KEY2/AES_GHASH_H[63:32] AES_KEY2/AES_GHASH_H[95:64] AES_KEY2/AES_GHASH_H[127:96]																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 22-67. AES_KEY

AES_KEY3_0 (Write Only), 32-bit Address Offset: 0x510 AES_KEY3_1(Write Only), 32-bit Address Offset: 0x514 AES_KEY3_2(Write Only), 32-bit Address Offset: 0x518 AES_KEY3_3(Write Only), 32-bit Address Offset: 0x51C																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AES_KEY3[31:0] / AES_KEY2[159:128] AES_KEY3[63:32] / AES_KEY2[191:160] AES_KEY3[95:64] / AES_KEY2[223:192] AES_KEY3[127:96] / AES_KEY2[255:224]																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

For GCM:

Bit	Name	Function
[127:0]	GHASH_H	The internally calculated GHASH key is stored in these registers. Only used for modes that use the GHASH function (GCM).
[255:128]	---	This register is used to store intermediate values and is initialized with zeroes when
		loading a new key.

For CCM:

Bit	Name	Function
[255:0]	---	This register is used to store intermediate values.

For CBC-MAC:

Bit	Name	Function
[255:0]	Zeroes	This register must remain zero

Reusing the AES_KEYn registers is allowed for sequential operations, however for GCM and CBC- MAC intermediate values must be cleared when programming the respective mode and length parameters:

When performing a GCM operation without loading a new key (via the key store), a write to one of the AES_KEY3 register locations is required to clear the register.

If a CBC-MAC operation is started without loading a new key (via the key store), while the previous operation was not a CBC-MAC operation, both AES_KEY2 and AES_KEY3 register locations must be written before starting the CBC-MAC operation. This is required to clear these two key registers.

22.2.3.5.2 AES Key Registers (internal)

These registers buffer the primary AES key for the AES module and are not accessible via the Host. This key is used for all operations. The key is loaded via the key store module. Refer to Section 22.2.3.7, *Key Store*, for details on AES key selection and loading.

Table 22-68.

AES_KEY1_0 AES_KEY1_1 AES_KEY1_2 AES_KEY1_3 AES_KEY1_4 AES_KEY1_5 AES_KEY1_6 AES_KEY1_7

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AES_KEY1[31:0] AES_KEY1[63:32] AES_KEY1[95:64] AES_KEY1[127:96] AES_KEY1[159:128] AES_KEY1[191:160] AES_KEY1[223:192] AES_KEY1[255:224]																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[255:0]	AES_KEY1	Primary AES key register used for all AES modes.
		The key size (see the AES_MODE register) determines which key registers are used, as indicated in the table below.

Table 22-69. Key Registers Used Per Key Size

Key Size	AES_KEY1_0	AES_KEY1_1	AES_KEY1_2	AES_KEY1_3	AES_KEY1_4	AES_KEY1_5	AES_KEY1_6	AES_KEY1_7
128-bit	√	√	√	√				
192-bit	√	√	√	√	√	√		
256-bit	√	√	√	√	√	√	√	√

22.2.3.5.3 AES Initialization Vector Registers

These registers are used to provide and read the IV from the AES engine.

Table 22-70. Table 39AES Initialization Vector Registers

AES_IV_0, (Read/Write), 32-bit Address Offset: 0x540 AES_IV_1, (Read/Write), 32-bit Address Offset: 0x544 AES_IV_2, (Read/Write), 32-bit Address Offset: 0x548 AES_IV_3, (Read/Write), 32-bit Address Offset: 0x54C

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AES_IV[31:0] AES_IV[63:32] AES_IV[95:64] AES_IV[127:96]																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Initialization Vector, used for regular non-ECB modes (CBC/CTR):

Bit	Name	Description
[127:0]	AES_IV	For regular AES operations (CBC and CTR) these registers must be written with a new 128-bit IV.
		After an operation, these registers contain the latest 128-bit result IV, generated by the crypto core.
		If CTR mode is selected, this value is incremented with 0x1 - after first use - when a new data block is submitted to the engine

For GCM:

Bit	Name	Description
[127:0]	AES_IV	For GCM operations, these registers must be written with a new 128-bit IV.
		After an operation, these registers contain the updated 128-bit result IV, generated by the crypto core.
		Note that bits [127:96] of the IV represent the initial counter value (which is „1" forGCM) and must therefore be initialized to 0x01000000.
		This value is incremented with 0x1 - after first use - when a new data block is submitted to the engine.

For CCM:

Bit	Name	Description
[127:0]	A0	For CCM this field must be written with value A0, this value is the concatenation of:
		A0-flags (5-bits of zero and 3-bits „L"), Nonce and counter value.
		„L" must be a copy from the „L" value of the AES_CTRL register. This „L" indicates the width of the Nonce and counter.
		The loaded counter must be initialized to zero.
		The total width of A0 is 128-bit.

For CBC-MAC:

Bit	Name	Description
[127:0]	Zeroes	For CBC-MAC this register must be written with zeroes at the start of each operation. After an operation, these registers contain the 128-bit TAG output, generated by the crypto core.

22.2.3.5.4 ES Input/Output Buffer Control & Mode Register

This register specifies the AES mode of operation for the crypto core.

Table 22-71. AES Input/Output Control and Mode Register

AES_CTRL, (Read/Write), 32-bit Address Offset: 0x550

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
context_ready	saved_context_ready	save_context	RESERVED				CCM-M			CCM-L			CCM	GCM		CBC-MAC	RESERVED							ctr_width		CTR	CBC	key_size (read only)		direction	input_ready	output_ready
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[0]	output_ready1	If „1", this status bit indicates that an AES output block is available to be retrieved by the Host.
		Writing a „0" clears the bit to zero and indicates that output data is read by the Host.
		The AES core can provide a next output data block.
		Writing a „1" to this bit will be ignored.
		Note:For DMA operations, this bit is automatically controlled by the crypto core.
[1]	input_ready1	If „1", this status bit indicates that the 16-byte AES input buffer is empty. The Host is permitted to write the next block of data.
		Writing a one clears the bit to zero and indicates that the AES core can use the
		provided input data block.
		Writing a „1" to this bit will be ignored.
		Note: For DMA operations, this bit is automatically controlled by the crypto core.
		After reset, this bit is „0". After writing a context1, this bit will become „1".
[2]	Direction	If set to „1" an encrypt operation is performed.
		If set to „0" a decrypt operation is performed.
		This bit must be written with a „1" when CBC-MAC is selected.
[4:3]	key_size	This read-only field specifies the key size.
		The key size is automatically configured when a new key is loaded via the key store module.
		00 = N/A - reserved
		01 = 128-bit
		10 = 192-bit
		11 = 256-bitRefer to section 4.7 for details on the AES key, key size section and loading.
[5]	CBC	If set to „1", cipher-block-chaining (CBC) mode is selected.
[6]	CTR	If set to „1", AES counter mode (CTR) is selected.
		Note: This bit must also be set for GCM and CCM, when encryption/decryption is required.
[8:7]	ctr_width	Specifies the counter width for AES-CTR mode
		00 = 32-bit counter
		01 = 64-bit counter

		10 = 96-bit counter
		11 = 128-bit counter
[14:9]	Reserved	Bits should be written with a „0" and ignored on a read
[15]	CBC-MAC	Set to „1" to select AES-CBC MAC mode.
		The direction bit must be set to „1" for this mode.
		Selecting this mode requires writing the length register after all other registers.
[17:16]	GCM	Set these bits to „11", to select AES-GCM mode.AES-GCM is a combined mode, using the Galois field multiplier GF(2128) for authentication and AES-CTR mode for encryption.
		Note: The CTR mode bit in this register must also be set to '1' to enable AES-CTR
		Bit combination description:
		00 = no GCM mode
		01 = reserved, do not select
		10 = reserved, do not select
		11 = autonomous GHASH (both H and Y0-encrypted calculated internally)
		Note: The crypto core -1 configuration only supports mode '11' (autonomous GHASH), other GCM modes are not allowed.
[18]	CCM	If set to „1", AES-CCM is selected
		AES-CCM is a combined mode, using AES for both authentication and encryption.
		Note: Selecting AES-CCM mode requires writing of the AAD length register after all other registers.
		Note:The CTR mode bit in this register must also be set to '1' to enable AES-CTR; selecting other AES modes than CTR mode is invalid.
[21:19]	CCM-L	Defines "L" that indicates the width of the length field for CCM operations; the length field in bytes equals the value of CMM-L plus one. All values are supported.
[24:22]	CCM-M	Defines "M" that indicates the length of the authentication field for CCM operations;
		the authentication field length equals two times (the value of CCM-M plus one).
		Note: The AES module always returns a 128-bit authentication field, of which the M
		least significant bytes are valid. All values are supported.
[28:25]	Reserved	Bits should be written with a „0" and ignored on a read.
	ECB	Electronic Codebook (ECB) Mode is automatically selected if bits [28:5] of this register are all zero.
[29]	save_context	This bit indicates that an authentication TAG or result IV needs to be stored as a result context
		Typically this bit must be set for authentication modes returning a TAG (CBC-MAC, GCM and CCM), or for basic encryption modes that require future continuation with the current result IV.

		If this bit is set, the engine will retain its full context until the TAG and/or IV registers are read.
		The TAG or IV must be read before the AES engine can start a new operation.
[30]	saved_context_ready(1)	If „1", this status bit indicates that an AES authentication TAG and/or IV block(s) is/are available for the Host to retrieve. This bit is only asserted if the save_context bit is set to „1". The bit is mutual exclusive with the context_ready bit.
		Writing a one clears the bit to zero, indicating the AES core can start its next operation. This bit is also cleared when the 4th word of the output TAG and/or IV is read
		Note: All other mode bit writes will be ignored when this mode bit is written with a one
		Note: This bit is controlled automatically by the AES module for TAG read DMA operations.
[31]	context_ready	If „1", this read-only status bit indicates that the context data registers can be overwritten and the Host is permitted to write the next context(2).

(1) For typical use, these bits do NOT need to be written, but are used for status reading only. In this case, the status bits are automatically maintained by the AES module. However, when writing a one to either data control bit [0], [1] or [30], writing to the other bits of this register is blocked.

(2) Writing a context means writing either a mode, the crypto length or AAD length register

NOTE: Note: Internal operation of the AES module can be interrupted by writing all mode bits to zero and writing zeroes to the length registers (AES_C_LENGTH_0, AES_C_LENGTH_1 and AES_AUTH_LENGTH)

22.2.3.5.5 AES Crypto Length Registers

These registers are used to write the Length values to the AES module. While processing, the length values decrement to zero. If both lengths are zero, the data stream is finished and a new context is requested. For basic AES modes (ECB/CBC/CTR), a crypto length of „0" can be written if multiple streams need to be processed with the same key. Writing a zero length results in continued data requests until a new context is written. For the other modes (CBC-MAC, GCM and CCM) no (new) data requests are done if the length decrements to or equals zero.

It is advised to write a new length per packet. If the length registers decrement to zero, no new data is processed until a new context or length value is written.

When writing a new mode without writing the length registers, the length register values from the previous context is reused.

Table 22-72. Crypto Data Length Register (LSW)

AES_C_LENGTH_0, (Write Only), 32-bit Address Offset: 0x554																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
c_length[31:0]																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 22-73. Crypto Data Length Register (MSW)

AES_C_LENGTH_1, (Write Only), 32-bit Address Offset: 0x558

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED			c_length[60:32]																												
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[60:0]	c-length	Bits [60:0] of the crypto length registers (LSW and MSW) store the cryptographic data length in bytes for all modes. Once processing with this context is started, this length decrements to zero. Data lengths up to (261 – 1) bytes are allowed.
		For GCM, any value up to 236 - 32 bytes can be used. This is because a 32-bit counter mode is used; the maximum number of 128-bit blocks is 232 – 2, resulting in a maximum number of bytes of 236 - 32.
		A write to this register triggers the engine to start using this context. This is valid for all modes except GCM and CCM.
		Note: For the combined modes (GCM and CCM), this length does not include the authentication only data; the authentication length is specified in the AES_AUTH_LENGTH register below.
		All modes must have a length > 0. For the combined modes, it is allowed to have one of the lengths equal to zero.
		For the basic encryption modes (ECB/CBC/CTR) it is allowed to program zero to the length field; in that case the length is assumed infinite.
		All data must be byte (8-bit) aligned for stream cipher modes; bit aligned data streams are not supported by the AES module. For block cipher modes, the data length must be programmed in multiples of the block cipher size, 16 bytes.
		For a Host read operation, these registers return all-zeroes.
[63:61]	RESERVED	Bits should be written with a „0" and ignored on a read.

22.2.3.5.6 Authentication Length Register

Table 22-74. Authentication Length Register

AES_AUTH_LENGTH, (Write Only), 32-bit Address Offset: 0x55C

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
auth_length[31:0]																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[31:0]	auth_length	Bits [31:0] of the authentication length register store the authentication data length in bytes for combined modes only (GCM or CCM)
		Supported AAD-lengths for CCM are from 0 to (216 - 28) bytes. For GCM any value up to (232 - 1) bytes can be used. Once processing with this context is started, this length decrements to zero.
		A write to this register triggers the engine to start using this context for GCM and CCM.
		For a Host read operation, these registers return all-zeroes.

22.2.3.5.7 Data Input/Output Registers

The data registers are typically accessed via DMA and not with Host writes and/or reads. However, for debugging purposes the Data Input/Output Registers can be accessed via Host write and read operations. The registers are used to buffer the input/output data blocks to/from the crypto core.

Note: The data input buffer (AES_DATA_IN_n) and data output buffer (AES_DATA_OUT_n) are mapped to the same address locations.

Writes (both DMA and Host) to these addresses load the Input Buffer while reads pull from the Output Buffer. Therefore, for write access, the data input buffer is written; for read access, the data output buffer is read. The data input buffer must be written prior to starting an operation. The data output buffer contains valid data on completion of an operation. Therefore, any 128-bit data block can be split over multiple 32-bit word transfers; these can be mixed with other Host transfers over the external interface.

Table 22-75. Data Input/Output Register

AES_DATA_IN_0 / AES_DATA_OUT_0, (Write Only), 32-bit Address Offset: 0x560 AES_DATA_IN_1 / AES_DATA_OUT_1, (Write Only), 32-bit Address Offset: 0x564 AES_DATA_IN_2 / AES_DATA_OUT_2, (Write Only), 32-bit Address Offset: 0x568 AES_DATA_IN_3 / AES_DATA_OUT_3, (Write Only), 32-bit Address Offset: 0x56C

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AES Input Data[31:0] / AES Output Data[31:0] AES Input Data[63:32] / AES Output Data[63:32] AES Input Data[95:64] / AES Output Data[95:64] AES Input Data[127:96] / AES Output Data[127:96]																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 22-76.

Bit	Name	Function
[127:0]	Data	Data registers for input/output block data to/from the crypto core.

Table 22-76. (continued)

		For normal operations, this register is not used, since data input and output is transferred from and to the AES core via DMA. For a Host write operation, these registers must be written with the 128-bit input block for the next AES operation. Writing at a word-aligned offset within this address range will store the word (4 bytes) of data into the corresponding position of 4-word deep (16 bytes = 128-bit AES block) data input buffer. This buffer is used for the next AES operation. If the last data block is not completely filled with valid data (see notes below), it is allowed to write only the words with valid data. Next AES operation is triggered by writing to theinput_ready flag of the AES_CTRL register.
		For a Host read operation, these registers contain the 128-bit output block from the latest AES operation. Reading from a word-aligned offset within this address range will read one word (4 bytes) of data out the 4-word deep (16 bytes = 128-bits AES block) data output buffer. The words (4 words, one full block) should be read before the core will move the next block to the data output buffer. To empty the data outputbuffer, the output_ready flag of the AES_CTRL register must be written.
		For the modes with authentication (CBC-MAC, GCM and CCM), the invalid (message) bytes/words can be written with any data.
		Note: AES typically operates on 128 bits block multiple input data. The CTR, GCM and CCM modes form an exception. The last block of a CTR-mode message may contain less than 128 bits (refer to [NIST 800-38A]): 0 < n <= 128 bits. For GCM/CCM, the last block of both AAD and message data may contain less than 128 bits (refer to [NIST 800-38D]). The AES module automatically pads or masks misaligned ending data blocks with zeroes for GCM, CCM and CBC- MAC. For CTR mode, the remaining data in an unaligned data block is ignored.
		Note: The AAD / authentication only data is not copied to the output buffer but only used for authentication.

Table 22-77. Input/Output Block Format Per Operating Mode

Operation	Data Input Buffer	Data Output Buffer
ECB/CBC encrypt	128-bit plaintext block	128-bit ciphertext block
ECB/CBC decrypt	128-bit ciphertext block	128-bit plaintext block
CTR encrypt	n-bit plaintext block	n-bit ciphertext block
CTR decrypt	n-bit ciphertext block	n-bit plaintext block
GCM/CCM AAD data	n-bit plaintext block	no output data
GCM/CCM encrypt data	n-bit plaintext block	n-bit ciphertext block
GCM/CCM decrypt data	n-bit ciphertext block	n-bit plaintext block
CBC-MAC data	n-bit plaintext block	no output data

22.2.3.5.8 TAG Registers

The tag registers can be accessed via DMA or directly with Host reads.

These registers buffer the TAG from the AES module. The registers are shared with the intermediate authentication result registers, but cannot be read until the processing is finished. While processing, a read from these registers returns zeroes. If an operation does not return a TAG, reading from these registers returns an IV. If an operation returns a TAG plus an IV and both need to be read by the Host, the Host must first read the TAG followed by the IV. Reading these in reverse order will return the IV twice.

Table 22-78. AES Tag Output Register

AES_TAG_OUT_0, (Read Only), 32-bit Address Offset: 0x570 AES_TAG_OUT_1, (Read Only), 32-bit Address Offset: 0x574 AES_TAG_OUT_2, (Read Only), 32-bit Address Offset: 0x578 AES_TAG_OUT_3, (Read Only), 32-bit Address Offset: 0x57C

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AES_TAG[31:0] AES_TAG[63:32] AES_TAG[95:64] AES_TAG[127:96]																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

For GCM, CCM, CBC-MAC:

Bit	Name	Function
[127:0]	AES_TAG	Bits [31:0] of the AES_TAG registers store the authentication value for the combined and authentication only modes.
		For a Host read operation, these registers contain the last 128-bit TAG output of the AES module; the TAG is available until the next context is written.
		This register will only contain valid data if the TAG is available and when thestore_ready bit from AES_CTRL register is set. During processing or for operations/modes that do not return a TAG, reads from this register return data from the IV register.

22.2.3.6 HASH Core

This section describes the integration and accessible registers for the SHA-256 hash core.

22.2.3.6.1 Introduction

The AES module contains a SHA-256 module that is connected through the wrapper module to the register and DMA interface. One of them is active at a time.

All hash module registers, except of the I/O control register are driving the hash engine"s interface.

Only the I/O control register has handshake logic associated with it.

22.2.3.6.2 Data Input Registers

The data input registers should be used to provide input data to the hash module via the slave interface.

Table 22-79. Hash Data Input Register

HASH_DATA_IN_0, (Write Only), 32-bit Address Offset: 0x600 HASH_DATA_IN_1, (Write Only), 32-bit Address Offset: 0x604 HASH_DATA_IN_2, (Write Only), 32-bit Address Offset: 0x608 HASH_DATA_IN_3, (Write Only), 32-bit Address Offset: 0x60C HASH_DATA_IN_4, (Write Only), 32-bit Address Offset: 0x610 HASH_DATA_IN_5, (Write Only), 32-bit Address Offset: 0x614 HASH_DATA_IN_6, (Write Only), 32-bit Address Offset: 0x618 HASH_DATA_IN_7, (Write Only), 32-bit Address Offset: 0x61C HASH_DATA_IN_8, (Write Only), 32-bit Address Offset: 0x620 HASH_DATA_IN_9, (Write Only), 32-bit Address Offset: 0x624 HASH_DATA_IN_10, (Write Only), 32-bit Address Offset: 0x628 HASH_DATA_IN_11, (Write Only), 32-bit Address Offset: 0x62C HASH_DATA_IN_12, (Write Only), 32-bit Address Offset: 0x630 HASH_DATA_IN_13, (Write Only), 32-bit Address Offset: 0x634 HASH_DATA_IN_14, (Write Only), 32-bit Address Offset: 0x638 HASH_DATA_IN_15, (Write Only), 32-bit Address Offset: 0x63C

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
hash_data_in[31:0] … hash_data_in[511:480]																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[31:0]	hash_data_in	These registers must be written with the 512-bit input data. The data lines are connected directly to the data input of the hash module and hence into the engine"s internal data buffer. Writing to each of the registers triggers a corresponding 32-bit write enable to the internal buffer.
		Note: The Host may only write the input data buffer when the rfd_in bit of the HASH_IO_BUF_CTRL register is high. If the rfd_in bit is zero, the engine is busy with processing. During processing, it is not allowed to write new input data.
		For message lengths larger than 64 bytes, multiple blocks of data are written to this input buffer using a handshake via flags of the HASH_IO_BUF_CTRL register. All blocks except the last are required to be 512 bits in size. If the last block is not 512 bits long, only the least significant bits of data have to be written, but they have to be padded with zeroes to the next 32-bit boundary.
		Host read operations from these register addresses, will return zeroes.

22.2.3.6.3 Input/Output Buffer Control & Status Register

This register pair shares a single address location and contains bits that control and monitor the data flow between the Host and the hash engine.

Table 22-80. Hash I/O Buffer Control

HASH_IO_BUF_CTRL, (Write/Read), 32-bit Address Offset: 0x640

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								pad_dma_message	get_digest	pad_message	RESERVED	RESERVED	rfd_in	data_in_av	output_full
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[0]	output_full	Indicates that the output buffer registers (HASH_DIGEST_n) are available for reading by the Host.
		When this bit reads '0', the output buffer registers are released; the hash engine is allowed to write new data to it. In this case, the registers should not be read by the Host.
		When this bit reads '1', the hash engine has stored the result of the latest hashoperation in the output buffer registers. As long as this bit reads '1', the Host may read output buffer registers and the hash engine is prevented from writing new data to theoutput buffer.
		After retrieving the hash result data from the output buffer, the Host must write a '1' to this bit to clear it. This makes the digest output buffer available for the hash engine to store new hash results.
		Writing a '0' to this bit has no effect.
		Note: If this bit is asserted (‘1’) no new operation should be started before the digest is retrieved from the hash engine and this bit is cleared (‘0’)
[1]	data_in_av	Note: The bit description below is only applicable when data is sent via the slave interface. This bit must be set to zero when data is received via DMA.
		This bit indicates that the HASH_DATA_IN registers contain new input data for processing.

		The Host must write a '1' to this bit to start processing the data in HASH_DATA_IN; the hash engine will process the new data as soon as it is ready for it (rfd_in bit is „1").
		Writing a '0' to this bit has no effect.
		This bit is automatically cleared (i.e. reads as '0') when the hash engine starts processing the HASH_DATA_IN contents. This bit reads '1' between the time it was set by the Host and the hash engine actually starts processing the input data block.
[2]	rfd_in	Note: The bit description below is only applicable when data is sent via the slave interface. This bit can be ignored when data is received via DMA.
		Read-only status of the input buffer of the hash engine.
		When „1", the hash engine"s input buffer can accept new data; the HASH_DATA_IN registers can safely be populated with new data.
		When „0", the hash engine"s input buffer is processing the data that is currently in HASH_DATA_IN; it is not allowed write new data to these registers.
[4:3]	Reserved	Write zeroes and ignore on reading
[5]	pad_message	Note: The bit description below is only applicable when data is sent via the slave interface. This bit must be set to zero when data is received via DMA.
		This bit indicates that the HASH_DATA_IN registers hold the last data of the message and hash padding must be applied.
		The Host must write this bit to '1' in order to indicate to the hash engine that the
		HASH_DATA_IN register currently holds the last data of the message. When pad_message is set to '1', the hash engine will add padding bits to the data currently in the HASH_DATA_IN register.
		When the last message block is smaller than 512 bits, the pad_message bit must be set to „1" together with the data_in_av bit.

		When the last message block is equal to 512 bits, pad_message may be set together with data_in_av. In this case the pad_message bit may also be set after the last data block has been written to the hash engine (so when the rfd_in bit has become „1" again after writing the last data block).
		Writing a '0' to this bit has no effect.
		This bit is automatically cleared (i.e. reads '0') by the hash engine. This bit reads '1' between the time it was set by the Host and the hash engine interpreted its value.
[6]	get_digest	Note: The bit description below is only applicable when data is sent via the slave interface. This bit must be set to zero when data is received via DMA.
		This bit indicates whether the hash engine should provide the hash digest.
		When provided simultaneously with data_in_av, the hash digest is provided after processing the data that is currently in the HASH_DATA_IN register. When provided without data_in_av, the current internal digest buffer value is copied to the HASH_DIGEST_n registers.
		The Host must write a '1' to this bit to make the intermediate hash digest available.
		Writing a '0' to this bit has no effect.
		This bit is automatically cleared (i.e. reads '0') when the hash engine has processed the contents of the HASH_DATA_IN register. In the period between this bit is set by the Host and the actual HASH_DATA_IN processing, this bit reads '1'.
[7]	pad_dma_message	Note:This bit must only be used when data is supplied via DMA. It should not be used when data is supplied via the slave interface.
		This bit indicates whether the hash engine has to pad the message, received via DMA and finalize the hash.
		When set to „1" the hash engine pads the last block using the programmed length.
		After padding, the final hash result is calculated.
		When set to „0" hash engine treats the last written block as block-size aligned and calculates the intermediate digest.

		This bit is automatically cleared when the last DMA data block is arrived in the hash engine.
[31:8]	Reserved	Write zeroes and ignore on reading

22.2.3.6.4 Mode Registers

This register provides the mode bits that allow the Host to initiate a hash computation or indicate that it has retrieved the hash result. A Host write has always the highest priority.

Note: This register does not trigger the hash operation; it only sets up the hash mode.

Table 22-81. Hash Mode Register

HASH_MODE_IN, (Write Only), 32-bit Address Offset: 0x644

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												sha256_mode	RESERVED	RESERVED	new_hash
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[0]	new_hash	When set to „1", it indicates that the hash engine must start processing a new hashsession. The HASH_DIGEST_n registers will automatically be loaded with the
		initial hash algorithm constants of the selected hash algorithm.
		When this bit is „0" while the hash processing is started, the initial hash algorithm
		constants are not loaded in the HASH_DIGEST_n registers. The hash engine will
		start processing with the digest that is currently in its internal HASH_DIGEST_n
		registers.
		This bit is automatically cleared when hash processing is started.
[1:2]	Reserved	Write zeroes and ignore on reading
[3]	sha256_mode	The Host must write this bit with „1" prior to processing a hash session.
[31:4]	Reserved	Write zeroes and ignore on reading

22.2.3.6.5 Length Registers

Table 22-82. Hash Length Register

HASH_LENGTH_IN_L, (Write Only), 32-bit Address Offset: 0x648
HASH_LENGTH_IN_H, (Write Only), 32-bit Address Offset: 0x64C

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
length_in[31:0] length_in[63:32]																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[31:0]	length_in	Message length registers. The content of these registers is used by the hash engine during the message padding phase of the hash session. The data lines of this registers are directly connected to the interface of the hash engine.
		For a write operation by the Host, these registers should be written with the message length in bits.
		Final hash operations:
		The total input data length must be programmed for new hash operations that require finalization (padding). The input data must be provided via the slave or DMA interface.
		Continued hash operations (finalized):
		For continued hash operations that require finalization, the total message length must be programmed, including the length of previously hashed data that corresponds to the written input digest.
		Non-final hash operations:
		For hash operations that do not require finalization (input data length is multiple of 512-bits which is SHA-256 data block size), the length field does not need to be programmed since not used by the operation.
		If the message length in bits is below (232-1), then only HASH_LENGTH_IN_L needs to be written. The hardware automatically sets HASH_LENGTH_IN_H to zeroes in this case.
		The Host may write the length register at any time during the hash session when the rfd_in bit of the HASH_IO_BUF_CTRL is high. The length register must be written before the last data of the active hash session is written into the hash engine.
		Host read operations from these register locations will return zeroes.
		Note: When getting data from DMA, this register must be programmed before DMA is programmed to start.

22.2.3.6.6 Hash Digest Registers

The hash digest registers consist of eight 32-bit registers, named HASH_DIGEST_A to HASH_DIGEST_H. After processing a message, the output digest can be read from these registers. These registers can be written with an intermediate hash result for continued hash operations.

Table 22-83. Hash Digest Registers

HASH_DIGEST_A, (Read/Write), 32-bit Address Offset: 0x650 HASH_DIGEST_B, (Read/Write), 32-bit Address Offset: 0x654 HASH_DIGEST_C, (Read/Write), 32-bit Address Offset: 0x658 HASH_DIGEST_D, (Read/Write), 32-bit Address Offset: 0x65C HASH_DIGEST_E, (Read/Write), 32-bit Address Offset: 0x660 HASH_DIGEST_F, (Read/Write), 32-bit Address Offset: 0x664 HASH_DIGEST_G, (Read/Write), 32-bit Address Offset: 0x668 HASH_DIGEST_H, (Read/Write), 32-bit Address Offset: 0x66C

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
hash_digest[31:0] … hash_digest[255:224]																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[31:0]	hash_digest	Hash Digest registers. Write operation:
		Continued hash:
		These registers should be written with the context data, prior to the start of a resumed hash session (the new_hash bit in the HASH_MODE register is „0" when starting a hash session).
		New hash:
		When initiating a new hash session (the new_hash bit in the HASH_MODE register is High), the internal digest registers are automatically set to the SHA- 256 algorithm constant and these register should not be written.
		Reading from these registers will provide the intermediate hash result (non-final hash operation) or the final hash result (final hash operation) after data processing.

22.2.3.7 Key Store

The key store module can store up to four 256-bit AES keys or eight 128-bit AES keys in the key store RAM. The key material in the key store is not accessible through read operations via the AHB master and slave interfaces.

22.2.3.7.1 Key Store Write Area Register

This register defines where the keys should be written in the key store RAM. After writing this register, the key store module is ready to receive the keys via a DMA operation. In case the key data transfer triggered an error in the key store, the error will be available in the interrupt status register after the DMA is finished, as described inSection 22.2.3.4.4.5, *Interrupt Status*. The key store write-error is asserted when the programmed/selected area is not completely written. This error is also asserted when the DMA operation writes to ram areas that are not selected.

The key store RAM is divided into 8 areas of 128 bits.

192-bit keys written in the key store RAM should start on boundaries of 256 bits. This means that writing a 192-bit key to the key store RAM must be done by writing 256 bits of data with the 64 most significant bits set to zero. These bits are ignored by the AES engine.

Table 22-84. Key Store Write Area Register

KEY_STORE_WRITE_AREA, (Read/Write), 32-bit Address Offset: 0x400

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								ram_area7	ram_area6	ram_area5	ram_area4	ram_area3	ram_area2	ram_area1	ram_area0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[7:0]	ram_areax	Each ram_areax represents an area of 128 bits.
		Select the key store RAM area(s) where the key(s) needs to be written.
		ram_areax:
		0 – ram_areax is not selected to be written.
		1 – ram_areax is selected to be written.
		Writing to multiple RAM locations is only possible when the selected RAM areas are sequential.
		Keys that require more than one RAM locations (key size is 192 or 256 bits), must start at one of the following areas: ram_areax0, ram_area2, ram_area4 or ram_area6.

22.2.3.7.2 Key Store Written Area Register

This register shows which areas of the key store RAM contain valid written keys.

When a new key needs to be written to the key store, on a location that is already occupied by a valid key, this key area must be cleared first. This can be done by writing this register before the new key is written to the key store memory.

Attempting to write to a key area that already contains a valid key is not allowed and will result in an error.

Table 22-85. Key Store Written Area (Status) Register

KEY_STORE_WRITTEN_AREA, (Read/Write), 32-bit Address Offset: 0x404

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								ram_area_written7	ram_area_written6	ram_area_written5	ram_area_written4	ram_area_written3	ram_area_written2	ram_area_written1	ram_area_written0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[7:0]	ram_area_writtenx	ram_area_writtenx (read):
		0 – This RAM area is not written with valid key information.
		1 – This RAM area is written with valid key information.
		Each individual ram_area_writtenx bit can be reset by writing a „1". Note: This register will be reset on a soft reset from the master control module.
		After a soft reset, all keys must be rewritten to the key store memory.

22.2.3.7.3 Key Store Size Register

This register defines the size of the keys that are written with DMA. This register should be configured before writing to the KEY_STORE_WRITE_AREA register.

Table 22-86. Key Store Size Register

KEY_STORE_SIZE, (Read/Write), 32-bit Address Offset: 0x408																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																															key_size
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Bit	Name	Function
[1:0]	key_size	key size:
		00 - Reserved
		01 – 128 bits
		10 – 192 bits
		11– 256 bits
		When writing this to this register, KEY_STORE_WRITTEN_AREA register will be reset.

22.2.3.7.4 Key Store Read Area Register

This register selects the key store RAM area from where the key needs to be read that will be used for an AES operation. The operation will directly start after writing this register. When the operation is finished, the status of the key store read operation is available in the interrupt status

register described in Section 22.2.3.4.4.5, *Interrupt Status*. Key store read error will be asserted when a ram area is selected which doesn"t contain valid written key

Table 22-87. Key Store Read Area Register

KEY_STORE_READ_AREA, (Read/Write), 32-bit Address Offset: 0x40C																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
busy	RESERVED																											ram_area			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Bit	Name	Function
[3:0]	ram_area	Selects the area of the key store RAM from where the key needs to be read that will be writen to the AES engine.
		ram_area:
		0000 – ram_area0
		0001 – ram_area1
		0010 – ram_area2
		0011 – ram_area3
		0100 – ram_area4
		0101 – ram_area5
		0110 – ram_area6
		0111 – ram_area7

		1000 – no ram area selected
		1001 .. 1111 – Reserved
		RAM areas ram_area0, ram_area2, ram_area4 and ram_area6 are the only valid read
		areas for 192 and 256 bits key sizes.
		Only RAM areas that contain valid written keys can be selected.
[30:4]	Reserved	Write zeroes and ignore on reading
[31]	busy	Key store operation busy status flag (read only):
		0 –operation is completed.
		1 –operation is not completed and the key store is busy.

22.2.4 Performance

22.2.4.1 Introduction

The following processing steps of the AES module are the basis for the performance calculations. Three major steps are identified for crypto operations using DMA:

1. Initialization (setup and initialization of the engines, DMA etc.)
2. Data processing for the complete message
3. Finalization (reading out the result, status checking)

The orange sections („full processing") in the figure below, are covered by step 1 and 3. These steps are under control of the Host CPU and therefore dependent on the performance of the Host. The second step is covered by the green section („data processing") and is fully handled by the

hardware. This part is not dependent on the performance of the Host CPU.

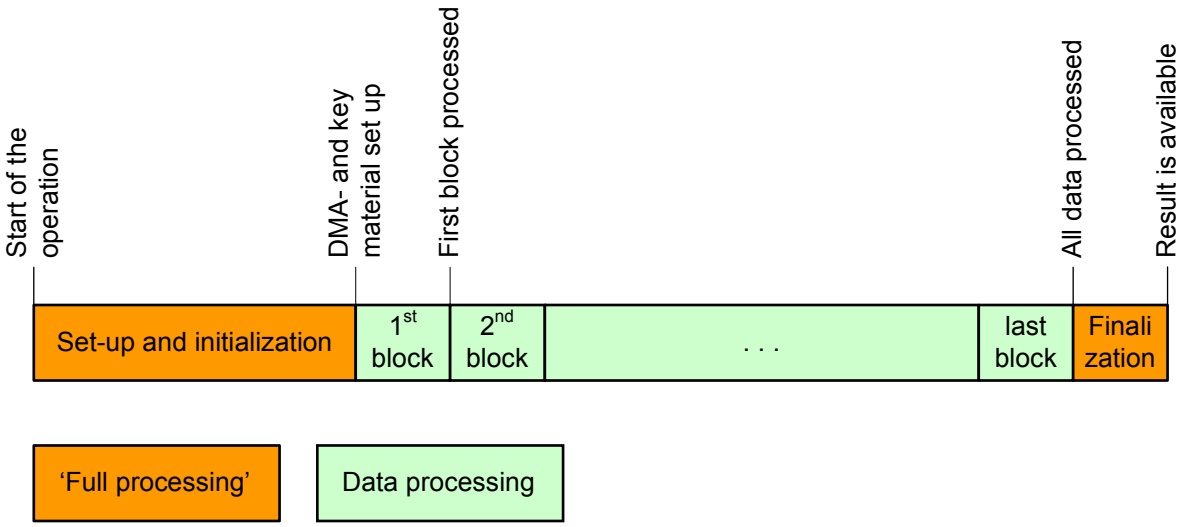


Figure 22-3. Symmetric Crypto Processing Steps

The „Full Processing" part is required once per processing command and precedes the processing of the first data block. The „Data Processing" blocks depend on the amount of data to be processed by the command. The „Finalization" is required when the operation produces a result digest or TAG.

The number of required blocks is determined by the block size requirements of the algorithms selected by the command:

- The AES block size is 128-bit;

- The HASH block size is 512-bit;

For longer data streams, the data processing time approaches the theoretical maximum throughput.

For operations that use the slave interface as alternative for the DMA, the performance depends on the performance of the Host CPU.

22.2.4.2 Performance

Table 22-88 shows the performance of the AES module running at 200MHz for DMA-based cryptographic operations.

Table 22-88. Performance Table for DMA-Based Operations

Performance in Mbps				
Crypto-mode	Raw engine performance	1 block packet performance[1]	20-block performance[1]	100-block performance[1]
AES-128 (1 block = 128-bits)				
AES-128-ECB	79	18	67	76
AES-128-CBC	77	17	65	75
AES-128-CTR	79	17	66	76
AES-128-GCM2	79	12	61	73
AES-192 (1 block = 128-bits)				
AES-192-ECB	66	17	58	64
AES-192-CBC	54	16	56	63
AES-192-CTR	66	16	57	64
AES-192-GCM[2]	66	12	53	62
AES-256 (1 block = 128-bits)				
AES-256-ECB	57	16	51	56
AES-256-CBC	56	15	49	55
AES-256-CTR	57	15	50	55
AES-256-GCM[2]	57	10	47	54
HASH (1 block = 512-bits)				
SHA-256	252	60	2178	244

(1) The performance assumes full programming of the engine, loading keys and setting up the DMA engine via the DMA slave. If the context is reused (mode and or keys) the performance is increased. The maximum number of cycles overhead per packet is between 100 and 150 for the various modes and algorithms.

(2) AES-GCM raw performance numbers exclude the final operation to create the TAG; the block performance numbers do include this overhead.
Note: The performance scales linearly with the clock frequency.

The engine performance heavily depends on the number of blocks that is processed per operation. Processing of a single block results in the minimum engine performance, in this case the configuration overhead is the most significant (assuming the engine is fully reconfigured for each operation). Therefore, processing multiple blocks per operation results in a significantly higher performance.

22.2.5 Programming Guidelines

This chapter contains example descriptions of how to program the AES module for the supported use cases.

22.2.5.1 One Time Initialization After a Reset

The purpose of the initialization is to set the AES module into the initial mode that is common to all used operations. The following initialization steps should be done after a hardware reset:

- Read out and check that the AES module version and configuration matches the expected hardware configuration.
- Program the DMAC run-time parameters with the desired values that are common for all DMA

operations.

- Initialize the desired interrupt type (level or pulse) and enable the interrupt output signal irq_result_av in the master control module.

22.2.5.2 DMAC and Master Control

This section contains general guidelines on how to program the DMAC to perform a specific operation.

22.2.5.2.1 Regular use

- To configure the DMA channels, the following registers need to be programmed: Clear any outstanding interrupts and error flags (if possible)
  - CTRL_INT_CLR
- Master control module algorithm selection register. Should be programmed such that it allows a DMA operation on the required internal module. This enables the DMA/AHB Master clock and keeps it enabled until clock is disabled by the Host.
  - CTRL_ALG_SEL
- Channel control registers with channel bits enabled
  - DMAC_CH0_CTRL_ADDR
  - DMAC_CH1_CTRL_ADDR
- Channel external address registers
  - DMAC_CH0_EXTADDR_ADDR
  - DMAC_CH1_EXTADDR_ADDR
- Channel DMA length registers. Writing this register starts the DMA operation on the corresponding channel.
  - DMAC_CH0_DMALEN_ADDR
  - DMAC_CH1_DMALEN_ADDR
- Completion of the operation is indicated by the result available interrupt output or the corresponding status register. Clear the interrupt after handling it.
  - The i r q _r e su l t _av output signal is asserted (status is also available in: CTRL_INT_STAT)
  - CTRL_INT_CLR
- Master control module algorithm selection register. Must be cleared to zero to switch off the DMA/AHB Master clock (refer to section 6.2.3 for guidelines on disabling this clock).
  - CTRL_ALG_SEL

NOTE: Note: The CTRL_INT_STAT register should be checked for possible errors if bus errors can occur in the system. This is typically valid in a debugging phase or in systems where bus errors can occur during a DMA operation.

22.2.5.2.2 Interrupting DMA Transfers

If the Host wants to stop a DMA transfer to abort the operation, it can disable a channel via the DMAC_CHx_CTRL registers. Once the En bit of this register is set to „0", no new DMA transfer is requested by this channel and the current active transfer will be finished. Alternatively, all active channels can be stopped by activating the DMAC soft reset (DMAC_SWRES).

NOTE: Note: When stopping the DMAC, the Host must stop all active channels.

The state of the DMAC channel must be checked via DMAC_STATUS register and when the CHx_ACT bit of this register for the disabled channel goes to „0" – DMAC channel is stopped.

To stop the DMAC in combination with the AES engine, the AES engine must be set in idle mode first. This is done by writing zeroes to the length registers followed by disabling all modes in the AES_CTRL register.

Stopping the DMAC channels might leave the master control module in an unfinished state due to pending events from the engines that will never occur. Therefore, to correctly recover the engine, the master control soft reset must be issued (via CTRL_SW_RESET) after all active DMAC channels are stopped (refer to section 4.4.3).

22.2.5.2.3 Interrupts and HW/SW Synchronization

This section describes the important relation of the irq_result_av interrupt activation and the data writing completion of the DMAC inside the crypto core.

The irq_result_av interrupt is activated when the AHB master finishes the data write transfer from the crypto core and the internal operation is completed. However, that does not guarantee that data has been written to the external memory due to latency from the AHB master to the destination (typically a memory). This latency might occur in the AHB bus subsystem outside of the crypto core, as this system can possibly contain bridges.

NOTE: Note: If the latency, described above can occur, the Host must ensure (via timeout or other synchronization mechanisms) that external memory reads are only performed when all memory write operations are finished.

22.2.5.3 Hashing

The hash engine has the following interfaces:

- It accepts input data from two sources: the AHB slave interface and DMA. Within one operation, it is possible to combine data from these two sources: write data from the slave interface and later write data from the DMA to complete the hash operation.
- The input digest (for resumed hash) and length must be programmed via the register interface.
- The result digest should be read via the slave interface or via DMA.

By using these interfaces, basic hash and HMAC operations may be performed with various scenarios. The following sections describe the required steps to perform these operations for the typical use cases.

22.2.5.3.1 Data Format and Byte Order

In most systems, the message data is stored in Host memory in little-endian fashion. The hash engine is designed as a little-endian core to prevent data swap in the system. Therefore, the message data has to be provided in a little-endian fashion to the core.

The following examples show how the data must be provided to the engine, based on the FIPS 180-2 and RFC 1321 specifications.

NOTE: Note: The highlighted byte is the first byte of the data or digest block.

FIPS 180-2, chapter B.2:

```
Data In:"abcdbcdecdefdefgefghfghighijhi jkijkljklmklmnlmnomnopnopq"
= 61626364 62636465 63646566 64656667
65666768 66676869 6768696a 696a6b6c
68696a6b 6a6b6c6d 6b6c6d6e 6c6d6e6f
6d6e6f70 6e6f7071

Digest Out:248d6a61 d20638b8 e5c02693 0c3e6039 a33ce459 64ff2167 f6ecedd4
                19db06c1
```

Hash data in external RAM, loaded via DMAC:

```
Word_0[31:0]:64636261
```

```
Word_1[31:0]:65646362
Word_2[31:0]:66656463
...
```

Output digest, read via slave interface or DMA:

```
HASH_DIGEST_A[31:0]:616a8d24
HASH_DIGEST_B[31:0]:b83806d2
HASH_DIGEST_C[31:0]:9326c0e5
```

22.2.5.3.2 Basic Hash With Data From the DMA

The hash engine hashes the data, received from the external memory via DMA and may be programmed to store the result digest into external memory using a DMA operation. The typical sequence for using the crypto core for such operations is the following:

- The hash engine mode is programmed via the slave interface.
- For resumed hash operations, the hash engine"s initial digest is programmed via the slave interface.

  If the resumed operation is a continuation of the previous (non-finished) hash operation, the hash engine holds its current state. In this case, it may be re-used as is for the next operation and the initial digest does not need to be programmed.
- The master controller is programmed to move data to the hash engine. If the result digest is required to be written to the external memory, the master controller should be programmed accordingly.
- DMAC channel 0 is programmed to read data from the external memory and copy it to the data input port of the hash engine.
- If the result digest must be written to the external memory, DMAC channel 1 is programmed to store the result digest into a pre-allocated area in the external memory.
- The interrupt status is observed to check when the engine has completed the operation.
- If the result digest needs to be read by the Host, it can be read via the slave interface; if the result digest is written to external memory via DMA, it is available in external memory (see Section 22.2.5.2.3, *Interrupts and HW/SW Synchronization*).

22.2.5.3.2.1 New Hash Session With Digest Read Via Slave

The following software example in pseudo code describes the actions that are typically executed by the Host software. It starts the hash engine with a new hash session that receives the input data via the DMA interface. In the end, the intermediate digest (non-final hash operation) or the finalized hash digest (final hash operation) is read as result digest via the slave interface.

```
// configure master control module
write CTRL_ALG_SEL 0x0000_0004 // enable DMA path to the SHA-256 engine
write CTRL_INT_CLR 0x0000_0001 // clear any outstanding events

// configure hash engine
write HASH_MODE = 0x0000_0009 // indicate the start of a new hash session and SHA256
write HASH_LENGTH_L// write the length of the message (lo)
write HASH_LENGTH_H// write the length of the message (hi)
// if the final digest is required (pad the input DMA data), write the following register
write HASH_IO_BUF_CTRL = 0x80 // pad the DMA-ed data

// configure DMAC
write DMAC_CH0_CTRL 0x0000_00001 // enable DMA channel 0
write DMAC_CH0_EXTADDR <ext_memory_address> // base address of the data in ext. memory
write DMAC_CH0_DMALENGTH <length>// input data length in bytes, equal to
                                 // the message length

wait CTRL_INT_STAT[0] = ‘1’// wait for operation done (hash and DMAC areready)
check CTRL_INT_STAT[31] == ‘0’ // check for the absence of errors
// read digest
read HASH_DIGEST_A
...
read HASH_DIGEST_H
```

```
// acknowledge result and clear interrupts
write HASH_IO_BUF_CTRL = 0x0000_0001 // acknowledge reading of the digest
write CTRL_INT_CLR 0x0000_0001// clear the interrupt
write CTRL_ALG_SEL 0x0000_0000// disable the master control/DMA clock
// end of algorithm
```

22.2.5.3.2.2 New Hash Session With Digest To External Memory

The following software example in pseudo code describes the actions that are typically executed by the Host software. It starts the hash engine with a new hash session that receives the input data via the DMA interface. In the end, the intermediate digest (non-final hash operation) or the finalized hash digest (final hash operation) is returned to the external memory via DMA.

```
// configure master control module
write CTRL_ALG_SEL 0x8000_0004 // enable DMA path to the SHA-256 engine + Digest readout
write CTRL_INT_CLR 0x0000_0001 // clear any outstanding events

// configure hash engine
write HASH_MODE = 0x0000_0009 // indicate start of a new hash session and SHA256
write HASH_LENGTH_L // write length of the message (lo)
write HASH_LENGTH_H // write length of the message (hi)
// if the final digest is required (pad the input DMA data), write the following register
write HASH_IO_BUF_CTRL = 0x80 // pad the DMA-ed data

// configure DMAC write DMAC_CH0_CTRL 0x0000_00001 // enable DMA channel 0 for message data
write DMAC_CH0_EXTADDR <ext_memory_address> // base address of the data in ext. memory
write DMAC_CH0_DMALENGTH <length> // input data length in bytes, equal to the message
                                  // length
write DMAC_CH1_CTRL 0x0000_00001 // enable DMA channel 1 for result digest
write DMAC_CH1_EXTADDR <ext_memory_address> // base address of the digest buffer
write DMAC_CH1_DMALENGTH <length> // length of the result digest

// wait for completion and acknowledge the interrupt
wait CTRL_INT_STAT[0] = '1'// wait for operation done (hash and DMAC are ready)
check CTRL_INT_STAT[31] == '0' // check for the absence of errors
write CTRL_INT_CLR 0x0000_0001 // clear the interrupt
write CTRL_ALG_SEL 0x0000_0000 // disable the master control/DMA clock
// the digest can now be read from the external memory (see note in section 6.2.3)
// end of algorithm
```

22.2.5.3.2.3 Resumed Hash Session

The following software example in pseudo code describes the actions that are typically executed by the Host software. It starts the hash engine with a resumed hash session that receives the input data via the DMA interface. In the end, the intermediate digest (non-final hash operation) or the finalized hash digest (final hash operation) is read as result digest via the slave interface.

```
// configure master control module
write CTRL_ALG_SEL 0x0000_0004 // enable the DMA path to the SHA-256 engine
write CTRL_INT_CLR 0x0000_0001 // clear any outstanding events

// configure hash engine
write HASH_MODE = 0x0000_0008 // indicate the start of a resumed hash session and SHA256
write HASH_LENGTH_L// write the length of the total message (lo)
write HASH_LENGTH_H// write the length of the total message (hi)
// write the initial digest
write HASH_DIGEST_A
...
write HASH_DIGEST_H
// if the final digest is required (pad the input DMA data), write the following register
write HASH_IO_BUF_CTRL = 0x80 // pad the DMA-ed data

// configure DMAC
```

```
write DMAC_CH0_CTRL 0x0000_00001// enable DMA channel 0
write DMAC_CH0_EXTADDR <ext_memory_address> // base address of the data in ext. memory
write DMAC_CH0_DMALENGTH <length>// input data length in bytes

wait CTRL_INT_STAT[0] = '1'// wait for operation done (hash and DMAC are ready)
check CTRL_INT_STAT[31] == '0' // check for the absence of errors

// read digest
read HASH_DIGEST_A
...
read HASH_DIGEST_H

// acknowledge result and clear interrupts
write HASH_IO_BUF_CTRL = 0x01 // acknowledge reading of the digest
write CTRL_INT_CLR 0x0000_0001 // clear the interrupt
write CTRL_ALG_SEL 0x0000_0000 // disable master control/DMA clock
// end of algorithm
```

22.2.5.3.3 HMAC

The crypto core supports HMAC operations in two steps (excluding optional pre-processing), according to algorithm defined in RFC2104.

Let:

- H(·) be a cryptographic hash function
- K be a secret key padded to the right with extra zeros to the block size of the hash function
- m be the message to be authenticated
- ‖ denote concatenation
- ⊕ denote exclusive or (XOR)
- opad be the outer padding (0x5c5c5c…5c5c, one-block-long hexadecimal constant)
- ipad be the inner padding (0x363636…3636, one-block-long hexadecimal constant)

Then HMAC(K,m) is mathematically defined by:

HMAC(K,m) = H((K ⊕ opad) ‖ H((K ⊕ ipad) ‖ m))

Where:

- H((K ⊕ ipad) – inner digest;
- H((K ⊕ opad) – outer digest;
- H((K ⊕ ipad) ‖ m) – intermediate digest.

22.2.5.3.3.1 Secure HMAC

The secure HMAC operation in crypto core is executed using several basic hash operations with the assumption that all security sensitive parameters (hash keys, intermediate products and message data) are stored / kept in DMA accessible memory.

The implementation is based on the following requirements:

- XOR-ed keys are prepared in external memory and are read via DMA (alternatively, these can be written via the Host interface).

  If the hash key is longer than the hash block size of 64-bytes, the Host must compress the key using the basic hash operation, which may be performed using a basic hash operation with the crypto core.
- The input message is located in external memory and is read via DMA.
- The intermediate digest state is stored in DMA accessible memory and later read back via DMA for the final hash (the state values can optionally be read and provided via the Host interface).
- The result digest is read via the Host interface.

Figure 22-4 shows the steps that must be performed to implement a secure HMAC using the crypto core.

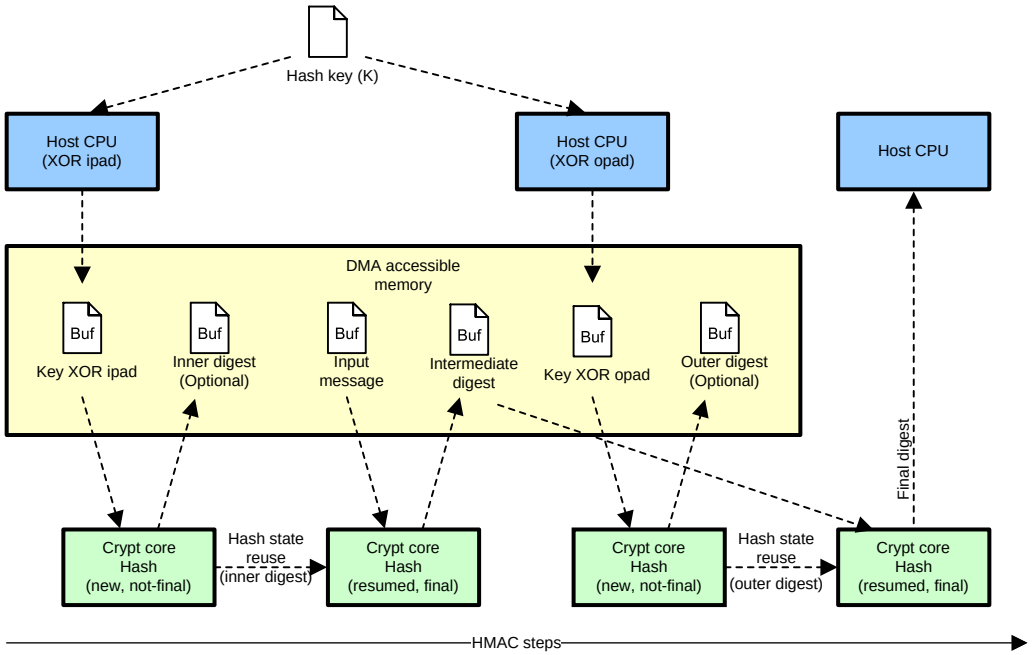


Figure 22-4. Implementation of Secure HMAC Operation

Secure HMAC uses the following basic hash operations with crypto core:

1. A new hash operation is established to hash the padded key (ipad), which is read via DMA, and produces the inner digest. The inner digest remains in the hash engine"s internal state and is used for the next resumed hash.
   The inner digest can be stored back to a pre-allocated area in the external memory such that it can be used for other HMAC operations that use the same key.
   Note that the inner digest calculation requires a new hash operation that in not finalized, since it needs to be resumed in the next step.
2. A resumed hash operation is established to hash the actual message. The initial digest is produced in the previous operation and is still available in the internal state (if the inner digest was prepared in external memory, the Host is able to read this digest and program it via the slave interface). The result of this operation is stored via DMA in the pre-allocated external memory.
3. A new hash operation is established to hash the padded key (opad), which is read via DMA, and produces the outer digest. The outer digest remains in the hash engine"s internal state and can be used for the next resumed hash.
   The outer digest can be stored back to a pre-allocated area in the external memory such that it can be used for other HMAC operation with the same key.
   Note that the outer digest calculation requires a new hash operation that in not finalized, since it needs to be resumed in the next step.
4. A resumed hash operation is established to hash the result of step 2 and produce the final HMAC digest. The initial digest is produced in the previous operation and is still available in the internal state (if the outer digest was prepared in external memory, the Host can read this digest and program it via the slave interface). The final HMAC digest is read via the slave interface.

22.2.5.3.4 Alternative Basic Hash Where Data Originates From the Slave Interface

22.2.5.3.4.1 New Hash Session

The following software example in pseudo code describes the actions that are typically executed by the Host software. It starts the hash engine with a new hash session that receives the input data via the slave interface. In the end, the intermediate digest (non-final hash operation) or the finalized hash digest (final hash operation) is read as result digest via the slave interface.

```
// disable the DMA path
write CTRL_ALG_SEL 0x00000000
// wait until the input buffer is available for writing by the Host
wait HASH_IO_BUF_STAT[2]=='1'
write HASH_MODE = 0x0000_0009 // indicate the start of a new hash session and SHA-256

write HASH_LENGTH_L // the length may be written at any time during session;
write HASH_LENGTH_H // the length must be written when last data has been written
// first block: write and hand-off the block of data
write HASH_DATA_IN_0
...
write HASH_DATA_IN_15
write HASH_IO_BUF_CTRL[6:0]= 0x02 // indicate that a block of data is available

loop: // intermediate blocks
       // wait until the input buffer available for writing by Host
       wait HASH_IO_BUF_STAT[2]=='1'
       // write the intermediate block and hand off the data
       write HASH_DATA_IN_0
       ...
       write HASH_DATA_IN_15
       write HASH_IO_BUF_CTRL[6:0]= 0x02 // indicate that a block of data is available
endloop
wait HASH_IO_BUF_STAT[2]=='1'
// write the last block and hand-off the data
// Note: if last block is misaligned, the last 32-bit word must be padded (any data is
// accepted)
write HASH_DATA_IN_0
...
write HASH_DATA_IN_15
// if an intermediate digest is required (input data is hash block size aligned)
write HASH_IO_BUF_CTRL[6:0]= 0x42 // indicate that data is available and get the
           // intermediate digest (no internal padding)
// else if final digest is required (input data padded by hash engine)
write HASH_IO_BUF_CTRL[6:0]= 0x22 // indicate that data is available and get the final
    // digest of the padded data
// wait until output data ready
wait HASH_IO_BUF_STAT[0] == '1'
// read digest
read HASH_DIGEST_A
...
read HASH_DIGEST_H
write HASH_IO_BUF_CTRL = 0x01 // acknowledge that the digest is read
// end of algorithm
```

22.2.5.3.4.2 Resumed Hash Session

The following software example in pseudo code describes the actions that are typically executed by the Host software. It starts the hash engine with a resumed hash session that receives the input data via the slave interface. In the end, the intermediate digest (non-final hash operation) or the finalized hash digest (final hash operation) is read as result digest via the slave interface.

```
// disable DMA path
write CTRL_ALG_SEL 0x00000000
// wait until the input buffer is available for writing by the Host
wait HASH_IO_BUF_STAT[2]=='1'
write HASH_MODE = 0x0000_0008 // indicate the start of a resumed hash session andSHA256

// write initial digest
write HASH_DIGEST_A
...
write HASH_DIGEST_H

write HASH_LENGTH_L // the length may be written at any time during session;
```

```
write HASH_LENGTH_H // the length must be written when last data has been written
// first block: write and hand-off the block of data
write HASH_DATA_IN_0
...
write HASH_DATA_IN_15
write HASH_IO_BUF_CTRL[6:0]= 0x02 // indicate that a block of data is available

loop: // intermediate blocks
       // wait until the input buffer available for writing by Host
       wait HASH_IO_BUF_STAT[2]=='1'
       // write the intermediate block and hand off the data
       write HASH_DATA_IN_0
       ...
       write HASH_DATA_IN_15
       write HASH_IO_BUF_CTRL[6:0]= 0x02 // indicate that a block of data is available
       if more than one input block, goto loop
endloop
wait HASH_IO_BUF_STAT[2]=='1'
// write the last block and hand-off the data
// Note: if last block is misaligned, the rest must be padded (any data is accepted)
write HASH_DATA_IN_0
...
write HASH_DATA_IN_15
// if intermediate digest is required (input data is hash block size aligned)
write HASH_IO_BUF_CTRL[6:0]= 0x42 // indicate that data is available and get the
     // intermediate digest
// else if final digest is required (input data padded by hash engine)
write HASH_IO_BUF_CTRL[6:0]= 0x22 // indicate that data is available and get the final
     // digest of the padded data
// wait until output data ready
wait HASH_IO_BUF_STAT[0] == '1'
// read digest
read HASH_DIGEST_A
...
read HASH_DIGEST_H
write HASH_IO_BUF_CTRL = 0x01 // acknowledge reading of the digest
// end of algorithm
```

22.2.5.4 Encryption/Decryption

The crypto engine (AES) transfers data over the following interfaces:

- It accepts input data from two sources: AHB slave interface and DMA. Within one operation, it is possible to combine data from these two sources: write data from the slave interface and later write data from the DMA to complete the operation.
- Input IV and length must be supplied via the AHB slave interface. The output IV can be read via the slave interface only.
- Result data must be read via the same interface as the input data: either via the slave interface or via DMA
- The result tag for operations with authentication can be read via the slave interface or via DMA.

22.2.5.4.1 Data Format and Byte Order

The following examples show how the data must be submitted to the AES Engine. Because the AHB slave interface is more or less transparent for data (no data modifications), the alignment for the register interface is identical as described in the following examples.

Note: The highlighted byte is a first byte of the key or message.

NIST SP 800-38a, AES-ECB 256-bit Encrypt:

```
AES Key In:  603deb10 15ca71be 2b73aef0 857d7781
             1f352c07 3b6108d7 2d9810a3 0914dff4
AES Data In: 6bc1bee2 2e409f96 e93d7e11 7393172a
```

```
AES Data Out: f3eed1bd b5d2a03c 064b5a7e 3db181f8
```

AES Key in external RAM, loaded via key store module:

```
Word_0[31:0]: 10eb3d60
Word_1[31:0]: be71ca15
Word_2[31:0]: f0ae732b
Word_3[31:0]: 81777d85
Word_4[31:0]: 072c351f
Word_5[31:0]: d708613b
Word_6[31:0]: a310982d
Word_7[31:0]: f4df1409
```

Input data via slave interface or DMA:

```
 AES_DATA_IN_0[31:0]:e2bec16b
AES_DATA_IN_1[31:0]:969f402e
AES_DATA_IN_2[31:0]:117e3de9
AES_DATA_IN_3[31:0]: 2a179373
```

Output data via slave interface or DMA:

```
AES_DATA_OUT_0[31:0]: bdd1eef3
AES_DATA_OUT_1[31:0]: 3ca0d2b5
AES_DATA_OUT_2[31:0]: 7e5a4b06
AES_DATA_OUT_3[31:0]: f881b13d
```

22.2.5.4.2 Key Store

Before any encryption/decryption operation starts, the key store module must have at least one key loaded and available for crypto operations. Keys can only be loaded from external memory using a DMA operation. DMAC channel 0 (inbound) is used for this purpose.

22.2.5.4.2.1 Load Keys From External Memory

The following software example in pseudo code describes the actions that are typically executed by the Host software to load one or more keys into the key store module.

```
// configure master control module
write CTRL_ALG_SEL 0x0000_0001 // enable DMA path to the key store module
write CTRL_INT_CLR 0x0000_0001 // clear any outstanding events

// configure key store module (area, size)
write KEY_STORE_SIZE 0x0000_0001 // 128-bit key size
write KEY_STORE_WRITE_AREA 0x0000_0001 // enable keys to write (e.g. Key 0)

// configure DMAC
write DMAC_CH0_CTRL 0x0000_00001 // enable DMA channel 0
write DMAC_CH0_EXTADDR <ext_memory_address> // base address of the key in ext. memory
write DMAC_CH0_DMALENGTH <length> // total key length in bytes (e.g. 16 for 1 x 128-bit
       // key)
// wait for completion
wait CTRL_INT_STAT[0]=='1' // wait for operation completed
check CTRL_INT_STAT[31:30] == '00' // check for absence of errors in DMA and key store

write CTRL_INT_CLR 0x0000_0001 // acknowledge the interrupt
write CTRL_ALG_SEL 0x0000_0000 // disable master control/DMA clock

// check status
check KEY_STORE_WRITTEN_AREA 0x0000_00001 // check that Key 0 was written
// end of algorithm
```

22.2.5.4.3 Basic AES Modes

22.2.5.4.3.1 AES-ECB

For AES-ECB operations, the following configuration parameters are required:

- Key from the key store module
- Control register settings (mode, direction, key size)
- Length of the data

The length field can have any value. If a data stream is done and the next data stream uses the same key and control, only the length field has to be written with a new value. The length field may also be zero, for continued processing.

22.2.5.4.3.2 AES-CBC

For AES-CBC operations, the following configuration parameters are required:

- Key from the key store module
- IV from the slave interface
- Control register settings (mode, direction, key size)
- Length of the data

The length field can have any value. If a data stream is done and the next data stream uses the same key and control; it is allowed to write only the IV and length field with a new value. The length field may also be zero, for continued processing.

If the result IV must be read by the Host, the save_context bit must be set to „1", after processing the programmed number of bytes.

22.2.5.4.3.3 AES-CTR

For AES-CTR operations, the following configuration parameters are required:

- Key from the key store module
- IV from the slave interface, including initial counter value (usually 0x0000_0001)
- Control register settings (mode, direction, key size)
- Length of the data (may be non-block size aligned)

The length field can have any value. If a data stream is done and the next data stream uses the same key and control, only the IV and length field are allowed to written with a new value. The length field can be zero, resulting in continued processing.

If the result IV must be read by the Host the save_context bit must be set to „1", after processing the programmed number of bytes.

22.2.5.4.3.4 Programming Sequence With DMA Data

The following software example in pseudo code, describes the actions that are typically executed by the Host software to encrypt (using a basic AES mode) a message, stored in external memory and place an encrypted result into a pre-allocated area in the external memory.

```
// configure the master control module
write CTRL_ALG_SEL 0x0000_0002 // enable the DMA path to the AES engine
write CTRL_INT_CLR 0x0000_0001 // clear any outstanding events

// configure the key store to provide pre-loaded AES key
write KEY_STORE_READ_AREA 0x0000_0000 // load the key from ram area 0 (NOTE: The key
              // must be pre-loaded to this area)

wait KEY_STORE_READ_AREA[31]=='0' // wait until the key is loaded to the AES module
check CTRL_INT_STAT[29] = '0' // check that the key is loaded without errors
// Write the IV for non-ECB modes
```

```
// The IV must be written with the same conventions as the data (refer to 6.4.1)
if ((not ECB mode) and (not IV reuse)) then:
// write the initialization vector when a new IV is required
write AES_IV_0
...
write AES_IV_3
endif

// configure AES engine
write AES_CTRL = 0b0010_0000_0000_0000_
            0000_0000_0010_1100 // program AES-CBC-128 encryption and save IV
write AES_C_LENGTH_0 // write length of the message (lo)
write AES_C_LENGTH_1 // write length of the message (hi)

write DMAC_CH0_CTRL 0x0000_00001 // enable DMA channel 0// configure DMAC
write DMAC_CH0_EXTADDR <address> // base address of the input data in ext. memory
write DMAC_CH0_DMALENGTH <length> // input data length in bytes, equal to the message
          // length (may be non-block size aligned)
write DMAC_CH1_CTRL 0x0000_00001 // enable DMA channel 1
write DMAC_CH1_EXTADDR <address> // base address of the output data buffer
write DMAC_CH1_DMALENGTH <length> // output data length in bytes, equal to the result
          // data length (may be non-block size aligned)

// wait for completion
wait CTRL_INT_STAT[0]=='1' // wait for operation completed
check CTRL_INT_STAT[31] == '0' // check for absence of errors
write CTRL_ALG_SEL 0x0000_0000 // disable master control/DMA clock

if (not ECB mode) then: // only if the IV needs to be re-used/read
wait AES_CTRL[30]=='1' // wait for context ready bit [30]
read AES_IV_0
...
read AES_IV_3 // this read clears the 'saved_context_ready' flag
endif

// end of algorithm
```

22.2.5.4.4 AES-GCM

For AES-GCM operations, the following configuration parameters are required:

- Key from the key store module
- IV from the slave interface, including initial counter value of „1"
- Control register settings (mode, direction, key size)
- Length of the cipher data (may be non-block size aligned)
- Length of the AAD data (may be non-block size aligned)

The AES module is used in the mode where it calculates the Y0 encrypted and H (hash key)

internally based on cipher key from the key store module.

The AAD and cryptographic data may end misaligned. In this case, the module internally pads both to a 128-bit boundary with zeroes. Padding is done as follows: the AAD and crypto data padding satisfies the bit string: 0n, with 0≤n≤127 such that the input AAD and data block lengths including padding are 128-bit aligned. The AAD data must be transferred to the AES engine with a separate DMA operation (it may not be combined with the payload data) or using slave transfers.

The length field can have any value. If a data stream is done and the next data stream uses the same key and control, only the IV and length fields can be written with a new value. It is not allowed to write both length fields with zeroes. A GCM operation cannot be interrupted.

The result TAG is typically read via the slave interface, but can also be written to an external memory location via a separate DMA operation.

22.2.5.4.4.1 Programming Sequence

The following software example in pseudo code, describes the actions that are typically executed by the Host software to encrypt and authenticate a message using AES-GCM mode. The message (AAD and payload data) is fetched from external memory and the encrypted result is placed into a pre-allocated area in the external memory. The result TAG is read via the slave interface.

The following sequence processes a packet of at least one byte of AAD data and at least one crypto data byte.

```
// configure the master control module
write CTRL_ALG_SEL 0x0000_0002 // enable the DMA path to the AES engine
write CTRL_INT_CLR 0x0000_0001 // clear any outstanding events

// configure the key store to provide a pre-loaded AES key
write KEY_STORE_READ_AREA 0x0000_0000 // load the key from ram area 0 (NOTE: The key
              // must be pre-loaded to this area)
wait KEY_STORE_READ_AREA[31]=='0' // wait until the key is loaded to the AES module
check CTRL_INT_STAT[29] = '0'// check that the key is loaded without errors

// write the initialization vector
write AES_IV_0
...
write AES_IV_3

// configure the AES engine
write AES_CTRL = 0b0010_0000_0000_0011_
              0000_0000_0100_1100 // program AES-GCM-128 encryption (autonomous)
write AES_C_LENGTH_0// write the length of the crypto block (lo)
write AES_C_LENGTH_1// write the length of the crypto block (hi)
             // (may be non-block size aligned)
write AES_AUTH_LENGTH // write the length of the AAD data block
             // (may be non-block size aligned)

// configure DMAC to fetch the AAD data
write DMAC_CH0_CTRL 0x0000_00001 // enable DMA channel 0
write DMAC_CH0_EXTADDR <address> // base address of the AAD data in ext. memory
write DMAC_CH0_DMALENGTH <length> // AAD data length in bytes, equal to the aad
              // length len({aad data})
              // (may be non-block size aligned)

// wait for completion of the AAD data transfer
wait CTRL_INT_STAT[1]=='1'// wait for DMA_IN_DONE
check CTRL_INT_STAT[31]=='0'// check for the absence of errors

// configure DMAC to process the payload data
write DMAC_CH0_CTRL 0x0000_00001 // enable DMA channel 0
write DMAC_CH0_EXTADDR <address> // base address of the payload data in ext. memory
write DMAC_CH0_DMALENGTH <length> // payload data length in bytes, equal to the payload
              // length len({crypto_data})
              // (may be non-block size aligned)
write DMAC_CH1_CTRL 0x0000_00001 // enable DMA channel 1
write DMAC_CH1_EXTADDR <address> // base address of the output data buffer
write DMAC_CH1_DMALENGTH <length> // output data length in bytes, equal to the result
             // data length len({crypto data})
             // (may be non-block size aligned)

// wait for completion
wait CTRL_INT_STAT[0]=='1'// wait for operation completed
check CTRL_INT_STAT[31]=='0'// check for the absence of errors
write CTRL_ALG_SEL 0x0000_0000// disable the master control/DMA clock

// read tag
wait AES_CTRL[30]=='1' // wait for the context ready bit [30]
read AES_TAG_OUT_0
...
```

```
read AES_TAG_OUT_3 // this read clears the 'saved_context_ready' flag
// end of algorithm
```

22.2.5.4.5 CBC-MAC

For CBC-MAC operations, the following configuration parameters are required:

- Key from the key store module
- IV must be written with zeroes
- Control register settings (mode, direction, key size)
- Length of the authenticated data (may be non-block size aligned)
- The input data may end misaligned for CBC-MAC operations. If this is the case, the crypto core will internally pad the last input data block.

The length field can have any value. If a data stream is done and the next data stream uses the same key and control, writing only a part of the next context is not allowed. A new data stream must always write the complete context. The length field may never be written with zeroes.

22.2.5.4.5.1 Programming Sequence

The following software example in pseudo code, describes the actions that are typically executed by the Host software to authenticate a message, stored in external memory, with AES-CBC-MAC mode. The result TAG is read via the slave interface.

The following sequence processes a packet of at least one input data byte.

```
// configure the master control module
write CTRL_ALG_SEL 0x0000_0002 // enable the DMA path to the AES engine
write CTRL_INT_CLR 0x0000_0001 // clear any outstanding events

// configure the key store to provide a pre-loaded AES key
write KEY_STORE_READ_AREA 0x0000_0000 // load the key from ram area 0 (NOTE: The key
              // must be pre-loaded to this area)
wait KEY_STORE_READ_AREA[31]=='0' // wait until the key is loaded to the AES module
check CTRL_INT_STAT[29] = '0'// check that the key is loaded without errors

// write the initialization vector
write AES_IV_0
...
write AES_IV_3

// configure the AES engine
write AES_CTRL = 0b0010_0000_0000_0000_
              1000_0000_0100_1100 // program AES-CBC-MAC-128 authentication
write AES_C_LENGTH_0 // write length of the crypto block (lo)
write AES_C_LENGTH_1 // write the length of the crypto block (hi)
              // (may be non-block size aligned)

/write DMAC_CH0_CTRL 0x0000_00001 // enable DMA channel 0/ configure DMAC
write DMAC_CH0_EXTADDR <address> // base address of the input data in ext. memory
write DMAC_CH0_DMALENGTH <length> // input data length in bytes, equal to the message
              // length len({aad data, pad, crypto_data, pad})
              // (may be non-block size aligned)

// wait for completion
wait CTRL_INT_STAT[0]=='1' // wait for operation completed
check CTRL_INT_STAT[31]=='0' // check for the absence of errors
write CTRL_ALG_SEL 0x0000_0000 // disable master control/DMA clock

// read tag
wait AES_CTRL[30]=='1' // wait for the context ready bit [30]
read AES_TAG_OUT_0
...
read AES_TAG_OUT_3 // this read clears the 'saved_context_ready' flag
```

```
// end of algorithm
```

22.2.5.4.6 AES-CCM

For AES-CCM operations, the following configuration parameters are required:

- Key from the key store module
- The IV must be written with the flags for the cryptographic operation and the NONCE bytes, for both authentication and encryption (see section 4.5.3)
- Control register settings (mode, direction, key size)
- Length of the crypto data (may be non-block size aligned)
- Length of the AAD data; must be less than 216 - 28 bytes (may be non-block size aligned)

CCM-L must be 001, 011 or 111, representing a crypto data length field of 2, 4 or 8 bytes respectively.

CCM-M can be set to any value and has no effect on the processing. The Host must select the valid TAG bytes from the 128-bit TAG.

The AAD and cryptographic data may end misaligned. In this case, the crypto core will pad both data types to a 128-bit boundary with zeroes. Padding is done as follows: the AAD and crypto data padding will satisfy the bit string: 0n, with 0≤n≤127, such that the input data block length including padding is 128-bit aligned. The AAD data must be transferred to the AES engine with a separate DMA operation (it may not be combined with the payload data) or using slave transfers.

The context length field can have any value. If a data stream is done and the next data stream uses the same key and control, only the IV and length fields can be written with a new value. It is not allowed to write both length fields with zeroes.

The result TAG is typically read via the slave interface, but can also be written to an external memory location via a separate DMA operation.

22.2.5.4.6.1 Programming Sequence

The following software example in pseudo code, describes the actions that are typically executed by the Host software to encrypt and authenticate a message (AAD and payload data), stored in external memory, with AES-CCM mode. The encrypted result is placed into a pre-allocated area in external memory. The result TAG is read via the slave interface.

The following sequence processes a packet of at least one byte of AAD data and at least one crypto data byte.

```
// configure the master control module
write CTRL_ALG_SEL 0x0000_0002 // enable the DMA path to the AES engine
write CTRL_INT_CLR 0x0000_0001 // clear any outstanding events

// configure the key store to provide pre-loaded AES key
write KEY_STORE_READ_AREA 0x0000_0000 // load the key from ram area 0 (NOTE: The key
             // must be pre-loaded to this area)
wait KEY_STORE_READ_AREA[31]=='0' // wait until the key is loaded to the AES module
check CTRL_INT_STAT[29] = '0'// check that the key is loaded without errors

// write the initialization vector
write AES_IV_0
...
write AES_IV_3

// configure the AES engine
write AES_CTRL = 0b0010_0000_0101_1100_
             0000_0000_0100_1100 // program AES-CCM-128 encryption (M=1, L=3)
write AES_C_LENGTH_0// write the length of the crypto block (lo)
write AES_C_LENGTH_1// write the length of the crypto block (hi)
             // (may be non-block size aligned)
write AES_AUTH_LENGTH // write the length of the AAD data block
             // (may be non-block size aligned)
```

```
// configure DMAC to fetch the AAD data

write DMAC_CH0_CTRL 0x0000_00001 // enable DMA channel 0
write DMAC_CH0_EXTADDR <address> // base address of the AAD input data in ext. memory
write DMAC_CH0_DMALENGTH <length> // AAD data length in bytes, equal to the AAD
              // length len({aad data})
              // (may be non-block size aligned)

// wait for completion of the AAD data transfer
wait CTRL_INT_STAT[1]==’1’// wait for DMA_IN_DONE
check CTRL_INT_STAT[31]==‘0’// check for the absence of errors

// configure DMAC
write DMAC_CH0_CTRL 0x0000_00001 // enable DMA channel 0
write DMAC_CH0_EXTADDR <address> // base address of the payload data in ext.    memory
write DMAC_CH0_DMALENGTH <length> // payload data length in bytes, equal to the message
              // length len({crypto_data})
write DMAC_CH1_CTRL 0x0000_00001 // enable DMA channel 1
write DMAC_CH1_EXTADDR <address> // base address of the output data buffer
write DMAC_CH1_DMALENGTH <length> // output data length in bytes, equal to the result
              // data length len({crypto data})

// wait for completion
wait CTRL_INT_STAT[0]==’1’// wait for operation completed
check CTRL_INT_STAT[31]==‘0’// check for the absence of errors
write CTRL_ALG_SEL 0x0000_0000 // disable the master control/DMA clock

// read tag
wait AES_CTRL[30]==’1‘ // wait for the context ready bit [30]

read AES_TAG_OUT_0
...
read AES_TAG_OUT_3 // this read clears the ‘saved_context_ready’ flag
// end of algorithm
```

22.2.5.5 Exceptions Handling

22.2.5.5.1 Soft Reset

If required, the AES module can be forced to abort its current active operation and put itself to its idle, state using the soft reset.

The "idle state" means:

- The DMAC is not actively performing DMA operations
- The cryptographic modules are in idle state
- The key store module does not have any keys loaded
- The master control module is in idle state
- A soft reset should be executed in the following order:
- If DMA is used and in operation, it must be stopped (refer to section 6.2.2).
- The master control module must be reset via the CTRL_SW_RESET register (refer to section 4.4.3).
- Write the mode and length registers (for both hash and crypto cores) with zeroes.
  These are:
  - AES_CTRL
  - AES_C_LENGTH_0
  - AES_C_LENGTH_1
  - AES_AUTH_LENGTH
  - HASH_MODE
  - HASH_LENGTH_L

– HASH_LENGTH_H

22.2.5.5.2 External Port Errors

The AHB master interface and the DMAC inside the crypto core can detect AHB port errors that are received via the m_h_resp signal.

In this situation, the DMAC disables all channels such that no new transfers are requested, while the error is captured in its status registers. The DMA port error status register (DMAC_PERSR) contains information about the active channel when the AHB port error occurred. DMAC indicates the channel completion to the master control module.The recovery procedure is as follows:

- Issue a soft reset to the DMAC via the DMAC_SWRES register to clear the DMAC_PERSR register and initialize the channels to their default state;
- Issue a soft reset to the master control module to clear its intermediate state.

22.2.5.5.3 Key Store Errors

Key store error generation is implemented for debugging purposes. In normal/specified operation, the crypto core key store writes and reads should not trigger any errors. A bus error is the only exceptional case (as explained in 6.5.2) that can result in a key store write error.

The key store module checks that the keys are properly written to the key store RAM. When a key write error occurs, the KEY_STR_WR_ERR flag is asserted in the CTRL_INT_STAT register. In this case, the key will not be stored. The Host must check the status of the KEY_STR_WR_ERR flag and must ensure that the corresponding RAM area is not used for AES operations.

If, due to software malfunction, the Host tries to use a key from a non-written RAM area, the key store module generates a read error. In this case the KEY_STR_RD_ERR flag is asserted in CTRL_INT_STAT. The Host must check the status of this flag and ensure that all the remaining steps for the AES operation are not performed.

Note: In case of a read error, the key store writes a key with all bytes set to zero to the AES engine.

22.2.6 Conventions and Compliances

22.2.6.1 Conventions Used in This Manual

22.2.6.1.1 Acronyms

AES	Advanced Encryption Standard
AES-CCM	AES Counter with CBC-MAC
AHB	Advanced High-speed Bus
AMBA	Advanced Micro-controller Bus Architecture
CBC	Cipher Block Chaining
CCM	Counter with CBC-MAC
CM	Crypto Module
CTR	Counter Mode
DMAC	DMA Controller
DPRAM	Dual port Random Access Memory
ECB	Electronic Code Book
EIP	Embedded Intellectual Property
FIFO	First In First Out
FIPS	Federal Information Processing Standard
GB	Gigabyte
Gbit	Giga bit
Gbps	Giga bits per second

HMAC	Hashed MAC
HW	Hardware
ICM	Integer Counter Mode
IETF	Internet Engineering Task Force
IP	Internet Protocol/ Intellectual Property
IV	Initialization Vector
kB	Kilo byte
kbit	Kilo bit
kbps	Kilo bit per second
LSB	Least Significant Bit
LSW	Least Significant Word
MAC	Message Authentication Code
MB	Megabyte
Mbit	Mega bit
Mbps	Mega bits per second
ME	Mobile Equipment
MSB	Most Significant Bit
MSW	Most Significant Word
OS	Operating System
RFC	Request for Comments
SHA	Secure Hash Algorithm
SPRAM	Single Port Random Access Memory
SRAM	Static Random Access Memory
TCM	Tightly Coupled Memory (memory interface protocol)

22.2.6.1.2 Terminology

This manual makes frequent use of certain terms. These terms refer to structures that the crypto core uses for operations.

External memory

A memory that is externally attached to the crypto core AHB master port, it is only accessible using DMAC operations.

Slave interface (Host processor bus)

Interface of the crypto core that is used by the Host processor to read or write registers of the engine.

Tag or digest

Two interchangeable terms that indicates the result of an authentication operation. Term „digest" is used for regular hash operations (e.g. SHA-256), while „tag" is used for authenticated encryption operations (AES-GCM/CCM).

Crypto context

Collection of parameters that define the crypto operation: mode, key, IV etc.

22.2.6.1.3 Formulae and Nomenclature

This document contains formulas and nomenclature for different data types. The presentation of syntax is given as follows:

0x00 or 0h	Hexadecimal value
0b	Binary value
0d	Decimal value
„0"	Digital logic 0 or LOW
„1"	Digital logic 1 or HIGH
bit	Binary digit
8 bits	1 byte
16 bits	half word
32 bits	word
64 bits	dual-word
128 bits	quad-word
MOD	MODulo
REM	REMainder
A & B	A Logical AND B
A OR B	A Logical OR B
NOR	Logical NOR
NOT A	Logical NOT
A NOR B	A Logical NOR B
AB	A logic eXclusive OR B or XOR
XNOR	logic eXclusive NOR
NAND	Logical NAND
DIV	Integer DIVision
||	Concatenation
[n:m]	Size of a register or signal in bits where $n > m^2$ (1)

(1) [31:0] indicates a size of 32 bits with most significant bit 31 and least significant bit 0. [11:3] indicates a size of 9 bits with most significant bit 11 and least significant bit 3.
[31:0] indicates a size of 32 bits with most significant bit 31 and least significant bit 0. [11:3] indicates a size of 9 bits with most significant bit 11 and least significant bit 3.

22.2.6.1.4 Register Information

Registers within this document are shown as follows:

REGISTER_HEAD, (Write only), 32-bit Address Offset: 0x7000C																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The register name, accessibility and the Host address location for direct access are on the top lines. The table shows all the register bit fields, a supporting description is included in the text below the register table. Reserved fields are shaded gray. The bottom row in the register graphic shows the power-up / reset default setting of the register for read.

22.2.6.2 Compliances

The list of crypto core features that operate in compliance with the standards that includes but is not limited to:

- AES encryption in ECB and CBC modes (FIPS-197)
- MD-5 and SHA-1 Hashing with HMAC (RFC 1321, RFC 2104, FIPS 180-1)
- MD-5, SHA-1 and SHA-2 Hashing with HMAC (RFC 1321, RFC 2104, FIPS 180-1, FIPS 180-2)

22.3 Public Key Processor

22.3.1 Advanced Interrupt Controller

The device contains an Advanced Interrupt Controller (AIC) module to provide a single interrupt output with the ability to enable or disable separate interrupt events. For further details about the AIC, see Section 22.3.3, *Advanced Interrupt Controller*.

22.3.2 Registers

22.3.2.1 Register Address Map

Mapping of the internal registers, including PKA engines, to the external address bus is illustrated in Table 22-89. The register spaces are selectable using bits [15:14] of the address bus.

Table 22-89. Mapping of Internal Address Bus to the External Address Bus

Register Address Map																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Masked for PLB configuration and skipped for AHB configuration																Register Space selection		Sub-Module Address (32-bit aligned)													
PKP internal registers and AIC registers																															
–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	1	0	A	A	A	A	A	A	A	A	A	A	A	A	0	0
PKA Engine																															
–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0	1	0	A	A	A	A	A	A	A	A	A	A	A	0	0
PKA RAM (Program RAM)																															
–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0	1	1	A	A	A	A	A	A	A	A	A	A	A	0	0

The register interface of the *PKP* supports only 32-bit word aligned accesses. If a misaligned access on the bus occurs, the slave module generates a ‘slave error’ interrupt.

The PKA engines use the 32-bit word addressing on a register address line, hence in Table21, their addresses are concatenated with two zero bits to become byte address.

22.3.2.2 Register Description

22.3.2.2.1 Engine Registers

Internal register information for the PKA is available in Section 22.1.5, *Interfaces*. This document also contains information on selecting the PKA size and connecting the PKA RAM modules.Internal register information for AIC is available in Section 22.3.3, *Advanced Interrupt Controller*.

22.3.2.2.1.1 PKP_REVISION Register

Table 22-90. PKP_REVISION

PKP_REVISION (read only), Address Offset (0xBFFC)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED				Major HW revision				Minor HW revision				HW patch level				Complement of basic EIP number								Basic EIP number							
0	0	0	0	2				0				x	x	x	x	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Bit	Name	Function
[7:0]	EIP-number	These bits encode the EIP number for the *PKP*.
[15:8]	Bit-by-bit complement of EIP-number	These bits simply contain the complement of bits [7:0], used by a driver to ascertain that the PKP_REVISION register is indeed read.
[19:16]	Patch level	These bits encode the hardware patch level for this module – they start at value 0 on the first release
[23:20]	Minor version number	These bits encode the minor version number for this module
[24:27]	Major version number	These bits encode the major version number for this module
[31:28]	Reserved	These bits should be ignored on a read

22.3.2.2.1.2 PKP_OPTIONS Register

Table 22-91. PKP_OPTIONS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																					aic_present	eip76_present	eip28_present	RESERVED				xi_interface	ahb_is_asynchronous	ahb_interface	plb_ interface
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	v	v	v	0	0	0	0	v	v	v	v

Bit	Name	Function
[0]	plb_interface	When set to '1': indicates that the *PKP* is equipped with a PLB interface.
[1]	ahb_interface	When set to '1': indicates that the *PKP* is equipped with an AHB interface.
[2]	ahb_is_asynchronous	When set to '1': indicates that the AHB interface is asynchronous. Only applicable when ahb_interface is '1'.
[3]	axi_interface	When set to '1': indicates that the *PKP* is equipped with an AXI interface.
[7:4]	Reserved	These bits should be ignored on a read.

[8]	eip28_present	When set to '1': indicates that an PKA is included in the *PKP*
[9]	eip76_present	When set to '1': indicates that an TRNG is included in the *PKP*
[10]	aic_present	When set to '1': indicates that an AIC is included in the *PKP*
[31:11]	Reserved	These bits should be ignored on a read.

22.3.3 Advanced Interrupt Controller (Optional)

22.3.3.1 Introduction

The device optionally includes an Advanced Interrupt Controller (AIC) that is program configured to generate an interrupt when certain operations are complete, for example, PKA operations and interrupts caused by errors.

22.3.3.2 Functional Description

The AIC provides an interrupt-combining unit. The interrupt controller receives interrupt signals from a number of interrupt sources and combines them into one interrupt output. The interrupt controller is managed using the following register groups:

- Five registers to control polarity, edge or level detection and enabling of individual interrupts,
- One acknowledge register (to clear edge detected interrupts),
- Two status registers:
  - A raw source status register after edge detection, if edge selected.
  - A status register after masking with the interrupt enable control bits.
- Two configuration status registers: module version and module options.

Figure 22-5 shows the functional logic of one interrupt bit 'slice' and the NOR gate that generates the active LOW interrupt output.

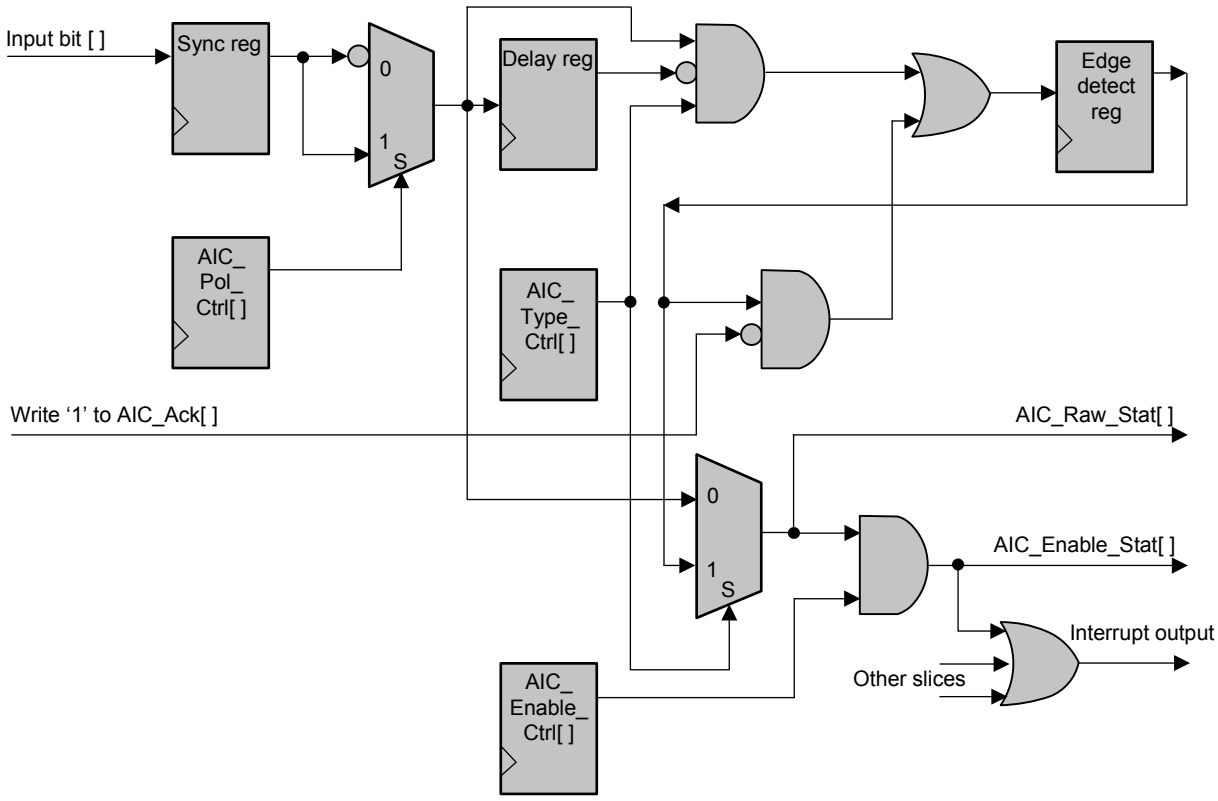


Figure 22-5. AIC: Functional Logic of one Interrupt Source

As Figure 22-5 shows, it is advisable to clear the edge detect register with a write to the AIC_ACK register location before switching interrupts from edge to level type. This prevents the immediate raising of an interrupt when edge detection is enabled.

22.3.3.3 Interrupt Sources

Table 22-92. PKP Interrupt Sources

Name	Description
PKA Interrupts	
pka_int[0]	The LNME is ready. This signal is available as separate output line. For debugging purposes only.
pka_int[1]	The main PKA interrupt that is a combined PKCP and Sequencer ready interrupt. This signal is available as a separate output line.When HIGH, indicates that the PKA engine is ready for the next command and results from previous commands have been written to registers and PKA data RAM. This is the main interrupt output.
pka_int[2]	The sequencer is ready. This signal is available as separate output line. For debugging purposes only.
TRNG Interrupts	
trng_irq	Active HIGH signal indicating an active interrupt being generated from within the TRNG – read the TRNG_STATUS register for more information.
Interface Interrupts	
sl_err_int	Slave error interrupt. This signal is available as separate output line.When HIGH this indicates slave bus error. Refer to the Section 3.1 for exact condition for specific bus type.

Note: *The TRNG and PKA interrupts are present only when the respective modules are present, otherwise these sources are tied to zero.*

22.3.3.4 AIC Registers

The interrupt registers is a set of six control and status registers that allow the *Host* system to enable or disable the interrupts, to check the status of the most recent interrupt and to force an interrupt.

The AIC allows programming connection of any interrupt source signal to the dedicated interrupt output of the *AIC*. The AIC_ENABLE_CTRL register provides the mask to select which interrupt source is enabled. Two status registers, AIC_RAW_STAT and AIC_ENABLED_STAT, allow the *Host* to read the status the interrupt source, either before or after the mask is applied.

All of the interrupt sources are level or edge events. The AIC_TYPE_CTRL register is set by the driver to either level or edge for each interrupt. The AIC_POL_CTRL register controls the polarity of the signal.

These interrupts are latched at both status registers in case edge detection is selected. The edge detectors are reset by clearing the interrupts using the AIC_ACK registers.

22.3.3.4.1 AIC Polarity Control Register (AIC_POL_CTRL)

This register is used to configure the signal polarity for each individual interrupt. During the initialization phase of the *PKP*, the driver must set each interrupt in this register to (high level/rising edge or low level/falling edge) as described in the table below.

Table 22-93. AIC_POL_CTRL

AIC_POL_CTRL (Read/Write), Address Offset (0x8000)																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																										sl_err_int	RESERVED	trng_irq	pka_int[2]	pka_int[1]	pka_int[0]
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
PolarityControl		Individual polarity (high level/rising edge or low level/falling edge) control bits per interrupt input. The signal is as follows: 0=Low level/falling edge 1=High level/rising edge Note: *The PolarityControl must be set to 1 for all interrupts during the initialization phase of the PKP.*
[0]	pka_int[0]	1 = high level/rising edge
[1]	pka_int[1]	1 = high level/rising edge
[2]	pka_int[2]	1 = high level/rising edge
[3]	trng_irq	1 = high level/rising edge
[4]	Reserved	This bit should be written with a ‘0’ and ignored on a read.
[5]	sl_err_int	1 = high level/rising edge
[31:6]	Reserved	These bits should be written with a ‘0’ and ignored on a read.

22.3.3.4.2 AIC Type Control Register (AIC_TYPE_CTRL)

This register is used to configure the signal type for each individual interrupt. During the initialization phase of the PKP, the driver must set each interrupt in this register to level or edge as described in the table below.

Table 22-94. AIC_TYPE_CTRL

AIC_TYPE_CTRL (Read/Write), Address Offset (0x8004)																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																										sl_err_int	RESERVED	trng_irq	pka_int[2]	pka_int[1]	pka_int[0]
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
TypeControl		Signal type (level or edge) control bits for each interrupt input. 0=level (the interrupt source level determines the raw status) 1=edge (the interrupt source is connected to edge detector and output of edge detector determines the raw status)
[0]	pka_int[0]	1 = Edge
[1]	pka_int[1]	1 = Edge
[2]	pka_int[2]	1 = Edge
[3]	trng_irq	1 = Edge
[4]	Reserved	Reserved These bits should be written with a ‘0’ and ignored on a read.
[5]	sl_err_int	1 = Edge
[31:6]	Reserved	Reserved These bits should be written with a ‘0’ and ignored on a read.

22.3.3.4.3 4.4.3 AIC Enable Control Register (AIC_ENABLE_CTRL)

Table 22-95. AIC_ENABLE_CTRL

AIC_ENABLE_CTRL (Read/Write), Address Offset (0x8008)																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																										sl_err_int	RESERVED	trng_irq	pka_int[2]	pka_int[1]	pka_int[0]
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[5], [3:0]	EnableControl	Individual enable control bits per interrupt input. 0=Disabled 1=Enabled
[31:6], [4]	Reserved	These bits should be written with a ‘0’ and ignored on a read.

22.3.3.4.4 AIC Raw Source Status Register (AIC_RAW_STAT)

Table 22-96. AIC_RAW_STAT

AIC_RAW_STAT (Read Only), Address Offset (0x800C)																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																										sl_err_int	RESERVED	trng_irq	pka_int[2]	pka_int[1]	pka_int[0]
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[5], [3:0]	RawStatus	These bits reflect the status of the interrupts after polarity control and optional edge detection (i.e. just before masking with the bits in the AIC_ENABLE_CTRL register): 0=Inactive 1=Pending
[31:6], [4]	Reserved	These bits should be written with a '0' and ignored on a read.

22.3.3.4.5 AIC Enabled Status Register (AIC_ENABLED_STAT)

Table 22-97. AIC_ENABLED_STAT

AIC_ENABLED_STAT (Read Only), Address Offset (0x8010)																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																										sl_err_int	RESERVED	trng_irq	pka_int[2]	pka_int[1]	pka_int[0]
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[5], [3:0]	EnableStatus	These bits reflect the status of the interrupts gated with the enable bits of the AIC_ENABLE_CTRL Register. 0=Inactive 1=Pending
[31:6], [4]	Reserved	These bits should be written with a '0' and ignored on a read.

22.3.3.4.6 AIC Acknowledge Register (AIC_ACK)

Table 22-98. AIC_ACK

AIC_ACK (Write Only), Address Offset (0x8010)																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																										sl_err_int	RESERVED	trng_irq	pka_int[2]	pka_int[1]	pka_int[0]
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[5], [3:0]	Acknowledge	Used to clear edge-detect interrupts. A '1' written to any one of the bit locations acknowledges the respective interrupt and clears the status bit. 0 =No effect. 1 = Acknowledge the interrupt signal, clears the status bit. The activated bit of Acknowledge will be cleared internally.
[31:6], [4]	Reserved	These bits should be written with a '0' and ignored on a read.

22.3.3.4.7 AIC Enable Set Register (AIC_ENABLE_SET)

Table 22-99. AIC_ENABLE_SET

AIC_ENABLE_SET (Write Only), Address Offset (0x800C)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																										sl_err_int	RESERVED	trng_irq	pka_int[2]	pka_int[1]	pka_int[0]
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[5], [3:0]	EnableSet	Individual interrupt enable bits per interrupt input 0 = No effect. 1 = Sets the corresponding bit in the AIC_ENABLE_CTRL register to '1', i.e. enables that interrupt. The activated bit of EnableSet will be cleared internally.
[31:6], [4]	Reserved	These bits should be written with a '0' and ignored on a read.

22.3.3.4.8 AIC Enable Clear Register (AIC_ENABLE_CLR)

Table 22-100. AIC_ENABLE_CLR

AIC_ENABLE_CLR (Write Only), Address Offset (0x8014)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																										sl_err_int	RESERVED	trng_irq	pka_int[2]	pka_int[1]	pka_int[0]
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Function
[5], [3:0]	EnableClear	Individual interrupt disable bits per interrupt input 0 = No effect. 1 = Resets the corresponding bit in the AIC_ENABLE_CTRL register to '0', i.e. disables that interrupt. The activated bit of EnableClear will be cleared internally.

[31:6], [4]	Reserved	These bits should be written with a '0' and ignored on a read.

22.3.3.4.9 AIC Options Register (AIC_OPTIONS)

Table 22-101. AIC_OPTIONS

AIC_OPTIONS (Read Only), Address Offset (0x8018)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																										Number of inputs					
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Bit	Name	Function
[5:0]	Number of inputs	Fixed to 0x06.
[31:6]	Reserved	These bits should be ignored on a read.

22.3.3.4.10 AIC Version Register (AIC_VERSION)

Table 22-102. AIC_VERSION

AIC_VERSION (Read Only), Address Offset (0x801C)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED				Major version number				Minor version number				Patch level				Bit-by-bit complement of EIP-number								EIP-number							
0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	1	0	1	1	0	1	1	0	0	1	0	0	1

Bit	Name	Function
[7:0]	EIP-number	These bits encode the EIP number for the AIC. This field contains the value 201 (decimal) or 0xC9.
[15:8]	Bit-by-bit complement of EIP-number	These bits simply contain the complement of bits [7:0] (0x36), used by a driver to ascertain that the AIC_VERSION register is indeed read.
[19:16]	Patch level	These bits encode the hardware patch level for this module – they start at value 0 on the first release
[23:20]	Minor version number	These bits encode the minor version number for this module – as this is version 1.1, the value will be 1 here.
[24:27]	Major version number	These bits encode the major version number for this module – as this is version 1.1, the value will be 1 here
[31:28]	Reserved	These bits should be ignored on reading.

22.4 AES and PKA Registers

22.4.1 AES Registers

22.4.1.1 AES Registers Mapping Summary

This section provides information on the AES module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 22-103. AES Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
AES_DMAC_CH0_CTRL	RW	32	0x0000 0000	0x000	0x4008 B000
AES_DMAC_CH0_EXTADDR	RW	32	0x0000 0000	0x004	0x4008 B004
AES_DMAC_CH0_DMALENGTH	RW	32	0x0000 0000	0x00C	0x4008 B00C
AES_DMAC_STATUS	RO	32	0x0000 0000	0x018	0x4008 B018
AES_DMAC_SWRES	WO	32	0x0000 0000	0x01C	0x4008 B01C
AES_DMAC_CH1_CTRL	RW	32	0x0000 0000	0x020	0x4008 B020
AES_DMAC_CH1_EXTADDR	RW	32	0x0000 0000	0x024	0x4008 B024
AES_DMAC_CH1_DMALENGTH	RW	32	0x0000 0000	0x02C	0x4008 B02C
AES_DMAC_MST_RUNPARAMS	RW	32	0x0000 2400	0x078	0x4008 B078
AES_DMAC_PERSR	RO	32	0x0000 0000	0x07C	0x4008 B07C
AES_DMAC_OPTIONS	RO	32	0x0000 0202	0x0F8	0x4008 B0F8
AES_DMAC_VERSION	RO	32	0x0101 2ED1	0x0FC	0x4008 B0FC
AES_KEY_STORE_WRITE_AREA	RW	32	0x0000 0000	0x400	0x4008 B400
AES_KEY_STORE_WRITTEN_AREA	RW	32	0x0000 0000	0x404	0x4008 B404
AES_KEY_STORE_SIZE	RW	32	0x0000 0001	0x408	0x4008 B408
AES_KEY_STORE_READ_AREA	RW	32	0x0000 0008	0x40C	0x4008 B40C
AES_AES_KEY2_0	WO	32	0x0000 0000	0x500	0x4008 B500
AES_AES_KEY2_1	WO	32	0x0000 0000	0x504	0x4008 B504
AES_AES_KEY2_2	WO	32	0x0000 0000	0x508	0x4008 B508
AES_AES_KEY2_3	WO	32	0x0000 0000	0x50C	0x4008 B50C
AES_AES_KEY3_0	WO	32	0x0000 0000	0x510	0x4008 B510
AES_AES_KEY3_1	WO	32	0x0000 0000	0x514	0x4008 B514
AES_AES_KEY3_2	WO	32	0x0000 0000	0x518	0x4008 B518

Table 22-103. AES Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
AES_AES_KEY3_3	WO	32	0x0000 0000	0x51C	0x4008 B51C
AES_AES_IV_0	RW	32	0x0000 0000	0x540	0x4008 B540
AES_AES_IV_1	RW	32	0x0000 0000	0x544	0x4008 B544
AES_AES_IV_2	RW	32	0x0000 0000	0x548	0x4008 B548
AES_AES_IV_3	RW	32	0x0000 0000	0x54C	0x4008 B54C
AES_AES_CTRL	RW	32	0x8000 0000	0x550	0x4008 B550
AES_AES_C_LENGTH_0	WO	32	0x0000 0000	0x554	0x4008 B554
AES_AES_C_LENGTH_1	WO	32	0x0000 0000	0x558	0x4008 B558
AES_AES_AUTH_LENGTH	WO	32	0x0000 0000	0x55C	0x4008 B55C
AES_AES_DATA_IN_OUT_0	WO	32	0x0000 0000	0x560	0x4008 B560
AES_AES_DATA_IN_OUT_1	WO	32	0x0000 0000	0x564	0x4008 B564
AES_AES_DATA_IN_OUT_2	WO	32	0x0000 0000	0x568	0x4008 B568
AES_AES_DATA_IN_OUT_3	WO	32	0x0000 0000	0x56C	0x4008 B56C
AES_AES_TAG_OUT_0	RO	32	0x0000 0000	0x570	0x4008 B570
AES_AES_TAG_OUT_1	RO	32	0x0000 0000	0x574	0x4008 B574
AES_AES_TAG_OUT_2	RO	32	0x0000 0000	0x578	0x4008 B578
AES_AES_TAG_OUT_3	RO	32	0x0000 0000	0x57C	0x4008 B57C
AES_HASH_DATA_IN_0	WO	32	0x0000 0000	0x600	0x4008 B600
AES_HASH_DATA_IN_1	WO	32	0x0000 0000	0x604	0x4008 B604
AES_HASH_DATA_IN_2	WO	32	0x0000 0000	0x608	0x4008 B608
AES_HASH_DATA_IN_3	WO	32	0x0000 0000	0x60C	0x4008 B60C
AES_HASH_DATA_IN_4	WO	32	0x0000 0000	0x610	0x4008 B610
AES_HASH_DATA_IN_5	WO	32	0x0000 0000	0x614	0x4008 B614
AES_HASH_DATA_IN_6	WO	32	0x0000 0000	0x618	0x4008 B618
AES_HASH_DATA_IN_7	WO	32	0x0000 0000	0x61C	0x4008 B61C
AES_HASH_DATA_IN_8	WO	32	0x0000 0000	0x620	0x4008 B620
AES_HASH_DATA_IN_9	WO	32	0x0000 0000	0x624	0x4008 B624

Table 22-103. AES Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
AES_HASH_DATA_IN_10	WO	32	0x0000 0000	0x628	0x4008 B628
AES_HASH_DATA_IN_11	WO	32	0x0000 0000	0x62C	0x4008 B62C
AES_HASH_DATA_IN_12	WO	32	0x0000 0000	0x630	0x4008 B630
AES_HASH_DATA_IN_13	WO	32	0x0000 0000	0x634	0x4008 B634
AES_HASH_DATA_IN_14	WO	32	0x0000 0000	0x638	0x4008 B638
AES_HASH_DATA_IN_15	WO	32	0x0000 0000	0x63C	0x4008 B63C
AES_HASH_IO_BUF_CTRL	RW	32	0x0000 0004	0x640	0x4008 B640
AES_HASH_MODE_IN	WO	32	0x0000 0000	0x644	0x4008 B644
AES_HASH_LENGTH_IN_L	WO	32	0x0000 0000	0x648	0x4008 B648
AES_HASH_LENGTH_IN_H	WO	32	0x0000 0000	0x64C	0x4008 B64C
AES_HASH_DIGEST_A	RW	32	0x0000 0000	0x650	0x4008 B650
AES_HASH_DIGEST_B	RW	32	0x0000 0000	0x654	0x4008 B654
AES_HASH_DIGEST_C	RW	32	0x0000 0000	0x658	0x4008 B658
AES_HASH_DIGEST_D	RW	32	0x0000 0000	0x65C	0x4008 B65C
AES_HASH_DIGEST_E	RW	32	0x0000 0000	0x660	0x4008 B660
AES_HASH_DIGEST_F	RW	32	0x0000 0000	0x664	0x4008 B664
AES_HASH_DIGEST_G	RW	32	0x0000 0000	0x668	0x4008 B668
AES_HASH_DIGEST_H	RW	32	0x0000 0000	0x66C	0x4008 B66C
AES_CTRL_ALG_SEL	RW	32	0x0000 0000	0x700	0x4008 B700
AES_CTRL_PROT_EN	RW	32	0x0000 0000	0x704	0x4008 B704
AES_CTRL_SW_RESET	RW	32	0x0000 0000	0x740	0x4008 B740
AES_CTRL_INT_CFG	RW	32	0x0000 0000	0x780	0x4008 B780
AES_CTRL_INT_EN	RW	32	0x0000 0000	0x784	0x4008 B784
AES_CTRL_INT_CLR	WO	32	0x0000 0000	0x788	0x4008 B788
AES_CTRL_INT_SET	WO	32	0x0000 0000	0x78C	0x4008 B78C

Table 22-103. AES Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
AES_CTRL_INT_STAT	RO	32	0x0000 0000	0x790	0x4008 B790
AES_CTRL_OPTIONS	RO	32	0x0101 01F7	0x7F8	0x4008 B7F8
AES_CTRL_VERSION	RO	32	0x9110 8778	0x7FC	0x4008 B7FC

22.4.1.2 AES Register Descriptions

AES_DMAC_CH0_CTRL

Address offset	0x000		
Physical Address	0x4008 B000	**Instance**	AES
Description	Channel control This register is used for channel enabling and priority selection. When a channel is disabled, it becomes inactive only when all ongoing requests are finished.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	RW	0x0000 0000
1	PRIO	Channel priority 0: Low 1: High If both channels have the same priority, access of the channels to the external port is arbitrated using the round robin scheme. If one channel has a high priority and another one low, the channel with the high priority is served first, in case of simultaneous access requests.	RW	0
0	EN	Channel enable 0: Disabled 1: Enable Note: Disabling an active channel interrupts the DMA operation. The ongoing block transfer completes, but no new transfers are requested.	RW	0

AES_DMAC_CH0_EXTADDR

Address offset	0x004		
Physical Address	0x4008 B004	**Instance**	AES
Description	Channel external address		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	ADDR	Channel external address value When read during operation, it holds the last updated external address after being sent to the master interface.	RW	0x0000 0000

AES_DMAC_CH0_DMALENGTH

Address offset	0x00C		
Physical Address	0x4008 B00C	**Instance**	AES
Description	Channel DMA length		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RW	0x0000
15:0	DMALEN	Channel DMA length in bytes During configuration, this register contains the DMA transfer length in bytes. During operation, it contains the last updated value of the DMA transfer length after being sent to the master interface. Note: Setting this register to a nonzero value starts the transfer if the channel is enabled. Therefore, this register must be written last when setting up a DMA channel.	RW	0x0000

AES_DMAC_STATUS

Address offset	0x018		
Physical Address	0x4008 B018	**Instance**	AES
Description	DMAC status This register provides the actual state of each DMA channel. It also reports port errors in case these were received by the master interface module during the data transfer.		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED														PORT_ERR	RESERVED															CH1_ACT	CH0_ACT

Bits	Field Name	Description	Type	Reset
31:18	RESERVED	This bit field is reserved.	RO	0x0000
17	PORT_ERR	Reflects possible transfer errors on the AHB port.	RO	0
16:2	RESERVED	This bit field is reserved.	RO	0x0000
1	CH1_ACT	A value of 1 indicates that channel 1 is active (DMA transfer on-going).	RO	0
0	CH0_ACT	A value of 1 indicates that channel 0 is active (DMA transfer on-going).	RO	0

AES_DMAC_SWRES

Address offset	0x01C		
Physical Address	0x4008 B01C	**Instance**	AES
Description	DMAC software reset register Software reset is used to reset the DMAC to stop all transfers and clears the port error status register. After the software reset is performed, all the channels are disabled and no new requests are performed by the channels. The DMAC waits for the existing (active) requests to finish and accordingly sets the DMAC status registers.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	WO	0x0000 0000
0	SWRES	Software reset enable 0 = Disabled 1 = Enabled (self-cleared to 0) Completion of the software reset must be checked through the DMAC_STATUS register.	WO	0

AES_DMAC_CH1_CTRL

Address offset	0x020		
Physical Address	0x4008 B020	**Instance**	AES
Description	Channel control This register is used for channel enabling and priority selection. When a channel is disabled, it becomes inactive only when all ongoing requests are finished.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	RW	0x0000 0000
1	PRIO	Channel priority 0: Low 1: High If both channels have the same priority, access of the channels to the external port is arbitrated using the round robin scheme. If one channel has a high priority and another one low, the channel with the high priority is served first, in case of simultaneous access requests.	RW	0
0	EN	Channel enable 0: Disabled 1: Enable Note: Disabling an active channel interrupts the DMA operation. The ongoing block transfer completes, but no new transfers are requested.	RW	0

AES_DMAC_CH1_EXTADDR

Address offset	0x024		
Physical Address	0x4008 B024	**Instance**	AES
Description	Channel external address		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	ADDR	Channel external address value. When read during operation, it holds the last updated external address after being sent to the master interface.	RW	0x0000 0000

AES_DMAC_CH1_DMALENGTH

Address offset	0x02C		
Physical Address	0x4008 B02C	**Instance**	AES
Description	Channel DMA length		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RW	0x0000
15:0	DMALEN	Channel DMA length in bytes. During configuration, this register contains the DMA transfer length in bytes. During operation, it contains the last updated value of the DMA transfer length after being sent to the master interface. Note: Setting this register to a nonzero value starts the transfer if the channel is enabled. Therefore, this register must be written last when setting up a DMA channel.	RW	0x0000

AES_DMAC_MST_RUNPARAMS

Address offset	0x078		
Physical Address	0x4008 B078	**Instance**	AES
Description	DMAC master run-time parameters This register defines all the run-time parameters for the AHB master interface port. These parameters are required for the proper functioning of the EIP-101m AHB master adapter.		
Type	RW		

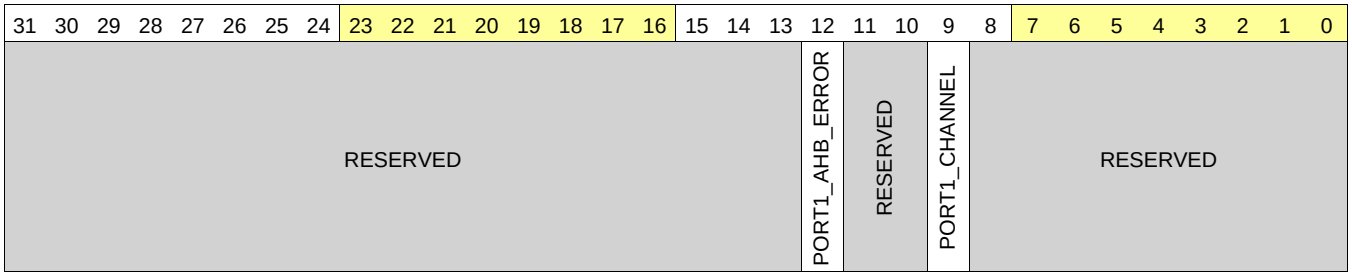
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																AHB_MST1_BURST_SIZE				AHB_MST1_IDLE_EN	AHB_MST1_INCR_EN	AHB_MST1_LOCK_EN	AHB_MST1_BIGEND	RESERVED							

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RW	0x0000
15:12	AHB_MST1_BURST_SIZE	Maximum burst size that can be performed on the AHB bus 0010b = 4 bytes (default) 0011b = 8 bytes 0100b = 16 bytes 0101b = 32 bytes 0110b = 64 bytes Others = Reserved	RW	0x2
11	AHB_MST1_IDLE_EN	Idle insertion between consecutive burst transfers on AHB 0: No Idle insertion 1: Idle insertion	RW	0
10	AHB_MST1_INCR_EN	Burst length type of AHB transfer 0: Unspecified length burst transfers 1: Fixed length burst or single transfers	RW	1
9	AHB_MST1_LOCK_EN	Locked transform on AHB 0: Transfers are not locked 1: Transfers are locked	RW	0

Bits	Field Name	Description	Type	Reset
8	AHB_MST1_BIGEND	Endianess for the AHB master 0: Little endian 1: Big endian	RW	0
7:0	RESERVED	This bit field is reserved.	RW	0x00

AES_DMAC_PERSR

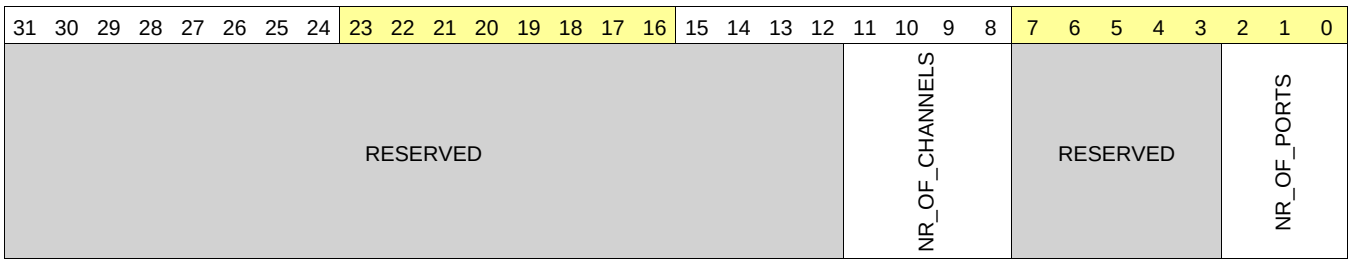
Address offset	0x07C		
Physical Address	0x4008 B07C	**Instance**	AES
Description	DMAC port error raw status register This register provides the actual status of individual port errors. It also indicates which channel is serviced by an external AHB port (which is frozen by a port error). A port error aborts operations on all serviced channels (channel enable bit is forced to 0) and prevents further transfers via that port until the error is cleared by writing to the DMAC_SWRES register.		
Type	RO		



Bits	Field Name	Description	Type	Reset
31:13	RESERVED	This bit field is reserved.	RO	0x0 0000
12	PORT1_AHB_ERROR	A value of 1 indicates that the EIP-101 has detected an AHB bus error	RO	0
11:10	RESERVED	This bit field is reserved.	RO	0x0
9	PORT1_CHANNEL	Indicates which channel has serviced last (channel 0 or channel 1) by AHB master port.	RO	0
8:0	RESERVED	This bit field is reserved.	RO	0x000

AES_DMAC_OPTIONS

Address offset	0x0F8		
Physical Address	0x4008 B0F8	**Instance**	AES
Description	DMAC options register These registers contain information regarding the different options configured in this DMAC.		
Type	RO		



Bits	Field Name	Description	Type	Reset
31:12	RESERVED	This bit field is reserved.	RO	0x0 0000
11:8	NR_OF_CHANNELS	Number of channels implemented, value in the range 1-8.	RO	0x2
7:3	RESERVED	This bit field is reserved.	RO	0x00

Bits	Field Name	Description	Type	Reset
2:0	NR_OF_PORTS	Number of ports implemented, value in range 1-4.	RO	0x2

AES_DMAC_VERSION

Address offset	0x0FC		
Physical Address	0x4008 B0FC	**Instance**	AES
Description	DMAC version register This register contains an indication (or signature) of the EIP type of this DMAC, as well as the hardware version/patch numbers.		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED				HW_MAJOR_VERSION				HW_MINOR_VERSION				HW_PATCH_LEVEL				EIP_NUMBER_COMPL								EIP_NUMBER							

Bits	Field Name	Description	Type	Reset
31:28	RESERVED	This bit field is reserved.	RO	0x0
27:24	HW_MAJOR_VERSION	Major version number	RO	0x1
23:20	HW_MINOR_VERSION	Minor version number	RO	0x0
19:16	HW_PATCH_LEVEL	Patch level Starts at 0 at first delivery of this version	RO	0x1
15:8	EIP_NUMBER_COMPL	Bit-by-bit complement of the EIP_NUMBER field bits.	RO	0x2E
7:0	EIP_NUMBER	Binary encoding of the EIP-number of this DMA controller (209)	RO	0xD1

AES_KEY_STORE_WRITE_AREA

Address offset	0x400		
Physical Address	0x4008 B400	**Instance**	AES
Description	Key store write area register This register defines where the keys should be written in the key store RAM. After writing this register, the key store module is ready to receive the keys through a DMA operation. In case the key data transfer triggered an error in the key store, the error will be available in the interrupt status register after the DMA is finished. The key store write-error is asserted when the programmed/selected area is not completely written. This error is also asserted when the DMA operation writes to ram areas that are not selected. The key store RAM is divided into 8 areas of 128 bits. 192-bit keys written in the key store RAM should start on boundaries of 256 bits. This means that writing a 192-bit key to the key store RAM must be done by writing 256 bits of data with the 64 most-significant bits set to 0. These bits are ignored by the AES engine.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RAM_AREA7	RAM_AREA6	RAM_AREA5	RAM_AREA4	RAM_AREA3	RAM_AREA2	RAM_AREA1	RAM_AREA0

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RW	0x00 0000
7	RAM_AREA7	Each RAM_AREAx represents an area of 128 bits. Select the key store RAM area(s) where the key(s) needs to be written 0: RAM_AREA7 is not selected to be written. 1: RAM_AREA7 is selected to be written. Writing to multiple RAM locations is possible only when the selected RAM areas are sequential. Keys that require more than one RAM locations (key size is 192 or 256 bits), must start at one of the following areas: RAM_AREA0, RAM_AREA2, RAM_AREA4, or RAM_AREA6.	RW	0
6	RAM_AREA6	Each RAM_AREAx represents an area of 128 bits. Select the key store RAM area(s) where the key(s) needs to be written 0: RAM_AREA6 is not selected to be written. 1: RAM_AREA6 is selected to be written. Writing to multiple RAM locations is possible only when the selected RAM areas are sequential. Keys that require more than one RAM locations (key size is 192 or 256 bits), must start at one of the following areas: RAM_AREA0, RAM_AREA2, RAM_AREA4, or RAM_AREA6.	RW	0
5	RAM_AREA5	Each RAM_AREAx represents an area of 128 bits. Select the key store RAM area(s) where the key(s) needs to be written 0: RAM_AREA5 is not selected to be written. 1: RAM_AREA5 is selected to be written. Writing to multiple RAM locations is possible only when the selected RAM areas are sequential. Keys that require more than one RAM locations (key size is 192 or 256 bits), must start at one of the following areas: RAM_AREA0, RAM_AREA2, RAM_AREA4, or RAM_AREA6.	RW	0
4	RAM_AREA4	Each RAM_AREAx represents an area of 128 bits. Select the key store RAM area(s) where the key(s) needs to be written 0: RAM_AREA4 is not selected to be written. 1: RAM_AREA4 is selected to be written. Writing to multiple RAM locations is possible only when the selected RAM areas are sequential. Keys that require more than one RAM locations (key size is 192 or 256 bits), must start at one of the following areas: RAM_AREA0, RAM_AREA2, RAM_AREA4, or RAM_AREA6.	RW	0
3	RAM_AREA3	Each RAM_AREAx represents an area of 128 bits. Select the key store RAM area(s) where the key(s) needs to be written 0: RAM_AREA3 is not selected to be written. 1: RAM_AREA3 is selected to be written. Writing to multiple RAM locations is possible only when the selected RAM areas are sequential. Keys that require more than one RAM locations (key size is 192 or 256 bits), must start at one of the following areas: RAM_AREA0, RAM_AREA2, RAM_AREA4, or RAM_AREA6.	RW	0
2	RAM_AREA2	Each RAM_AREAx represents an area of 128 bits. Select the key store RAM area(s) where the key(s) needs to be written 0: RAM_AREA2 is not selected to be written. 1: RAM_AREA2 is selected to be written. Writing to multiple RAM locations is possible only when the selected RAM areas are sequential. Keys that require more than one RAM locations (key size is 192 or 256 bits), must start at one of the following areas: RAM_AREA0, RAM_AREA2, RAM_AREA4, or RAM_AREA6.	RW	0

Bits	Field Name	Description	Type	Reset
1	RAM_AREA1	Each RAM_AREAx represents an area of 128 bits. Select the key store RAM area(s) where the key(s) needs to be written 0: RAM_AREA1 is not selected to be written. 1: RAM_AREA1 is selected to be written. Writing to multiple RAM locations is possible only when the selected RAM areas are sequential. Keys that require more than one RAM locations (key size is 192 or 256 bits), must start at one of the following areas: RAM_AREA0, RAM_AREA2, RAM_AREA4, or RAM_AREA6.	RW	0
0	RAM_AREA0	Each RAM_AREAx represents an area of 128 bits. Select the key store RAM area(s) where the key(s) needs to be written 0: RAM_AREA0 is not selected to be written. 1: RAM_AREA0 is selected to be written. Writing to multiple RAM locations is possible only when the selected RAM areas are sequential. Keys that require more than one RAM locations (key size is 192 or 256 bits), must start at one of the following areas: RAM_AREA0, RAM_AREA2, RAM_AREA4, or RAM_AREA6.	RW	0

AES_KEY_STORE_WRITTEN_AREA

Address offset	0x404		
Physical Address	0x4008 B404	**Instance**	AES
Description	Key store written area register This register shows which areas of the key store RAM contain valid written keys. When a new key needs to be written to the key store, on a location that is already occupied by a valid key, this key area must be cleared first. This can be done by writing this register before the new key is written to the key store memory. Attempting to write to a key area that already contains a valid key is not allowed and results in an error.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RAM_AREA_WRITTEN7	RAM_AREA_WRITTEN6	RAM_AREA_WRITTEN5	RAM_AREA_WRITTEN4	RAM_AREA_WRITTEN3	RAM_AREA_WRITTEN2	RAM_AREA_WRITTEN1	RAM_AREA_WRITTEN0

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RW	0x00 0000
7	RAM_AREA_WRITTEN7	Read operation: 0: This RAM area is not written with valid key information. 1: This RAM area is written with valid key information. Each individual ram_area_writtenx bit can be reset by writing 1. Note: This register is reset on a soft reset from the master control module. After a soft reset, all keys must be rewritten to the key store memory.	RW	0
6	RAM_AREA_WRITTEN6	Read operation: 0: This RAM area is not written with valid key information. 1: This RAM area is written with valid key information. Each individual ram_area_writtenx bit can be reset by writing 1. Note: This register is reset on a soft reset from the master control module. After a soft reset, all keys must be rewritten to the key store memory.	RW	0

Bits	Field Name	Description	Type	Reset
5	RAM_AREA_WRITTEN5	Read operation: 0: This RAM area is not written with valid key information. 1: This RAM area is written with valid key information. Each individual ram_area_writtenx bit can be reset by writing 1. Note: This register is reset on a soft reset from the master control module. After a soft reset, all keys must be rewritten to the key store memory.	RW	0
4	RAM_AREA_WRITTEN4	Read operation: 0: This RAM area is not written with valid key information. 1: This RAM area is written with valid key information. Each individual ram_area_writtenx bit can be reset by writing 1. Note: This register is reset on a soft reset from the master control module. After a soft reset, all keys must be rewritten to the key store memory.	RW	0
3	RAM_AREA_WRITTEN3	Read operation: 0: This RAM area is not written with valid key information. 1: This RAM area is written with valid key information. Each individual ram_area_writtenx bit can be reset by writing 1. Note: This register is reset on a soft reset from the master control module. After a soft reset, all keys must be rewritten to the key store memory.	RW	0
2	RAM_AREA_WRITTEN2	Read operation: 0: This RAM area is not written with valid key information. 1: This RAM area is written with valid key information. Each individual ram_area_writtenx bit can be reset by writing 1. Note: This register is reset on a soft reset from the master control module. After a soft reset, all keys must be rewritten to the key store memory.	RW	0
1	RAM_AREA_WRITTEN1	Read operation: 0: This RAM area is not written with valid key information. 1: This RAM area is written with valid key information. Each individual ram_area_writtenx bit can be reset by writing 1. Note: This register is reset on a soft reset from the master control module. After a soft reset, all keys must be rewritten to the key store memory.	RW	0
0	RAM_AREA_WRITTEN0	Read operation: 0: This RAM area is not written with valid key information. 1: This RAM area is written with valid key information. Each individual ram_area_writtenx bit can be reset by writing 1. Note: This register is reset on a soft reset from the master control module. After a soft reset, all keys must be rewritten to the key store memory.	RW	0

AES_KEY_STORE_SIZE

Address offset	0x408		
Physical Address	0x4008 B408	**Instance**	AES
Description	Key store size register This register defines the size of the keys that are written with DMA. This register should be configured before writing to the KEY_STORE_WRITE_AREA register.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	RW	0x0000 0000
1:0	KEY_SIZE	Key size: 00: Reserved 01: 128 bits 10: 192 bits 11: 256 bits When writing this to this register, the KEY_STORE_WRITTEN_AREA register is reset.	RW	0x1

AES_KEY_STORE_READ_AREA

Address offset	0x40C		
Physical Address	0x4008 B40C	**Instance**	AES
Description	Key store read area register This register selects the key store RAM area from where the key needs to be read that will be used for an AES operation. The operation directly starts after writing this register. When the operation is finished, the status of the key store read operation is available in the interrupt status register. Key store read error is asserted when a RAM area is selected which does not contain valid written key.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31	BUSY	Key store operation busy status flag (read only): 0: Operation is complete. 1: Operation is not completed and the key store is busy.	RO	0
30:4	RESERVED	This bit field is reserved.	RW	0x000 0000
3:0	RAM_AREA	Selects the area of the key store RAM from where the key needs to be read that will be writen to the AES engine RAM_AREA: 0000: RAM_AREA0 0001: RAM_AREA1 0010: RAM_AREA2 0011: RAM_AREA3 0100: RAM_AREA4 0101: RAM_AREA5 0110: RAM_AREA6 0111: RAM_AREA7 1000: no RAM area selected 1001-1111: Reserved RAM areas RAM_AREA0, RAM_AREA2, RAM_AREA4 and RAM_AREA6 are the only valid read areas for 192 and 256 bits key sizes. Only RAM areas that contain valid written keys can be selected.	RW	0x8

AES_AES_KEY2_0

Address offset	0x500		
Physical Address	0x4008 B500	**Instance**	AES
Description	AES_KEY2_0 / AES_GHASH_H_IN_0 Second Key / GHASH Key (internal, but clearable) The following registers are not accessible through the host for reading and writing. They are used to store internally calculated key information and intermediate results. However, when the host performs a write to the any of the respective AES_KEY2_n or AES_KEY3_n addresses, respectively the whole 128-bit AES_KEY2_n or AES_KEY3_n register is cleared to 0s. The AES_GHASH_H_IN_n registers (required for GHASH, which is part of GCM) are mapped to the AES_KEY2_n registers. The (intermediate) authentication result for GCM and CCM is stored in the AES_KEY3_n register.		
Type	WO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AES_KEY2																															

Bits	Field Name	Description	Type	Reset
31:0	AES_KEY2	AES_KEY2/AES_GHASH_H[31:0] For GCM: -[127:0] - GHASH_H - The internally calculated GHASH key is stored in these registers. Only used for modes that use the GHASH function (GCM). -[255:128] - This register is used to store intermediate values and is initialized with 0s when loading a new key. For CCM: -[255:0] - This register is used to store intermediate values. For CBC-MAC: -[255:0] - ZEROES - This register must remain 0.	WO	0x0000 0000

AES_AES_KEY2_1

Address offset	0x504		
Physical Address	0x4008 B504	**Instance**	AES
Description	AES_KEY2_1 / AES_GHASH_H_IN_1 Second Key / GHASH Key (internal, but clearable) The following registers are not accessible through the host for reading and writing. They are used to store internally calculated key information and intermediate results. However, when the host performs a write to the any of the respective AES_KEY2_n or AES_KEY3_n addresses, respectively the whole 128-bit AES_KEY2_n or AES_KEY3_n register is cleared to 0s. The AES_GHASH_H_IN_n registers (required for GHASH, which is part of GCM) are mapped to the AES_KEY2_n registers. The (intermediate) authentication result for GCM and CCM is stored in the AES_KEY3_n register.		
Type	WO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AES_KEY2																															

Bits	Field Name	Description	Type	Reset
31:0	AES_KEY2	AES_KEY2/AES_GHASH_H[63:32] For GCM: -[127:0] - GHASH_H - The internally calculated GHASH key is stored in these registers. Only used for modes that use the GHASH function (GCM). -[255:128] - This register is used to store intermediate values and is initialized with 0s when loading a new key. For CCM: -[255:0] - This register is used to store intermediate values. For CBC-MAC: -[255:0] - ZEROES - This register must remain 0.	WO	0x0000 0000

AES_AES_KEY2_2

Address offset	0x508		
Physical Address	0x4008 B508	**Instance**	AES
Description	AES_KEY2_2 / AES_GHASH_H_IN_2 Second Key / GHASH Key (internal, but clearable) The following registers are not accessible through the host for reading and writing. They are used to store internally calculated key information and intermediate results. However, when the host performs a write to the any of the respective AES_KEY2_n or AES_KEY3_n addresses, respectively the whole 128-bit AES_KEY2_n or AES_KEY3_n register is cleared to 0s. The AES_GHASH_H_IN_n registers (required for GHASH, which is part of GCM) are mapped to the AES_KEY2_n registers. The (intermediate) authentication result for GCM and CCM is stored in the AES_KEY3_n register.		
Type	WO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AES_KEY2																															

Bits	Field Name	Description	Type	Reset
31:0	AES_KEY2	AES_KEY2/AES_GHASH_H[95:64] For GCM: -[127:0] - GHASH_H - The internally calculated GHASH key is stored in these registers. Only used for modes that use the GHASH function (GCM). -[255:128] - This register is used to store intermediate values and is initialized with 0s when loading a new key. For CCM: -[255:0] - This register is used to store intermediate values. For CBC-MAC: -[255:0] - ZEROES - This register must remain 0.	WO	0x0000 0000

AES_AES_KEY2_3

Address offset	0x50C		
Physical Address	0x4008 B50C	**Instance**	AES
Description	AES_KEY2_3 / AES_GHASH_H_IN_3 Second Key / GHASH Key (internal, but clearable) The following registers are not accessible through the host for reading and writing. They are used to store internally calculated key information and intermediate results. However, when the host performs a write to the any of the respective AES_KEY2_n or AES_KEY3_n addresses, respectively the whole 128-bit AES_KEY2_n or AES_KEY3_n register is cleared to 0s. The AES_GHASH_H_IN_n registers (required for GHASH, which is part of GCM) are mapped to the AES_KEY2_n registers. The (intermediate) authentication result for GCM and CCM is stored in the AES_KEY3_n register.		
Type	WO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AES_KEY2																															

Bits	Field Name	Description	Type	Reset
31:0	AES_KEY2	AES_KEY2/AES_GHASH_H[127:96] For GCM: -[127:0] - GHASH_H - The internally calculated GHASH key is stored in these registers. Only used for modes that use the GHASH function (GCM). -[255:128] - This register is used to store intermediate values and is initialized with 0s when loading a new key. For CCM: -[255:0] - This register is used to store intermediate values. For CBC-MAC: -[255:0] - ZEROES - This register must remain 0.	WO	0x0000 0000

AES_AES_KEY3_0

Address offset	0x510		
Physical Address	0x4008 B510	**Instance**	AES
Description	AES_KEY3_0 / AES_KEY2_4 Third Key / Second Key (internal, but clearable) The following registers are not accessible through the host for reading and writing. They are used to store internally calculated key information and intermediate results. However, when the host performs a write to the any of the respective AES_KEY2_n or AES_KEY3_n addresses, respectively the whole 128-bit AES_KEY2_n or AES_KEY3_n register is cleared to 0s. The AES_GHASH_H_IN_n registers (required for GHASH, which is part of GCM) are mapped to the AES_KEY2_n registers. The (intermediate) authentication result for GCM and CCM is stored in the AES_KEY3_n register.		
Type	WO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AES_KEY3																															

Bits	Field Name	Description	Type	Reset
31:0	AES_KEY3	AES_KEY3[31:0]/AES_KEY2[159:128] For GCM: -[127:0] - GHASH_H - The internally calculated GHASH key is stored in these registers. Only used for modes that use the GHASH function (GCM). -[255:128] - This register is used to store intermediate values and is initialized with 0s when loading a new key. For CCM: -[255:0] - This register is used to store intermediate values. For CBC-MAC: -[255:0] - ZEROES - This register must remain 0.	WO	0x0000 0000

AES_AES_KEY3_1

Address offset	0x514		
Physical Address	0x4008 B514	**Instance**	AES
Description	AES_KEY3_1 / AES_KEY2_5 Third Key / Second Key (internal, but clearable) The following registers are not accessible through the host for reading and writing. They are used to store internally calculated key information and intermediate results. However, when the host performs a write to the any of the respective AES_KEY2_n or AES_KEY3_n addresses, respectively the whole 128-bit AES_KEY2_n or AES_KEY3_n register is cleared to 0s. The AES_GHASH_H_IN_n registers (required for GHASH, which is part of GCM) are mapped to the AES_KEY2_n registers. The (intermediate) authentication result for GCM and CCM is stored in the AES_KEY3_n register.		
Type	WO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AES_KEY3																															

Bits	Field Name	Description	Type	Reset
31:0	AES_KEY3	AES_KEY3[63:32]/AES_KEY2[191:160] For GCM: -[127:0] - GHASH_H - The internally calculated GHASH key is stored in these registers. Only used for modes that use the GHASH function (GCM). -[255:128] - This register is used to store intermediate values and is initialized with 0s when loading a new key. For CCM: -[255:0] - This register is used to store intermediate values. For CBC-MAC: -[255:0] - ZEROES - This register must remain 0.	WO	0x0000 0000

AES_AES_KEY3_2

Address offset	0x518		
Physical Address	0x4008 B518	**Instance**	AES
Description	AES_KEY3_2 / AES_KEY2_6 Third Key / Second Key (internal, but clearable) The following registers are not accessible through the host for reading and writing. They are used to store internally calculated key information and intermediate results. However, when the host performs a write to the any of the respective AES_KEY2_n or AES_KEY3_n addresses, respectively the whole 128-bit AES_KEY2_n or AES_KEY3_n register is cleared to 0s. The AES_GHASH_H_IN_n registers (required for GHASH, which is part of GCM) are mapped to the AES_KEY2_n registers. The (intermediate) authentication result for GCM and CCM is stored in the AES_KEY3_n register.		
Type	WO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AES_KEY3																															

Bits	Field Name	Description	Type	Reset
31:0	AES_KEY3	AES_KEY3[95:64]/AES_KEY2[223:192] For GCM: -[127:0] - GHASH_H - The internally calculated GHASH key is stored in these registers. Only used for modes that use the GHASH function (GCM). -[255:128] - This register is used to store intermediate values and is initialized with 0s when loading a new key. For CCM: -[255:0] - This register is used to store intermediate values. For CBC-MAC: -[255:0] - ZEROES - This register must remain 0.	WO	0x0000 0000

AES_AES_KEY3_3

Address offset	0x51C		
Physical Address	0x4008 B51C	**Instance**	AES
Description	AES_KEY3_3 / AES_KEY2_7 Third Key / Second Key (internal, but clearable) The following registers are not accessible through the host for reading and writing. They are used to store internally calculated key information and intermediate results. However, when the host performs a write to the any of the respective AES_KEY2_n or AES_KEY3_n addresses, respectively the whole 128-bit AES_KEY2_n or AES_KEY3_n register is cleared to 0s. The AES_GHASH_H_IN_n registers (required for GHASH, which is part of GCM) are mapped to the AES_KEY2_n registers. The (intermediate) authentication result for GCM and CCM is stored in the AES_KEY3_n register.		
Type	WO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AES_KEY3																															

Bits	Field Name	Description	Type	Reset
31:0	AES_KEY3	AES_KEY3[127:96]/AES_KEY2[255:224] For GCM: -[127:0] - GHASH_H - The internally calculated GHASH key is stored in these registers. Only used for modes that use the GHASH function (GCM). -[255:128] - This register is used to store intermediate values and is initialized with 0s when loading a new key. For CCM: -[255:0] - This register is used to store intermediate values. For CBC-MAC: -[255:0] - ZEROES - This register must remain 0.	WO	0x0000 0000

AES_AES_IV_0

Address offset	0x540		
Physical Address	0x4008 B540	**Instance**	AES
Description	AES initialization vector registers These registers are used to provide and read the IV from the AES engine.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	AES_IV	AES_IV[31:0] Initialization vector Used for regular non-ECB modes (CBC/CTR): -[127:0] - AES_IV - For regular AES operations (CBC and CTR) these registers must be written with a new 128-bit IV. After an operation, these registers contain the latest 128-bit result IV, generated by the EIP-120t. If CTR mode is selected, this value is incremented with 0x1: After first use - When a new data block is submitted to the engine For GCM: -[127:0] - AES_IV - For GCM operations, these registers must be written with a new 128-bit IV. After an operation, these registers contain the updated 128-bit result IV, generated by the EIP-120t. Note that bits [127:96] of the IV represent the initial counter value (which is 1 for GCM) and must therefore be initialized to 0x01000000. This value is incremented with 0x1: After first use - When a new data block is submitted to the engine. For CCM: -[127:0] - A0: For CCM this field must be written with value A0, this value is the concatenation of: A0-flags (5-bits of 0 and 3-bits 'L'), Nonce and counter value. 'L' must be a copy from the 'L' value of the AES_CTRL register. This 'L' indicates the width of the Nonce and counter. The loaded counter must be initialized to 0. The total width of A0 is 128-bit. For CBC-MAC: -[127:0] - Zeroes - For CBC-MAC this register must be written with 0s at the start of each operation. After an operation, these registers contain the 128-bit TAG output, generated by the EIP-120t.	RW	0x0000 0000

AES_AES_IV_1

Address offset	0x544		
Physical Address	0x4008 B544	**Instance**	AES
Description	AES initialization vector registers These registers are used to provide and read the IV from the AES engine.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	AES_IV	AES_IV[63:32] Initialization vector Used for regular non-ECB modes (CBC/CTR): -[127:0] - AES_IV - For regular AES operations (CBC and CTR) these registers must be written with a new 128-bit IV. After an operation, these registers contain the latest 128-bit result IV, generated by the EIP-120t. If CTR mode is selected, this value is incremented with 0x1: After first use - When a new data block is submitted to the engine For GCM: -[127:0] - AES_IV - For GCM operations, these registers must be written with a new 128-bit IV. After an operation, these registers contain the updated 128-bit result IV, generated by the EIP-120t. Note that bits [127:96] of the IV represent the initial counter value (which is 1 for GCM) and must therefore be initialized to 0x01000000. This value is incremented with 0x1: After first use - When a new data block is submitted to the engine. For CCM: -[127:0] - A0: For CCM this field must be written with value A0, this value is the concatenation of: A0-flags (5-bits of 0 and 3-bits 'L'), Nonce and counter value. 'L' must be a copy from the 'L' value of the AES_CTRL register. This 'L' indicates the width of the Nonce and counter. The loaded counter must be initialized to 0. The total width of A0 is 128-bit. For CBC-MAC: -[127:0] - Zeroes - For CBC-MAC this register must be written with 0s at the start of each operation. After an operation, these registers contain the 128-bit TAG output, generated by the EIP-120t.	RW	0x0000 0000

AES_AES_IV_2

Address offset	0x548		
Physical Address	0x4008 B548	**Instance**	AES
Description	AES initialization vector registers These registers are used to provide and read the IV from the AES engine.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	AES_IV	AES_IV[95:64] Initialization vector Used for regular non-ECB modes (CBC/CTR): -[127:0] - AES_IV - For regular AES operations (CBC and CTR) these registers must be written with a new 128-bit IV. After an operation, these registers contain the latest 128-bit result IV, generated by the EIP-120t. If CTR mode is selected, this value is incremented with 0x1: After first use - When a new data block is submitted to the engine For GCM: -[127:0] - AES_IV - For GCM operations, these registers must be written with a new 128-bit IV. After an operation, these registers contain the updated 128-bit result IV, generated by the EIP-120t. Note that bits [127:96] of the IV represent the initial counter value (which is 1 for GCM) and must therefore be initialized to 0x01000000. This value is incremented with 0x1: After first use - When a new data block is submitted to the engine. For CCM: -[127:0] - A0: For CCM this field must be written with value A0, this value is the concatenation of: A0-flags (5-bits of 0 and 3-bits 'L'), Nonce and counter value. 'L' must be a copy from the 'L' value of the AES_CTRL register. This 'L' indicates the width of the Nonce and counter. The loaded counter must be initialized to 0. The total width of A0 is 128-bit. For CBC-MAC: -[127:0] - Zeroes - For CBC-MAC this register must be written with 0s at the start of each operation. After an operation, these registers contain the 128-bit TAG output, generated by the EIP-120t.	RW	0x0000 0000

AES_AES_IV_3

Address offset	0x54C		
Physical Address	0x4008 B54C	**Instance**	AES
Description	AES initialization vector registers These registers are used to provide and read the IV from the AES engine.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	AES_IV	AES_IV[127:96] Initialization vector Used for regular non-ECB modes (CBC/CTR): -[127:0] - AES_IV - For regular AES operations (CBC and CTR) these registers must be written with a new 128-bit IV. After an operation, these registers contain the latest 128-bit result IV, generated by the EIP-120t. If CTR mode is selected, this value is incremented with 0x1: After first use - When a new data block is submitted to the engine For GCM: -[127:0] - AES_IV - For GCM operations, these registers must be written with a new 128-bit IV. After an operation, these registers contain the updated 128-bit result IV, generated by the EIP-120t. Note that bits [127:96] of the IV represent the initial counter value (which is 1 for GCM) and must therefore be initialized to 0x01000000. This value is incremented with 0x1: After first use - When a new data block is submitted to the engine. For CCM: -[127:0] - A0: For CCM this field must be written with value A0, this value is the concatenation of: A0-flags (5-bits of 0 and 3-bits 'L'), Nonce and counter value. 'L' must be a copy from the 'L' value of the AES_CTRL register. This 'L' indicates the width of the Nonce and counter. The loaded counter must be initialized to 0. The total width of A0 is 128-bit. For CBC-MAC: -[127:0] - Zeroes - For CBC-MAC this register must be written with 0s at the start of each operation. After an operation, these registers contain the 128-bit TAG output, generated by the EIP-120t.	RW	0x0000 0000

AES_AES_CTRL

Address offset	0x550		
Physical Address	0x4008 B550	**Instance**	AES
Description	AES input/output buffer control and mode register This register specifies the AES mode of operation for the EIP-120t. Electronic codebook (ECB) mode is automatically selected if bits [28:5] of this register are all 0.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CONTEXT_READY	SAVED_CONTEXT_READY	SAVE_CONTEXT	RESERVED					CCM_M			CCM_L		CCM	GCM		CBC_MAC	RESERVED							CTR_WIDTH		CTR	CBC	KEY_SIZE		DIRECTION	INPUT_READY	OUTPUT_READY

Bits	Field Name	Description	Type	Reset
31	CONTEXT_READY	If 1, this read-only status bit indicates that the context data registers can be overwritten and the host is permitted to write the next context.	RO	1
30	SAVED_CONTEXT_READY	If 1, this status bit indicates that an AES authentication TAG and/or IV block(s) is/are available for the host to retrieve. This bit is only asserted if the save_context bit is set to 1. The bit is mutual exclusive with the context_ready bit. Writing one clears the bit to 0, indicating the AES core can start its next operation. This bit is also cleared when the 4th word of the output TAG and/or IV is read. Note: All other mode bit writes are ignored when this mode bit is written with 1. Note: This bit is controlled automatically by the EIP-120t for TAG read DMA operations.	RW	0
29	SAVE_CONTEXT	This bit indicates that an authentication TAG or result IV needs to be stored as a result context. Typically this bit must be set for authentication modes returning a TAG (CBC-MAC, GCM and CCM), or for basic encryption modes that require future continuation with the current result IV. If this bit is set, the engine retains its full context until the TAG and/or IV registers are read. The TAG or IV must be read before the AES engine can start a new operation.	RW	0
28:25	RESERVED	This bit field is reserved.	RW	0x0
24:22	CCM_M	Defines M, which indicates the length of the authentication field for CCM operations; the authentication field length equals two times (the value of CCM-M plus one). Note: The EIP-120t always returns a 128-bit authentication field, of which the M least significant bytes are valid. All values are supported.	RW	0x0
21:19	CCM_L	Defines L, which indicates the width of the length field for CCM operations; the length field in bytes equals the value of CMM-L plus one. All values are supported.	RW	0x0
18	CCM	If set to 1, AES-CCM is selected AES-CCM is a combined mode, using AES for authentication and encryption. Note: Selecting AES-CCM mode requires writing of the AAD length register after all other registers. Note: The CTR mode bit in this register must also be set to 1 to enable AES-CTR; selecting other AES modes than CTR mode is invalid.	RW	0
17:16	GCM	Set these bits to 11 to select AES-GCM mode. AES-GCM is a combined mode, using the Galois field multiplier GF(2 to the power of 128) for authentication and AES-CTR mode for encryption. Note: The CTR mode bit in this register must also be set to 1 to enable AES-CTR Bit combination description: 00 = No GCM mode 01 = Reserved, do not select 10 = Reserved, do not select 11 = Autonomous GHASH (both H- and Y0-encrypted calculated internally) Note: The EIP-120t-1 configuration only supports mode 11 (autonomous GHASH), other GCM modes are not allowed.	RW	0x0
15	CBC_MAC	Set to 1 to select AES-CBC MAC mode. The direction bit must be set to 1 for this mode. Selecting this mode requires writing the length register after all other registers.	RW	0
14:9	RESERVED	This bit field is reserved.	RW	0x00
8:7	CTR_WIDTH	Specifies the counter width for AES-CTR mode 00 = 32-bit counter 01 = 64-bit counter 10 = 96-bit counter 11 = 128-bit counter	RW	0x0

Bits	Field Name	Description	Type	Reset
6	CTR	If set to 1, AES counter mode (CTR) is selected. Note: This bit must also be set for GCM and CCM, when encryption/decryption is required.	RW	0
5	CBC	If set to 1, cipher-block-chaining (CBC) mode is selected.	RW	0
4:3	KEY_SIZE	This read-only field specifies the key size. The key size is automatically configured when a new key is loaded through the key store module. 00 = N/A - Reserved 01 = 128-bit 10 = 192-bit 11 = 256-bit	RO	0x0
2	DIRECTION	If set to 1 an encrypt operation is performed. If set to 0 a decrypt operation is performed. This bit must be written with a 1 when CBC-MAC is selected.	RW	0
1	INPUT_READY	If 1, this status bit indicates that the 16-byte AES input buffer is empty. The host is permitted to write the next block of data. Writing 0 clears the bit to 0 and indicates that the AES core can use the provided input data block. Writing 1 to this bit is ignored. Note: For DMA operations, this bit is automatically controlled by the EIP-120t. After reset, this bit is 0. After writing a context, this bit becomes 1.	RW	0
0	OUTPUT_READY	If 1, this status bit indicates that an AES output block is available to be retrieved by the host. Writing 0 clears the bit to 0 and indicates that output data is read by the host. The AES core can provide a next output data block. Writing 1 to this bit is ignored. Note: For DMA operations, this bit is automatically controlled by the EIP-120t.	RW	0

AES_AES_C_LENGTH_0

Address offset	0x554		
Physical Address	0x4008 B554	**Instance**	AES
Description	AES crypto length registers (LSW) These registers are used to write the Length values to the EIP-120t. While processing, the length values decrement to 0. If both lengths are 0, the data stream is finished and a new context is requested. For basic AES modes (ECB, CBC, and CTR), a crypto length of 0 can be written if multiple streams need to be processed with the same key. Writing 0 length results in continued data requests until a new context is written. For the other modes (CBC-MAC, GCM, and CCM) no (new) data requests are done if the length decrements to or equals 0. It is advised to write a new length per packet. If the length registers decrement to 0, no new data is processed until a new context or length value is written. When writing a new mode without writing the length registers, the length register values from the previous context is reused.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	C_LENGTH	C_LENGTH[31:0] Bits [60:0] of the crypto length registers (LSW and MSW) store the cryptographic data length in bytes for all modes. Once processing with this context is started, this length decrements to 0. Data lengths up to (261: 1) bytes are allowed. For GCM, any value up to 236 - 32 bytes can be used. This is because a 32-bit counter mode is used; the maximum number of 128-bit blocks is 232 - 2, resulting in a maximum number of bytes of 236 - 32. A write to this register triggers the engine to start using this context. This is valid for all modes except GCM and CCM. Note: For the combined modes (GCM and CCM), this length does not include the authentication only data; the authentication length is specified in the AES_AUTH_LENGTH register below. All modes must have a length greater than 0. For the combined modes, it is allowed to have one of the lengths equal to 0. For the basic encryption modes (ECB, CBC, and CTR) it is allowed to program zero to the length field; in that case the length is assumed infinite. All data must be byte (8-bit) aligned for stream cipher modes; bit aligned data streams are not supported by the EIP-120t. For block cipher modes, the data length must be programmed in multiples of the block cipher size, 16 bytes. For a host read operation, these registers return all-0s.	WO	0x0000 0000

AES_AES_C_LENGTH_1

Address offset	0x558		
Physical Address	0x4008 B558	**Instance**	AES
Description	AES crypto length registers (MSW) These registers are used to write the Length values to the EIP-120t. While processing, the length values decrement to 0. If both lengths are 0, the data stream is finished and a new context is requested. For basic AES modes (ECB, CBC, and CTR), a crypto length of 0 can be written if multiple streams need to be processed with the same key. Writing 0 length results in continued data requests until a new context is written. For the other modes (CBC-MAC, GCM and CCM) no (new) data requests are done if the length decrements to or equals 0. It is advised to write a new length per packet. If the length registers decrement to 0, no new data is processed until a new context or length value is written. When writing a new mode without writing the length registers, the length register values from the previous context is reused.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:29	RESERVED	This bit field is reserved.	WO	0x0
28:0	C_LENGTH	C_LENGTH[60:32] Bits [60:0] of the crypto length registers (LSW and MSW) store the cryptographic data length in bytes for all modes. Once processing with this context is started, this length decrements to 0. Data lengths up to (261: 1) bytes are allowed. For GCM, any value up to 236 - 32 bytes can be used. This is because a 32-bit counter mode is used; the maximum number of 128-bit blocks is 232 - 2, resulting in a maximum number of bytes of 236 - 32. A write to this register triggers the engine to start using this context. This is valid for all modes except GCM and CCM. Note: For the combined modes (GCM and CCM), this length does not include the authentication only data; the authentication length is specified in the AES_AUTH_LENGTH register below. All modes must have a length greater than 0. For the combined modes, it is allowed to have one of the lengths equal to 0. For the basic encryption modes (ECB, CBC, and CTR) it is allowed to program zero to the length field; in that case the length is assumed infinite. All data must be byte (8-bit) aligned for stream cipher modes; bit aligned data streams are not supported by the EIP-120t. For block cipher modes, the data length must be programmed in multiples of the block cipher size, 16 bytes. For a host read operation, these registers return all-0s.	WO	0x0000 0000

AES_AES_AUTH_LENGTH

Address offset	0x55C		
Physical Address	0x4008 B55C	**Instance**	AES
Description	Authentication length register		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	AUTH_LENGTH	Bits [31:0] of the authentication length register store the authentication data length in bytes for combined modes only (GCM or CCM). Supported AAD-lengths for CCM are from 0 to (2^16 - 2^8) bytes. For GCM any value up to (2^32 - 1) bytes can be used. Once processing with this context is started, this length decrements to 0. A write to this register triggers the engine to start using this context for GCM and CCM. For a host read operation, these registers return all-0s.	WO	0x0000 0000

AES_AES_DATA_IN_OUT_0

Address offset	0x560		
Physical Address	0x4008 B560	**Instance**	AES
Description	Data input/output registers The data registers are typically accessed through the DMA and not with host writes and/or reads. However, for debugging purposes the data input/output registers can be accessed via host write and read operations. The registers are used to buffer the input/output data blocks to/from the EIP-120t. Note: The data input buffer (AES_DATA_IN_n) and data output buffer (AES_DATA_OUT_n) are mapped to the same address locations. Writes (both DMA and host) to these addresses load the Input Buffer while reads pull from the Output Buffer. Therefore, for write access, the data input buffer is written; for read access, the data output buffer is read. The data input buffer must be written before starting an operation. The data output buffer contains valid data on completion of an operation. Therefore, any 128-bit data block can be split over multiple 32-bit word transfers; these can be mixed with other host transfers over the external interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	AES_DATA_IN_OUT	AES input data[31:0] / AES output data[31:0] Data registers for input/output block data to/from the EIP-120t. For normal operations, this register is not used, since data input and output is transferred from and to the AES core via DMA. For a host write operation, these registers must be written with the 128-bit input block for the next AES operation. Writing at a word-aligned offset within this address range stores the word (4 bytes) of data into the corresponding position of 4-word deep (16 bytes = 128-bit AES block) data input buffer. This buffer is used for the next AES operation. If the last data block is not completely filled with valid data (see notes below), it is allowed to write only the words with valid data. Next AES operation is triggered by writing to the input_ready flag of the AES_CTRL register. For a host read operation, these registers contain the 128-bit output block from the latest AES operation. Reading from a word-aligned offset within this address range reads one word (4 bytes) of data out the 4-word deep (16 bytes = 128-bits AES block) data output buffer. The words (4 words, one full block) should be read before the core will move the next block to the data output buffer. To empty the data output buffer, the output_ready flag of the AES_CTRL register must be written. For the modes with authentication (CBC-MAC, GCM and CCM), the invalid (message) bytes/words can be written with any data. Note: AES typically operates on 128 bits block multiple input data. The CTR, GCM and CCM modes form an exception. The last block of a CTR-mode message may contain less than 128 bits (refer to [NIST 800-38A]). For GCM/CCM, the last block of both AAD and message data may contain less than 128 bits (refer to [NIST 800-38D]). The EIP-120t automatically pads or masks misaligned ending data blocks with 0s for GCM, CCM and CBC-MAC. For CTR mode, the remaining data in an unaligned data block is ignored. Note: The AAD / authentication only data is not copied to the output buffer but only used for authentication.	WO	0x0000 0000

AES_AES_DATA_IN_OUT_1

Address offset	0x564		
Physical Address	0x4008 B564	**Instance**	AES
Description	Data Input/Output Registers The data registers are typically accessed via DMA and not with host writes and/or reads. However, for debugging purposes the Data Input/Output Registers can be accessed via host write and read operations. The registers are used to buffer the input/output data blocks to/from the EIP-120t. Note: The data input buffer (AES_DATA_IN_n) and data output buffer (AES_DATA_OUT_n) are mapped to the same address locations. Writes (both DMA and host) to these addresses load the Input Buffer while reads pull from the Output Buffer. Therefore, for write access, the data input buffer is written; for read access, the data output buffer is read. The data input buffer must be written before starting an operation. The data output buffer contains valid data on completion of an operation. Therefore, any 128-bit data block can be split over multiple 32-bit word transfers; these can be mixed with other host transfers over the external interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	AES_DATA_IN_OUT	AES input data[63:32] / AES output data[63:32] Data registers for input/output block data to/from the EIP-120t. For normal operations, this register is not used, since data input and output is transferred from and to the AES core via DMA. For a host write operation, these registers must be written with the 128-bit input block for the next AES operation. Writing at a word-aligned offset within this address range stores the word (4 bytes) of data into the corresponding position of 4-word deep (16 bytes = 128-bit AES block) data input buffer. This buffer is used for the next AES operation. If the last data block is not completely filled with valid data (see notes below), it is allowed to write only the words with valid data. Next AES operation is triggered by writing to the input_ready flag of the AES_CTRL register. For a host read operation, these registers contain the 128-bit output block from the latest AES operation. Reading from a word-aligned offset within this address range reads one word (4 bytes) of data out the 4-word deep (16 bytes = 128-bits AES block) data output buffer. The words (4 words, one full block) should be read before the core will move the next block to the data output buffer. To empty the data output buffer, the output_ready flag of the AES_CTRL register must be written. For the modes with authentication (CBC-MAC, GCM and CCM), the invalid (message) bytes/words can be written with any data. Note: AES typically operates on 128 bits block multiple input data. The CTR, GCM and CCM modes form an exception. The last block of a CTR-mode message may contain less than 128 bits (refer to [NIST 800-38A]). For GCM/CCM, the last block of both AAD and message data may contain less than 128 bits (refer to [NIST 800-38D]). The EIP-120t automatically pads or masks misaligned ending data blocks with 0s for GCM, CCM and CBC-MAC. For CTR mode, the remaining data in an unaligned data block is ignored. Note: The AAD / authentication only data is not copied to the output buffer but only used for authentication.	WO	0x0000 0000

AES_AES_DATA_IN_OUT_2

Address offset	0x568		
Physical Address	0x4008 B568	**Instance**	AES
Description	Data Input/Output Registers The data registers are typically accessed via DMA and not with host writes and/or reads. However, for debugging purposes the Data Input/Output Registers can be accessed via host write and read operations. The registers are used to buffer the input/output data blocks to/from the EIP-120t. Note: The data input buffer (AES_DATA_IN_n) and data output buffer (AES_DATA_OUT_n) are mapped to the same address locations. Writes (both DMA and host) to these addresses load the Input Buffer while reads pull from the Output Buffer. Therefore, for write access, the data input buffer is written; for read access, the data output buffer is read. The data input buffer must be written before starting an operation. The data output buffer contains valid data on completion of an operation. Therefore, any 128-bit data block can be split over multiple 32-bit word transfers; these can be mixed with other host transfers over the external interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	AES_DATA_IN_OUT	AES input data[95:64] / AES output data[95:64] Data registers for input/output block data to/from the EIP-120t. For normal operations, this register is not used, since data input and output is transferred from and to the AES core via DMA. For a host write operation, these registers must be written with the 128-bit input block for the next AES operation. Writing at a word-aligned offset within this address range stores the word (4 bytes) of data into the corresponding position of 4-word deep (16 bytes = 128-bit AES block) data input buffer. This buffer is used for the next AES operation. If the last data block is not completely filled with valid data (see notes below), it is allowed to write only the words with valid data. Next AES operation is triggered by writing to the input_ready flag of the AES_CTRL register. For a host read operation, these registers contain the 128-bit output block from the latest AES operation. Reading from a word-aligned offset within this address range reads one word (4 bytes) of data out the 4-word deep (16 bytes = 128-bits AES block) data output buffer. The words (4 words, one full block) should be read before the core will move the next block to the data output buffer. To empty the data output buffer, the output_ready flag of the AES_CTRL register must be written. For the modes with authentication (CBC-MAC, GCM and CCM), the invalid (message) bytes/words can be written with any data. Note: AES typically operates on 128 bits block multiple input data. The CTR, GCM and CCM modes form an exception. The last block of a CTR-mode message may contain less than 128 bits (refer to [NIST 800-38A]). For GCM/CCM, the last block of both AAD and message data may contain less than 128 bits (refer to [NIST 800-38D]). The EIP-120t automatically pads or masks misaligned ending data blocks with 0s for GCM, CCM and CBC-MAC. For CTR mode, the remaining data in an unaligned data block is ignored. Note: The AAD / authentication only data is not copied to the output buffer but only used for authentication.	WO	0x0000 0000

AES_AES_DATA_IN_OUT_3

Address offset	0x56C		
Physical Address	0x4008 B56C	**Instance**	AES
Description	Data Input/Output Registers The data registers are typically accessed via DMA and not with host writes and/or reads. However, for debugging purposes the Data Input/Output Registers can be accessed via host write and read operations. The registers are used to buffer the input/output data blocks to/from the EIP-120t. Note: The data input buffer (AES_DATA_IN_n) and data output buffer (AES_DATA_OUT_n) are mapped to the same address locations. Writes (both DMA and host) to these addresses load the Input Buffer while reads pull from the Output Buffer. Therefore, for write access, the data input buffer is written; for read access, the data output buffer is read. The data input buffer must be written before starting an operation. The data output buffer contains valid data on completion of an operation. Therefore, any 128-bit data block can be split over multiple 32-bit word transfers; these can be mixed with other host transfers over the external interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	AES_DATA_IN_OUT	AES input data[127:96] / AES output data[127:96] Data registers for input/output block data to/from the EIP-120t. For normal operations, this register is not used, since data input and output is transferred from and to the AES core via DMA. For a host write operation, these registers must be written with the 128-bit input block for the next AES operation. Writing at a word-aligned offset within this address range stores the word (4 bytes) of data into the corresponding position of 4-word deep (16 bytes = 128-bit AES block) data input buffer. This buffer is used for the next AES operation. If the last data block is not completely filled with valid data (see notes below), it is allowed to write only the words with valid data. Next AES operation is triggered by writing to the input_ready flag of the AES_CTRL register. For a host read operation, these registers contain the 128-bit output block from the latest AES operation. Reading from a word-aligned offset within this address range reads one word (4 bytes) of data out the 4-word deep (16 bytes = 128-bits AES block) data output buffer. The words (4 words, one full block) should be read before the core will move the next block to the data output buffer. To empty the data output buffer, the output_ready flag of the AES_CTRL register must be written. For the modes with authentication (CBC-MAC, GCM and CCM), the invalid (message) bytes/words can be written with any data. Note: AES typically operates on 128 bits block multiple input data. The CTR, GCM and CCM modes form an exception. The last block of a CTR-mode message may contain less than 128 bits (refer to [NIST 800-38A]). For GCM/CCM, the last block of both AAD and message data may contain less than 128 bits (refer to [NIST 800-38D]). The EIP-120t automatically pads or masks misaligned ending data blocks with 0s for GCM, CCM and CBC-MAC. For CTR mode, the remaining data in an unaligned data block is ignored. Note: The AAD / authentication only data is not copied to the output buffer but only used for authentication.	WO	0x0000 0000

AES_AES_TAG_OUT_0

Address offset	0x570		
Physical Address	0x4008 B570	**Instance**	AES
Description	TAG registers The tag registers can be accessed via DMA or directly with host reads. These registers buffer the TAG from the EIP-120t. The registers are shared with the intermediate authentication result registers, but cannot be read until the processing is finished. While processing, a read from these registers returns 0s. If an operation does not return a TAG, reading from these registers returns an IV. If an operation returns a TAG plus an IV and both need to be read by the host, the host must first read the TAG followed by the IV. Reading these in reverse order will return the IV twice.		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:0	AES_TAG	AES_TAG[31:0] Bits [31:0] of the AES_TAG registers store the authentication value for the combined and authentication only modes. For a host read operation, these registers contain the last 128-bit TAG output of the EIP-120t; the TAG is available until the next context is written. This register will only contain valid data if the TAG is available and when the store_ready bit from AES_CTRL register is set. During processing or for operations/modes that do not return a TAG, reads from this register return data from the IV register.	RO	0x0000 0000

AES_AES_TAG_OUT_1

Address offset	0x574		
Physical Address	0x4008 B574	**Instance**	AES
Description	TAG registers The tag registers can be accessed via DMA or directly with host reads. These registers buffer the TAG from the EIP-120t. The registers are shared with the intermediate authentication result registers, but cannot be read until the processing is finished. While processing, a read from these registers returns 0s. If an operation does not return a TAG, reading from these registers returns an IV. If an operation returns a TAG plus an IV and both need to be read by the host, the host must first read the TAG followed by the IV. Reading these in reverse order returns the IV twice.		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:0	AES_TAG	AES_TAG[63:32] For a host read operation, these registers contain the last 128-bit TAG output of the EIP-120t; the TAG is available until the next context is written. This register contains valid data only if the TAG is available and when the store_ready bit from AES_CTRL register is set. During processing or for operations/modes that do not return a TAG, reads from this register return data from the IV register.	RO	0x0000 0000

AES_AES_TAG_OUT_2

Address offset	0x578		
Physical Address	0x4008 B578	**Instance**	AES
Description	TAG registers The tag registers can be accessed via DMA or directly with host reads. These registers buffer the TAG from the EIP-120t. The registers are shared with the intermediate authentication result registers, but cannot be read until the processing is finished. While processing, a read from these registers returns 0s. If an operation does not return a TAG, reading from these registers returns an IV. If an operation returns a TAG plus an IV and both need to be read by the host, the host must first read the TAG followed by the IV. Reading these in reverse order returns the IV twice.		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:0	AES_TAG	AES_TAG[95:64] For a host read operation, these registers contain the last 128-bit TAG output of the EIP-120t; the TAG is available until the next context is written. This register contains valid data only if the TAG is available and when the store_ready bit from AES_CTRL register is set. During processing or for operations/modes that do not return a TAG, reads from this register return data from the IV register.	RO	0x0000 0000

AES_AES_TAG_OUT_3

Address offset	0x57C		
Physical Address	0x4008 B57C	**Instance**	AES
Description	TAG registers The tag registers can be accessed via DMA or directly with host reads. These registers buffer the TAG from the EIP-120t. The registers are shared with the intermediate authentication result registers, but cannot be read until the processing is finished. While processing, a read from these registers returns 0s. If an operation does not return a TAG, reading from these registers returns an IV. If an operation returns a TAG plus an IV and both need to be read by the host, the host must first read the TAG followed by the IV. Reading these in reverse order returns the IV twice.		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:0	AES_TAG	AES_TAG[127:96] For a host read operation, these registers contain the last 128-bit TAG output of the EIP-120t; the TAG is available until the next context is written. This register contains valid data only if the TAG is available and when the store_ready bit from AES_CTRL register is set. During processing or for operations/modes that do not return a TAG, reads from this register return data from the IV register.	RO	0x0000 0000

AES_HASH_DATA_IN_0

Address offset	0x600		
Physical Address	0x4008 B600	**Instance**	AES
Description	HASH data input registers The data input registers should be used to provide input data to the hash module through the slave interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DATA_IN	HASH_DATA_IN[31:0] These registers must be written with the 512-bit input data. The data lines are connected directly to the data input of the hash module and hence into the engine's internal data buffer. Writing to each of the registers triggers a corresponding 32-bit write enable to the internal buffer. Note: The host may only write the input data buffer when the rfd_in bit of the HASH_IO_BUF_CTRL register is high. If the rfd_in bit is 0, the engine is busy with processing. During processing, it is not allowed to write new input data. For message lengths larger than 64 bytes, multiple blocks of data are written to this input buffer using a handshake through flags of the HASH_IO_BUF_CTRL register. All blocks except the last are required to be 512 bits in size. If the last block is not 512 bits long, only the least significant bits of data must be written, but they must be padded with 0s to the next 32-bit boundary. Host read operations from these register addresses return 0s.	WO	0x0000 0000

AES_HASH_DATA_IN_1

Address offset	0x604		
Physical Address	0x4008 B604	**Instance**	AES
Description	HASH data input registers The data input registers should be used to provide input data to the hash module through the slave interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DATA_IN	HASH_DATA_IN[63:32] These registers must be written with the 512-bit input data. The data lines are connected directly to the data input of the hash module and hence into the engine's internal data buffer. Writing to each of the registers triggers a corresponding 32-bit write enable to the internal buffer. Note: The host may only write the input data buffer when the rfd_in bit of the HASH_IO_BUF_CTRL register is high. If the rfd_in bit is 0, the engine is busy with processing. During processing, it is not allowed to write new input data. For message lengths larger than 64 bytes, multiple blocks of data are written to this input buffer using a handshake through flags of the HASH_IO_BUF_CTRL register. All blocks except the last are required to be 512 bits in size. If the last block is not 512 bits long, only the least significant bits of data must be written, but they must be padded with 0s to the next 32-bit boundary. Host read operations from these register addresses return 0s.	WO	0x0000 0000

AES_HASH_DATA_IN_2

Address offset	0x608		
Physical Address	0x4008 B608	**Instance**	AES
Description	HASH data input registers The data input registers should be used to provide input data to the hash module through the slave interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DATA_IN	HASH_DATA_IN[95:64] These registers must be written with the 512-bit input data. The data lines are connected directly to the data input of the hash module and hence into the engine's internal data buffer. Writing to each of the registers triggers a corresponding 32-bit write enable to the internal buffer. Note: The host may only write the input data buffer when the rfd_in bit of the HASH_IO_BUF_CTRL register is high. If the rfd_in bit is 0, the engine is busy with processing. During processing, it is not allowed to write new input data. For message lengths larger than 64 bytes, multiple blocks of data are written to this input buffer using a handshake through flags of the HASH_IO_BUF_CTRL register. All blocks except the last are required to be 512 bits in size. If the last block is not 512 bits long, only the least significant bits of data must be written, but they must be padded with 0s to the next 32-bit boundary. Host read operations from these register addresses return 0s.	WO	0x0000 0000

AES_HASH_DATA_IN_3

Address offset	0x60C		
Physical Address	0x4008 B60C	**Instance**	AES
Description	HASH data input registers The data input registers should be used to provide input data to the hash module through the slave interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DATA_IN	HASH_DATA_IN[127:96] These registers must be written with the 512-bit input data. The data lines are connected directly to the data input of the hash module and hence into the engine's internal data buffer. Writing to each of the registers triggers a corresponding 32-bit write enable to the internal buffer. Note: The host may only write the input data buffer when the rfd_in bit of the HASH_IO_BUF_CTRL register is high. If the rfd_in bit is 0, the engine is busy with processing. During processing, it is not allowed to write new input data. For message lengths larger than 64 bytes, multiple blocks of data are written to this input buffer using a handshake through flags of the HASH_IO_BUF_CTRL register. All blocks except the last are required to be 512 bits in size. If the last block is not 512 bits long, only the least significant bits of data must be written, but they must be padded with 0s to the next 32-bit boundary. Host read operations from these register addresses return 0s.	WO	0x0000 0000

AES_HASH_DATA_IN_4

Address offset	0x610		
Physical Address	0x4008 B610	**Instance**	AES
Description	HASH data input registers The data input registers should be used to provide input data to the hash module through the slave interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DATA_IN	HASH_DATA_IN[159:128] These registers must be written with the 512-bit input data. The data lines are connected directly to the data input of the hash module and hence into the engine's internal data buffer. Writing to each of the registers triggers a corresponding 32-bit write enable to the internal buffer. Note: The host may only write the input data buffer when the rfd_in bit of the HASH_IO_BUF_CTRL register is high. If the rfd_in bit is 0, the engine is busy with processing. During processing, it is not allowed to write new input data. For message lengths larger than 64 bytes, multiple blocks of data are written to this input buffer using a handshake through flags of the HASH_IO_BUF_CTRL register. All blocks except the last are required to be 512 bits in size. If the last block is not 512 bits long, only the least significant bits of data must be written, but they must be padded with 0s to the next 32-bit boundary. Host read operations from these register addresses return 0s.	WO	0x0000 0000

AES_HASH_DATA_IN_5

Address offset	0x614		
Physical Address	0x4008 B614	**Instance**	AES
Description	HASH data input registers The data input registers should be used to provide input data to the hash module through the slave interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DATA_IN	HASH_DATA_IN[191:160] These registers must be written with the 512-bit input data. The data lines are connected directly to the data input of the hash module and hence into the engine's internal data buffer. Writing to each of the registers triggers a corresponding 32-bit write enable to the internal buffer. Note: The host may only write the input data buffer when the rfd_in bit of the HASH_IO_BUF_CTRL register is high. If the rfd_in bit is 0, the engine is busy with processing. During processing, it is not allowed to write new input data. For message lengths larger than 64 bytes, multiple blocks of data are written to this input buffer using a handshake through flags of the HASH_IO_BUF_CTRL register. All blocks except the last are required to be 512 bits in size. If the last block is not 512 bits long, only the least significant bits of data must be written, but they must be padded with 0s to the next 32-bit boundary. Host read operations from these register addresses return 0s.	WO	0x0000 0000

AES_HASH_DATA_IN_6

Address offset	0x618		
Physical Address	0x4008 B618	**Instance**	AES
Description	HASH data input registers The data input registers should be used to provide input data to the hash module through the slave interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DATA_IN	HASH_DATA_IN[223:192] These registers must be written with the 512-bit input data. The data lines are connected directly to the data input of the hash module and hence into the engine's internal data buffer. Writing to each of the registers triggers a corresponding 32-bit write enable to the internal buffer. Note: The host may only write the input data buffer when the rfd_in bit of the HASH_IO_BUF_CTRL register is high. If the rfd_in bit is 0, the engine is busy with processing. During processing, it is not allowed to write new input data. For message lengths larger than 64 bytes, multiple blocks of data are written to this input buffer using a handshake through flags of the HASH_IO_BUF_CTRL register. All blocks except the last are required to be 512 bits in size. If the last block is not 512 bits long, only the least significant bits of data must be written, but they must be padded with 0s to the next 32-bit boundary. Host read operations from these register addresses return 0s.	WO	0x0000 0000

AES_HASH_DATA_IN_7

Address offset	0x61C		
Physical Address	0x4008 B61C	**Instance**	AES
Description	HASH data input registers The data input registers should be used to provide input data to the hash module through the slave interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DATA_IN	HASH_DATA_IN[255:224] These registers must be written with the 512-bit input data. The data lines are connected directly to the data input of the hash module and hence into the engine's internal data buffer. Writing to each of the registers triggers a corresponding 32-bit write enable to the internal buffer. Note: The host may only write the input data buffer when the rfd_in bit of the HASH_IO_BUF_CTRL register is high. If the rfd_in bit is 0, the engine is busy with processing. During processing, it is not allowed to write new input data. For message lengths larger than 64 bytes, multiple blocks of data are written to this input buffer using a handshake through flags of the HASH_IO_BUF_CTRL register. All blocks except the last are required to be 512 bits in size. If the last block is not 512 bits long, only the least significant bits of data must be written, but they must be padded with 0s to the next 32-bit boundary. Host read operations from these register addresses return 0s.	WO	0x0000 0000

AES_HASH_DATA_IN_8

Address offset	0x620		
Physical Address	0x4008 B620	**Instance**	AES
Description	HASH data input registers The data input registers should be used to provide input data to the hash module through the slave interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DATA_IN	HASH_DATA_IN[287:256] These registers must be written with the 512-bit input data. The data lines are connected directly to the data input of the hash module and hence into the engine's internal data buffer. Writing to each of the registers triggers a corresponding 32-bit write enable to the internal buffer. Note: The host may only write the input data buffer when the rfd_in bit of the HASH_IO_BUF_CTRL register is high. If the rfd_in bit is 0, the engine is busy with processing. During processing, it is not allowed to write new input data. For message lengths larger than 64 bytes, multiple blocks of data are written to this input buffer using a handshake through flags of the HASH_IO_BUF_CTRL register. All blocks except the last are required to be 512 bits in size. If the last block is not 512 bits long, only the least significant bits of data must be written, but they must be padded with 0s to the next 32-bit boundary. Host read operations from these register addresses return 0s.	WO	0x0000 0000

AES_HASH_DATA_IN_9

Address offset	0x624		
Physical Address	0x4008 B624	**Instance**	AES
Description	HASH data input registers The data input registers should be used to provide input data to the hash module through the slave interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DATA_IN	HASH_DATA_IN[319:288] These registers must be written with the 512-bit input data. The data lines are connected directly to the data input of the hash module and hence into the engine's internal data buffer. Writing to each of the registers triggers a corresponding 32-bit write enable to the internal buffer. Note: The host may only write the input data buffer when the rfd_in bit of the HASH_IO_BUF_CTRL register is high. If the rfd_in bit is 0, the engine is busy with processing. During processing, it is not allowed to write new input data. For message lengths larger than 64 bytes, multiple blocks of data are written to this input buffer using a handshake through flags of the HASH_IO_BUF_CTRL register. All blocks except the last are required to be 512 bits in size. If the last block is not 512 bits long, only the least significant bits of data must be written, but they must be padded with 0s to the next 32-bit boundary. Host read operations from these register addresses return 0s.	WO	0x0000 0000

AES_HASH_DATA_IN_10

Address offset	0x628		
Physical Address	0x4008 B628	**Instance**	AES
Description	HASH data input registers The data input registers should be used to provide input data to the hash module through the slave interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DATA_IN	HASH_DATA_IN[351:320] These registers must be written with the 512-bit input data. The data lines are connected directly to the data input of the hash module and hence into the engine's internal data buffer. Writing to each of the registers triggers a corresponding 32-bit write enable to the internal buffer. Note: The host may only write the input data buffer when the rfd_in bit of the HASH_IO_BUF_CTRL register is high. If the rfd_in bit is 0, the engine is busy with processing. During processing, it is not allowed to write new input data. For message lengths larger than 64 bytes, multiple blocks of data are written to this input buffer using a handshake through flags of the HASH_IO_BUF_CTRL register. All blocks except the last are required to be 512 bits in size. If the last block is not 512 bits long, only the least significant bits of data must be written, but they must be padded with 0s to the next 32-bit boundary. Host read operations from these register addresses return 0s.	WO	0x0000 0000

AES_HASH_DATA_IN_11

Address offset	0x62C		
Physical Address	0x4008 B62C	**Instance**	AES
Description	HASH data input registers The data input registers should be used to provide input data to the hash module through the slave interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DATA_IN	HASH_DATA_IN[383:352] These registers must be written with the 512-bit input data. The data lines are connected directly to the data input of the hash module and hence into the engine's internal data buffer. Writing to each of the registers triggers a corresponding 32-bit write enable to the internal buffer. Note: The host may only write the input data buffer when the rfd_in bit of the HASH_IO_BUF_CTRL register is high. If the rfd_in bit is 0, the engine is busy with processing. During processing, it is not allowed to write new input data. For message lengths larger than 64 bytes, multiple blocks of data are written to this input buffer using a handshake through flags of the HASH_IO_BUF_CTRL register. All blocks except the last are required to be 512 bits in size. If the last block is not 512 bits long, only the least significant bits of data must be written, but they must be padded with 0s to the next 32-bit boundary. Host read operations from these register addresses return 0s.	WO	0x0000 0000

AES_HASH_DATA_IN_12

Address offset	0x630		
Physical Address	0x4008 B630	**Instance**	AES
Description	HASH data input registers The data input registers should be used to provide input data to the hash module through the slave interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DATA_IN	HASH_DATA_IN[415:384] These registers must be written with the 512-bit input data. The data lines are connected directly to the data input of the hash module and hence into the engine's internal data buffer. Writing to each of the registers triggers a corresponding 32-bit write enable to the internal buffer. Note: The host may only write the input data buffer when the rfd_in bit of the HASH_IO_BUF_CTRL register is high. If the rfd_in bit is 0, the engine is busy with processing. During processing, it is not allowed to write new input data. For message lengths larger than 64 bytes, multiple blocks of data are written to this input buffer using a handshake through flags of the HASH_IO_BUF_CTRL register. All blocks except the last are required to be 512 bits in size. If the last block is not 512 bits long, only the least significant bits of data must be written, but they must be padded with 0s to the next 32-bit boundary. Host read operations from these register addresses return 0s.	WO	0x0000 0000

AES_HASH_DATA_IN_13

Address offset	0x634		
Physical Address	0x4008 B634	**Instance**	AES
Description	HASH data input registers The data input registers should be used to provide input data to the hash module through the slave interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DATA_IN	HASH_DATA_IN[447:416] These registers must be written with the 512-bit input data. The data lines are connected directly to the data input of the hash module and hence into the engine's internal data buffer. Writing to each of the registers triggers a corresponding 32-bit write enable to the internal buffer. Note: The host may only write the input data buffer when the rfd_in bit of the HASH_IO_BUF_CTRL register is high. If the rfd_in bit is 0, the engine is busy with processing. During processing, it is not allowed to write new input data. For message lengths larger than 64 bytes, multiple blocks of data are written to this input buffer using a handshake through flags of the HASH_IO_BUF_CTRL register. All blocks except the last are required to be 512 bits in size. If the last block is not 512 bits long, only the least significant bits of data must be written, but they must be padded with 0s to the next 32-bit boundary. Host read operations from these register addresses return 0s.	WO	0x0000 0000

AES_HASH_DATA_IN_14

Address offset	0x638		
Physical Address	0x4008 B638	**Instance**	AES
Description	HASH data input registers The data input registers should be used to provide input data to the hash module through the slave interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DATA_IN	HASH_DATA_IN[479:448] These registers must be written with the 512-bit input data. The data lines are connected directly to the data input of the hash module and hence into the engine's internal data buffer. Writing to each of the registers triggers a corresponding 32-bit write enable to the internal buffer. Note: The host may only write the input data buffer when the rfd_in bit of the HASH_IO_BUF_CTRL register is high. If the rfd_in bit is 0, the engine is busy with processing. During processing, it is not allowed to write new input data. For message lengths larger than 64 bytes, multiple blocks of data are written to this input buffer using a handshake through flags of the HASH_IO_BUF_CTRL register. All blocks except the last are required to be 512 bits in size. If the last block is not 512 bits long, only the least significant bits of data must be written, but they must be padded with 0s to the next 32-bit boundary. Host read operations from these register addresses return 0s.	WO	0x0000 0000

AES_HASH_DATA_IN_15

Address offset	0x63C		
Physical Address	0x4008 B63C	**Instance**	AES
Description	HASH data input registers The data input registers should be used to provide input data to the hash module through the slave interface.		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DATA_IN	HASH_DATA_IN[511:480] These registers must be written with the 512-bit input data. The data lines are connected directly to the data input of the hash module and hence into the engine's internal data buffer. Writing to each of the registers triggers a corresponding 32-bit write enable to the internal buffer. Note: The host may only write the input data buffer when the rfd_in bit of the HASH_IO_BUF_CTRL register is high. If the rfd_in bit is 0, the engine is busy with processing. During processing, it is not allowed to write new input data. For message lengths larger than 64 bytes, multiple blocks of data are written to this input buffer using a handshake through flags of the HASH_IO_BUF_CTRL register. All blocks except the last are required to be 512 bits in size. If the last block is not 512 bits long, only the least significant bits of data must be written, but they must be padded with 0s to the next 32-bit boundary. Host read operations from these register addresses return 0s.	WO	0x0000 0000

AES_HASH_IO_BUF_CTRL

Address offset	0x640		
Physical Address	0x4008 B640	**Instance**	AES
Description	Input/output buffer control and status register This register pair shares a single address location and contains bits that control and monitor the data flow between the host and the hash engine.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PAD_DMA_MESSAGE	GET_DIGEST	PAD_MESSAGE	RESERVED		RFD_IN	DATA_IN_AV	OUTPUT_FULL

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RW	0x00 0000
7	PAD_DMA_MESSAGE	Note: This bit must only be used when data is supplied through the DMA. It should not be used when data is supplied through the slave interface. This bit indicates whether the hash engine has to pad the message, received through the DMA and finalize the hash. When set to 1, the hash engine pads the last block using the programmed length. After padding, the final hash result is calculated. When set to 0, the hash engine treats the last written block as block-size aligned and calculates the intermediate digest. This bit is automatically cleared when the last DMA data block is arrived in the hash engine.	RW	0
6	GET_DIGEST	Note: The bit description below is only applicable when data is sent through the slave interface. This bit must be set to 0 when data is received through the DMA. This bit indicates whether the hash engine should provide the hash digest. When provided simultaneously with data_in_av, the hash digest is provided after processing the data that is currently in the HASH_DATA_IN register. When provided without data_in_av, the current internal digest buffer value is copied to the HASH_DIGEST_n registers. The host must write a 1 to this bit to make the intermediate hash digest available. Writing 0 to this bit has no effect. This bit is automatically cleared (that is, reads 0) when the hash engine has processed the contents of the HASH_DATA_IN register. In the period between this bit is set by the host and the actual HASH_DATA_IN processing, this bit reads 1.	RW	0
5	PAD_MESSAGE	Note: The bit description below is only applicable when data is sent through the slave interface. This bit must be set to 0 when data is received through the DMA. This bit indicates that the HASH_DATA_IN registers hold the last data of the message and hash padding must be applied. The host must write this bit to 1 in order to indicate to the hash engine that the HASH_DATA_IN register currently holds the last data of the message. When pad_message is set to 1, the hash engine will add padding bits to the data currently in the HASH_DATA_IN register. When the last message block is smaller than 512 bits, the pad_message bit must be set to 1 together with the data_in_av bit. When the last message block is equal to 512 bits, pad_message may be set together with data_in_av. In this case the pad_message bit may also be set after the last data block has been written to the hash engine (so when the rfd_in bit has become 1 again after writing the last data block). Writing 0 to this bit has no effect. This bit is automatically cleared (i.e. reads 0) by the hash engine. This bit reads 1 between the time it was set by the host and the hash engine interpreted its value.	RW	0
4:3	RESERVED	This bit field is reserved.	RW	0x0
2	RFD_IN	Note: The bit description below is only applicable when data is sent through the slave interface. This bit can be ignored when data is received through the DMA. Read-only status of the input buffer of the hash engine. When 1, the input buffer of the hash engine can accept new data; the HASH_DATA_IN registers can safely be populated with new data. When 0, the input buffer of the hash engine is processing the data that is currently in HASH_DATA_IN; writing new data to these registers is not allowed.	RW	1

Bits	Field Name	Description	Type	Reset
1	DATA_IN_AV	Note: The bit description below is only applicable when data is sent through the slave interface. This bit must be set to 0 when data is received through the DMA. This bit indicates that the HASH_DATA_IN registers contain new input data for processing. The host must write a 1 to this bit to start processing the data in HASH_DATA_IN; the hash engine will process the new data as soon as it is ready for it (rfd_in bit is 1). Writing 0 to this bit has no effect. This bit is automatically cleared (i.e. reads as 0) when the hash engine starts processing the HASH_DATA_IN contents. This bit reads 1 between the time it was set by the host and the hash engine actually starts processing the input data block.	RW	0
0	OUTPUT_FULL	Indicates that the output buffer registers (HASH_DIGEST_n) are available for reading by the host. When this bit reads 0, the output buffer registers are released; the hash engine is allowed to write new data to it. In this case, the registers should not be read by the host. When this bit reads 1, the hash engine has stored the result of the latest hash operation in the output buffer registers. As long as this bit reads 1, the host may read output buffer registers and the hash engine is prevented from writing new data to the output buffer. After retrieving the hash result data from the output buffer, the host must write a 1 to this bit to clear it. This makes the digest output buffer available for the hash engine to store new hash results. Writing 0 to this bit has no effect. Note: If this bit is asserted (1) no new operation should be started before the digest is retrieved from the hash engine and this bit is cleared (0).	RW	0

AES_HASH_MODE_IN

Address offset	0x644		
Physical Address	0x4008 B644	**Instance**	AES
Description	Hash mode register		
Type	WO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												SHA256_MODE	RESERVED		NEW_HASH

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	This bit field is reserved.	WO	0x000 0000
3	SHA256_MODE	The host must write this bit with 1 before processing a hash session.	WO	0
2:1	RESERVED	This bit field is reserved.	WO	0x0
0	NEW_HASH	When set to 1, it indicates that the hash engine must start processing a new hash session. The HASH_DIGEST_n registers will automatically be loaded with the initial hash algorithm constants of the selected hash algorithm. When this bit is 0 while the hash processing is started, the initial hash algorithm constants are not loaded in the HASH_DIGEST_n registers. The hash engine will start processing with the digest that is currently in its internal HASH_DIGEST_n registers. This bit is automatically cleared when hash processing is started.	WO	0

AES_HASH_LENGTH_IN_L

Address offset	0x648		
Physical Address	0x4008 B648	**Instance**	AES
Description	Hash length register		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	LENGTH_IN	LENGTH_IN[31:0] Message length registers. The content of these registers is used by the hash engine during the message padding phase of the hash session. The data lines of this registers are directly connected to the interface of the hash engine. For a write operation by the host, these registers should be written with the message length in bits. Final hash operations: The total input data length must be programmed for new hash operations that require finalization (padding). The input data must be provided through the slave or DMA interface. Continued hash operations (finalized): For continued hash operations that require finalization, the total message length must be programmed, including the length of previously hashed data that corresponds to the written input digest. Non-final hash operations: For hash operations that do not require finalization (input data length is multiple of 512-bits which is SHA-256 data block size), the length field does not need to be programmed since not used by the operation. If the message length in bits is below (2^32-1), then only HASH_LENGTH_IN_L needs to be written. The hardware automatically sets HASH_LENGTH_IN_H to 0s in this case. The host may write the length register at any time during the hash session when the rfd_in bit of the HASH_IO_BUF_CTRL is high. The length register must be written before the last data of the active hash session is written into the hash engine. host read operations from these register locations will return 0s. Note: When getting data from DMA, this register must be programmed before DMA is programmed to start.	WO	0x0000 0000

AES_HASH_LENGTH_IN_H

Address offset	0x64C		
Physical Address	0x4008 B64C	**Instance**	AES
Description	Hash length register		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:0	LENGTH_IN	LENGTH_IN[63:32] Message length registers. The content of these registers is used by the hash engine during the message padding phase of the hash session. The data lines of this registers are directly connected to the interface of the hash engine. For a write operation by the host, these registers should be written with the message length in bits. Final hash operations: The total input data length must be programmed for new hash operations that require finalization (padding). The input data must be provided through the slave or DMA interface. Continued hash operations (finalized): For continued hash operations that require finalization, the total message length must be programmed, including the length of previously hashed data that corresponds to the written input digest. Non-final hash operations: For hash operations that do not require finalization (input data length is multiple of 512-bits which is SHA-256 data block size), the length field does not need to be programmed since not used by the operation. If the message length in bits is below (2^32-1), then only HASH_LENGTH_IN_L needs to be written. The hardware automatically sets HASH_LENGTH_IN_H to 0s in this case. The host may write the length register at any time during the hash session when the rfd_in bit of the HASH_IO_BUF_CTRL is high. The length register must be written before the last data of the active hash session is written into the hash engine. host read operations from these register locations will return 0s. Note: When getting data from DMA, this register must be programmed before DMA is programmed to start.	WO	0x0000 0000

AES_HASH_DIGEST_A

Address offset	0x650		
Physical Address	0x4008 B650	**Instance**	AES
Description	Hash digest registers The hash digest registers consist of eight 32-bit registers, named HASH_DIGEST_A to HASH_DIGEST_H. After processing a message, the output digest can be read from these registers. These registers can be written with an intermediate hash result for continued hash operations.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DIGEST	HASH_DIGEST[31:0] Hash digest registers Write operation: Continued hash: These registers should be written with the context data, before the start of a resumed hash session (the new_hash bit in the HASH_MODE register is 0 when starting a hash session). New hash: When initiating a new hash session (the new_hash bit in the HASH_MODE register is high), the internal digest registers are automatically set to the SHA-256 algorithm constant and these register should not be written. Reading from these registers provides the intermediate hash result (non-final hash operation) or the final hash result (final hash operation) after data processing.	RW	0x0000 0000

AES_HASH_DIGEST_B

Address offset	0x654		
Physical Address	0x4008 B654	**Instance**	AES
Description	Hash digest registers The hash digest registers consist of eight 32-bit registers, named HASH_DIGEST_A to HASH_DIGEST_H. After processing a message, the output digest can be read from these registers. These registers can be written with an intermediate hash result for continued hash operations.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DIGEST	HASH_DIGEST[63:32] Hash digest registers Write operation: Continued hash: These registers should be written with the context data, before the start of a resumed hash session (the new_hash bit in the HASH_MODE register is 0 when starting a hash session). New hash: When initiating a new hash session (the new_hash bit in the HASH_MODE register is high), the internal digest registers are automatically set to the SHA-256 algorithm constant and these register should not be written. Reading from these registers provides the intermediate hash result (non-final hash operation) or the final hash result (final hash operation) after data processing.	RW	0x0000 0000

AES_HASH_DIGEST_C

Address offset	0x658		
Physical Address	0x4008 B658	**Instance**	AES
Description	Hash digest registers The hash digest registers consist of eight 32-bit registers, named HASH_DIGEST_A to HASH_DIGEST_H. After processing a message, the output digest can be read from these registers. These registers can be written with an intermediate hash result for continued hash operations.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DIGEST	HASH_DIGEST[95:64] Hash digest registers Write operation: Continued hash: These registers should be written with the context data, before the start of a resumed hash session (the new_hash bit in the HASH_MODE register is 0 when starting a hash session). New hash: When initiating a new hash session (the new_hash bit in the HASH_MODE register is high), the internal digest registers are automatically set to the SHA-256 algorithm constant and these register should not be written. Reading from these registers provides the intermediate hash result (non-final hash operation) or the final hash result (final hash operation) after data processing.	RW	0x0000 0000

AES_HASH_DIGEST_D

Address offset	0x65C		
Physical Address	0x4008 B65C	**Instance**	AES
Description	Hash digest registers The hash digest registers consist of eight 32-bit registers, named HASH_DIGEST_A to HASH_DIGEST_H. After processing a message, the output digest can be read from these registers. These registers can be written with an intermediate hash result for continued hash operations.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DIGEST	HASH_DIGEST[127:96] Hash digest registers Write operation: Continued hash: These registers should be written with the context data, before the start of a resumed hash session (the new_hash bit in the HASH_MODE register is 0 when starting a hash session). New hash: When initiating a new hash session (the new_hash bit in the HASH_MODE register is high), the internal digest registers are automatically set to the SHA-256 algorithm constant and these register should not be written. Reading from these registers provides the intermediate hash result (non-final hash operation) or the final hash result (final hash operation) after data processing.	RW	0x0000 0000

AES_HASH_DIGEST_E

Address offset	0x660		
Physical Address	0x4008 B660	**Instance**	AES
Description	Hash digest registers The hash digest registers consist of eight 32-bit registers, named HASH_DIGEST_A to HASH_DIGEST_H. After processing a message, the output digest can be read from these registers. These registers can be written with an intermediate hash result for continued hash operations.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DIGEST	HASH_DIGEST[159:128] Hash digest registers Write operation: Continued hash: These registers should be written with the context data, before the start of a resumed hash session (the new_hash bit in the HASH_MODE register is 0 when starting a hash session). New hash: When initiating a new hash session (the new_hash bit in the HASH_MODE register is high), the internal digest registers are automatically set to the SHA-256 algorithm constant and these register should not be written. Reading from these registers provides the intermediate hash result (non-final hash operation) or the final hash result (final hash operation) after data processing.	RW	0x0000 0000

AES_HASH_DIGEST_F

Address offset	0x664		
Physical Address	0x4008 B664	**Instance**	AES
Description	Hash digest registers The hash digest registers consist of eight 32-bit registers, named HASH_DIGEST_A to HASH_DIGEST_H. After processing a message, the output digest can be read from these registers. These registers can be written with an intermediate hash result for continued hash operations.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DIGEST	HASH_DIGEST[191:160] Hash digest registers Write operation: Continued hash: These registers should be written with the context data, before the start of a resumed hash session (the new_hash bit in the HASH_MODE register is 0 when starting a hash session). New hash: When initiating a new hash session (the new_hash bit in the HASH_MODE register is high), the internal digest registers are automatically set to the SHA-256 algorithm constant and these register should not be written. Reading from these registers provides the intermediate hash result (non-final hash operation) or the final hash result (final hash operation) after data processing.	RW	0x0000 0000

AES_HASH_DIGEST_G

Address offset	0x668		
Physical Address	0x4008 B668	**Instance**	AES
Description	Hash digest registers The hash digest registers consist of eight 32-bit registers, named HASH_DIGEST_A to HASH_DIGEST_H. After processing a message, the output digest can be read from these registers. These registers can be written with an intermediate hash result for continued hash operations.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DIGEST	HASH_DIGEST[223:192] Hash digest registers Write operation: Continued hash: These registers should be written with the context data, before the start of a resumed hash session (the new_hash bit in the HASH_MODE register is 0 when starting a hash session). New hash: When initiating a new hash session (the new_hash bit in the HASH_MODE register is high), the internal digest registers are automatically set to the SHA-256 algorithm constant and these register should not be written. Reading from these registers provides the intermediate hash result (non-final hash operation) or the final hash result (final hash operation) after data processing.	RW	0x0000 0000

AES_HASH_DIGEST_H

Address offset	0x66C		
Physical Address	0x4008 B66C	**Instance**	AES
Description	Hash digest registers The hash digest registers consist of eight 32-bit registers, named HASH_DIGEST_A to HASH_DIGEST_H. After processing a message, the output digest can be read from these registers. These registers can be written with an intermediate hash result for continued hash operations.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:0	HASH_DIGEST	HASH_DIGEST[255:224] Hash digest registers Write operation: Continued hash: These registers should be written with the context data, before the start of a resumed hash session (the new_hash bit in the HASH_MODE register is 0 when starting a hash session). New hash: When initiating a new hash session (the new_hash bit in the HASH_MODE register is high), the internal digest registers are automatically set to the SHA-256 algorithm constant and these register should not be written. Reading from these registers provides the intermediate hash result (non-final hash operation) or the final hash result (final hash operation) after data processing.	RW	0x0000 0000

AES_CTRL_ALG_SEL

Address offset	0x700		
Physical Address	0x4008 B700	**Instance**	AES
Description	Algorithm select This algorithm selection register configures the internal destination of the DMA controller.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TAG	RESERVED																												HASH	AES	KEYSTORE

Bits	Field Name	Description	Type	Reset
31	TAG	If this bit is cleared to 0, the DMA operation involves only data. If this bit is set, the DMA operation includes a TAG (Authentication Result / Digest). For SHA-256 operation, a DMA must be set up for both input data and TAG. For any other selected module, setting this bit only allows a DMA that reads the TAG. No data allowed to be transferred to or from the selected module via the DMA.	RW	0
30:3	RESERVED	This bit field is reserved.	RW	0x000 0000
2	HASH	If set to one, selects the hash engine as destination for the DMA The maximum transfer size to DMA engine is set to 64 bytes for reading and 32 bytes for writing (the latter is only applicable if the hash result is written out through the DMA).	RW	0
1	AES	If set to one, selects the AES engine as source/destination for the DMA The read and write maximum transfer size to the DMA engine is set to 16 bytes.	RW	0

Bits	Field Name	Description	Type	Reset
0	KEYSTORE	If set to one, selects the Key Store as destination for the DMA The maximum transfer size to DMA engine is set to 32 bytes (however transfers of 16, 24 and 32 bytes are allowed)	RW	0

AES_CTRL_PROT_EN

Address offset	0x704		
Physical Address	0x4008 B704	**Instance**	AES
Description	Master PROT privileged access enable This register enables the second bit (bit [1]) of the AHB HPROT bus of the AHB master interface when a read action of key(s) is performed on the AHB master interface for writing keys into the store module.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	RW	0x0000 0000
0	PROT_EN	If this bit is cleared to 0, m_h_prot[1] on the AHB mater interface always remains 0. If this bit is set to one, the m_h_prot[1] signal on the master AHB bus is asserted to 1 if an AHB read operation is performed, using DMA, with the key store module as destination.	RW	0

AES_CTRL_SW_RESET

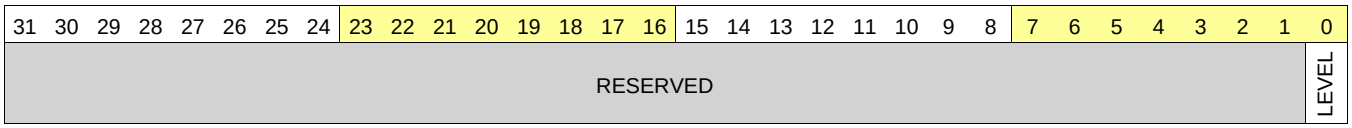
Address offset	0x740		
Physical Address	0x4008 B740	**Instance**	AES
Description	Software reset		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	RW	0x0000 0000
0	SW_RESET	If this bit is set to 1, the following modules are reset: - Master control internal state is reset. That includes interrupt, error status register, and result available interrupt generation FSM. - Key store module state is reset. That includes clearing the written area flags; therefore, the keys must be reloaded to the key store module. Writing 0 has no effect. The bit is self cleared after executing the reset.	RW	0

AES_CTRL_INT_CFG

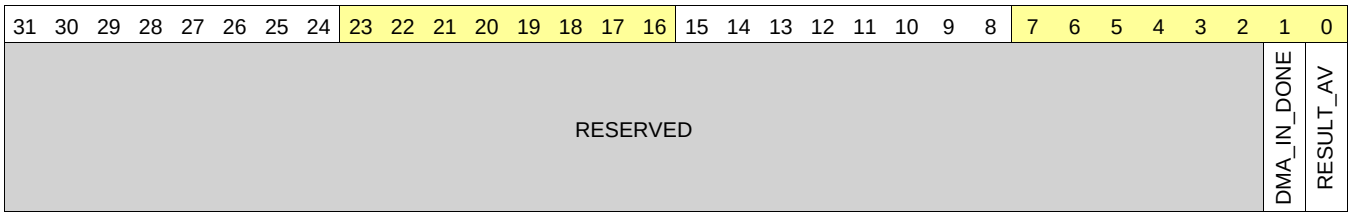
Address offset	0x780		
Physical Address	0x4008 B780	**Instance**	AES
Description	Interrupt configuration		
Type	RW		



Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	RW	0x0000 0000
0	LEVEL	If this bit is 0, the interrupt output is a pulse. If this bit is set to 1, the interrupt is a level interrupt that must be cleared by writing the interrupt clear register. This bit is applicable for both interrupt output signals.	RW	0

AES_CTRL_INT_EN

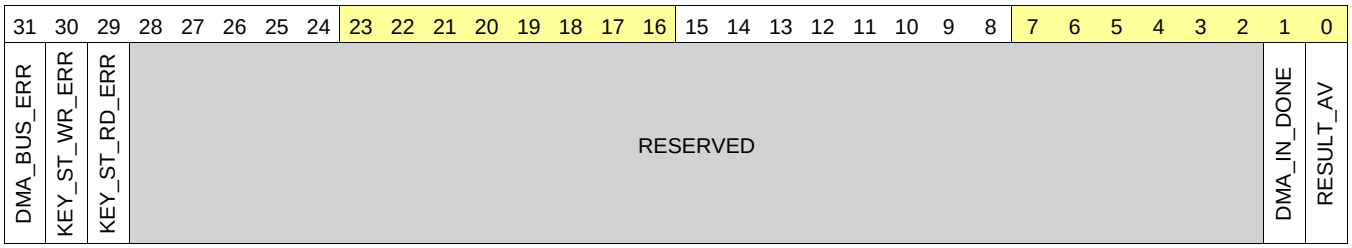
Address offset	0x784		
Physical Address	0x4008 B784	**Instance**	AES
Description	Interrupt enable		
Type	RW		



Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	RW	0x0000 0000
1	DMA_IN_DONE	If this bit is set to 0, the DMA input done (irq_dma_in_done) interrupt output is disabled and remains 0. If this bit is set to 1, the DMA input done interrupt output is enabled.	RW	0
0	RESULT_AV	If this bit is set to 0, the result available (irq_result_av) interrupt output is disabled and remains 0. If this bit is set to 1, the result available interrupt output is enabled.	RW	0

AES_CTRL_INT_CLR

Address offset	0x788		
Physical Address	0x4008 B788	**Instance**	AES
Description	Interrupt clear		
Type	WO		



Bits	Field Name	Description	Type	Reset
31	DMA_BUS_ERR	If 1 is written to this bit, the DMA bus error status is cleared. Writing 0 has no effect.	WO	0
30	KEY_ST_WR_ERR	If 1 is written to this bit, the key store write error status is cleared. Writing 0 has no effect.	WO	0

Bits	Field Name	Description	Type	Reset
29	KEY_ST_RD_ERR	If 1 is written to this bit, the key store read error status is cleared. Writing 0 has no effect.	WO	0
28:2	RESERVED	This bit field is reserved.	WO	0x000 0000
1	DMA_IN_DONE	If 1 is written to this bit, the DMA in done (irq_dma_in_done) interrupt output is cleared. Writing 0 has no effect. Note that clearing an interrupt makes sense only if the interrupt output is programmed as level (refer to CTRL_INT_CFG).	WO	0
0	RESULT_AV	If 1 is written to this bit, the result available (irq_result_av) interrupt output is cleared. Writing 0 has no effect. Note that clearing an interrupt makes sense only if the interrupt output is programmed as level (refer to CTRL_INT_CFG).	WO	0

AES_CTRL_INT_SET

Address offset	0x78C		
Physical Address	0x4008 B78C	**Instance**	AES
Description	Interrupt set		
Type	WO		

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

Bits	Field Name	Description	Type	Reset
31:2	RESERVED	This bit field is reserved.	WO	0x0000 0000
1	DMA_IN_DONE	If 1 is written to this bit, the DMA data in done (irq_dma_in_done) interrupt output is set to one. Writing 0 has no effect. If the interrupt configuration register is programmed to pulse, clearing the DMA data in done (irq_dma_in_done) interrupt is not needed. If it is programmed to level, clearing the interrupt output should be done by writing the interrupt clear register (CTRL_INT_CLR).	WO	0
0	RESULT_AV	If 1 is written to this bit, the result available (irq_result_av) interrupt output is set to one. Writing 0 has no effect. If the interrupt configuration register is programmed to pulse, clearing the result available (irq_result_av) interrupt is not needed. If it is programmed to level, clearing the interrupt output should be done by writing the interrupt clear register (CTRL_INT_CLR).	WO	0

AES_CTRL_INT_STAT

Address offset	0x790		
Physical Address	0x4008 B790	**Instance**	AES
Description	Interrupt status		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMA_BUS_ERR	KEY_ST_WR_ERR	KEY_ST_RD_ERR	RESERVED																											DMA_IN_DONE	RESULT_AV

Bits	Field Name	Description	Type	Reset
31	DMA_BUS_ERR	This bit is set when a DMA bus error is detected during a DMA operation. The value of this register is held until it is cleared through the CTRL_INT_CLR register. Note: This error is asserted if an error is detected on the AHB master interface during a DMA operation.	RO	0
30	KEY_ST_WR_ERR	This bit is set when a write error is detected during the DMA write operation to the key store memory. The value of this register is held until it is cleared through the CTRL_INT_CLR register. Note: This error is asserted if a DMA operation does not cover a full key area or more areas are written than expected.	RO	0
29	KEY_ST_RD_ERR	This bit is set when a read error is detected during the read of a key from the key store, while copying it to the AES core. The value of this register is held until it is cleared through the CTRL_INT_CLR register. Note: This error is asserted if a key location is selected in the key store that is not available.	RO	0
28:2	RESERVED	This bit field is reserved.	RO	0x000 0000
1	DMA_IN_DONE	This read only bit returns the actual DMA data in done (irq_data_in_done) interrupt status of the DMA data in done interrupt output pin (irq_data_in_done).	RO	0
0	RESULT_AV	This read only bit returns the actual result available (irq_result_av) interrupt status of the result available interrupt output pin (irq_result_av).	RO	0

AES_CTRL_OPTIONS

Address offset	0x7F8		
Physical Address	0x4008 B7F8	**Instance**	AES
Description	Options register		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TYPE								RESERVED							AHBINTERFACE	RESERVED							SHA_256	AES_CCM	AES_GCM	AES_256	AES_128	RESERVED	HASH	AES	KEYSTORE

Bits	Field Name	Description	Type	Reset
31:24	TYPE	This field is 0x01 for the TYPE1 device.	RO	0x01
23:17	RESERVED	This bit field is reserved.	RO	0x00
16	AHBINTERFACE	AHB interface is available If this bit is 0, the EIP-120t has a TCM interface.	RO	1
15:9	RESERVED	This bit field is reserved.	RO	0x00
8	SHA_256	The HASH core supports SHA-256.	RO	1
7	AES_CCM	AES-CCM is available as a single operation.	RO	1
6	AES_GCM	AES-GCM is available as a single operation.	RO	1

Bits	Field Name	Description	Type	Reset
5	AES_256	AES core supports 256-bit keys Note: If both AES-128 and AES-256 are set to one, the AES core supports 192-bit keys as well.	RO	1
4	AES_128	AES core supports 128-bit keys.	RO	1
3	RESERVED	This bit field is reserved.	RO	0
2	HASH	HASH Core is available.	RO	1
1	AES	AES core is available.	RO	1
0	KEYSTORE	KEY STORE is available.	RO	1

AES_CTRL_VERSION

Address offset	0x7FC		
Physical Address	0x4008 B7FC	**Instance**	AES
Description	Version register		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED				MAJOR_VERSION				MINOR_VERSION				PATCH_LEVEL				EIP_NUMBER_COMPL								EIP_NUMBER							

Bits	Field Name	Description	Type	Reset
31:28	RESERVED	This bit field is reserved.	RO	0x9
27:24	MAJOR_VERSION	Major version number	RO	0x1
23:20	MINOR_VERSION	Minor version number	RO	0x1
19:16	PATCH_LEVEL	Patch level Starts at 0 at first delivery of this version	RO	0x0
15:8	EIP_NUMBER_COMPL	These bits simply contain the complement of bits [7:0] (0x87), used by a driver to ascertain that the EIP-120t register is indeed read.	RO	0x87
7:0	EIP_NUMBER	These bits encode the EIP number for the EIP-120t, this field contains the value 120 (decimal) or 0x78.	RO	0x78

22.4.2 PKA Registers

22.4.2.1 PKA Registers Mapping Summary

This section provides information on the PKA module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 22-104. PKA Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
PKA_APTR	RW	32	0x0000 0000	0x00	0x4400 4000
PKA_BPTR	RW	32	0x0000 0000	0x04	0x4400 4004
PKA_CPTR	RW	32	0x0000 0000	0x08	0x4400 4008
PKA_DPTR	RW	32	0x0000 0000	0x0C	0x4400 400C
PKA_ALENGTH	RW	32	0x0000 0000	0x10	0x4400 4010
PKA_BLENGTH	RW	32	0x0000 0000	0x14	0x4400 4014

Table 22-104. PKA Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
PKA_SHIFT	RW	32	0x0000 0000	0x18	0x4400 4018
PKA_FUNCTION	RW	32	0x0000 8000	0x1C	0x4400 401C
PKA_COMPARE	RO	32	0x0000 0001	0x20	0x4400 4020
PKA_MSW	RO	32	0x0000 8000	0x24	0x4400 4024
PKA_DIVMSW	RO	32	0x0000 8000	0x28	0x4400 4028
PKA_SEQ_CTRL	RW	32	0x0000 0100	0xC8	0x4400 40C8
PKA_OPTIONS	RO	32	0x0000 0020	0xF4	0x4400 40F4
PKA_SW_REV	RO	32	0x2142 0000	0xF8	0x4400 40F8
PKA_REVISION	RO	32	0x0151 E31C	0xFC	0x4400 40FC

22.4.2.2 PKA Register Descriptions

PKA_APTR

Address offset	0x00		
Physical Address	0x4400 4000	**Instance**	PKA
Description	PKA vector A address During execution of basic PKCP operations, this register is double buffered and can be written with a new value for the next operation; when not written, the value remains intact. During the execution of sequencer-controlled complex operations, this register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:11	RESERVED	This bit field is reserved.	RW	0x00 0000
10:0	APTR	This register specifies the location of vector A within the PKA RAM. Vectors are identified through the location of their least-significant 32-bit word. Note that bit [0] must be zero to ensure that the vector starts at an 8-byte boundary.	RW	0x000

PKA_BPTR

Address offset	0x04		
Physical Address	0x4400 4004	**Instance**	PKA
Description	PKA vector B address During execution of basic PKCP operations, this register is double buffered and can be written with a new value for the next operation; when not written, the value remains intact. During the execution of sequencer-controlled complex operations, this register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:11	RESERVED	This bit field is reserved.	RW	0x00 0000
10:0	BPTR	This register specifies the location of vector B within the PKA RAM. Vectors are identified through the location of their least-significant 32-bit word. Note that bit [0] must be zero to ensure that the vector starts at an 8-byte boundary.	RW	0x000

PKA_CPTR

Address offset	0x08		
Physical Address	0x4400 4008	**Instance**	PKA
Description	PKA vector C address During execution of basic PKCP operations, this register is double buffered and can be written with a new value for the next operation; when not written, the value remains intact. During the execution of sequencer-controlled complex operations, this register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:11	RESERVED	This bit field is reserved.	RW	0x00 0000
10:0	CPTR	This register specifies the location of vector C within the PKA RAM. Vectors are identified through the location of their least-significant 32-bit word. Note that bit [0] must be zero to ensure that the vector starts at an 8-byte boundary.	RW	0x000

PKA_DPTR

Address offset	0x0C		
Physical Address	0x4400 400C	**Instance**	PKA
Description	PKA vector D address During execution of basic PKCP operations, this register is double buffered and can be written with a new value for the next operation; when not written, the value remains intact. During the execution of sequencer-controlled complex operations, this register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:11	RESERVED	This bit field is reserved.	RW	0x00 0000
10:0	DPTR	This register specifies the location of vector D within the PKA RAM. Vectors are identified through the location of their least-significant 32-bit word. Note that bit [0] must be zero to ensure that the vector starts at an 8-byte boundary.	RW	0x000

PKA_ALENGTH

Address offset	0x10		
Physical Address	0x4400 4010	**Instance**	PKA
Description	PKA vector A length During execution of basic PKCP operations, this register is double buffered and can be written with a new value for the next operation; when not written, the value remains intact. During the execution of sequencer-controlled complex operations, this register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:9	RESERVED	This bit field is reserved.	RW	0x00 0000
8:0	ALENGTH	This register specifies the length (in 32-bit words) of Vector A.	RW	0x000

PKA_BLENGTH

Address offset	0x14		
Physical Address	0x4400 4014	**Instance**	PKA
Description	PKA vector B length During execution of basic PKCP operations, this register is double buffered and can be written with a new value for the next operation; when not written, the value remains intact. During the execution of sequencer-controlled complex operations, this register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:9	RESERVED	This bit field is reserved.	RW	0x00 0000
8:0	BLENGTH	This register specifies the length (in 32-bit words) of Vector B.	RW	0x000

PKA_SHIFT

Address offset	0x18		
Physical Address	0x4400 4018	**Instance**	PKA
Description	PKA bit shift value For basic PKCP operations, modifying the contents of this register is made impossible while the operation is being performed. For the ExpMod-variable and ExpMod-CRT operations, this register is used to indicate the number of odd powers to use (directly as a value in the range 1-16). For the ModInv and ECC operations, this register is used to hold a completion code.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:5	RESERVED	This bit field is reserved.	RW	0x000 0000
4:0	NUM_BITS_TO_SHIFT	This register specifies the number of bits to shift the input vector (in the range 0-31) during a Rshift or Lshift operation.	RW	0x00

PKA_FUNCTION

Address offset	0x1C		
Physical Address	0x4400 401C	**Instance**	PKA
Description	PKA function This register contains the control bits to start basic PKCP as well as complex sequencer operations. The run bit can be used to poll for the completion of the operation. Modifying bits [11:0] is made impossible during the execution of a basic PKCP operation. During the execution of sequencer-controlled complex operations, this register is modified; the run and stall result bits are set to zero at the conclusion, but other bits are undefined. Attention: Continuously reading this register to poll the run bit is not allowed when executing complex sequencer operations (the sequencer cannot access the PKCP when this is done). Leave at least one sysclk cycle between poll operations.		
Type	RW		

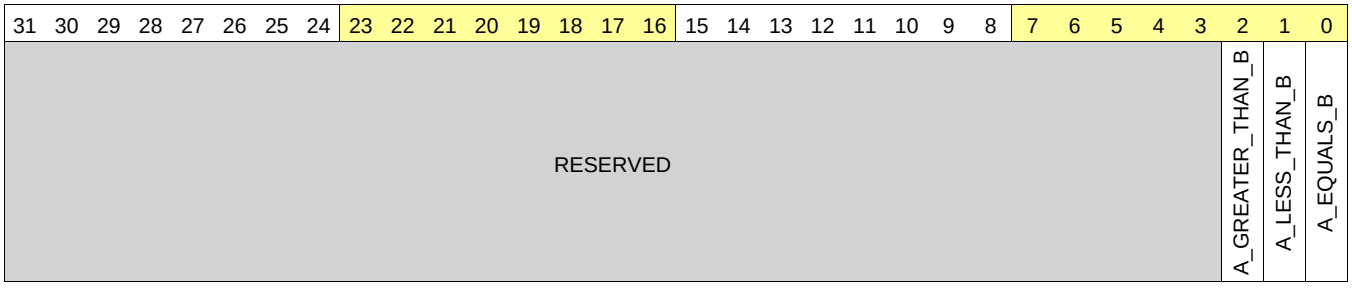
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED							STALL_RESULT	RESERVED								RUN	SEQUENCER_OPERATIONS			COPY	COMPARE	MODULO	DIVIDE	LSHIFT	RSHIFT	SUBTRACT	ADD	MS_ONE	RESERVED	ADDSUB	MULTIPLY

Bits	Field Name	Description	Type	Reset
31:25	RESERVED	This bit field is reserved.	RW	0x00
24	STALL_RESULT	When written with a 1b, updating of the PKA_COMPARE, PKA_MSW and PKA_DIVMSW registers, as well as resetting the run bit is stalled beyond the point that a running operation is actually finished. Use this to allow software enough time to read results from a previous operation when the newly started operation is known to take only a short amount of time. If a result is waiting, the result registers is updated and the run bit is reset in the clock cycle following writing the stall result bit back to 0b. The Stall result function may only be used for basic PKCP operations.	RW	0
23:16	RESERVED	This bit field is reserved.	RW	0x00
15	RUN	The host sets this bit to instruct the PKA module to begin processing the basic PKCP or complex sequencer operation. This bit is reset low automatically when the operation is complete. The complement of this bit is output as interrupts[1]. After a reset, the run bit is always set to 1b. Depending on the option, program ROM or program RAM, the following applies: Program ROM - The first sequencer instruction sets the bit to 0b. This is done immediately after the hardware reset is released. Program RAM - The sequencer must set the bit to 0b. As a valid firmware may not have been loaded, the sequencer is held in software reset after the hardware reset is released (the reset bit in PKA_SEQ_CRTL is set to 1b). After the FW image is loaded and the Reset bit is cleared, the sequencer starts to execute the FW. The first instruction clears the run bit. In both cases a few clock cycles are needed before the first instruction is executed and the run bit state has been propagated.	RW	1

Bits	Field Name	Description	Type	Reset
14:12	SEQUENCER_OPERATIONS	These bits select the complex sequencer operation to perform: 000b: None 001b: ExpMod-CRT 010b: ExpMod-ACT4 (compatible with EIP2315) 011b: ECC-ADD (if available in firmware, otherwise reserved) 100b: ExpMod-ACT2 (compatible with EIP2316) 101b: ECC-MUL (if available in firmware, otherwise reserved) 110b: ExpMod-variable 111b: ModInv (if available in firmware, otherwise reserved) The encoding of these operations is determined by sequencer firmware.	RW	0x0
11	COPY	Perform copy operation	RW	0
10	COMPARE	Perform compare operation	RW	0
9	MODULO	Perform modulo operation	RW	0
8	DIVIDE	Perform divide operation	RW	0
7	LSHIFT	Perform left shift operation	RW	0
6	RSHIFT	Perform right shift operation	RW	0
5	SUBTRACT	Perform subtract operation	RW	0
4	ADD	Perform add operation	RW	0
3	MS_ONE	Loads the location of the Most Significant one bit within the result word indicated in the PKA_MSW register into bits [4:0] of the PKA_DIVMSW register - can only be used with basic PKCP operations, except for Divide, Modulo and Compare.	RW	0
2	RESERVED	This bit field is reserved.	RW	0
1	ADDSUB	Perform combined add/subtract operation	RW	0
0	MULTIPLY	Perform multiply operation	RW	0

PKA_COMPARE

Address offset	0x20		
Physical Address	0x4400 4020	**Instance**	PKA
Description	PKA compare result This register provides the result of a basic PKCP compare operation. It is updated when the run bit in the PKA_FUNCTION register is reset at the end of that operation. Status after a complex sequencer operation is unknown		
Type	RO		



Bits	Field Name	Description	Type	Reset
31:3	RESERVED	This bit field is reserved.	RO	0x0000 0000
2	A_GREATER_THAN_B	Vector_A is greater than Vector_B	RO	0
1	A_LESS_THAN_B	Vector_A is less than Vector_B	RO	0
0	A_EQUALS_B	Vector_A is equal to Vector_B	RO	1

PKA_MSW

Address offset	0x24		
Physical Address	0x4400 4024	**Instance**	PKA
Description	PKA most-significant-word of result vector This register indicates the (word) address in the PKA RAM where the most significant nonzero 32-bit word of the result is stored. Should be ignored for modulo operations. For basic PKCP operations, this register is updated when the run bit in the PKA_FUNCTION register is reset at the end of the operation. For the complex-sequencer controlled operations, updating of the final value matching the actual result is done near the end of the operation; note that the result is only meaningful if no errors were detected and that for ECC operations, the PKA_MSW register will provide information for the x-coordinate of the result point only.		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15	RESULT_IS_ZERO	The result vector is all zeroes, ignore the address returned in bits [10:0]	RO	1
14:11	RESERVED	This bit field is reserved.	RO	0x0
10:0	MSW_ADDRESS	Address of the most-significant nonzero 32-bit word of the result vector in PKA RAM	RO	0x000

PKA_DIVMSW

Address offset	0x28		
Physical Address	0x4400 4028	**Instance**	PKA
Description	PKA most-significant-word of divide remainder This register indicates the (32-bit word) address in the PKA RAM where the most significant nonzero 32-bit word of the remainder result for the basic divide and modulo operations is stored. Bits [4:0] are loaded with the bit number of the most-significant nonzero bit in the most-significant nonzero word when MS one control bit is set. For divide, modulo, and MS one reporting, this register is updated when the RUN bit in the PKA_FUNCTION register is reset at the end of the operation. For the complex sequencer controlled operations, updating of bits [4:0] of this register with the most-significant bit location of the actual result is done near the end of the operation. The result is meaningful only if no errors were detected and that for ECC operations; the PKA_DIVMSW register provides information for the x-coordinate of the result point only.		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:16	RESERVED	This bit field is reserved.	RO	0x0000
15	RESULT_IS_ZERO	The result vector is all zeroes, ignore the address returned in bits [10:0]	RO	1
14:11	RESERVED	This bit field is reserved.	RO	0x0

Bits	Field Name	Description	Type	Reset
10:0	MSW_ADDRESS	Address of the most significant nonzero 32-bit word of the remainder result vector in PKA RAM	RO	0x000

PKA_SEQ_CTRL

Address offset	0xC8		
Physical Address	0x4400 40C8	**Instance**	PKA
Description	PKA sequencer control and status register The sequencer is interfaced with the outside world through a single control and status register. With the exception of bit [31], the actual use of bits in the separate sub-fields of this register is determined by the sequencer firmware. This register need only be accessed when the sequencer program is stored in RAM. The reset value of the RESTE bit depends upon the option chosen for sequencer program storage.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET	RESERVED															SEQUENCER_STATUS								SW_CONTROL_STATUS							

Bits	Field Name	Description	Type	Reset
31	RESET	Option program ROM: Reset value = 0. Read/Write, reset value 0b (ZERO). Writing 1b resets the sequencer, write to 0b to restart operations again. As the reset value is 0b, the sequencer will automatically start operations executing from program ROM. This bit should always be written with zero and ignored when reading this register. Option Program RAM: Reset value =1. Read/Write, reset value 1b (ONE). When 1b, the sequencer is held in a reset state and the PKA_PROGRAM area is accessible for loading the sequencer program (while the PKA_DATA_RAM is inaccessible), write to 0b to (re)start sequencer operations and disable PKA_PROGRAM area accessibility (also enables the PKA_DATA_RAM accesses). Resetting the sequencer (in order to load other firmware) should only be done when the PKA Engine is not performing any operations (i.e. the run bit in the PKA_FUNCTION register should be zero).	RW	0
30:16	RESERVED	This bit field is reserved.	RW	0x0000
15:8	SEQUENCER_STATUS	These read-only bits can be used by the sequencer to communicate status to the outside world. Bit [8] is also used as sequencer interrupt, with the complement of this bit ORed into the run bit in PKA_FUNCTION. This field should always be written with zeroes and ignored when reading this register.	RO	0x01
7:0	SW_CONTROL_STATUS	These bits can be used by software to trigger sequencer operations. External logic can set these bits by writing 1b, cannot reset them by writing 0b. The sequencer can reset these bits by writing 0b, cannot set them by writing 1b. Setting the run bit in PKA_FUNCTION together with a nonzero sequencer operations field automatically sets bit [0] here. This field should always be written with zeroes and ignored when reading this register.	RW	0x00

PKA_OPTIONS

Address offset	0xF4		
Physical Address	0x4400 40F4	**Instance**	PKA
Description	PKA hardware options register This register provides the host with a means to determine the hardware configuration implemented in this PKA engine, focused on options that have an effect on software interacting with the module. Note: (32 x (1st LNME nr. of PEs + 1st LNME FIFO RAM depth - 10)) equals the maximum modulus vector length (in bits) that can be handled by the modular exponentiation and ECC operations executed on a PKA engine that includes an LNME.		
Type	RO		

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
FIRST_LNME_FIFO_DEPTH	RESERVED / FIRST_LNME_NR_OF_PES	RESERVED / MMM3A / INT_MASKING / PROTECTION_OPTION	PROGRAM_RAM / SEQUENCER_CONFIGURATION / LNME_CONFIGURATION / PKCP_CONFIGURATION

Bits	Field Name	Description	Type	Reset
31:24	FIRST_LNME_FIFO_DEPTH	Number of words in the first LNME's FIFO RAM Should be ignored if LNME configuration is 0. The contents of this field indicate the actual depth as selected by the LNME FIFO RAM size strap input, fifo_size_sel. Note: Reset value is undefined	RO	0x00
23:22	RESERVED	This bit field is reserved.	RO	0x0
21:16	FIRST_LNME_NR_OF_PES	Number of processing elements in the pipeline of the first LNME Should be ignored if LNME configuration is 0. Note: Reset value is undefined.	RO	0x00
15:13	RESERVED	This bit field is reserved.	RO	0x0
12	MMM3A	Reserved for a future functional extension to the LNME Always 0b	RO	0
11	INT_MASKING	Value 0b indicates that the main interrupt output (bit [1] of the interrupts output bus) is the direct complement of the run bit in the PKA_CONTROL register, value 1b indicates that interrupt masking logic is present for this output. Note: Reset value is undefined	RO	0
10:8	PROTECTION_OPTION	Value 0 indicates no additional protection against side channel attacks, value 1 indicates the SCAP option, value 3 indicates the PROT option; other values are reserved. Note: Reset value is undefined	RO	0x0
7	PROGRAM_RAM	Value 1b indicates sequencer program storage in RAM, value 0b in ROM. Note: Reset value is undefined	RO	0
6:5	SEQUENCER_CONFIGURATION	Value 1 indicates a standard sequencer; other values are reserved.	RO	0x1
4:2	LNME_CONFIGURATION	Value 0 indicates NO LNME, value 1 indicates one standard LNME (with alpha = 32, beta = 8); other values reserved. Note: Reset value is undefined	RO	0x0
1:0	PKCP_CONFIGURATION	Value 1 indicates a PKCP with a 16x16 multiplier, value 2 indicates a PKCP with a 32x32 multiplier, other values reserved. Note: Reset value is undefined.	RO	0x0

PKA_SW_REV

Address offset	0xF8		
Physical Address	0x4400 40F8	**Instance**	PKA
Description	PKA firmware revision and capabilities register This register allows the host access to the internal firmware revision number of the PKA Engine for software driver matching and diagnostic purposes. This register also contains a field that encodes the capabilities of the embedded firmware. The PKA_SW_REV register is written by the firmware within a few clock cycles after starting up that firmware. The hardware reset value is zero, indicating that the information has not been written yet.		
Type	RO		

31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
FW_CAPABILITIES	MAJOR_FW_REVISION	MINOR_FW_REVISION	FW_PATCH_LEVEL	RESERVED	

Bits	Field Name	Description	Type	Reset
31:28	FW_CAPABILITIES	4-bit binary encoding for the functionality implemented in the firmware. Value 0 indicates basic ModExp with/without CRT. Value 1 adds Modular Inversion, value 2 adds Modular Inversion and ECC operations. Values 3-15 are reserved.	RO	0x2
27:24	MAJOR_FW_REVISION	4-bit binary encoding of the major firmware revision number	RO	0x1
23:20	MINOR_FW_REVISION	4-bit binary encoding of the minor firmware revision number	RO	0x4
19:16	FW_PATCH_LEVEL	4-bit binary encoding of the firmware patch level, initial release will carry value zero Patches are used to remove bugs without changing the functionality or interface of a module.	RO	0x2
15:0	RESERVED	This bit field is reserved.	RO	0x0000

PKA_REVISION

Address offset	0xFC		
Physical Address	0x4400 40FC	**Instance**	PKA
Description	PKA hardware revision register This register allows the host access to the hardware revision number of the PKA engine for software driver matching and diagnostic purposes. It is always located at the highest address in the access space of the module and contains an encoding of the EIP number (with its complement as signature) for recognition of the hardware module.		
Type	RO		

31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
RESERVED	MAJOR_HW_REVISION	MINOR_HW_REVISION	HW_PATCH_LEVEL	COMPLEMENT_OF_BASIC_EIP_NUMBER	BASIC_EIP_NUMBER

Bits	Field Name	Description	Type	Reset
31:28	RESERVED	This bit field is reserved.	RO	0x0
27:24	MAJOR_HW_REVISION	4-bit binary encoding of the major hardware revision number	RO	0x1
23:20	MINOR_HW_REVISION	4-bit binary encoding of the minor hardware revision number	RO	0x5

Bits	Field Name	Description	Type	Reset
19:16	HW_PATCH_LEVEL	4-bit binary encoding of the hardware patch level, initial release will carry value zero Patches are used to remove bugs without changing the functionality or interface of a module.	RO	0x1
15:8	COMPLEMENT_OF_BASIC_EIP_NUMBER	Bit-by-bit logic complement of bits [7:0], EIP-28 gives 0xE3	RO	0xE3
7:0	BASIC_EIP_NUMBER	8-bit binary encoding of the EIP number, EIP-28 gives 0x1C	RO	0x1C

Radio

The **RF Core** controls the analog and digital radio modules. In addition, it provides an interface between the microcontroller unit (MCU) and the radio which makes it possible to issue commands, read status, and automate and sequence radio events.

Topic **Page**

23.1 RF Core

The FSM submodule controls the RF transceiver state, the transmitter and receiver FIFOs (TX FIFO and RX FIFO, respectively), and most of the dynamically controlled analog signals, such as power up and power down of analog modules. The FSM provides the correct sequencing of events (such as performing an FS calibration before enabling the receiver or transmitter). Also, it provides step-by-step processing of incoming frames from the demodulator: reading the frame length, counting the number of bytes received, checking the FCS, and finally, optionally handling automatic transmission of ACK frames after successful frame reception. It performs similar tasks in TX, including performing an optional CCA before transmission and automatically going to RX after the end of transmission to receive an ACK frame. Finally, the FSM controls the transfer of data between the modulator or demodulator and the TX FIFO or RX FIFO in RAM.

The modulator transforms raw data into I/Q signals to the transmitter digital-to-analog converters (DACs). This is done in compliance with the IEEE 802.15.4 standard.

The demodulator retrieves the over-the-air data from the received signal.

The amplitude information from the demodulator is used by the automatic gain control (AGC). The AGC adjusts the gain of the analog LNA so that the signal level within the receiver is approximately constant.

The frame filtering and source matching supports the FSM in the RF Core by performing all operations needed to do frame filtering and source address matching, as defined by IEEE 802.15.4.

The frequency synthesizer (FS) generates the carrier wave for the RF signal.

The command strobe processor (CSP) processes all commands issued by the CPU. It also has a short program memory of 24 bytes, making it possible to automate CSMA-CA algorithms.

The radio RAM holds a FIFO for transmit data (TX FIFO) and a FIFO for receive data (RX FIFO). Both FIFOs are 128 bytes long. In addition, the RAM holds parameters for frame filtering and source matching; 128 bytes are reserved for each parameter.

The MAC timer is used for timing of radio events and to capture time stamps of incoming packets. This timer keeps counting even in power modes PM1 and PM2.

23.1.1 Interrupts

The radio is associated with two interrupt vectors on the CPU. These are the RFERR interrupt (interrupt 142) and the RF interrupt (interrupt 141) with the following functions.

- RFERR: Error situations in the radio are signaled using this interrupt.
- RF: Interrupts coming from normal operation are signaled using this interrupt.

23.1.2 Interrupt Registers

Interrupts are enabled through the nested vectored interrupt controller (NVIC) Interrupt Set Enable n (ENn) register and prioritized with the NVIC Interrupt Priority n (PRIn) registers. For more information see Chapter 5, *Interrupts.*

The two interrupts generated by the RF Core are a combination of several sources within the RF Core. Each individual source has its own enable and interrupt flags in the RF Core. Flags are found in the **RFIRQF0**, **RFIRQF1**, and **RFERRF** registers. Interrupt masks are found in the **RFIRQM0**, **RFIRQM1**, and **RFERRM** registers.

The interrupt-enable bits in the mask registers are used to enable individual interrupt sources for the two RF interrupts. Masking an interrupt source does not affect the updating of the status in the flag registers.

Due to the use of individual interrupt masks in the RF Core, the interrupts coming from the RF Core have 2-layered masking. Take care when processing these interrupts. The procedure is described as follows:

To clear an interrupt from the RF Core, one must clear two flags, both the flag set in RF Core and the one set in NVIC. If a flag is cleared in the RF Core and other unmasked flags are standing, another interrupt is generated.

Table 23-1. Interrupt Registers Register Map

Offset	Name	Type	Reset	Description	Link
0x34	RFIRQF0	R/W	0x0000 0000	RF interrupt flags 0	RFCORE_SFR_RFIRQF0
0x30	RFIRQF1	R/W	0x0000 0000	RF interrupt flags 1	RFCORE_SFR_RFIRQF1
0x2C	RFERRF	R/W	0x0000 0000	RF error interrupt flags	RFCORE_SFR_RFERRF
0x8C	RFIRQM0	R/W	0x0000 0000	RF interrupt masks 0	RFCORE_XREG_RFIRQM0
0x90	RFIRQM1	R/W	0x0000 0000	RF interrupt masks 1	RFCORE_XREG_RFIRQM1
0x94	RFERRM		0x0000 0000	RF error interrupt masks	RFCORE_XREG_RFERRM
0x28	RFDATA	R/W	0x0000 0000	RF data	RFCORE_SFR_RFDATA

23.2 FIFO Access

The TX FIFO and RX FIFO may be accessed though the **RFDATA** register (0x4008 8828). Data is written to the TX FIFO when writing to the **RFDATA** register. Data is read from the RX FIFO when the **RFDATA** register is read.

The **RXFIFOCNT** and **TXFIFOCNT** registers provide information on the amount of data in the FIFOs. The FIFO contents can be cleared by issuing ISFLUSHRX (or SFLUSHRX) and ISFLUSHTX (or SFLUSHTX).

23.3 DMA

It is possible to use direct memory access (DMA) to move data between memory and the radio. The DMA controller is described in Chapter 10. For a detailed description on how to set up and use DMA transfers, see this section.

To support the DMA controller, one DMA trigger is associated with the radio, the RF_Core trigger. The DMA trigger is activated by two events. The first event to cause a RF_Core trigger occurs when the first data is present in the RX FIFO (that is, when the RX FIFO goes from the empty state to a nonempty state). The second event that causes a RF_Core trigger occurs when data is read from the RX FIFO (through **RFDATA**) and there is still more data available in the RX FIFO.

23.4 Memory Map

The RF Core configuration and status registers are located at addresses from 0x4008 8600 to 0x4008 8844. Configuration registers, RX FIFO, and TX FIFO are all preserved during sleep modes.

23.4.1 RX FIFO

The RX FIFO memory area is located at address 0x4008 8000 and is 128 bytes long. Although this memory area is intended for the RX FIFO, it is not protected in any way, so it is still accessible in the memory. Normally, the contents of the RX FIFO are manipulated only by the designated instructions. The RX FIFO can contain more than one frame at a time.

23.4.2 TX FIFO

The TX FIFO memory area is located at address 0x4008 8200 and is 128 bytes long. Although this memory area is intended for the TX FIFO, it is not protected in any way, so it is still accessible in the memory. Normally, the contents of the TX FIFO are manipulated only by the designated instructions. The TX FIFO can only contain one frame at a time.

23.4.3 Frame-Filtering and Source-Matching Memory Map

The frame-filtering and source-address-matching functions use a block of the RF Core RAM to store local-address information and source-matching configuration and results; this is located in the area 0x4008 8580 to 0x4008 85D8. Table 23-2 describes this memory space. Values that do not fill an entire byte or word are in the least-significant part of the byte or word. The values in these registers are unknown after reset. However, the values are retained during power modes.

Table 23-2. Frame Filtering and Source Matching Memory Map

ADDRESS	REGISTER/VARIABLE	ENDIAN	DESCRIPTION
RESERVED			
0x4008 85D8	Temporary storage		Memory space used for temporary storage of variables, 10 bytes
LOCAL ADDRESS INFORMATION			
0x4008 85D4	SHORT_ADDR1		The short address used during destination address filtering, SHORT_ADDR[15:8]
0x4008 85D0	SHORT_ADDR0		The short address used during destination address filtering, SHORT_ADDR[7:0]
0x4008 85CC	PAN_ID1		The PAN ID used during destination address filtering, PAN_ID[15:8]
0x4008 85C8	PAN_ID0		The PAN ID used during destination address filtering, PAN_ID[7:0]
0x4008 85C4	EXT_ADDR7		The IEEE extended address used during destination address filtering, EXT_ADDR[63:56]
0x4008 85C0	EXT_ADDR6		The IEEE extended address used during destination address filtering, EXT_ADDR[55:48]
0x4008 85BC	EXT_ADDR5		The IEEE extended address used during destination address filtering, EXT_ADDR[47:40]
0x4008 85B8	EXT_ADDR4		The IEEE extended address used during destination address filtering, EXT_ADDR[39:32]
0x4008 85B4	EXT_ADDR3		The IEEE extended address used during destination address filtering, EXT_ADDR[31:24]
0x4008 85B0	EXT_ADDR2		The IEEE extended address used during destination address filtering, EXT_ADDR[23:16]
0x4008 85AC	EXT_ADDR1		The IEEE extended address used during destination address filtering, EXT_ADDR[15:8]
0x4008 85A8	EXT_ADDR0		The IEEE extended address used during destination address filtering, EXT_ADDR[7:0]
SOURCE ADDRESS MATCHING CONTROL			
0x4008 85A4	SRCSHORTPENDEN2		8 MSBs of the 24-bit mask that enables or disables automatic pending for each of the 24 short addresses
0x4008 85A0	SRCSHORTPENDEN1		
0x4008 859C	SRCSHORTPENDEN0		8 LSBs of the 24-bit mask that enables or disables automatic pending for each of the 24 short addresses
0x4008 8598	SRCEXTPENDEN2		8 MSBs of the 24-bit mask that enables or disables automatic pending for each of the 12 extended addresses. Entry n is mapped to **SRCEXTPENDEN[2n]**. All **SRCEXTPENDEN[2n + 1]** bits are don't care.
0x4008 8594	SRCEXTPENDEN1		
0x4008 8590	SRCEXTPENDEN0		8 LSBs of the 24-bit mask that enables or disables automatic pending for each of the 12 extended addresses. Entry n is mapped to **SRCEXTPENDEN[2n]**. All **SRCEXTPENDEN[2n + 1]** bits are don't care.
SOURCE ADDRESS MATCHING RESULT			
0x4008 858C	SRCRESINDEX		The bit index of the least-significant 1 in **SRCRESMASK**, or 0x3F when there is no source match. On a match, bit [5] is 0 when the match is on a short address and 1 when it is on an extended address. On a match, bit [6] is 1 when the conditions for automatic pending bit in acknowledgment have been met (see the description of the **AUTOPEND** bit of the **SRCMATCH** register). The bit gives no indication of whether or not the acknowledgment actually is transmitted, and does not consider the **PENDING_OR** register bit and the SACK/SACKPEND/SNACK strobes.

Table 23-2. Frame Filtering and Source Matching Memory Map (continued)

ADDRESS	REGISTER/VARIABLE		ENDIAN		DESCRIPTION
0x4008 8588	SRCRESMASK2				24-bit mask that indicates source address match for each individual entry in the source address table
0x4008 8584	SRCRESMASK1				Short address matching. When there is a match on entry panid_n + short_n, bit n is set in **SRCRESMASK**.
0x4008 8580	SRCRESMASK0				Extended address matching. When there is a match on entry ext_n, bits 2n and 2n + 1 are set in **SRCRESMASK**.
SOURCE ADDRESS TABLE					
0x4008 857C	short_23	ext_11	LE	LE	Two individual short-address entries (combination of 16-bit PAN ID and 16-bit short address) or one extended address entry
0x4008 8578	short_23				
0x4008 8574	panid_23		LE		
0x4008 8570	panid_23				
0x4008 856C	short_22		LE		
0x4008 8568	short_22				
0x4008 8564	panid_22		LE		
0x4008 8560	panid_22				
...	...	...	...	...	...
0x4008 843C	short_03	ext_01	LE	LE	Two individual short address entries (combination of 16-bit PAN ID and 16-bit short address) or one extended address entry
0x4008 8438	short_03				
0x4008 8434	panid_03		LE		
0x4008 8430	panid_03				
0x4008 842C	short_02		LE		
0x4008 8428	short_02				
0x4008 8424	panid_02		LE		
0x4008 8420	panid_02				
0x4008 841C	short_01	ext_00	LE	LE	Two individual short address entries (combination of 16-bit PAN ID and 16-bit short address) or one extended address entry
0x4008 8418	short_01				
0x4008 8414	panid_01		LE		
0x4008 8410	panid_01				
0x4008 840C	short_00		LE		
0x4008 8408	short_00				
0x4008 8404	panid_00		LE		
0x4008 8400	panid_00				

23.5 Frequency and Channel Programming

The carrier frequency is set by programming the 7-bit frequency word in the **FREQ[6:0]** bits of the **FREQCTRL** register. Changes take effect after the next recalibration. Carrier frequencies in the range from 2394 to 2507 MHz are supported. The carrier frequency f_C, in MHz, is given by f_C = (2394 + **FREQCTRL.FREQ[6:0]**) MHz, and is programmable in 1-MHz steps.

IEEE 802.15.4-2006 specifies 16 channels within the 2.4-GHz band. These channels are numbered 11 through 26 and are 5 MHz apart. The RF frequency of channel k is given by Equation 1.

$$f_C = 2405 + 5(k - 11) \text{ [MHz]} \quad k \in [11, 26] \tag{1}$$

For operation in channel k, the **FREQCTRL.FREQ** register should therefore be set to **FREQCTRL.FREQ** = 11 + 5 (k – 11).

23.6 IEEE 802.15.4-2006 Modulation Format

This section is meant as an introduction to the 2.4-GHz direct-sequence spread-spectrum (DSSS) RF modulation format defined in IEEE 802.15.4-2006. For a complete description, see the standard document.

Figure 23-1 shows the modulation and spreading functions at the block level. Each byte is divided into two symbols of 4 bits each. The least-significant symbol is transmitted first. For multibyte fields, the least-significant byte is transmitted first, except for security-related fields, where the most-significant byte is transmitted first.

Each symbol is mapped to 1 of 16 pseudo-random sequences, 32 chips each. Table 23-3 shows the symbol-to-chip mapping. The chip sequence is then transmitted at 2 Mchips/s, with the least-significant chip (C_0) transmitted first for each symbol. The transmitted bit stream and the chip sequences are observable on GPIO pins. For details on how to configure the GPIO to do this, see Chapter 9, *GPIOs.*

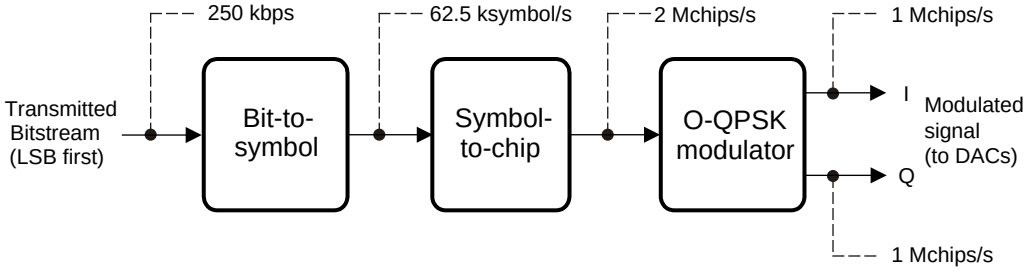


Figure 23-1. Modulation

Table 23-3. IEEE 802.15.4-2006 Symbol-to-Chip Mapping

Symbol	Chip Sequence (C0 through C31)
0	1 1 0 1 1 0 0 1 1 1 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0 1 1 1 0
1	1 1 1 0 1 1 0 1 1 0 0 1 1 1 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0
2	0 0 1 0 1 1 1 0 1 1 0 1 1 0 0 1 1 1 0 0 0 0 1 1 0 1 0 1 0 0 1 0
3	0 0 1 0 0 0 1 0 1 1 1 0 1 1 0 1 1 0 0 1 1 1 0 0 0 0 1 1 0 1 0 1
4	0 1 0 1 0 0 1 0 0 0 1 0 1 1 1 0 1 1 0 1 1 0 0 1 1 1 0 0 0 0 1 1
5	0 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0 1 1 1 0 1 1 0 1 1 0 0 1 1 1 0 0
6	1 1 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0 1 1 1 0 1 1 0 1 1 0 0 1
7	1 0 0 1 1 1 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0 1 1 1 0 1 1 0 1
8	1 0 0 0 1 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 1 1 1 0 1 1 1 1 0 1 1
9	1 0 1 1 1 0 0 0 1 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 1 1 1 0 1 1 1
10	0 1 1 1 1 0 1 1 1 0 0 0 1 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 1 1 1
11	0 1 1 1 0 1 1 1 1 0 1 1 1 0 0 0 1 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0
12	0 0 0 0 0 1 1 1 0 1 1 1 1 0 1 1 1 0 0 0 1 1 0 0 1 0 0 1 0 1 1 0
13	0 1 1 0 0 0 0 0 0 1 1 1 0 1 1 1 1 0 1 1 1 0 0 0 1 1 0 0 1 0 0 1
14	1 0 0 1 0 1 1 0 0 0 0 0 0 0 1 1 1 0 1 1 1 1 0 1 1 1 0 0 0 1 1 0 0
15	1 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 1 1 1 0 1 1 1 1 0 1 1 1 0 0 0

The modulation format is offset – quadrature phase shift keying (O-QPSK) with half-sine chip shaping. This is equivalent to minimum shift keying (MSK) modulation. Each chip is shaped as a half-sine, transmitted alternately in the I and Q channels with one-half chip-period offset. Figure 23-2 shows the zero-symbol chip sequence.

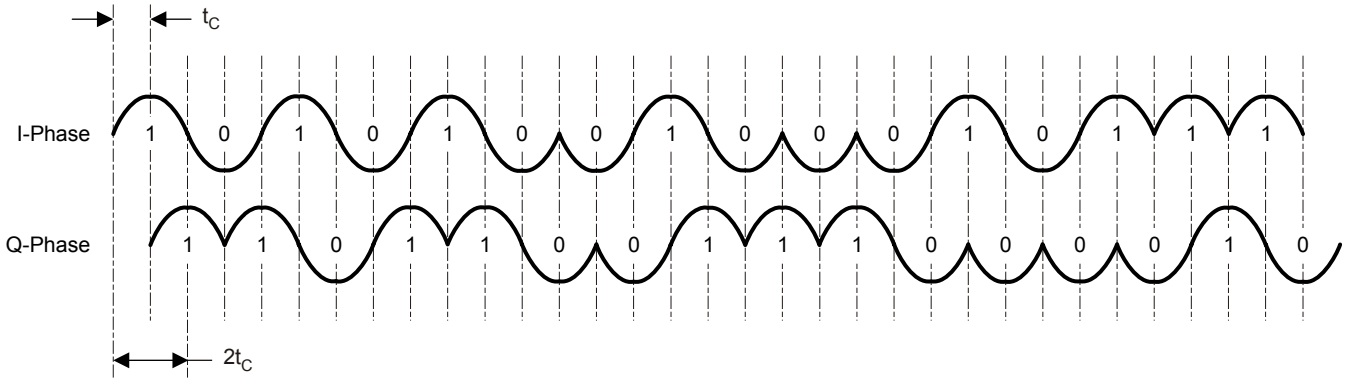


Figure 23-2. I and Q Phases When Transmitting a Zero-Symbol Chip Sequence, t_C = 0.5 µs

23.7 IEEE 802.15.4-2006 Frame Format

This section gives a brief summary of the IEEE 802.15.4 frame format. The radio has built-in support for processing of parts of the frame. This is described in the following sections.

Figure 23-3 shows a schematic view of the IEEE 802.15.4 frame format. Similar figures describing specific frame formats (data frames, beacon frames, acknowledgment frames, and MAC command frames) are included in the IEEE 802.15.4 standard document.

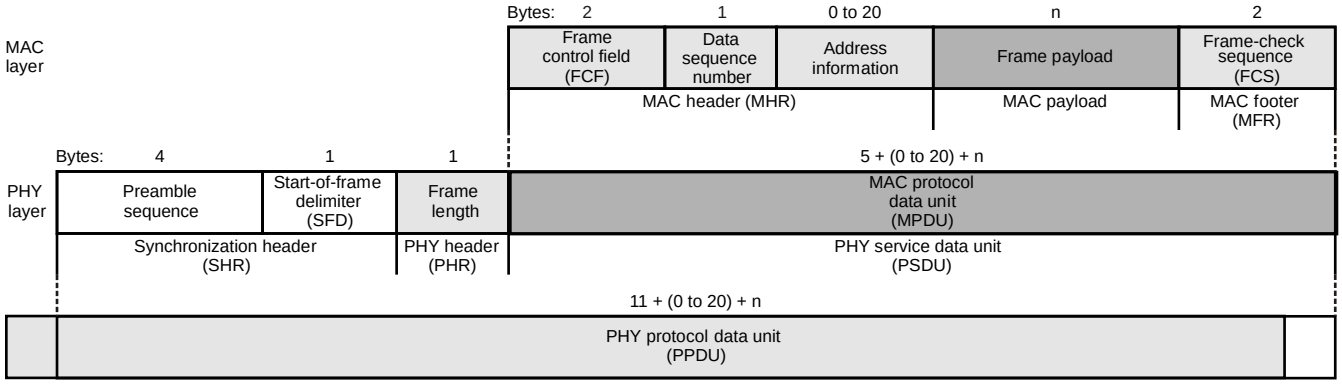


Figure 23-3. Schematic View of the IEEE 802.15.4 Frame Format

23.7.1 PHY Layer

Synchronization Header

The synchronization header (SHR) consists of the preamble sequence followed by the start-of-frame delimiter (SFD). In the IEEE 802.15.4 specification, the preamble sequence is defined to be 4 bytes of 0x00. The SFD is 1 byte with value 0xA7.

PHY Header

The PHY header consists only of the frame-length field. The frame-length field defines the number of bytes in the MAC protocol data unit (MPDU). The value of the frame-length field does not include the frame-length field itself. It does, however, include the frame-check sequence (FCS), even if this is inserted automatically by the hardware.

The frame-length field is 7 bits long and has a maximum value of 127. The most-significant bit (MSB) in the frame-length field is reserved, and should always be set to 0

PHY Service Data Unit
The PHY service data unit (PSDU) contains the MPDU. The function of the MAC layer is to generate and interpret the MPDU, and the radio has built-in support for processing of some of the MPDU subfields.

23.7.2 MAC Layer

The FCF, data sequence number, and address information follow the frame-length field as shown in Figure 23-3. Together with the MAC data payload and frame check sequence, they form the MPDU. The format of the FCF is shown in Figure 23-4. For full details, see the IEEE 802.15.4 specification.

Bits: 0–2	3	4	5	6	7–9	10–11	12–13	14–15
Frame type	Security enabled	Frame pending	Acknowledge request	Intra PAN	Reserved	Destination addressing mode	Reserved	Source addressing mode

Figure 23-4. Format of the Frame Control Field (FCF)

Frame-Check Sequence
A 2-byte FCS follows the last MAC payload byte as shown in Figure 23-3. The FCS is calculated over the MPDU; that is, the frame-length field is not part of the FCS.

The FCS polynomial defined in IEEE 802.15.4 is

$G(s) = x^{16} + x^{12} + x^5 + 1$

The radio supports automatic calculation and verification of the FCS. See Section 23.8.10 for details.

23.8 Transmit Mode

This section describes how to control the transmitter, how to control the integrated frame processing, and how to use the TX FIFO.

23.8.1 TX Control

The radio has many built-in features for frame processing and status reporting. The radio provides features that precisely control the timing of outgoing frames. Such precise control of the timing of the outgoing frames is important in an IEEE 802.15.4/ZigBee® system, because there are strict timing requirements to such systems.

Frame transmission is started by the following actions:

- The STXON command strobe: does not update the **SAMPLED_CCA** signal.
- The STXONCCA command strobe, provided that the CCA signal is high.
  - Aborts ongoing transmission and reception and forces a TX calibration followed by transmission
  - Updates the **SAMPLED_CCA** signal

Section 23.8.12 describes the clear channel assessment in detail.

Frame transmission is aborted by the following command actions:

- The SRXON command strobe: aborts ongoing transmission and forces an RX calibration.
- The SRFOFF command strobe: aborts ongoing transmission and reception and forces the FSM to the IDLE state.
- The STXON command strobe: aborts ongoing transmission and forces an RX calibration.

To enable the receiver after transmission with STXON, set the **SET_RXENMASK_ON_TX** bit of the **FRMCTRL1** register. This sets bit [6] in the **RXENABLE** register when STXON executes. When transmitting with STXONCCA, the receiver is on before the transmission and is turned back on afterward (unless the **RXENABLE** registers are cleared in the meantime).

23.8.2 TX State Timing

Transmission of preamble begins 192 µs after the STXON or STXONCCA command strobe. This is referred to as *TX turnaround time* in the IEEE 802.15.4 standard. There is an equal delay when returning to receive mode.

When returning to idle or receive mode, there is a 2-µs delay while the modulator ramps down the signals to the DACs. The down-ramping occurs automatically after the complete MPDU (as defined by the length byte) transmits or if TX underflow occurs. This affects the following:

- The SFD signal, which is stretched by 2 µs
- The radio FSM transition to the IDLE state, which is delayed by 2 µs

23.8.3 TX FIFO Access

The TX FIFO can hold 128 bytes and only one frame at a time. Frame buffering can occur before or after the TX command strobe executes, as long as buffering does not generate TX underflow (see the error conditions listed in Error Conditions).

Figure 23-5 shows what must be written to the TX FIFO (marked blue). Additional bytes are ignored, unless TX overflow occurs (see the error conditions listed in Error Conditions).

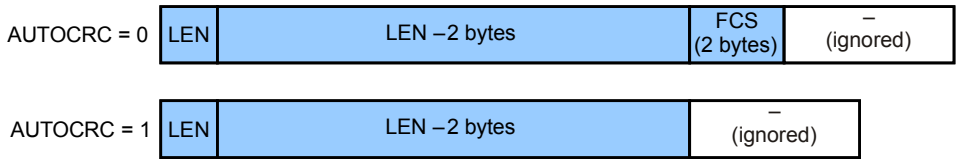


Figure 23-5. Frame Data Written to the TX FIFO

There are two ways to write to the TX FIFO:

- Write to the **RFDATA** register.
- Frame buffering always begins at the start of the TX FIFO memory. By enabling the **IGNORE_TX_UNDERF** bit of the **FRMCTRL1** register, it is possible to write directly into the RAM area in the radio memory, which holds the TX FIFO. The **RFDATA** register is recommended for writing data to the TX FIFO.

The number of bytes in the TX FIFO is stored in the **TXFIFOCNT** register.

The SFLUSHTX command strobe manually empties the TX FIFO. TX underflow occurs if the FIFO is emptied during transmission.

23.8.4 Retransmission

To support simple retransmission of frames, the radio does not delete the TX FIFO contents as they are transmitted. After a frame successfully transmits, the FIFO contents are left unchanged. To retransmit the same frame, simply restart TX by issuing an STXON or STXONCCA command strobe. Retransmission of a packet is possible only if the packet has been completely transmitted; that is, a packet cannot be aborted and then retransmitted.

To transmit a different frame, issue an ISFLUSHTX strobe and then write the new frame to the TX FIFO.

23.8.5 Error Conditions

Two error conditions are associated with the TX FIFO:

- Overflow occurs when the TX FIFO is full and another byte write is attempted.
- Underflow occurs when the TX FIFO is empty and the radio tries to fetch another byte for transmission.

TX overflow is indicated when the TX_OVERFLOW interrupt flag is set. When this error occurs, the writing is aborted, (that is, the data byte that caused the overflow is lost). The SFLUSHTX strobe clears the error condition.

TX underflow is indicated when the TX_UNDERFLOW interrupt flag is set. When this error occurs, the ongoing transmission is aborted. The SFLUSHTX strobe clears the error condition.

Setting the **IGNORE_TX_UNDERF** bit of the **FRMCTRL1** register can disable the TX_UNDERFLOW exception. In this case, the radio continues transmitting the bytes that happen to be in the TX FIFO memory, until the number of bytes given by the first byte (that is, the length byte) are transmitted.

23.8.6 TX Flow Diagram

Figure 23-6 summarizes the transmitter flow in a flow diagram:

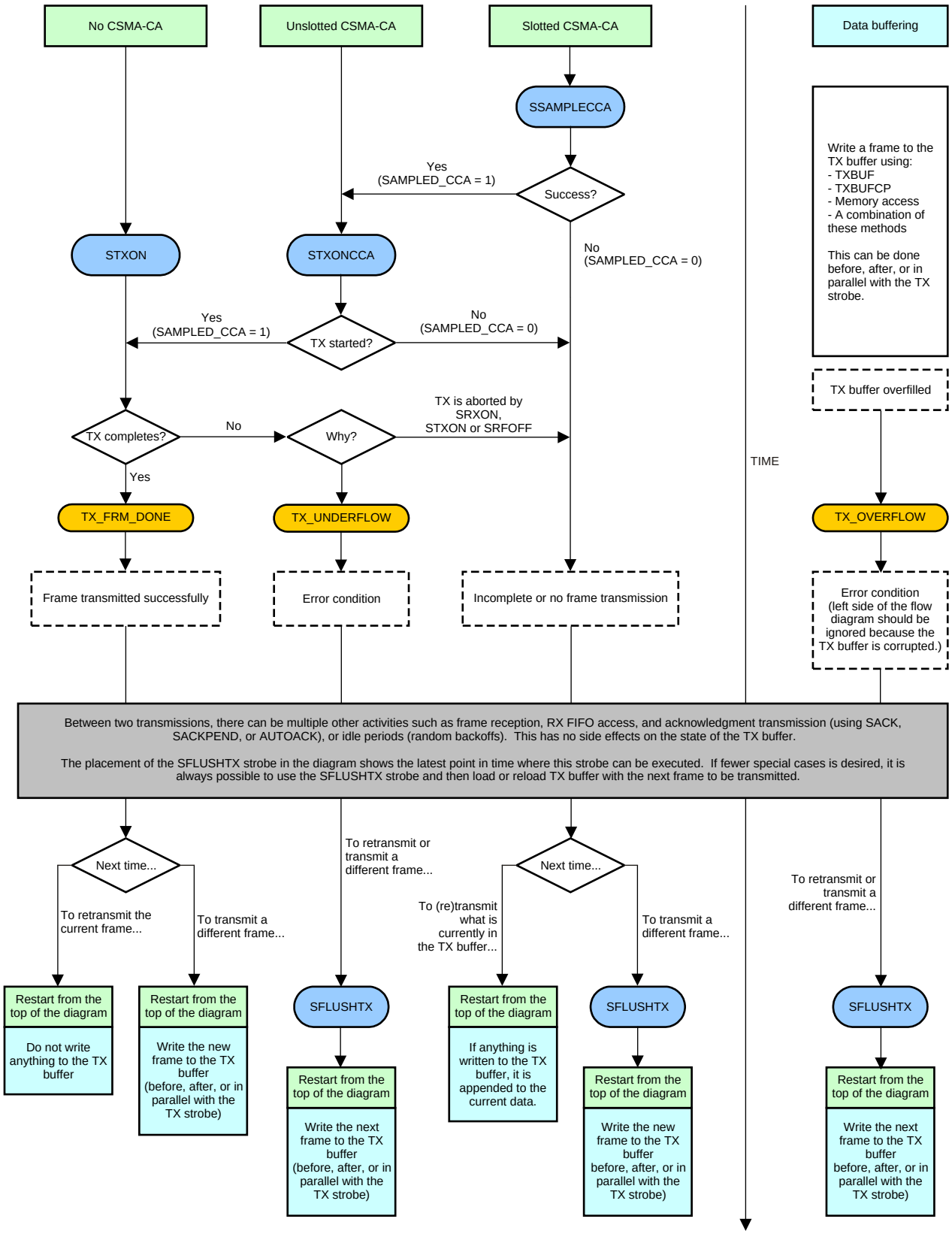


Figure 23-6. TX Flow

23.8.7 Frame Processing

The radio performs the following frame generation tasks for TX frames:

(1) Generation and automatic transmission of the PHY synchronization header, which consists of the preamble and the SFD

(2) Transmission of the number of bytes specified by the frame-length field

(3) Calculation of and automatic transmission of the FCS (can be disabled)

The recommended usage is to write the frame-length field followed by the MAC header and MAC payload to the TX FIFO and let the radio handle the rest. The frame-length field must include the 2 FCS bytes, even though the radio adds these automatically.

23.8.8 Synchronization Header

The radio has programmable preamble length. The default value is compliant with IEEE 802.14.5, and changing the value makes the system noncompliant to the IEEE 802.15.4 standard.

The preamble sequence length is set by the **PREAMBLE_LENGTH** bit of the **MDMCTRL0** register. Figure 23-7 shows how the synchronization header relates to the IEEE 802.15.4 specification.

When the required number of preamble bytes is transmitted, the radio automatically transmits the 1-byte SFD. The SFD is fixed, and it is not possible to change this value from software.

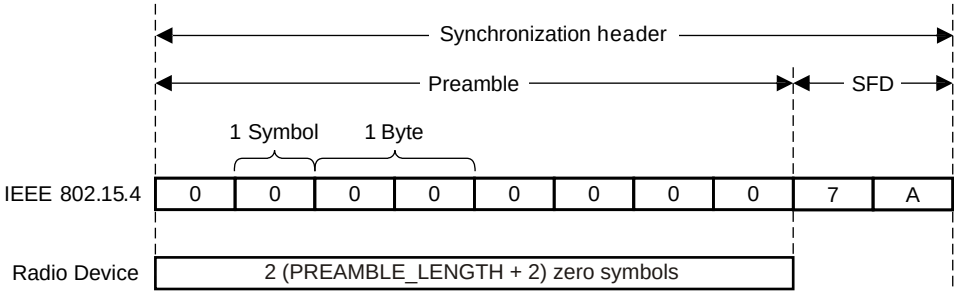


Figure 23-7. Transmitted Synchronization Header

23.8.9 Frame-Length Field

When the SFD is transmitted, the modulator starts to read data from the TX FIFO. The modulator expects to find the frame-length field followed by the MAC header and MAC payload. The frame-length field is used to determine how many bytes are transmitted.

The minimum frame length is 3 bytes when **AUTOCRC** = 1 and 1 byte when **AUTOCRC** = 0.

23.8.10 Frame-Check Sequence

When the **AUTOCRC** control bit of the **FRMCTRL0** register is set, the FCS field is automatically generated and appended to the transmitted frame at the position defined by the frame-length field. The FCS is not written to the TX FIFO, but is stored in a separate 16-bit register. It is recommended always to have the **AUTOCRC** bit enabled, except possibly for debug purposes. If the **AUTOCRC** control bit of the **FRMCTRL0** register is set to 0, then the modulator expects to find the FCS in the TX FIFO, so software must generate the FCS and write it to the TX FIFO with the rest of the MPDU.

Figure 23-8 shows the hardware implementation of the FCS calculator.

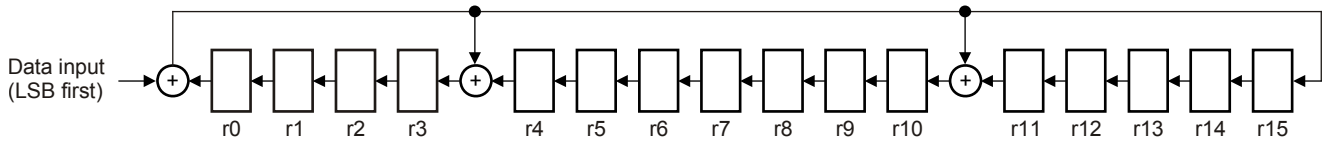


Figure 23-8. FCS Hardware Implementation

23.8.11 Interrupts

The SFD interrupt is raised when the SFD field of the frame transmits. At the end of the frame, the TX_FRM_DONE interrupt is raised when the complete frame successfully transmits.

A second SFD signal is available on the GPIO (through radio observation mux); do not confuse this second SFD signal with the SFD interrupt.

23.8.12 Clear-Channel Assessment

The clear-channel assessment (CCA) status signal indicates whether the channel is available for transmission or not. The CCA function is used to implement the CSMA-CA functionality specified in the IEEE 802.15.4 specification. The CCA signal is valid when the receiver remains enabled for at least eight symbol periods. Use the **RSSI_VALID** status signal to verify this.

The CCA is based on the RSSI value and a programmable threshold. The exact behavior is configurable in the **CCACTRL0** and **CCACTRL1** registers.

There are two variations of the CCA signal, one that is updated at every new RSSI sample and one that is updated only on SSAMPLECCA or ISAMPLECCA and STXONCCA or ISTXONCCA command strobes. They are both available in the **FSMSTAT1** register.

The CCA signal is updated four clock cycles (system clock) after the **RSSI_VALID** signal is set.

23.8.13 Output Power Programming

The RF output power is controlled by the 8-bit value in the **TXPOWER** register. The data sheet for the CC2538 device shows typical output power and current consumption for recommended settings when the center frequency is set to 2.440 GHz. The recommended settings are only a small subset of all the possible register settings.

23.8.14 Tips and Tricks

- There is no requirement to have the complete frame in the TX FIFO before starting a transmission. During transmission, it is possible to add bytes to the TX FIFO.

23.9 Receive Mode

This section describes how to control the receiver, control the integrated RX frame processing, and how to use the RX FIFO.

23.9.1 RX Control

The receiver is turned on and off with the SRXON and SRFOFF command strobes, and with the **RXENABLE** registers. The command strobes provide a hard on and off mechanism, whereas **RXENABLE** manipulation provides a soft on and off mechanism.

The receiver is turned on by the following actions:

- The SRXON command strobe:
  - Sets **RXENABLE[7]**
  - Aborts ongoing transmission or reception by forcing a transition to RX calibration.
- The STXON command strobe, when the **SET_RXENMASK_ON_TX** bit of the **FRMCTRL1** register is enabled:

  - Sets **RXENABLE[6]**
  - Enables the receiver after transmission completes.
- Setting **RXENABLE** != 0x00 by writing to **RXENMASKOR**:
  - Does not abort ongoing transmission or reception.

The receiver is turned off by the following actions:

- The SRFOFF command strobe:
  - Clears **RXENABLE[7:0]**
  - Aborts ongoing transmission or reception by forcing the transition to IDLE mode.
- Setting **RXENABLE** = 0x00 by writing to RXENMASKAND
  - Does not abort ongoing transmission or reception. Once the ongoing transmission or reception is finished, the radio returns to the IDLE state.

There are several ways to manipulate the **RXENABLE** registers:

- The SRXMASKBITSET and SRXMASKBITCLR strobes (affecting **RXENABLE[5]**)
- The SRXON, SRFOFF, and STXON command strobes, including the setting of the **SET_RXMASK_ON_TX** bit of the **FRMCTRL1** register

23.9.2 RX State Timing

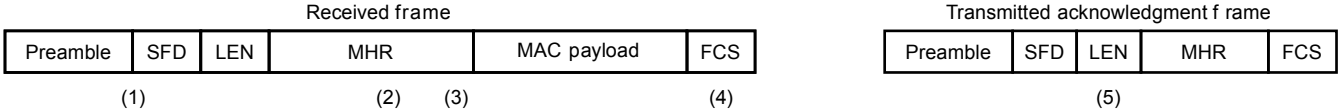
The receiver is ready 192 µs after RX is enabled by one of the methods described in Section 23.9.1, *RX Control*. This is referred to as RX turnaround time in the IEEE 802.15.4 standard.

After frame reception, there is by default an interval of 192 µs where SFD detection is disabled. This interval can be removed by clearing the **RX2RX_TIME_OFF** bit of the **FSMCTRL** register.

23.9.3 Frame Processing

The radio integrates critical portions of the RX requirements in IEEE 802.15.4-2003 and -2006 in hardware. This reduces the CPU interruption rate, simplifies the software that handles frame reception, and provides the results with minimum latency.

During reception of a single frame, the following frame-processing steps are performed:



(1) Detection and removal of the received PHY synchronization header (preamble and SFD), and reception of the number of bytes specified by the frame-length field

(2) Frame filtering as specified by IEEE 802.15.4, section 7.5.6.2, third filtering level

(3) Matching of the source address against a table containing up to 24 short addresses or 12 extended IEEE addresses. The source address table is stored in the radio RAM.

(4) Automatic FCS checking, and attaching this result and other status values (RSSI, correlation and source match result) to received frames

(5) Automatic acknowledgment transmission with correct timing, and correct setting of the frame pending bit, based on the results from source address matching and FCS checking

23.9.4 Synchronization Header and Frame-Length Fields

Frame reception starts with detection of an SFD, followed by the length byte, which determines when the reception is complete. The SFD signal, which can be output on GPIO, can be used to capture the start of received frames (see Figure 23-9):

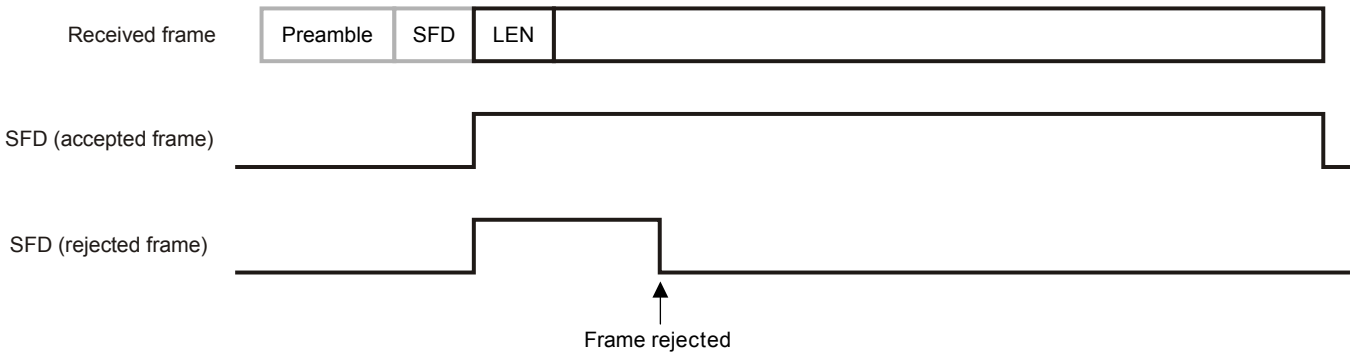


Figure 23-9. SFD Signal Timing

Preamble and SFD are not written to the RX FIFO.

The radio uses a correlator to detect the SFD. The correlation threshold value in the **CORR_THR** bit of the **MDMCTRL1** register determines how closely the received SFD must match an ideal SFD. Adjust the threshold with care:

- If the radio is set too high, it misses many actual SFDs, effectively reducing the receiver sensitivity.
- If the radio is set too low, it detects many false SFDs. Although a low setting does not reduce the receiver sensitivity, the effect is similar, because false frames might overlap with the SFDs of actual frames. A setting that is too low also increases the risk of receiving false frames with correct FCS.

In addition to SFD detection, it is also possible to require a number of valid preamble symbols (also above the correlation threshold) before SFD detection. See the descriptions of the **MDMCTRL0** and **MDMCTRL1** registers for available options and recommended settings.

23.9.5 Frame Filtering

The frame filtering function rejects unintended frames as specified by IEEE 802.15.4, section 7.5.6.2, third filtering level. In addition, it provides filtering on:

- The eight different frame types (see the **FRMFILT1** register)
- The reserved bits in the frame control field (FCF)

The function is controlled by:

- The **FRMFILT0** and **FRMFILT1** registers
- The PAN_ID, SHORT_ADDR, and EXT_ADDR values in RAM

23.9.5.1 Filtering Algorithm

The **FRM_FILTER_EN** bit of the **FRMFILT0** register controls whether frame filtering is applied or not. When disabled, the radio accepts all received frames. When enabled (which is the default setting), the radio only accepts frames that fulfill all of the following requirements:

- The length byte must be equal to or higher than the minimum frame length, which is derived from the source- and destination-address mode and PAN_ID compression subfields of the FCF.
- The reserved FCF bits [9:7] ANDed together with the **FCF_RESERVED_BITMASK** bit of the **FRMFILT0** register must equal 000b.
- The value of the frame version subfield of the FCF cannot be higher than the **MAX_FRAME_VERSION** bit of the **FRMFILT0** register.
- The source- and destination-address modes cannot be reserved values (1).
- Destination address:
  - If a destination PAN_ID is included in the frame, it must match PAN_ID or must be the broadcast PAN identifier (0xFFFF).
  - If a short destination address is included in the frame, it must match either SHORT_ADDR or the broadcast address (0xFFFF).

  - If an extended destination address is included in the frame, it must match EXT_ADDR.
- Frame type:
  - Beacon frames (0) are accepted only when:
    - The **ACCEPT_FT0_BEACON** bit of the **FRMFILT1** register is 1.
    - Length byte ≥ 9
    - The destination-address mode is 0 (no destination address).
    - The source-address mode is 2 or 3 (that is, a source address is included).
    - The source PAN_ID matches PAN_ID, or PAN_ID equals 0xFFFF.
  - Data (1) frames are accepted only when:
    - The **ACCEPT_FT1_DATA** bit of the **FRMFILT1** register is 1.
    - Length byte ≥ 9
    - A destination address and/or source address is included in the frame. If no destination address is included in the frame, the **PAN_COORDINATOR** bit of the **FRMFILT0** register must be set, and the source PAN_ID must equal PAN_ID.
  - Acknowledgment (2) frames are accepted only when:
    - The **ACCEPT_FT2_ACK** bit of the **FRMFILT1** register is 1.
    - Length byte = 5
  - MAC command (3) frames are accepted only when:
    - The **ACCEPT_FT3_MAC_CMD** bit of the **FRMFILT1** register is 1.
    - Length byte ≥ 9
    - A destination address and/or source address is included in the frame. If no destination address is included in the frame, the **PAN_COORDINATOR** bit of the **FRMFILT0** register must be set, and the source PAN_ID must equal PAN_ID for the frame to be accepted.
  - Reserved frame types (4, 5, 6, and 7) are accepted only when:
    - The **ACCEPT_FT4TO7_RESERVED** bit of the **FRMFILT1** register is 1 (default is 0).
    - Length byte ≥ 9

The following operations are performed before filtering begins, with no effect on the frame data stored in the RX FIFO:

- Bit 7 of the length byte is masked out (don't care).
- If the **MODIFY_FT_FILTER** bit of the **FRMFILT1** register is not equal to 0, the MSB of the frame type subfield of the FCF is either inverted or forced to 0 or 1.

If a frame is rejected, the radio starts searching for a new frame only after the rejected frame is completely received (as defined by the frame-length field) to avoid detecting false SFDs within the frame. A rejected frame can generate RX overflow if it occurs before the frame is rejected.

23.9.5.2 Interrupts

When frame filtering is enabled and the filtering algorithm accepts a received frame, an RX_FRM_ACCEPTED interrupt is generated. This interrupt is not generated if frame filtering is disabled or if RX_OVERFLOW or RX_FRM_ABORTED is generated before the filtering result is known.

Figure 23-10 shows the three different scenarios (not including the overflow and abort-error conditions).

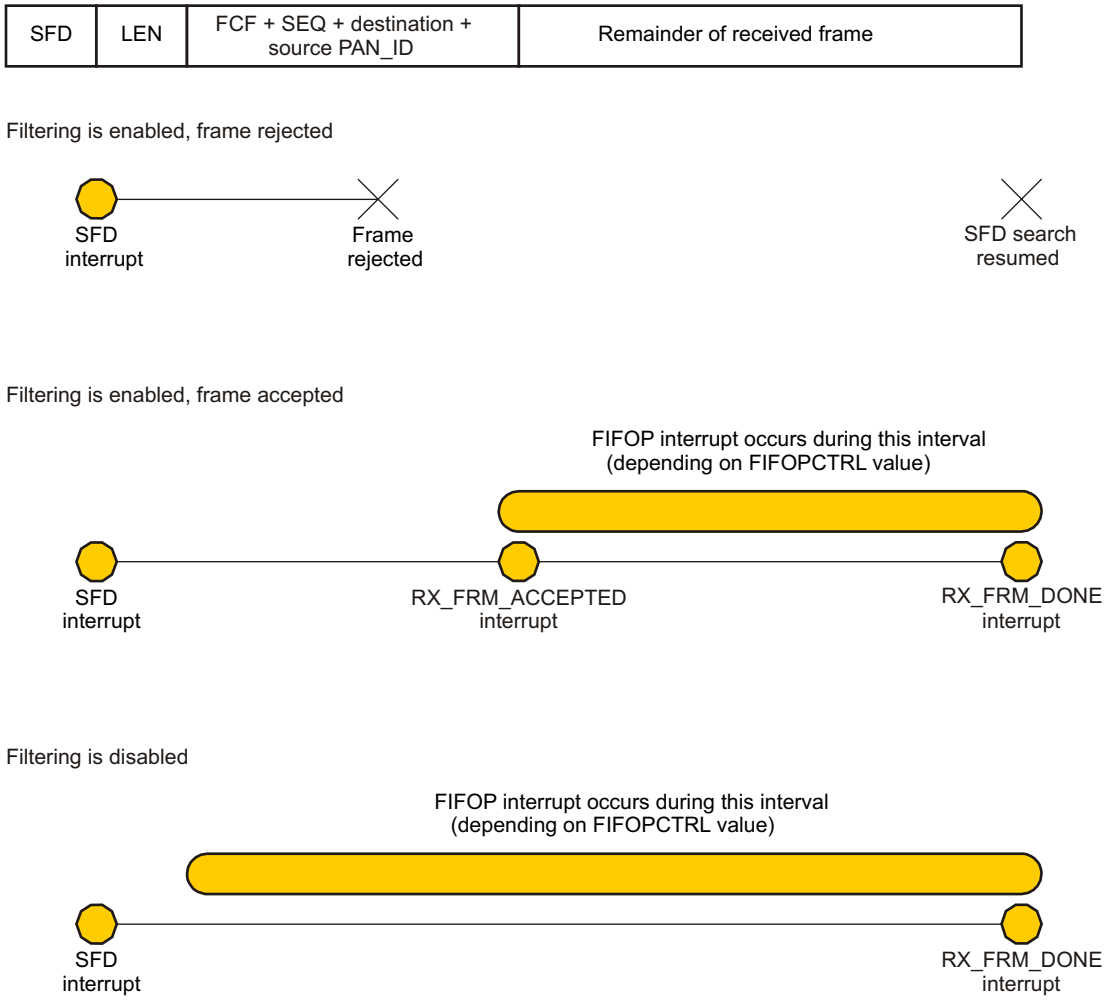


Figure 23-10. Filtering Scenarios (Exceptions Generated During Reception)

The **SFD** bit of the **FSMSTAT1** register goes high when a start-of-frame delimiter is completely received and remains high until either the last byte in MPDU is received or the received frame fails to pass address recognition and is rejected.

23.9.5.3 Tips and Tricks

The following register settings must be configured correctly:

- Set the **PAN_COORDINATOR** bit of the **FRMFILT0** register if the device is a PAN coordinator, or clear this bit if the device is not a PAN coordinator.
- The **MAX_FRAME_VERSION** bit of the **FRMFILT0** register must correspond to the supported versions of the IEEE 802.15.4 standard.
- The local address information must be loaded into RAM.

To avoid completely the receipt of frames during energy-detection scanning, set the **RX_MODE** bit of the **FRMCTRL0** register to 11b and then restart RX. This disables symbol search and thereby prevents SFD detection.

To resume normal RX mode, set the **RX_MODE** bit of the **FRMCTRL0** register to 00b and restart RX.

During operation in a busy IEEE 802.15.4 environment, the radio receives large numbers of unintended acknowledgment frames. To block reception of these frames effectively, use the **ACCEPT_FT2_ACK** bit of the **FRMFILT1** register to control the expected receipt of acknowledgment frames:

- Set the **ACCEPT_FT2_ACK** bit of the **FRMFILT1** register after successfully starting a transmission

with acknowledgment request, and clear the bit again after the acknowledgment frame is received or the time-out is reached.

- Otherwise, keep the bit cleared.

It is not necessary to turn off the receiver while changing the values of the **FRMFILT0** and **FRMFILT1** registers and the local address information stored in RAM. However, if the changes occur between reception of the SFD byte and the source PAN_ID (that is, between the SFD and RX_FRM_ACCEPTED exceptions), consider the modified values as don't care for that particular frame (the radio uses either the old or the new value).

23.9.6 Source Address Matching

The radio supports matching of the source address in received frames against a table stored in the on-chip memory. The table is 96 bytes long, and hence it can contain up to:

- 24 short addresses (2-byte PAN ID + 2-byte short address) or
- 12 IEEE extended addresses (8 bytes each)

Source address matching is performed only when frame filtering is also enabled and the received frame was accepted. The function is controlled by:

- The **SRCMATCH**, **SRCSHORTEN0**, **SRCSHORTEN1**, **SRCSHORTEN2**, **SRCEXTEN0**, **SRCEXTEN1**, and **SRCEXTEN2** registers
- The source address table in RAM

23.9.6.1 Applications

- Automatic acknowledgment transmission with correct setting of the frame-pending bit:

  When using indirect frame transmission, the devices send data requests to poll frames stored on the coordinator. To indicate whether it actually has a frame stored for the device, the coordinator must set or clear the frame-pending bit in the returned acknowledgment frame. On most 8- and 16-bit MCUs, however, there is not enough time to determine this, and so the coordinator ends up setting the pending bit regardless of whether there are pending frames for the device (as required by IEEE 802.15.4). This is wasteful in terms of power consumption, because the polling device must keep its receiver enabled for a considerable period of time, even if there are no frames for it. By loading the destination addresses in the indirect frame queue into the source address table and enabling the AUTOPEND function, the radio sets the pending bit in outgoing acknowledgment frames automatically. This way, the operation is no longer timing-critical, because the effort done by the MCU occurs when adding or removing frames in the indirect frame queue and updating the source address table accordingly.
- Security material look-up:

  To reduce the time needed to process secured frames, the source address table can be set up so the entries match the table of security keys on the CPU. A second level of masking on the table entries allows this application to be combined with automatic setting of the pending bit in acknowledgment frames.
- Other applications:

  The two previous applications are the main targets for the source-address matching function. However, for proprietary protocols that rely only on the basic IEEE 802.15.4 frame format, there are several other useful applications. For instance, it is possible to create firewall functionality where only a specified set of nodes is acknowledged.

23.9.6.2 The Source Address Table

The source address table begins at address 0x4008 8400 in RAM. The space is shared between short and extended addresses, and the **SRCSHORTEN0** through **SRCSHORTEN2** and **SRCEXTEN0** through **SRCEXTEN2** registers are used to control which entries are enabled. All values in the table are little-endian (as in the received frames).

- A short address entry starts with the 16-bit PAN_ID followed by the 16-bit short address. These entries are stored at address 0x4008 8400 + (16 × n), where n is a number from 0 to 23.

- An extended address entry consists only of the 64-bit IEEE extended address. These entries are stored at address 0x4008 8400 + (32 × n), where n is a number from 0 to 11.

23.9.6.3 Address Enable Registers

Software allocates table entries and makes sure that active short and extended address entries do not overlap. There are separate enable bits for short and extended addresses:

- Short address entries are enabled in the **SRCSHORTEN0**, **SRCSHORTEN1**, and **SRCSHORTEN2** registers. Register bit **n** corresponds to short address entry **n.**
- Extended address entries are enabled in the **SRCEXTEN0**, **SRCEXTEN1**, and **SRCEXTEN2** registers. In this case, register bit **2n** corresponds to extended address entry **n**. This mapping is convenient when creating a combined bit vector (of short and extended enable bits) to find unused entries. Moreover, when read, register bits **2n + 1** are always 0. This does not affect functionality since they are don't care.

23.9.6.4 Matching Algorithm

The **SRC_MATCH_EN** bit of the **SRCMATCH** register controls whether source address matching is enabled or not. When source address matching is enabled (which is the default setting) and a frame passes the filtering algorithm, the radio applies one of the algorithms outlined in Figure 23-11, depending on which type of source address is present.

The result is reported in two different forms:

- A 24-bit vector called SRCRESMASK contains a 1 for each enabled short entry with a match, or two 1s for each enabled extended entry with a match (the bit mapping is the same as for the address-enable registers on read access).
- A 7-bit value called SRCRESINDEX:
  - When no source address is present in the received frame, or there is no match on the received source address:
    - Bits [6:0]: 011 1111
  - If there is a match on the received source address:
    - Bits [4:0]: The index of the first entry (that is, the one with the lowest index number) with a match, 0–23 for short addresses or 0–11 for extended addresses
    - Bit [5]: 0 if the match is on a short address, 1 if the match is on an extended address
    - Bit [6]: The result of the AUTOPEND function

Short Source Address (Mode 2)

The received source PAN ID is called srcPanid. The received short address is called srcShort.

```
SRCRESMASK = 0x000000;
 SRCRESINDEX = 0x3F;
 for (n = 0; n < 24; n++) {
   bitVector = 0x000001 << n;
   if (SRCSHORTEN & bitVector) {
     if ((panid[n] == srcPanid) &&
       (short[n] == srcShort)) {
       SRCRESMASK |= bitVector;
       if (SRCRESINDEX == 0x3F) {
         SRCRESINDEX = n;
       }
     }
   }
 }
```

Extended Source Address (Mode 3)

The received extended address is called srcExt.

```
SRCRESMASK = 0x000000;
 SRCRESINDEX = 0x3F;
 for (n = 0; n < 12; n++) {
   bitVector = 0x000003 << (2*n);
   if (SRCEXTEN & bitVector) {
     if (ext[n] == srxExt) {
       SRCRESMASK |= bitVector;
       if (SRCRESINDEX == 0x3F) {
         SRCRESINDEX = n | 0x20;
       }
     }
   }
 }
```

Figure 23-11. Matching Algorithm for Short and Extended Addresses

SRCRESMASK and SRCRESINDEX are written to RF Core memory as soon as the result is available.

SRCRESINDEX is also appended to received frames if the **AUTOCRC** and **APPEND_DATA_MODE** bits of the **FRMCTRL0** register have been set. The value then replaces the 7-bit correlation value of the 16-bit status word.

23.9.6.5 Interrupts

When source address matching is enabled and the matching algorithm completes, the SRC_MATCH_DONE interrupt flag is set, regardless of the result. If a match is found, the SRC_MATCH_FOUND flag is also set immediately before SRC_MATCH_DONE.

Figure 23-12 shows the timing of the setting of flags:

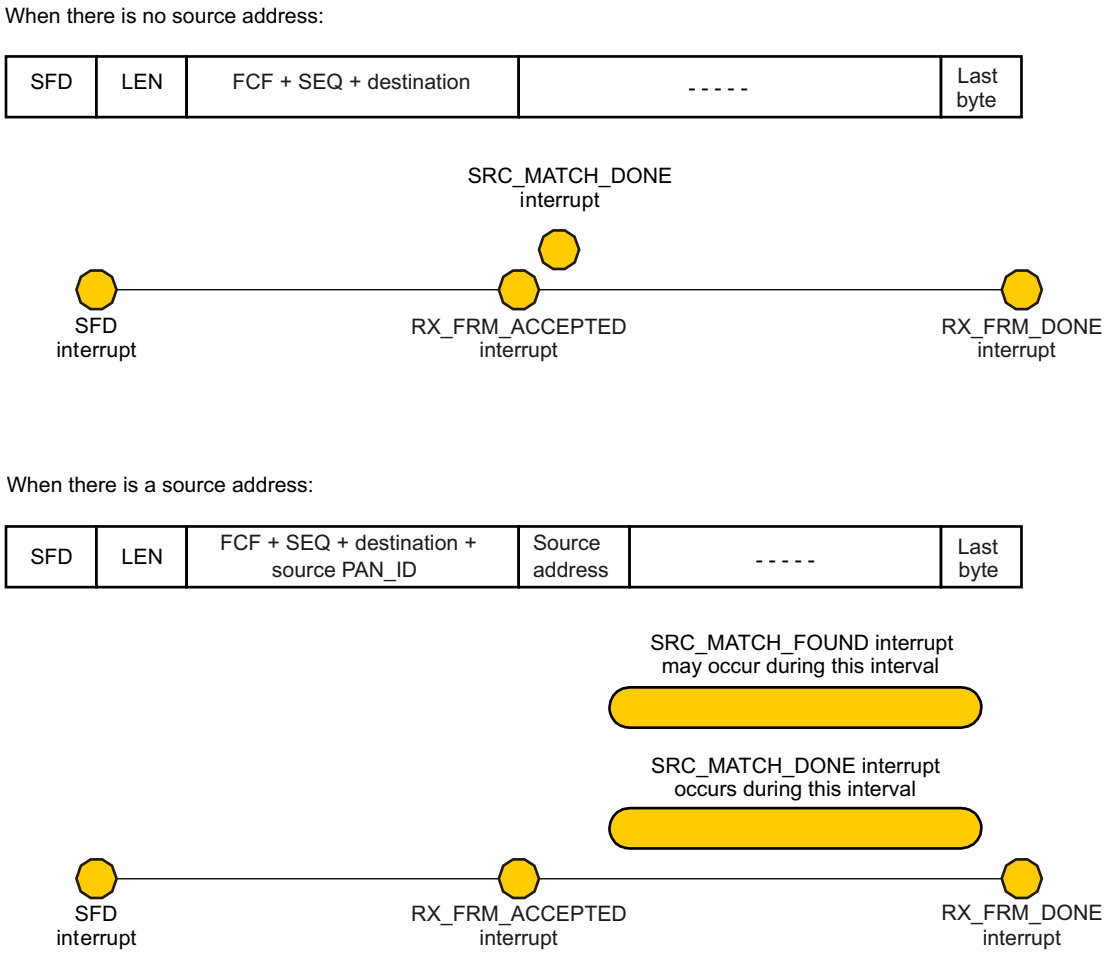


Figure 23-12. Interrupts Generated by Source Address Matching

23.9.6.6 Tips and Tricks

- The source address table can be modified safely during frame reception. If one address replaces another while the receiver is active, the corresponding enable bit should be turned off during the modification. This prevents the RF Core from using a combination of old and new values, because it considers only entries that are enabled throughout the whole source matching process.

Take the following measures to avoid the next received frame from overwriting the results from source address matching:

- Use the appended SRCRESINDEX result instead of the value written to RAM (this is the recommended approach).
- Read the results from RAM before RX_FRM_ACCEPTED occurs in the next received frame. For the shortest frame type, this occurs after the sequence number, so the total available time (absolute worst-

case with a small safety margin) becomes:

16 µs (required preamble) + 32 µs (SFD) + 128 µs (4 bytes) = 176 µs

- To increase the available time, set the **RX2RX_TIME_OFF** bit of the **FSMCTRL** register. This adds another 192 µs, for a total of 368 µs, and also reduces the risk of RX overflow.

23.9.7 Frame-Check Sequence

In receive mode, the FCS is verified by hardware if the **AUTOCRC** bit of the **FRMCTRL0** register is enabled. The user is normally only interested in the correctness of the FCS, not the FCS sequence itself. The FCS sequence is therefore not written to the RX FIFO during receive. Instead, when the **AUTOCRC** bit is set, the 2 FCS bytes are replaced by other more-useful values. The values that are substituted for the FCS sequence are configurable in the **FRMCTRL0** register.

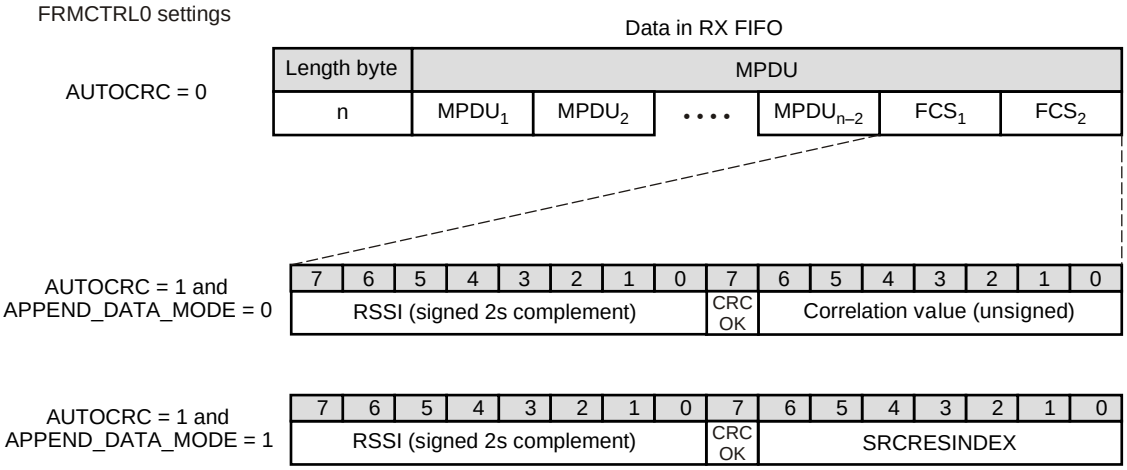


Figure 23-13. Data in RX FIFO for Different Settings

Field Descriptions:

- The RSSI value is measured over the first eight symbols following the SFD.
- The **CRC_OK** bit indicates whether the FCS is correct (1) or incorrect (0). When incorrect, software discards the frame.
- The correlation value is the average correlation value over the first eight symbols following the SFD.
- SRCRESINDEX is the same value that is written to RAM after completion of source-address matching.

Calculation of the link quality indication (LQI) value used by IEEE 802.15.4 specification is described in Section 23.10.4, *Link Quality Indication*.

23.9.8 Acknowledgement Transmission

The radio includes hardware support for acknowledgment transmission after successful frame reception (that is, the FCS of the received frame must be correct). Figure 23-14 shows the format of the acknowledgment frame.

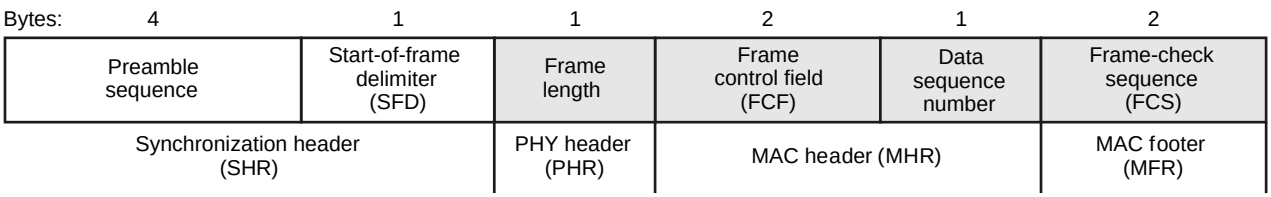


Figure 23-14. Acknowledge Frame Format

The generated acknowledgment frame contains three variable fields:

- The pending bit, which may be controlled with command strobes and the AUTOPEND feature (in the

FCF)

- The data sequence number (DSN), which is taken automatically from the last received frame
- The FCS, which is given implicitly

Three different sources can set the pending bit in an ACK frame (that is, the SACKPEND strobe, the **PENDING_OR** register bit, and the AUTOPEND feature). The pending bit is set if one or more of these sources are set.

23.9.8.1 Transmission Timing

Transmission of acknowledgment frames is possible only after a frame is received. The transmission timing is controlled by the **SLOTTED_ACK** bit of the **FSMCTRL** register. Figure 23-15 shows the acknowledgment timing.

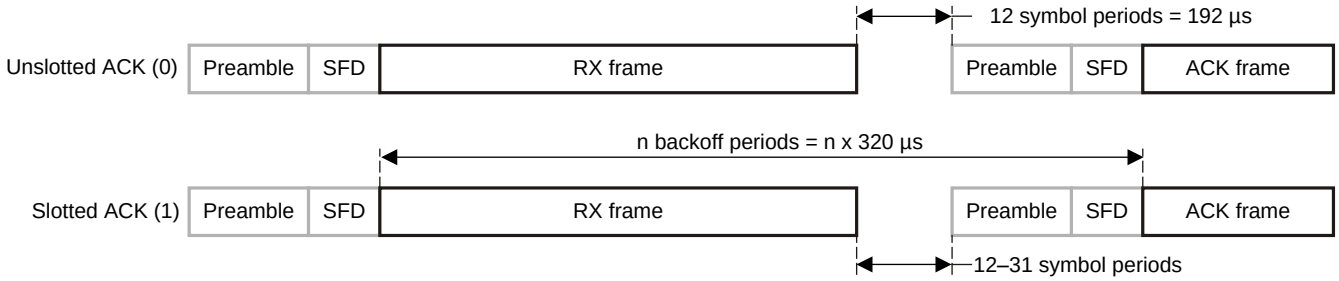


Figure 23-15. Acknowledgement Timing

The IEEE 802.15.4 specification requires unslotted mode in nonbeacon-enabled PANs, and slotted mode for beacon-enabled PANs.

23.9.8.2 Manual Control

The SACK, SACKPEND, and SNACK command strobes are issued only during frame reception. If the strobes are issued at any other time, their only effect is to generate a STROBE_ERROR interrupt. Figure 23-16 shows the command strobe timing.

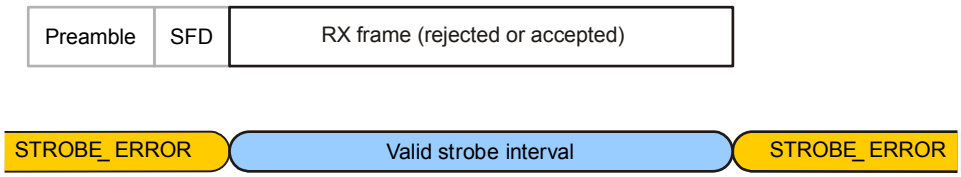


Figure 23-16. Command Strobe Timing

The command strobes may be issued several times during reception; however, only the last strobe has an effect:

- No strobe, SNACK, or incorrect FCS: No acknowledgment transmission
- SACK: Acknowledgment transmission with the frame pending-bit cleared
- SACKPEND: Acknowledgment transmission with the frame pending-bit set

23.9.8.3 Automatic Control (AUTOACK)

When the **FRM_FILTER_EN** bit of the **FRMFILT0** register and the **AUTOACK** bit of the **FRMCTRL0** register are both enabled, the radio determines automatically whether or not to transmit acknowledgment frames:

- Frame filtering must accept the RX frame (indicated by the RX_FRM_ACCEPTED exception).
- The acknowledgment request bit must be set in the RX frame.
- The RX frame must not be a beacon or an acknowledgment frame.

- The FCS of the RX frame must be correct.

Automatic acknowledgments can be overridden by the SACK, SACKPEND, or SNACK command strobes. For instance, if the MCU is low on memory resources and cannot store a received frame, the SNACK command strobe issued during reception prevents acknowledging the discarded frame.

By default, the AUTOACK feature never sets the frame-pending bit in the acknowledgment frames. Apart from manual override with command strobes, there are two options:

- Automatic control, using the AUTOPEND feature
- Manual control, using the **PENDING_OR** bit of the **FRMCTRL1** register

23.9.8.4 Automatic Setting of the Frame Pending Field (AUTOPEND)

When the **AUTOPEND** bit of the **SRCMATCH** register is set, the result from source-address matching determines the value of the frame-pending field. On reception of a frame, the frame-pending field in the (possibly) returned acknowledgment is set, given that:

- The **FRAME_FILTER_EN** bit of the **FRMFILT0** register is set.
- The **SRC_MATCH_EN** bit of the **SRCMATCH** register is set.
- The **AUTOPEND** bit of the **SRCMATCH** register is set.
- The received frame matches the current setting of the **PEND_DATAREQ_ONLY** bit of the **SRCMATCH** register.
- The received source-address matches at least one source-match table entry, which is enabled in both **SRCSHORTEN** and **SRCSHORTPENDEN**, or **SRCEXTEN** and **SRCEXTPENDEN**.

If the source-matching table runs full, use the **PENDING_OR** bit of the **FRMCTRL1** register to override the AUTOPEND feature and temporarily acknowledge all frames with the frame-pending field set.

23.10 RX FIFO Access

The RX FIFO can hold one or more received frames, provided that the total number of bytes is 128 or less. There are two ways to determine the number of bytes in the RX FIFO:

- Reading the **RXFIFOCNT** register
- Using the **FIFOP** and **FIFO** signals in combination with the setting of the **FIFOPTHR** bit of the **FIFOPCTRL** register

The RX FIFO is accessed through the **RFDATA** register.

Accessing the radio RAM directly can also access the data in the RX FIFO. The FIFO pointers are readable in the **RXFIRST_PTR**, **RXLAST_PTR**, and **RXP1_PTR** registers. This is useful if one wants to quickly access a certain byte within a frame without having to read the entire frame first. When using this direct accessing, no FIFO pointers are updated.

The ISFLUSHRX command strobe resets the RX FIFO, resetting all FIFO pointers and clearing all counters, status signals, and sticky-error conditions. However, if the receiver is actively receiving a frame when the FIFO is flushed, the **RFERRF.ABO** flag is asserted.

The SFLUSHRX command strobe resets the RX FIFO, removing all received frames and clearing all counters, status signals, and sticky-error conditions.

23.10.1 Using the FIFO and FIFOP Signals

The FIFO and FIFOP signals are useful when reading out received frames in small portions while the frame is received:

- The **FIFO** bit of the **FSMSTAT1** register goes high when 1 or more bytes are in the RX FIFO, but low when RX overflow occurs.
- The **FIFOP** bit of the **FSMSTAT1** register goes high when:
  - The number of bytes in RX FIFO is greater than **FIFOPCTRL.FIFOP_THR** after frame filtering.
  - There is at least one complete frame in the RX FIFO.
  - RX FIFO overflows.

Figure 23-17 shows the behavior of the FIFO and FIFOP signals.

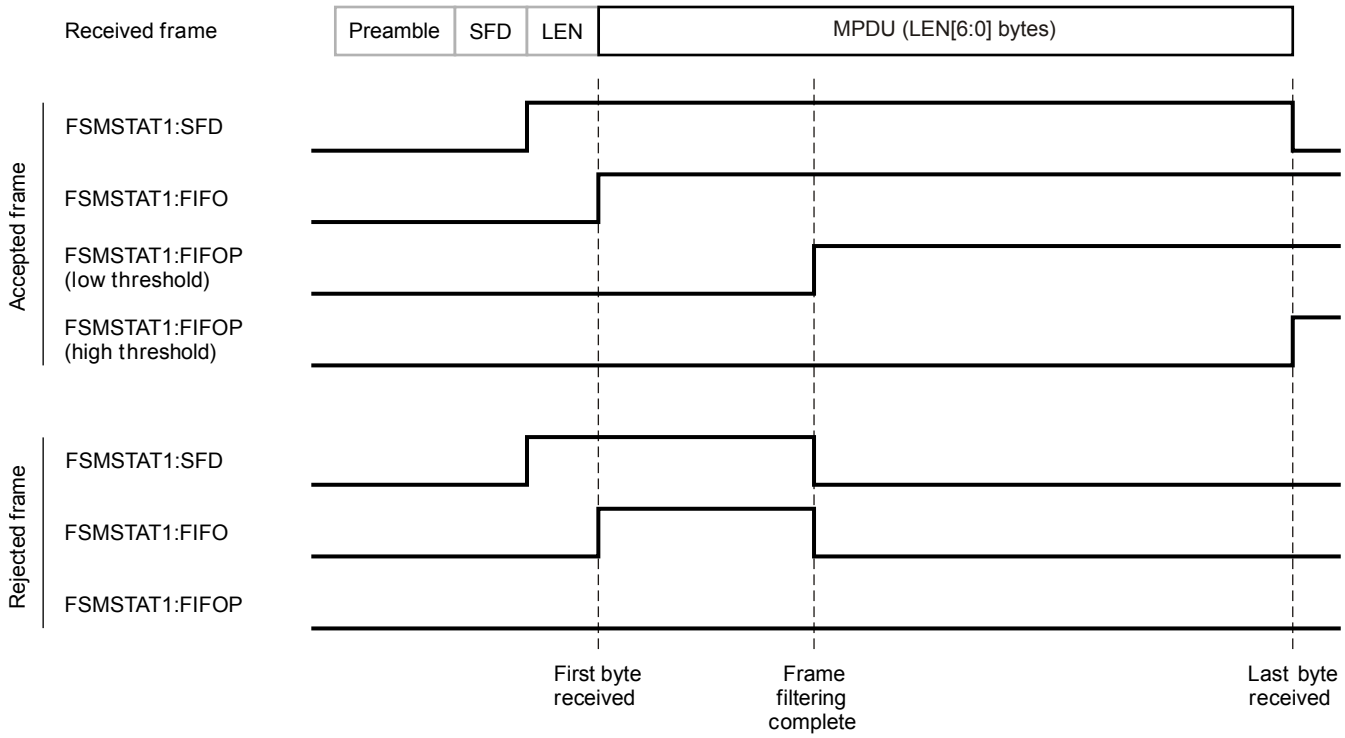


Figure 23-17. Behavior of FIFO and FIFOP Signals

When using the FIFOP as an interrupt source for the MCU, the interrupt service routine (ISR) should adjust the FIFOP threshold to prepare for the next interrupt. When preparing for the last interrupt for a frame, the threshold should match the number of bytes remaining.

23.10.2 Error Conditions

Two error conditions are associated with the RX FIFO:

- Overflow, in which case the RX FIFO is full when another byte is received
- Underflow, in which case software tries to read a byte from an empty RX FIFO

RX overflow is indicated when the **RFERRF.RXOVERF** flag is set and by the signal values **FSMSTAT1.FIFO** = 0 and **FSMSTAT1.FIFOP** = 1. When this error occurs, frame reception halts. The frames currently stored in the RX FIFO may be read out before the ISFLUSHRX command strobe clears the condition. Rejected frames can generate RX overflow if the condition occurs before the frame is rejected.

RX underflow is indicated when the **RFERRF.RXUNDERF** flag is set. RX underflow is a serious error condition that should not occur in error-free software, and the **RXUNDERF** event should be used only for debugging or in a watchdog function. The **RXUNDERF** error is not generated when the read operation occurs simultaneously with the reception of a new byte.

23.10.3 RSSI

The radio has a built-in received signal-strength indication (RSSI), which calculates an 8-bit signed digital value that can be read from a register or automatically appended to received frames. The RSSI value is the result of averaging the received power over eight symbol periods (128 µs) as specified by the IEEE 802.15.4 standard.

The RSSI value is a 2s-complement signed number on a logarithmic scale with 1-dB steps.

Before reading the **RSSI** value register, check the **RSSI_VALID** status bit. The **RSSI_VALID** bit indicates that the RSSI value in the register is in fact valid, which means that the receiver has been enabled for at least eight symbol periods.

To find the actual signal power P at the RF pins with reasonable accuracy, an offset must be added to the RSSI value.

P = RSSI – OFFSET [dBm]

For example, with an offset of 73 dB, reading an RSSI value of –10 from the **RSSI** register means that the RF input power is approximately –83 dBm. See the data sheet for the correct offset value to use.

After the **RSSI** register first becomes valid, the radio can update it two ways: If the **ENERGY_SCAN** bit of the **FRMCTRL0** register is 0 (default), the **RSSI** register contains the most-recent value; however, if this bit is set to 1, a peak search is performed, and the **RSSI** register contains the largest value since the energy scan was enabled.

23.10.4 Link Quality Indication

The LQI is a measurement of the strength and/or quality of the received frame as defined by the IEEE 802.15.4 standard. The IEEE 802.15.4 standard requires that the LQI value is limited to the range 0 through 255, with at least 8 unique values. The radio does not provide an LQI value directly, but reports several measurements that the MCU can use to calculate an LQI value.

The MAC software can use the RSSI value to calculate the LQI value. A disadvantage of this approach is that a narrowband interferer inside the channel bandwidth can increase the RSSI and thus the LQI value, although the true link quality actually decreases. The radio therefore also provides an average correlation value for each incoming frame, based on the first eight symbols that follow the SFD. This unsigned 7-bit value is considered as a measurement of the chip error rate, although the radio does not do chip decision.

As described in Section 23.9.7, *Frame-Check Sequence*, the average correlation value for the first 8 symbols is appended to each received frame, together with the RSSI and CRC OK or not OK, when the **AUTOCRC** bit of the **FRMCTRL0** register is set. A correlation value of approximately 110 indicates a maximum-quality frame, whereas a value of approximately 50 is typically the lowest-quality frame detectable by the radio.

Software must convert the correlation value to the range 0–255 as defined by the IEEE 802.15.4 standard, for instance by calculating:

LQI = (CORR – a)b

limited to the range 0–255, where a and b are found empirically based on PER measurements as a function of the correlation value.

A combination of RSSI and correlation values can also generate the LQI value.

23.11 Radio Control State-Machine

The finite state-machine (FSM) module:

- Maintains the TX FIFO and RX FIFO pointers
- Controls analog dynamic signals such as power up and power down
- Controls the data flow within the RF Core
- Generates automatic acknowledgement frames
- Controls all analog RF calibration

Figure 23-18 shows the main FSM.

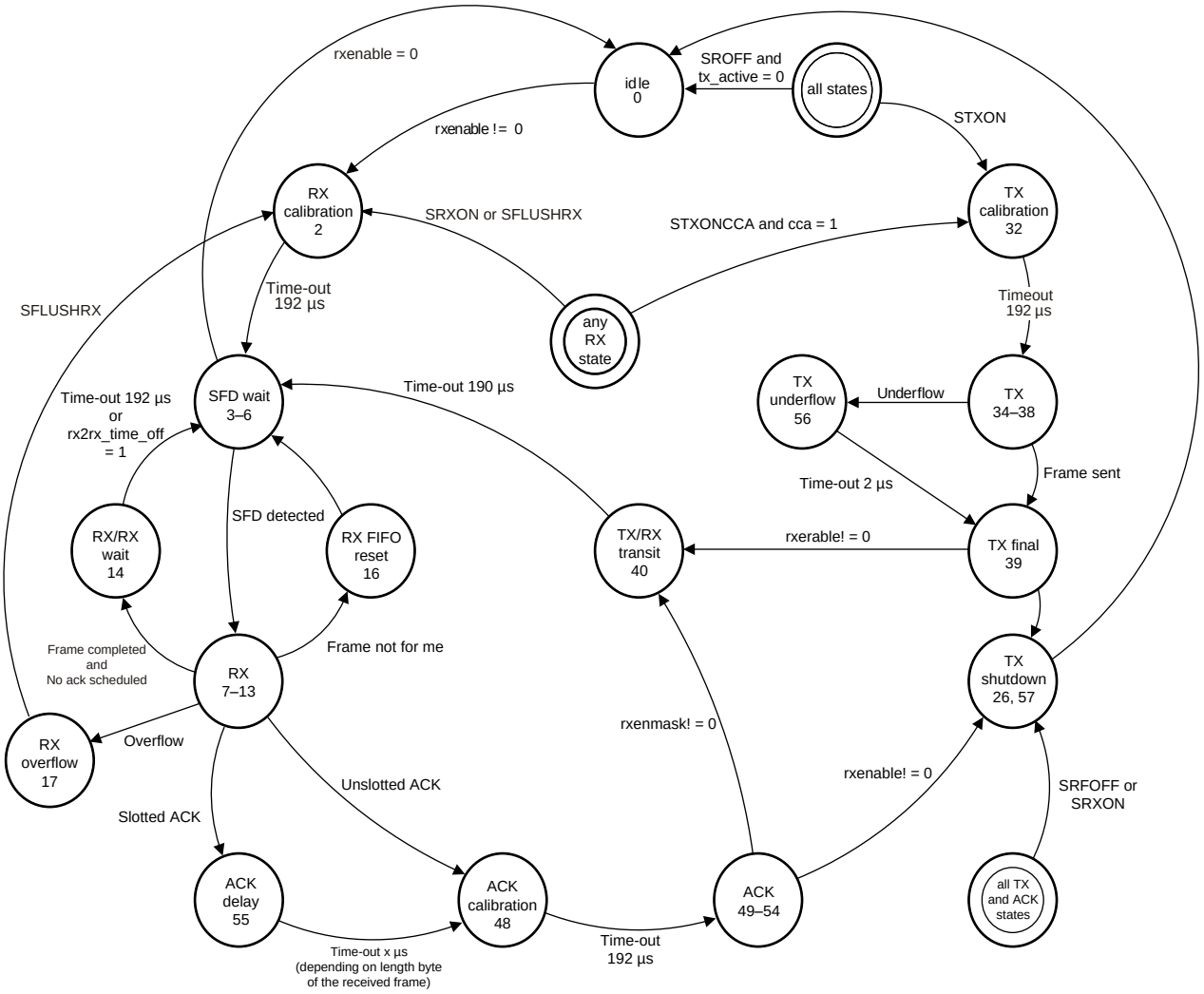


Figure 23-18. Main FSM

Table 23-4 lists the mapping from FSM state to the number which can be read from the **FSMSTAT0** register. Although it is possible to read the state of the FSM, do not use this information to control the program flow in the application software.

Table 23-4. FSM State Mapping

State Name	State Number, Decimal	Number, Hex	TX_ACTIVE	RX_ACTIVE
Idle	0	0x00	0	0
RX calibration	2	0x02	0	1
SFD wait	3–6	0x03–0x06	0	1
RX	7–13	0x07–0x0D	0	1
RX or RX wait	14	0x0E	0	1
RX FIFO reset	16	0x10	0	1
RX overflow	17	0x11	0	0
TX calibration	32	0x20	1	0
TX	34–38	0x22–0x26	1	0
TX final	39	0x27	1	0
TX or RX transit	40	0x28	1	0

Table 23-4. FSM State Mapping (continued)

State Name	State Number, Decimal	Number, Hex	TX_ACTIVE	RX_ACTIVE
ACK calibration	48	0x30	1	0
ACK	49–54	0x31–0x36	1	0
ACK delay	55	0x37	1	0
TX underflow	56	0x38	1	0
TX shutdown	26, 57	0x1A, 0x39	1	0

23.12 Random Number Generation

The RF Core can generate random bits. The chip should be in RX when generation of random bits is required. One must also make sure that the chip has been in RX long enough for the transients to have died out. A convenient way to do this is to wait for the RSSI-valid signal to go high.

Single random bits from either the I or Q channel can be read from the **RFRND** register.

Randomness tests show good results for this module. However, a slight DC component exists. In a simple test where the **RFRND.IRND** register was read a number of times and the data was grouped into bytes, about 20 million bytes were read. When interpreted as unsigned integers between 0 and 255, the mean value was 127.6518, which indicates that there is a DC component.

Figure 23-19 shows the FFT of the first 2^{14} bytes. The DC component is clearly visible. Figure 23-20 shows a histogram (32 bins) of the 20 million values.

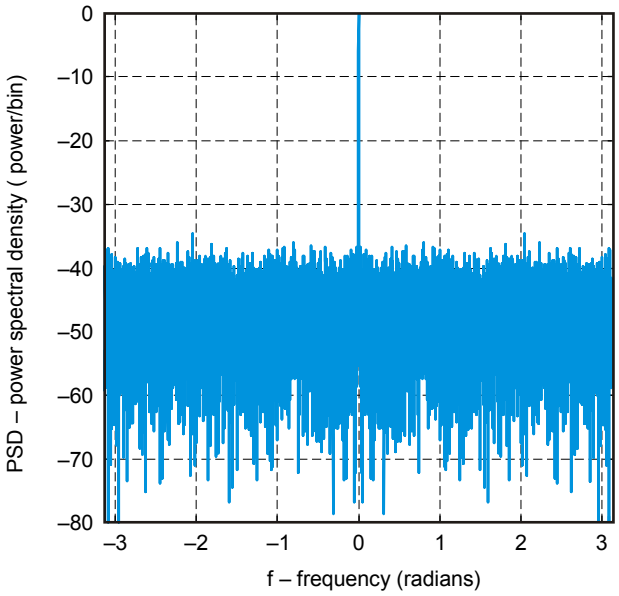


Figure 23-19. FFT of the Random Bytes

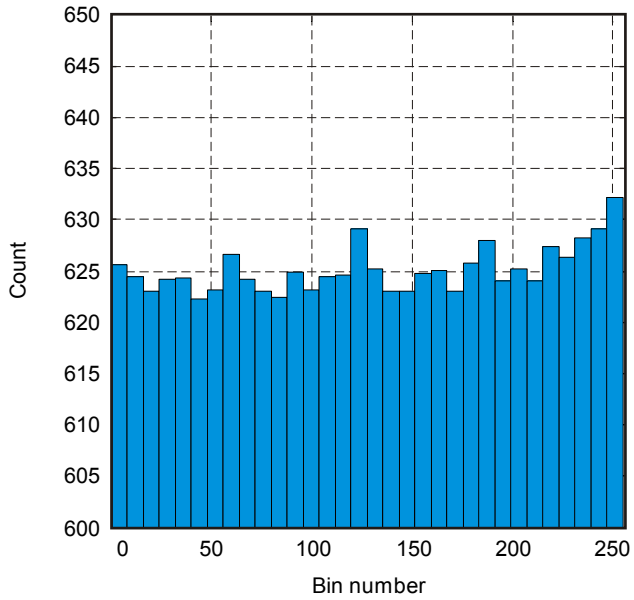


Figure 23-20. Histogram of 20 Million Bytes Generated With the RANDOM Instruction

For the first 20 million individual bits, the probability of a 1 is P(1) = 0.500602 and P(0) = 1 – P(1) = 0.499398.

To fully qualify the random generator as true random, much more elaborate tests are required. Software packages that may be useful in this respect are available on the internet.

23.13 Packet Sniffing and Radio Test Output Signals

Packet sniffing is a nonintrusive way of observing data that is either being transmitted or received. The packet sniffer outputs a clock and a data signal, which should be sampled on the rising edges of the clock. The two packet sniffer signals are observable as GPIO outputs. For accurate time stamping, the SFD signal should also be output, however, it should not be used as an enable signal for SPI slave device as the data signal and SFD are not exactly aligned.

Because the radio has a data rate of 250 kbps, the packet sniffer clock frequency is 250 kHz. The data is output serially, with the MSB of each byte first, which is opposite of the actual RF transmission, but more convenient when processing the data. It is possible to use an SPI slave to receive the data stream.

When sniffing frames in TX mode, the data that the modulator reads from the TX FIFO is the same data that is output by the packet sniffer. However, if automatic cyclic redundancy check (CRC) generation is enabled, the packet sniffer does not output these 2 bytes. Instead, it replaces the CRC bytes with 0x8080. This value can never occur as the last 2 bytes of a received frame (when automatic CRC checking is enabled), and thus it provides a way for the receiver of the sniffed data to separate frames that were transmitted and frames that were received.

When sniffing frames in RX mode, the data that the demodulator writes to the RX FIFO is the same data that is output by the packet sniffer. In other words, the last 2 bytes are either the received CRC value or the CRC OK, RSSI, correlation, SRCRESINDEX value that may automatically replace the CRC value, depending on configuration settings.

To set up the packet sniffer signals or some of the other RF Core observation outputs (in total maximum 3; **rfc_obs_sig0**, **rfc_obs_sig1**, and **rfc_obs_sig2**), the user must perform the following steps:

1. Determine which signal (**rfc_obs_sig**) to output on which GPIO pin. This is done using the OBSSELx control registers (**OBSSEL0**–**OBSSEL7**) that control the observation output to pins PC0:7 (overriding the standard GPIO behavior for those pins).
2. Set the **RFC_OBS_CTRL** control registers (**RFC_OBS_CTRL0**–**RFC_OBS_CTRL2**) to select the correct signals (**rfc_obs_sig**); for example, for packet sniffing, one needs the rfc_sniff_data for the packet sniffer data signal and rfc_sniff_clk for the corresponding clock signal.

3. For packet sniffing, the packet sniffer module must be enabled in the **MDMTEST1** register.

23.14 Command Strobe/CSMA-CA Processor

The command strobe/CSMA-CA processor (CSP) provides the control interface between the CPU and the radio.

The CSP interfaces with the CPU through the registers **RFST**, **CSPX**, **CSPY**, **CSPZ**, **CSPT**, **CSPSTAT**, **CSPCTRL**, and **CSPPROG<n>** (where **n** is in the range 0 to 23). The CSP produces interrupt requests (IRQs) to the CPU. In addition, the CSP interfaces with the MAC timer by observing MAC timer events.

The CSP allows the CPU to issue command strobes to the radio, thus controlling the operation of the radio.

The CSP has two modes of operation, which are described as follows:

- Immediate command strobe execution

  Immediate command strobes are written as Immediate Command Strobe instructions to the CSP, which are issued instantly to the radio module. The Immediate Command Strobe instructions are also used to control the CSP. The Immediate Command Strobe instructions are described in Section 23.14.8, *Instruction Set Summary*.

- Program execution

  Program execution mode means that the CSP executes a sequence of instructions, comprising a short user-defined program, from a program memory or instruction memory. The available instructions are from a set of 20 instructions. The instruction set is defined in Section 23.14.8, *Instruction Set Summary*. The required program is first loaded into the CSP by the CPU, and then the CPU instructs the CSP to start executing the program.

  Program execution mode, together with the MAC timer, allows the CSP to automate CSMA-CA algorithms and thus act as a coprocessor for the CPU.

The operation of the CSP is described in detail in the following sections. The command strobes and other instructions supported by the CSP are given in Section 23.14.9, *Instruction Set Definition*.

23.14.1 Instruction Memory

The CSP executes single-byte program instructions which are read from a 24-byte instruction memory. Writes to the instruction memory are sequential, written through the **RFST** register. An instruction write pointer is maintained within the CSP to hold the location within the instruction memory where the next instruction written to the **RFST** register is to be stored. For debugging purposes, the program currently loaded into the CSP can be read from the registers **CSPPROG<n>**. Following a reset, the write pointer is reset to location 0. During each **RFST** register write, the write pointer is incremented by 1 until the end of memory is reached, at which time the write pointer stops incrementing. The first instruction written to **RFST** is stored in location 0, the location where program execution starts. Thus, a complete 24-instruction program is written to the instruction memory by writing each instruction in the desired order to the **RFST** register.

Writing the immediate command strobe instruction ISSTOP resets the write pointer to 0. In addition, the write pointer is reset to 0 when the command strobe instruction SSTOP is executed in a program.

Following a reset, the instruction memory is filled with SNOP (No Operation) instructions (opcode value 0xC0). The immediate strobe ISCLEAR clears the instruction memory, filling it with SNOP instructions.

While the CSP is executing a program, there must be no attempts to write instructions to the instruction memory by writing to the **RFST** register. Failure to observe this rule can lead to incorrect program execution and corrupt instruction memory contents. However, Immediate Command Strobe instructions may be written to **RFST** (see Section 23.14.3, *Program Execution*).

23.14.2 Data Registers

The CSP has four data registers, **CSPT**, **CSPX**, **CSPY**, and **CSPZ**, which are read and write accessible for the CPU. These registers are read or modified by some instructions, thus letting the CPU set parameters for a CSP program, or letting the CPU read the status of a CSP program.

The **CSPT** data register is not modified by any instruction. The **CSPT** data register is used to set a MAC timer overflow-compare value. Once program execution starts on the CSP, the content of this register is decremented by 1 each time the MAC timer overflows. When **CSPT** reaches 0, program execution is halted and the interrupt IRQ_CSP_STOP is asserted. The **CSPT** register is not decremented if the CPU writes 0xFF to this register.

NOTE: If the **CSPT** register compare function is not used, this register must be set to 0xFF before the program execution starts.

23.14.3 Program Execution

After the instruction memory has been filled, program execution is started by writing the immediate command strobe instruction ISSTART to the **RFST** register. Program execution continues until the instruction at the last location executes, the **CSPT** data register content is 0, an SSTOP instruction executes, an immediate ISSTOP instruction is written to the **RFST** register, or a SKIP instruction returns a location beyond the last location in the instruction memory. The CSP runs at the set system clock frequency, which must be set to 32 MHz for correct radio operation.

Immediate command strobe instructions may be written to the **RFST** register while a program is executing. In this case, the immediate instruction executes before the instruction in the instruction memory, which executes once the immediate instruction completes.

During program execution, reading the **RFST** register returns the current instruction that is executing. An exception to this is the execution of immediate command strobes, during which **RFST** returns 0xD0.

23.14.4 Interrupt Requests

The CSP has three interrupt flags which can produce the RF interrupt vector. These interrupt flags are as follows:

- IRQ_CSP_STOP: asserted when the processor stops because of an SSTOP or ISSTOP instruction or the **CSPT** register is equal to 0
- IRQ_CSP_WT: asserted when the processor continues executing the next instruction after a WAIT W or WAITX instruction
- IRQ_CSP_INT: asserted when the processor executes an INT instruction

23.14.5 Random Number Instruction

There is a delay in the update of the random number used by the RANDXY instruction. Therefore, if the RANDXY instruction, which uses this value, is issued immediately after a previous RANDXY instruction, the random value read may be the same in both cases.

23.14.6 Running CSP Programs

Figure 23-21 shows the basic flow for loading and running a program on the CSP. When program execution stops at the end of the program, the current program remains in program memory so that the same program can rerun by starting execution again with the ISSTART command. To clear the program contents, use the ISCLEAR instruction.

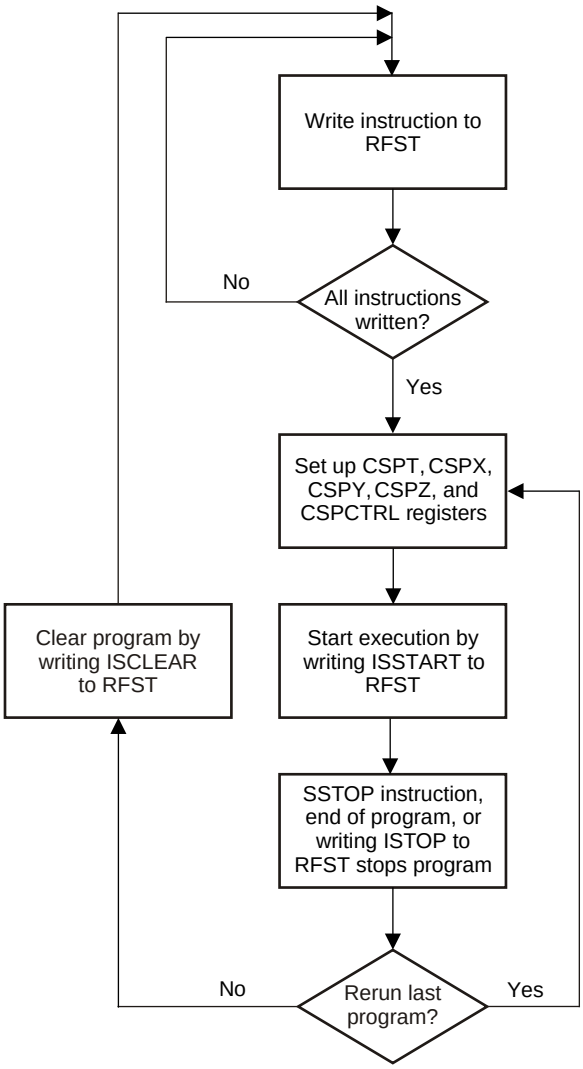


Figure 23-21. Running a CSP Program

23.14.7 CSP Registers

Table 23-5 lists the CSP registers.

Table 23-5. CSP Registers Register Map

Offset	Name	Type	Reset	Description	Link
0x38	RFST	R/W	0x0000 00D0	RF CSMA-CA/strobe processor	RFCORE_SFR_RFST
0x100 + 4n	CSPPROG	R/W	0x0000 00D0	Byte n of CSP program	
0x180	CSPCTRL	R/W	0x0000 0000	CSP control bit	RFCORE_XREG_CSPCTRL
0x184	CSPSTAT	R/W	0x0000 0000	CSP status register	RFCORE_XREG_CSPSTAT
0x188	CSPX	R/W	0x0000 0000	CSP X register	RFCORE_XREG_CSPX
0x18C	CSPY	R/W	0x0000 0000	CSP Y register	RFCORE_XREG_CSPY
0x190	CSPZ	R/W	0x0000 0000	CSP Z register	RFCORE_XREG_CSPY

Table 23-5. CSP Registers Register Map (continued)

Offset	Name	Type	Reset	Description	Link
0x194	CSPT	R/W	0x0000 0000	CSP T register	RFCORE_XREG_CSPT

23.14.8 Instruction Set Summary

This section gives an overview of the instruction set. This is intended as a summary and definition of instruction opcodes. For a description of each instruction, see Section 23.14.9, *Instruction Set Definition*. Each instruction consists of 1 byte, which is written to the **RFST** register for storage in the instruction memory.

The Immediate Strobe instructions (ISxxx) are not used in a program. When these instructions are written to the **RFST** register, they are executed immediately. If the CSP is already executing a program, the current instruction is delayed until the immediate strobe instruction completes.

For undefined opcodes, the behavior of the CSP is defined as a no-operation strobe command (SNOP). Table 23-6 lists the instruction set summary.

Table 23-6. Instruction Set Summary

Mnemonic	7	6	5	4	3	2	1	0	Description
SKIP <C>, <S>	0	S2	S1	S0	N	C2	C1	C0	Skip S instructions on condition C. When condition (C XOR N) is true, skip the next S instructions, else execute the next instruction. If S = 0, re-execute the conditional jump (that is, busy loop until condition is false). Skipping past the last instruction in the command buffer results in an implicit STOP command. The conditions are: C = 0 CCA true C = 1 Synchronization word received and still receiving packet or synchronization word transmitted and still transmitting packet (SFD found, not yet frame end) C = 2 MCU control bit is 1. C = 3 Reserved C = 4 Register X = 0 C = 5 Register Y = 0 C = 6 Register Z = 0 C = 7 **RSSI_VALID** = 1
WAIT <W>	1	0	0	W4	W3	W2	W1	W0	Wait for MAC timer to overflow W times. Waits until the MAC timer has overflowed W times (W = 0 waits 32 times), then continues execution. Generates an IRQ_CSP_WAIT interrupt request when execution continues.
RPT <C>	1	0	1	0	N	C2	C1	C0	Repeat loop while condition C. If condition C is true, go to the instruction following the last LABEL instruction (address in loop-start register); if the condition is false or no LABEL instruction has been executed, go to the next instruction. Note condition C is as defined for SKIP, defined previously in this table. It is not possible to have an RPT instruction placed at index 23 of the command buffer.
WEVENT1	1	0	1	1	1	0	0	0	Wait for mact_event1 to go high, and then continue execution.
WEVENT2	1	0	1	1	1	0	0	1	Wait for mact_event2 to go high, and then continue execution.
INT	1	0	1	1	1	0	1	0	Generate an IRQ_CSP_MANINT. Issues an IRQ_CSP_MANINT interrupt request.
LABEL	1	0	1	1	1	0	1	1	Set the next instruction as the start of a repeat loop. Enters the address of the next instruction into the loop-start register.
WAITX	1	0	1	1	1	1	0	0	Wait for MAC timer to overflow [X] times, where [X] is the value of register X. Each time a MAC timer overflow is detected, X is decremented. Execution continues as soon as X = 0. (If X = 0 when instruction is run, no wait is performed and execution continues directly.) An IRQ_CSP_WAIT interrupt request is generated when execution continues.

Table 23-6. Instruction Set Summary (continued)

Mnemonic	7	6	5	4	3	2	1	0	Description
RANDXY	1	0	1	1	1	1	0	1	Load the [Y] LSBs of register X with random value.
SETCMP1	1	0	1	1	1	1	1	0	Set the output csp_mact_setcmp1 high. This sets the compare value of the MAC timer to the current timer value.
INCX	1	1	0	0	0	0	0	0	Increment register X.
INCY	1	1	0	0	0	0	0	1	Increment register Y.
INCZ	1	1	0	0	0	0	1	0	Increment register Z.
DECX	1	1	0	0	0	0	1	1	Decrement register X.
DECY	1	1	0	0	0	1	0	0	Decrement register Y.
DECZ	1	1	0	0	0	1	0	1	Decrement register Z.
INCMAXY <M>	1	1	0	0	1	M2	M1	M0	Register Y ≤ min(Y + 1, M). Increment Y, but not beyond M.
Sxxx	1	1	0	1	S3	S2	S1	S0	Execute command strobe S. Send command strobe S to FFCTRL. Up to 32 command strobes are supported. In addition to the regular command strobes, two additional command strobes that only apply to the command strobe processor are supported: SNOP: Do nothing. SSTOP: Stops the command strobe processor execution and invalidates any set label. An IRQ_CSP_STOP interrupt request is issued.
ISxxx	1	1	1	0	S3	S2	S1	S0	Execute command strobe S immediately. Send command strobe S to FFCTRL immediately, bypassing the instructions in the command buffer. If the current buffer instruction is a strobe, it is delayed. In addition to the regular command strobes, two additional command strobes that only apply to the command strobe processor are supported: ISSTART: The command strobe processor starts execution at the first instruction in the command buffer. Do not issue an ISSTART instruction if the CSP is already running. ISSTOP: Stops the command strobe processor execution and invalidates any set label. An IRQ_CSP_STOP interrupt request is issued.
ISCLEAR	1	1	1	1	1	1	1	1	Clear the CSP program. Reset PC.

23.14.9 Instruction Set Definition

There are 20 basic instruction types. Furthermore, each command-strobe and immediate-strobe instruction can be divided into 16 sub-instructions, giving an effective number of 42 different instructions. The following subsections describe each instruction in detail.

NOTE: The following definitions are used in this section:

PC = CSP program counter
X = RF register **CSPX**
Y = RF register **CSPY**
Z = RF register **CSPZ**
T = RF register **CSPT**

23.14.9.1 DECZ

Function: Decrement Z
Description: The Z register is decremented by 1. An original value of 0x00 underflows to 0xFF.
Operation: Z = Z – 1

Opcode: 0xC5

7	6	5	4	3	2	1	0
1	1	0	0	0	1	0	1

23.14.9.2 DECY

Function: Decrement Y
Description: The Y register is decremented by 1. An original value of 0x00 underflows to 0xFF.
Operation: Y = Y – 1

Opcode: 0xC4

7	6	5	4	3	2	1	0
1	1	0	0	0	1	0	0

23.14.9.3 DECX

Function: Decrement X
Description: The X register is decremented by 1. An original value of 0x00 underflows to 0xFF.
Operation: X = X – 1

Opcode: 0xC3

7	6	5	4	3	2	1	0
1	1	0	0	0	0	1	1

23.14.9.4 INCZ

Function: Increment Z
Description: The X register is incremented by 1. An original value of 0xFF overflows to 0x00.
Operation: Z = Z + 1

Opcode: 0xC2

7	6	5	4	3	2	1	0
1	1	0	0	0	0	1	0

23.14.9.5 INCY

Function: Increment Y
Description: The Y register is incremented by 1. An original value of 0xFF overflows to 0x00.
Operation: Y = Y + 1

Opcode: 0xC1

7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	1

23.14.9.6 INCX

Function: Increment X
Description: The X register is incremented by 1. An original value of 0xFF overflows to 0x00.
Operation: X = X + 1

Opcode: 0xC0

7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0

23.14.9.7 INCMAXY

Function: Increment Y not greater than M.
Description: The Y register is incremented by 1 if the result is less than M; otherwise, Y register is loaded with value M.
Operation: Y = min(Y + 1, M)

Opcode: 0xC8 | M (M = 0–7)

7	6	5	4	3	2	1	0
1	1	0	0	1	M		

23.14.9.8 RANDXY

Function: Load random value into X.
Description: The [Y] LSBs of the X register are loaded with a random value. If a second RANDXY instruction is issued immediately (within 13 clock cycles) after the first, the same random value is used in both cases. If Y equals 0 or is greater than 7, then an 8-bit random value is loaded into X.
Operation: X[(Y – 1):0]: = RNG_DOUT[(Y – 1):0], X[7:Y]: = 0

Opcode: 0xBD

7	6	5	4	3	2	1	0
1	0	1	1	1	1	0	1

23.14.9.9 INT

Function: Interrupt
Description: The interrupt IRQ_CSP_INT is asserted when this instruction executes.
Operation: IRQ_CSP_INT = 1

Opcode: 0xBA

7	6	5	4	3	2	1	0
1	0	1	1	1	0	1	0

23.14.9.10 WAITX

Function: Wait for X MAC timer overflows

Description: Wait for MAC timer to overflow [X] times, where [X] is the value of register X. Each time a MAC timer overflow is detected, the value in register X decrements. Program execution continues as soon as X = 0. (If X = 0 when instruction is run, no wait is performed and execution continues directly.) An IRQ_CSP_WAIT interrupt request is generated when execution continues. **Note:** The difference compared to WAIT W is that W is a fixed value, whereas X is a register value (which could potentially be changed, such that the number of overflows does not correspond to the value of X at the time WAITX instruction is run).

Operation: X = X – 1 when MAC timer overflow = true
PC = PC while X > 0
PC = PC + 1 when X = 0

Opcode: 0xBC

7	6	5	4	3	2	1	0
1	0	1	1	1	1	0	0

23.14.9.11 SETCMP1

Function: Set the compare value of the MAC timer to the current timer value.

Description: Set the compare value of the MAC timer to the current timer value.

Operation: Csp_mact_setcmp1 = 1

Opcode: 0xBE

7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0

23.14.9.12 WAIT W

Function: Wait for W MAC timer overflows

Description: Wait until MAC timer overflows a number of times equal to the value of W. If W = 0, the instruction waits for 32 overflows. Program execution continues with the next instruction, and the interrupt flag IRQ_CSP_WT is asserted when the wait condition is true.

Operation: PC = PC while number of MAC timer overflows < W
PC = PC + 1 when number of MAC timer overflows = W

Opcode: 0x80 | W (W = 0–31)

7	6	5	4	3	2	1	0
1	0	0	W				

23.14.9.13 WEVENT1

Function: Wait until MAC timer event 1

Description: Wait until next MAC timer event. Program execution continues with the next instruction when the wait condition is true.

Operation: PC = PC while MAC timer compare = false
PC = PC + 1 when MAC timer compare = true

Opcode: 0xB8

7	6	5	4	3	2	1	0
1	0	1	1	1	0	0	0

23.14.9.14 WEVENT2

Function: Wait until MAC timer event 2

Description: Wait until next MAC timer event. Program execution continues with the next instruction when the wait condition is true.

Operation: PC = PC while MAC timer compare = false
PC = PC + 1 when MAC timer compare = true

Opcode: 0xB9

7	6	5	4	3	2	1	0
1	0	1	1	1	0	0	1

23.14.9.15 LABEL

Function: Set loop label

Description: Sets next instruction as start of loop. If the current instruction is the last instruction in the instruction memory, then the current PC is set as start of loop. If several label instructions are executed, the last label executed is the active label. Earlier labels are removed, which means that only one level of loops is supported.

Operation: LABEL: = PC + 1

Opcode: 0xBB

7	6	5	4	3	2	1	0
1	0	1	1	1	0	1	1

23.14.9.16 RPT C

Function: Conditional repeat

Description: If condition C is true, then jump to the instruction defined by the last LABEL instruction (that is, jump to start of loop). If the condition is false or if a LABEL instruction has not been executed, then execution continues from next instruction. The condition C may be negated by setting N = 1 and is described in the following table.

Condition Code C	Description	Function
000	CCA is true	CCA = 1
001	Synchronization word received and still receiving packet or synchronization word transmitted and still transmitting packet	SFD = 1
010	CPU control true	**CSPCTRL.CPU_CTRL** = 1
011	Reserved	
100	Register X = 0	X = 0
101	Register Y = 0	Y = 0
110	Register Z = 0	Z = 0

Condition Code C	Description	Function
111	RSSI is valid	**RSSI_VALID** = 1

Operation: PC = LABEL when (C XOR N) = true
PC = PC + 1 when (C XOR N) = false or LABEL = not set

Opcode: 0xA0 | N | C (N = 0, 8; C = 0–7)

7	6	5	4	3	2	1	0
1	0	1	0	N	C		

23.14.9.17 SKIP S, C

Function: Conditional skip instruction

Description: Skip S instructions on condition C (where condition C may be negated; N = 1). When condition (C XOR N) is true, skip the next S instructions, else execute the next instruction. If S = 0, re-execute the conditional jump (that is, busy loop until condition is false). Skipping past the last instruction in the command buffer results in an implicit STOP command.

Condition Code C	Description	Function
000	CCA is true	CCA = 1
001	Synchronization word received and still receiving packet or synchronization word transmitted and still transmitting packet	SFD = 1
010	CPU control true	**CSPCTRL.CPU_CTRL** = 1
011	Reserved	
100	Register X = 0	X = 0
101	Register Y = 0	Y = 0
110	Register Z = 0	Z = 0
111	RSSI is valid	**RSSI_VALID** = 1

Operation: PC = PC + S + 1 when (C XOR N) = true
PC = PC + 1 when (C XOR N) = false

Opcode: 0x00 | S | N | C

7	6	5	4	3	2	1	0
0	S			N	C		

23.14.9.18 STOP

Function: Stop program execution

Description: The SSTOP instruction stops the CSP program execution.

Operation: Stop execution

Opcode: 0xD2

7	6	5	4	3	2	1	0
1	1	0	1	0	0	1	0

23.14.9.19 SNOP

Function: No operation

Description: Operation continues at the next instruction.

Operation: PC = PC + 1

Opcode: 0xD0

7	6	5	4	3	2	1	0
1	1	0	1	0	0	0	0

23.14.9.20 SRXON

Function: Enable and calibrate frequency synthesizer for RX

Description: The SRXON instruction asserts the output FFCTL_SRXON_STRB to enable and calibrate the frequency synthesizer for RX. The instruction waits for the radio to acknowledge the command before executing the next instruction.

Operation: SRXON

Opcode: 0xD3

7	6	5	4	3	2	1	0
1	1	0	1	0	0	1	1

23.14.9.21 STXON

Function: Enable TX after calibration

Description: The STXON instruction enables TX after calibration. The instruction waits for the radio to acknowledge the command before executing the next instruction. Sets a bit in **RXENABLE** if the **SET_RXENMASK_ON_TX** bit is set.

Operation: STXON

Opcode: 0xD9

7	6	5	4	3	2	1	0
1	1	0	1	1	0	0	1

23.14.9.22 STXONCCA

Function: Enable calibration and TX if CCA indicates a clear channel

Description: The STXONCCA instruction enables TX after calibration if CCA indicates a clear channel.

Operation: STXONCCA

Opcode: 0xDA

7	6	5	4	3	2	1	0
1	1	0	1	1	0	1	0

23.14.9.23 SSAMPLECCA

Function: Sample the current CCA value to **SAMPLED_CCA**
Description: The current CCA value is written to **SAMPLED_CCA** in XREG.
Operation: SSAMPLECCA

Opcode: 0xDB

7	6	5	4	3	2	1	0
1	1	0	1	1	0	1	1

23.14.9.24 SRFOFF

Function: Disable RX or TX and frequency synthesizer.
Description: The SRFOFF instruction disables RX or TX and the frequency synthesizer.
Operation: SRFOFF

Opcode: 0xDF

7	6	5	4	3	2	1	0
1	1	0	1	1	1	1	1

23.14.9.25 SFLUSHRX

Function: Flush RX FIFO buffer and reset demodulator
Description: The SFLUSHRX instruction flushes the RX FIFO buffer and resets the demodulator. The instruction waits for the radio to acknowledge the command before executing the next instruction.
Operation: SFLUSHRX

Opcode: 0xDD

7	6	5	4	3	2	1	0
1	1	0	1	1	1	0	1

23.14.9.26 SFLUSHTX

Function: Flush TX FIFO buffer
Description: The SFLUSHTX instruction flushes the TX FIFO buffer. The instruction waits for the radio to acknowledge the command before executing the next instruction.
Operation: SFLUSHTX

Opcode: 0xDE

7	6	5	4	3	2	1	0
1	1	0	1	1	1	1	0

23.14.9.27 SACK

Function: Send acknowledge frame with pending field cleared

Description: The SACK instruction sends an acknowledge frame. The instruction waits for the radio to acknowledge the command before executing the next instruction.

Operation: SACK

Opcode: 0xD6

7	6	5	4	3	2	1	0
1	1	0	1	0	1	1	0

23.14.9.28 SACKPEND

Function: Send acknowledge frame with the pending field set

Description: The SACKPEND instruction sends an acknowledge frame with the pending field set. The instruction waits for the radio to acknowledge the command before executing the next instruction.

Operation: SACKPEND

Opcode: 0xD7

7	6	5	4	3	2	1	0
1	1	0	1	0	1	1	1

23.14.9.29 SNACK

Function: Abort sending of acknowledge frame

Description: The SACKPEND instruction aborts sending acknowldedge to the frame currently being received.

Operation: SNACK

Opcode: 0xD8

7	6	5	4	3	2	1	0
1	1	0	1	1	0	0	0

23.14.9.30 SRXMASKBITSET

Function: Set bit in **RXENABLE** register

Description: The SRXMASKBITSET instruction sets bit [5] in the **RXENABLE** register.

Operation: SRXMASKBITSET

Opcode: 0xD4

7	6	5	4	3	2	1	0
1	1	0	1	0	1	0	0

23.14.9.31 SRXMASKBITCLR

Function: Clear bit in **RXENABLE** register
Description: The SRXMASKBITCLR instruction clears bit [5] in the **RXENABLE** register.
Operation: SRXMASKBITCLR

Opcode: 0xD5

7	6	5	4	3	2	1	0
1	1	0	1	0	1	0	1

23.14.9.32 ISSTOP

Function: Stop program execution
Description: The ISSTOP instruction stops the CSP program execution and the IRQ_CSP_STOP interrupt flag is asserted.
Operation: Stop execution

Opcode: 0xE2

7	6	5	4	3	2	1	0
1	1	1	0	0	0	1	0

23.14.9.33 ISSTART

Function: Start program execution
Description: The ISSTART instruction starts the CSP program execution from first instruction written to instruction memory. Do not issue an ISSTART instruction if CSP is already running.
Operation: PC = 0, start execution

Opcode: 0xE1

7	6	5	4	3	2	1	0
1	1	1	0	0	0	0	1

23.14.9.34 ISRXON

Function: Enable and calibrate frequency synthesizer for RX
Description: The ISRXON instruction immediately enables and calibrates the frequency synthesizer for RX.
Operation: SRXON

Opcode: 0xE3

7	6	5	4	3	2	1	0
1	1	1	0	0	0	1	1

23.14.9.35 ISRXMASKBITSET

Function: Set bit in **RXENABLE**
Description: The ISRXMASKBITSET instruction immediately sets bit [5] in the **RXENABLE** register.
Operation: SRXMASKBITSET

Opcode: 0xE4

7	6	5	4	3	2	1	0
1	1	1	0	0	1	0	0

23.14.9.36 ISRXMASKBITCLR

Function: Clear bit in **RXENABLE**

Description: The ISRXMASKBITCLR instruction immediately clears bit [5] in the **RXENABLE** register.

Operation: SRXMASKBITCLR

Opcode: 0xE5

7	6	5	4	3	2	1	0
1	1	1	0	0	1	0	1

23.14.9.37 ISTXON

Function: Enable TX after calibration

Description: The ISTXON instruction immediately enables TX after calibration. The instruction waits for the radio to acknowledge the command before executing the next instruction.

Operation: STXON_STRB

Opcode: 0xE9

7	6	5	4	3	2	1	0
1	1	1	0	1	0	0	1

23.14.9.38 ISTXONCCA

Function: Enable calibration and TX if CCA indicates a clear channel

Description: The ISTXONCCA instruction immediately enables TX after calibration if CCA indicates a clear channel.

Operation: STXONCCA

Opcode: 0xEA

7	6	5	4	3	2	1	0
1	1	1	0	1	0	1	0

23.14.9.39 ISSAMPLECCA

Function: Sample the current CCA value to **SAMPLED_CCA**

Description: The current CCA value is immediately written to **SAMPLED_CCA** in XREG.

Operation: SSAMPLECCA

Opcode: 0xEB

7	6	5	4	3	2	1	0
1	1	1	0	1	0	1	1

23.14.9.40 ISRFOFF

Function: Disable RX or TX, and the frequency synthesizer.
Description: The ISRFOFF instruction immediately disables RX or TX, and the frequency synthesizer.
Operation: FFCTL_SRFOFF_STRB = 1

Opcode: 0xEF

7	6	5	4	3	2	1	0
1	1	1	0	1	1	1	1

23.14.9.41 ISFLUSHRX

Function: Flush RX FIFO buffer and reset demodulator
Description: The ISFLUSHRX instruction immediately flushes the RX FIFO buffer and resets the demodulator.
Operation: SFLUSHRX

Opcode: 0xED

7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	1

23.14.9.42 ISFLUSHTX

Function: Flush TX FIFO buffer
Description: The ISFLUSHTX instruction immediately flushes the TX FIFO buffer.
Operation: SFLUSHTX

Opcode: 0xEE

7	6	5	4	3	2	1	0
1	1	1	0	1	1	1	0

23.14.9.43 ISACK

Function: Send acknowledge frame with the pending field cleared
Description: The ISACK instruction immediately sends an acknowledge frame.
Operation: SACK

Opcode: 0xE6

7	6	5	4	3	2	1	0
1	1	1	0	0	1	1	0

23.14.9.44 ISACKPEND

Function: Send acknowledge frame with the pending field set
Description: The ISACKPEND instruction immediately sends an acknowledge frame with the pending field set. The instruction waits for the radio to receive and interpret the command before executing the next instruction.

Operation: SACKPEND

Opcode: 0xE7

7	6	5	4	3	2	1	0
1	1	1	0	0	1	1	1

23.14.9.45 ISNACK

Function: Abort sending of acknowledge frame

Description: The ISNACK instruction immediately prevents sending of an acknowledge frame to the currently received frame.

Operation: SNACK

Opcode: 0xE8

7	6	5	4	3	2	1	0
1	1	1	0	1	0	0	0

23.14.9.46 ISCLEAR

Function: Clear CSP program memory, reset program counter

Description: The ISCLEAR clears the program memory, resets the program counter, and aborts any running program. No stop interrupt is generated. The LABEL pointer is cleared. The ISCLEAR instruction must be issued twice to reset the program counter.

Operation: PC = 0, clear program memory

Opcode: 0xFF

7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1

23.15 Register Settings Update

This section contains a summary of the register settings that must be updated from their default value to have optimal RF performance.

The following settings should be set for both RX and TX. Although not all settings are necessary for both RX and TX, it is recommended for simplicity (writing one set of settings when the code is initialized).

Table 23-7. Registers That Require Update From Their Default Value

Register Name	New Value (Hex)	Description
AGCCTRL1	0x15	Adjusts AGC target value.
TXFILTCFG	0x09	Sets TX anti-aliasing filter to appropriate bandwidth.
IVCTRL	0x0B	Controls bias currents.
FSCAL1	0x01	Tune frequency calibration

23.16 Radio Registers

23.16.1 RFCORE_FFSM Registers

23.16.1.1 RFCORE_FFSM Registers Mapping Summary

This section provides information on the RFCORE_FFSM module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 23-8. RFCORE_FFSM Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
RFCORE_FFSM_SRCRESMASK0	RW	32	0x0000 0000	0x80	0x4008 8580
RFCORE_FFSM_SRCRESMASK1	RW	32	0x0000 0000	0x84	0x4008 8584
RFCORE_FFSM_SRCRESMASK2	RW	32	0x0000 0000	0x88	0x4008 8588
RFCORE_FFSM_SRCRESINDEX	RW	32	0x0000 0000	0x8C	0x4008 858C
RFCORE_FFSM_SRCEXTPENDEN0	RW	32	0x0000 0000	0x90	0x4008 8590
RFCORE_FFSM_SRCEXTPENDEN1	RW	32	0x0000 0000	0x94	0x4008 8594
RFCORE_FFSM_SRCEXTPENDEN2	RW	32	0x0000 0000	0x98	0x4008 8598
RFCORE_FFSM_SRCSHORTPENDEN0	RW	32	0x0000 0000	0x9C	0x4008 859C
RFCORE_FFSM_SRCSHORTPENDEN1	RW	32	0x0000 0000	0xA0	0x4008 85A0
RFCORE_FFSM_SRCSHORTPENDEN2	RW	32	0x0000 0000	0xA4	0x4008 85A4
RFCORE_FFSM_EXT_ADDR0	RW	32	0x0000 0000	0xA8	0x4008 85A8
RFCORE_FFSM_EXT_ADDR1	RW	32	0x0000 0000	0xAC	0x4008 85AC
RFCORE_FFSM_EXT_ADDR2	RW	32	0x0000 0000	0xB0	0x4008 85B0
RFCORE_FFSM_EXT_ADDR3	RW	32	0x0000 0000	0xB4	0x4008 85B4
RFCORE_FFSM_EXT_ADDR4	RW	32	0x0000 0000	0xB8	0x4008 85B8
RFCORE_FFSM_EXT_ADDR5	RW	32	0x0000 0000	0xBC	0x4008 85BC
RFCORE_FFSM_EXT_ADDR6	RW	32	0x0000 0000	0xC0	0x4008 85C0
RFCORE_FFSM_EXT_ADDR7	RW	32	0x0000 0000	0xC4	0x4008 85C4
RFCORE_FFSM_PAN_ID0	RW	32	0x0000 0000	0xC8	0x4008 85C8

Table 23-8. RFCORE_FFSM Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
RFCORE_FFSM_PAN_ID1	RW	32	0x0000 0000	0xCC	0x4008 85CC
RFCORE_FFSM_SHORT_ADDR0	RW	32	0x0000 0000	0xD0	0x4008 85D0
RFCORE_FFSM_SHORT_ADDR1	RW	32	0x0000 0000	0xD4	0x4008 85D4

23.16.1.2 RFCORE_FFSM Register Descriptions

RFCORE_FFSM_SRCRESMASK0

Address offset	0x80		
Physical Address	0x4008 8580	**Instance**	RFCORE_FFSM
Description	Source address matching result This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								SRCRESMASK0							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	SRCRESMASK0	Extended address matching When there is a match on entry ext_n, bits 2n and 2n + 1 are set in SRCRESMASK.	RW	0x00

RFCORE_FFSM_SRCRESMASK1

Address offset	0x84		
Physical Address	0x4008 8584	**Instance**	RFCORE_FFSM
Description	Source address matching result This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								SRCRESMASK1							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	SRCRESMASK1	Short address matching When there is a match on entry panid_n + short_n, bit n is set in SRCRESMASK.	RW	0x00

RFCORE_FFSM_SRCRESMASK2

Address offset	0x88		
Physical Address	0x4008 8588	**Instance**	RFCORE_FFSM
Description	Source address matching result This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								SRCRESMASK2							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	SRCRESMASK2	24-bit mask that indicates source address match for each individual entry in the source address table	RW	0x00

RFCORE_FFSM_SRCRESINDEX

Address offset	0x8C		
Physical Address	0x4008 858C	**Instance**	RFCORE_FFSM
Description	Source address matching result This register is stored in RAM; the reset value is undefined.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	SRCRESINDEX	The bit index of the least-significant entry (0-23 for short addresses or 0-11 for extended addresses) in SRCRESMASK, or 0x3F when there is no source match On a match, bit 5 is 0 when the match is on a short address and 1 when it is on an extended address. On a match, bit 6 is 1 when the conditions for automatic pending bit in acknowledgment have been met (see the description of SRCMATCH.AUTOPEND). The bit does not indicate if the acknowledgment is actually transmitted, and does not consider the PENDING_OR register bit and the SACK/SACKPEND/SNACK strobes.	RW	0x00

RFCORE_FFSM_SRCEXTPENDEN0

Address offset	0x90		
Physical Address	0x4008 8590	**Instance**	RFCORE_FFSM
Description	Source address matching control This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								SRCEXTPENDEN0							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	SRCEXTPENDEN0	8 LSBs of the 24-bit mask that enables or disables automatic pending for each of the 12 extended addresses. Entry n is mapped to SRCEXTPENDEN[2n]. All SRCEXTPENDEN[2n + 1] bits are don't care.	RW	0x00

RFCORE_FFSM_SRCEXTPENDEN1

Address offset	0x94		
Physical Address	0x4008 8594	**Instance**	RFCORE_FFSM
Description	Source address matching control This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								SRCEXTPENDEN1							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	SRCEXTPENDEN1	8 middle bits of the 24-bit mask that enables or disables automatic pending for each of the 12 extended addresses Entry n is mapped to SRCEXTPENDEN[2n]. All SRCEXTPENDEN[2n + 1] bits are don't care.	RW	0x00

RFCORE_FFSM_SRCEXTPENDEN2

Address offset	0x98		
Physical Address	0x4008 8598	**Instance**	RFCORE_FFSM
Description	Source address matching control This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								SRCEXTPENDEN2							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	SRCEXTPENDEN2	8 MSBs of the 24-bit mask that enables or disables automatic pending for each of the 12 extended addresses Entry n is mapped to SRCEXTPENDEN[2n]. All SRCEXTPENDEN[2n + 1] bits are don't care.	RW	0x00

RFCORE_FFSM_SRCSHORTPENDEN0

Address offset	0x9C		
Physical Address	0x4008 859C	**Instance**	RFCORE_FFSM
Description	Source address matching control This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								SRCSHORTPENDEN0							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	SRCSHORTPENDEN0	8 LSBs of the 24-bit mask that enables or disables automatic pending for each of the 24 short addresses	RW	0x00

RFCORE_FFSM_SRCSHORTPENDEN1

Address offset	0xA0		
Physical Address	0x4008 85A0	**Instance**	RFCORE_FFSM
Description	Source address matching control This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								SRCSHORTPENDEN1							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	SRCSHORTPENDEN1	8 middle bits of the 24-bit mask that enables or disables automatic pending for each of the 24 short addresses	RW	0x00

RFCORE_FFSM_SRCSHORTPENDEN2

Address offset	0xA4		
Physical Address	0x4008 85A4	**Instance**	RFCORE_FFSM
Description	Source address matching control This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								SRCSHORTPENDEN2							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	SRCSHORTPENDEN2	8 MSBs of the 24-bit mask that enables or disables automatic pending for each of the 24 short addresses	RW	0x00

RFCORE_FFSM_EXT_ADDR0

Address offset	0xA8		
Physical Address	0x4008 85A8	**Instance**	RFCORE_FFSM
Description	Local address information This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								EXT_ADDR0							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	EXT_ADDR0	EXT_ADDR[7:0] The IEEE extended address used during destination address filtering	RW	0x00

RFCORE_FFSM_EXT_ADDR1

Address offset	0xAC		
Physical Address	0x4008 85AC	**Instance**	RFCORE_FFSM
Description	Local address information This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								EXT_ADDR1							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	EXT_ADDR1	EXT_ADDR[15:8] The IEEE extended address used during destination address filtering	RW	0x00

RFCORE_FFSM_EXT_ADDR2

Address offset	0xB0		
Physical Address	0x4008 85B0	**Instance**	RFCORE_FFSM
Description	Local address information This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								EXT_ADDR2							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	EXT_ADDR2	EXT_ADDR[23:16] The IEEE extended address used during destination address filtering	RW	0x00

RFCORE_FFSM_EXT_ADDR3

Address offset	0xB4		
Physical Address	0x4008 85B4	**Instance**	RFCORE_FFSM
Description	Local address information This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								EXT_ADDR3							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	EXT_ADDR3	EXT_ADDR[31:24] The IEEE extended address used during destination address filtering	RW	0x00

RFCORE_FFSM_EXT_ADDR4

Address offset	0xB8		
Physical Address	0x4008 85B8	**Instance**	RFCORE_FFSM
Description	Local address information This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								EXT_ADDR4							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	EXT_ADDR4	EXT_ADDR[39:32] The IEEE extended address used during destination address filtering	RW	0x00

RFCORE_FFSM_EXT_ADDR5

Address offset	0xBC		
Physical Address	0x4008 85BC	**Instance**	RFCORE_FFSM
Description	Local address information This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								EXT_ADDR5							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	EXT_ADDR5	EXT_ADDR[47:40] The IEEE extended address used during destination address filtering	RW	0x00

RFCORE_FFSM_EXT_ADDR6

Address offset	0xC0		
Physical Address	0x4008 85C0	**Instance**	RFCORE_FFSM
Description	Local address information This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								EXT_ADDR6							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	EXT_ADDR6	EXT_ADDR[55:48] The IEEE extended address used during destination address filtering	RW	0x00

RFCORE_FFSM_EXT_ADDR7

Address offset	0xC4		
Physical Address	0x4008 85C4	**Instance**	RFCORE_FFSM
Description	Local address information This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								EXT_ADDR7							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	EXT_ADDR7	EXT_ADDR[63:56] The IEEE extended address used during destination address filtering	RW	0x00

RFCORE_FFSM_PAN_ID0

Address offset	0xC8		
Physical Address	0x4008 85C8	**Instance**	RFCORE_FFSM
Description	Local address information This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PAN_ID0							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	PAN_ID0	PAN_ID[7:0] The PAN ID used during destination address filtering	RW	0x00

RFCORE_FFSM_PAN_ID1

Address offset	0xCC		
Physical Address	0x4008 85CC	**Instance**	RFCORE_FFSM
Description	Local address information This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PAN_ID1							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	PAN_ID1	PAN_ID[15:8] The PAN ID used during destination address filtering	RW	0x00

RFCORE_FFSM_SHORT_ADDR0

Address offset	0xD0		
Physical Address	0x4008 85D0	**Instance**	RFCORE_FFSM
Description	Local address information This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								SHORT_ADDR0							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	SHORT_ADDR0	SHORT_ADDR[7:0] The short address used during destination address filtering	RW	0x00

RFCORE_FFSM_SHORT_ADDR1

Address offset	0xD4		
Physical Address	0x4008 85D4	**Instance**	RFCORE_FFSM
Description	Local address information This register is stored in RAM; the reset value is undefined.		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								SHORT_ADDR1							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	SHORT_ADDR1	SHORT_ADDR[15:8] The short address used during destination address filtering	RW	0x00

23.16.2 RFCORE_XREG Registers

23.16.2.1 RFCORE_XREG Registers Mapping Summary

This section provides information on the RFCORE_XREG module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 23-9. RFCORE_XREG Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
RFCORE_XREG_FRMFILT0	RW	32	0x0000 000D	0x000	0x4008 8600
RFCORE_XREG_FRMFILT1	RW	32	0x0000 0078	0x004	0x4008 8604
RFCORE_XREG_SRCMATCH	RW	32	0x0000 0007	0x008	0x4008 8608
RFCORE_XREG_SRCSHORTEN0	RW	32	0x0000 0000	0x00C	0x4008 860C
RFCORE_XREG_SRCSHORTEN1	RW	32	0x0000 0000	0x010	0x4008 8610
RFCORE_XREG_SRCSHORTEN2	RW	32	0x0000 0000	0x014	0x4008 8614
RFCORE_XREG_SRCEXTEN0	RW	32	0x0000 0000	0x018	0x4008 8618
RFCORE_XREG_SRCEXTEN1	RW	32	0x0000 0000	0x01C	0x4008 861C
RFCORE_XREG_SRCEXTEN2	RW	32	0x0000 0000	0x020	0x4008 8620
RFCORE_XREG_FRMCTRL0	RW	32	0x0000 0040	0x024	0x4008 8624
RFCORE_XREG_FRMCTRL1	RW	32	0x0000 0001	0x028	0x4008 8628
RFCORE_XREG_RXENABLE	RO	32	0x0000 0000	0x02C	0x4008 862C
RFCORE_XREG_RXMASKSET	RW	32	0x0000 0000	0x030	0x4008 8630
RFCORE_XREG_RXMASKCLR	RW	32	0x0000 0000	0x034	0x4008 8634
RFCORE_XREG_FREQTUNE	RW	32	0x0000 000F	0x038	0x4008 8638
RFCORE_XREG_FREQCTRL	RW	32	0x0000 000B	0x03C	0x4008 863C
RFCORE_XREG_TXPOWER	RW	32	0x0000 00F5	0x040	0x4008 8640

Table 23-9. RFCORE_XREG Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
RFCORE_XREG_TXCTRL	RW	32	0x0000 0069	0x044	0x4008 8644
RFCORE_XREG_FSMSTAT0	RO	32	0x0000 0000	0x048	0x4008 8648
RFCORE_XREG_FSMSTAT1	RO	32	0x0000 0000	0x04C	0x4008 864C
RFCORE_XREG_FIFOPCTRL	RW	32	0x0000 0040	0x050	0x4008 8650
RFCORE_XREG_FSMCTRL	RW	32	0x0000 0001	0x054	0x4008 8654
RFCORE_XREG_CCACTRL0	RW	32	0x0000 00F8	0x058	0x4008 8658
RFCORE_XREG_CCACTRL1	RW	32	0x0000 001A	0x05C	0x4008 865C
RFCORE_XREG_RSSI	RO	32	0x0000 0080	0x060	0x4008 8660
RFCORE_XREG_RSSISTAT	RO	32	0x0000 0000	0x064	0x4008 8664
RFCORE_XREG_RXFIRST	RO	32	0x0000 0000	0x068	0x4008 8668
RFCORE_XREG_RXFIFOCNT	RO	32	0x0000 0000	0x06C	0x4008 866C
RFCORE_XREG_TXFIFOCNT	RO	32	0x0000 0000	0x070	0x4008 8670
RFCORE_XREG_RXFIRST_PTR	RO	32	0x0000 0000	0x074	0x4008 8674
RFCORE_XREG_RXLAST_PTR	RO	32	0x0000 0000	0x078	0x4008 8678
RFCORE_XREG_RXP1_PTR	RO	32	0x0000 0000	0x07C	0x4008 867C
RFCORE_XREG_TXFIRST_PTR	RO	32	0x0000 0000	0x084	0x4008 8684
RFCORE_XREG_TXLAST_PTR	RO	32	0x0000 0000	0x088	0x4008 8688
RFCORE_XREG_RFIRQM0	RW	32	0x0000 0000	0x08C	0x4008 868C
RFCORE_XREG_RFIRQM1	RW	32	0x0000 0000	0x090	0x4008 8690
RFCORE_XREG_RFERRM	RW	32	0x0000 0000	0x094	0x4008 8694
RFCORE_XREG_RFRND	RO	32	0x0000 0000	0x09C	0x4008 869C
RFCORE_XREG_MDMCTRL0	RW	32	0x0000 0085	0x0A0	0x4008 86A0
RFCORE_XREG_MDMCTRL1	RW	32	0x0000 0014	0x0A4	0x4008 86A4
RFCORE_XREG_FREQEST	RO	32	0x0000 0000	0x0A8	0x4008 86A8
RFCORE_XREG_RXCTRL	RW	32	0x0000 003F	0x0AC	0x4008 86AC

Table 23-9. RFCORE_XREG Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
RFCORE_XREG_FSCTRL	RW	32	0x0000 005A	0x0B0	0x4008 86B0
RFCORE_XREG_FSCAL0	RW	32	0x0000 0024	0x0B4	0x4008 86B4
RFCORE_XREG_FSCAL1	RW	32	0x0000 0000	0x0B8	0x4008 86B8
RFCORE_XREG_FSCAL2	RW	32	0x0000 0020	0x0BC	0x4008 86BC
RFCORE_XREG_FSCAL3	RW	32	0x0000 002A	0x0C0	0x4008 86C0
RFCORE_XREG_AGCCTRL0	RW	32	0x0000 005F	0x0C4	0x4008 86C4
RFCORE_XREG_AGCCTRL1	RW	32	0x0000 0011	0x0C8	0x4008 86C8
RFCORE_XREG_AGCCTRL2	RW	32	0x0000 00FE	0x0CC	0x4008 86CC
RFCORE_XREG_AGCCTRL3	RW	32	0x0000 002E	0x0D0	0x4008 86D0
RFCORE_XREG_ADCTEST0	RW	32	0x0000 0010	0x0D4	0x4008 86D4
RFCORE_XREG_ADCTEST1	RW	32	0x0000 000E	0x0D8	0x4008 86D8
RFCORE_XREG_ADCTEST2	RW	32	0x0000 0003	0x0DC	0x4008 86DC
RFCORE_XREG_MDMTEST0	RW	32	0x0000 0075	0x0E0	0x4008 86E0
RFCORE_XREG_MDMTEST1	RW	32	0x0000 0008	0x0E4	0x4008 86E4
RFCORE_XREG_DACTEST0	RW	32	0x0000 0000	0x0E8	0x4008 86E8
RFCORE_XREG_DACTEST1	RW	32	0x0000 0000	0x0EC	0x4008 86EC
RFCORE_XREG_DACTEST2	RW	32	0x0000 0028	0x0F0	0x4008 86F0
RFCORE_XREG_ATEST	RW	32	0x0000 0000	0x0F4	0x4008 86F4
RFCORE_XREG_PTEST0	RW	32	0x0000 0000	0x0F8	0x4008 86F8
RFCORE_XREG_PTEST1	RW	32	0x0000 0000	0x0FC	0x4008 86FC
RFCORE_XREG_CSPPROG__0 - CSPPROG__23	RO	32	0x0000 00D0	0x100-0x15C	0x4008 8700-0x4008 875C
RFCORE_XREG_CSPCTRL	RW	32	0x0000 0000	0x180	0x4008 8780
RFCORE_XREG_CSPSTAT	RO	32	0x0000 0000	0x184	0x4008 8784
RFCORE_XREG_CSPX	RO	32	0x0000 0000	0x188	0x4008 8788
RFCORE_XREG_CSPY	RO	32	0x0000 0000	0x18C	0x4008 878C

Table 23-9. RFCORE_XREG Register Summary (continued)

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
RFCORE_XREG_CSPZ	RO	32	0x0000 0000	0x190	0x4008 8790
RFCORE_XREG_CSPT	RO	32	0x0000 00FF	0x194	0x4008 8794
RFCORE_XREG_RFC_OBS_CTRL0	RW	32	0x0000 0000	0x1AC	0x4008 87AC
RFCORE_XREG_RFC_OBS_CTRL1	RW	32	0x0000 0000	0x1B0	0x4008 87B0
RFCORE_XREG_RFC_OBS_CTRL2	RW	32	0x0000 0000	0x1B4	0x4008 87B4
RFCORE_XREG_TXFILTCFG	RW	32	0x0000 000F	0x1E8	0x4008 87E8

23.16.2.2 RFCORE_XREG Register Descriptions

RFCORE_XREG_FRMFILT0

Address offset	0x000		
Physical Address	0x4008 8600	**Instance**	RFCORE_XREG
Description	The frame filtering function rejects unintended frames as specified by IEEE 802.15.4, section 7.5.6.2, third filtering level. In addition, it provides filtering on: - The eight different frame types (see the FRMFILT1 register) - The reserved bits in the frame control field (FCF) The function is controlled by: - The FRMFILT0 and FRMFILT1 registers - The PAN_ID, SHORT_ADDR, and EXT_ADDR values in RAM		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED	RESERVED			MAX_FRAME_VERSION		PAN_COORDINATOR	FRAME_FILTER_EN

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	RESERVED	This bit field is reserved.	RO	0
6:4	RESERVED	This bit field is reserved.	RW	0x0
3:2	MAX_FRAME_VERSION	Used for filtering on the frame version field of the frame control field (FCF) If FCF[13:12] (the frame version subfield) is higher than MAX_FRAME_VERSION[1:0] and frame filtering is enabled, the frame is rejected.	RW	0x3
1	PAN_COORDINATOR	Should be set high when the device is a PAN coordinator, to accept frames with no destination address (as specified in Section 7.5.6.2 in IEEE 802.15.4) 0: Device is not a PAN coordinator 1: Device is a PAN coordinator	RW	0

Bits	Field Name	Description	Type	Reset
0	FRAME_FILTER_EN	Enables frame filtering When this bit is set, the radio performs frame filtering as specified in section 7.5.6.2 of IEEE 802.15.4(b), third filtering level. FRMFILT0[6:1] and FRMFILT1[7:1], together with the local address information, define the behavior of the filtering algorithm. 0: Frame filtering off. (FRMFILT0[6:1], FRMFILT1[7:1] and SRCMATCH[2:0] are don't care.) 1: Frame filtering on.	RW	1

RFCORE_XREG_FRMFILT1

Address offset	0x004		
Physical Address	0x4008 8604	**Instance**	RFCORE_XREG
Description	The frame filtering function rejects unintended frames as specified by IEEE 802.15.4, section 7.5.6.2, third filtering level. In addition, it provides filtering on: - The eight different frame types (see the FRMFILT1 register) - The reserved bits in the frame control field (FCF) The function is controlled by: - The FRMFILT0 and FRMFILT1 registers - The PAN_ID, SHORT_ADDR, and EXT_ADDR values in RAM		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED	ACCEPT_FT_3_MAC_CMD	ACCEPT_FT_2_ACK	ACCEPT_FT_1_DATA	ACCEPT_FT_0_BEACON	MODIFY_FT_FILTER		RESERVED

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	RESERVED	This bit field is reserved.	RW	0
6	ACCEPT_FT_3_MAC_CMD	Defines whether MAC command frames are accepted or not. MAC command frames have frame type = 011. 0: Reject 1: Accept	RW	1
5	ACCEPT_FT_2_ACK	Defines whether acknowledgment frames are accepted or not. Acknowledgement frames have frame type = 010. 0: Reject 1: Accept	RW	1
4	ACCEPT_FT_1_DATA	Defines whether data frames are accepted or not. Data frames have frame type = 001. 0: Reject 1: Accept	RW	1
3	ACCEPT_FT_0_BEACON	Defines whether beacon frames are accepted or not. Beacon frames have frame type = 000. 0: Reject 1: Accept	RW	1
2:1	MODIFY_FT_FILTER	These bits are used to modify the frame type field of a received frame before frame type filtering is performed. The modification does not influence the frame that is written to the RX FIFO. 00: Leave the frame type as it is. 01: Invert MSB of the frame type. 10: Set MSB of the frame type to 0. 11: Set MSB of the frame type to 1.	RW	0x0

Bits	Field Name	Description	Type	Reset
0	RESERVED	This bit field is reserved.	RW	0

RFCORE_XREG_SRCMATCH

Address offset	0x008		
Physical Address	0x4008 8608	**Instance**	RFCORE_XREG
Description	Source address matching and pending bits		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED					PEND_DATAREQ_ONLY	AUTOPEND	SRC_MATCH_EN

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:3	RESERVED	This bit field is reserved.	RO	0x00
2	PEND_DATAREQ_ONLY	When this bit is set, the AUTOPEND function also requires that the received frame is a DATA REQUEST MAC command frame.	RW	1
1	AUTOPEND	Automatic acknowledgment pending flag enable When a frame is received, the pending bit in the (possibly) returned acknowledgment is set automatically when the following conditions are met: - FRMFILT.FRAME_FILTER_EN is set. - SRCMATCH.SRC_MATCH_EN is set. - SRCMATCH.AUTOPEND is set. - The received frame matches the current SRCMATCH.PEND_DATAREQ_ONLY setting. - The received source address matches at least one source match table entry, which is enabled in SHORT_ADDR_EN and SHORT_PEND_EN or in EXT_ADDR_EN and EXT_PEND_EN.	RW	1
0	SRC_MATCH_EN	Source address matching enable (requires that FRMFILT.FRAME_FILTER_EN = 1)	RW	1

RFCORE_XREG_SRCSHORTEN0

Address offset	0x00C		
Physical Address	0x4008 860C	**Instance**	RFCORE_XREG
Description	Short address matching		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	SHORT_ADDR_EN	7:0 part of the 24-bit word SHORT_ADDR_EN that enables or disables source address matching for each of the 24 short address table entries Optional safety feature: To ensure that an entry in the source matching table is not used while it is being updated, set the corresponding SHORT_ADDR_EN bit to 0 while updating.	RW	0x00

RFCORE_XREG_SRCSHORTEN1

Address offset	0x010		
Physical Address	0x4008 8610	**Instance**	RFCORE_XREG
Description	Short address matching		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	SHORT_ADDR_EN	15:8 part of the 24-bit word SHORT_ADDR_EN See description of SRCSHORTEN0.SHORT_ADDR_EN.	RW	0x00

RFCORE_XREG_SRCSHORTEN2

Address offset	0x014		
Physical Address	0x4008 8614	**Instance**	RFCORE_XREG
Description	Short address matching		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	SHORT_ADDR_EN	23:16 part of the 24-bit word SHORT_ADDR_EN See description of SRCSHORTEN0.SHORT_ADDR_EN.	RW	0x00

RFCORE_XREG_SRCEXTEN0

Address offset	0x018		
Physical Address	0x4008 8618	**Instance**	RFCORE_XREG
Description	Extended address matching		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	EXT_ADDR_EN	7:0 part of the 24-bit word EXT_ADDR_EN that enables or disables source address matching for each of the 12 extended address table entries Write access: Extended address enable for table entry n (0 to 11) is mapped to EXT_ADDR_EN[2n]. All EXT_ADDR_EN[2n + 1] bits are read only. Read access: Extended address enable for table entry n (0 to 11) is mapped to EXT_ADDR_EN[2n] and EXT_ADDR_EN[2n + 1]. Optional safety feature: To ensure that an entry in the source matching table is not used while it is being updated, set the corresponding EXT_ADDR_EN bit to 0 while updating.	RW	0x00

RFCORE_XREG_SRCEXTEN1

Address offset	0x01C		
Physical Address	0x4008 861C	**Instance**	RFCORE_XREG
Description	Extended address matching		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	EXT_ADDR_EN	15:8 part of the 24-bit word EXT_ADDR_EN See description of SRCEXTEN0.EXT_ADDR_EN.	RW	0x00

RFCORE_XREG_SRCEXTEN2

Address offset	0x020		
Physical Address	0x4008 8620	**Instance**	RFCORE_XREG
Description	Extended address matching		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	EXT_ADDR_EN	23:16 part of the 24-bit word EXT_ADDR_EN See description of SRCEXTEN0.EXT_ADDR_EN.	RW	0x00

RFCORE_XREG_FRMCTRL0

Address offset	0x024		
Physical Address	0x4008 8624	**Instance**	RFCORE_XREG
Description	Frame handling		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								APPEND_DATA_MODE	AUTOCRC	AUTOACK	ENERGY_SCAN	RX_MODE		TX_MODE	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	APPEND_DATA_MODE	When AUTOCRC = 0: Don't care When AUTOCRC = 1: 0: RSSI + The CRC_OK bit and the 7-bit correlation value are appended at the end of each received frame 1: RSSI + The CRC_OK bit and the 7-bit SRCRESINDEX are appended at the end of each received frame.	RW	0

Bits	Field Name	Description	Type	Reset
6	AUTOCRC	In TX 1: A CRC-16 (ITU-T) is generated in hardware and appended to the transmitted frame. There is no need to write the last 2 bytes to TXBUF. 0: No CRC-16 is appended to the frame. The last 2 bytes of the frame must be generated manually and written to TXBUF (if not, TX_UNDERFLOW occurs). In RX 1: The CRC-16 is checked in hardware, and replaced in the RXFIFO by a 16-bit status word which contains a CRC OK bit. The status word is controllable through APPEND_DATA_MODE. 0: The last 2 bytes of the frame (CRC-16 field) are stored in the RX FIFO. The CRC (if any) must be done manually. This setting does not influence acknowledgment transmission (including AUTOACK).	RW	1
5	AUTOACK	Defines whether the radio automatically transmits acknowledge frames or not. When autoack is enabled, all frames that are accepted by address filtering, have the acknowledge request flag set, and have a valid CRC are automatically acknowledged 12 symbol periods after being received. 0: Autoack disabled 1: Autoack enabled	RW	0
4	ENERGY_SCAN	Defines whether the RSSI register contains the most-recent signal strength or the peak signal strength since the energy scan was enabled. 0: Most-recent signal strength 1: Peak signal strength	RW	0
3:2	RX_MODE	Set RX modes. 00: Normal operation, use RX FIFO 01: Receive serial mode, output received data on to IOC; infinite RX 10: RX FIFO looping ignore overflow in RX FIFO; infinite reception 11: Same as normal operation except that symbol search is disabled. Can be used for RSSI or CCA measurements when finding symbol is not desired.	RW	0x0
1:0	TX_MODE	Set test modes for TX. 00: Normal operation, transmit TX FIFO 01: Reserved, should not be used 10: TX FIFO looping ignore underflow in TX FIFO and read cyclic; infinite transmission 11: Send random data from CRC; infinite transmission	RW	0x0

RFCORE_XREG_FRMCTRL1

Address offset	0x028		
Physical Address	0x4008 8628	**Instance**	RFCORE_XREG
Description	Frame handling		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								Reserved					PENDING_OR	IGNORE_TX_UNDERF	SET_RXENMASK_ON_TX

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RW	0x00 0000
7:3	Reserved	Reserved. Read as 0.	RW	0x00
2	PENDING_OR	Defines whether the pending data bit in outgoing acknowledgment frames is always set to 1 or controlled by the main FSM and the address filtering 0: Pending data bit is controlled by main FSM and address filtering. 1: Pending data bit is always 1.	RW	0
1	IGNORE_TX_UNDERF	Defines whether or not TX underflow should be ignored 0: Normal TX operation. TX underflow is detected and TX is aborted if underflow occurs. 1: Ignore TX underflow. Transmit the number of bytes given by the frame-length field.	RW	0
0	SET_RXENMASK_ON_TX	Defines whether STXON sets bit 6 in the RXENABLE register or leaves it unchanged 0: Does not affect RXENABLE 1: Sets bit 6 in RXENABLE. Used for backward compatibility with the CC2420.	RW	1

RFCORE_XREG_RXENABLE

Address offset	0x02C		
Physical Address	0x4008 862C	**Instance**	RFCORE_XREG
Description	RX enabling		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	RXENMASK	RXENABLE enables the receiver. A nonzero value in this register causes FFCTRL to enable the receiver when in idle, after transmission and after acknowledgement transmission. The following strobes can modify RXENMASK: SRXON: Set bit 7 in RXENMASK. STXON: Set bit 6 in RXENMASK if SET_RXENMASK_ON_TX = 1. SRFOFF: Clears all bits in RXENMASK. SRXMASKBITSET: Set bit 5 in RXENMASK. SRXMASKBITCLR: Clear bit 5 in RXENMASK. There could be conflicts between the CSP and xreg_bus write operations if both operations try to modify RXENMASK simultaneously. To handle the case of simultaneous access to RXENMASK the following rules apply: - If the two sources agree (they modify different parts of the register) both of their requests to modify RXENMASK are processed. - If both operations try to modify the mask simultaneously, bus write operations to RXMASKSET and RXMASKCLR have priority over the CSP. This situation must be avoided.	RO	0x00

RFCORE_XREG_RXMASKSET

Address offset	0x030		
Physical Address	0x4008 8630	**Instance**	RFCORE_XREG
Description	RX enabling		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	RXENMASKSET	When written, the written data is ORed with the RXENMASK and stored in RXENMASK.	RW	0x00

RFCORE_XREG_RXMASKCLR

Address offset	0x034		
Physical Address	0x4008 8634	**Instance**	RFCORE_XREG
Description	RX disabling		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	RXENMASKCLR	When written, the written data is inverted and ANDed with the RXENMASK and stored in RXENMASK. For example, if 1 is written to one or more bit positions in this register, the corresponding bits are cleared in RXENMASK.	RW	0x00

RFCORE_XREG_FREQTUNE

Address offset	0x038		
Physical Address	0x4008 8638	**Instance**	RFCORE_XREG
Description	Crystal oscillator frequency tuning		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED				XOSC32M_TUNE			

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:4	RESERVED	This bit field is reserved.	RO	0x0
3:0	XOSC32M_TUNE	Tune crystal oscillator The default setting 1111 leaves the XOSC untuned. Changing the setting from the default setting (1111) switches in extra capacitance to the oscillator, effectively lowering the XOSC frequency. Hence, a higher setting gives a higher frequency.	RW	0xF

RFCORE_XREG_FREQCTRL

Address offset	0x03C		
Physical Address	0x4008 863C	**Instance**	RFCORE_XREG
Description	Controls the RF frequency		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	RESERVED	This bit field is reserved.	RO	0
6:0	FREQ	Frequency control word The frequency word in FREQ[6:0] is an offset value from 2394 (fRF = FREQ[6 0] + 2394). The RF-frequency is specified from 2405 to 2480 MHz in 1-MHz steps; hence, the only valid settings for FREQ[6:0] are 11 to 86 (11 + 2394 = 2405 and 86 + 2394 = 2480). The device supports the frequency range from 2394 to 2507 MHz. Consequently, the usable settings for FREQ[6:0] are 0 to 113. Settings outside of the usable range (114 to 127) give a frequency of 2507 MHz. IEEE 802.15.4-2006 specifies a frequency range from 2405 MHz to 2480 MHz with 16 channels 5 MHz apart. The channels are numbered 11 through 26. For an IEEE 802.15.4-2006 compliant system, the only valid settings are thus FREQ[6:0] = 11 + 5 (channel number - 11).	RW	0x0B

RFCORE_XREG_TXPOWER

Address offset	0x040		
Physical Address	0x4008 8640	**Instance**	RFCORE_XREG
Description	Controls the output power		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:4	PA_POWER	PA power control	NA	0xF
3:0	PA_BIAS	PA bias control	RW	0x5

RFCORE_XREG_TXCTRL

Address offset	0x044		
Physical Address	0x4008 8644	**Instance**	RFCORE_XREG
Description	Controls the TX settings		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED	DAC_CURR			DAC_DC		TXMIX_CURRENT	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	RESERVED	This bit field is reserved.	RO	0
6:4	DAC_CURR	Change the current in the DAC.	RW	0x6
3:2	DAC_DC	Adjusts the DC level to the TX mixer.	RW	0x2
1:0	TXMIX_CURRENT	Transmit mixers core current Current increases with increasing setting.	RW	0x1

RFCORE_XREG_FSMSTAT0

Address offset	0x048		
Physical Address	0x4008 8648	**Instance**	RFCORE_XREG
Description	Radio status register		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								CAL_DONE	CAL_RUNNING	FSM_FFCTRL_STATE					

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	CAL_DONE	Frequency synthesis calibration has been performed since the last time the FS was turned on.	RO	0
6	CAL_RUNNING	Frequency synthesis calibration status 0: Calibration is complete or not started. 1: Calibration is in progress.	RO	0
5:0	FSM_FFCTRL_STATE	Gives the current state of the FIFO and frame control (FFCTRL) finite state-machine.	RO	0x00

RFCORE_XREG_FSMSTAT1

Address offset	0x04C		
Physical Address	0x4008 864C	**Instance**	RFCORE_XREG
Description	Radio status register		
Type	RO		

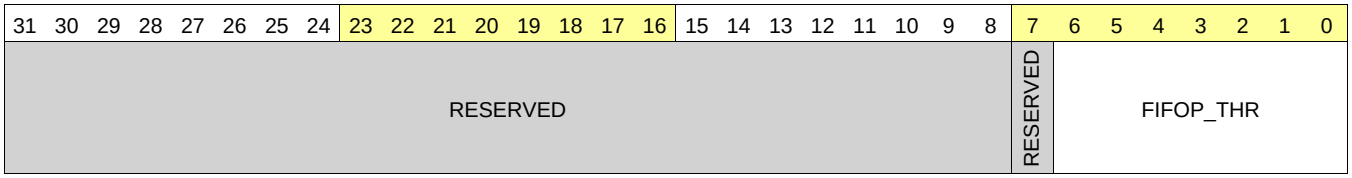
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								FIFO	FIFOP	SFD	CCA	SAMPLED_CCA	LOCK_STATUS	TX_ACTIVE	RX_ACTIVE

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	FIFO	FIFO is high when there is data in the RX FIFO. FIFO is low during RX FIFO overflow.	RO	0
6	FIFOP	FIFOP is set high when there are at more than FIFOP_THR bytes of data in the RX FIFO that has passed frame filtering. FIFOP is set high when there is at least one complete frame in the RX FIFO. FIFOP is high during RX FIFO overflow.	RO	0

Bits	Field Name	Description	Type	Reset
5	SFD	In TX 0: When a complete frame with SFD was sent or no SFD was sent 1: SFD was sent. In RX 0: When a complete frame was received or no SFD was received 1: SFD was received.	RO	0
4	CCA	Clear channel assessment Dependent on CCA_MODE settings. See CCACTRL1 for details.	RO	0
3	SAMPLED_CCA	Contains a sampled value of the CCA The value is updated when a SSAMPLECCA or STXONCCA strobe is issued.	RO	0
2	LOCK_STATUS	1 when PLL is in lock; otherwise 0	RO	0
1	TX_ACTIVE	Status signal Active when FFC is in one of the transmit states	RO	0
0	RX_ACTIVE	Status signal Active when FFC is in one of the receive states	RO	0

RFCORE_XREG_FIFOPCTRL

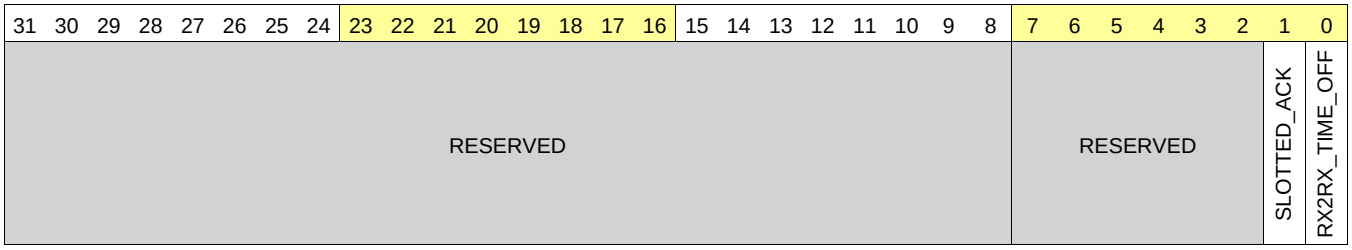
Address offset	0x050		
Physical Address	0x4008 8650	**Instance**	RFCORE_XREG
Description	FIFOP threshold		
Type	RW		



Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	RESERVED	This bit field is reserved.	RO	0
6:0	FIFOP_THR	Threshold used when generating FIFOP signal	RW	0x40

RFCORE_XREG_FSMCTRL

Address offset	0x054		
Physical Address	0x4008 8654	**Instance**	RFCORE_XREG
Description	FSM options		
Type	RW		



Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:2	RESERVED	This bit field is reserved.	RO	0x00
1	SLOTTED_ACK	Controls timing of transmission of acknowledge frames 0: The acknowledge frame is sent 12 symbol periods after the end of the received frame which requests the aknowledge. 1: The acknowledge frame is sent at the first backoff-slot boundary more than 12 symbol periods after the end of the received frame which requests the aknowledge.	RW	0
0	RX2RX_TIME_OFF	Defines whether or not a 12-symbol time-out should be used after frame reception has ended. 0: No time-out 1: 12-symbol-period time-out	RW	1

RFCORE_XREG_CCACTRL0

Address offset	0x058		
Physical Address	0x4008 8658	**Instance**	RFCORE_XREG
Description	CCA threshold		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	CCA_THR	Clear-channel-assessment threshold value, signed 2's-complement number for comparison with the RSSI. The unit is 1 dB, offset is 73dB The CCA signal goes high when the received signal is below this value. The CCA signal is available on the CCA pin and in the FSMSTAT1 register. The value must never be set lower than CCA_HYST - 128 to avoid erroneous behavior of the CCA signal. Note: The reset value translates to an input level of approximately -32 - 73 = -105 dBm, which is well below the sensitivity limit. This means that the CCA signal never indicates a clear channel. This register should be updated to 0xF8, which translates to an input level of about -8 - 73 = -81 dBm.	RW	0xF8

RFCORE_XREG_CCACTRL1

Address offset	0x05C		
Physical Address	0x4008 865C	**Instance**	RFCORE_XREG
Description	Other CCA Options		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:5	RESERVED	This bit field is reserved.	RO	0x0
4:3	CCA_MODE	00: CCA always set to 1 01: CCA = 1 when RSSI < CCA_THR - CCA_HYST; CCA = 0 when RSSI >= CCA_THR 10: CCA = 1 when not receiving a frame, else CCA = 0 11: CCA = 1 when RSSI < CCA_THR - CCA_HYST and not receiving a frame; CCA = 0 when RSSI >= CCA_THR or when receiving a frame	RW	0x3
2:0	CCA_HYST	Sets the level of CCA hysteresis. Unsigned values given in dB	RW	0x2

RFCORE_XREG_RSSI

Address offset	0x060		
Physical Address	0x4008 8660	**Instance**	RFCORE_XREG
Description	RSSI status register		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	RSSI_VAL	RSSI estimate on a logarithmic scale, signed number on 2's complement Unit is 1 dB, offset is 73dB. The RSSI value is averaged over eight symbol periods. The RSSI_VALID status bit should be checked before reading RSSI_VAL for the first time. The reset value of -128 also indicates that the RSSI value is invalid.	RO	0x80

RFCORE_XREG_RSSISTAT

Address offset	0x064		
Physical Address	0x4008 8664	**Instance**	RFCORE_XREG
Description	RSSI valid status register		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:1	RESERVED	This bit field is reserved.	RO	0x00
0	RSSI_VALID	RSSI value is valid. Occurs eight symbol periods after entering RX.	RO	0

RFCORE_XREG_RXFIRST

Address offset	0x068		
Physical Address	0x4008 8668	**Instance**	RFCORE_XREG
Description	First byte in RX FIFO		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	DATA	First byte of the RX FIFO Note: Reading this register does not modify the contents of the FIFO.	RO	0x00

RFCORE_XREG_RXFIFOCNT

Address offset	0x06C		
Physical Address	0x4008 866C	**Instance**	RFCORE_XREG
Description	Number of bytes in RX FIFO		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	RXFIFOCNT	Number of bytes in the RX FIFO (unsigned integer)	RO	0x00

RFCORE_XREG_TXFIFOCNT

Address offset	0x070		
Physical Address	0x4008 8670	**Instance**	RFCORE_XREG
Description	Number of bytes in TX FIFO		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	TXFIFOCNT	Number of bytes in the TX FIFO (unsigned integer)	RO	0x00

RFCORE_XREG_RXFIRST_PTR

Address offset	0x074		
Physical Address	0x4008 8674	**Instance**	RFCORE_XREG
Description	RX FIFO pointer		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	RXFIRST_PTR	RAM address offset of the first byte in the RX FIFO	RO	0x00

RFCORE_XREG_RXLAST_PTR

Address offset	0x078		
Physical Address	0x4008 8678	**Instance**	RFCORE_XREG
Description	RX FIFO pointer		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	RXLAST_PTR	RAM address offset of the last byte + 1 byte in the RX FIFO	RO	0x00

RFCORE_XREG_RXP1_PTR

Address offset	0x07C		
Physical Address	0x4008 867C	**Instance**	RFCORE_XREG
Description	RX FIFO pointer		
Type	RO		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RXP1_PTR							

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	RXP1_PTR	RAM address offset of the first byte of the first frame in the RX FIFO	RO	0x00

RFCORE_XREG_TXFIRST_PTR

Address offset	0x084		
Physical Address	0x4008 8684	**Instance**	RFCORE_XREG
Description	TX FIFO pointer		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	TXFIRST_PTR	RAM address offset of the next byte to be transmitted from the TX FIFO	RO	0x00

RFCORE_XREG_TXLAST_PTR

Address offset	0x088		
Physical Address	0x4008 8688	**Instance**	RFCORE_XREG
Description	TX FIFO pointer		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	TXLAST_PTR	RAM address offset of the last byte + 1 byte of the TX FIFO	RO	0x00

RFCORE_XREG_RFIRQM0

Address offset	0x08C		
Physical Address	0x4008 868C	**Instance**	RFCORE_XREG
Description	RF interrupt masks		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	RFIRQM	Bit mask is masking out interrupt sources. Bit position: 7: RXMASKZERO 6: RXPKTDONE 5: FRAME_ACCEPTED 4: SRC_MATCH_FOUND 3: SRC_MATCH_DONE 2: FIFOP 1: SFD 0: ACT_UNUSED	RW	0x00

RFCORE_XREG_RFIRQM1

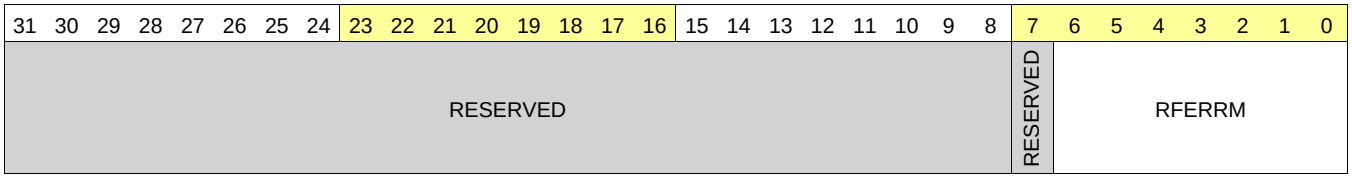
Address offset	0x090		
Physical Address	0x4008 8690	**Instance**	RFCORE_XREG
Description	RF interrupt masks		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:6	RESERVED	This bit field is reserved.	RO	0x0
5:0	RFIRQM	Bit mask is masking out interrupt sources. Bit position: 5: CSP_WAIT 4: CSP_STOP 3: CSP_MANINT 2: RF_IDLE 1: TXDONE 0: TXACKDONE	RW	0x00

RFCORE_XREG_RFERRM

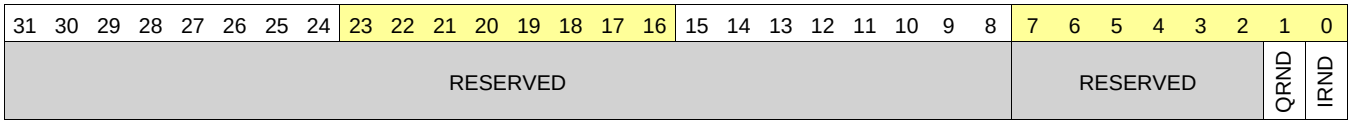
Address offset	0x094		
Physical Address	0x4008 8694	**Instance**	RFCORE_XREG
Description	RF error interrupt mask		
Type	RW		



Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	RESERVED	This bit field is reserved.	RO	0
6:0	RFERRM	Bit mask is masking out interrupt sources. Bit position: 6: STROBE_ERR 5: TXUNDERF 4: TXOVERF 3: RXUNDERF 2: RXOVERF 1: RXABO 0: NLOCK	RW	0x00

RFCORE_XREG_RFRND

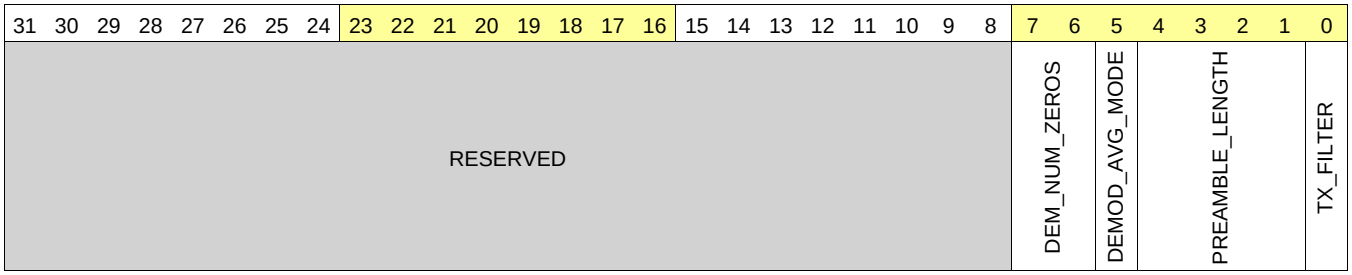
Address offset	0x09C		
Physical Address	0x4008 869C	**Instance**	RFCORE_XREG
Description	Random data		
Type	RO		



Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:2	RESERVED	This bit field is reserved.	RO	0x00
1	QRND	Random bit from the Q channel of the receiver	RO	0
0	IRND	Random bit from the I channel of the receiver	RO	0

RFCORE_XREG_MDMCTRL0

Address offset	0x0A0		
Physical Address	0x4008 86A0	**Instance**	RFCORE_XREG
Description	Controls modem		
Type	RW		



Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:6	DEM_NUM_ZEROS	Sets how many zero symbols must be detected before the sync word when searching for sync. Only one zero symbol is required to have a correlation value above the correlation threshold set in the MDMCTRL1 register. 00: Reserved 01: 1 zero symbol 10: 2 zero symbols 11: 3 zero symbols	RW	0x2
5	DEMOD_AVG_MODE	Defines the behavior or the frequency offset averaging filter. 0: Lock average level after preamble match. Restart frequency offset calibration when searching for the next frame. 1: Continuously update average level.	RW	0
4:1	PREAMBLE_LENGTH	The number of preamble bytes (two zero-symbols) to be sent in TX mode before the SFD, encoded in steps of 2 symbols (1 byte). The reset value of 2 is compliant with IEEE 802.15.4. 0000: 2 leading-zero bytes 0001: 3 leading-zero bytes 0010: 4 leading-zero bytes ... 1111: 17 leading-zero bytes	RW	0x2
0	TX_FILTER	Defines the kind of TX filter that is used. The normal TX filter is as defined by the IEEE 802.15.4 standard. Extra filtering may be applied to lower the out-of-band emissions. 0: Normal TX filtering 1: Enable extra filtering	RW	1

RFCORE_XREG_MDMCTRL1

Address offset	0x0A4		
Physical Address	0x4008 86A4	**Instance**	RFCORE_XREG
Description	Controls modem		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:6	RESERVED	This bit field is reserved.	RO	0x0
5	CORR_THR_SFD	Defines requirements for SFD detection: 0: The correlation value of one of the zero symbols of the preamble must be above the correlation threshold. 1: The correlation value of one zero symbol of the preamble and both symbols in the SFD must be above the correlation threshold.	RW	0
4:0	CORR_THR	Demodulator correlator threshold value, required before SFD search. Threshold value adjusts how the receiver synchronizes to data from the radio. If the threshold is set too low, sync can more easily be found on noise. If set too high, the sensitivity is reduced, but sync is not likely to be found on noise. In combination with DEM_NUM_ZEROS, the system can be tuned so sensitivity is high with less sync found on noise.	RW	0x14

RFCORE_XREG_FREQEST

Address offset	0x0A8		
Physical Address	0x4008 86A8	**Instance**	RFCORE_XREG
Description	Estimated RF frequency offset		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	FREQEST	Signed 2's-complement value. Contains an estimate of the frequency offset between carrier and the receiver LO. The offset frequency is FREQEST x 7800 Hz. DEM_AVG_MODE controls when this estimate is updated. If DEM_AVG_MODE = 0, it is updated until sync is found. Then the frequency offset estimate is frozen until the end of the received frame. If DEM_AVG_MODE = 1, it is updated as long as the demodulator is enabled. To calculate the correct value, one must use an offset (FREQEST_offset), which can be found in the device data sheet. Real FREQEST value = FREQEST - FREQEST_offset.	RO	0x00

RFCORE_XREG_RXCTRL

Address offset	0x0AC		
Physical Address	0x4008 86AC	**Instance**	RFCORE_XREG
Description	Tune receive section		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED		GBIAS_LNA2_REF		GBIAS_LNA_REF		MIX_CURRENT	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:6	RESERVED	This bit field is reserved.	RO	0x0
5:4	GBIAS_LNA2_REF	Adjusts front-end LNA2/mixer PTAT current output (from M = 3 to M = 6), default: M = 5	RW	0x3
3:2	GBIAS_LNA_REF	Adjusts front-end LNA PTAT current output (from M = 3 to M = 6), default: M = 5	RW	0x3
1:0	MIX_CURRENT	Control of the output current from the receiver mixers The current increases with increasing setting set.	RW	0x3

RFCORE_XREG_FSCTRL

Address offset	0x0B0		
Physical Address	0x4008 86B0	**Instance**	RFCORE_XREG
Description	Tune frequency synthesizer		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PRE_CURRENT		LODIV_BUF_CURRENT_TX		LODIV_BUF_CURRENT_RX		LODIV_CURRENT	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:6	PRE_CURRENT	Prescaler current setting	RW	0x1
5:4	LODIV_BUF_CURRENT_TX	Adjusts current in mixer and PA buffers Used when TX_ACTIVE = 1	RW	0x1
3:2	LODIV_BUF_CURRENT_RX	Adjusts current in mixer and PA buffers Used when TX_ACTIVE = 0	RW	0x2
1:0	LODIV_CURRENT	Adjusts divider currents, except mixer and PA buffers	RW	0x2

RFCORE_XREG_FSCAL0

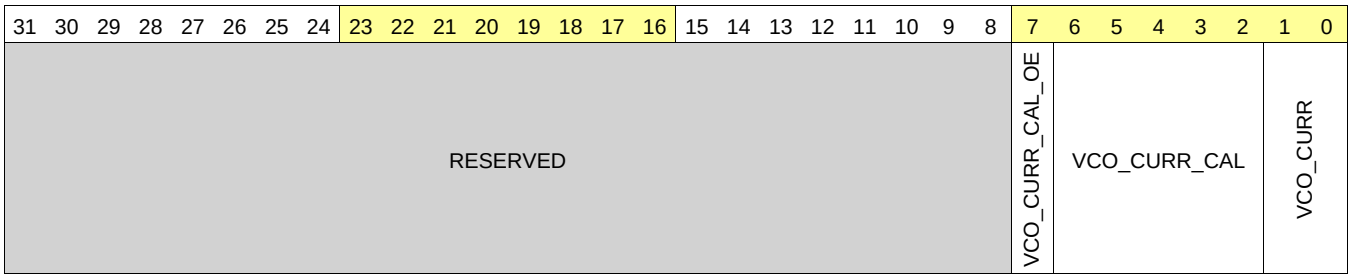
Address offset	0x0B4		
Physical Address	0x4008 86B4	**Instance**	RFCORE_XREG
Description	Tune frequency calibration		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								VCO_CURR_COMP_EN_OV	CHP_DISABLE	CHP_CURRENT				BW_BOOST_MODE	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	VCO_CURR_COMP_EN_OV	Force on the current comparator in the VCO. This signal is ORed with the signal coming from the calibration module.	RW	0
6	CHP_DISABLE	Set this bit to manually disable charge pump by masking the up and down pulses from the phase detector.	RW	0
5:2	CHP_CURRENT	Digital bit vector defining the charge-pump output current on an exponential scale If FFC_BW_BOOST = 0, the read value is the value stored in CHP_CURRENT. If FFC_BW_BOOST = 1, the read value is CHP_CURRENT + 4. If the addition causes overflow, the signal is saturated.	RW	0x9
1:0	BW_BOOST_MODE	Control signal Defines the synthesizer boost mode 00: No BW_BOOST 01: BW_BOOST is high during calibration and approximately 30 us into the settling. 10: BW_BOOST is always on (or high). 11: Reserved	RW	0x0

RFCORE_XREG_FSCAL1

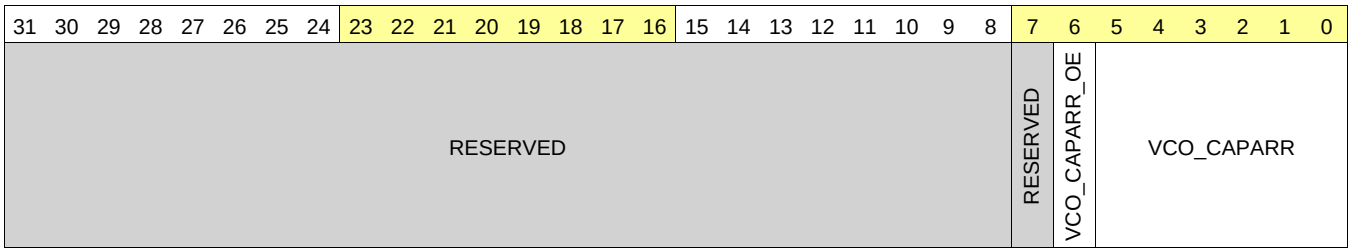
Address offset	0x0B8		
Physical Address	0x4008 86B8	**Instance**	RFCORE_XREG
Description	Tune frequency calibration		
Type	RW		



Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	VCO_CURR_CAL_OE	Override current calibration	RW	0
6:2	VCO_CURR_CAL	Calibration result Override value if VCO_CURR_CAL_OE = 1	RW	0x00
1:0	VCO_CURR	Defines current in VCO core Sets the multiplier between calibrated current and VCO current.	RW	0x0

RFCORE_XREG_FSCAL2

Address offset	0x0BC		
Physical Address	0x4008 86BC	**Instance**	RFCORE_XREG
Description	Tune frequency calibration		
Type	RW		



Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	RESERVED	This bit field is reserved.	RO	0
6	VCO_CAPARR_OE	Override the calibration result with the value from VCO_CAPARR[5:0].	RW	0
5:0	VCO_CAPARR	VCO capacitor array setting Programmed during calibration Override value when VCO_CAPARR_OE = 1	RW	0x20

RFCORE_XREG_FSCAL3

Address offset	0x0C0		
Physical Address	0x4008 86C0	**Instance**	RFCORE_XREG
Description	Tune frequency calibration		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED	VCO_DAC_EN_OV	VCO_VC_DAC				VCO_CAPARR_CAL_CTRL	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	RESERVED	This bit field is reserved.	RO	0
6	VCO_DAC_EN_OV	Enables the VCO DAC when 1	RW	0
5:2	VCO_VC_DAC	Bit vector for programming varactor control voltage from VC DAC	RW	0xA
1:0	VCO_CAPARR_CAL_CTRL	Calibration accuracy setting for the cap_array calibration part of the calibration 00: 80 XOSC periods 01: 100 XOSC periods 10: 125 XOSC periods 11: 250 XOSC periods	RW	0x2

RFCORE_XREG_AGCCTRL0

Address offset	0x0C4		
Physical Address	0x4008 86C4	**Instance**	RFCORE_XREG
Description	AGC dynamic range control		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	RESERVED	This bit field is reserved.	RO	0
6	AGC_DR_XTND_EN	0: The AGC performs no adjustment of attenuation in the AAF. 1: The AGC adjusts the gain in the AAF to achieve extra dynamic range for the receiver.	RW	1
5:0	AGC_DR_XTND_THR	If the measured error between the AGC reference magnitude and the actual magnitude in dB is larger than this threshold, the extra attenuation is enabled in the front end. This threshold must be set higher than 0x0C. This feature is enabled by AGC_DR_XTND_EN.	RW	0x1F

RFCORE_XREG_AGCCTRL1

Address offset	0x0C8		
Physical Address	0x4008 86C8	**Instance**	RFCORE_XREG
Description	AGC reference level		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:6	RESERVED	This bit field is reserved.	RO	0x0
5:0	AGC_REF	Target value for the AGC control loop, given in 1-dB steps	RW	0x11

RFCORE_XREG_AGCCTRL2

Address offset	0x0CC		
Physical Address	0x4008 86CC	**Instance**	RFCORE_XREG
Description	AGC gain override		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								LNA1_CURRENT		LNA2_CURRENT		LNA3_CURRENT		LNA_CURRENT_OE	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:6	LNA1_CURRENT	Overrride value for LNA 1 Used only when LNA_CURRENT_OE = 1 When read, this register returns the current applied gain setting. 00: 0-dB gain (reference level) 01: 3-dB gain 10: Reserved 11: 6-dB gain	RW	0x3
5:3	LNA2_CURRENT	Overrride value for LNA 2 Used only when LNA_CURRENT_OE = 1 When read, this register returns the current applied gain setting. 000: 0-dB gain (reference level) 001: 3-dB gain 010: 6-dB gain 011: 9-dB gain 100: 12-dB gain 101: 15-dB gain 110: 18-dB gain 111: 21-dB gain	RW	0x7
2:1	LNA3_CURRENT	Overrride value for LNA 3 Used only when LNA_CURRENT_OE = 1 When read, this register returns the current applied gain setting. 00: 0-dB gain (reference level) 01: 3-dB gain 10: 6-dB gain 11: 9-dB gain	RW	0x3
0	LNA_CURRENT_OE	Write 1 to override the AGC LNA current setting with the values above (LNA1_CURRENT, LNA2_CURRENT, and LNA3_CURRENT).	RW	0

RFCORE_XREG_AGCCTRL3

Address offset	0x0D0		
Physical Address	0x4008 86D0	**Instance**	RFCORE_XREG
Description	AGC control		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED	AGC_SETTLE_WAIT			AGC_WIN_SIZE		AAF_RP	AAF_RP_OE

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	RESERVED	This bit field is reserved.	RO	0
6:5	AGC_SETTLE_WAIT	Timing for AGC to wait for analog gain to settle after a gain change. During this period, the energy measurement in the AGC is paused. 00: 15 periods 01: 20 periods 10: 25 periods 11: 30 periods	RW	0x1
4:3	AGC_WIN_SIZE	Window size for the accumulate-and-dump function in the AGC. 00: 16 samples 01: 32 samples 10: 64 samples 11: 128 samples	RW	0x1
2:1	AAF_RP	Overrides the control signals of the AGC to AAF when AAF_RP_OE = 1. When read, it returns the applied signal to the AAF. 00: 9-dB attenuation in AAF 01: 6-dB attenuation in AAF 10: 3-dB attenuation in AAF 11: 0-dB attenuation in AAF (reference level)	RW	0x3
0	AAF_RP_OE	Override the AAF control signals of the AGC with the values stored in AAF_RP.	RW	0

RFCORE_XREG_ADCTEST0

Address offset	0x0D4		
Physical Address	0x4008 86D4	**Instance**	RFCORE_XREG
Description	ADC tuning		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								ADC_VREF_ADJ		ADC_QUANT_ADJ		ADC_GM_ADJ			ADC_DAC2_EN

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:6	ADC_VREF_ADJ	Quantizer threshold control for test and debug	RW	0x0
5:4	ADC_QUANT_ADJ	Quantizer threshold control for test and debug	RW	0x1
3:1	ADC_GM_ADJ	Gm-control for test and debug	RW	0x0
0	ADC_DAC2_EN	Enables DAC2 for enhanced ADC stability	RW	0

RFCORE_XREG_ADCTEST1

Address offset	0x0D8		
Physical Address	0x4008 86D8	**Instance**	RFCORE_XREG
Description	ADC tuning		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								ADC_TEST_CTRL				ADC_C2_ADJ		ADC_C3_ADJ	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:4	ADC_TEST_CTRL	ADC test mode selector	RW	0x0
3:2	ADC_C2_ADJ	Used to adjust capacitor values in ADC	RW	0x3
1:0	ADC_C3_ADJ	Used to adjust capacitor values in ADC	RW	0x2

RFCORE_XREG_ADCTEST2

Address offset	0x0DC		
Physical Address	0x4008 86DC	**Instance**	RFCORE_XREG
Description	ADC tuning		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED	ADC_TEST_MODE			AAF_RS		ADC_FF_ADJ		ADC_DAC_ROT	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	RESERVED	This bit field is reserved.	RO	0
6:5	ADC_TEST_MODE	Test mode to enable output of ADC data from demodulator. When enabled, raw ADC data is clocked out on the GPIO pins. 00: Test mode disabled 01: Data from the I and Q ADCs are output (data rate 76 MHz) 10: Data from the I ADC is output. Two and two ADC samples grouped (data rate 38 MHz) 11: Data from the Q ADC is output. Two and two ADC samples grouped (data rate 38 MHz)	RW	0x0
4:3	AAF_RS	Controls series resistance of AAF	RW	0x0

Bits	Field Name	Description	Type	Reset
2:1	ADC_FF_ADJ	Adjust feed forward	RW	0x1
0	ADC_DAC_ROT	Control of DAC DWA scheme 0 = DWA (scrambling) disabled 1 = DWA enabled	RW	1

RFCORE_XREG_MDMTEST0

Address offset	0x0E0		
Physical Address	0x4008 86E0	**Instance**	RFCORE_XREG
Description	Test register for modem		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								TX_TONE				DC_WIN_SIZE		DC_BLOCK_MODE	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:4	TX_TONE	Enables the possibility to transmit a baseband tone by picking samples from the sine tables with a controllable phase step between the samples. The step size is controlled by TX_TONE. If MDMTEST1.MOD_IF is 0, the tone is superpositioned on the modulated data, effectively giving modulation with an IF. If MDMTEST1.MOD_IF is 1, only the tone is transmitted. 0000: -6 MHz 0001: -4 MHz 0010: -3 MHz 0011: -2 MHz 0100: -1 MHz 0101: -500 kHz 0110: -4 kHz 0111: 0 1000: 4 kHz 1001: 500 kHz 1010: 1 MHz 1011: 2 MHz 1100: 3 MHz 1101: 4 MHz 1110: 6 MHz Others: Reserved	RW	0x7
3:2	DC_WIN_SIZE	Controls the numbers of samples to be accumulated between each dump of the accumulate-and-dump filter used in DC removal 00: 32 samples 01: 64 samples 10: 128 samples 11: 256 samples	RW	0x1
1:0	DC_BLOCK_MODE	Selects the mode of operation 00: The input signal to the DC blocker is passed to the output without any attempt to remove DC. 01: Enable DC cancellation. Normal operation 10: Freeze estimates of DC when sync is found. Resume estimating DC when searching for the next frame. 11: Reserved	RW	0x1

RFCORE_XREG_MDMTEST1

Address offset	0x0E4		
Physical Address	0x4008 86E4	**Instance**	RFCORE_XREG
Description	Test Register for Modem		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED		USEMIRROR_IF	MOD_IF	RAMP_AMP	RFC_SNIFF_EN	RESERVED	LOOPBACK_EN

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:6	RESERVED	This bit field is reserved.	RO	0x0
5	USEMIRROR_IF	0: Use the normal IF frequency (MDMTEST0.TX_TONE) for automatic IF compensation of channel frequency on TX. 1: Use mirror IF frequency for automatic compensation of channel frequency on TX.	RW	0
4	MOD_IF	0: Modulation is performed at an IF set by MDMTEST0.TX_TONE. The tone set by MDMTEST0.TX_TONE is superimposed on the data. 1: Modulate a tone set by MDMTEST0.TX_TONE. A tone is transmitted with frequency set by MDMTEST0.TX_TONE.	RW	0
3	RAMP_AMP	1: Enable ramping of DAC output amplitude during startup and finish. 0: Disable ramping of DAC output amplitude.	RW	1
2	RFC_SNIFF_EN	0: Packet sniffer module disabled 1: Packet sniffer module enabled. The received and transmitted data can be observed on GPIO pins.	RW	0
1	RESERVED	This bit field is reserved.	RW	0
0	LOOPBACK_EN	Enables loopback of modulated data into the receiver chain 0: An STXCAL instruction calibrates for TX. Use STXON to continue to active TX. 1: An STXCAL instruction enables the loopback mode.	RW	0

RFCORE_XREG_DACTEST0

Address offset	0x0E8		
Physical Address	0x4008 86E8	**Instance**	RFCORE_XREG
Description	DAC override value		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	DAC_Q_O	Q-branch DAC override value when DAC_SRC = 001 If DAC_SRC is set to be ADC data, CORDIC magnitude, or channel filtered data, then DAC_Q_O controls the part of the word in question that is actually multiplexed to the DAC, as described below. 000111: Bits 7:0 001000: Bits 8:1 001001: Bits 9:2 ... If an invalid setting is chosen, the DAC outputs only zeros (minimum value).	RW	0x00

RFCORE_XREG_DACTEST1

Address offset	0x0EC		
Physical Address	0x4008 86EC	**Instance**	RFCORE_XREG
Description	DAC override value		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	DAC_I_O	I-branch DAC override value when DAC_SRC = 001 If DAC_SRC is set to be ADC data, CORDIC magnitude, channel filtered data, or DC filtered data, then DAC_I_O controls the part of the word in question that is actually multiplexed to the DAC as described below. 000111: Bits 7:0 001000: Bits 8:1 001001: Bits 9:2 ... If an invalid setting is chosen, then the DAC outputs only zeros (minimum value).	RW	0x00

RFCORE_XREG_DACTEST2

Address offset	0x0F0		
Physical Address	0x4008 86F0	**Instance**	RFCORE_XREG
Description	DAC test setting		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED		DAC_DEM_EN		DAC_CASC_CTRL		DAC_SRC	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:6	RESERVED	This bit field is reserved.	RO	0x0
5	DAC_DEM_EN	Enable and disable dynamic element matching Drives RFR_DAC_DEM_EN	RW	1

Bits	Field Name	Description	Type	Reset
4:3	DAC_CASC_CTRL	Adjustment of output stage Drives RFR_DAC_CASC_CTRL	RW	0x1
2:0	DAC_SRC	The TX DACs data source is selected by DAC_SRC according to: 000: Normal operation (from modulator) 001: The DAC_I_O and DAC_Q_O override values 010: ADC data after decimation, magnitude controlled by DAC_I_O and DAC_Q_O 011: I/Q after decimation, channel and DC filtering, magnitude controlled by DAC_I_O and DAC_Q_O 100: CORDIC magnitude output and front-end gain is output, magnitude controlled by DAC_I_O and DAC_Q_O 101: RSSI I output on the I DAC 111: Reserved	RW	0x0

RFCORE_XREG_ATEST

Address offset	0x0F4		
Physical Address	0x4008 86F4	**Instance**	RFCORE_XREG
Description	Analog test control		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RW	0x00 0000
7:6	RESERVED	This bit field is reserved.	RO	0x0
5:0	ATEST_CTRL	Controls the analog test mode: 00 0000: Disabled 00 0001: Enables the temperature sensor (see also the CCTEST_TR0 register description). Other values reserved.	RW	0x00

RFCORE_XREG_PTEST0

Address offset	0x0F8		
Physical Address	0x4008 86F8	**Instance**	RFCORE_XREG
Description	Override power-down register		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								PRE_PD	CHP_PD	ADC_PD	DAC_PD	LNA_PD		TXMIX_PD	AAF_PD

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	PRE_PD	Prescaler power-down signal When PD_OVERRIDE = 1	RO	0
6	CHP_PD	Charge pump power-down signal When PD_OVERRIDE = 1	RW	0
5	ADC_PD	ADC power-down signal When PD_OVERRIDE = 1	RW	0

Bits	Field Name	Description	Type	Reset
4	DAC_PD	DAC power-down signal When PD_OVERRIDE = 1	RW	0
3:2	LNA_PD	Low-noise amplifier power-down signal Defines LNA/mixer power-down modes: 00: Power up 01: LNA off, mixer/regulator on 10: LNA/mixer off, regulator on 11: PD When PD_OVERRIDE = 1	RW	0x0
1	TXMIX_PD	Transmit mixer power-down signal When PD_OVERRIDE = 1	RW	0
0	AAF_PD	Antialiasing filter power-down signal When PD_OVERRIDE = 1	RW	0

RFCORE_XREG_PTEST1

Address offset	0x0FC		
Physical Address	0x4008 86FC	**Instance**	RFCORE_XREG
Description	Override power-down register		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED				PD_OVERRIDE	PA_PD	VCO_PD	LODIV_PD

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:4	RESERVED	This bit field is reserved.	RO	0x0
3	PD_OVERRIDE	Override enabling and disabling of various modules (for debug and testing only) It is impossible to override hard-coded BIAS_PD[1:0] depenancy.	RW	0
2	PA_PD	Power amplifier power-down signal When PD_OVERRIDE = 1	RW	0
1	VCO_PD	VCO power-down signal When PD_OVERRIDE = 1	RW	0
0	LODIV_PD	LO power-down signal When PD_OVERRIDE = 1	RW	0

RFCORE_XREG_CSPPROG__0 - CSPPROG__23

Address offset	0x100-0x15C in 0x4 byte increments		
Physical Address	0x4008 8700- 0x4008 875C	**Instance**	RFCORE_XREG
Description	CSP program		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	CSP_INSTR	Byte N of the CSP program memory	RO	0xD0

RFCORE_XREG_CSPCTRL

Address offset	0x180		
Physical Address	0x4008 8780	**Instance**	RFCORE_XREG
Description	CSP control bit		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:1	RESERVED	This bit field is reserved.	RO	0x00
0	MCU_CTRL	CSP MCU control input	RW	0

RFCORE_XREG_CSPSTAT

Address offset	0x184		
Physical Address	0x4008 8784	**Instance**	RFCORE_XREG
Description	CSP status register		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:6	RESERVED	This bit field is reserved.	RO	0x0
5	CSP_RUNNING	1: CSP is running. 0: CSP is idle.	RO	0
4:0	CSP_PC	CSP program counter	RO	0x00

RFCORE_XREG_CSPX

Address offset	0x188		
Physical Address	0x4008 8788	**Instance**	RFCORE_XREG
Description	CSP X data register		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	CSPX	Used by CSP instructions WAITX, RANDXY, INCX, DECX, and conditional instructions.	RO	0x00

RFCORE_XREG_CSPY

Address offset	0x18C		
Physical Address	0x4008 878C	**Instance**	RFCORE_XREG
Description	CSP Y data register		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	CSPY	Used by CSP instructions RANDXY, INCY, DECY, and conditional instructions.	RO	0x00

RFCORE_XREG_CSPZ

Address offset	0x190		
Physical Address	0x4008 8790	**Instance**	RFCORE_XREG
Description	CSP Z data register		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	CSPZ	Used by CSP instructions INCZ, DECZ, and conditional instructions.	RO	0x00

RFCORE_XREG_CSPT

Address offset	0x194		
Physical Address	0x4008 8794	**Instance**	RFCORE_XREG
Description	CSP T data register		
Type	RO		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	CSPT	Content is decremented each time the MAC Timer overflows while the CSP program is running. The SCP program stops when decremented to 0. Setting CSPT = 0xFF prevents the register from being decremented.	RO	0xFF

RFCORE_XREG_RFC_OBS_CTRL0

Address offset	0x1AC		
Physical Address	0x4008 87AC	**Instance**	RFCORE_XREG
Description	RF observation mux control		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED	RFC_OBS_POL0	RFC_OBS_MUX0					

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RW	0x00 0000
7	RESERVED	This bit field is reserved.	RO	0
6	RFC_OBS_POL0	The signal chosen by RFC_OBS_MUX0 is XORed with this bit.	RW	0
5:0	RFC_OBS_MUX0	Controls which observable signal from RF Core is to be muxed out to rfc_obs_sigs[0]. 00 0000: 0 - Constant value 00 0001: 1 - Constant value 00 1000: rfc_sniff_data - Data from packet sniffer. Sample data on rising edges of sniff_clk. 00 1001: rfc_sniff_clk - 250kHz clock for packet sniffer data. 00 1100: rssi_valid - Pin is high when the RSSI value has been updated at least once since RX was started. Cleared when leaving RX. 00 1101: demod_cca - Clear channel assessment. See FSMSTAT1 register for details on how to configure the behavior of this signal. 00 1110: sampled_cca - A sampled version of the CCA bit from demodulator. The value is updated whenever a SSAMPLECCA or STXONCCA strobe is issued. 00 1111: sfd_sync - Pin is high when a SFD has been received or transmitted. Cleared when leaving RX/TX respectively. Not to be confused with the SFD exception. 01 0000: tx_active - Indicates that FFCTRL is in one of the TX states. Active-high. Note: This signal might have glitches, because it has no output flip-flop and is based on the current state register of the FFCTRL FSM. 01 0001: rx_active - Indicates that FFCTRL is in one of the RX states. Active-high. Note: This signal might have glitches, because it has no output flip-flop and is based on the current state register of the FFCTRL FSM. 01 0010: ffctrl_fifo - Pin is high when one or more bytes are in the RXFIFO. Low during RXFIFO overflow. 01 0011: ffctrl_fifop - Pin is high when the number of bytes in the RXFIFO exceeds the programmable threshold or at least one complete frame is in the RXFIFO. Also high during RXFIFO overflow. Not to be confused with the FIFOP exception. 01 0100: packet_done - A complete frame has been received. I.e., the number of bytes set by the frame-length field has been received. 01 0110: rfc_xor_rand_i_q - XOR between I and Q random outputs. Updated at 8 MHz. 01 0111: rfc_rand_q - Random data output from the Q channel of the receiver. Updated at 8 MHz. 01 1000: rfc_rand_i - Random data output from the I channel of the receiver. Updated at 8 MHz 01 1001: lock_status - 1 when PLL is in lock, otherwise 0 10 1000: pa_pd - Power amplifier power-down signal 10 1010: lna_pd - LNA power-down signal Others: Reserved	RW	0x00

RFCORE_XREG_RFC_OBS_CTRL1

Address offset	0x1B0		
Physical Address	0x4008 87B0	**Instance**	RFCORE_XREG
Description	RF observation mux control		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED	RFC_OBS_POL1	RFC_OBS_MUX1					

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RW	0x00 0000
7	RESERVED	This bit field is reserved.	RO	0
6	RFC_OBS_POL1	The signal chosen by RFC_OBS_MUX1 is XORed with this bit.	RW	0
5:0	RFC_OBS_MUX1	Controls which observable signal from RF Core is to be muxed out to rfc_obs_sigs[1]. See description of RFC_OBS_CTRL0 for details.	RW	0x00

RFCORE_XREG_RFC_OBS_CTRL2

Address offset	0x1B4		
Physical Address	0x4008 87B4	**Instance**	RFCORE_XREG
Description	RF observation mux control		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED	RFC_OBS_POL2	RFC_OBS_MUX2					

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	RESERVED	This bit field is reserved.	RO	0
6	RFC_OBS_POL2	The signal chosen by RFC_OBS_MUX2 is XORed with this bit.	RW	0
5:0	RFC_OBS_MUX2	Controls which observable signal from RF Core is to be muxed out to rfc_obs_sigs[2]. See description of RFC_OBS_CTRL0 for details.	RW	0x00

RFCORE_XREG_TXFILTCFG

Address offset	0x1E8		
Physical Address	0x4008 87E8	**Instance**	RFCORE_XREG
Description	TX filter configuration		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:4	RESERVED	This bit field is reserved.	RO	0x0
3:0	FC	Drives signal rfr_txfilt_fc	RW	0xF

23.16.3 RFCORE_SFR Registers

23.16.3.1 RFCORE_SFR Registers Mapping Summary

This section provides information on the RFCORE_SFR module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 23-10. RFCORE_SFR Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
RFCORE_SFR_RFDATA	RW	32	0x0000 0000	0x28	0x4008 8828
RFCORE_SFR_RFERRF	RW	32	0x0000 0000	0x2C	0x4008 882C
RFCORE_SFR_RFIRQF1	RW	32	0x0000 0000	0x30	0x4008 8830
RFCORE_SFR_RFIRQF0	RW	32	0x0000 0000	0x34	0x4008 8834
RFCORE_SFR_RFST	RW	32	0x0000 00D0	0x38	0x4008 8838

23.16.3.2 RFCORE_SFR Register Descriptions

RFCORE_SFR_RFDATA

Address offset	0x28		
Physical Address	0x4008 8828	**Instance**	RFCORE_SFR
Description	The TX FIFO and RX FIFO may be accessed through this register. Data is written to the TX FIFO when writing to the RFD register. Data is read from the RX FIFO when the RFD register is read. The XREG registers RXFIFOCNT and TXFIFOCNT provide information on the amount of data in the FIFOs. The FIFO contents can be cleared by issuing SFLUSHRX and SFLUSHTX.		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	RFD	Data written to the register is written to the TX FIFO. When reading this register, data from the RX FIFO is read.	RW	0x00

RFCORE_SFR_RFERRF

Address offset	0x2C		
Physical Address	0x4008 882C	**Instance**	RFCORE_SFR
Description	RF error interrupt flags		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED	STROBEERR	TXUNDERF	TXOVERF	RXUNDERF	RXOVERF	RXABO	NLOCK

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	RESERVED	This bit field is reserved.	RO	0
6	STROBEERR	A command strobe was issued when it could not be processed. Triggered if trying to disable the radio when it is already disabled, or when trying to do a SACK, SACKPEND, or SNACK command when not in active RX. 0: No interrupt pending 1: Interrupt pending	RW	0
5	TXUNDERF	TX FIFO underflowed. 0: No interrupt pending 1: Interrupt pending	RW	0
4	TXOVERF	TX FIFO overflowed. 0: No interrupt pending 1: Interrupt pending	RW	0
3	RXUNDERF	RX FIFO underflowed. 0: No interrupt pending 1: Interrupt pending	RW	0
2	RXOVERF	RX FIFO overflowed. 0: No interrupt pending 1: Interrupt pending	RW	0
1	RXABO	Reception of a frame was aborted. 0: No interrupt pending 1: Interrupt pending	RW	0
0	NLOCK	The frequency synthesizer failed to achieve lock after time-out, or lock is lost during reception. The receiver must be restarted to clear this error situation. 0: No interrupt pending 1: Interrupt pending	RW	0

RFCORE_SFR_RFIRQF1

Address offset	0x30		
Physical Address	0x4008 8830	**Instance**	RFCORE_SFR
Description	RF interrupt flags		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED		CSP_WAIT	CSP_STOP	CSP_MANINT	RFIDLE	TXDONE	TXACKDONE

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:6	RESERVED	This bit field is reserved.	RO	0x0
5	CSP_WAIT	Execution continued after a wait instruction in CSP. 0: No interrupt pending 1: Interrupt pending	RW	0
4	CSP_STOP	CSP has stopped program execution. 0: No interrupt pending 1: Interrupt pending	RW	0
3	CSP_MANINT	Manual interrupt generated from CSP 0: No interrupt pending 1: Interrupt pending	RW	0
2	RFIDLE	Radio state-machine has entered the IDLE state. 0: No interrupt pending 1: Interrupt pending	RW	0

Bits	Field Name	Description	Type	Reset
1	TXDONE	A complete frame has been transmitted. 0: No interrupt pending 1: Interrupt pending	RW	0
0	TXACKDONE	An acknowledgement frame has been completely transmitted. 0: No interrupt pending 1: Interrupt pending	RW	0

RFCORE_SFR_RFIRQF0

Address offset	0x34		
Physical Address	0x4008 8834	**Instance**	RFCORE_SFR
Description	RF interrupt flags		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RXMASKZERO	RXPKTDONE	FRAME_ACCEPTED	SRC_MATCH_FOUND	SRC_MATCH_DONE	FIFOP	SFD	ACT_UNUSED

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	RXMASKZERO	The RXENABLE register has gone from a nonzero state to an all-zero state. 0: No interrupt pending 1: Interrupt pending	RW	0
6	RXPKTDONE	A complete frame has been received. 0: No interrupt pending 1: Interrupt pending	RW	0
5	FRAME_ACCEPTED	Frame has passed frame filtering. 0: No interrupt pending 1: Interrupt pending	RW	0
4	SRC_MATCH_FOUND	Source match is found. 0: No interrupt pending 1: Interrupt pending	RW	0
3	SRC_MATCH_DONE	Source matching is complete. 0: No interrupt pending 1: Interrupt pending	RW	0
2	FIFOP	The number of bytes in the RX FIFO is greater than the threshold. Also raised when a complete frame is received, and when a packet is read out completely and more complete packets are available. 0: No interrupt pending 1: Interrupt pending	RW	0
1	SFD	SFD has been received or transmitted. 0: No interrupt pending 1: Interrupt pending	RW	0
0	ACT_UNUSED	Reserved 0: No interrupt pending 1: Interrupt pending	RW	0

RFCORE_SFR_RFST

Address offset	0x38		
Physical Address	0x4008 8838	**Instance**	RFCORE_SFR
Description	RF CSMA-CA/strobe processor		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7:0	INSTR	Data written to this register is written to the CSP instruction memory. Reading this register returns the CSP instruction currently being executed.	RW	0xD0

23.16.4 CCTEST Registers

23.16.4.1 CCTEST Registers Mapping Summary

This section provides information on the CCTEST module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 23-11. CCTEST Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
CCTEST_IO	RW	32	0x0000 0000	0x00	0x4401 0000
CCTEST_OBSSEL0	RW	32	0x0000 0000	0x14	0x4401 0014
CCTEST_OBSSEL1	RW	32	0x0000 0000	0x18	0x4401 0018
CCTEST_OBSSEL2	RW	32	0x0000 0000	0x1C	0x4401 001C
CCTEST_OBSSEL3	RW	32	0x0000 0000	0x20	0x4401 0020
CCTEST_OBSSEL4	RW	32	0x0000 0000	0x24	0x4401 0024
CCTEST_OBSSEL5	RW	32	0x0000 0000	0x28	0x4401 0028
CCTEST_OBSSEL6	RW	32	0x0000 0000	0x2C	0x4401 002C
CCTEST_OBSSEL7	RW	32	0x0000 0000	0x30	0x4401 0030
CCTEST_TR0	RW	32	0x0000 0000	0x34	0x4401 0034
CCTEST_USBCTRL	RW	32	0x0000 0000	0x50	0x4401 0050

23.16.4.2 CCTEST Register Descriptions

CCTEST_IO

Address offset	0x00		
Physical Address	0x4401 0000	**Instance**	CC_TESTCTRL
Description	Output strength control		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	RO	0x0000 0000
0	SC	I/O strength control bit Common to all digital output pads Should be set when unregulated voltage is below approximately 2.6 V.	RW	0

CCTEST_OBSSEL0

Address offset	0x14		
Physical Address	0x4401 0014	**Instance**	CC_TESTCTRL
Description	Select output signal on observation output 0		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	EN	Observation output 0 enable control for PC0 0: Observation output disabled 1: Observation output enabled Note: If enabled, this overwrites the standard GPIO behavior of PC0.	RW	0
6:0	SEL	n - obs_sigs[n] output on output 0: 0: rfc_obs_sig0 1: rfc_obs_sig1 2: rfc_obs_sig2 Others: Reserved	RW	0x00

CCTEST_OBSSEL1

Address offset	0x18		
Physical Address	0x4401 0018	**Instance**	CC_TESTCTRL
Description	Select output signal on observation output 1		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	EN	Observation output 1 enable control for PC1 0: Observation output disabled 1: Observation output enabled Note: If enabled, this overwrites the standard GPIO behavior of PC1.	RW	0
6:0	SEL	n - obs_sigs[n] output on output 1: 0: rfc_obs_sig0 1: rfc_obs_sig1 2: rfc_obs_sig2 Others: Reserved	RW	0x00

CCTEST_OBSSEL2

Address offset	0x1C		
Physical Address	0x4401 001C	**Instance**	CC_TESTCTRL
Description	Select output signal on observation output 2		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	EN	Observation output 2 enable control for PC2 0: Observation output disabled 1: Observation output enabled Note: If enabled, this overwrites the standard GPIO behavior of PC2.	RW	0
6:0	SEL	n - obs_sigs[n] output on output 2: 0: rfc_obs_sig0 1: rfc_obs_sig1 2: rfc_obs_sig2 Others: Reserved	RW	0x00

CCTEST_OBSSEL3

Address offset	0x20		
Physical Address	0x4401 0020	**Instance**	CC_TESTCTRL
Description	Select output signal on observation output 3		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	EN	Observation output 3 enable control for PC3 0: Observation output disabled 1: Observation output enabled Note: If enabled, this overwrites the standard GPIO behavior of PC3.	RW	0
6:0	SEL	n - obs_sigs[n] output on output 3: 0: rfc_obs_sig0 1: rfc_obs_sig1 2: rfc_obs_sig2 Others: Reserved	RW	0x00

CCTEST_OBSSEL4

Address offset	0x24		
Physical Address	0x4401 0024	**Instance**	CC_TESTCTRL
Description	Select output signal on observation output 4		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	EN	Observation output 4 enable control for PC4 0: Observation output disabled 1: Observation output enabled Note: If enabled, this overwrites the standard GPIO behavior of PC4.	RW	0
6:0	SEL	n - obs_sigs[n] output on output 4: 0: rfc_obs_sig0 1: rfc_obs_sig1 2: rfc_obs_sig2 Others: Reserved	RW	0x00

CCTEST_OBSSEL5

Address offset	0x28		
Physical Address	0x4401 0028	**Instance**	CC_TESTCTRL
Description	Select output signal on observation output 5		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	EN	Observation output 5 enable control for PC5 0: Observation output disabled 1: Observation output enabled Note: If enabled, this overwrites the standard GPIO behavior of PC5.	RW	0
6:0	SEL	n - obs_sigs[n] output on output 5: 0: rfc_obs_sig0 1: rfc_obs_sig1 2: rfc_obs_sig2 Others: Reserved	RW	0x00

CCTEST_OBSSEL6

Address offset	0x2C		
Physical Address	0x4401 002C	**Instance**	CC_TESTCTRL
Description	Select output signal on observation output 6		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	EN	Observation output 6 enable control for PC6 0: Observation output disabled 1: Observation output enabled Note: If enabled, this overwrites the standard GPIO behavior of PC6.	RW	0
6:0	SEL	n - obs_sigs[n] output on output 6: 0: rfc_obs_sig0 1: rfc_obs_sig1 2: rfc_obs_sig2 Others: Reserved	RW	0x00

CCTEST_OBSSEL7

Address offset	0x30		
Physical Address	0x4401 0030	**Instance**	CC_TESTCTRL
Description	Select output signal on observation output 7		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RO	0x00 0000
7	EN	Observation output 7 enable control for PC7 0: Observation output disabled 1: Observation output enabled Note: If enabled, this overwrites the standard GPIO behavior of PC7.	RW	0
6:0	SEL	n - obs_sigs[n] output on output 7: 0: rfc_obs_sig0 1: rfc_obs_sig1 2: rfc_obs_sig2 Others: Reserved	RW	0x00

CCTEST_TR0

Address offset	0x34		
Physical Address	0x4401 0034	**Instance**	CC_TESTCTRL
Description	Test register 0		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:3	RESERVED	This bit field is reserved.	RO	0x0000 0000
2	RESERVED	This bit field is reserved.	RW	0
1	ADCTM	Set to 1 to connect the temperature sensor to the SOC_ADC. See also RFCORE_XREG_ATEST register description to enable the temperature sensor.	RW	0
0	RESERVED	This bit field is reserved.	RW	0

CCTEST_USBCTRL

Address offset	0x50		
Physical Address	0x4401 0050	**Instance**	CC_TESTCTRL
Description	USB PHY stand-by control		
Type	RW		

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

Bits	Field Name	Description	Type	Reset
31:1	RESERVED	This bit field is reserved.	RO	0x0000 0000
0	USB_STB	USB PHY stand-by override bit When this bit is cleared to 0 (default state) the USB module cannot change the stand-by mode of the PHY (USB pads) and the PHY is forced out of stand-by mode. This bit must be 1 as well as the stand-by control from the USB controller, before the mode of the PHY is stand-by.	RW	0

23.16.5 ANA_REGS Registers

23.16.5.1 ANA_REGS Registers Mapping Summary

This section provides information on the ANA_REGS module instance within this product. Each of the registers within the module instance is described separately below.

Register fields should be considered static unless otherwise noted as dynamic.

Table 23-12. ANA_REGS Register Summary

Register Name	Type	Register Width (Bits)	Register Reset	Address Offset	Physical Address
ANA_REGS_IVCTRL	RW	32	0x0000 0013	0x04	0x400D 6004

23.16.5.2 ANA_REGS Register Descriptions

ANA_REGS_IVCTRL

Address offset	0x04		
Physical Address	0x400D 6004	**Instance**	ANA_REGS
Description	Analog control register		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								RESERVED	Reserved	DAC_CURR_CTRL		LODIV_BIAS_CTRL	TXMIX_DC_CTRL	PA_BIAS_CTRL	

Bits	Field Name	Description	Type	Reset
31:8	RESERVED	This bit field is reserved.	RW	0x00 0000
7	RESERVED	This bit field is reserved.	RO	0
6	Reserved	Reserved. Always read 0.	RW	0
5:4	DAC_CURR_CTRL	Controls bias current to DAC 00: 100% IVREF, 0% IREF bias 01: 60% IVREF, 40% IREF bias 10: 40% IVREF, 60% IREF bias 11: 0% IVREF, 100% IREF bias	RW	0x1
3	LODIV_BIAS_CTRL	Controls bias current to LODIV 1: PTAT bias 0: IVREF bias	RW	0
2	TXMIX_DC_CTRL	Controls DC bias in TXMIX	RW	0

Bits	Field Name	Description	Type	Reset
1:0	PA_BIAS_CTRL	Controls bias current to PA 00: IREF bias 01: IREF and IVREF bias (CC2530 mode) 10: PTAT bias 11: Increased PTAT slope bias	RW	0x3

Voltage Regulator

The digital voltage regulator is used to power the digital core. The output of this regulator is available on the DCOUPL pin and requires capacitive decoupling to function properly (for example, see the reference design of the CC2538 device).

The voltage regulator is disabled in power modes PM2 and PM3. When the voltage regulator is disabled, most of the register and RAM contents are retained while the unregulated 2- to 3.6-V power supply is present, see Section 7.6 for details.

NOTE: Do not use the voltage regulator to provide power to external circuits.

Available Software

This chapter presents the various available software solutions relevant to the CC2538 SoC. They are all available free of charge on the TI Web site at www.ti.com/lprf when used with TI LPRF devices.

Topic **Page**

A.1 SmartRF™ Studio Software for Evaluation (www.ti.com/smartrfstudio)

Texas Instruments' SmartRF Studio can be used for radio performance and functionality evaluation and is great for exploring and gaining knowledge about the RF-IC products. This software helps the designers of radio systems to evaluate the RF-ICs easily at an early stage in the design process. It is especially useful for generation of the configuration data and for finding optimized external component values.

SmartRF Studio software runs on Microsoft™ Windows™ XP (32 bit), Windows Vista (32 & 64 bit) and Windows 7 (32 & 64 bit). SmartRF Studio can be downloaded from the Texas Instruments Web page: www.ti.com/smartrfstudio.

Features

- Link tests. Send and Receive packets between nodes.
- Antenna and radiation tests. Set the radio in continuous TX and RX states.
- Easy Mode for packet testing and for getting basic register values.
- A set of recommended/typical register settings for all devices.
- Read and write individual RF registers.
- Detailed information about the bit fields for each register.
- Save/Load device configuration data from file.
- Exports register settings to a user definable format.
- Communication with evaluation boards through USB port or the parallel port.
- Up to 32 evaluation boards are supported on a single computer.

A.2 TIMAC Software (www.ti.com/timac)

TIMAC software is an IEEE 802.15.4 medium-access-control software stack for TI's IEEE 802.15.4 transceivers and System-on-Chips.

You can use TIMAC when you:

- Need a wireless point-to-point or point-to-multipoint solution; e.g., multiple sensors reporting directly to a master
- Need a standardized wireless protocol
- Have battery-powered and/or mains-powered nodes
- Need support for acknowledgement and retransmission
- Have low data-rate requirements (around 100-kbps effective data rate)

Features

- Support for IEEE 802.15.4 standard
- Support for beacon-enabled and non-beaconing systems
- Multiple platforms
- Easy application development

The TIMAC software stack is certified to be compliant with the IEEE 802.15.4 standard. TIMAC software is distributed as object code free of charge. There are no royalties for using TIMAC software.

For more information about TIMAC software, see the Texas Instruments TIMAC Web site www.ti.com/timac.

A.3 Z-Stack™ Software (www.ti.com/z-stack)

Z-Stack™ is TI's ZigBee compliant protocol stack for a growing portfolio of IEEE 802.15.4 products and platforms. Z-Stack™ is compliant with the ZigBee® 2012 specification. Z-Stack™ supports the Smart Energy, Light Link, Home Automation, Building Automation and Health Care public application profiles.

Z-Stack software notables include:

• ZigBee Compliant Platform certification

- ZigBee and ZigBee PRO feature sets
- A range of sample applications for ZigBee Smart Energy, ZigBee Light Link and ZigBee Home Automation profiles
- Over-the-air download and serial boot loader support
- Support for RF front-ends, CC2590 and CC2591, with 10dBm and 20dBm output power, respectively, and improved receiver sensitivity.

The Z-Stack™ software has been awarded the ZigBee Alliance's golden-unit status for both the ZigBee and ZigBee PRO stack profiles and is used by ZigBee developers worldwide.

Z-Stack™ software is well suited for:

- Smart energy
- Home automation
- Connected lighting
- Commercial building automation
- Medical, assisted living, or personal health and hospital care
- Monitoring and control applications
- Wireless sensor networks
- Alarm and security
- Asset tracking
- Applications that require interoperability

For more information, visit the Texas Instruments Z-Stack™ software page: www.ti.com/z-stack.

Abbreviations

Abbreviations used in this user's guide:

AAF	Anti-aliasing filter
ACK	Acknowledge
ADC	Analog-to-digital converter
AES	Advanced Encryption Standard
AGC	Automatic gain control
ARIB	Association of Radio Industries and Businesses
BCD	Binary-coded decimal
BER	Bit error rate
BLE	Bluetooth low-energy
BOD	Brownout detector
BOM	Bill of materials
BSP	Bit-stream process
CBC	Cipher block chaining
CBC-MAC	Cipher block chaining message authentication code
CCA	Clear channel assessment or Customer configuration area. Context dependent.
CCM	Counter mode + CBC-MAC
CFB	Cipher feedback
CFR	Code of Federal Regulations
CMRR	Common-mode rejection ratio
CPU	Central processing unit
CRC	Cyclic redundancy check
CSMA-CA	Carrier sense multiple access with collision avoidance
CSP	CSMA/CA strobe processor
CTR	Counter mode (encryption)
CW	Continuous wave
DAC	Digital-to-analog converter
DC	Direct current
DMA	Direct memory access
DSM	Delta-sigma modulator
DSSS	Direct-sequence spread spectrum
ECB	Electronic code book (encryption)
EM	Evaluation module
ENOB	Effective number of bits
ETSI	European Telecommunications Standards Institute
EVM	Error vector magnitude
FCC	Federal Communications Commission
FCF	Frame control field
FCS	Frame check sequence

FFCTRL	FIFO and frame control
FIFO	First in, first out
FPB	Flash patch and breakpoint unit
FS	Full scale
GPIO	General-purpose input/output
HF	High frequency
HSSD	High-speed serial data
I/O	Input/output
I/Q	In-phase/quadrature-phase
IEEE	Institute of Electrical and Electronics Engineers
IF	Intermediate frequency
IOC	I/O controller
IRQ	Interrupt request
IR	Infrared
ISM	Industrial, scientific and medical
ITM	Instrumentation trace macrocell
ITU-T	International Telecommunication Union – Telecommunication
IV	Initialization vector
KB	1024 bytes
kbps	Kilobits per second
LFSR	Linear feedback shift register
LLE	Link-layer engine
LNA	Low-noise amplifier
LO	Local oscillator
LQI	Link quality indication
LSB	Least-significant bit/byte
MAC	Medium access control
MAC	Message authentication code
MCU	Microcontroller unit
MFR	MAC footer
MHR	MAC header
MIC	Message integrity code
MISO	Master in, slave out
MOSI	Master out, slave in
MPDU	MAC protocol data unit
MSB	Most-significant bit/byte
MSDU	MAC service data unit
MUX	Multiplexer
NA	Not applicable/available
NC	Not connected
OFB	Output feedback (encryption)
O-QPSK	Offset – quadrature phase-shift keying
PA	Power amplifier
PC	Program counter
PCB	Printed circuit board
PER	Packet error rate
PHR	PHY header
PHY	Physical layer
PLL	Phase-locked loop

PM1, PM2, PM3	Power mode 1, 2, and 3
PMC	Power management controller
PN7, PN9	7-bit or 9-bit pseudo-random sequence
POR	Power-on reset
PSDU	PHY service data unit
PWM	Pulse-width modulator
RAM	Random access memory
RBW	Resolution bandwidth
RC	Resistor-capacitor
RCOSC	RC oscillator
RF	Radio frequency
RSSI	Receive signal strength indicator
RTC	Real-time clock
RX	Receive
SCK	Serial clock
SFD	Start of frame delimiter
SFR	Special function register
SHR	Synchronization header
SINAD	Signal-to-noise and distortion ratio
SIR	Serial Infrared
SPI	Serial peripheral interface
SRAM	Static random-access memory
ST	Sleep timer
T/R	Tape and reel
T/R	Transmit/receive
TCK	Test clock
TDI	Test data in
TDO	Test data out
THD	Total harmonic distortion
TI	Texas Instruments
TMS	Test mode select
TPIU	Trace Port Interface Unit
TRST	Test reset
TTL	Transistor-transistor logic
TX	Transmit
UART	Universal asynchronous receiver/transmitter
USART	Universal synchronous/asynchronous receiver/transmitter
VCO	Voltage-controlled oscillator
VGA	Variable-gain amplifier
WDT	Watchdog timer
XOSC	Crystal oscillator

Additional Information

Texas Instruments offers a wide selection of cost-effective, low-power RF solutions for proprietary and standard-based wireless applications for use in industrial and consumer applications. Our selection includes RF transceivers, RF transmitters, RF front ends and System-on-Chips as well as various software solutions for the sub-1 and 2.4-GHz frequency bands.

In addition, Texas Instruments provides a large selection of support collateral such as development tools, technical documentation, reference designs, application expertise, customer support, third-party and university programs.

The Low-Power RF E2E Online Community provides you with technical support forums, videos and blogs, and the chance to interact with fellow engineers from all over the world.

With a broad selection of product solutions, end application possibilities, and the range of technical support, Texas Instruments offers the broadest low-power RF portfolio. **We make RF easy!**

The following subsections point to where to find more information.

Topic **Page**

C.1 Texas Instruments Low-Power RF Web Site

Texas Instruments' Low-Power RF Web site has all our latest products, application and design notes, FAQ section, news and events updates, and much more. Just go to www.ti.com/lprf.

C.2 Low-Power RF Online Community

- Forums, videos, and blogs
- RF design help
- E2E interaction - Posting one's own and reading other users' questions

Join us today at www.ti.com/lprf-forum

C.3 Texas Instruments Low-Power RF Developer Network

Texas Instruments has launched an extensive network of low-power RF development partners to help customers speed up their application development. The network consists of recommended companies, RF consultants, and independent design houses that provide a series of hardware module products and design services, including:

- RF circuit, low-power RF and ZigBee design services
- Low-power RF and ZigBee module solutions and development tools
- RF certification services and RF circuit manufacturing

Need help with modules, engineering services or development tools?

Search the Low-Power RF Developer Network to find a suitable partner! www.ti.com/lprfnetwork

C.4 Low-Power RF eNewsletter

The Low-Power RF eNewsletter keeps you up to date on new products, news releases, developers' news, and other news and events associated with low-power RF products from TI. The Low-Power RF eNewsletter articles include links to get more online information.

Sign up today on www.ti.com/lprfnewsletter.

References

References and other useful material:

1. IEEE Std. 802.15.4-2006: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Low Rate Wireless Personal Area Networks (LR-WPANs) http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf
2. CC2538 Data Sheet http://www.ti.com/lit/ds/symlink/cc2538.pdf
3. CC2538 ROM User's Guide
4. CC2538 Driver Library User's Guide
5. Universal Serial Bus Specification Rev. 2.0

Revision History

E.1 Revision History – External

The following table summarizes the document versions.

Note: Numbering may vary from previous versions.

Version	Literature Number	Date	Notes
*	SWRU319	April 2012	See (1)
A	SWRU319A	November 2012	See (2)
B	SWRU319B	April 2013	See (3)
C	SWRU319C	May 2013	See (4)

(1) *CC2538 System-on-Chip Solution User's Guide* (SWRU319) - initial release.

(2) *CC2538 System-on-Chip Solution User's Guide*, version A (SWRU319A):
- Most of resolved and closed action items have been updated.

(3) *CC2538 System-on-Chip Solution User's Guide,* version B (SWRU319B):
- Registers updated for all chapters
- Document released to web

(4) *CC2538 System-on-Chip Solution User's Guide,* version C (SWRU319C):
- Registers updated for all chapters

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265