ADuCM302x Ultra Low Power ARM Cortex-M3 MCU with Integrated Power Management Hardware Reference

Revision 1.0, March 2017

Part Number
82-100125-01

Analog Devices, Inc.
One Technology Way
Norwood, MA 02062-9106

ANALOG
DEVICES

Notices

Copyright Information

Disclaimer

Trademark and Service Mark Notice

Contents

Power Management (PMG)

Preface

Thank you for purchasing and developing systems using an ADuCM302x Micro Controller Unit (MCU) from Analog Devices, Inc.

Purpose of This Manual

The *ADuCM302x Ultra Low Power ARM® Cortex®-M3 MCU with Integrated Power Management Hardware Reference* provides architectural information about the ADuCM302x microcontrollers. This hardware reference provides the main architectural information about these microcontrollers. This includes power management, clocking, memories, peripherals, and the AFE.

For programming information, visit the ARM Information Center at: http://infocenter.arm.com/help/

The applicable documentation for programming the ARM Cortex-M3 processor include:

- *ARM Cortex-M3 Devices Generic User Guide*
- *ARM Cortex-M3 Technical Reference Manual*

For timing, electrical, and package specifications, refer to the *ADuCM3027/ADuCM3029 Ultra Low Power ARM Cortex-M3 MCU with Integrated Power Management Data Sheet*.

Intended Audience

The primary audience for this manual is a programmer who is familiar with the Analog Devices processors. The manual assumes that the audience has a working knowledge of the appropriate processor architecture and instruction set. Programmers who are unfamiliar with Analog Devices processors can use this manual, but should supplement it with other texts, such as programming reference books and data sheets, that describe their target architecture.

What's New in This Manual?

This is the first revision (1.0) of the *ADuCM302x Ultra Low Power ARM Cortex-M3 MCU with Integrated Power Management Hardware Reference*.

Technical or Customer Support

You can reach customer and technical support for processors from Analog Devices in the following ways:

- Post your questions in the analog microcontrollers support community at ***EngineerZone***:

  http://ez.analog.com/community/analog-microcontrollers/aducm302x

- Submit your questions to technical support at ***Connect with ADI Specialists***:

  http://www.analog.com/support

- E-mail your questions about software/hardware development tools to:

  processor.tools.support@analog.com

- E-mail your questions about processors to:

  iot_support@analog.com (world wide support)

- Phone questions to *1-800-ANALOGD* (USA only)

- Contact your Analog Devices sales office or authorized distributor. Locate one at:

  http://www.analog.com/adi-sales

- Send questions by mail to:

  Analog Devices, Inc.

  Three Technology Way

  P.O. Box 9106

  Norwood, MA 02062-9106 USA

Product Information

Product information can be obtained from the Analog Devices Web site and the online help system.

Analog Devices Web Site

The Analog Devices Website (http://www.analog.com) provides information about a broad range of products—analog integrated circuits, amplifiers, converters, and digital signal processors.

To access a complete technical library for each processor family, go to http://www.analog.com/processors/technical_library. The manuals selection opens a list of current manuals related to the product as well as a link to the previous revisions of the manuals. When locating your manual title, note a possible errata check mark next to the title that leads to the current correction report against the manual.

MyAnalog.com is a free feature of the Analog Devices Web site that allows customization of a Web page to display only the latest information about products you are interested in. You can choose to receive weekly e-mail notifications containing updates to the Web pages that meet your interests, including documentation errata against all manuals. MyAnalog.com provides access to books, application notes, data sheets, code examples, and more.

Visit MyAnalog.com to sign up. If you are a registered user, just log on. Your user name is your e-mail address.

EngineerZone

EngineerZone is a technical support forum from Analog Devices. It allows you direct access to ADI technical support engineers. You can search FAQs and technical information to get quick answers to your embedded processing and DSP design questions.

Use EngineerZone to connect with other MCU developers who face similar design challenges. You can also use this open forum to share knowledge and collaborate with the ADI support team and your peers. Visit http://ez.analog.com to sign up.

Notation Conventions

Text conventions used in this manual are identified and described as follows. Additional conventions, which apply only to specific chapters, may appear throughout this document.

Example	Description
File > ***Close***	Titles in reference sections indicate the location of an item within the CrossCore Embedded Studio IDE's menu system (for example, the ***Close*** command appears on the ***File*** menu).
`{this	that}`	Alternative required items in syntax descriptions appear within curly brackets and separated by vertical bars; read the example as `this` or `that`. One or the other is required.
`[this	that]`	Optional items in syntax descriptions appear within brackets and separated by vertical bars; read the example as an optional `this` or `that`.
`[this, …]`	Optional item lists in syntax descriptions appear within brackets delimited by commas and terminated with an ellipse; read the example as an optional comma-separated list of `this`.
`.SECTION`	Commands, directives, keywords, and feature names are in text with `Letter Gothic` font.
`filename`	Non-keyword placeholders appear in text with italic style format.
NOTE:	*NOTE:* For correct operation, ... A note provides supplementary information on a related topic. In the online version of this book, the word *NOTE:* appears instead of this symbol.
CAUTION:	*CAUTION:* Incorrect device operation may result if ... *CAUTION:* Device damage may result if ... A caution identifies conditions or inappropriate usage of the product that could lead to undesirable results or product damage. In the online version of this book, the word *CAUTION* appears instead of this symbol.
ATTENTION:	*ATTENTION:* Injury to device users may result if ... A warning identifies conditions or inappropriate usage of the product that could lead to conditions that are potentially hazardous for devices users. In the online version of this book, the word *ATTENTION* appears instead of this symbol.

Register Diagram Conventions

Register diagrams use the following conventions:

- The descriptive name of the register appears at the top, followed by the short form of the name in parentheses.
- If the register is read-only (RO), write-1-to-set (W1S), or write-1-to-clear (W1C), this information appears under the name. Read/write is the default and is not noted. Additional descriptive text may follow.
- If any bits in the register do not follow the overall read/write convention, this is noted in the bit description after the bit name.
- If a bit has a short name, the short name appears first in the bit description, followed by the long name in parentheses.
- The reset value appears in binary in the individual bits and in hexadecimal to the right of the register.
- Bits marked X have an unknown reset value. Consequently, the reset value of registers that contain such bits is undefined or dependent on pin values at reset.
- Shaded bits are reserved.

NOTE: To ensure upward compatibility with future implementations, write back the value that is read for reserved bits in a register, unless specified.

Register description tables use the following conventions:

- Each bit's or bit field's access type appears beneath the bit number in the table in the form (read-access/write-access). The access types include:
- R = read, RC = read clear, RS = read set, R0 = read zero, R1 = read one, Rx = read undefined
- W = write, NW = no write, W1C = write one to clear, W1S = write one to set, W0C = write zero to clear, W0S = write zero to set, WS = write to set, WC = write to clear, W1A = write one action.
- Several bits and bit field descriptions include enumerations, identifying bit values and related functionality. Unless indicated (with a prefix), these enumerations are decimal values.

1 Introduction

The ADuCM302x MCU is an ultra low power microcontroller system with integrated power management for processing, control, and connectivity. It minimizes power requirements to optimize system solution, enables reliable system operation, and addresses security requirements. The MCU subsystem is based on ARM Cortex-M3 processor, a collection of digital peripherals, embedded SRAM and flash memory, and an analog subsystem which provides clocking, reset and power management capability in addition to an analog-to-digital converter (ADC) subsystem.

The ADuCM302x MCU provides a collection of power modes and features such as dynamic and software controlled clock gating and power gating to support extremely low dynamic and hibernate power.

ADuCM302x MCU Features

The ADuCM302x MCU supports the following features:

- A 26 MHz ARM Cortex-M3 processor
- Up to 256 KB of embedded flash memory with ECC
- 32 KB system SRAM with parity
- 32 KB user configurable instruction/data SRAM with parity

  4 KB of SRAM may be used as cache memory to reduce active power consumption by reducing access to flash memory
- Power Management Unit (PMU)
- Power-on-Reset (POR) and Power Supply Monitor (PSM)
- A buck converter for improved efficiency in active and Flexi™ states
- A multilayer AMBA bus matrix
- A multi-channel central direct memory access (DMA) controller
- Programmable GPIOs
- I^{2}C and UART peripheral interfaces
- Three SPI interfaces capable of interfacing gluelessly with a range of sensors and converters

- A serial port (SPORT) capable of interfacing with a wide range of radios and converters. Two single direction half SPORT or one bidirectional full SPORT
- A beeper driver to produce single and multi-tone playback options
- A real-time clock (RTC) capable of maintaining accurate wall clock time
- A flexible real-time clock (FLEX_RTC) that supports a wide range of wake up times
- Three general purpose timers and one watchdog timer
- Hardware cryptographic accelerator supporting AES-128, AES-256 along with various modes (ECB, CBC, CTR, CBC-MAC, CCM, CCM*) and SHA-256
- True Random Number Generator (TRNG)
- Hardware CRC with programmable generator polynomial
- Multi parity bit protected SRAM
- Up to 1.8 MSPS, 12-bit SAR ADC

ADuCM302x Functional Description

This section provides information on the function of the ADuCM302x MCU.

ADuCM302x Block Diagram

The ARM Cortex-M3 core is a 32-bit reduced instruction set computer (RISC) offering up to 33 MIPS of peak performance at 26 MHz. A central DMA controller is used to efficiently move data between peripherals and memory.

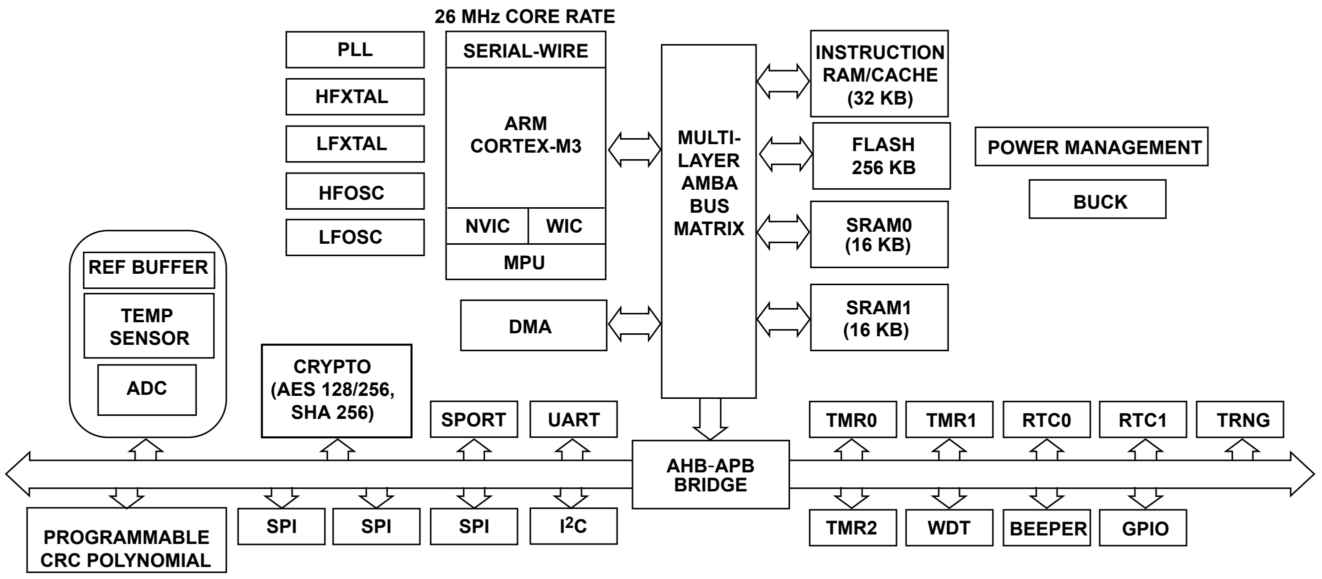


Figure 1-1: ADuCM302x Block Diagram

Memory Architecture

The ADuCM3027/ADuCM3029 MCU incorporates 128 KB/256 KB of embedded flash memory for program code and nonvolatile data storage, 32 KB of data SRAM, and 32 KB of SRAM (configurable between instruction and data space). For added robustness and reliability, ECC is enabled for the flash memory and multi bit parity can be used to detect random soft errors in SRAM.

SRAM Region

This memory space contains the application instructions and literal (constant) data which must be executed real time. It supports read/write access by the M3 core and read/write DMA access by system peripherals. SRAM is divided into Data SRAM of 32 KB and Instruction SRAM of 32 KB. If instruction SRAM is not enabled, then the associated 32 KB can be mapped as data SRAM, resulting in a 64 KB Data SRAM.

Internal SRAM Data Region

This space can contain read/write data. Internal SRAM can be partitioned between CODE and DATA (SRAM region in M3 space) in 32 KB blocks. Access to this region occurs at core clock speed, with no wait states. It supports read/write access by the M3 core and read/write DMA access by system devices. It supports exclusive memory access through the global exclusive access monitor within the Analog Devices Cortex-M3 platform.

The figure shows the address map of the SRAM for various user selectable configurations. For information about the configuration, refer to Static Random Access Memory (SRAM).

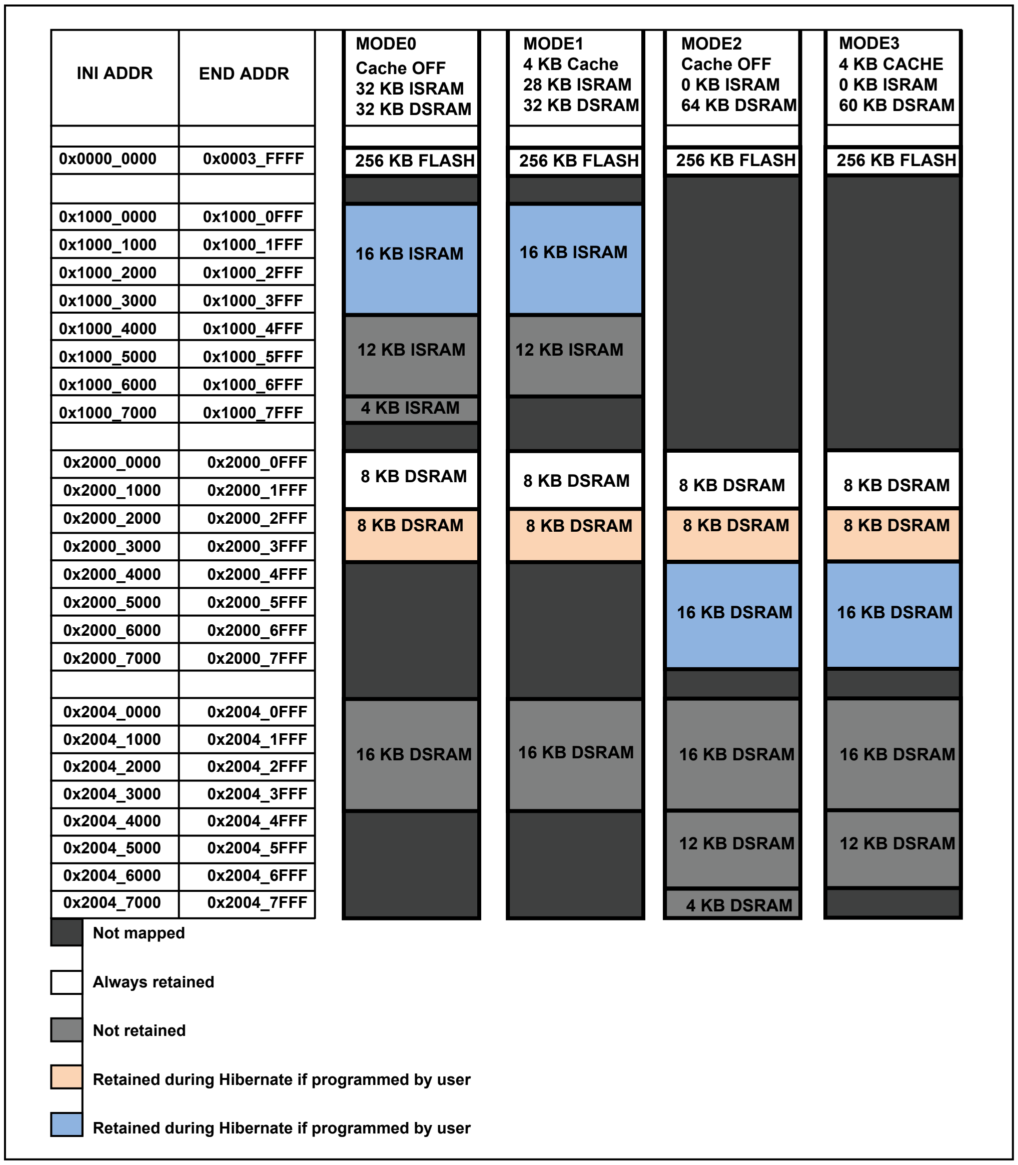


Figure 1-2: Selectable Configuration

System Region

The ARM Cortex-M3 processor accesses the system region on its SYS interface and is handled within the Cortex-M3 platform.

CoreSight™ ROM: The ROM table entries point to the debug components of the processor.

ARM APB Peripherals: This space is defined by ARM and occupies the bottom 256 KB (ADuCM3029) /128 KB (ADuCM3027) of the SYS region. The space supports read/write access by the M3 core to the ARM cores internal peripherals (SCS, NVIC, and WIC) and the CoreSight ROM. It is not accessible by the system DMA.

Platform Control Registers: This space has registers within the Cortex-M3 platform component that control the ARM core, its memory, and the code cache. It is accessible by the M3 core through its SYS port (but is not accessible by the system DMA).

Flash Controller

The ADuCM302x includes up to 256 KB of embedded flash memory, accessed using the flash controller.

The flash controller is coupled with a cache controller module, which provides two Advanced High Performance Bus (AHB) ports:

- DCode for reading data
- ICode for reading instructions

A prefetch mechanism is implemented in the flash controller to optimize ICode read performance. Flash writes are supported by a keyhole mechanism from Advanced Peripheral Bus (APB) writes to memory mapped registers. The flash controller provides support for DMA based key hole writes.

The flash controller supports the following:

- A fixed user key required for running protected commands including mass erase and page erase.
- An optional and user definable user failure analysis key (FAA Key). Analog Devices personnel needs this key while performing failure analysis.
- An optional and user definable write protection for user accessible memory.
- An optional 8-bit error correction code (ECC). This code can be disabled by user code (enabled by default).

  It is recommended not to disable it, as there is no change in access time or power consumption. It ensures a robust environment.

Cache Controller

The ADuCM302x family has an optional 4 KB instruction cache. When the cache controller is enabled, 4 KB of instruction SRAM is reserved for cache data. In hibernate mode, the cache memory is not retained.

ARM Cortex-M3 Memory Subsystem

The memory map of the ADuCM302x family is based on the ARM Cortex-M3 model. By retaining the standardized memory mapping, it becomes easier to port applications across M3 platforms. The ADuCM302x application development is typically based on memory blocks across Code/SRAM regions. Sufficient internal memory is available from internal SRAM and internal flash.

Booting

The ADuCM302x MCU supports the following boot modes:

- Booting from internal flash

- Upgrading software through UART download

If the SYS_BMODE0 pin (GPIO17) is pulled low during power-up or a hard reset, the MCU enters into serial download mode. In this mode, an on-chip loader routine is initiated in the kernel, which configures the UART port and communicates with the host to manage the firmware upgrade via a specific serial download protocol.

Table 1-1: Boot Modes

Boot Mode	Description
0	UART download mode.
1	Flash boot. Boot from integrated flash memory.

Security Features

The ADuCM302x MCU provides a combination of hardware and software protection mechanisms that lock out access to the part in secure mode, but grant access in open mode. These mechanisms include password-protected slave boot modes (UART), as well as password-protected SWD debug interfaces. During boot, the system is clocked from an internal on-chip oscillator. Reset computes a hardware checksum of the information area and then permit the CPU to execute the Analog Devices boot loader in the Flash information area if the checksum passes. The Analog Devices boot loader inspects the GPIO boot pin (GPIO17) which determines if user code or a UART download is executed.

There is a mechanism to protect the device contents (Flash, SRAM, CPU registers, and peripheral registers) from being read through an external interface by an unauthorized user. This is referred to as read protection.

It is possible to protect the device from being reprogrammed in-circuit with unauthorized code. This is referred to as in-circuit write protection. The device can be configured with no protection, read protection, or read and in-circuit write protection. It is not necessary to provide in-circuit write protection without read protection.

Safety Features

The ADuCM302x MCU provides several features that help achieve certain levels of system safety and reliability. While the level of safety is mainly dominated by system considerations, the following features are provided to enhance robustness.

Multi Parity Bit Protected L1 Memories

In the MCU's SRAM and cache L1 memory space, each word is protected by multiple parity bits to detect the single event upsets that occur in all RAMs.

Programmable GPIOs

The ADuCM302x MCU has 44/36 GPIO pins in the LFCSP/WLCSP packages, most of which have multiple, configurable functions defined by user code. They can be configured as an input/output and have programmable pull up resistors (pull down for GPIO06). All I/O pins are functional over the full supply range.

In deep sleep modes, GPIO pins retain state. On reset, they tristate.

Timers

The ADuCM302x MCU contains general purpose timers and a watchdog timer.

General Purpose Timers

The ADuCM302x has three identical general purpose timers, each with a 16-bit up/down counter. The up/down counter can be clocked from one of four user selectable clock sources. Any selected clock source can be scaled down using a prescaler of 1, 16, 64, or 256.

Watchdog Timer (WDT)

The watchdog timer is a 16-bit down timer with a programmable prescaler. The prescaler source is selectable and can be scaled by a factor of 1, 16, or 256. The watchdog timer is clocked by the 32 kHz on-chip oscillator (LFOSC).The WDT is used to recover from an illegal software state. The WDT requires periodic servicing to prevent it from forcing a reset or interrupt of the MCU.

Power Management

The ADuCM302x MCU includes power management and clocking features.

Power Modes

The PMU provides control of the ADuCM302x power modes, and allows the ARM Cortex-M3 to control the clocks and power gating to reduce the dynamic power and hibernate power.

The power modes are as follows:

- **Active Mode:** All peripherals can be enabled. Active power is managed by optimized clock management.
- **Flexi Mode:** The core is clock gated, but the remainder of the system is active. No instructions can be executed in this mode, but DMA transfers can continue between peripherals and memory as well as memory to memory.
- **Hibernate Mode:** This mode provides port pin retention and configurable SRAM, a limited number of wake-up interrupts, and an active RTC (optional). In this mode, the registers of the Cortex core and control registers of all peripherals are retained to ensure that the user need not reprogram these registers after waking up. However, any volatile registers such as status registers, internal state machines, FIFOs are not retained. If a peripheral is active prior to the device going into hibernate, on wake up from hibernate, the peripheral is disabled and does not continue from where it left prior to going to hibernate. Ensure that all peripheral and DMA activity has completed before going to hibernate.
- **Shutdown Mode:** This mode is the deepest sleep mode, in which all the digital and analog circuits are powered down with an option to wake from four possible wake-up sources. This mode provides port pin retention. The RTC can be optionally enabled in this mode and the device can be periodically woken up by the RTC interrupt.

Clocking

Two on-chip oscillators and driver circuitry for two external crystals are available on the ADuCM302x MCU:

- LFOSC: 32 kHz internal oscillator
- HFOSC: 26 MHz internal oscillator
- LFXTAL: 32 kHz external crystal oscillator
- HFXTAL: 26 MHz / 16 MHz external crystal oscillator

Real-Time Clock (RTC)

The ADuCM302x MCU has two real-time clock blocks, RTC0 and RTC1 (also called FLEX_RTC).

The clock blocks share a low-power crystal oscillation circuit that operates in conjunction with a 32,768 Hz external crystal or LFOSC.

The RTC has an alarm that interrupts the core when the programmed alarm value matches the RTC count. The software enables and configures the RTC and correlates the count to the time of day.

The RTC also has a digital trim capability to allow a positive or negative adjustment to the RTC count at fixed intervals.

The FLEX_RTC supports SensorStrobe™ mechanism. Using this mechanism, the ADuCM302x MCU can be used as a programmable clock generator in all power modes except shutdown mode. In this way, the external sensors can have their timing domains mastered by the ADuCM302x MCU, as SensorStrobe can output a programmable divider from the FLEX_RTC, which can operate up to a resolution of 30.7 µs. The sensors and microcontroller are in sync, which removes the need for additional resampling of data to time align it.

In the absence of this mechanism:

- The external sensor uses an RC oscillator (~ ±30% typical variation). The MCU has to sample the data and resample it on the MCU's time domain before using it.

  Or

- The MCU remains in a higher power state and drives each data conversion on the sensor side.

This mechanism allows the ADuCM302x MCU to be in hibernate mode for a long duration and also avoids unnecessary data processing which extends the battery life of the end product.

The following table shows the key differences between RTC0 and RTC1.

Table 1-2: RTC Features

Features	RTC0	RTC1 (FLEX_RTC)
Resolution of time base (prescaling)	Counts time at 1 Hz in units of seconds. Operationally, always prescales to 1 Hz (for example, divide by 32,768) and always counts real time in units of seconds.	Can prescale the clock by any power of two from 0 to 15. It can count time in units of any of these 16 possible prescale settings. For example, the clock can be prescaled by 1, 2, 4, 8, …, 16384, or 32768.
Source clock	LFXTAL	Depending on the low frequency multiplexer (LFMUX) configuration, the RTC is clocked by the LFXTAL or the LFOSC.
Wake-up time	Time is specified in units of seconds.	Supports alarm times down to a resolution of 30.7 μs, i.e., where the time is specified down to a specific 32 kHz clock cycle.
Number of alarms	One alarm only. Uses an absolute, non-repeating alarm time, specified in units of one second.	Two alarms. One absolute alarm time and one periodic alarm, repeating every 60 prescaled time units.
SensorStrobe mechanism	Not available.	It is an alarm mechanism in the RTC which causes an output pulse to be sent via `RTC1_SS1` pin to an external device to instruct the device to take a measurement or perform some action at a specific time. SensorStrobe events are scheduled at a specific target time relative to the real-time count of the RTC. This feature can be enabled in active, Flexi, and hibernate modes.
Input capture	Not available.	Input capture takes a snapshot of the RTC real-time count when an external device signals an event via a transition on one of the GPIO inputs to the ADuCM302x MCU. Typically, an input capture event is triggered by an autonomous measurement or action on such a device, which then signals to the ADuCM302x MCU that the RTC must take a snapshot of time corresponding to the event. The taking of this snapshot can wake up the ADuCM302x MCU and cause an interrupt to its CPU. The CPU can subsequently obtain information from the RTC on the exact 32 kHz cycle on which the input capture event occurred.

System Debug

The ADuCM302x MCU supports serial wire debug.

Beeper Driver

The ADuCM302x MCU has an integrated audio driver for a beeper. The beeper driver module in the ADuCM302x generates a differential square wave of programmable frequency. It drives an external piezoelectric sound component whose two terminals connect to the differential square wave output. The beeper driver consists of a module that can deliver frequencies from 8 kHz to ~0.25 kHz. It operates on a fixed independent 32 kHz (32,768 Hz) clock source that is unaffected by changes in system clocks.

It allows for programmable tone durations from 4 ms to 1.02 s in 4 ms increments. Single-tone (pulse) and multi-tone (sequence) modes provide versatile playback options. In sequence mode, the beeper can be programmed to play any number of tone pairs from 1 to 254 (2 to 508 tones) or be programmed to play forever (until stopped by the user). Interrupts are available to indicate start or end of any beep, end of a sequence, or that the sequence is nearing completion.

Cryptographic Accelerator (Crypto)

Crypto is a 32-bit APB DMA-capable peripheral. There are two buffers provided for data I/O operations. These buffers are 32-bit wide and read in or read out 128 bits in 4 data accesses. Big endian and little endian data formats are supported.

The following modes are supported:

- ECB Mode - AES mode
- CTR Mode - Counter mode
- CBC Mode - Cipher Block Chaining mode
- MAC Mode - Message Authentication Code mode
- CCM/CCM* Mode - Cipher Block Chaining-Message Authentication Code Mode
- SHA-256 Mode

CRC Accelerator

The CRC accelerator can be used to compute the CRC for a block of memory locations. The exact memory location can be in the SRAM, flash, or any combination of memory mapped registers. The CRC accelerator generates a checksum that can be used to compare it with an expected signature.

The following are the features of the CRC:

- Generate a CRC signature for a block of data
- Supports programmable polynomial length of up to 32 bits
- Operates on 32 bits of data at a time
- Supports MSB-first and LSB-first implementations of CRC
- Various data mirroring capabilities
- Initial seed to be programmed by user
- DMA controller (memory to memory transfer) can be used for data transfer to offload the MCU

True Random Number Generator (TRNG)

The TRNG is used during operations where nondeterministic values are required. This may include generating challenges for secure communication or keys used for an encrypted communication channel. The generator can be run multiple times to generate a sufficient number of bits for the strength of the intended operation. The TRNG can be used to seed a deterministic random bit generator.

Serial Ports (SPORTs)

The synchronous serial ports provide an inexpensive interface to a wide variety of digital and mixed-signal peripheral devices such as Analog Devices audio codecs, ADCs, and DACs. The serial port consist of two half SPORTs, each identical and made up of one bidirectional data line, a clock, frame sync and `SPT_CNV`. The data line can be programmed to either transmit or receive and each data line has a dedicated DMA channel. The serial port data can be automatically transferred to and from on-chip memory/external memory through dedicated DMA channels. It is possible that one half SPORT provides a transmit signal while the other half SPORT provides the receive signal. The frame sync and clock between two half SPORTS can also be shared. Some ADCs/DACs require two control signals for their conversion process. To interface with such devices, an extra signal called `SPT_CNV` signal is provided.

SPORT can operate in two modes:

- Standard serial mode
- Timer-enable mode

Serial Peripheral Interface (SPI) Ports

The ADuCM302x MCU provides three SPIs, SPI0, SPI1 and SPI2 (also known as SPIH). They are identical, but SPI2 is connected to a high performance Advanced Peripheral Bus (APB) which is always clocked at the higher system clock rate (HCLK) and contains fewer modules requiring arbitration. SPI0 and SPI1 are connected to the main APB, which can be selectively clocked at a lower rate (PCLK) and whose latency is more uncertain due to a greater number of modules requiring arbitration. SPI is an industry standard, full-duplex, synchronous serial interface that allows eight bits of data to be synchronously transmitted and simultaneously received. Each SPI incorporates two DMA channels that interface with the DMA controller. One DMA channel is used for transmit and the other is used for receive. The SPI on the MCU eases interfacing to external serial flash devices.

The SPI features available include the following:

- Serial clock phase mode and serial clock polarity mode
- Loopback mode
- Continuous transfer mode
- Wired-OR output mode
- Read-command mode for half duplex operation
- Flow control support
- Multiple CS line support
- CS software override support

- Support for 3-pin SPI
- Master or slave mode
- LSB first transfer option
- Interrupt mode: Interrupt after 1, 2, 3, 4, 5, 6, 7, or 8 bytes

UART Ports

The ADuCM302x MCU provides a full-duplex UART port, which is fully compatible with PC-standard UARTs. The UART port provides a simplified UART interface to other peripherals or hosts, supporting full-duplex, DMA-supported, asynchronous transfers of serial data. The UART port includes support for five to eight data bits, and none, even, or odd parity. A frame is terminated by one, one and a half, or two stop bits.

Inter-Integrated Circuit (I^2C)

The ADuCM302x MCU provides an I^2C interface. The I^2C bus peripheral has two pins for data transfer. SCL is a serial clock pin, and SDA is a serial data pin. The pins are configured in a Wired-AND format that allows arbitration in a multi-master system. A master device can be configured to generate the serial clock.

The frequency is programmed by the user in the serial clock divisor register. The master channel can be set to operate in fast mode (400 kHz) or standard mode (100 kHz).

Analog-to-Digital Converter (ADC) Subsystem

The ADuCM302x MCU integrates a 12-bit SAR ADC with up to eight input channels. Conversions can be performed in single or auto cycle mode. In single mode, the ADC can be configured to convert on a particular channel by selecting one of the channels. Auto cycle mode is provided to reduce processor overhead of sampling and reading individual channel registers, by automatically selecting a sequence of channels for conversion.

Reference Designs

The Circuit from the Lab® provides information on the following:

- Graphical circuit block diagram presentation of signal chains for a variety of circuit types and applications
- Links for components in each chain to selection guides and application information
- Reference designs applying best practice design techniques

Development Tools

In addition to this Hardware Reference document, the development support for the ADuCM302x MCU includes evaluation hardware and development software tools.

Hardware

The ADuCM3029 EZ-KIT ® is available to prototype a user sensor configuration with the ADuCM302x MCU.

Software

The ADuCM3029 EZ-KIT includes a complete development and debug environment for the ADuCM302x MCU. The Board Support Package (BSP) for the ADuCM302x MCU is provided for the IAR Embedded Workbench for ARM, Keil ™, and CrossCore Embedded Studio ® (CCES) environments.

The BSP also includes operating system (OS) aware drivers and example code for all the peripherals on the device.

ADuCM302x Peripheral Memory Map

The following table shows the base address of different ADuCM302x peripherals.

Table 1-3: Instance Summary

Name	Module	Address
TMR0	General Purpose Timer	0x40000000
TMR1	General Purpose Timer	0x40000400
TMR2	General Purpose Timer	0x40000800
RTC0	Real-Time Clock	0x40001000
RTC1	Real-Time Clock	0x40001400
SYS	System	0x40002000
WDT0	Watchdog Timer	0x40002C00
I2C0	I^{2}C Interface	0x40003000
SPI0	SPI	0x40004000
SPI1	SPI	0x40004400
UART0	UART	0x40005000
BEEP0	Beeper	0x40005C00
ADC0	ADC	0x40007000
DMA0	DMA	0x40010000
FLCC0	Flash/Cache Controller	0x40018000
GPIO0	GPIO	0x40020000
GPIO1	GPIO	0x40020040
GPIO2	GPIO	0x40020080
SPI2	SPI	0x40024000
SPORT0	SPORT	0x40038000
CRC0	CRC Accelerator	0x40040000
RNG0	Random Number Generator	0x40040400
CRYPT0	Cryptographic Module	0x40044000

Table 1-3: Instance Summary (Continued)

Name	Module	Address
PMG0	PMG	0x4004C000
XINT0	External Interrupt Module	0x4004C080
CLKG0	Clocking Subsystem	0x4004C100

2 Debug (DBG)

The ADuCM302x MCU supports the 2-wire serial wire debug (SWD) interface.

DBG Features

The ADuCM302x MCU contains several system debug components that facilitate low-cost debug, trace and profiling, breakpoints, watchpoints, and code patching.

This product was intended to be a lightweight implementation of the ARM Cortex-M3 processor. Minimal debug features are included in the MCU. All debug units are present but the debug capability is reduced.

The supported system debug components on this part are:

- Flash Patch and Breakpoint (FPB) unit to implement two hardware breakpoints. Unlimited software breakpoints can be added using Segger JLink. Flash patching is not supported.
- Data Watchpoint and Trace (DWT) unit to implement one watchpoint, trigger resources, and system profiling. The DWT does not support data matching for watchpoint generation because it only has one comparator.

NOTE: JTAG debug interface is not supported. In addition, embedded trace features of the MCU are supported.

DBG Functional Description

This section provides information on the function of the SWD interface used by the ADuCM302x MCU.

Serial Wire Interface

The serial wire interface consists of two pins:

- SWCLK: It is driven by the debug probe.
- SWDIO: It is a bidirectional signal which may be driven by the debug probe or target depending on the protocol phase. A non driving idle turnaround cycle is inserted on the bus whenever data direction changes to avoid contention.

SWD Pull-up

There must be a pull-up resistor on the SWDIO pin. The default state of this pin is pull-up.

The SWCLK pin must be pulled to a defined state (high or low). It is recommended to pull this pin low, which is the default state of this pin.

SWD Timing

The target samples and drives SWDIO data on posedge SWCLK.

For optimum timing, the debug probe should drive SWDIO on negedge SWCLK and sample SWDIO on posedge SWCLK.

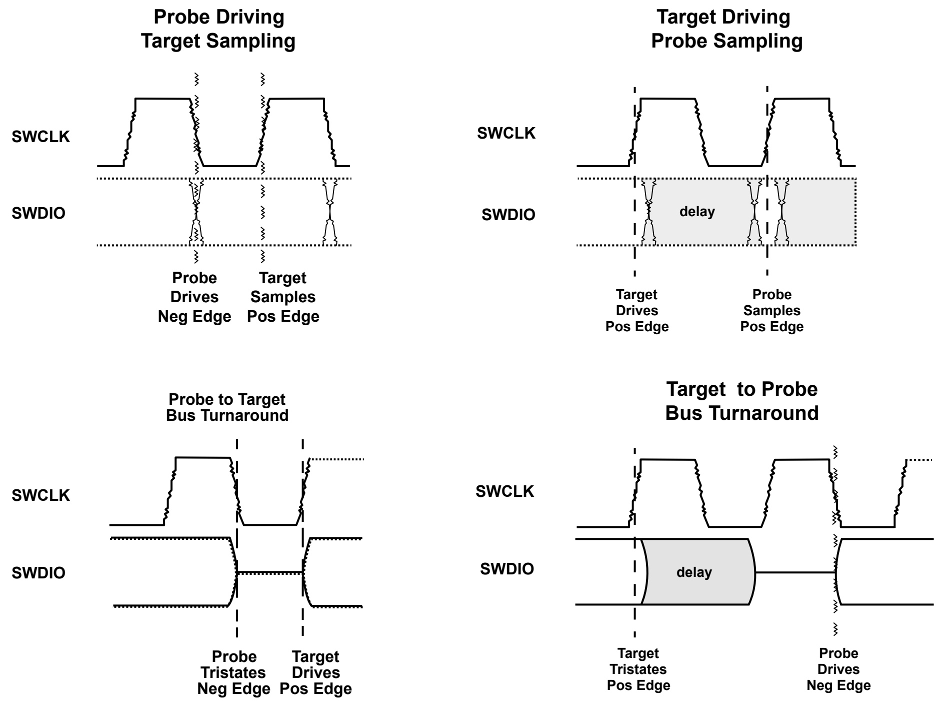


Figure 2-1: SWD Timing

DBG Operating Modes

Serial Wire Protocol

The Serial Wire Protocol consists of three phases:

- Packet Request
- Acknowledge Response
- Data Transfer

The debug probe always sends the packet request.

The target always sends the acknowledge response.

The data transfer direction depends on the packet request and acknowledge response.

Packet Request

The packet request sent by the debug probe is 8 bits and is always followed by a turnaround bit.

Table 2-1: Packet Request

Bit	Description
Start	Always High: Logic 1
APnDP	Access Port: 1 or Data Port: 0
RnW	Read: 1 or Write: 0
A[2:3]	Address sent LSB first
Parity	Parity = APnDP + RnW + A[2] + A[3]
Stop	Always Low: Logic 0
Park	Always High: Logic 1
Turn	High Impedance / Non-Driving

Acknowledge Response

The acknowledge response sent by the target is 3 bits. If the target sends data for a read (RnW=1), it is followed directly by the data transfer, there is no turnaround bit. If the debug probe sends the data transfer for a write (RnW=0), the acknowledge is followed by a turnaround bit.

Table 2-2: Acknowledge Response

Bit	Description
ACK[0:2]	Acknowledge Sent LSB first 100: Ok 010: Wait 001: Fault

Data Transfer

The data transfer phase consists of 32 bits of data sent LSB-first [0:31], followed by a single parity bit. If the number of bits set in the data field is odd, the parity bit is set to 1.

For reads where data is sent by the target, the data transfer is followed by a turnaround bit. For writes where data is sent by the debug probe, there is no turnaround bit.

The data transfer phase is omitted for Wait/Fault responses unless the CTRL/STAT ORUNDETECT bit is set. If the ORUNDETECT bit is set, the data transfer phase is always sent and STICKYORUN is set if a Wait/Fault response occurs.

Protocol Format

The figure shows the protocol format.

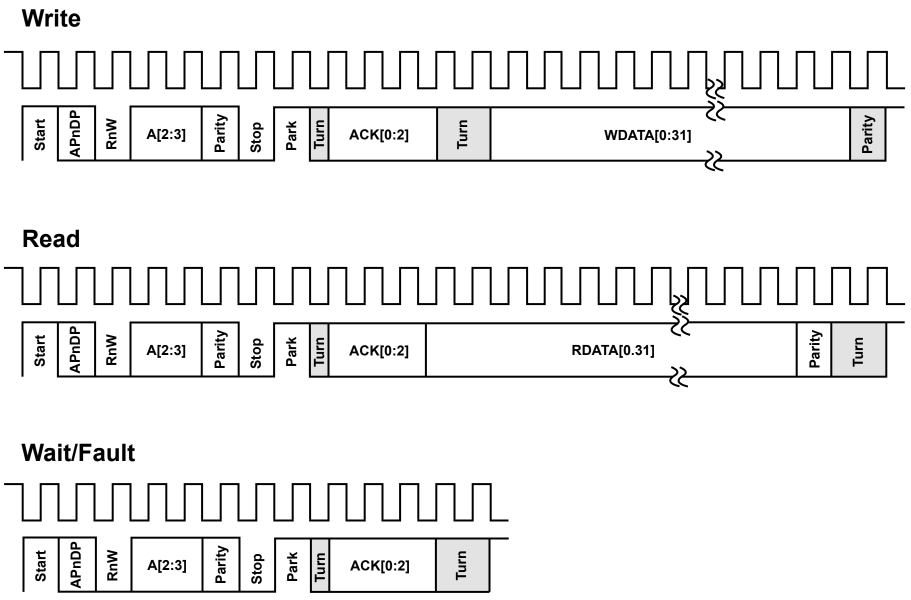


Figure 2-2: Protocol Format

Debug Access Port (DAP)

The DAP is split into two ports:

- Debug Port (DP)
- Access Port (AP)

There is a single Debug Port. There may be multiple Access Ports.

The Debug Port has a 2-bit (4-word) address space which is separate from the system memory address space. These registers can only be accessed through the serial wire debug interface, they cannot be accessed by the CPU.

Each Access-Port has a separate 6-bit address space allowing up to 64 word registers per access port. This address space is separate from the debug port and system CPU address spaces. It can only be accessed by the serial wire interface through the debug port.

On this device, there is only one Access Port available, the Memory Access Port located at `APSEL=00`.

Table 2-3: Access Ports

APSEL [7:0]	Access Port
00	Memory Access Port

Debug Port

The Debug Port has four 32-bit read write register locations. These are used to identify and configure the interface and to select and communicate with a particular Access Port.

Table 2-4: Debug Port Registers

Address [3:2]	Read	Write	DPBANKSEL
00	DPIDR	ABORT	X
01	CTRL/STAT		0
	DLCR		1
10	RESEND	SELECT	X
11	RDBUFF	Reserved	X

ADuCM302x SYS Register Descriptions

System Identification and Debug Enable (SYS) contains the following registers.

Table 2-5: ADuCM302x SYS Register List

Name	Description
SYS_ADIID	ADI Identification
SYS_CHIPID	Chip Identifier
SYS_SWDEN	Serial Wire Debug Enable

ADI Identification

ADI Cortex device identification.

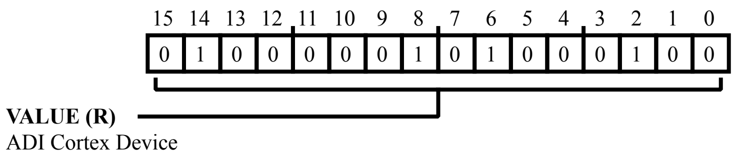


Figure 2-3: SYS_ADIID Register Diagram

Table 2-6: SYS_ADIID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	ADI Cortex Device. Reads a fixed value of 0x4144 to indicate to debuggers that they are connected to an Analog Devices implemented Cortex based part

Chip Identifier

Chip identification.

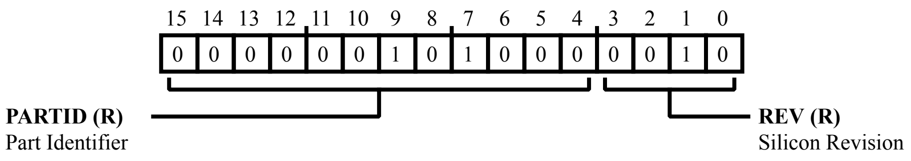


Figure 2-4: SYS_CHIPID Register Diagram

Table 2-7: SYS_CHIPID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:4 (R/NW)	PARTID	Part Identifier.
3:0 (R/NW)	REV	Silicon Revision.

Serial Wire Debug Enable

The SYS_SWDEN register is used to enable the Serial Wire Debug (SWD) interface. This register is reset upon an internal power on reset or an external pin reset. This register is not affected by a software reset.

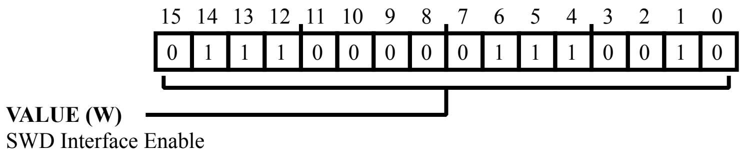


Figure 2-5: SYS_SWDEN Register Diagram

Table 2-8: SYS_SWDEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (RX/W)	VALUE	SWD Interface Enable. Writing to this register with "en" (0x6E65) or "EN" (0x4E45) enables the SWD interface. Writing to this register with "rp" (0x7072) or "RP" (0x5052) disables the SWD interface. Writes of any other value are ignored. The SWDEN register cannot be modified once written with EN, en, RP or rp. The register is reset by a POR or PIN reset, but not by a software or WDT reset.

3 Events (Interrupts and Exceptions)

The ADuCM302x MCU supports a number of system exceptions and peripheral interrupts.

Events Features

The ADuCM302x Events have the following features:

- 16 system exceptions with highest priority.
- 64 system interrupts with programmable priority.
- Four external interrupts which can be configured separately to wake up the core from low power modes.

Events Interrupts and Exceptions

Cortex Exceptions

Exceptions 1 to 15 are system exceptions.

Table 3-1: System Exceptions

Exception Number	IRQ Number	Exception Type	Priority	Description
1	–	Reset	-3 (highest)	Any reset
2	-14	NMI	-2	Non-maskable interrupt connected to a combination of (logical OR) of VREG under or VBAT (under 1.8 V)
3	-13	Hard fault	-1	All fault conditions, if the corresponding fault handler is not enabled
4	-12	MemManagement fault	Programmable	Memory management fault; access to illegal locations
5	-11	Bus fault	Programmable	Prefetch fault, memory access fault, and other address/ memory related faults
6	-10	Usage fault	Programmable	Exceptions such as undefined instruction executed or illegal state transition attempt
7 to 10	–	Reserved	Not applicable	

Table 3-1: System Exceptions (Continued)

Exception Number	IRQ Number	Exception Type	Priority	Description
11	-5	SVCALL	Programmable	System service call with SVC instruction
12	-4	Debug monitor	Programmable	Debug monitor (breakpoint, watchpoint, or external debug requests)
13	-	Reserved	Not applicable	
14	-2	PENDSV	Programmable	Pendable request for system service
15	-1	SYSTICK	Programmable	System tick timer

Nested Vectored Interrupt Controller

Interrupts are controlled by the Nested Vectored Interrupt Controller (NVIC), and eight levels of priority are available. Only a limited number of interrupts can wake up the device from hibernate mode. When the device is woken up from Flexi, hibernate, or shutdown modes, it always returns to active mode.

Table 3-2: Interrupt Sources

Exception Number	IRQ Number	Vector	Wake-up From		
			Flexi	Hibernate	Shutdown
16	0	Real-Time Clock 1/Wake-up Timer/Hibernate RTC	Yes	Yes	No
17	1	External Interrupt 0	Yes	Yes	Yes
18	2	External Interrupt 1	Yes	Yes	Yes
19	3	External Interrupt 2	Yes	Yes	Yes
20	4	External Interrupt 3//UART0_RX_Wakeup	Yes	Yes	No
21	5	Watchdog Timer	Yes	No	No
22	6	VREG Over Voltage	Yes	No	No
23	7	Battery Voltage Range	Yes	Yes	No
24	8	Real-Time Clock	Yes	Yes	Yes
25	9	GPIO IntA	Yes	No	No
26	10	GPIO IntB	Yes	No	No
27	11	General Purpose Timer 0	Yes	No	No
28	12	General Purpose Timer 1	Yes	No	No
29	13	Flash Controller	Yes	No	No
30	14	UART	Yes	No	No
31	15	SPI0	Yes*	No	No

Table 3-2: Interrupt Sources (Continued)

Exception Number	IRQ Number	Vector	Wake-up From		
			Flexi	Hibernate	Shutdown
32	16	SPIH	Yes*	No	No
33	17	I2C0 Slave	Yes*	No	No
34	18	I2C0 Master	Yes*	No	No
35	19	DMA Error	Yes	No	No
36	20	DMA Channel 0 Done	Yes	No	No
37	21	DMA Channel 1 Done	Yes	No	No
38	22	DMA Channel 2 Done	Yes	No	No
39	23	DMA Channel 3 Done	Yes	No	No
40	24	DMA Channel 4 Done	Yes	No	No
41	25	DMA Channel 5 Done	Yes	No	No
42	26	DMA Channel 6 Done	Yes	No	No
43	27	DMA Channel 7 Done	Yes	No	No
44	28	DMA Channel 8 Done	Yes	No	No
45	29	DMA Channel 9 Done	Yes	No	No
46	30	DMA Channel 10 Done	Yes	No	No
47	31	DMA Channel 11 Done	Yes	No	No
48	32	DMA Channel 12 Done	Yes	No	No
49	33	DMA Channel 13 Done	Yes	No	No
50	34	DMA Channel 14 Done	Yes	No	No
51	35	DMA Channel 15 Done	Yes	No	No
52	36	SPORT0A	Yes	No	No
53	37	SPORT0B	Yes	No	No
54	38	Crypto	Yes	No	No
55	39	DMA Channel 24 Done	Yes	No	No
56	40	General Purpose Timer 2	Yes	No	No
57	41	Crystal Oscillator	Yes	No	No
58	42	SPI1	Yes	No	No
59	43	PLL	Yes	No	No
60	44	Random Number Generator	Yes	No	No
61	45	Beeper	Yes	No	No

Table 3-2: Interrupt Sources (Continued)

Exception Number	IRQ Number	Vector	Wake-up From		
			Flexi	Hibernate	Shutdown
62	46	ADC	Yes	No	No
63	47-55	Reserved	-	-	-
72	56	DMA Channel 16 Done	Yes	No	No
73	57	DMA Channel 17 Done	Yes	No	No
74	58	DMA Channel 18 Done	Yes	No	No
75	59	DMA Channel 19 Done	Yes	No	No
76	60	DMA Channel 20 Done	Yes	No	No
77	61	DMA Channel 21 Done	Yes	No	No
78	62	DMA Channel 22 Done	Yes	No	No
79	63	DMA Channel 23 Done	Yes	No	No

* Corresponding PCLK is required to generate the interrupt.

Internally, the highest user-programmable priority (0) is treated as the fourth priority, after a reset, NMI, and a hard fault. Note that 0 is the default priority for all programmable priorities.

If the same priority level is assigned to two or more interrupts, their hardware priority (the lower the position number) determines the order in which the MCU activates them. For example, if both SPI0 and SPI1 are Priority Level 1, SPI0 has higher priority.

For more information on the exceptions and interrupts, refer to Chapter 5 (Exceptions) and Chapter 8 (Nested Vectored Interrupt Controller) in the *ARM Cortex-M3 Technical Reference Manual*.

Handling Interrupt Registers

In the Interrupt Service Routine (ISR) for any interrupt source, the first action usually taken is clearing of the interrupt source. In case of write-1-to-clear interrupts, the interrupt status register must be written to clear the interrupt source. Due to internal delays in the bus matrix, this write may be delayed before it reaches the destination. Meanwhile, the ISR execution might have been completed and there is a possibility of mistaking the previous interrupt for a second one. Therefore, it is always recommended to ensure that the interrupt source has been cleared before exiting the ISR. This can be done by reading the interrupt status again at the end of the ISR.

External Interrupt Configuration

Four external interrupts are implemented. These external interrupts can be separately configured to detect any combination of the following types of events:

- Rising edge: The logic detects a transition from low to high and generates a pulse. Only one pulse is sent to the Cortex-M3 per rising edge.

- Falling edge: The logic detects a transition from high to low and generates a pulse. Only one pulse is sent to the Cortex-M3 per falling edge.
- Rising or falling edge: The logic detects a transition from low to high or high to low and generates a pulse. Only one pulse is sent to the Cortex-M3 per edge.
- High level: The logic detects a high level. The appropriate interrupt is asserted and sent to the Cortex-M3. The interrupt line is held asserted until the external source deasserts. The high level needs to be maintained for one core clock cycle minimum to be detected.
- Low level: The logic detects a low level. The appropriate interrupt is asserted and sent to the Cortex-M3. The interrupt line is held asserted until the external source deasserts. The low level needs to be maintained a minimum of one core clock cycle to be detected.

The external interrupt detection unit block is always on and allows external interrupt to wake up the device when in hibernate and shutdown modes.

Apart from the four external interrupts, activity (any combination of events listed above) on the UART0_RX pin can also be configured to wake up the device from Flexi or hibernate modes. Interrupt No. 4 is assigned to both External Interrupt 3 and UART0_RX pin. Both pins can be configured to generate Interrupt No.4. The `XINT_CFG0.UART_RX_EN` bit must be set to enable UART0_RX pin based wake-up. `XINT_EXT_STAT.STAT_UART_RXWKUP` bit is set whenever Interrupt No. 4 is asserted due to an event on the UART0_RX pin.

NOTE: For the use of the external interrupt on the corresponding pads, the pads must be configured as GPIOs and the corresponding GPIO input enable should be set high. For example, when using the pad `p0_15` as external interrupt, set the `GP0CON[31:30]` as 2'b00 and `GP0IE[15]` as 1'b1.

ADuCM302x XINT Register Descriptions

External interrupt configuration (XINT) contains the following registers.

Table 3-3: ADuCM302x XINT Register List

Name	Description
XINT_CFG0	External Interrupt Configuration
XINT_CLR	External Interrupt Clear
XINT_EXT_STAT	External Wakeup Interrupt Status
XINT_NMICLR	Non-Maskable Interrupt Clear

External Interrupt Configuration

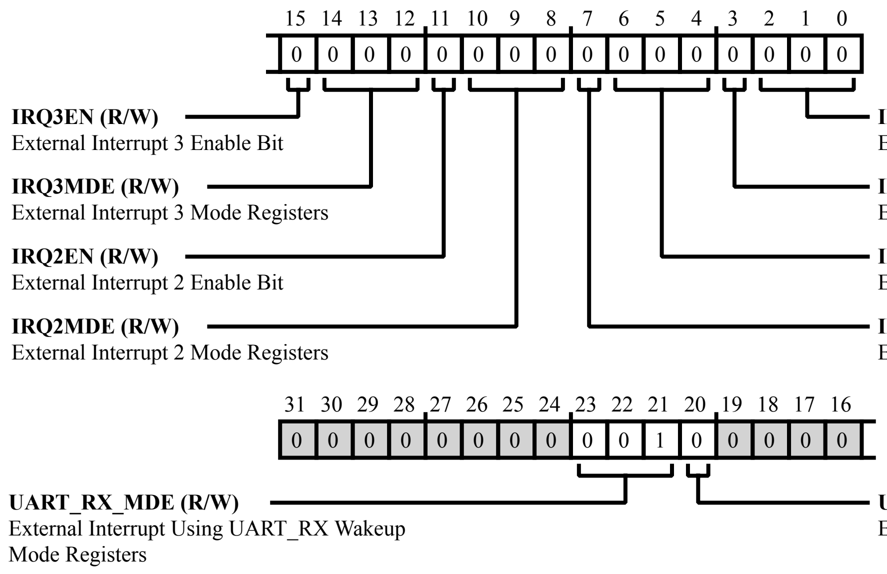

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IRQ3EN (R/W)
External Interrupt 3 Enable Bit
IRQ3MDE (R/W)
External Interrupt 3 Mode Registers
IRQ2EN (R/W)
External Interrupt 2 Enable Bit
IRQ2MDE (R/W)
External Interrupt 2 Mode Registers
IRQ0MDE (R/W)
External Interrupt 0 Mode Registers
IRQ0EN (R/W)
External Interrupt 0 Enable Bit
IRQ1MDE (R/W)
External Interrupt 1 Mode Registers
IRQ1EN (R/W)
External Interrupt 1 Enable Bit
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
UART_RX_MDE (R/W)
External Interrupt Using UART_RX Wakeup Mode Registers
UART_RX_EN (R/W)
External Interrupt Enable Bit

Figure 3-1: XINT_CFG0 Register Diagram

Table 3-4: XINT_CFG0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:21 (R/W)	UART_RX_MDE	External Interrupt Using UART_RX Wakeup Mode Registers.	
		0	Rising edge
		1	Falling edge
		2	Rising or falling edge
		3	High level
		4	Low level
		5	Falling edge (same as 001)
		6	Rising or falling edge (same as 010)
		7	High level (same as 011)
20 (R/W)	UART_RX_EN	External Interrupt Enable Bit. This bit enables 'UART_RX' pin to generate interrupt on Interrupt no. 4 (Refer to the Table 3-2: List of System Exceptions).	
		0	UART_RX wakeup interrupt is disabled
		1	UART_RX wakeup interrupt is enabled

Table 3-4: XINT_CFG0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	IRQ3EN	External Interrupt 3 Enable Bit.	
		0	External Interrupt 3 disabled
		1	External Interrupt 3 enabled
14:12 (R/W)	IRQ3MDE	External Interrupt 3 Mode Registers.	
		0	Rising edge
		1	Falling edge
		2	Rising or falling edge
		3	High level
		4	Low level
		5	Falling edge (same as 001)
		6	Rising or falling edge (same as 010)
		7	High level (same as 011)
11 (R/W)	IRQ2EN	External Interrupt 2 Enable Bit.	
		0	External Interrupt 2 disabled
		1	External Interrupt 2 enabled
10:8 (R/W)	IRQ2MDE	External Interrupt 2 Mode Registers.	
		0	Rising edge
		1	Falling edge
		2	Rising or falling edge
		3	High level
		4	Low level
		5	Falling edge (same as 001)
		6	Rising or falling edge (same as 010)
		7	High level (same as 011)
7 (R/W)	IRQ1EN	External Interrupt 1 Enable Bit.	
		0	External Interrupt 0 disabled
		1	External Interrupt 0 enabled
6:4 (R/W)	IRQ1MDE	External Interrupt 1 Mode Registers.	
		0	Rising edge
		1	Falling edge

Table 3-4: XINT_CFG0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		2	Rising or falling edge
		3	High level
		4	Low level
		5	Falling edge (same as 001)
		6	Rising or falling edge (same as 010)
		7	High level (same as 011)
3 (R/W)	IRQ0EN	External Interrupt 0 Enable Bit.	
		0	External Interrupt 0 disabled
		1	External Interrupt 0 enabled
2:0 (R/W)	IRQ0MDE	External Interrupt 0 Mode Registers.	
		0	Rising edge
		1	Falling edge
		2	Rising or falling edge
		3	High level
		4	Low level
		5	Falling edge (same as 001)
		6	Rising or falling edge (same as 010)
		7	High level (same as 011)

External Interrupt Clear

This register has W1C bits that are used to clear the corresponding XINT_EXT_STAT bits.

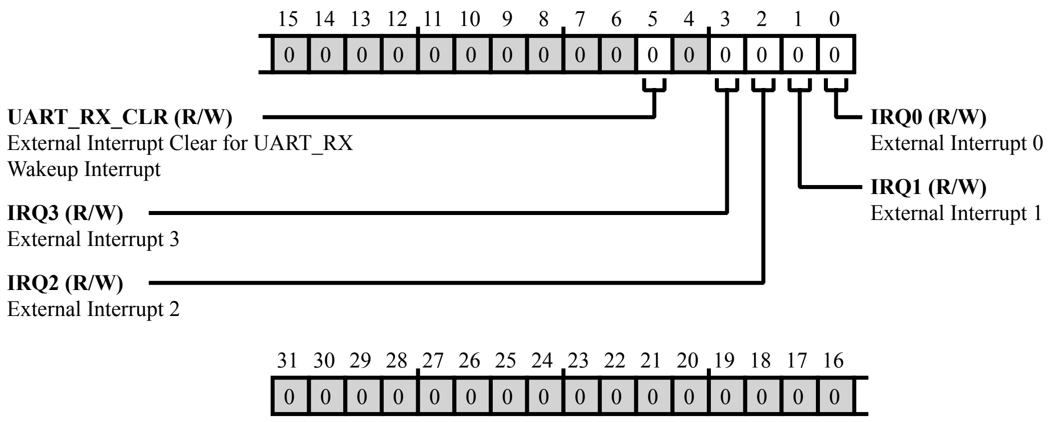


Figure 3-2: XINT_CLR Register Diagram

Table 3-5: XINT_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W)	UART_RX_CLR	External Interrupt Clear for UART_RX Wakeup Interrupt. Set to 1 to clear interrupt status flag. Cleared automatically by hardware.
3 (R/W)	IRQ3	External Interrupt 3. Set to 1 to clear interrupt status flag. Cleared automatically by hardware.
2 (R/W)	IRQ2	External Interrupt 2. Set to 1 to clear interrupt status flag. Cleared automatically by hardware.
1 (R/W)	IRQ1	External Interrupt 1. Set to 1 to clear interrupt status flag. Cleared automatically by hardware.
0 (R/W)	IRQ0	External Interrupt 0. Set to 1 to clear interrupt status flag. Cleared automatically by hardware.

External Wakeup Interrupt Status

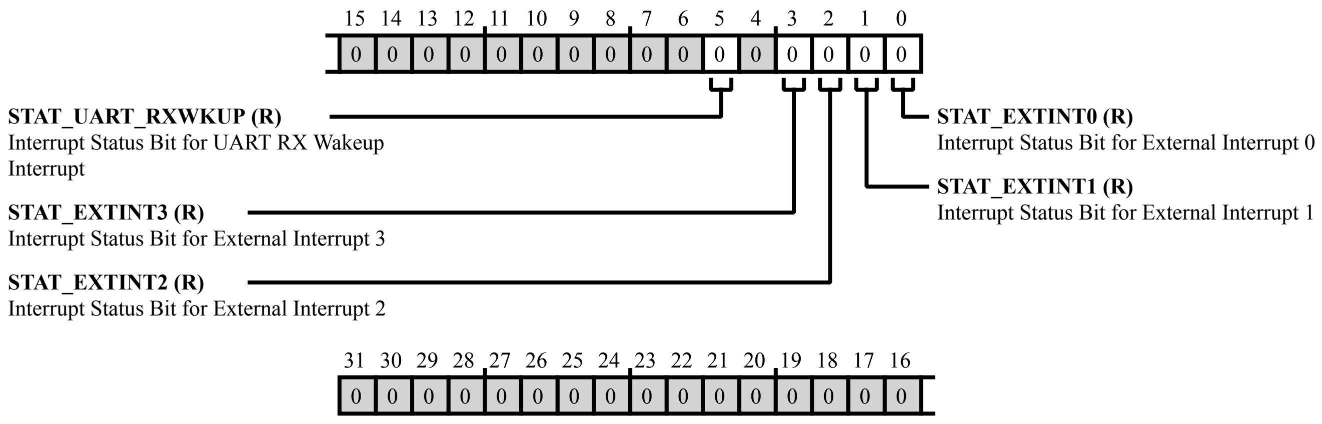


Figure 3-3: XINT_EXT_STAT Register Diagram

Table 3-6: XINT_EXT_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/NW)	STAT_UART_RXWKUP	Interrupt Status Bit for UART RX Wakeup Interrupt. This bit is valid if an interrupt asserted on INTERRUPT No. 4. 0 - UART_RX Wakeup did not generate the interrupt 1 - UART_RX Wakeup generated the interrupt Read-only register bit. Cleared by writing 1 to `XINT_CLR.UART_RX_CLR` bit. Note : Interrupt No. 4 is shared with External Interrupt 3, UART RX wakeup.
3 (R/NW)	STAT_EXTINT3	Interrupt Status Bit for External Interrupt 3. This bit is valid if there is an interrupt asserted on INTERRUPT No 4 . 0 - External interrupt 3 did not generate the interrupt 1 - External interrupt 3 generated the interrupt Read-only register bit. Cleared by writing 1 to `XINT_CLR.IRQ3` bit.
2 (R/NW)	STAT_EXTINT2	Interrupt Status Bit for External Interrupt 2. This bit is valid if there is an interrupt asserted on INTERRUPT No. 3. 0 - External interrupt 2 did not generate the interrupt 1 - External interrupt 2 generated the interrupt Read-only register bit. Cleared by writing 1 to `XINT_CLR.IRQ2` bit.

Table 3-6: XINT_EXT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/NW)	STAT_EXTINT1	Interrupt Status Bit for External Interrupt 1. This bit is valid if there is an interrupt asserted on INTERRUPT No. 2. 0 - External interrupt 1 did not generate the interrupt 1 - External interrupt 1 generated the interrupt Read-only register bit. Cleared by writing 1 to `XINT_CLR.IRQ1` bit.
0 (R/NW)	STAT_EXTINT0	Interrupt Status Bit for External Interrupt 0. This bit is valid if there is an interrupt asserted on INTERRUPT No. 1. 0 - External interrupt 0 did not generate the interrupt 1 - External interrupt 0 generated the interrupt Read-only register bit. Cleared by writing 1 to `XINT_CLR.IRQ0` bit.

Non-Maskable Interrupt Clear

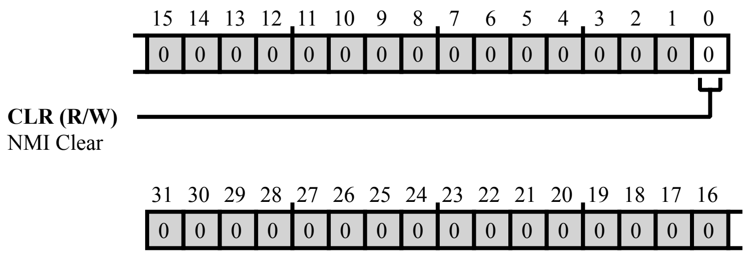


Figure 3-4: XINT_NMICLR Register Diagram

Table 3-7: XINT_NMICLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	CLR	NMI Clear. Set to 1 to clear an interrupt status flag when the NMI interrupt is set. Cleared automatically by hardware.

4 Power Management (PMG)

This chapter provides an overview of the functional architecture and implementation of power management and power modes used in the ADuCM302x MCU.

Power Management Features

The features include:

- High efficiency buck converters to reduce power.
    - Buck converter for active mode. Needs two external flying capacitors.
- Customized clock gating for active and Flexi modes.
- Power gating to reduce leakage in deep sleep modes.
- Voltage monitoring.
- Flexible sleep modes with smart peripherals.
- Deep sleep modes with and without retention.

Power Management Functional Description

This section provides information on the power management function of the ADuCM302x MCU.

The system has two voltage domains:

- VBAT: Input voltage with a range from 1.74 V - 3.6 V
- VREG: Internally generated voltage with a range of 1.2 V ± 10%

Power-up Sequence

The systems powers up when the battery is above 1.6 V. The buck converter is disabled during the power up sequence, so the system always starts in LDO mode. The user can switch to buck mode if required. The buck provides the lowest dynamic power at the expense of four extra pins and two flying caps. User can tradeoff between external components and active power. The buck can be enabled by writing to bit 0 of the `PMG_CTL1` register.

Operating Modes

The ADuCM302x MCU supports the following power modes:

- Active Mode: Cortex-M3 executes from flash/SRAM.
- Flexi Mode: Cortex-M3 is disabled. User selects the peripherals to be enabled: DMA/SPI, DMA/I^{2}C and so on.
- Hibernate Mode: System power gated. 8 KB of SRAM is always retained. Up to an addition of 24 KB SRAM can be selected to be retained.
- Shutdown Mode: Non-retain state. Pins retain values.

Active Mode

In this mode, the part is up and running. The Cortex-M3 executes instructions from flash and/or SRAM.

The system is not only clock gated using automatic clock gating techniques, but also clock gates can be selected using the `CLKG_CLK_CTL5` register. Most of the peripherals are automatically clock gated when disabled. The clock runs only when the peripheral is enabled. An exception to this is I2C, UART, and general purpose timer. Those blocks must be manually clock gated using the `CLKG_CLK_CTL5` register.

Writing 1 to the bit in the `CLKG_CLK_CTL5` register stops the corresponding clock to the peripheral. After the clock stops, if the user/software accesses any register in that peripheral, the clock is automatically enabled . Also, writing 0 to the bit in the `CLKG_CLK_CTL5` register enables the corresponding clock to the peripheral.

The `CLKG_CLK_CTL5.PERCLKOFF` bit can be used to disable all peripherals. This is useful when the Cortex-M3 processor is executing exclusively from SRAM or flash, and no other peripherals are required.

When the `CLKG_CLK_CTL5.PERCLKOFF` bit is 1, the clock is gated. The clock is re-enabled if the MCU or DMA is accessing the register of any gated peripherals.

The chips power-up with the LDO. Buck can be enabled to save power consumption, by writing to the `PMG_CTL1` register.

Flexi Mode

In this mode, the Cortex-M3 is always disabled. The core can be switched to this mode by setting the `PMG_PWRMOD.MODE` bit to 0 and executing WFI instruction.

Writing 1 to the bit in the `CLKG_CLK_CTL5` register stops the corresponding clock to the peripheral.

After the clock stops, if the user/software accesses any register in that peripheral , the clock is auto enabled . Also, writing 0 to the bit in `CLKG_CLK_CTL5` register enables the corresponding clock to the peripheral.

The clock for all other blocks in the system runs if the peripheral is enabled.

This mode adds flexibility to the user to determine the blocks that are enabled while the Cortex is sleeping. For example, the user can perform DMA transactions through SPI or I2C, or ADC conversions to DMA without MCU interaction. This mode can be used to reduce active power when an activity is expected to complete (for example, reading a certain number of bytes from a sensor, and so on) before the MCU can be woken up for processing the data.

Hibernate Mode

In this mode, the Cortex-M3 and all digital peripherals are OFF. The SRAM does not retain the entire 64 KB, but only the lower 32 KB, 24 KB, 16 KB, or 8 KB. Important information must be kept on the lower SRAM before moving to hibernate mode.

As a user, you can select:

1. The amount of SRAM to retain 16 KB or 8 KB or both. This is in addition to the 8 KB of SRAM always retained in hibernate mode. This selection is controlled using the `PMG_TST_SRAM_CTL` register.
2. The RTC/WakeUp Timer with a 32 kHz crystal or with the internal 32 kHz oscillator. The crystal provides a higher accuracy clock at the expense of extra power consumption.
3. Control battery monitoring during hibernate mode. The regulated 1.2 V supply is always monitored to guarantee data is never corrupted by the supply going below the minimum retention voltage. If the regulated supply falls below 1 V, the chip resets before any data is corrupted. Though the regulated supply is always monitored, there is an option to also monitor the battery in hibernate mode. This is done using the `PMG_PWRMOD` register.

NOTE: Switch to HFOSC before going to hibernate mode if running from PLL, external clock, or HFXTAL.

For more information, refer to Configuring Hibernate Mode.

Memory Configuration

There is no option to retain the higher 32 KB of SRAM in hibernate mode. The lowest 8 KB is always retained in hibernate mode, irrespective of the configuration set in the `PMG_TST_SRAM_CTL` register.

For more information, refer to SRAM Region.

Shutdown Mode

This is the deepest sleep mode where all the digital and analog circuits are powered down except for a wake up controller and a failsafe. These circuits are powered from the battery voltage. The LDO is off, so the 1.2 V region is shutdown.

The state of the digital core is not retained and the SRAM memory content is not preserved. When the part wakes up from shutdown mode, it follows the POR sequence and the code execution starts from the beginning. The user can read the `PMG_SHDN_STAT` register to see the wake up source from this mode. There are four possible wake up sources:

- External interrupts, limited to three external interrupts to save power.
- External reset.

- Battery falling below 1.6 V.
- RTC Timer (if this option is enabled by the customer). Only available with the crystal oscillator. The LF Oscillator is Off during shutdown.

During this mode, the state of the pads and the wakeup interrupt configuration are preserved.

The configuration of the pads is preserved, but it is also locked after waking up from shutdown mode. The user needs to unlock the state of the pads by writing the value 0x58FA to register `PMG_TST_CLR_LATCH_GPIOS`. It is recommended to perform the write inside the ISR routine.

The user can only go to shutdown with the HFOSC. It will not go to shutdown mode if running from PLL, external clock or HFXTAL. The user needs to switch to HFOSC before going to shutdown mode.

There is also a scratch pad available for the user during shutdown mode. The scratch pad consists of a 32-bit register. The intention of the scratch pad is for the user to save some data on 3 V before all the information is lost in shutdown mode.

The user can write to the `PMG_TST_SCRPAD_IMG` at any time. This is a write-read register. The information saved in this register is copied to an area in VBAT before going to shutdown mode. This information in VBAT can be accessed through `PMG_TST_SCRPAD_3V_RD`, which is a read-only register.

NOTE: Switch to HFOSC before going to shutdown mode if running from PLL, external clock, or HFXTAL.

For more information, refer to Configuring Shutdown Mode.

Programming Sequence

Configuring Hibernate Mode

This is the lowest power-down mode with state retention. Digital information and SRAM 0 is always retained, but retaining SRAM 1 and SRAM2 is optional. Refer to SRAM Region.

It is also optional to monitor the battery in hibernate mode. The `PMG_CTL1` register provides information about the battery state. These bits also generate interrupts. Therefore, the battery is within safe margin voltages 3.6 V and 2.75 V. The battery voltage between 2.75 V and 2.3 V in the battery region 1 (BR1) can also be considered a safe region. However, it is an indication that the battery is decaying. In the battery region 2 (BR2), the battery voltage below 2.3 V is an indication that the battery is dying.

NOTE: If the battery voltage is in safe or battery region 1 (BR1), stop monitoring the battery when entering hibernate mode.

1. Set `PMG_PWRMOD.MODE` = 2.
2. Select `SLEEPDEEP` bit in the System Control Register (SCR) of the Cortex-M3.
3. Enable the desired wake-up interrupt.
4. If a wake-up from RTC1 is required, select the desired clock source (LFOSC or crystal).

5. If battery monitor is required, set the `PMG_PWRMOD.MONVBATN` bit to 0.
6. Execute the WFI or WFE instruction.

Now, the chip is in hibernate mode.

When a wake-up interrupt arrives, the device exits hibernate mode and serves the interrupt routine.

NOTE: Switch to HFOSC before going to hibernate mode if running from PLL, external clock, or HFXTAL.

Configuring Shutdown Mode

This is the deepest power-down mode. State information is not retained in this mode and the chip restarts from reset after waking up.

The wake-up sources available are external reset, external interrupts limited only to three and RTC0 with the crystal oscillator if the application needs periodic wake ups. The internal low frequency oscillator is disabled during this mode. The only clock available in this mode is the low frequency crystal (optional).

The following steps describe the sequence to enter this mode:

1. Set `PMG_PWRMOD.MODE` = 3.
2. Set the `SLEEPDEEP` bit in the SCR register of the Cortex-M3.
3. Enable the desired wake-up interrupt.
4. If a wake-up from RTC0 is required, enable LFXTAL.
5. Execute the WFI instruction.

Now, the chip is in shutdown mode.

NOTE: Switch to HFOSC before going to shutdown mode if running from PLL, external clock, or HFXTAL.

When a wake-up interrupt arrives, the device exits shutdown mode and the chip wakes up to a POR scenario.

The sequence is as follows:

1. Read the `PMG_SHDN_STAT` register.
2. Enable the interrupt routine that caused the wake-up event.
3. The chip jumps to the enabled ISR and served the interrupt routine.

NOTE: Write `PMG_PWRMOD` back to 0 in the ISR after waking up from shutdown mode.

Wake-up Sequence

This section describes the wake-up mechanism for different power modes. The wake-up is triggered by an interrupt or a reset. The sequence for the wake-up is different depending on the power mode.

If the wake-up is triggered by a coming interrupt, then the system first executes the interrupt routine.

The wake-up sequence is different for shutdown mode. The information is not retained, and therefore the system will always start from reset state after waking up. The interrupt triggering the wake-up sequence remains pending in the Cortex-M3 until interrupt routine is enabled at the restart of the code. The status register can be read after waking up from shutdown mode and the interrupt routine can be enabled.

For more information about interrupts and wake-up sources for different power modes, refer to Table 3-2 Interrupt Sources.

Monitor Voltage Control

Voltage Supervisory circuits are enabled at all times to guarantee that the battery and the regulated supply are always within operating levels. The circuit monitoring these supplies is called Power Management (PMG).

The main features of the PMG during active mode are:

- Monitor battery voltage
  - Generates an alarm to the MCU if battery supply is below 1.83 V
  - Generates a reset to the chip if battery supply is below 1.6 V
  - Monitors the state of the battery.

    The following battery regions are defined:

    Battery between: 3.6 V - 2.75 V

    Battery between: 2.75 V - 2.3 V. Generates interrupt BR1

    Battery between: 2.3 V - 1.6 V. Generates interrupt BR2
- Monitors regulated supply
  - Generates an interrupt if the regulated supply is above 1.32 V (overvoltage)
  - Generates an interrupt if the regulated supply is below 1.1 V (undervoltage)
  - Generates a reset if the regulated supply is below 1.08 V

The main features for the PMG during hibernate mode are:

- Monitor battery voltage
  - Generates an alarm to the MCU if supply is below 1.83 V - Optional
  - Generates a reset to the chip if supply is below 1.6 V - Optional
  - Monitors the state of the battery - Optional

    Battery between: 3.6 V - 2.75 V

    Battery between: 2.75 V - 2.3 V. Generates interrupt BR1

    Battery between: 2.3 V - 1.6 V. Generates interrupt BR2

- Monitors regulated supply
    - Generates a reset if the regulated supply is below 1.08 V

There is also the possibility to have an interrupt when the voltage is between 3.6 V - 2.75 V. The battery and VREG supply monitors are shown below.

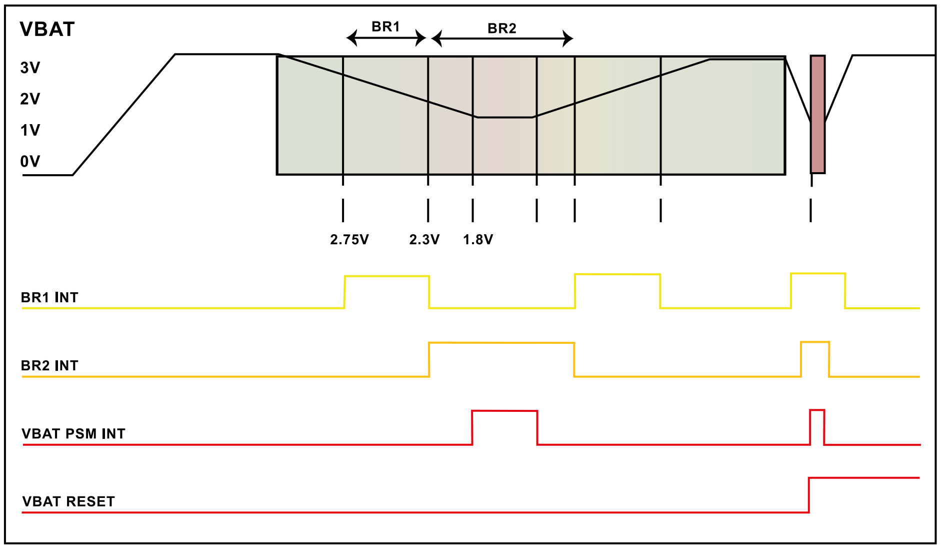


Figure 4-1: Battery Supply Monitor

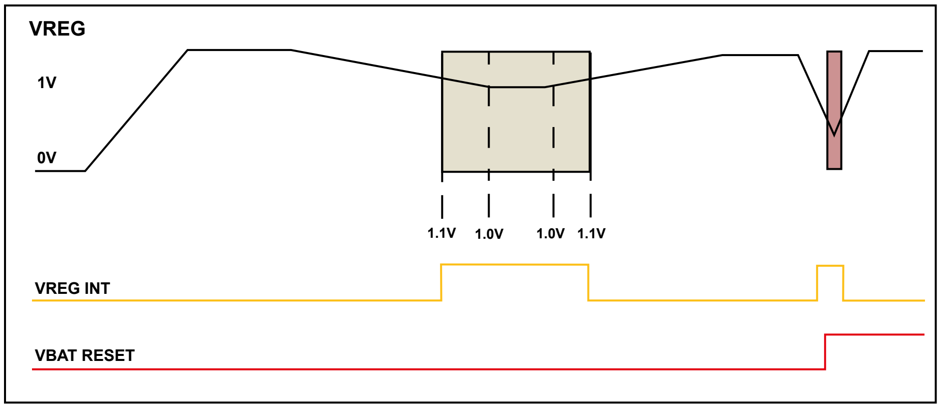


Figure 4-2: VREG Supply Monitor

ADuCM302x PMG Register Descriptions

Power Management Registers (PMG) contains the following registers.

Table 4-1: ADuCM302x PMG Register List

Name	Description
PMG_CTL1	HP Buck Control
PMG_IEN	Power Supply Monitor Interrupt Enable
PMG_PSM_STAT	Power Supply Monitor Status
PMG_PWRKEY	Key Protection for PMG_PWRMOD and PMG_SRAMRET
PMG_PWRMOD	Power Mode Register
PMG_RST_STAT	Reset Status
PMG_SHDN_STAT	Shutdown Status Register
PMG_SRAMRET	Control for Retention SRAM in Hibernate Mode

HP Buck Control

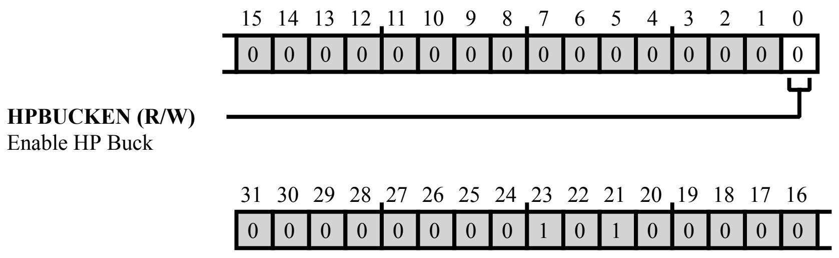


Figure 4-3: PMG_CTL1 Register Diagram

Table 4-2: PMG_CTL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	HPBUCKEN	Enable HP Buck.

Power Supply Monitor Interrupt Enable

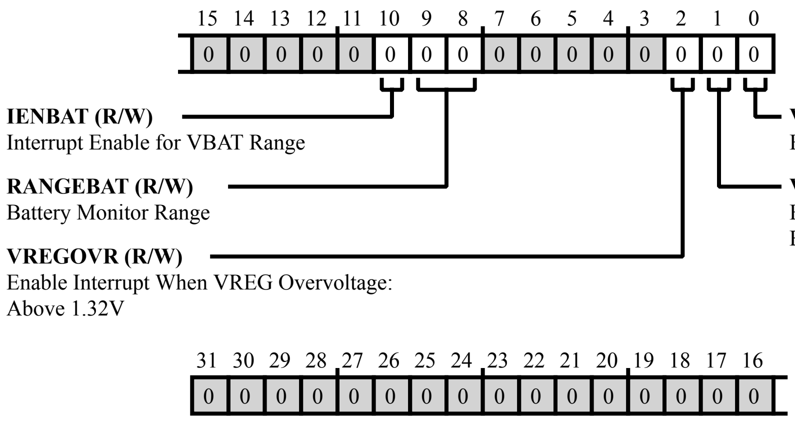


Figure 4-4: PMG_IEN Register Diagram

Table 4-3: PMG_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W)	IENBAT	Interrupt Enable for VBAT Range. Set this bit if interrupt needs to be generated for appropriate `PMG_IEN.RANGEBAT`. Configure the appropriate `PMG_IEN.RANGEBAT` for which interrupt to be generated and then set this bit. Interrupt is generated if VBAT falls in the `PMG_IEN.RANGEBAT`. For example, if the battery is good, to monitor the battery, configure in `PMG_PSM_STAT.RANGE2` or `PMG_PSM_STAT.RANGE3`, clear all `PMG_PSM_STAT` flags and then enable this interrupt. Otherwise, `PMG_PSM_STAT.RANGE1` of the battery keeps firing the interrupt.	
9:8 (R/W)	RANGEBAT	Battery Monitor Range. Configure appropriate `PMG_IEN.RANGEBAT` for which interrupt has to be generated	
		0	Configure to generate interrupt if VBAT > 2.75 V
		1	Configure to generate interrupt if VBAT between 2.75 V - 1.6 V
		2	Configure to generate interrupt if VBAT between 2.3 V - 1.6 V
		3	N/A
2 (R/W)	VREGOVR	Enable Interrupt When VREG Overvoltage: Above 1.32V.	
1 (R/W)	VREGUNDR	Enable Interrupt When VREG Undervoltage: Below 1V. If enabled, the irq connects to NMI (non-maskable interrupt)	

Table 4-3: PMG_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	VBAT	Enable Interrupt for VBAT. If enabled, the irq connects to NMI (non-maskable interrupt) If enabled, it generates an interrupt if VBAT < 1.83 V

Power Supply Monitor Status

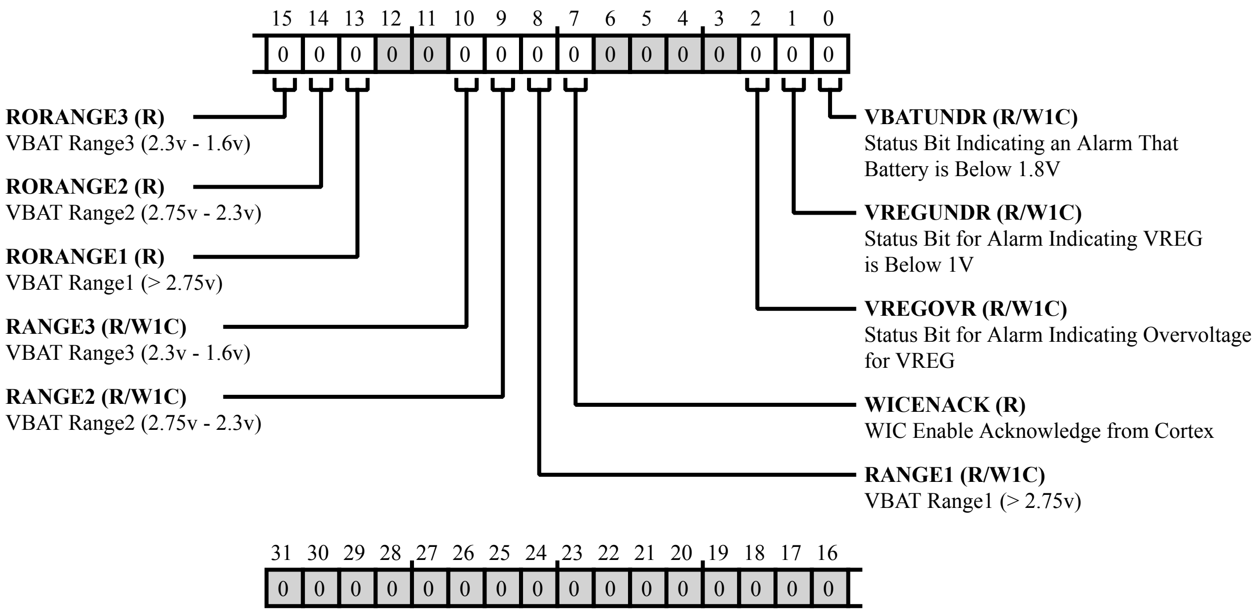


Figure 4-5: PMG_PSM_STAT Register Diagram

Table 4-4: PMG_PSM_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/NW)	RORANGE3	VBAT Range3 (2.3 V - 1.6 V). Read-only status bit.	
		0	VBAT not in the range specified
		1	VBAT in the range specified
14 (R/NW)	RORANGE2	VBAT Range2 (2.75 V - 2.3 V). Read-only status bit.	
		0	VBAT not in the range specified
		1	VBAT in the range specified
13 (R/NW)	RORANGE1	VBAT Range1 (> 2.75 V). Read-only status bit.	
		0	VBAT not in the range specified
		1	VBAT in the range specified

Table 4-4: PMG_PSM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W1C)	RANGE3	VBAT Range3 (2.3 V - 1.6 V). This is a write-one-to-clear status bit indicating the relevant VBAT range. Generates VBAT range interrupt if `PMG_IEN.IENBAT` is set. Note : The status bit is set again even after '1' is written (to clear the flag) to the flag, if VBAT falls in the range specified.	
		0	VBAT not in the range specified
		1	VBAT in the range specified
9 (R/W1C)	RANGE2	VBAT Range2 (2.75 V - 2.3 V). This is a write-one-to-clear status bit indicating the relevant VBAT range. Generates VBAT range interrupt if `PMG_IEN.IENBAT` is set. Note: The status bit is set again even after '1' is written (to clear the flag) to the flag, if VBAT falls in the range specified.	
		0	VBAT not in the range specified
		1	VBAT in the range specified
8 (R/W1C)	RANGE1	VBAT Range1 (> 2.75 V). This is a write-one-to-clear status bit indicating the relevant VBAT range. Generates VBAT range interrupt if `PMG_IEN.IENBAT` is set. Note : The status bit is set again even after '1' is written (to clear the flag) to the flag, if VBAT falls in the range specified.	
		0	VBAT not in the range specified
		1	VBAT in the range specified
7 (R/NW)	WICENACK	WIC Enable Acknowledge from Cortex.	
2 (R/W1C)	VREGOVR	Status Bit for Alarm Indicating Overvoltage for VREG. Generates an interrupt if `PMG_IEN.VREGOVR` is set and VREG (LDO out) > 1.32 V. This is write-1-to-clear bit. Note: The status bit is set again even after 1 is written (to clear the flag) to the flag, if VREG is > 1.32 V.	
1 (R/W1C)	VREGUNDR	Status Bit for Alarm Indicating VREG is below 1 V. Generates an interrupt if `PMG_IEN.VREGUNDR` is set and VREG < 1 V. This is a write-one-to-clear bit. Note : The status bit is set again even after '1' is written (to clear the flag) to the flag, if VREG is < 1 V	

Table 4-4: PMG_PSM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W1C)	VBATUNDR	Status Bit Indicating an Alarm that battery is below 1.8 V. Generates an interrupt if `PMG_IEN.VBAT` is set and VBAT < 1.83 V. This is a write-one-to-clear bit. Note : The status bit is again set even after 1 is written (to clear the flag) to the flag, if VBAT is < 1.83 V

Key Protection for PMG_PWRMOD and PMG_SRAMRET

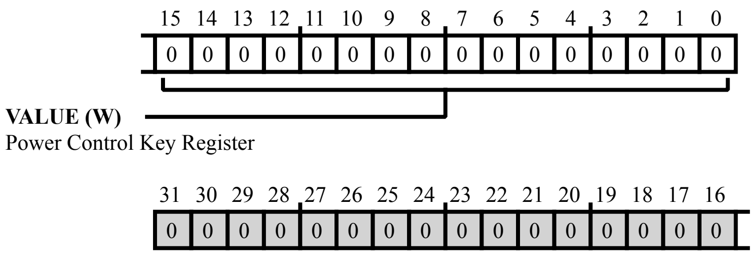


Figure 4-6: PMG_PWRKEY Register Diagram

Table 4-5: PMG_PWRKEY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (RX/W)	VALUE	Power Control Key Register. The PMG_PWRMOD and PMG_SRAMRET are key-protected registers. One write to the key is necessary to change the value in the PMG_PWRMOD and PMG_SRAMRET register. Key to be written is 0x4859. The PMG_PWRMOD and PMG_SRAMRET registers should then be written. A write to any other register on the APB bus before writing to PMG_PWRMOD or PMG_SRAMRET will return the protection to the lock state.

Power Mode Register

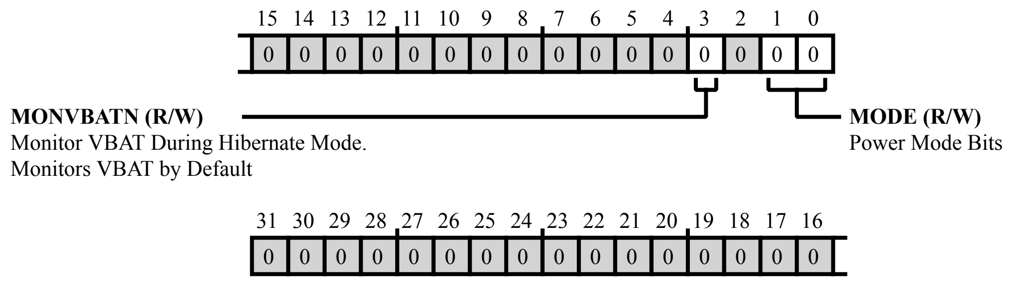


Figure 4-7: PMG_PWRMOD Register Diagram

Table 4-6: PMG_PWRMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	MONVBATN	Monitor VBAT During Hibernate Mode. Monitors VBAT by Default. VDD Monitoring cannot be disabled.	
		0	VBAT monitor enabled in PMG block.
		1	VBAT monitor disabled in PMG block.
1:0 (R/W)	MODE	Power Mode Bits.	
		0	Flexi Mode
		1	Reserved
		2	Hibernate Mode
		3	Shutdown Mode

Reset Status

This register is recommended to be read at the beginning of the user code to determine the cause of the reset. Default values of this register is unspecified as the cause of reset can be any source.

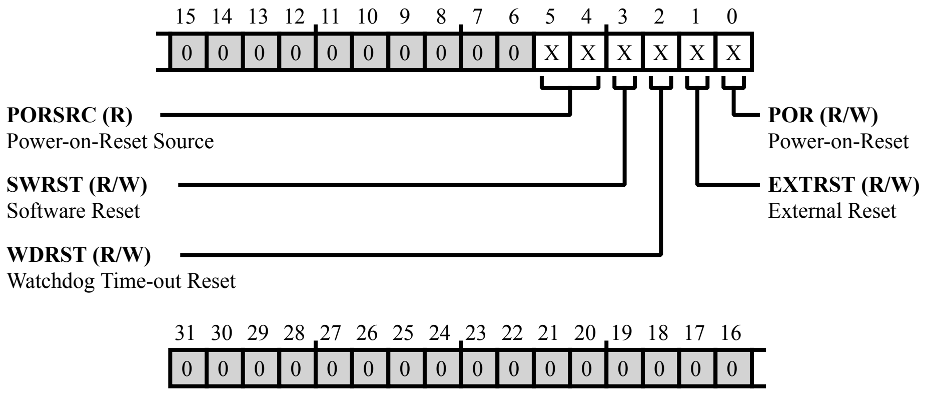


Figure 4-8: PMG_RST_STAT Register Diagram

Table 4-7: PMG_RST_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/NW)	PORSRC	Power-on-Reset Source. This bit contains additional details after a Power-on-Reset occurs.	
		0	POR triggered because VBAT drops below Fail Safe
		1	POR trigger because VBAT supply (VBAT < 1.7 V)
		2	POR triggered because VDD supply (VDD < 1.08 V)
		3	POR triggered because VREG drops below Fail Safe
3 (R/W)	SWRST	Software Reset. Set automatically to 1 when the system reset is generated. Cleared by writing 1 to the bit. This bit is also cleared when power-on-reset is triggered.	
2 (R/W)	WDRST	Watchdog Time-out Reset. Set automatically to 1 when a watchdog timeout occurs. Cleared by writing 1 to the bit. This bit is also cleared when power-on-reset is triggered	
1 (R/W)	EXTRST	External Reset. Set automatically to 1 when an external reset occurs. Cleared by writing 1 to the bit. This bit is also cleared when power-on-reset is triggered	
0 (R/W)	POR	Power-on-Reset. Set automatically when a Power-on-Reset occurs. Cleared by writing one to the bit.	

Shutdown Status Register

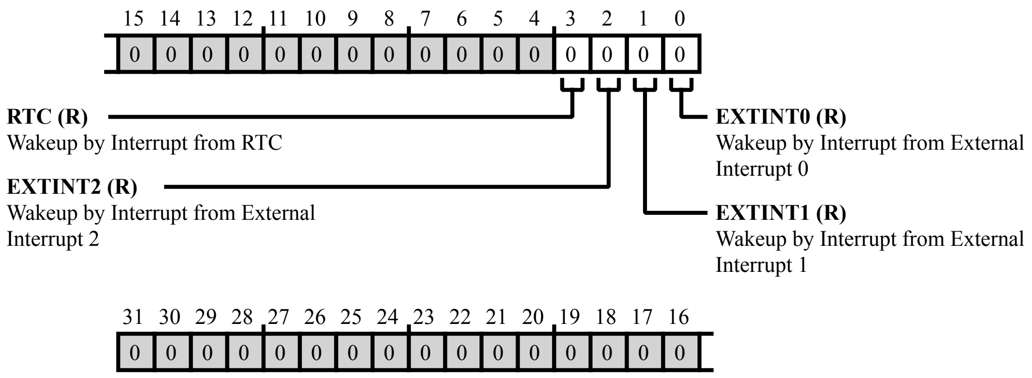


Figure 4-9: PMG_SHDN_STAT Register Diagram

Table 4-8: PMG_SHDN_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/NW)	RTC	Wakeup by Interrupt from RTC.
2 (R/NW)	EXTINT2	Wakeup by Interrupt from External Interrupt 2.
1 (R/NW)	EXTINT1	Wakeup by Interrupt from External Interrupt 1.
0 (R/NW)	EXTINT0	Wakeup by Interrupt from External Interrupt 0.

Control for Retention SRAM in Hibernate Mode

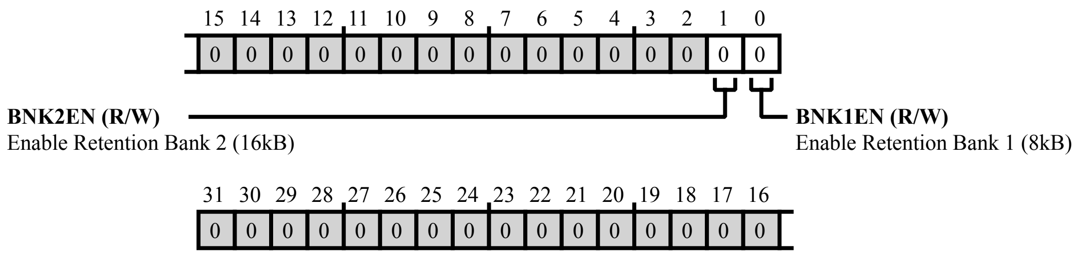


Figure 4-10: PMG_SRAMRET Register Diagram

Table 4-9: PMG_SRAMRET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	BNK2EN	Enable Retention Bank 2 (16 KB).
0 (R/W)	BNK1EN	Enable Retention Bank 1 (8 KB).

ADuCM302x PMG_TST Register Descriptions

Power Management Registers (PMG_TST) contains the following registers.

Table 4-10: ADuCM302x PMG_TST Register List

Name	Description
PMG_TST_CLR_LATCH_GPIOS	Clear GPIO After Shutdown Mode
PMG_TST_SCRPAD_3V_RD	Scratch Pad Saved in Battery Domain
PMG_TST_SCRPAD_IMG	Scratch Pad Image
PMG_TST_SRAM_CTL	Control for SRAM Parity and Instruction SRAM
PMG_TST_SRAM_INITSTAT	Initialization Status Register

Clear GPIO After Shutdown Mode

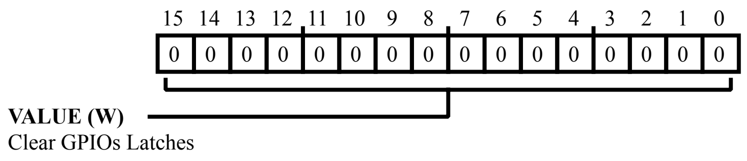


Figure 4-11: PMG_TST_CLR_LATCH_GPIOS Register Diagram

Table 4-11: PMG_TST_CLR_LATCH_GPIOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (RX/W)	VALUE	Clear GPIOs Latches. Writing 0x58FA creates a pulse to clear the latches for the GPIOs.

Scratch Pad Saved in Battery Domain

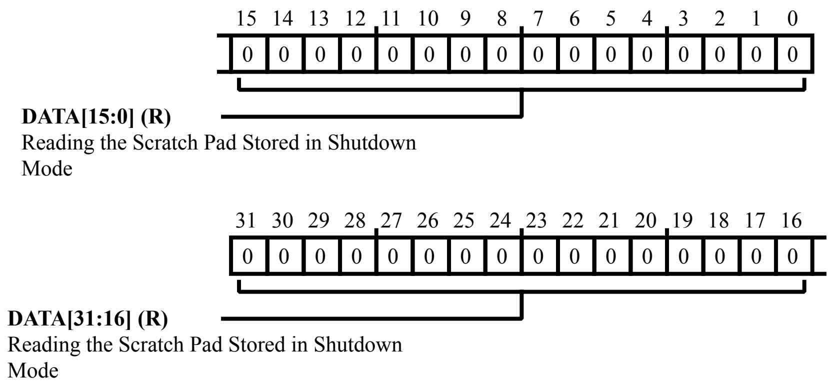


Figure 4-12: PMG_TST_SCRPAD_3V_RD Register Diagram

Table 4-12: PMG_TST_SCRPAD_3V_RD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	DATA	Reading the Scratch Pad Stored in Shutdown Mode. Read Only Register.

Scratch Pad Image

This register is useful in Shutdown mode. The GPIO configuration, OUT registers lose its contents when in shutdown mode. 32-bit Scratch register can be used to store the GPIO configuration. Note that the bits are not wide enough to store all GPIO related configuration and output values. It is envisaged that the user might have only few combination of shutdown port configurations. Hence these combination can be remembered by storing an equivalent encoded data in the scratch 32-bit registers. User can determine the GPIO configuration after reading back the `PMG_TST_SCRPAD_3V_RD` register (wakeup from shutdown).

Note: The content in the Scratch register is lost in shutdown mode if VBAT < 1.54 V

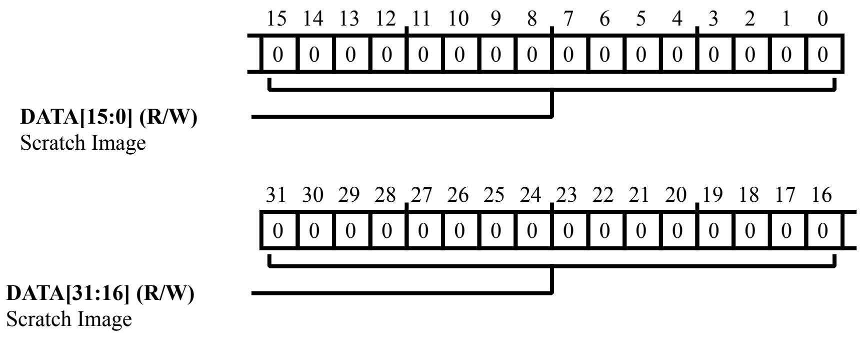


Figure 4-13: PMG_TST_SCRPAD_IMG Register Diagram

Table 4-13: PMG_TST_SCRPAD_IMG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Scratch Image. Anything written to this register will be saved in 3V when going to shutdown mode.

Control for SRAM Parity and Instruction SRAM

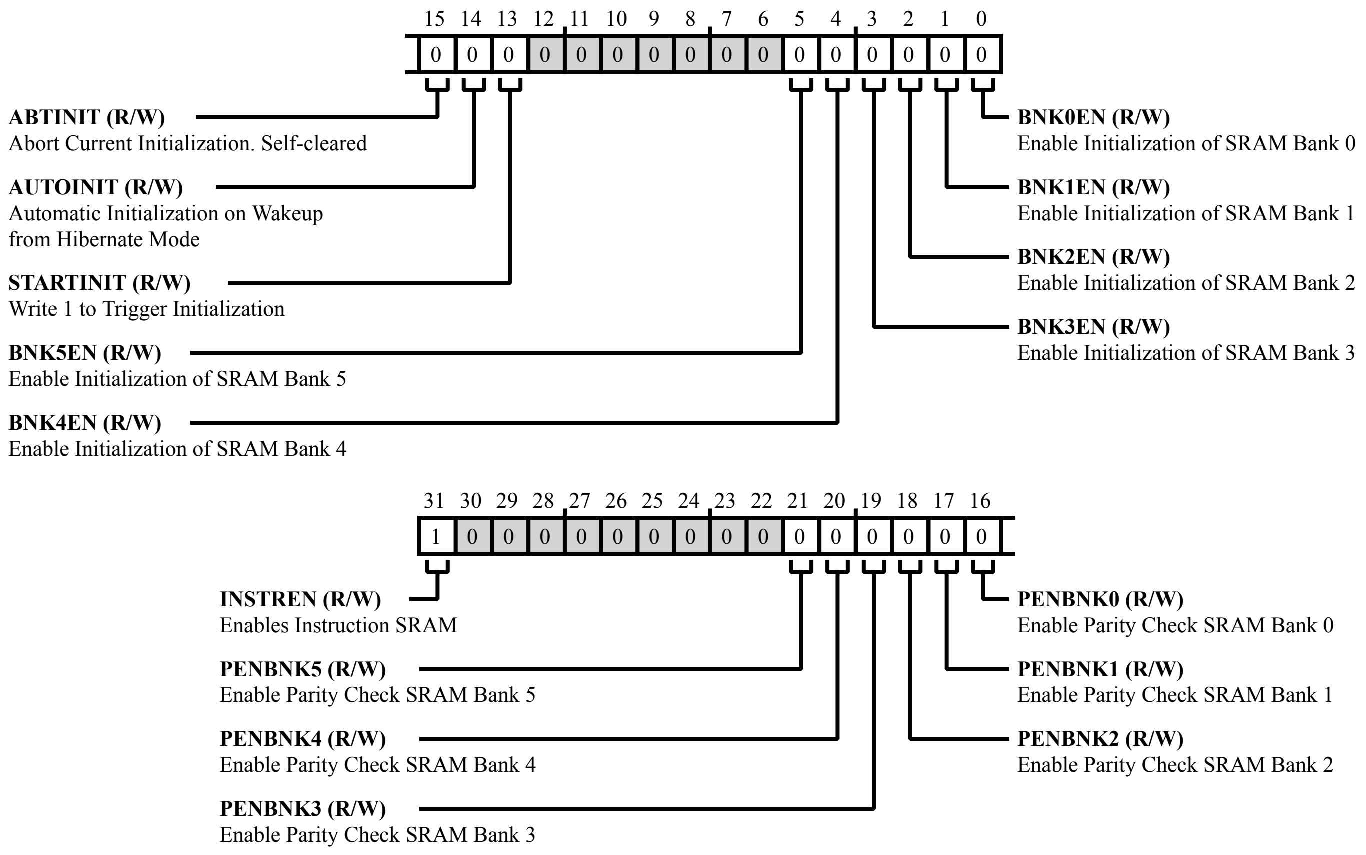

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ABTINIT (R/W)
Abort Current Initialization. Self-cleared
AUTOINIT (R/W)
Automatic Initialization on Wakeup from Hibernate Mode
STARTINIT (R/W)
Write 1 to Trigger Initialization
BNK5EN (R/W)
Enable Initialization of SRAM Bank 5
BNK4EN (R/W)
Enable Initialization of SRAM Bank 4
BNK0EN (R/W)
Enable Initialization of SRAM Bank 0
BNK1EN (R/W)
Enable Initialization of SRAM Bank 1
BNK2EN (R/W)
Enable Initialization of SRAM Bank 2
BNK3EN (R/W)
Enable Initialization of SRAM Bank 3
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INSTREN (R/W)
Enables Instruction SRAM
PENBNK5 (R/W)
Enable Parity Check SRAM Bank 5
PENBNK4 (R/W)
Enable Parity Check SRAM Bank 4
PENBNK3 (R/W)
Enable Parity Check SRAM Bank 3
PENBNK0 (R/W)
Enable Parity Check SRAM Bank 0
PENBNK1 (R/W)
Enable Parity Check SRAM Bank 1
PENBNK2 (R/W)
Enable Parity Check SRAM Bank 2

Figure 4-14: PMG_TST_SRAM_CTL Register Diagram

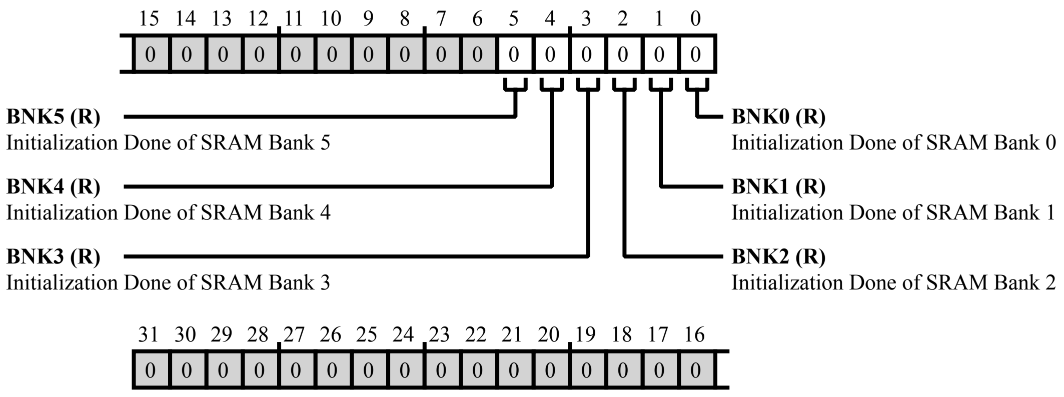
Table 4-14: PMG_TST_SRAM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	INSTREN	Enables Instruction SRAM. 0: Disable, 1: Enable
21 (R/W)	PENBNK5	Enable Parity Check SRAM Bank 5. 0: Disable, 1: Enable
20 (R/W)	PENBNK4	Enable Parity Check SRAM Bank 4. 0: Disable, 1: Enable
19 (R/W)	PENBNK3	Enable Parity Check SRAM Bank 3. 0: Disable, 1: Enable
18 (R/W)	PENBNK2	Enable Parity Check SRAM Bank 2. 0: Disable, 1: Enable

Table 4-14: PMG_TST_SRAM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/W)	PENBNK1	Enable Parity Check SRAM Bank 1. 0: Disable, 1: Enable
16 (R/W)	PENBNK0	Enable Parity Check SRAM Bank 0. 0: Disable, 1: Enable
15 (R/W)	ABTINIT	Abort Current Initialization. Self-cleared.
14 (R/W)	AUTOINIT	Automatic Initialization on Wakeup from Hibernate Mode.
13 (R/W)	STARTINIT	Write 1 to Trigger Initialization. Self-cleared.
5 (R/W)	BNK5EN	Enable Initialization of SRAM Bank 5. 0: Disable, 1: Enable
4 (R/W)	BNK4EN	Enable Initialization of SRAM Bank 4. 0: Disable, 1: Enable
3 (R/W)	BNK3EN	Enable Initialization of SRAM Bank 3. 0: Disable, 1: Enable
2 (R/W)	BNK2EN	Enable Initialization of SRAM Bank 2. 0: Disable, 1: Enable
1 (R/W)	BNK1EN	Enable Initialization of SRAM Bank 1. 0: Disable, 1: Enable
0 (R/W)	BNK0EN	Enable Initialization of SRAM Bank 0. 0: Disable, 1: Enable

Initialization Status Register



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
BNK5 (R)
Initialization Done of SRAM Bank 5
BNK4 (R)
Initialization Done of SRAM Bank 4
BNK3 (R)
Initialization Done of SRAM Bank 3
BNK0 (R)
Initialization Done of SRAM Bank 0
BNK1 (R)
Initialization Done of SRAM Bank 1
BNK2 (R)
Initialization Done of SRAM Bank 2
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 4-15: PMG_TST_SRAM_INITSTAT Register Diagram

Table 4-15: PMG_TST_SRAM_INITSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/NW)	BNK5	Initialization Done of SRAM Bank 5. 0:Not initialized; 1:Initialization completed
4 (R/NW)	BNK4	Initialization Done of SRAM Bank 4. 0:Not initialized; 1:Initialization completed
3 (R/NW)	BNK3	Initialization Done of SRAM Bank 3. 0:Not initialized; 1:Initialization completed
2 (R/NW)	BNK2	Initialization Done of SRAM Bank 2. 0:Not initialized; 1:Initialization completed
1 (R/NW)	BNK1	Initialization Done of SRAM Bank 1. 0:Not initialized; 1:Initialization completed
0 (R/NW)	BNK0	Initialization Done of SRAM Bank 0. 0:Not initialized; 1:Initialization completed

5 General Purpose Input/Output (GPIO)

This section describes the General Purpose Input/Output (GPIO) functionality.

GPIO Features

The GPIO ports include the following features:

- Input and output modes of GPIO operation.
- Port multiplexing controlled by individual pin-per-pin base.
- All port pins provide interrupt functionality.
- Programmable pull-up/pull-down drive compatibility.

GPIO Functional Description

The ADuCM302x MCU controls the GPIO pins through a set of Memory Mapped Registers (MMR). These MMRs are implemented as part of an APB32 peripheral. Multiple functions are mapped on most of the GPIO pins. These functions can be selected by appropriately configuring the corresponding ports of the `GPIO_CFG` registers.

GPIO Block Diagram

The figures show the block diagrams of the GPIO pins with pull-up and pull-down resistors.

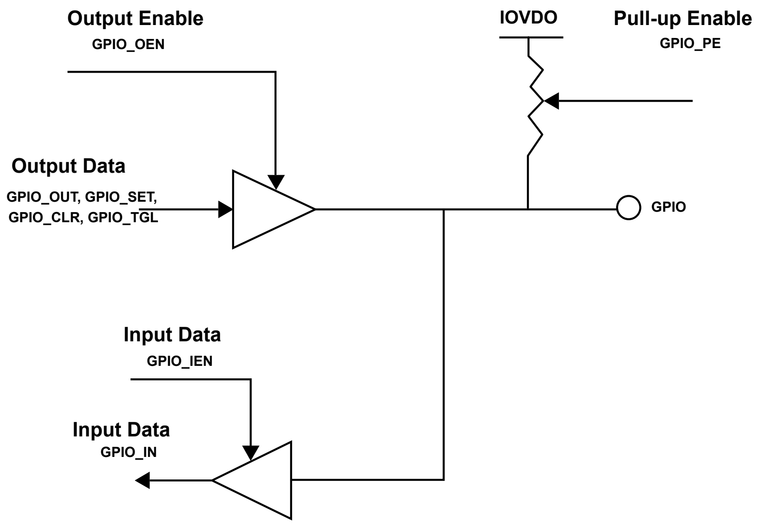


Figure 5-1: GPIO Pins with Pull-up Resistor

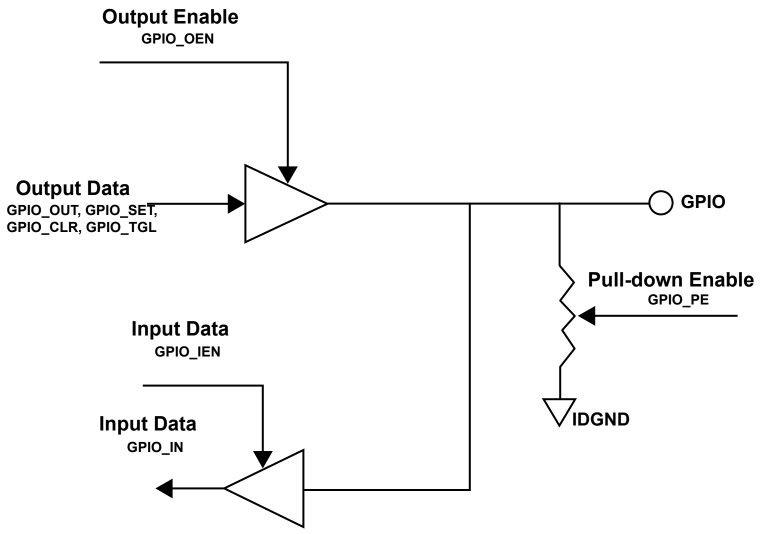


Figure 5-2: GPIO Pins with Pull-down Resistor

In applications where the input is not controlled by the system, the input voltage may exceed the maximum voltage rating of the device. When the external overvoltage conditions are applied to the ADuCM302x MCU, ESD diodes must be used between the amplifier and electrical over stress.

All GPIOs have a diode clamping circuit to protect against overvoltage and ESD.

The following figure shows the equivalent circuit of diode clamping.

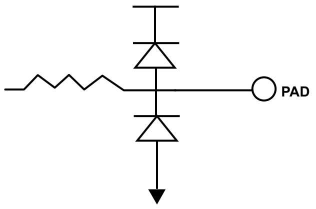


Figure 5-3: Diode Clamping Circuit

ADuCM302x GPIO Multiplexing

The following tables show the pin functions that are multiplexed on the GPIO pins of the package.

Table 5-1: Signal Muxing - Port 0

Signal Name	Multiplexed Function 0	Multiplexed Function 1	Multiplexed Function 2	Multiplexed Function 3
P0_00	GPIO00	SPI0_CLK	SPT0_BCLK	
P0_01	GPIO01	SPI0_MOSI	SPT0_BFS	
P0_02	GPIO02	SPI0_MISO	SPT0_BD0	
P0_03	GPIO03	SPI0_CS0	SPT0_BCNV	SPI2_RDY
P0_04	GPIO04	I2C0_SCL		
P0_05	GPIO05	I2C0_SDA		
P0_06	SWD0_CLK	GPIO06		
P0_07	SWD0_DATA	GPIO07		
P0_08	GPIO08	BEEP0_TONE		
P0_09	GPIO09	BEEP0_TONE	SPI2_CS1	
P0_10	GPIO10	UART0_TX		
P0_11	GPIO11	UART0_RX		
P0_12	GPIO12	SPT0_AD0		
P0_13	GPIO13			
P0_14	GPIO14	TMR0_OUT	SPI1_RDY	
P0_15	GPIO15			

Table 5-2: Signal Muxing - Port 1

Signal Name	Multiplexed Function 0	Multiplexed Function 1	Multiplexed Function 2	Multiplexed Function 3
P1_00	GPIO16			
P1_01	SYS_BMODE0	GPIO17		
P1_02	GPIO18	SPI2_CLK		

Table 5-2: Signal Muxing - Port 1 (Continued)

Signal Name	Multiplexed Function 0	Multiplexed Function 1	Multiplexed Function 2	Multiplexed Function 3
P1_03	GPIO19	SPI2_MOSI		
P1_04	GPIO20	SPI2_MISO		
P1_05	GPIO21	SPI2_CS0		
P1_06	GPIO22	SPI1_CLK		
P1_07	GPIO23	SPI1_MOSI		
P1_08	GPIO24	SPI1_MISO		
P1_09	GPIO25	SPI1_CS0		
P1_10	GPIO26	SPI0_CS1	SYS_CLKIN	SPI1_CS3
P1_11	GPIO27		TMR1_OUT	
P1_12	GPIO28			
P1_13	GPIO29			
P1_14	GPIO30		SPI0_RDY	
P1_15	GPIO31	SPT0_ACLK		

Table 5-3: Signal Muxing - Port 2

Signal Name	Multiplexed Function 0	Multiplexed Function 1	Multiplexed Function 2	Multiplexed Function 3
P2_00	GPIO32	SPT0_AFS		
P2_01	GPIO33		TMR2_OUT	
P2_02	GPIO34	SPT0_ACNV	SPI1_CS2	
P2_03	GPIO35	ADC_VIN0		
P2_04	GPIO36	ADC_VIN1		
P2_05	GPIO37	ADC_VIN2		
P2_06	GPIO38	ADC_VIN3		
P2_07	GPIO39	ADC_VIN4	SPI2_CS3	
P2_08	GPIO40	ADC_VIN5	SPI0_CS2	
P2_09	GPIO41	ADC_VIN6	SPI0_CS3	
P2_10	GPIO42	ADC_VIN7	SPI2_CS2	
P2_11	GPIO43	SPI1_CS1	SYS_CLKOUT	RTC1_SS1

GPIO Operating Modes

IO Pull-Up or Pull-Down Enable

All GPIO pins can be grouped into two categories: GPIO pins with a pull-up resistor and GPIO pins with a pull-down resistor. Each GPIO port has a corresponding `GPIO_PE` register. Using the `GPIO_PE` registers, it is possible to enable/disable pull-up/pull-down registers on the pins when they are configured as inputs.

IO Data In

When configured as an input using the `GPIO_IEN` register, the GPIO input levels are available in the `GPIO_IN` register.

IO Data Out

When the GPIOs are configured as outputs using the `GPIO_OEN` register, the values in the `GPIO_OUT` register are reflected on the GPIOs.

Bit Set

Each GPIO port has a corresponding bit set register, `GPIO_SET`. Use the bit set register to set one or more GPIO data outs without affecting others within the port. Only the GPIO corresponding with the write data bit equal to one is set, the remaining GPIOs are unaffected.

Bit Clear

Each GPIO port has a corresponding bit clear register, `GPIO_CLR`. Use the bit clear register to clear one or more GPIO data outs without affecting others within the port. Only the GPIO corresponding with the write data bit equal to one is cleared, the remaining GPIOs are unaffected. This is a write only register. To read, the `GPIO_OUT` register is read back. This feature speeds up the software access to the GPIO registers as they share the same address offset (the offset of `GPIO_OUT` register).

Bit Toggle

Each GPIO port has a corresponding bit toggle register, `GPIO_TGL`. Using the bit toggle register, it is possible to invert one or more GPIO data outs without affecting others within the port. Only the GPIO corresponding with the write data bit equal to one is toggled, the remaining GPIOs are unaffected. This is a write only register. To read, the `GPIO_OUT` register is read back. This feature speeds up the software access to the GPIO registers as they share the same address offset (the offset of `GPIO_OUT` register).

IO Data Output Enable

Each GPIO port has a data output enable register, `GPIO_OEN`, by which the data output path is enabled. When the data output enable register bits are set, the values in `GPIO_OUT` are reflected on the corresponding GPIO pins.

Interrupts

Each GPIO pin can be associated with an interrupt. Interrupts can be independently enabled for each GPIO pin and are always edge detect; only one interrupt will be generated with each GPIO pin transition. The polarity of the detected edge can be positive (low-to-high) or negative (high-to-low). Each GPIO interrupt event can be mapped to one of two interrupts (GPIO INTA or GPIO INTB). This allows the system more flexibility in terms of how GPIO interrupts are grouped for servicing, and how interrupt priorities are set. The interrupt status of each GPIO pin can be determined and cleared by accessing the associated status registers.

Interrupt Polarity

The polarity of the interrupt determines if the interrupt is accepted on the rising or the falling edge. Each GPIO port has a corresponding interrupt register (`GPIO_POL`) by which the interrupt polarity of each pin is configured. When set to 0, an interrupt event will be latched on a high-to-low transition on the corresponding pin. When set to 1, an interrupt event will be latched on a low-to-high transition on the corresponding pin.

Interrupt A Enable

Each GPIO port has an Interrupt Enable A register (`GPIO_IENA`) that is enabled or masked for each pin in the port. These register bits determine if a latched edge event should be allowed to interrupt the core (Interrupt A) or be masked. In either case, the occurrence of the event will be captured in the corresponding bit of the `GPIO_INT` status register. When set to 0, Interrupt A is not enabled (masked). No interrupts to the core will be generated by this GPIO pin. When set to 1, Interrupt A is enabled. On a valid detected edge, an interrupt source to the core will be generated.

Interrupt B Enable

Each GPIO port has a corresponding Interrupt Enable B (`GPIO_IENB`) register that is enabled or masked for each pin in the port. These register bits determine if a latched edge event should be allowed to interrupt the core (Interrupt B) or be masked. In either case, the occurrence of the event will be captured in the corresponding bit of the `GPIO_INT` status register. When set to 0, Interrupt B is not enabled (masked). No interrupts to the core will be generated by this GPIO pin. When set to 1, Interrupt B is enabled. On a valid detected edge, an interrupt source to the core will be generated.

Interrupt Status

Each GPIO port has an interrupt status register (`GPIO_INT`) that captures the interrupts occurring on its pins. These register bits indicate that the appropriately configured rising or falling edge has been detected on the corresponding GPIO pin.

Once an event is detected, `GPIO_INT` will remain set until cleared; this is true even if the GPIO pin transitions back to a nonactive state. Out of reset, pull ups combined with falling edge detect can result in the `GPIO_INT` status being cleared; however, this may not be the case based on pin activity. It is therefore recommended that the status of `GPIO_INT` be checked before enabling interrupts (`GPIO_IENA` and `GPIO_IENB`) initially as well as anytime GPIO pins are configured.

Interrupt bits are cleared by writing a 1 to the appropriate bit location. Writing a 0 has no effect. If interrupts are enabled to the core (`GPIO_IENA`, `GPIO_IENB`), an interrupt (`GPIO_INT`) value of 1 will result in an interrupt to the core. This bit should be cleared during servicing of the interrupt. When read as 0, a rising or falling edge was not detected on the corresponding GPIO pin because this bit was last cleared. When read as 1, a rising or falling edge (`GPIO_POL` selectable) was detected on the corresponding GPIO pin. This bit can be software cleared by writing a 1 to this bit location. This bit can only be subsequently set again after the pin returns to its nonasserted state and then returns to an asserted state.

GPIO Programming Model

Programming sequence for output functionality:

1. Program the `GPIO_OEN` register for the appropriate GPIO pins.
2. The data on the selected pin is driven high or low by writing to the `GPIO_SET`/`GPIO_CLR`/`GPIO_TGL` registers.

Programming Sequence for input functionality:

1. Program the `GPIO_IEN` register for the appropriate port pins.
2. The data latched by the input port pin is registered in the `GPIO_IN` register. Interrupts can be enabled by writing to the `GPIO_IENA` or `GPIO_IENB` register.

ADuCM302x GPIO Register Descriptions

GPIO contains the following registers.

Table 5-4: ADuCM302x GPIO Register List

Name	Description
GPIO_CFG	Port Configuration
GPIO_CLR	Port Data Out Clear
GPIO_DS	Port Drive Strength Select
GPIO_IEN	Port Input Path Enable
GPIO_IENA	Port Interrupt A Enable
GPIO_IENB	Port Interrupt B Enable
GPIO_IN	Port Registered Data Input
GPIO_INT	Port Interrupt Status
GPIO_OEN	Port Output Enable
GPIO_OUT	Port Data Output
GPIO_PE	Port Output Pull-up/Pull-down Enable

Table 5-4: ADuCM302x GPIO Register List (Continued)

Name	Description
GPIO_POL	Port Interrupt Polarity
GPIO_SET	Port Data Out Set
GPIO_TGL	Port Pin Toggle

Port Configuration

The GPIO_CFG register is reserved for top-level pin muxing for the GPIO block.

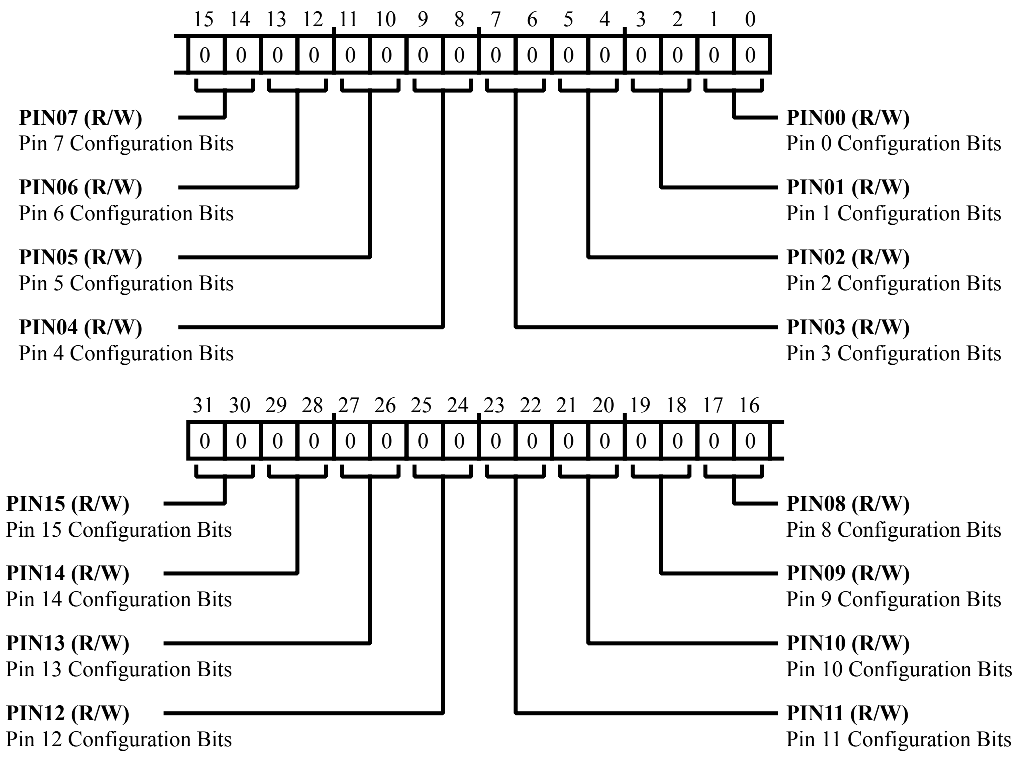


Figure 5-4: GPIO_CFG Register Diagram

Table 5-5: GPIO_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:30 (R/W)	PIN15	Pin 15 Configuration Bits.
29:28 (R/W)	PIN14	Pin 14 Configuration Bits.
27:26 (R/W)	PIN13	Pin 13 Configuration Bits.
25:24 (R/W)	PIN12	Pin 12 Configuration Bits.
23:22 (R/W)	PIN11	Pin 11 Configuration Bits.

Table 5-5: GPIO_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
21:20 (R/W)	PIN10	Pin 10 Configuration Bits.
19:18 (R/W)	PIN09	Pin 9 Configuration Bits.
17:16 (R/W)	PIN08	Pin 8 Configuration Bits.
15:14 (R/W)	PIN07	Pin 7 Configuration Bits.
13:12 (R/W)	PIN06	Pin 6 Configuration Bits.
11:10 (R/W)	PIN05	Pin 5 Configuration Bits.
9:8 (R/W)	PIN04	Pin 4 Configuration Bits.
7:6 (R/W)	PIN03	Pin 3 Configuration Bits.
5:4 (R/W)	PIN02	Pin 2 Configuration Bits.
3:2 (R/W)	PIN01	Pin 1 Configuration Bits.
1:0 (R/W)	PIN00	Pin 0 Configuration Bits.

Port Data Out Clear

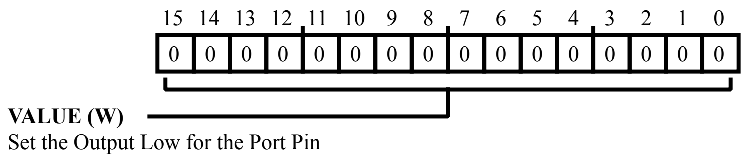


Figure 5-5: GPIO_CLR Register Diagram

Table 5-6: GPIO_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (RX/W)	VALUE	Set the Output Low for the Port Pin. Each bit is set to drive the corresponding GPIO pin low. Clearing this bit has no effect.

Port Drive Strength Select

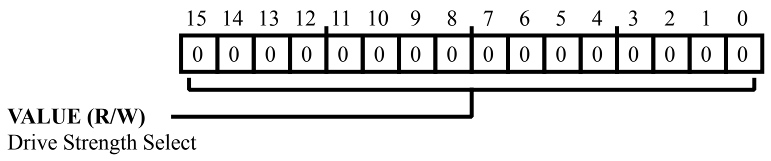


Figure 5-6: GPIO_DS Register Diagram

Table 5-7: GPIO_DS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Drive Strength Select. Each bit is configured to set the Drive strength of the corresponding GPIO Pin.

Port Input Path Enable

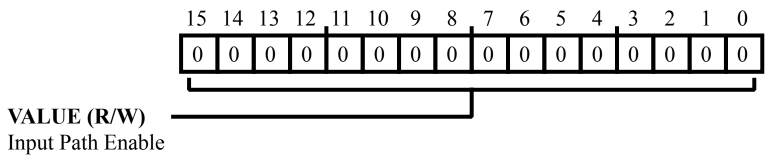


Figure 5-7: GPIO_IEN Register Diagram

Table 5-8: GPIO_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Input Path Enable. Each bit is set to enable the input path and cleared to disable the input path for the GPIO pin.

Port Interrupt A Enable

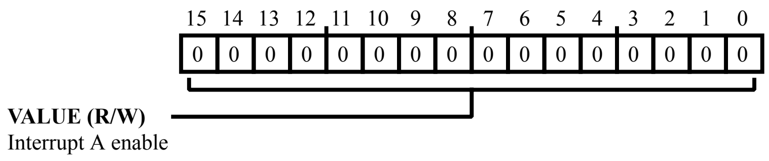


Figure 5-8: GPIO_IENA Register Diagram

Table 5-9: GPIO_IENA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Interrupt A enable. Determines if a latched edge event should be allowed to interrupt the core (GPIO IntA) or be masked. In either case the occurrence of the event will be captured in the GPIO_INT status register. When cleared, Interrupt A is not enabled (masked). When set Interrupt A is enabled.

Port Interrupt B Enable

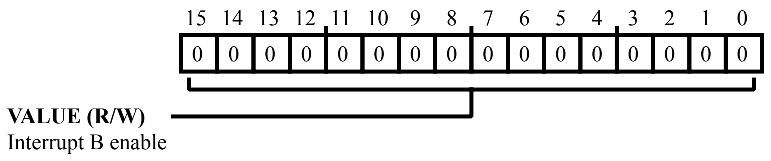


Figure 5-9: GPIO_IENB Register Diagram

Table 5-10: GPIO_IENB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Interrupt B enable. Determines if a latched edge event must be allowed to interrupt the core (GPIO IntB) or be masked. In either case, the occurrence of the event is captured in the `GPIO_INT` status register. When set to 0, Interrupt B is not enabled (masked). When set to 1, Interrupt B is enabled.

Port Registered Data Input

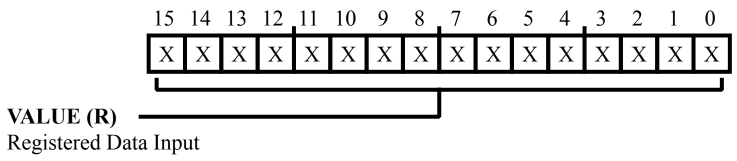


Figure 5-10: GPIO_IN Register Diagram

Table 5-11: GPIO_IN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Registered Data Input. Each bit reflects the state of the GPIO pin if the corresponding input buffer is enabled. If the pin input buffer is disabled, the value seen is zero.

Port Interrupt Status

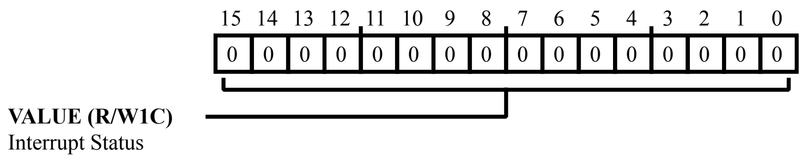


Figure 5-11: GPIO_INT Register Diagram

Table 5-12: GPIO_INT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	VALUE	Interrupt Status. Indicates that the appropriately configured rising or falling edge has been detected on the corresponding GPIO pin. Once an event is detected, `GPIO_INT.VALUE` bit remains set until cleared. This is true even if the GPIO pin transitions back to a non active state. `GPIO_INT.VALUE` bits are cleared by writing 1 to the appropriate bit location. Writing 0 has no effect.

Port Output Enable

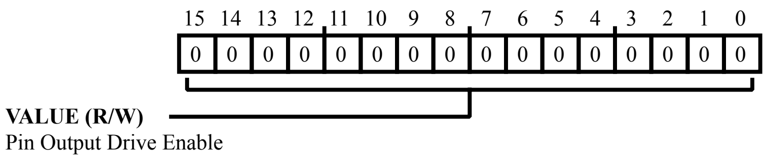


Figure 5-12: GPIO_OEN Register Diagram

Table 5-13: GPIO_OEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Pin Output Drive Enable. Each bit is set to enable the output for that particular pin. It is cleared to disable the output for each pin.

Port Data Output

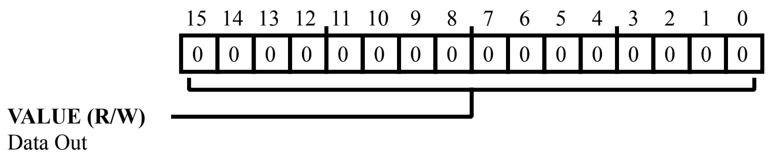


Figure 5-13: GPIO_OUT Register Diagram

Table 5-14: GPIO_OUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Data Out. Set by user code to drive the corresponding GPIO high. Cleared by user to drive the corresponding GPIO low.

Port Output Pull-up/Pull-down Enable

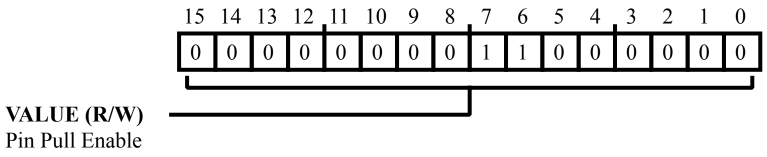


Figure 5-14: GPIO_PE Register Diagram

Table 5-15: GPIO_PE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Pin Pull Enable. Each bit is set to enable the pull-up/pull-down for that particular pin. It is cleared to disable the pull-up/pull-down for each pin. The values shown in the Register Diagram are for GPIO Port 0.

Port Interrupt Polarity

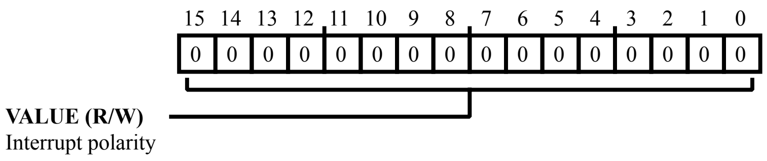


Figure 5-15: GPIO_POL Register Diagram

Table 5-16: GPIO_POL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Interrupt polarity. Determines if the interrupts are generated on the rising or falling edge of the corresponding GPIO pin. When cleared, an interrupt event is latched on a high-to-low transition. When set, an interrupt event is latched on a low-to-high transition.

Port Data Out Set

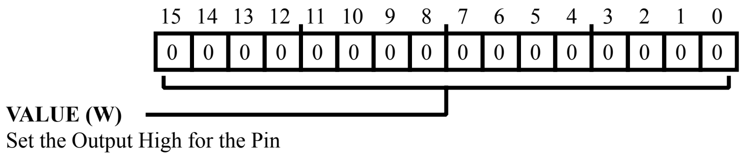


Figure 5-16: GPIO_SET Register Diagram

Table 5-17: GPIO_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (RX/W)	VALUE	Set the Output High for the Pin. Set by user code to drive the corresponding GPIO high. Clearing this bit has no effect.

Port Pin Toggle

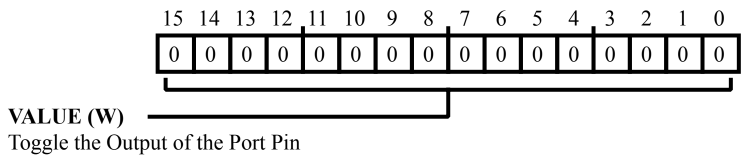


Figure 5-17: GPIO_TGL Register Diagram

Table 5-18: GPIO_TGL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (RX/W)	VALUE	Toggle the Output of the Port Pin. Each bit is set to invert the corresponding GPIO pin. Clearing this bit has not effect.

6 System Clocks

The ADuCM302x MCU is integrated with two on-chip oscillators and the circuitry for two external crystals.

System Clock Features

The System Clock features include the following:

- LFOSC is a 32 kHz internal oscillator.
- HFOSC is a 26 MHz internal oscillator.
- LFXTAL is a 32 kHz external crystal oscillator.
- HFXTAL is a 16 MHz or 26 MHz external crystal oscillator.
- External clock input through SYS_CLKIN.
- One on-chip phase locked loop (PLL) is available: the system PLL (SPLL). PLL can use HFOSC or HFXTAL as an input clock.
- The high frequency oscillators (HFOSC and HFXTAL), along with the output of the SPLL and SYS_CLKIN, can be used to generate the root clock.
- The 32 kHz clock (LF_CLK) is generated from either LFXTAL or LFOSC to drive certain blocks.
- The general purpose timers have their own multiplexers to select the clock source.
- The RTC1 works from LFXTAL or LFOSC in active, Flexi, and hibernate modes. The RTC0 always works from LFXTAL in all power modes.
- Watchdog timer always works on LFOSC oscillator. It cannot be run using LFXTAL to avoid reliability issues. It does not work in hibernate or shutdown modes.
- The root clock is divided into several internal clocks.
- RCLK clocks the reference timer (counter) in flash controller. This is used to time flash erase, write operations. This is generated by ½ divider connected to HFOSC (26 MHz). So, the default values of flash timer registers correspond to 13 MHz clock.

If `CLKG_CLK_CTL0.RCLKMUX` = 2'b11 (HFXTAL = 16 MHz), the ½ DIV (divider) is automatically bypassed and 16 MHz HFXTAL is directly connected to the RCLK. Therefore, if HFXTAL 16 MHz is used, flash timer registers need to be programmed to 16 MHz before any flash erase/write operations.

- HP_BUCK_CLK clocks the HP Buck module. When HP Buck is enabled, this clock is always 200 kHz.

System Clock Functional Description

This section provides information on the clocks and the clock gates required for power management.

System Clock Block Diagram

The clocking diagram is shown below.

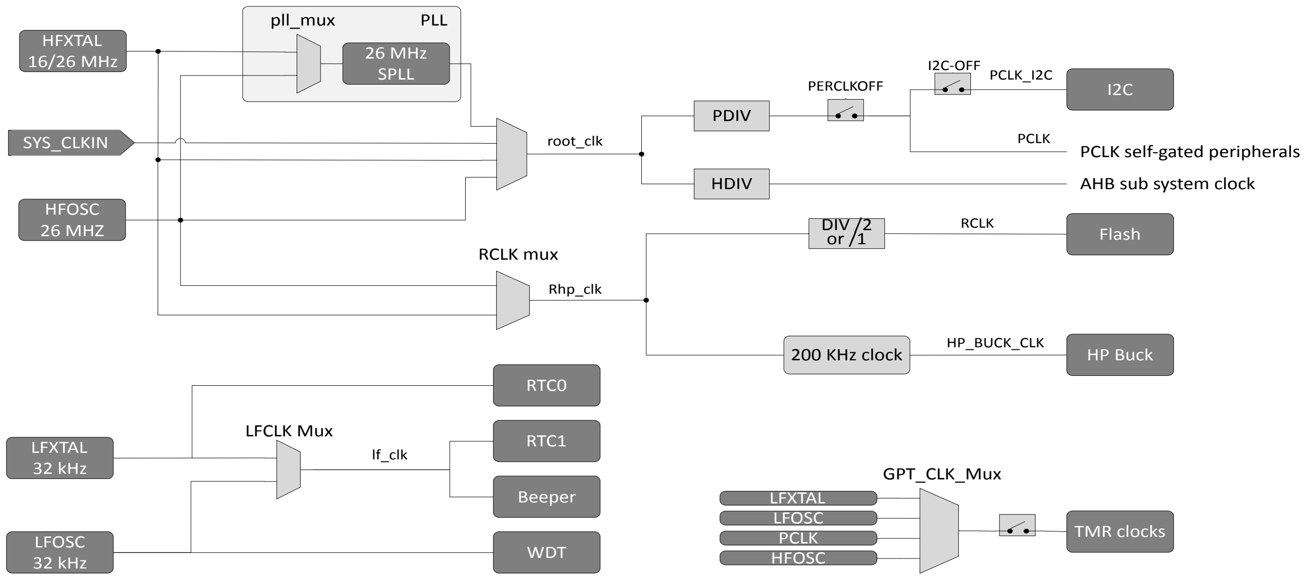


Figure 6-1: Clocking Diagram

Clock Muxes

As shown in the *Clocking Diagram*, there are five clock source multiplexers:

- Root clock selection Mux (Root clock Mux)
- Low frequency clock Mux (LFCLK Mux)
- SPLL input Mux (SPLL Mux)
- RCLK Mux
- GPT_CLK Mux

The multiplexer for general purpose timers are controlled through registers within the Timer block.

NOTE: Ensure that the desired clock input is available and stable for the clock selection made for the clock multiplexer. Otherwise, situations where the system can be locked out of a stable clock can arise.

Table 6-1: Clock Muxes

Clocks	Registers	Selection
Root Clock Mux	`CLKG_CLK_CTL0.CLKMUX`	00: HFOSC 01: HFXTAL 10: SPLL 11: External clock through SYS_CLKIN
SPLL Mux	`CLKG_CLK_CTL0.SPLLIPSEL`	0: HFOSC 1: HFXTAL
LFCLK Mux	`CLKG_OSC_CTL.LFCLKMUX`	0: LFOSC 1: LFXTAL
RCLK Mux	`CLKG_CLK_CTL0.RCLKMUX`	00: HFOSC 01: Reserved 10: HFXTAL 26 MHz 11: HFXTAL 16 MHz
GPT_CLK Mux	`TMR_CTL.CLK`	Controlled by Timer 0/Timer 1/Timer 2 00: PCLK 01: HFOSC 10: LFOSC 11: LFXTAL

Clock Dividers

Three programmable clock dividers are available to generate the clocks in the system. A clock divider integer divides the input clock down to a new clock. The division range is from 1 to 32. Division selection can be made on the fly. The output remains glitch free and stretches the high time, not creating a high time shorter than pre or post value of the clock period.

Two clock dividers use the root clock as an input and generate the core and peripheral synchronized clocks. The clock dividers are cascaded in such a way that each stage initially releases its divided clock in a sequence. The last stage informs all other stages that its divided clock is ready to be output. The effect of this cascading is that divided clocks are released synchronously when new divider values are programmed. Initial edges of each clock are mutually aligned.

The ***Clock Dividers*** table summarizes the inputs and outputs of each clock divider along with the register bits to program them.

Table 6-2: Clock Dividers

Divider	Input Clock	Output Clocks	Register, Bits
HDIV	Root clock	HCLK_CORE, HCLK_BUS	`CLKG_CLK_CTL1.HCLKDIVCNT`
PDIV	Root clock	all peripheral clocks (PCLK)	`CLKG_CLK_CTL1.PCLKDIVCNT`

Only certain divider ratios are legal between PCLK and HCLK. The frequency of PCLK must be less than or equal to the frequency of HCLK, and the ratio of the dividers must be an integer.

In general, clock division can be changed on the fly during normal operation.

Clock Gating

In the case of certain clocks, clocks can be individually gated depending on the power mode or register settings. For more information about clock gating and power modes, refer to Power Management (PMG).

The clock gates of the peripheral clocks are user controllable in certain power modes. The `CLKG_CLK_CTL5` register can be programmed to turn off certain clocks, depending on user application. To disable the clock, set the respective bits in the `CLKG_CLK_CTL5` register to 1.

PLL Settings

The ***PLL Diagram*** shows the abstract PLL structure for SPLL. It has the multiplier coefficient N and divider coefficient M to decide the output clock ratio of N/M. There is an optional DIV2 built in the PLL to either divide down the PLL output clock by 2 or directly output it. This is also an optional MUL2 built-in to PLL to multiply the clock out of PLL by 2 to increase the range of the PLL.

The PLL must always switch away from ROOT_CLK when changing any coefficients or clock sources.

When the reference clock for the phase frequency detector (PFD) has a 2 MHz input, the PLL has its best phase margin. Therefore, it is recommended that for a 26 MHz crystal clock input, configure M as 13, and for a 16 MHz crystal clock input, configure M as 8.

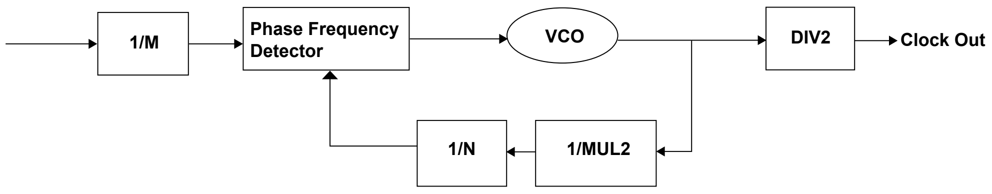


Figure 6-2: PLL Diagram

The ***PLL Settings*** table shows the recommend PLL settings for a variety of input and output clocks. The PLL settings can be programmed using the `CLKG_CLK_CTL3` register.

To enable the SPLL, the `CLKG_CLK_CTL3.SPLLEN` bit must be set.

Table 6-3: PLL Settings

PLL	Input Clock (MHz)	M	MUL2	N	DIV2	Output Clock (MHz)
SPLL	26	13	1	26	2	26
SPLL	16	8	1	26	2	26

The recommended VCO output range is 32 MHz to 60 MHz and the PLL output range is 16 MHz to 60 MHz. The minimum value of N is 8 (if MUL2=2) and maximum is 31. The recommended output frequency from 1/M divider is 2 MHz and the minimum value of M = 2.

PLL Output Frequency = Clock in* ((MUL2*NSEL)/ (DIV2*MSEL))

Where,

MUL2 = `CLKG_CLK_CTL3.SPLLMUL2`+1

DIV2 = `CLKG_CLK_CTL3.SPLLDIV2` + 1

N = `CLKG_CLK_CTL3.SPLLNSEL`

M = `CLKG_CLK_CTL3.SPLLMSEL`

PLL Interrupts

The PLL can interrupt the core when it locks or when it loses its lock. To enable the SPLL interrupts, the `CLKG_CLK_CTL3.SPLLIE` bit must be set. The `CLKG_CLK_STAT0.SPLLUNLK` bit indicates that the SPLL unlock event has happened.

The `CLKG_CLK_STAT0.SPLLLK` bit indicates that a SPLL lock event has happened. Both the bits are used to interrupt the core when PLL interrupts are enabled. Both bits are sticky and must write a 1 to be cleared. These bits are different from the `CLKG_CLK_STAT0.SPLL` bit, which mirrors the value of the SPLL lock signal (1: Locked, 0: Unlocked).

PLL Programming Sequence

The following sequence shows how to configure a 26 MHz input clock from HFXTAL to a 26 MHz SPLL output, and use the PLL output as a root clock. The HCLK is set to 26 MHz and PCLK to 6.5 MHz.

1. Enable PLL interrupts by setting `CLKG_CLK_CTL3.SPLLIE` to 0x1.
2. Set HFXTAL as input to PLL by setting `CLKG_CLK_CTL0.SPLLIPSEL` to 0x1.
3. Enable HFXTAL by setting `CLKG_OSC_CTL.HFXTALEN` to 0x1.
4. Set the clock dividers consistent with the intended system clock rates by setting `CLKG_CLK_CTL1.PCLKDIVCNT` to 0x04 and `CLKG_CLK_CTL1.HCLKDIVCNT` to 0x01.
5. Enable the PLL and configure the PLL M and N values by setting `CLKG_CLK_CTL3.SPLLEN` to 0x1, `CLKG_CLK_CTL3.SPLLMSEL` to 0xD, and `CLKG_CLK_CTL3.SPLLNSEL` to 0x1A.

6. Wait for the PLL interrupt indicating that the PLL has locked. Optionally, also check that the crystal is stable at this stage, even though the PLL should not lock if the crystal is not stable.
7. Clear the PLL interrupt and select PLL as the system clock source by setting `CLKG_CLK_STAT0.SPLLLK` to 0x1 and `CLKG_CLK_CTL0.CLKMUX` to 0x2.

The following sequence shows how to configure a 26 MHz input clock from HFOSC to a 16 MHz SPLL output, and use the PLL output as a root clock. HCLKDIV and PCLKDIV are set to 0x1.

1. Enable PLL interrupts by setting `CLKG_CLK_CTL3.SPLLIE` to 0x1.
2. Set HFOSC as input to PLL by setting `CLKG_CLK_CTL0.SPLLIPSEL` to 0x0.
3. Disable the PLL by setting `CLKG_CLK_CTL3.SPLLEN` to 0x0.
4. Program appropriate MSEL and NSEL to get PLL clock out as 16 MHz by setting `CLKG_CLK_CTL3.SPLLMSEL` to 0xD and `CLKG_CLK_CTL3.SPLLNSEL` to 0x10.
5. Enable the PLL by setting `CLKG_CLK_CTL3.SPLLEN` to 0x1.
6. Wait for the PLL interrupt indicating that the PLL has locked. Optionally, check if the crystal is stable at this stage.
7. Clear the PLL interrupt and select PLL as the system clock source by setting `CLKG_CLK_STAT0.SPLLLK` to 0x1 and `CLKG_CLK_CTL0.CLKMUX` to 0x2.

Crystal Programming

The crystals are disabled by default and can be programmed using the `CLKG_OSC_CTL` register. The crystals can be enabled by setting the `CLKG_OSC_CTL.HFXTALEN` or `CLKG_OSC_CTL.LFXTALEN` bits. The stable signal status bits are also mirrored in this register.

NOTE: The `CLKG_OSC_CTL.HFOSCEN` bit must be set before issuing a `SYSRESETREQ` and allowing the Cortex-M3 to assert a reset request signal to the systems reset generator. This ensures that all system components are reset properly. This is independent of the Root Clock Mux and SPLL Clock Mux settings.

Interrupts

Each crystal can interrupt the core when its output clock becomes stable. The interrupts are enabled by setting the `CLKG_CLK_CTL0.HFXTALIE` and `CLKG_CLK_CTL0.LFXTALIE` bits.

The `CLKG_CLK_STAT0.HFXTAL` and `CLKG_CLK_STAT0.LFXTAL` bits contain the current state of the stable signals of the crystals. The `CLKG_CLK_STAT0.HFXTALOK` or `CLKG_CLK_STAT0.LFXTALOK` bits are set when an event is detected on the stable signals of the crystals.

The `CLKG_CLK_STAT0.HFXTALOK`, `CLKG_CLK_STAT0.HFXTALNOK`, `CLKG_CLK_STAT0.LFXTALOK`, and `CLKG_CLK_STAT0.LFXTALNOK` bits are sticky and must be cleared by writing to 1.

NOTE: CLKG_CLK_STAT0.HFXTALNOK and CLKG_CLK_STAT0.LFXTALNOK bits are not continuous crystal monitors and are only set as a confirmation that the corresponding crystal has been properly disabled.

PLL Clock Protection

In the event the clock source to the PLL is lost, the PLL will maintain operation at a reduced VCO output frequency of approximately 20 MHz, for approximately 3 to 5 ms, this clock will be active/running until PLL is disabled. This behavior allows PLL interrupt sources such as CLKG_CLK_STAT0.SPLLUNLK to be serviced and enables the appropriate action to be taken by the core. This feature protects the core from an indefinite stall due to broken or shorted leads of the crystal circuit.

Oscillator Programming

Both the internal oscillators are enabled by default.

Before issuing a SYSRESETREQ and allowing the Cortex-M3 to assert a reset request signal to the systems reset generator, the CLKG_OSC_CTL.HFOSCEN must be set. This ensures that all system components are reset properly. This is independent of Root Clock Mux and SPLL Clock Mux settings.

HFOSC is enabled by using the CLKG_OSC_CTL.HFOSCEN bit.

NOTE: Before stopping the HFOSC internal oscillator, the HFXTAL should be running the system. Otherwise, the part locks because its clock has been stopped by the user without possibility of recovery.

Peripherals driven with a 32 kHz clock must be switched over to the LFXTAL external crystal oscillator only after ensuring that LFXTAL is stable and running. LFOSC internal oscillator cannot be disabled.

Oscillators

There are four types of oscillators:

- HF Oscillator: The 26 MHz high frequency oscillator is enabled by the CLKG_OSC_CTL.HFOSCEN bit. The oscillator accuracy is ±3%.
- LF Oscillator: The 32.768 kHz low frequency oscillator is enabled always, it cannot be disabled, and it is disabled automatically in the shutdown mode. The oscillator accuracy is ±5%.
- HFXTAL Oscillator: The high frequency oscillator is enabled by the CLKG_OSC_CTL.HFXTALEN bit. Lock time is the time the XTAL takes to give stable clock out after it is enabled. The locking of the oscillator to the correct frequency is signified by the setting of CLKG_CLK_STAT0.HFXTALOK status bit.
- LFXTAL Oscillator: The LFXTAL oscillator can be used as a clock source for the RTC. It is used to keep the time of the system. It provides a 32.768 kHz output clock. This is enabled by setting the CLKG_OSC_CTL.LFXTALEN bit.

  Once the oscillator is enabled, it remains always on, even in hibernate and shutdown modes.

System Clock Interrupts and Exceptions

Refer to Events (Interrupts and Exceptions), for more information.

Setting the System Clocks

Set System Clock to PLL Input Source

The following timing diagrams describe the sequence of events to change system clock from internal oscillator to PLL input source based. XTAL refers to HFXTAL (26 MHz or 16 MHz crystal oscillator). Timers have following expiry periods.

HFXTAL lock timer expiry: 20 ms

LFXTAL lock timer expiry: 1.5 s

If the oscillators are not locked beyond the expiry period, it indicates that there are issues in the crystal.

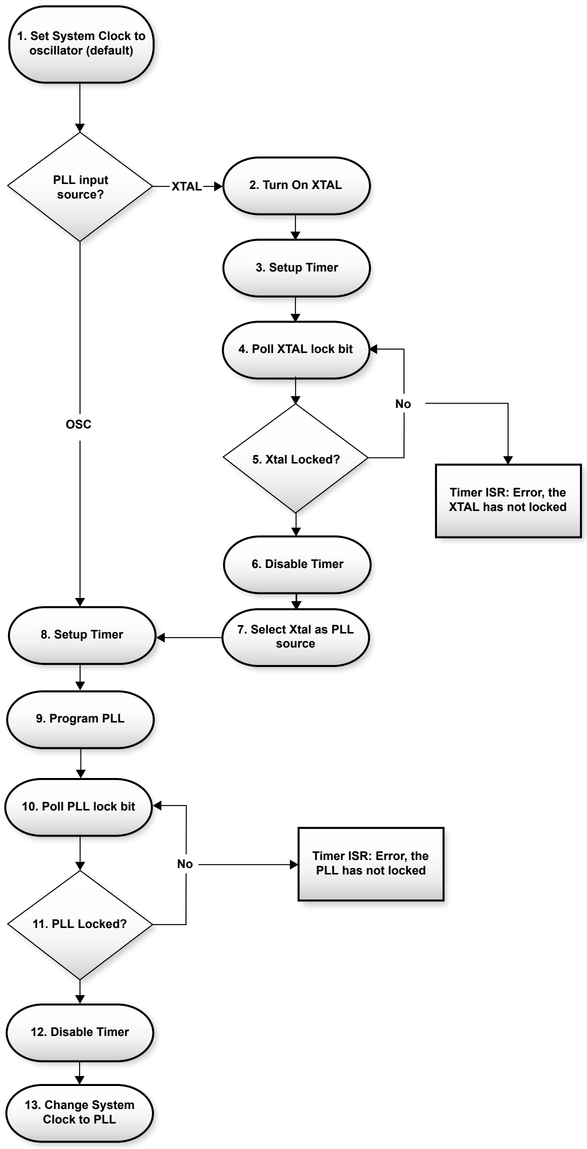


Figure 6-3: Change System Clock to PLL (POLL)

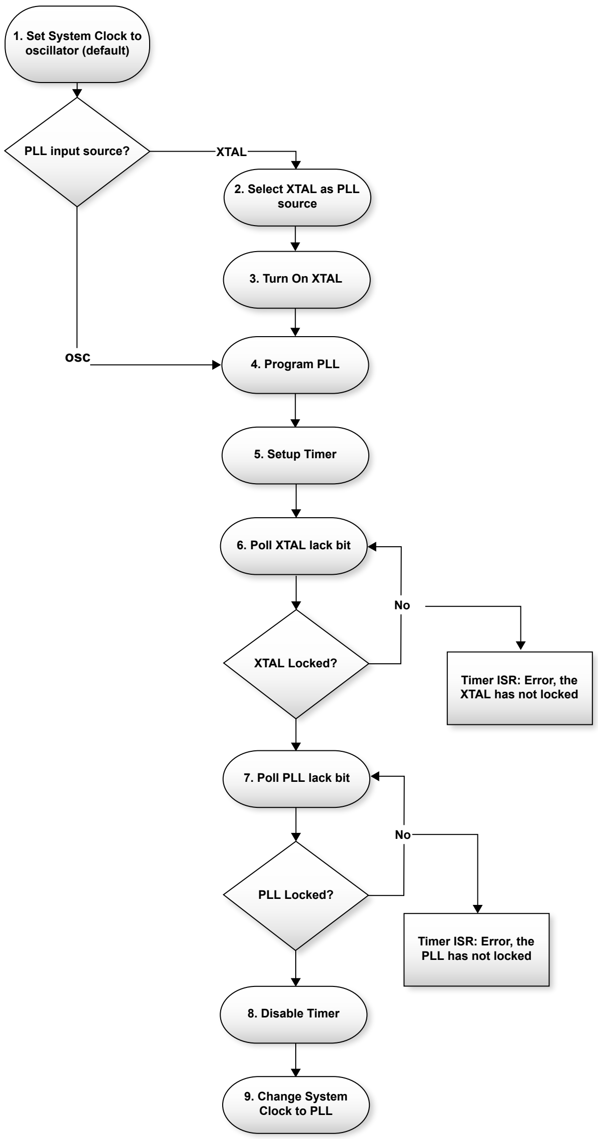

1. Set System Clock to oscillator (default)
PLL input source?
XTAL
2. Select XTAL as PLL source
3. Turn On XTAL
osc
4. Program PLL
5. Setup Timer
6. Poll XTAL lack bit
No
XTAL Locked?
Timer ISR: Error, the XTAL has not locked
7. Poll PLL lack bit
No
PLL Locked?
Timer ISR: Error, the PLL has not locked
8. Disable Timer
9. Change System Clock to PLL

Figure 6-4: Change System Clock to PLL (POLL Alternative)

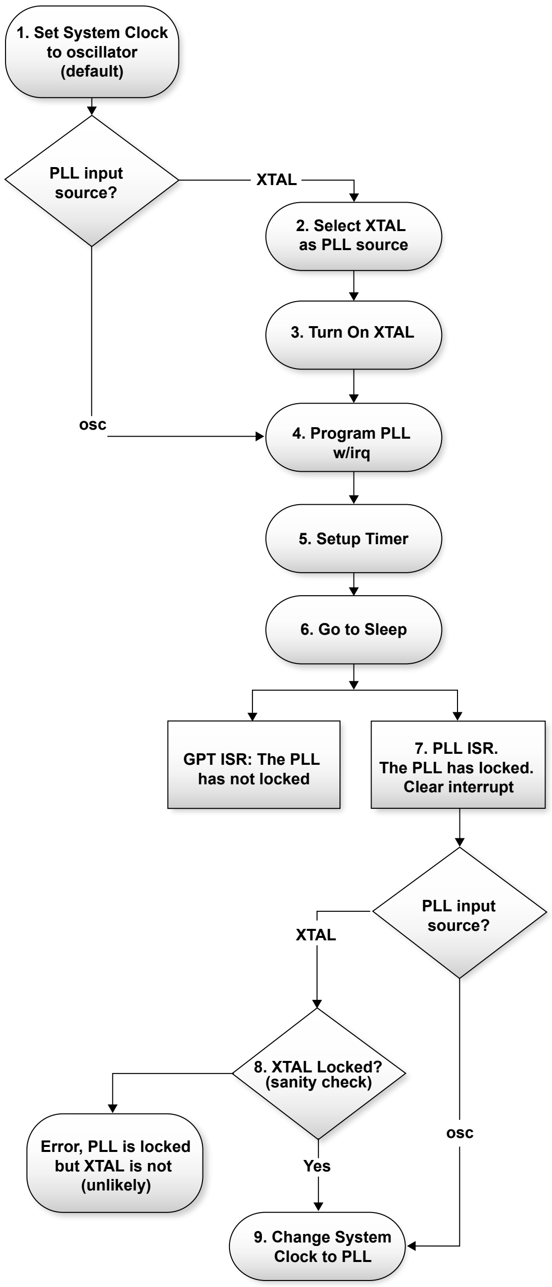

1. Set System Clock to oscillator (default)
PLL input source?
XTAL
2. Select XTAL as PLL source
3. Turn On XTAL
osc
4. Program PLL w/irq
5. Setup Timer
6. Go to Sleep
GPT ISR: The PLL has not locked
7. PLL ISR. The PLL has locked. Clear interrupt
PLL input source?
XTAL
8. XTAL Locked? (sanity check)
Error, PLL is locked but XTAL is not (unlikely)
Yes
osc
9. Change System Clock to PLL

Figure 6-5: Change System Clock to PLL (IRQ Method)

Set System Clock to XTAL

The following figures show the sequence of events to change system clock from internal oscillator based to XTAL based source.

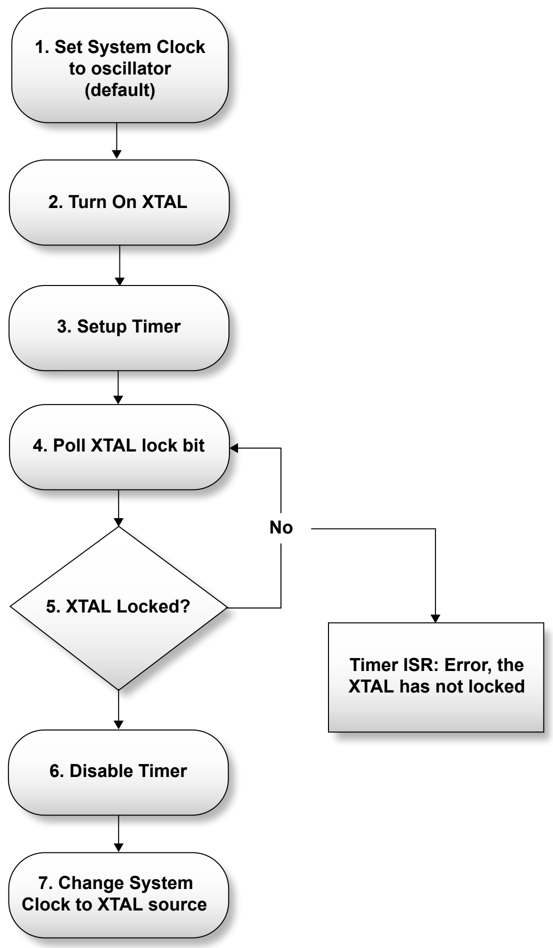

1. Set System Clock to oscillator (default)
2. Turn On XTAL
3. Setup Timer
4. Poll XTAL lock bit
5. XTAL Locked?
No
Timer ISR: Error, the XTAL has not locked
6. Disable Timer
7. Change System Clock to XTAL source

Figure 6-6: Change System Clock To XTAL (Poll Method)

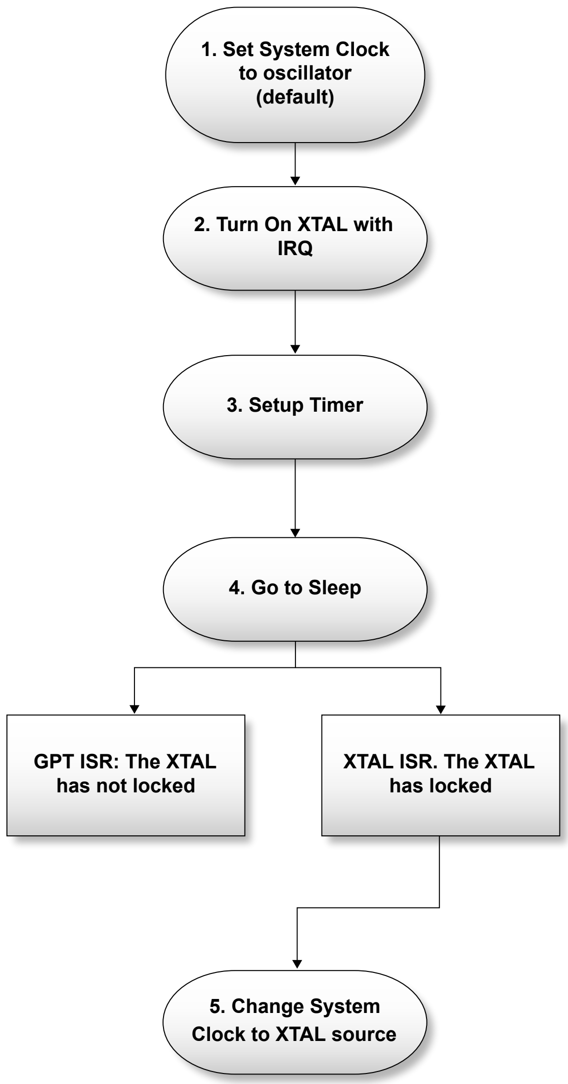

1. Set System Clock to oscillator (default)
2. Turn On XTAL with IRQ
3. Setup Timer
4. Go to Sleep
GPT ISR: The XTAL has not locked
XTAL ISR. The XTAL has locked
5. Change System Clock to XTAL source

Figure 6-7: Change System Clock to XTAL (IRQ Method)

Changing System Clock Source

The figure shows the sequence to change the system clock source from oscillator to either XTAL input or PLL based input source.

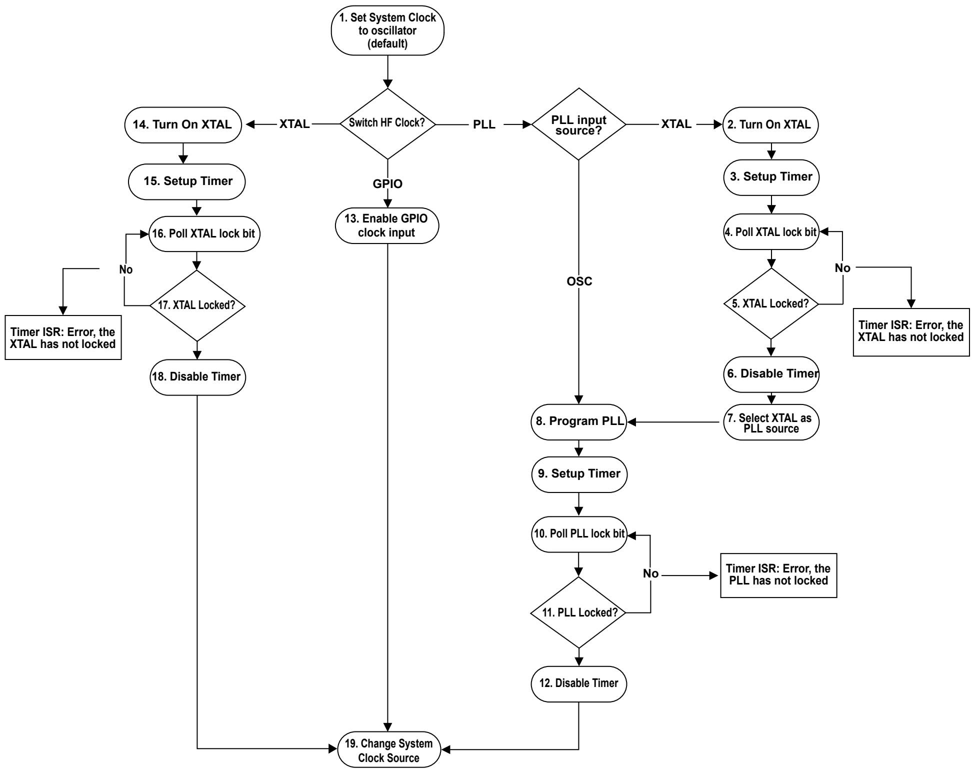


Figure 6-8: Changing System Clock

ADuCM302x CLKG_OSC Register Descriptions

Clocking registers (CLKG_OSC) contains the following registers.

Table 6-4: ADuCM302x CLKG_OSC Register List

Name	Description
CLKG_OSC_CTL	Oscillator Control
CLKG_OSC_KEY	Key Protection for CLKG_OSC_CTL

Oscillator Control

The CLKG_OSC_CTL register is key-protected. To unlock this protection 0xCB14 should be written to CLKG_OSC_KEY before writing to CLKG_OSC_CTL. A write to any other register on the APB bus before writing to CLKG_OSC_CTL will return the protection to the lock state.

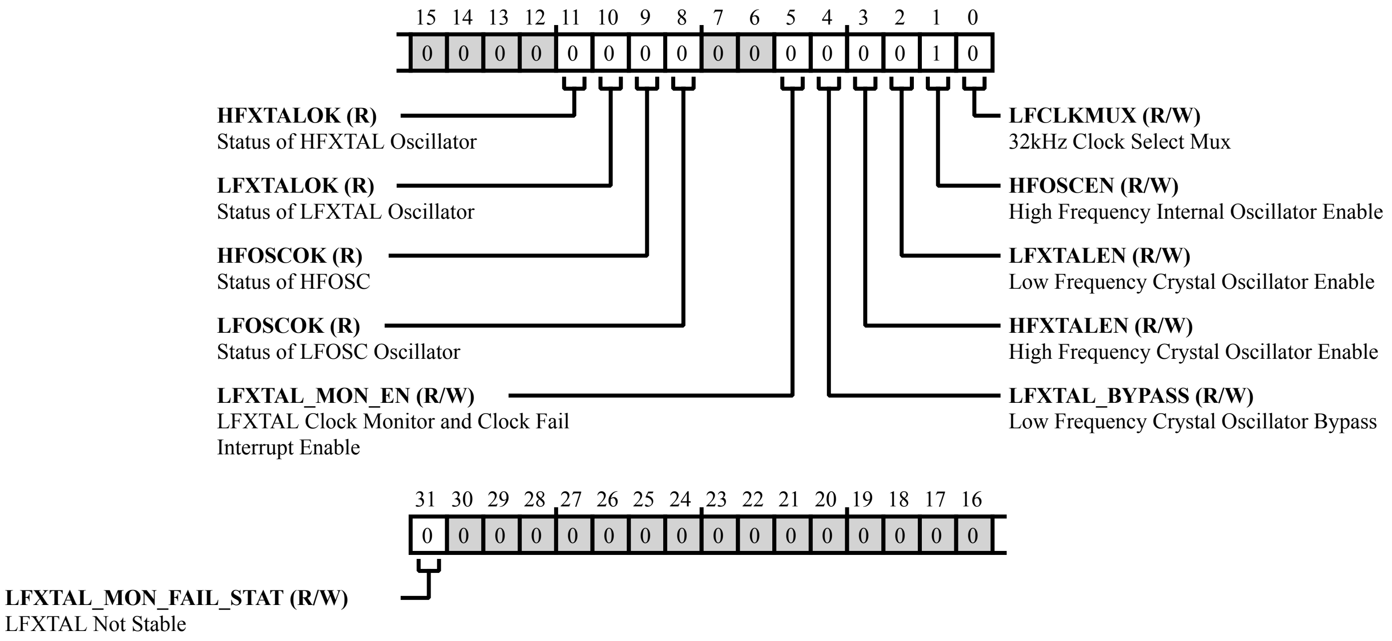


Figure 6-9: CLKG_OSC_CTL Register Diagram

Table 6-5: CLKG_OSC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LFXTAL_MON_FAIL_STAT	LFXTAL Not Stable. This is a sticky bit. If set, it generates interrupt on the RTC1 IRQ line. It can generate interrupt during Hibernate or Active modes. If the flag is set in hibernate mode, the part will wakeup from hibernate mode by asserting interrupt on RTC1 IRQ line. The interrupt and this bit can be cleared by writing 1 to this bit.	
		0	LFXTAL is running fine
		1	LFXTAL is not running The Crystal connection in the board to the chip has possible snapped. Or the External clock driver has stopped giving out clock.

Table 6-5: CLKG_OSC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/NW)	HFXTALOK	Status of HFXTAL Oscillator. This bit indicates when the crystal is stable after it is enabled. This bit is not a monitor and will not indicate a subsequent loss of stability.	
		0	Oscillator is not yet stable or is disabled
		1	Oscillator is enabled and is stable and ready for use
10 (R/NW)	LFXTALOK	Status of LFXTAL Oscillator. This bit indicates when the crystal is stable after it is enabled. This bit is not a monitor and will not indicate a subsequent loss of stability.	
		0	Oscillator is not yet stable or is disabled
		1	Oscillator is enabled and is stable and ready for use
9 (R/NW)	HFOSCOK	Status of HFOSC. This bit indicates when the oscillator is stable after it is enabled. This bit is not a monitor and will not indicate a subsequent loss of stability.	
		0	Oscillator is not yet stable or is disabled
		1	Oscillator is enabled and is stable and ready for use
8 (R/NW)	LFOSCOK	Status of LFOSC Oscillator. This bit indicates when the oscillator is stable after it is enabled. This bit is not a monitor and will not indicate a subsequent loss of stability.	
		0	Oscillator is not yet stable or is disabled
		1	Oscillator is enabled and is stable and ready for use
5 (R/W)	LFXTAL_MON_EN	LFXTAL Clock Monitor and Clock Fail Interrupt Enable. If set, the LFXTAL clock will be monitored using the on chip 32 kHz low frequency oscillator. This clock will be monitored in both Hibernate and Active/Flexi modes. If the LFXTAL clock stops toggling for 2 ms the LFXTAL_MON_FAIL flag is set and interrupt is generated on the RTC1 IRQ line. If cleared, the Monitor circuit is reset and the LFXTAL clock FAIL interrupt is not generated.	
		0	LFXTAL Clock Monitor and clock FAIL interrupt disabled
		1	LFXTAL Clock Monitor and clock FAIL interrupt enabled

Table 6-5: CLKG_OSC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	LFXTAL_BYPASS	Low Frequency Crystal Oscillator Bypass. This bit is used to bypass the low frequency crystal oscillator, and if a clock is supplied externally on one of the LF XTAL pin. The oscillator must be disabled before setting this bit. Disabling of LFXTAL could take a while, hence ensure that the oscillator is disabled by reading the LFXTALEN bit and it reads 0.	
		0	The LFXTAL oscillator is disabled and placed in a low power state
		1	The LFXTAL oscillator is enabled
3 (R/W)	HFXTALEN	High Frequency Crystal Oscillator Enable. This bit is used to enable/disable the oscillator. The oscillator must be stable before use.	
		0	The HFXTAL oscillator is disabled and placed in a low power state
		1	The HFXTAL oscillator is enabled
2 (R/W)	LFXTALEN	Low Frequency Crystal Oscillator Enable. This bit is used to enable/disable the oscillator. The oscillator must be stable before use.	
		0	The LFXTAL oscillator is disabled and placed in a low power state
		1	The LFXTAL oscillator is enabled
1 (R/W)	HFOSCEN	High Frequency Internal Oscillator Enable. This bit is used to enable/disable the oscillator. The oscillator must be stable before use. This bit must be set before the SYSRESETREQ system reset can be initiated.	
		0	The HFOSC oscillator is disabled and placed in a low power state
		1	The HFOSC oscillator is enabled
0 (R/W)	LFCLKMUX	32kHz Clock Select Mux. This clock connects to beeper, RTC. This bit is not reset to default value when soft reset is asserted. While all other bits will be reset, this bit is reset during power-up, external reset, and so on. User must program appropriate value when the software executes after a soft reset. Soft reset is generated by setting the bit in Cortex register AIRCR.SYSRESETREQ.	
		0	Internal 32 kHz oscillator is selected
		1	External 32 kHz crystal is selected

Key Protection for CLKG_OSC_CTL

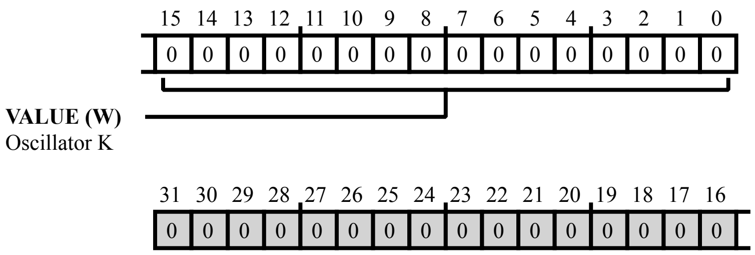


Figure 6-10: CLKG_OSC_KEY Register Diagram

Table 6-6: CLKG_OSC_KEY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (RX/W)	VALUE	Oscillator K. The register is key-protected with value 0xCB14. Must be written after proper key value entered. A write to any other register on the APB bus before writing to `CLKG_OSC_CTL` will return the protection to the lock state.

ADuCM302x CLKG_CLK Register Descriptions

Clocking registers (CLKG_CLK) contains the following registers.

Table 6-7: ADuCM302x CLKG_CLK Register List

Name	Description
`CLKG_CLK_CTL0`	Miscellaneous Clock Settings
`CLKG_CLK_CTL1`	Clock Dividers
`CLKG_CLK_CTL3`	System PLL
`CLKG_CLK_CTL5`	User Clock Gating Control
`CLKG_CLK_STAT0`	Clocking Status

Miscellaneous Clock Settings

This register is used to configure clock sources used by various systems such as the core and memories and peripherals. All unused bits are read only returning a value of 0. Writing unused bits has no effect.

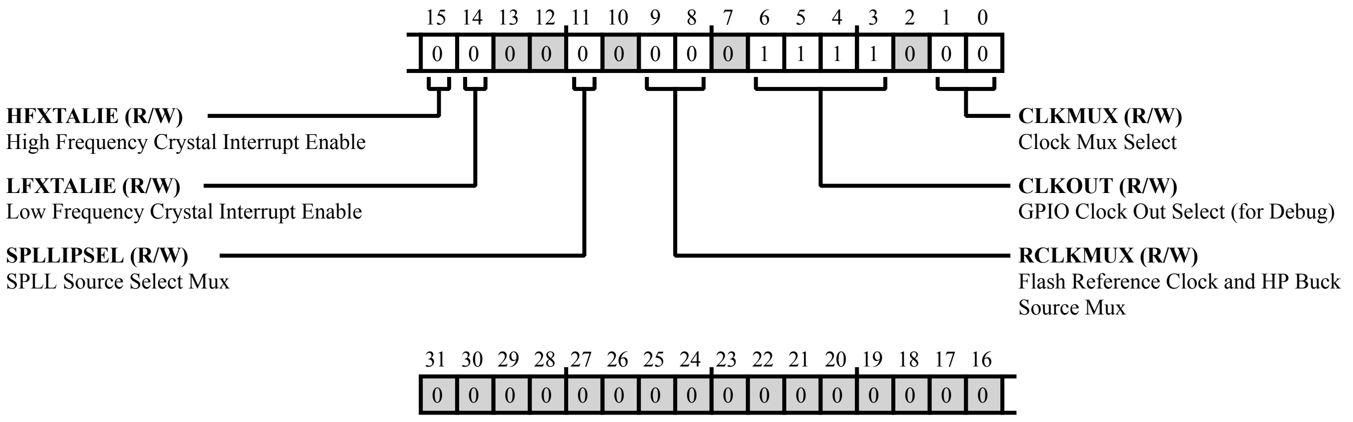


Figure 6-11: CLKG_CLK_CTL0 Register Diagram

Table 6-8: CLKG_CLK_CTL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	HFXTALIE	High Frequency Crystal Interrupt Enable. Controls if the core should be interrupted on a `CLKG_CLK_STAT0.HFXTALOK` or `CLKG_CLK_STAT0.HFXTALNOK` status or if no interrupt should be generated. This bit should not be cleared while a core interrupt is pending.	
		0	An interrupt to the core is not generated on a HFXTALOK or HFXTALNOK.
		1	An interrupt to the core is generated on a HFXTALOK or HFXTALNOK.
14 (R/W)	LFXTALIE	Low Frequency Crystal Interrupt Enable. Controls if the core should be interrupted on a `CLKG_CLK_STAT0.LFXTALOK` or `CLKG_CLK_STAT0.LFXTALNOK` status or if no interrupt should be generated. This bit should not be cleared while a core interrupt is pending.	
		0	An interrupt to the core is not generated on a LFXTALOK or LFXTALNOK.
		1	An interrupt to the core is generated on a LFXTALOK or LFXTALNOK.

Table 6-8: CLKG_CLK_CTL0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	SPLLIPSEL	SPLL Source Select Mux. `CLKG_CLK_CTL0.SPLLIPSEL` selects which source clock is feed to the SPLL. The selection should be made before the SPLL is enabled. Selection should not be changed after the SPLL is enabled.	
		0	Internal HF oscillator is selected
		1	External HF XTAL oscillator is selected
9:8 (R/W)	RCLKMUX	Flash Reference Clock and HP Buck Source Mux. These bits are to be programmed when both external crystal and HF oscillator are stable and running. The clock muxed output supplies Flash reference clock and HP buck clock (200 kHz).	
		0	Sourcing from HFOSC Clock - 26MHz Set `CLKG_CLK_CTL0.RCLKMUX` = 00 if HFOSC (26 MHz) is the source for HP Buck divider and flash reference clock generators
		1	Reserved
		2	Sourcing from External HFXTAL (26MHz) Set `CLKG_CLK_CTL0.RCLKMUX` = 10 if following is true: 1. HFXTAL to be the source for HP Buck divider and flash reference clock generators 2. Frequency of HFXTAL connected to chip = 26Mhz
		3	Sourcing from External HFXTAL (16MHz) Set `CLKG_CLK_CTL0.RCLKMUX` = 11 if following is true: 1. HFXTAL to be the source for HP Buck Divider and flash-reference clock generators 2. Frequency of HFXTAL connected to chip = 16Mhz If this option is programmed, the clock out is 16 MHz instead of 26 MHz, the dividers of following clocks get auto-configured. RCLK divider is bypassed. RCLK = 16 MHz. HP Buck clock divider is autoconfigured to generate 200 kHz clock. Note: If external crystal clock is 16 MHz instead of 26 MHz, HP Buck non-overlapping phases increases. This is not recommended if HP Buck is enabled.

Table 6-8: CLKG_CLK_CTL0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6:3 (R/W)	CLKOUT	GPIO Clock Out Select (for Debug). Used for debug purpose only - default value is F	
		0	ROOT_CLK
		1	LF_CLK
		2	ACLK
		3	HCLK_BUS
		4	HCLK_CORE
		5	PCLK
		6	RCLK (Ref clock for flash controller timer)
		7	RHP_CLK (mux of HF OSC, HF XTAL clock)
		8	GPT0_CLK
		9	GPT1_CLK
		10	HCLK_P (Connects to peripherals operating on HCLK)
		11	PLL out clock
		12	RTC0 1Hz generated clock
		13	HPBUCK_CLK
		14	HPBUCK_Non_overlap_clk
		15	RTC1 1Hz generated clock
1:0 (R/W)	CLKMUX	Clock Mux Select. Determines which single shared clock source is used by the PCLK, and HCLK dividers. Ensure that an enabled active stable clock source is selected.	
		0	High frequency internal oscillator is selected
		1	High frequency external crystal oscillator is selected
		2	System PLL is selected
		3	External GPIO port is selected

Clock Dividers

This register is used to set the divide rates for the HCLK, PCLK and ACLK dividers. It can be written to at any time. All unused bits are read only, returning a value of 0. Writing to unused bits has no effect.

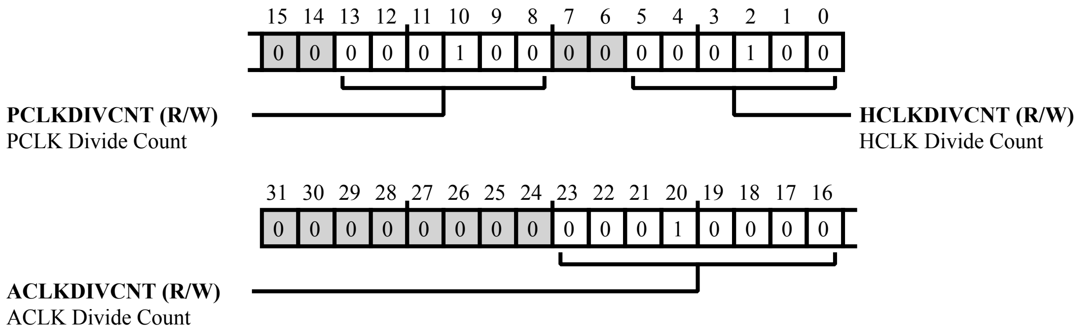


Figure 6-12: CLKG_CLK_CTL1 Register Diagram

Table 6-9: CLKG_CLK_CTL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	ACLKDIVCNT	ACLK Divide Count. Determines the ACLK rate based on the following equation: ACLK = ROOT_CLK/ACLKDIVCNT For example, if ROOT_CLK is 26 MHz and ACLKDIVCNT = 0x1, ACLK operates at 26 MHz. The value of `CLKG_CLK_CTL1.ACLKDIVCNT` takes effect after a write access to this register and typically takes one ACLK cycle. This register can be read at any time and can be written to at any time. The reset divider count is 0x4. Value range is from 1 to 32. Values larger than 32 are saturated to 32. Values 0 and 1 have the same results as divide by 1. Default value is configured such that ACLK = 6.5 MHz
13:8 (R/W)	PCLKDIVCNT	PCLK Divide Count. Determines the PCLK rate based on the following equation: PCLK = ROOT_CLK/PCLKDIVCNT For example, if ROOT_CLK is 26 MHz and PCLKDIVCNT = 0x2, PCLK operates at 13 MHz. The value of `CLKG_CLK_CTL1.PCLKDIVCNT` takes effect after a write access to this register and typically takes 2 to 4 PCLK cycles. This register can be read at any time and can be written to at any time. The reset divider count is 0x4. Value range is from 1 to 32. Values larger than 32 are saturated to 32. Values 0 and 1 have the same results as divide by 1. The default value of this register is configured such that PCLK = 6.5 MHz

Table 6-9: CLKG_CLK_CTL1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
5:0 (R/W)	HCLKDIVCNT	HCLK Divide Count. Determines the HCLK rate based on the following equation: HCLK = ROOT_CLK/HCLKDIVCNT For example, if ROOT_CLK is 26 MHz and HCLKDIVCNT = 0x1, HCLK operates at 26 MHz. The value of `CLKG_CLK_CTL1.HCLKDIVCNT` takes effect after a write access to this register and typically takes 2 to 4 PCLK cycles (not HCLK cycles). This register can be read at any time and can be written to at any time. The reset divider count is 0x4. Value range is from 1 to 32. Values larger than 32 are saturated to 32. Values 0 and 1 have the same results as divide by 1. Default value of this register is configured such that HCLK = 6.5 MHz

System PLL

This register is used to control the system PLL. It should be written to only when the PLL is not selected as the clock source (ROOT_CLK). All unused bits are read only, returning a value of 0. Writing to unused bits has no effect.

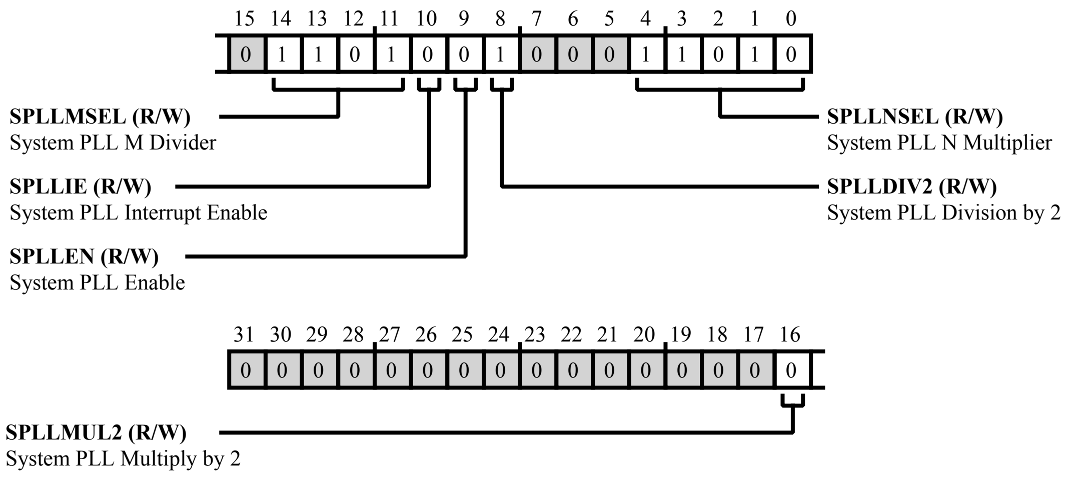


Figure 6-13: CLKG_CLK_CTL3 Register Diagram

Table 6-10: CLKG_CLK_CTL3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	SPLLMUL2	System PLL Multiply by 2. This bit is used to configure if the VCO clock frequency should be multiplied by 2 or 1.	
		0	PLL out frequency = SPLLNSEL * 1 / ((SPLLDIV2+1) * SPLLMSEL)
		1	PLL out frequency = SPLLNSEL * 2 / ((SPLLDIV2+1) * SPLLMSEL)
14:11 (R/W)	SPLLMSEL	System PLL M Divider. Sets the M value used to obtain the multiplication factor N/M of the PLL. if `CLKG_CLK_CTL3.SPLLMSEL` <= 2, Divider M value = 2. For example, 0000: M set to 2 (Divide by 2) 0001: M set to 2 (Divide by 2) 0010: M set to 2 (Divide by 2) 0011: M set to 3 (Divide by 3) 0100: M set to 4 (Divide by 4) 1111: M set to 15 (Divide by 15)	

Table 6-10: CLKG_CLK_CTL3 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W)	SPLLIE	System PLL Interrupt Enable. Controls if the core should be interrupted on a PLL lock/PLL unlock or no interrupt generated. This bit must not be cleared while a core interrupt is pending.	
		0	An interrupt to the core will not be generated on a PLL lock or PLL unlock
		1	An interrupt to the core will be generated on a PLL lock or PLL unlock
9 (R/W)	SPLLEN	System PLL Enable. Controls if the PLL should be enabled or placed in its low power state. This bit should only be set while the System PLL is not selected as the system clock source (`CLKG_CLK_CTL0.CLKMUX`).	
		0	The PLL is disabled and is in its power down state
		1	The PLL is enabled. Initially the PLL will not run at the selected frequency. After a stabilization period the PLL will lock onto the selected frequency at which time it can be selected as a system clock source (CLKMUX bits)
8 (R/W)	SPLLDIV2	System PLL Division by 2. Controls if an optional divide by two is placed on the PLL output. This guarantees a balanced output duty cycle output at the cost of doubling the PLL frequency (power). This bit should not be modified after `CLKG_CLK_CTL3.SPLLEN` is set. This bit can be written at the same time `CLKG_CLK_CTL3.SPLLEN` is set. This bit is set or reset based on the following constraints: The VCO output range is 32 MHz to 60 MHz The SPLL output range is 16 MHz to 60 MHz By default, this bit is set.	
		0	The System PLL is not divided. Its output frequency equals that selected by the N/M ratio
		1	The System PLL is divided by two. Its output frequency equals that selected by the N/M ratio with an additional /2 divide
4:0 (R/W)	SPLLNSEL	System PLL N Multiplier. Sets the N value used to obtain the multiplication factor N/M of the PLL. The default value is 011010 (Mul by 26). Minimum valid value is 8 and writing any value less than 8 will force it to be 8. Maximum valid value is 31. Do not program the SPLL to an output clock lower than 16 MHz or higher than 60 MHz.	

User Clock Gating Control

This register is used to control the gates of the peripheral clocks.

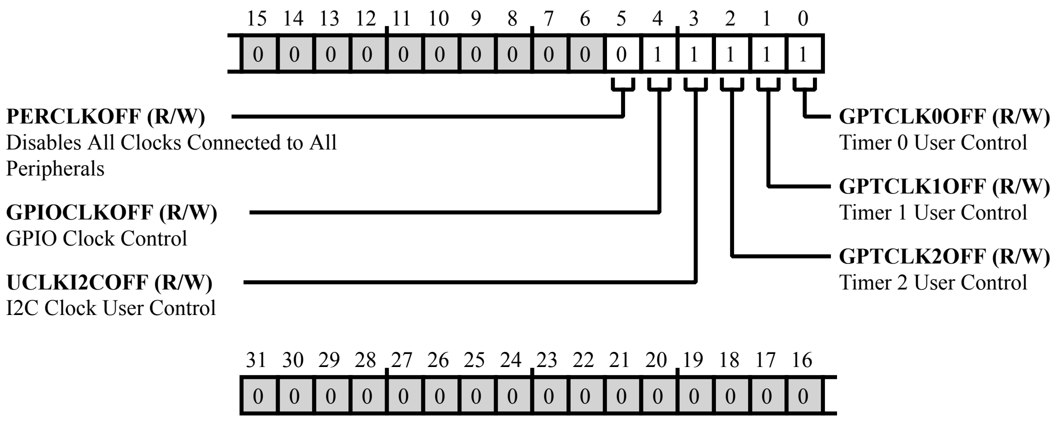


Figure 6-14: CLKG_CLK_CTL5 Register Diagram

Table 6-11: CLKG_CLK_CTL5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	PERCLKOFF	Disables All Clocks Connected to All Peripherals. After setting this bit, any read/write to any of the above module register will automatically reset the `CLKG_CLK_CTL5.PERCLKOFF` to 0, and the read/write transaction is honored. After setting `CLKG_CLK_CTL5.PERCLKOFF` = 1, if user reads the `CLKG_CLK_CTL5` register, `CLKG_CLK_CTL5.PERCLKOFF` will be automatically cleared and read as 0. Recommended usage: 1. Ensure that DMA transactions are done and no more transaction is expected from the DMA. 2. Ensure that `CLKG_CLK_CTL5.PERCLKOFF` bit write is last write and no write/read to any of above module registers is done after setting this bit. Otherwise, the bit will be cleared.	
		0	Clocks to all peripherals are active
		1	Clocks to all peripherals are gated off

Table 6-11: CLKG_CLK_CTL5 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	GPIOCLKOFF	GPIO Clock Control. This bit disables the ACLK. It controls the gate on the ACLK out from ACLK Divider. This ACLK control is in Active and Flexi modes. In Hibernate and Shutdown modes, the ACLK is always off, and this bit has no effect. This bit is not automatically cleared. User has to explicitly enable or disable this bit to control ACLK out. Note: Before programming `CLKG_CLK_CTL1.ACLKDIVCNT` register, this bit (`CLKG_CLK_CTL5.GPIOCLKOFF`) needs to be cleared to 0. Otherwise, the `CLKG_CLK_CTL1.ACLKDIVCNT`is not taken into effect.	
		0	GPIO Clock is enabled
		1	GPIO Clock is disabled
3 (R/W)	UCLKI2COFF	I2C Clock User Control. This bit disables the I2C UCLK. It controls the gate on the I2C UCLK in Active and Flexi modes. In Hibernate and Shutdown modes the I2C UCLK is always off, and this bit has no effect. Note will automatically clear this bit if I2C is accessed via the APB bus.	
		0	I2C Clock is enabled
		1	I2C Clock is disabled
2 (R/W)	GPTCLK2OFF	Timer 2 User Control. This bit disables the Timer 2 clock (muxed version). It controls the gate on the Timer 2 clock in Active and Flexi power modes. In Hibernate and Shutdown modes, the Timer clock is always off, and this bit has no effect. Note will automatically clear this bit if Timer 2 is accessed via the APB bus.	
		0	TMR2 clock is enabled
		1	TMR2 clock is disabled
1 (R/W)	GPTCLK1OFF	Timer 1 User Control. This bit disables the Timer 1 clock (muxed version). It controls the gate on the Timer 1 clock in Active and Flexi power modes. In Hibernate and Shutdown modes, the Timer clock is always off, and this bit has no effect. Note will automatically clear this bit if Timer 1 is accessed via the APB bus.	
		0	TMR1 clock is enabled
		1	TMR1 clock is disabled

Table 6-11: CLKG_CLK_CTL5 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	GPTCLK0OFF	Timer 0 User Control. This bit disables the Timer 0 clock (muxed version). It controls the gate on the Timer 0 clock in Active and Flexi power modes. In Hibernate and Shutdown modes, the Timer clock is always off, and this bit has no effect. Note will automatically clear this bit if Timer 0 is accessed via the APB bus.	
		0	TMR0 clock is enabled
		1	TMR0 clock is disabled

Clocking Status

Monitors PLL and Oscillator status. With interrupts enabled, the user can run initialization code or idle the core while the clock components stabilize.

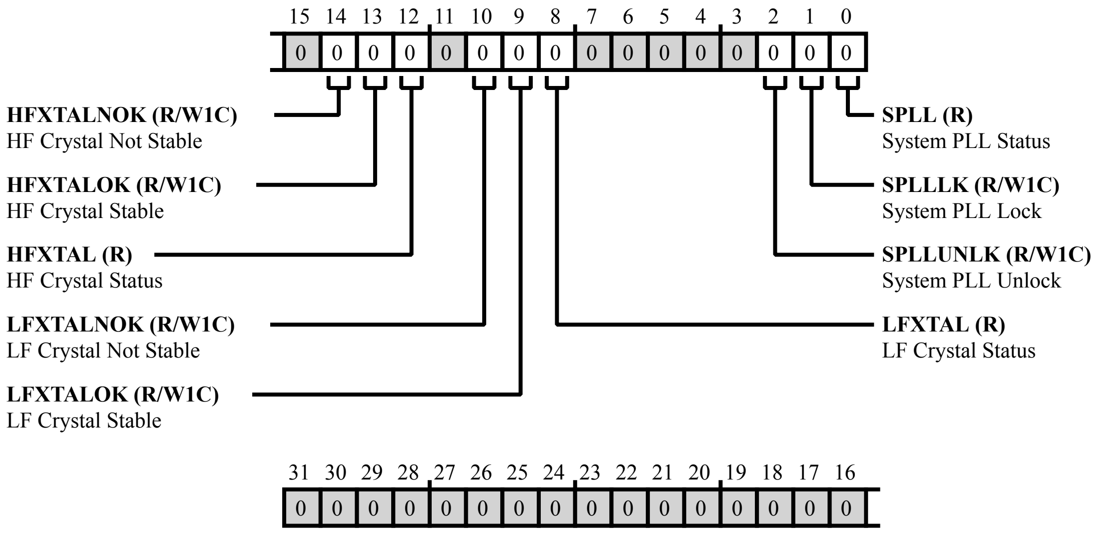


Figure 6-15: CLKG_CLK_STAT0 Register Diagram

Table 6-12: CLKG_CLK_STAT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W1C)	HFXTALNOK	HF Crystal Not Stable. This bit indicates the XTAL was successfully disabled. This bit is not associated with continuous monitoring of the XTAL and will not be set in the event the XTAL becomes unstable. This bit is sticky. Write a 1 to this location to clear it. If enabled, an interrupt can be associated with this bit.	
		0	HF crystal stable signal has not been de-asserted.
		1	HF crystal stable signal has been de-asserted.
13 (R/W1C)	HFXTALOK	HF Crystal Stable. This bit is sticky. It is used to interrupt the core when interrupts are enabled. Write a 1 to this location to clear it.	
		0	HF crystal stable signal has not been asserted.
		1	HF crystal stable signal has been asserted.

Table 6-12: CLKG_CLK_STAT0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/NW)	HFXTAL	HF Crystal Status. This bit assists in determining when the XTAL is initially stable and ready to use. This bit does not perform a continuous monitoring function and will not clear in the event an XTAL becomes unstable.	
		0	HF crystal is not stable or not enabled.
		1	HF crystal is stable.
10 (R/W1C)	LFXTALNOK	LF Crystal Not Stable. This bit indicates the XTAL was successfully disabled. This bit is not associated with continuous monitoring of the XTAL and will not be set in the event the XTAL becomes unstable. This bit is sticky. Write a 1 to this location to clear it. If enabled, an interrupt can be associated with this bit.	
		0	LF crystal stable signal has not been de-asserted.
		1	LF crystal stable signal has been de-asserted.
9 (R/W1C)	LFXTALOK	LF Crystal Stable. This bit is sticky. It is used to interrupt the core when interrupts are enabled. Write a 1 to this location to clear it.	
		0	LF crystal stable signal has not been asserted.
		1	LF crystal stable signal has been asserted.
8 (R/NW)	LFXTAL	LF Crystal Status. This bit assists in determining when the XTAL is initially stable and ready to use. This bit does not perform a continuous monitoring function and will not clear in the event an XTAL becomes unstable.	
		0	LF crystal is not stable or not enabled.
		1	LF crystal is stable
2 (R/W1C)	SPLLUNLK	System PLL Unlock. This bit is sticky. `CLKG_CLK_STAT0.SPLLUNLK` is set when the PLL looses its lock. `CLKG_CLK_STAT0.SPLLUNLK` is used as the interrupt source to signal the core that a lock was lost. Writing a one to this bit clears it. `CLKG_CLK_STAT0.SPLLUNLK` will not set again unless the System PLL gains a lock and subsequently looses again.	
		0	No loss of PLL lock was detected
		1	A PLL loss of lock was detected

Table 6-12: CLKG_CLK_STAT0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	SPLLLK	System PLL Lock. This bit is sticky. `CLKG_CLK_STAT0.SPLLLK` is set when the PLL locks. `CLKG_CLK_STAT0.SPLLLK` is used as the interrupt source to signal the core that a lock was detected. Writing a one to this bit clears it. `CLKG_CLK_STAT0.SPLLLK` will not set again unless the System PLL looses lock and subsequently locks again.	
		0	No PLL lock event was detected
		1	A PLL lock event was detected
0 (R/NW)	SPLL	System PLL Status. Indicates the current status of the PLL. Initially the System PLL will be unlocked. After a stabilization period the PLL will lock and be ready for use as the system clock source. This is a read only bit. A write has no effect.	
		0	The PLL is not locked or not properly configured. The PLL is not ready for use as the system clock source
		1	The PLL is locked and is ready for use as the system clock source

7 Reset (RST)

The ADuCM302x MCU has the following resets:

- External
- Power-on
- Watchdog timeout
- Software system

The software system reset is provided as part of the Cortex-M3 core.

To generate a system reset through software, the application interrupt/reset control register must be set to a value of 0x05FA0004. This register is part of the NVIC subsystem and is located at address 0xE000ED0C.

For more information about software reset, refer to the *ARM Cortex-M3 Technical Reference Manual*.

A hardware reset is performed by toggling the SYS_HWRST pin, which is active low.

The `PMG_RST_STAT` register indicates the source of the last reset. The register can be used during a reset exception service routine to identify the source of the reset.

Table 7-1: Reset Types

Reset	Reset External Pins to Default State	Execute Kernel	Reset All MMRs Except **`PMG_RST_STAT`**	Reset All Peripherals	Valid SRAM	**`PMG_RST_STAT`** After Reset Event
Software	Yes[*1]	Yes	Yes	Yes	Yes/no[*2]	`PMG_RST_STAT.SWRST` = 1
Watchdog	Yes	Yes	Yes	Yes	Yes/no[*2]	`PMG_RST_STAT.WDRST` = 1
External reset pin	Yes	Yes	Yes	Yes	Yes/no[*2]	`PMG_RST_STAT.EXTRST` = 1
POR	Yes	Yes	Yes	Yes	No	`PMG_RST_STAT.POR` = 1 `PMG_RST_STAT.PORSRC` has info on cause of POR reset

*1 GPIOx returns to its default state, that is, same as a POR event.

*2 RAM is not valid in the case of a reset following a UART download.

ADuCM302x Reset Register Description

Table 7-2: ADuCM302x Reset Register List

Name	Description	Reset	Access
`PMG_RST_STAT`	Reset Status	0x000000XX	R/W

Reset Status

This register is recommended to be read at the beginning of the user code to determine the cause of the reset. Default values of this register is unspecified as the cause of reset can be any source.

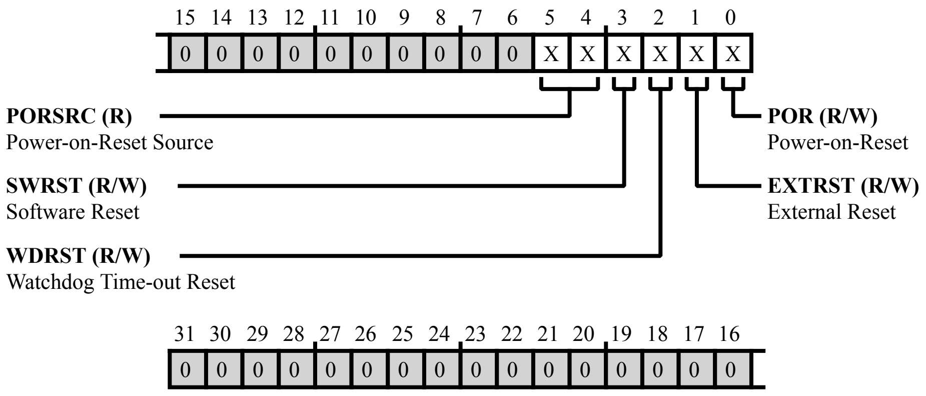


Figure 7-1: PMG_RST_STAT Register Diagram

Table 7-3: PMG_RST_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/NW)	PORSRC	Power-on-Reset Source. This bit contains additional details after a Power-on-Reset occurs.	
		0	POR triggered because VBAT drops below Fail Safe
		1	POR trigger because VBAT supply (VBAT < 1.7 V)
		2	POR triggered because VDD supply (VDD < 1.08 V)
		3	POR triggered because VREG drops below Fail Safe
3 (R/W)	SWRST	Software Reset. Set automatically to 1 when the system reset is generated. Cleared by writing 1 to the bit. This bit is also cleared when power-on-reset is triggered.	
2 (R/W)	WDRST	Watchdog Time-out Reset. Set automatically to 1 when a watchdog timeout occurs. Cleared by writing 1 to the bit. This bit is also cleared when power-on-reset is triggered	
1 (R/W)	EXTRST	External Reset. Set automatically to 1 when an external reset occurs. Cleared by writing 1 to the bit. This bit is also cleared when power-on-reset is triggered	
0 (R/W)	POR	Power-on-Reset. Set automatically when a Power-on-Reset occurs. Cleared by writing one to the bit.	

8 Flash Controller (FLASH)

The ADuCM302x MCU includes up to 256 KB of embedded flash memory available for access through the flash controller. The embedded flash has a 72-bit wide data bus providing two 32-bit words of data and one corresponding 8-bit ECC byte per access.

The flash controller is coupled with a cache controller module, which provides two AHB ports:

- DCode for reading data
- ICode for reading instructions

A prefetch mechanism is implemented in the flash controller to optimize ICode read performance.

Flash writes are supported by a key hole mechanism through APB writes to memory mapped registers. The flash controller includes support for DMA based key-hole writes.

Flash Features

The flash memory used by the ADuCM302x MCU has the following features:

- Prefetch buffer provides optimal performance when reading consecutive addresses on the ICode interface.
- Simultaneous ICode and DCode read accesses (DCode has priority on contention; simultaneous reads are possible if ICode returns buffered data from prefetch).
- DMA based key hole writes (including address auto-increment for sequential accesses).
- ECC for error detection and/or correction. Errors and corrections may be reported as Bus Errors (on the ICode/DCode buses), interrupts, or ignored.

Supported Commands

- READ: Supported through the ICode and DCode interfaces.
- WRITE: Provided by a key-hole mechanism through memory mapped registers.
- MASSERASE: Clears all user data and program code.
- ERASEPAGE: Clears user data and/or program code from a 2 KB page in flash.
- SIGN: Generate and verify signatures for any set of contiguous whole pages of user data and/or program code.

- ABORT: Terminate a command in progress.

Protection and Integrity Features

- Fixed user key required for running protected commands including mass erase and page erase.
- Optional and user definable Write protection for user accessible memory.
- Optional 8-bit Error Correction Code (ECC): It is enabled by default. It can be disabled by user code.

Flash Functional Description

This section provides information on the flash memory functions used by the ADuCM302x MCU.

Organization

The flash IP provides a 64-bit data bus, plus 8 bits for ECC meta data corresponding to that data. The memory is organized as a number of pages, each 2 KB in size plus 256 bytes reserved for ECC.

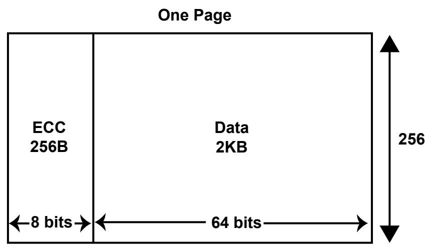


Figure 8-1: One Page of Flash Memory

These pages of memory are categorically divided into two sections: Info Space and User Space. Total device storage is generally described as the size of User Space.

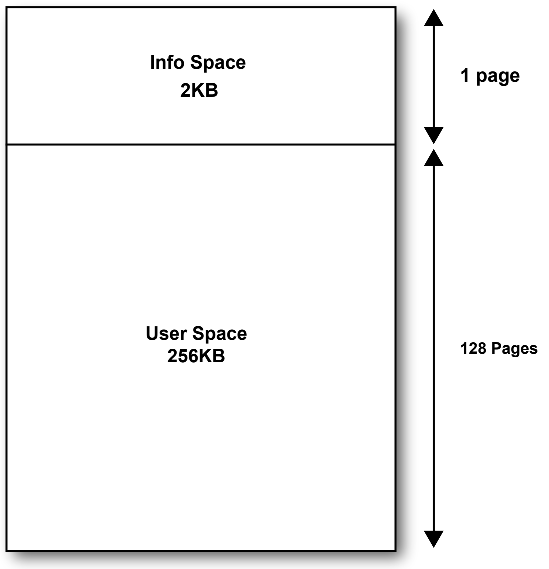


Figure 8-2: Flash Info and User Spaces

Info Space is reserved for use by Analog Devices and generally stores a boot loader (kernel), several trim and calibration values, and other device specific meta data. Most of info space is left readable by user code but attempted erasures and writes are denied.

User Space is the portion of flash memory intended for user data and program code. Several small address ranges in user space are used by the flash controller as meta data to enable various protection and integrity features.

The ***Flash Info and User Spaces*** figure is for the ADuCM3029 device that has 128 pages. ADuCM3027 device has 64 pages.

Info Space

User Accessible Data

While Info space is generally reserved for use only by Analog Devices, all but the top 128 bytes of info space are readable by user code. A 64-bit Manufacture ID is located at address 0x40764. The ADI secure bootloader is responsible to place the Manufacture ID on this location. Other potential read-only data may be made available to the user within the scope of the info space. Such meta data is software defined. Therefore, the addresses and data types are not defined by the flash controller.

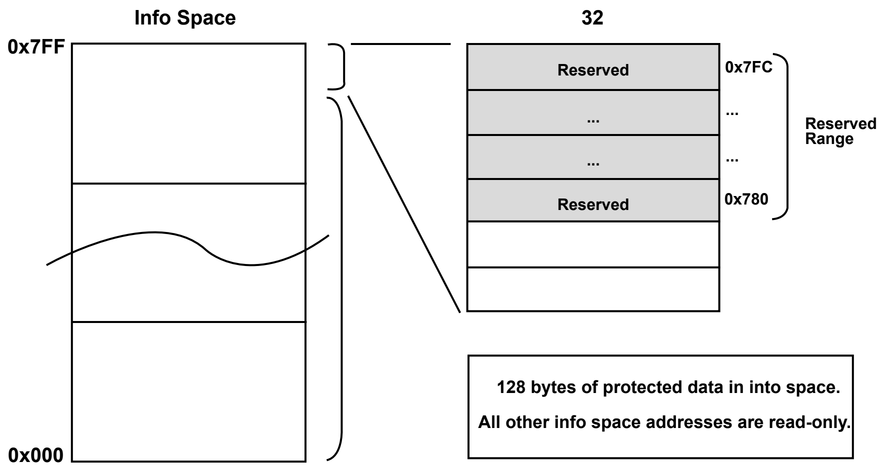

Info Space
0x7FF
0x000
32
Reserved
...
...
Reserved
0x7FC
...
...
0x780
Reserved Range
128 bytes of protected data in into space.
All other info space addresses are read-only.

Figure 8-3: User Readable Locations in Info Space

The top 128 bytes of info space are protected and cannot be read by user code (attempted reads return bus errors). The remainder of the info space is freely readable by user code. None of info space may be programmed or erased by the user (command is denied).

Bus errors are generated if user code attempts to read the protected range of info space. Writes and Erases targeting info space are denied and appropriate bits are set in the `FLCC_STAT` register.

User code may perform a Mass Erase command on the part without affecting the content of the Info Space. This provides a mechanism to upload new user firmware and data to a device without affecting ADIs secure boot loader.

User Space Meta Data

User Space is the portion of flash memory intended for user data and program code. Several small address ranges in user space are used by the flash controller as meta-data to enable various protection and integrity features.

The top four words of user space are reserved for a Signature, the user Write Protection word, and two reserved words.

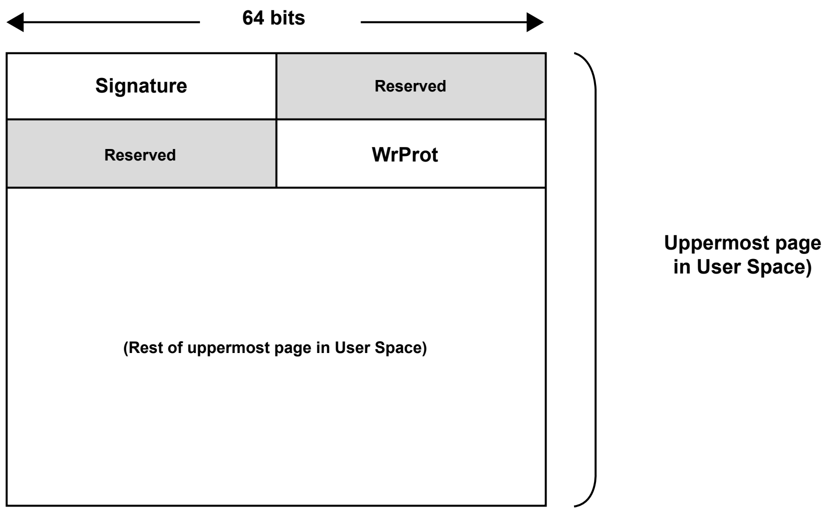


Figure 8-4: User Space Meta Data

When writing to these locations, user code must take care to always write 0xFFFF_FFFF to the reserved locations. If the user intends to write to either location at runtime, these reserved locations must remain all 1s (0xFFFFFFFF). If data is stored to these reserved locations prior to runtime modification of the neighboring meta-data spurious, ECC errors are likely to be generated when this meta data is read.

The meta-data described above is utilized by the flash controller to enable Protection and Integrity features. For more information about this, refer to Protection and Integrity. User must program these values. A brief description of these fields are as follows:

- Signature (optional): A 32-bit CRC checksum stored in the Signature field enables user code to request an integrity check of User Space.
- WrProt: A user programmable bit used to make blocks flash pages read-only (protected from both writes and erases). ADI secure bootloader reads and enforces the programmed WrProt meta data.

Flash Access

Flash memory may be read, written, and erased by user code. Read access is provided through the cache controller using two AHB ports: ICode for instructions and DCode for data. Write access is provided through key-hole writes using APB control of memory mapped registers. The key hole write implementation includes support for both DMA based and manual user initiated writes.

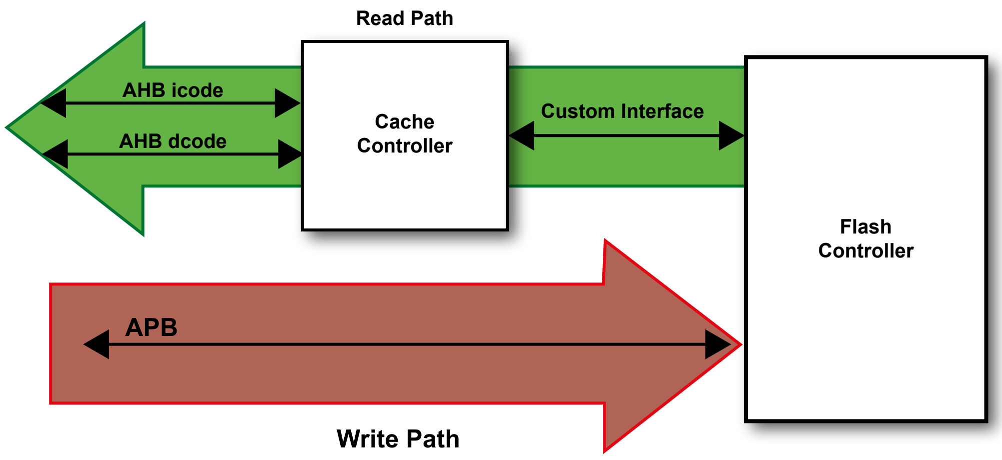


Figure 8-5: Read and Write Data Paths

Bus errors are generated if user code reads from a protected or out of bounds address. Writes or Erasures of protected addresses result in appropriate error flags set in the `FLCC_STAT` register. Address setup for writes and erasures is automatically constrained to the flash address range.

Reading Flash

The flash controller provides two interfaces for reading non-volatile storage: ICode and DCode.

The ICode and DCode interfaces are accessed through the cache controller module through AHB. The flash controller includes a pre-fetch buffer for ICode. Due to this prefetch buffer, it is possible to return data on both ICode and DCode interfaces in the same cycle.

Flash memory is available to be read only after an automatic initialization process. Attempts to read during the flash controller initialization will stall. Reads will also stall if the flash controller is already busy performing another command (for example, writing the flash) unless those reads would be satisfied by the prefetch buffer.

Erasing Flash

The flash controller provides page-level granularity when erasing user space (ERASEPAGE command). Alternatively, user code may erase the entirety of user space at once (MASSERASE command). The two commands have the same execution time.

Write protected pages cannot be erased; the command would be denied. If any pages are write protected, a mass erase command is also be denied. Users wishing to write protect user space and planning to perform mass erases (for example, for future firmware upgrades) must take this into consideration.

Writing Flash

Flash memory operates by setting bits to 1 when erased, and clearing them to 0 when writing data. Any generalized write accesses must be prefixed by an Erase operation.

Initial uploading of user content generally occurs immediately following a Mass Erase operation.

Subsequent modifications of already-written locations in the flash involve the following operations by the user code:

- Copying a full page to SRAM
- Erasing the affected page
- Modifying the in-memory content
- Writing the page back to flash

User Space protections may prevent a page erase operation. For more information about this, refer to Protection and Integrity.

All User Space protections are cleared automatically after a successful Mass Erase or Blank Check (a blank check passes only if the entire user space is already erased). In these two cases, there is no user space content to protect.

Key Hole Writes

A key hole write is an indirect write operation where the user code programs the memory mapped registers with target Address and Data values, then commands the flash controller to perform a write operation in the background. The flash controller supports Write access to the flash memory only through key hole writes. This constraint on Write access enables the flash controller to guarantee that writes occur properly as atomic DWORD (64-bit) operations with an associated ECC byte, if ECC is enabled. For more information about this, refer to ECC.

If ECC is enabled, multiple writes to a single DWORD (64-bit) location cannot be performed without erasing the affected page between writes else ECC errors are reported.

A maximum of two total writes are permitted to a single flash location (64-bit DWORD) between erasures (regardless of ECC or data values), as per the flash IP specifications. Writing any location more than twice per erasure may damage the non-volatile memory or reduce its useful life. If multiple writes per location are required, user code must disable ECC for some region of flash and target that region for these write operations.

Key hole operations consist of writes to:

- `FLCC_KH_ADDR.VALUE`: The target address in flash (for example, 0x104). The flash controller automatically trims the lower bits to make the address DWORD aligned.
- `FLCC_KH_DATA0.VALUE`: Bits [31:0] of the 64-bit DWORD to be written (for example, 0x7654_3210).
- `FLCC_KH_DATA1.VALUE`: Bits [63:32] of the 64-bit DWORD to be written (for example, 0xFEDC_BA98).
- `FLCC_CMD.VALUE`: Assert the write command (0x4) in flash.

After the write command is asserted, the flash controller initiates a 64-bit dual word write to the address provided in `FLCC_KH_ADDR.VALUE`.

Word (32-bit), Halfword (16-bit), and Byte (8-bit) writes are not supported. As only 0s are written to the flash, masking may be used to write individual bits or bytes (as per ECC and flash IP specifications).

User code must not write to any of these key hole registers when DMA access is enabled (`FLCC_UCFG.KHDMAEN` is set). Writing to these registers manually when DMA is enabled may result in spurious flash writes and could put the DMA and flash controller out of sync, potentially hanging the DMA controller for long duration (~20-40 us).

Burst Writes

Each 2 KB page of flash memory consists of eight rows, 256 bytes each. Design constraints for programming the flash IP enable back-to-back writes within a single row to complete more quickly than the equivalent writes across row boundaries. For optimizing writes, user code must attempt to write flash memory in (aligned) blocks of up to 256 bytes.

Table 8-1: Flash Addressing by Rows

Row 1023	0x3FFFC 0x3FFF8 0x3FF04 0x3FF00
Row 0	0xFC 0xF8 0x4 0x0

To benefit from this write-performance gain, the second (and subsequent) 64-bit write must be requested before the flash controller completes the first write. The flag `FLCC_STAT.WRALCOMP` is set when the first 64-bit write operation is nearing completion (~20 us remaining in the write cycle). This provides user code with a manageable window of time in which to assert the next write request. This flag may be polled or an interrupt may optionally be generated when it is asserted.

The following status flags are also available:

- `FLCC_STAT.CMDBUSY`: High when the controller is processing a command from the `FLCC_CMD` register.
- `FLCC_STAT.WRCLOSE`: High during the first half of a write command, cleared when key hole registers are free to be programmed for a subsequent write command.
- `FLCC_STAT.CMDCOMP`: Sticky flag, set when a command is completed (write, erase and so on). Write one to clear.
- `FLCC_STAT.WRALCOMP`: Sticky flag, set when ~20-40 us remains for an ongoing write command. Write one to clear.

The following steps describe how to perform multiple writes with the potential to burst within a row:

- Wait for command busy flag to clear (to ensure on prior command is ongoing).
- Disable all interrupts so that sequenced writes to `FLCC_KH_ADDR`, `FLCC_KH_DATA0`, `FLCC_KH_DATA1`, and `FLCC_CMD` registers are not interrupted by an ISR which might also have flash writes.
- Request first 64-bit write through the key hole access.
- The flash controller starts the write process after the write command is written to the `FLCC_CMD` register.
- Check if the write command is accepted (read `FLCC_STAT` register to see if any error flags are set. If the command results in an error, the write is not performed).

- Set the FLCC_IEN.WRALCMPLT bit (to enable interrupt generation when WrAlComp is asserted) and re-enable other interrupts.
- Continue the user program. WrAlComp interrupt will vector to an interrupt service routine to request the next write at appropriate time.
- BurstWriteISR: Interrupt service routine (ISR) called when n interrupt is generated by WrAlComp.
- Disable all interrupts so that sequenced writes to FLCC_KH_ADDR, FLCC_KH_DATA0, FLCC_KH_DATA1, and FLCC_CMD registers are not interrupted by another ISR which may also have flash writes.
- Read the FLCC_STAT register to verify the state of several status flags. Clear the flags by writing back the same value:
    - FLCC_STAT.WRALCOMP: The flag must be set indicating that there is still time for the next write to occur for a Burst Write.
    - FLCC_STAT.CMDBUSY: The flag must be set indicating that the current write operation is still on-going.
    - FLCC_STAT.CMDCOMP: The flag must not be set. This bit would indicate that a command had already completed and therefore the command currently in progress is not the earlier write command (another function has initiated a new command).
    - FLCC_STAT.WRCLOSE: The flag must not be set. If set, the key hole registers are closed and cannot be written. This flag must clear when FLCC_STAT.WRALCOMP is asserted.
- If no further writes are required, wait for FLCC_STAT.CMDCOMP to be set, clear it, and then exit the subroutine.
- Request the next 64-bit dualword write through the key hole access (write FLCC_KH_ADDR, FLCC_KH_DATA0, FLCC_KH_DATA1, and FLCC_CMD registers).
- Check if the write command was accepted (read the FLCC_STAT register to see if any error flags are set).
- Re-enable interrupts.
- Exit the ISR.

If possible, user code with several write accesses to flash must be executed from SRAM to prevent thrashing the flash. Write operations cause ICode reads to stall, resulting in degraded performance when ICode access is required to fetch the next instruction (this may be partially alleviated by the cache controller).

Note that during a BurstWrite, the flash controller overlaps the write operations and therefore any reported write failures (for example, access denied due to write protection) may reflect the status of either of the two overlapped writes. User code must interpret a BurstWrite failure as a failure for both writes being overlapped (that is, both the current and the prior write requests).

DMA Writes

Key hole writes generally require writing four memory-mapped registers per flash write access. An optional address auto-increment feature may reduce the APB traffic required per flash write transaction to three register writes (`FLCC_KH_DATA0`, `FLCC_KH_DATA1`, and `FLCC_CMD`) for all but the first in a series of sequential writes (first write requires setting up the start address). DMA Writes reduces the number of APB transactions to two: every pair of `FLCC_KH_DATA0` and `FLCC_KH_DATA1` registers writes results in a single flash write command executing and the address automatically incrementing to the next DWORD (regardless of the value of auto-increment, DMA writes always increment in the address this manner).

To perform DMA based writes, user code must first configure the DMA controller (a separate peripheral module) for basic access (this DMA mode supports transferring data to/from peripherals, including the flash controller). Once the DMA controller module has been it will sit IDLE until a DMA_REQ signal is asserted by the flash controller.

The user code should manually write `FLCC_KH_ADDR` to setup the initial target address for DMA writes. The DMA controller must be configured to write all output data to `FLCC_KH_DATA1`. Each pair of DMA writes to `FLCC_KH_DATA1` execute a single flash write command and wait for a corresponding delay before the next DMA_REQ is made.

To start the flash controller process of requesting data from the DMA controller, the flash controller must itself be configured for DMA access. To enable DMA mode, user code must set the `FLCC_UCFG.KHDMAEN` bit.

Once the DMA mode is enabled (and the flash controller is IDLE), the flash controller drives the DMA_REQ signal to the DMA module. DMA_REQ is driven at the appropriate time to support BurstWrites (new requests are made once the current write operation is nearly complete).

For more information about DMA functionality, refer to Direct Memory Access (DMA).

Protection and Integrity

Integrity of Info Space

User Code does not have any control over the content of Info Space. If this integrity check fails on a Power-on-Reset, it will be attempted repeatedly until a preset number of attempts has occurred or the integrity check passes (this provides some recoverability from power faults occurring during device power-up). For all other resets (except software reset which does not re-initialize the flash controller), the integrity check is attempted just once.

In the event of an info space integrity check failure, it is expected that the part has failed and will either be disposed of or returned to ADI for failure analysis. If the Info Space integrity check does not pass, the flash controller will enter a special purpose debugging mode. In this mode, User Space protection is automatically asserted and the flash controller local JTAG protection is set to allow JTAG (or other Serial Wire type interface) to interact with the ICode, DCode and APB interfaces.

In this special purpose debugging mode:

- All ICode reads return Bus Errors (code is not executed from the flash memory without passing the integrity check).

- All DCode reads to User Space returns Bus Errors.
- All DCode reads to Info Space except the top 128 bytes are permitted. Reads of the top 128 bytes return Bus Errors.
- All write commands are denied (write attempts set the appropriate error bits in the `FLCC_STAT` register).
- User Space protections may be bypassed only by satisfying the security requirements.

The status of the signature check after reset is read from the `FLCC_STAT.SIGNERR` bit. The status of ECC during signature check is available in the `FLCC_STAT.ECCINFOSIGN` bit. These values are read through a normal JTAG read if the JTAG interface is enabled.

User Space Protection

Two layers of user space protections are provided:

- Access Protection: Protects user space from all read or write operations. This protection mechanism may be manually triggered but is typically automatically asserted in the event of system failure and/or the Serial Wire Debug interface being enabled.
- Write Protection: User facing feature enabling blocks of user space pages to be protected against all write or erase commands. May be set by user at runtime or by ADIs secure bootloader (user would store the desired value in flash for the boot loader to consume during boot).

Access Protection

Access Protection is meant to prevent third parties from reading or tampering user data and program code through JTAG or Serial Wire. Access Protection applies to the entirety of User Space. Access Protection is enabled by one of the following events:

- Serial Wire Debug is enabled
- Flash initialization (info space sign check) fails

The first two enabling mechanisms are automatic features; user code does not have to configure or enable anything for these mechanisms to enable/enforce Access Protection.

While Access Protection is enabled all User Space reads return Bus Errors, writes are denied, and erases are subject to the `FLCC_WRPROT.WORD` bit.

Access Protection may be bypassed by successfully executing a MASSERASE or BLANKCHECK command. Note that the MASSERASE command is disallowed in the event that the `FLCC_WRPROT` (Write Protection) register has been modified from its reset value. BLANKCHECK command is always permitted to execute but will only pass successfully if all of User Space is already in an erased state.

Write Protection

User definable regions of User Space may be configured such that the flash controller will deny any attempts to modify them (this affects both WRITE and ERASE commands). Write Protection may be configured at runtime or may be stored in User Space meta-data to be loaded by the ADI secure bootloader during device boot.

NOTE: MASSERASE command is disallowed if any of the bits in the FLCC_WRPROT register have been modified from the default value.

Runtime Configuration

Write Protection is configured by modifying the FLCC_WRPROT memory mapped register. FLCC_WRPROT.WORD is a 32-bit wide bit field representing the Write Protection state for 32 similar size blocks of User Space pages. The flash memory is divided into 128 pages of User Space storage. For Write Protection these are logically divided into 32 blocks of four pages each. Write Protection is independently controlled for each of these 32 blocks; each bit of FLCC_WRPROT.WORD controls the protection mechanism for a unique block of four pages of User Space. The least significant bits of FLCC_WRPROT.WORD correspond to the least significant pages of User Space.

The bits in the FLCC_WRPROT register are active low. 0 represents active Write Protection and 1 represents no protection for the corresponding block of pages. The FLCC_WRPROT register is sticky at 0. Once protection is enabled, it cannot be disabled without resetting the device.

User code may assert write protection for any block of pages by clearing the appropriate bit of FLCC_WRPROT.WORD at any time. It is advisable to assert write protection as early as possible in user code; it is also advisable for the user to write-protect block zero (that is, flash pages 0-3) and to place user boot and integrity checking code in this block. In this manner, the user can fully control the write protection scheme without relying on ADI bootloader to setup the FLCC_WRPROT register.

Meta Data Configuration

The most significant page of User Space contains a single 32-bit field representing a set of 1-bit Write Protect flags for each of these 32 logical blocks (see User Space Meta Data), matching the functionality of the FLCC_WRPROT register.

These Write Protection bits are read from the flash by ADIs secure bootloader and stored in the FLCC_WRPROT register after a reset operation. The default (erased) state of flash memory is all 1s, therefore the default FLCC_WRPROT register value is to disable protection for all pages in User Space. Each bit of FLCC_WRPROT.WORD corresponds to the protection state for one block of four User Space pages.

User code may clear bits in the WrProt meta-data word at runtime, or this word may be included in the initial upload of user data and program code. Writing the WrProt meta-data word at runtime does not immediately affect the Write Protection state; if immediate protection is required, user code should also write the same values to the FLCC_WRPROT register. When writing the WrProt meta-data the user should consider including Write Protection of the most significant page (thus protecting the meta-data from a page erase or other modification). As with all flash locations, repeated writes (without intermediate erasures) are discouraged and will likely cause ECC errors during read-back.

Once protection has been enabled (the corresponding bit has been cleared in FLCC_WRPROT.WORD) it cannot be disabled without resetting the device. For this reason, once Write Protection has been enabled for any block of pages through the User Space meta-data, it cannot be disabled without erasing the most significant page of User Space (the page hosting the relevant meta-data) or otherwise affecting the flow of ADIs secure bootloader.

The following sequence outlines the process of programming the Write Protection meta-data word in flash:

- Ensure the most significant page of user space has been erased since the last time the meta-data was written (if ever).
- Write the desired value for the WrProt meta-data word directly to flash (write '0' to a particular bit to enable protection for the corresponding block). In 256 KB flash, this word is available at the address 0x3FFF0.
- Verify that the write completed successfully by polling the status register.
- Reset the device. The WrProt bits will automatically be uploaded by the ADI secure bootloader from user space to the `FLCC_WRPROT` register and used to enforce the protection scheme.

Signatures

Signatures are used to check the integrity of the flash device contents. Signature calculations do not include the ECC portion of the flash data bus nor the most significant word read from the set of pages being signed (this word is considered meta data and is meant to hold the expected signature value for the given set of pages).

The flash controller implements its own stand-alone CRC engine for generating and verifying signatures. Note however that this implementation matches the CRC Accelerator peripheral (with an intial value of 0xFFFF_FFFF). For more information about the CRC algorithm used, refer to Cyclic Redundancy Check (CRC).

Signatures may be included in the initial upload of user data and program code (generated before being uploaded), or may be generated and stored to flash at runtime. Generation at runtime may utilize either the CRC Accelerator or may call on the flash controllers signature generation logic.

ECC bytes correspond to 64-bit DWORDS in the flash memory, therefore if ECC is enabled the most significant 64 bits (including the 32-bit signature word) must be written all at once (else the ECC byte gets corrupted by the second write). If using the flash controller to generate the signature value, leave the unused 32 bits paired with the signature word in their erased state (0xFFFF_FFFF). Else, it may result in spurious ECC errors after device reset (depending on `FLCC_WRPROT` configuration) and the device may become unusable.

The Sign command generates a signature for all data from the start page to the end page (excluding the signature meta-data word). User code must define the start and end pages by writing `FLCC_PAGE_ADDR0.VALUE` and `FLCC_PAGE_ADDR1.VALUE` respectively.

The following procedure must be followed to generate or verify a signature:

- Write to `FLCC_PAGE_ADDR0.VALUE`: Start address of a contiguous set of pages (if out of bounds, the command is denied).
- Write to `FLCC_PAGE_ADDR1.VALUE`: End address of a contiguous set of pages (if out of bounds, the command is denied).
- Write to `FLCC_KEY.VALUE`: Write the User Key value (0x676C7565) to the `FLCC_KEY` register.
- Write to `FLCC_CMD.VALUE`: Write the SIGN command (0x3) to the `FLCC_CMD` register.
- WAIT: When the command is completed, `FLCC_STAT.CMDCOMP` bit is set.

- If using the flash controller to generate a signature to write into the flash meta-data, the signature value may be read from the `FLCC_SIGNATURE` register.
- The generated signature is automatically compared with the data stored in the most significant 32-bit word of the region being signed. If the generated result does not match the stored value, a fail is reported in the `FLCC_STAT` register by asserting verify error in the `FLCC_STAT.CMDFAIL` bit field (0x2).

While the signature is being computed all other accesses to the flash are stalled. Generating/verifying the signature for a 256 KB block (full User Space) results in a stall duration of 32 KB flash reads (64 bits per read operation) or approximately 64k HCLK periods.

NOTE: `FLCC_PAGE_ADDR0.VALUE` and `FLCC_PAGE_ADDR1.VALUE` addresses may be written as byte addresses but are consumed by the flash controller as page addresses (the lower 10 address bits are ignored). SIGN command is denied if `FLCC_PAGE_ADDR1.VALUE` is less than `FLCC_PAGE_ADDR0.VALUE`. Signatures are always performed at a page-level granularity over continuous address ranges.

Key Register

To prevent spurious and potentially damaging flash accesses, some commands and registers are key protected. The User Key is not a security element and is not meant to be a secret. Instead, this key protects users from negative consequences of software bugs (especially during early software development).

UserKey

This key serves to prevent accidental access to some flash features and addresses. The key value is 0x676C_7565. This key must be entered to run protected user commands (ERASEPAGE, SIGN, MASSERASE, and ABORT), or to enable write access to the `FLCC_UCFG` or `FLCC_TIME_PARAM0` and `FLCC_TIME_PARAM1` registers. Once entered, the key remains valid until an incorrect value is written to the key register, or a command is written to the `FLCC_CMD` register when any command is requested this key is automatically cleared. If this key is entered to enable write access to the `FLCC_UCFG` or `FLCC_TIME_PARAM0` and `FLCC_TIME_PARAM1` registers then it is recommended to clear the key immediately after updating the register(s).

ECC

The flash controller provides ECC based error detection and correction for flash reads. ECC is enabled by default for Info Space, and thus provides assurance that flash initialization functions work properly (info space signature check unconditionally takes ECC into account).

The flash controller uses an 8-bit Hamming modified code to correct 1-bit errors or detect 2-bit errors for any dual-word (64-bit) flash data access.

When enabled, the ECC engine is active during signature checks (refer to Signatures section). User code may request a signature check of the entirety of User Space then check the `FLCC_STAT.ECCERRCMD` bit to determine if any single or dual bit data corruptions are present in User Space.

Defaults and Configuration

In User Space, ECC is enabled by default.

ECC can be disabled by the user code. It provides extra security. However, it is not recommended to disable ECC, as it does not provide any power or performance enhancement.

Disabling ECC requires resetting the `FLCC_ECC_CFG.EN` bit. When enabled, ECC may apply to the entirety of User Space or may be configured to apply only to a limited range. A single page address pointer (`FLCC_ECC_CFG.PTR`) is used to define the start address for ECC; all flash addresses from the start page through the top of User Space (inclusive) will have ECC enabled when `FLCC_ECC_CFG.EN` is set.

ECC errors may be optionally reported as Bus Errors for ICode or DCode reads, or may generate interrupts. Independent error reporting options are available for 1-bit corrections and 2-bit error detections by writing to the `FLCC_IEN.ECC_ERROR` field.

Error Handling

The impact of ECC errors during the information space signature check is described in Signatures. On any read operation, if the ECC engine observes a 1 bit error it will be corrected automatically (the 1-bit error could be in the ECC byte itself, or in the 64-bit dualword being read by the user). If a 2-bit error is observed the ECC engine can only report the detection event (2-bit errors cannot be corrected).

Depending on when the read happens (for example, during an ICode or DCode read, or as part of a built-in command such as a signature check) appropriate flags are set in the status register (`FLCC_STAT.ECCERRCMD`, `FLCC_STAT.ECCRDERR`, `FLCC_STAT.ECCINFOSIGN`, and so on).

If interrupt generation is enabled in the `FLCC_IEN.ECC_ERROR` field, the source address of the ECC error causing the interrupt are available in the `FLCC_ECC_ADDR` register for the interrupt service routine to read.

ECC Errors during Sign Command Execution

ECC errors observed during signature checks generate the appropriate status register flags after completion but will not populate the `FLCC_ECC_ADDR` register.

Concurrent Errors

If ECC errors occur on DCODE and ICODE simultaneously (for example, from an ICODE prefetch match and a DCODE flash read), ECC error status information is prioritized as follows:

- First Priority: 2-bit ECC errors are given priority over 1-bit ECC errors/corrections.

  For example, if a 2-bit ECC error is observed on a DCODE read in the same cycle as a 1-bit ecc error/correction on an ICODE read, the ECC error status is updated for DCODE only.

- Second Priority: ICODE is given priority over DCODE.

  For example, if a 2-bit error is observed on an ICODE read and a DCODE read in the same cycle, the ECC error status is updated for ICODE only.

Read of Erased Location

When erased, the flash memory holds a value of all 1s, including the ECC byte appended to every 64-bit DWORD. The proper ECC meta-data for 64 ONES is not 0xFF, therefore in its erased state the flash memory holds data and ECC meta-data representing some number of bit errors.

For this reason any flash reads of erased locations will automatically bypass the ECC engine. If user code reads a location with all 1s in both 64-bit DWORD and ECC byte, the read returns the data without indicating any ECC errors.

Clock and Timings

The flash controller is preconfigured to provide safe timing parameters for all flash operations for core clock frequencies of 26 MHz or less and reference clock frequency of 13 MHz (+/- 10%).

If the reference clock is not within 10% of 13 MHz, user code should adjust the timing parameters accordingly.

Flash Operating Modes

The flash memory used by the ADuCM302x MCU supports the following power optimizing features:

Sleep Mode

The user code may put the flash IP into a low power sleep mode by writing the SLEEP command (0x2) to the `FLCC_CMD` register. The flash controller wakes the flash IP from sleep automatically on the first flash access following a SLEEP command. The user code may observe the sleep state of flash by reading the `FLCC_STAT.SLEEPING` bit.

NOTE: The cache controller, DMA reads, other peripheral, or user code may attempt to read flash memory at any time, any of which will trigger a flash wakeup event; it is advisable that user code occasionally poll the `FLCC_STAT.SLEEPING` bit to verify that the flash IP is still sleeping when the user expects it to be.

The flash controller will not honor any new commands (WRITE, ERASE, and do on) while the flash IP is in sleep mode; the only supported commands in this mode are IDLE and ABORT. DMA write requests will be stalled automatically by entering sleep mode.

System Interrupt based aborts (as configured through the `FLCC_ABORT_EN_LO` and `FLCC_ABORT_EN_HI` registers) are generally used to abort any ongoing flash commands in the event of an enabled system interrupt. However, such an interrupt will not wake the flash from sleep mode. If the system interrupt can be serviced with accessing the flash, it will remain in sleep mode; if servicing the interrupt requires accessing the flash then the flash access itself will serve to wake the flash IP.

Waking from sleep incurs a ~5 µs latency before executing any reads or commands (this is a requirement of the flash IP). User code may wake the flash IP early by executing an IDLE command. This will wake the flash IP without any other effect on the controller.

For consistency, the ABORT command can also be used to wake the flash IP. Waking with the ABORT command differs from waking by the IDLE command only in that the status register will report `FLCC_STAT.CMDFAIL` bit (0x3) to match the expected user code checks for status register values.

Power-down Mode

The ADuCM302x MCU automatically powers down the flash IP when the device hibernates. To support this feature, the flash controller interoperates with the power management unit and delays hibernate until any ongoing flash accesses are completed. User code is responsible for reading and evaluating flash status registers prior to entering hibernate (status registers are not retained in hibernate). The abort command may be used to abruptly end an ongoing flash command but must be used sparingly to avoid eventual damage to the flash array.

Clock Gating

A series of clock gates have been inserted into the flash controller to automatically gate off unused components of the module. User configuration or control is not required. Unused portions of the flash controller is automatically gated off when appropriate (for example, while in sleep mode the majority of the flash controller is gated off to save power).

Flash Interrupts and Exceptions

The flash controller may selectively generate interrupts for many events. The following table outlines the events that may generate interrupts and bit fields of `FLCC_IEN` used to control interrupt generation for each event.

Table 8-2: Interrupts and Bit fields

Name (Bit field)	Description
CMDFAIL	IRQ generated when a command or write operation completes with an error status.
WRALCOMP	IRQ generated when an active flash write is nearly complete and the key-hole registers are open for another write (if fulfilled in time a BurstWrite will occur).
CMDCMPLT	IRQ generated when a command or flash write operation completes.
ECC_ERROR	IRQ generated when 2-bit ECC errors are observed when this field is set to 2.

Flash Programming Model

The following section provides an example sequence to execute the Page erase command using the flash controller. The same sequence can be used for other commands with some modifications.

Programming Guidelines

Here are the programming guidelines:

1. Program the `FLCC_PAGE_ADDR0` or `FLCC_PAGE_ADDR1` register with the address of the page which needs to be erased.

2. Write the Flash User Key (0x676C7565) to the FLCC_KEY.VALUE bits.
3. Write the command to be executed into the FLCC_CMD register.
4. Poll for the FLCC_STAT.CMDCOMP bit to be set.

ADuCM302x FLCC Register Descriptions

Flash Controller (FLCC) contains the following registers.

Table 8-3: ADuCM302x FLCC Register List

Name	Description
FLCC_ABORT_EN_HI	IRQ Abort Enable (Upper Bits)
FLCC_ABORT_EN_LO	IRQ Abort Enable (Lower Bits)
FLCC_POR_SEC	Flash Security
FLCC_VOL_CFG	Volatile Flash Configuration
FLCC_CMD	Command
FLCC_ECC_ADDR	ECC Status (Address)
FLCC_ECC_CFG	ECC Configuration
FLCC_IEN	Interrupt Enable
FLCC_KEY	Key
FLCC_KH_ADDR	Write Address
FLCC_KH_DATA0	Write Lower Data
FLCC_KH_DATA1	Write Upper Data
FLCC_PAGE_ADDR0	Lower Page Address
FLCC_PAGE_ADDR1	Upper Page Address
FLCC_SIGNATURE	Signature
FLCC_STAT	Status
FLCC_TIME_PARAM0	Time Parameter 0
FLCC_TIME_PARAM1	Time Parameter 1
FLCC_UCFG	User Configuration
FLCC_WRPROT	Write Protection
FLCC_WR_ABORT_ADDR	Write Abort Address

IRQ Abort Enable (Upper Bits)

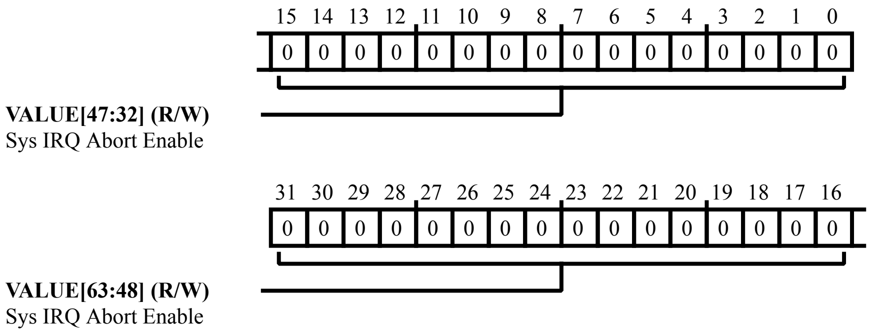


Figure 8-6: FLCC_ABORT_EN_HI Register Diagram

Table 8-4: FLCC_ABORT_EN_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Sys IRQ Abort Enable. To allow a system interrupt to abort an ongoing flash command (for example, Erase, Write, Sign, and so on) write a '1' to the bit in this register corresponding with the desired system IRQ number.

IRQ Abort Enable (Lower Bits)

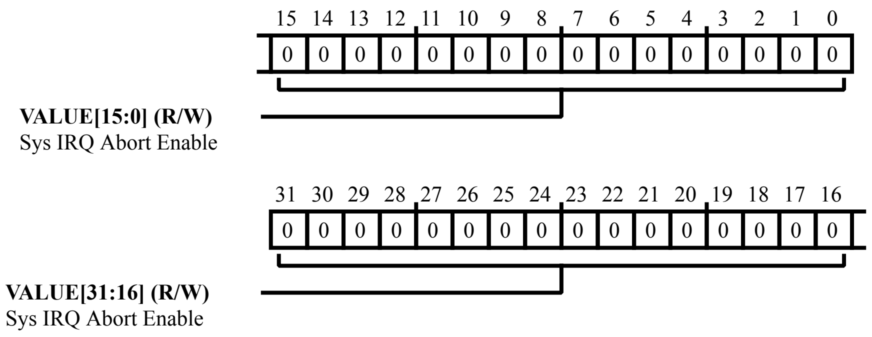


Figure 8-7: FLCC_ABORT_EN_LO Register Diagram

Table 8-5: FLCC_ABORT_EN_LO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Sys IRQ Abort Enable. To allow a system interrupt to abort an ongoing flash command (for example, Erase, Write, Sign, and so on) write a '1' to the bit in this register corresponding with the desired system IRQ number.

Flash Security

Controls the flash security features.

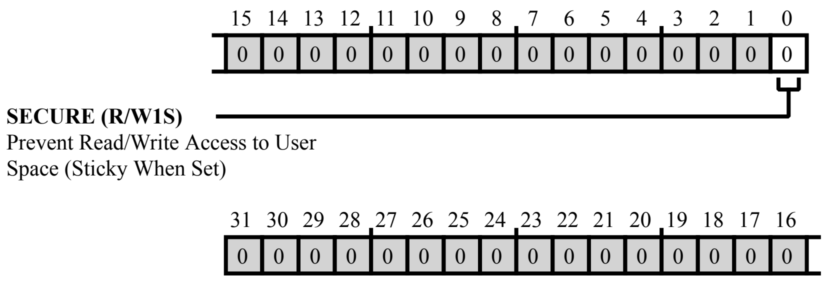


Figure 8-8: FLCC_POR_SEC Register Diagram

Table 8-6: FLCC_POR_SEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W1S)	SECURE	Prevent Read/Write Access to User Space (Sticky When Set). Requires User key. Once set it cannot be cleared without resetting the device (POR or PIN reset only). It plays a direct role in User Space security enforcement. Once set, this bit prevents access to User Space: DCode Reads: Return bus faults with the data bus == 0 ICode Reads: Return bus faults with the data bus == 0 APB Writes: command will be denied, flash content unchanged. When set, user code may still perform MassErase and PageErase operations. Note: WRPROT register still applies. Only unprotected pages may be erased (Mass Erase is disallowed if any pages are protected).

Volatile Flash Configuration

This register resets on any/all resets (POR/PIN/SW/WDT). The User Key must be written to the KEY Register prior to writing this register.

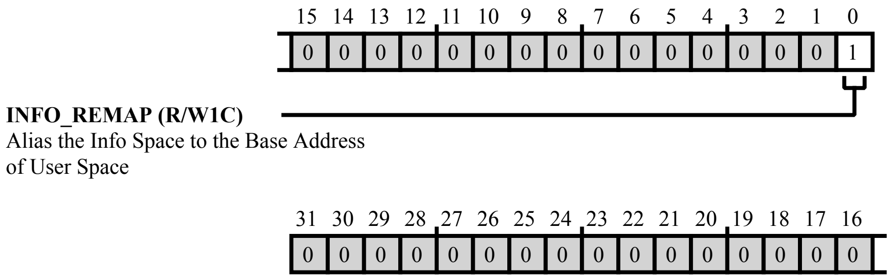


Figure 8-9: FLCC_VOL_CFG Register Diagram

Table 8-7: FLCC_VOL_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W1C)	INFO_REMAP	Alias the Info Space to the Base Address of User Space. When this bit is set, the boot loader is mirrored at address 0. This bit is automatically cleared by the boot loader.

Command

Write this register to execute a specified command. The user key must first be written to the FLCC_KEY register for most command requests to be honored (see details below).

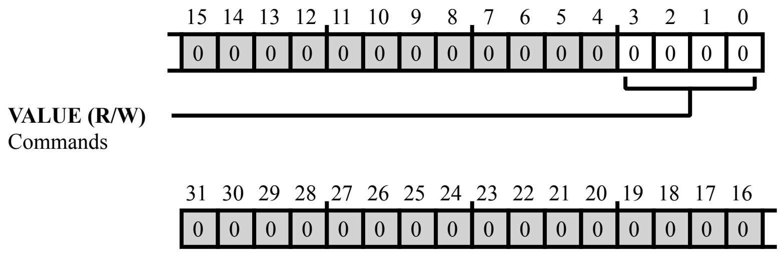


Figure 8-10: FLCC_CMD Register Diagram

Table 8-8: FLCC_CMD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0 (R/W)	VALUE	Commands. Write command values to this register to begin a specific operation. All commands but WRITE and IDLE require first writing the User Key to the FLCC_KEY register	
		0	IDLE No key is required. No command executed; if Flash IP was in the SLEEP state, this command gracefully wakes the flash (returns command complete and no error codes)

Table 8-8: FLCC_CMD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		1	ABORT User key is required. The ABORT command should be used sparingly as a last resort mechanism to gain access to the flash IP during time-sensitive events. For example a low voltage alarm may be cause to abort an ongoing flash write or erase command to enable user code to shutdown the part gracefully. Note that the flash array may be damaged by over-use of the ABORT command. If this command is issued then any command currently in progress will be abruptly stopped (if possible). The status will indicate command completed with an error of ABORT. ABORT is the only command that can be issued while another command is already in progress (with one exception: user code may stack one WRITE command on top of a single on-going WRITE command; all other overlapping command combinations are invalid unless the new command is an ABORT). If a write or erase is aborted then some flash IP timing requirements may be violated and it is not possible to determine if the write or erase completed successfully. User code should read the affected locations to determine the outcome of the aborted commands. Note that an aborted command may result in a weakly programmed flash: it is always advisable to erase the affected region and reprogram it. Depending on how far along the flash controller is in the process of performing a command, an ABORT is not always possible (some flash IP timing parameters must not be violated). This is difficult (if not impossible) to predict in software, therefore ABORTS should be considered a request which may have no affect on actual command duration.

Table 8-8: FLCC_CMD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		2	Requests flash to enter Sleep mode User Key is required. Requests the flash controller to put the flash to sleep (a low power mode). When sleeping, any ICode, DCode, or DMA transaction will wake the flash automatically. Wake-up process takes ~5us (configurable in FLCC_TIME_PARAM1 register). If user code can predict ~5us ahead of time that the flash will be required, user may write an IDLE command to the FLCC_CMD register to manually wake the flash. An ABORT command is also respected for waking the part (and will return appropriate status indicating that the sleep command was aborted). Once awoken for any reason, the part will remain awake until user code once again asserts a SLEEP command.
		3	SIGN User key is required. Use this command to generate a signature for a block of data. Signatures may be generated for blocks of whole pages only. The address of the start page should be written to the FLCC_PAGE_ADDR0 register, the address of the end page written to the FLCC_PAGE_ADDR1 register, and then write this code to the FLCC_CMD register to start the signature generation. When the command has complete the signature will be readable from the FLCC_SIGNATURE register. More information on this command is in 'signature' section
		4	WRITE No key is required. This command takes the address and data from the FLCC_KH_ADDR, FLCC_KH_DATA0, and FLCC_KH_DATA1 registers and executes a single 64-bit WRITE operation targeting the specified address. More information can be found in 'writing to flash' and 'write protection' section.

Table 8-8: FLCC_CMD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		5	Checks all of User Space; fails if any bits in user space are cleared User Key is required. Performs a blank check on all of user space. If any bits in user space are cleared the command will fail with a READ VERIFY status. If all of user space is FFs the command will pass. This command is intended to support early customer software development. When an unprogrammed part boots with security features preventing reads and writes of user space, this command may be used to verify that the user space contains no proprietary information. If this command passes read and write protection of user space will be cleared.
		6	ERASEPAGE User key is required. Write the address of the page to be erased to the `FLCC_PAGE_ADDR0` register, then write this code to the `FLCC_CMD` register. When the erase has completed the full page will be verified (read) automatically to ensure a complete erasure. If there is a read verify error this will be indicated in the `FLCC_STAT` register. To erase multiple pages wait until a previous page erase has completed check the status then issue a command to start the next page erase.
		7	MASSERASE User key is required. Erase all of flash user space. When the erase has completed the full user space will be verified (read) automatically to ensure a complete erasure. If there is a read verify error this will be indicated in the Status register.

ECC Status (Address)

This register is updated on ECC errors or corrections as selected to generate interrupts (IRQ) in the FLCC_IEN register. this register is not updated in the event of an ECC error or correction which instead generates a bus fault.

This register records the address of the first ECC error or correction event to generate an interrupt since the last time the ECC status bits were cleared (or since reset).

If the status bits are cleared in the same cycle as a new ECC event (selected to generate an IRQ), a new address will be recorded and the status bits will remain set.

Errors have priority over corrections (2 or more bits corrupt = ERROR; a correction results in proper data being returned after a single bit is corrected). If an error and a correction occur in the same cycle, this register will report the ERROR address.

When two of the same priority ECC events occur (both ERROR or both CORRECTION) the ICODE bus has priority over DCODE. Therefore if both ICODE and DCODE buses generate the same type of ECC event in the same cycle, the ICODE address will be stored in this register.

The register cannot be cleared except by reset; it will always hold the address of the most recently reported ECC correction or error.

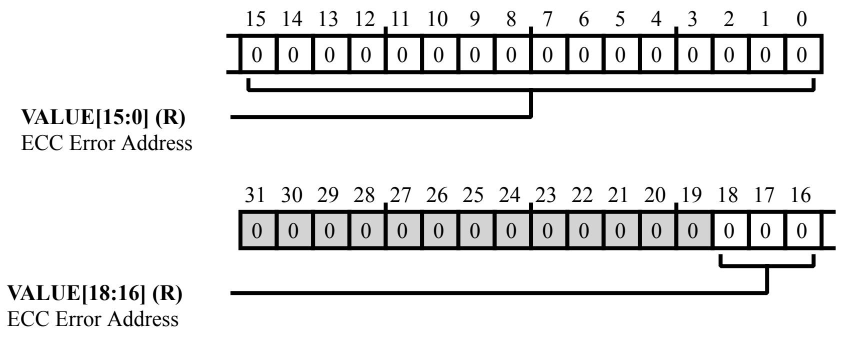


Figure 8-11: FLCC_ECC_ADDR Register Diagram

Table 8-9: FLCC_ECC_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18:0 (R/NW)	VALUE	ECC Error Address. This register has the address for which ECC error is detected.

ECC Configuration

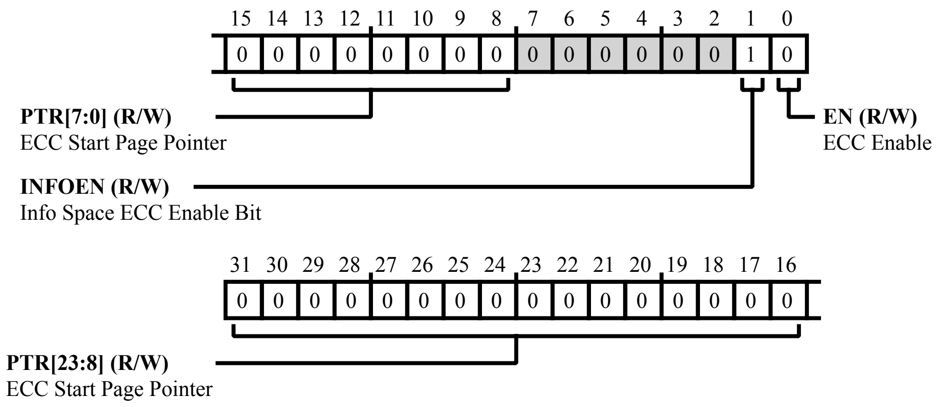


Figure 8-12: FLCC_ECC_CFG Register Diagram

Table 8-10: FLCC_ECC_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:8 (R/W)	PTR	ECC Start Page Pointer. ECC start page pointer (user should write bits [31:8] of the start page address into bits [31:8] of this register). A byte-address for any page in flash User Space. The bottom bits of this address will be ignored by the flash controller, forming a Page Address. When ECC is enabled and user code reads any address from within the page specified or any more significant page, ECC functions will be performed. Reads from less significant pages will bypass ECC entirely.
1 (R/W)	INFOEN	Info Space ECC Enable Bit. ECC is enabled by default for Info Space; clearing this bit disables ECC in info space. this bit is not key protected.
0 (R/W)	EN	ECC Enable. Set this bit to enable ECC on User Space. ECC will be enabled on all future flash reads in user space from any address between `FLCC_ECC_CFG.PTR` through the top of User Space (inclusive). When cleared (or accessing addresses outside the enabled range), the flash controller will return the raw data in response to both ICode and DCode reads of User Space; no error corrections will be made or reported.

Interrupt Enable

Used to specify when interrupts will be generated.

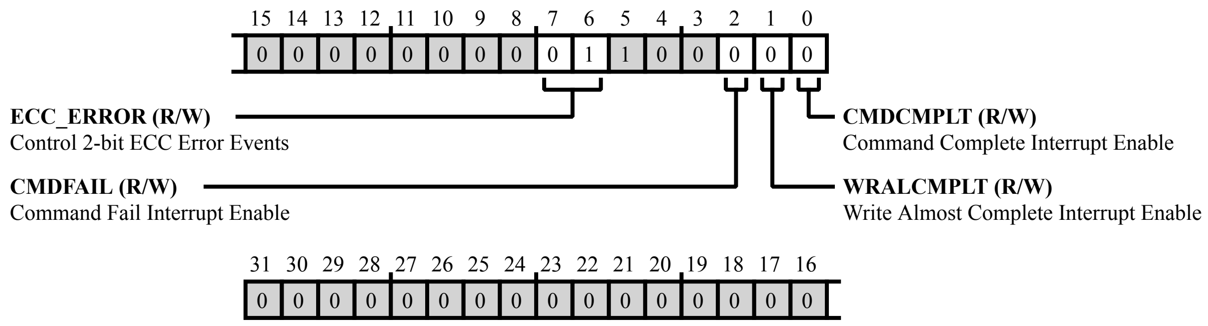


Figure 8-13: FLCC_IEN Register Diagram

Table 8-11: FLCC_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/W)	ECC_ERROR	Control 2-bit ECC Error Events. Control whether to generate bus errors, interrupts, or neither in response to 2-bit ECC Error events.	
		0	Do not generate a response to ECC events
		1	Generate Bus Errors in response to ECC events
		2	Generate IRQs in response to ECC events
2 (R/W)	CMDFAIL	Command Fail Interrupt Enable. If this bit is set then an interrupt will be generated when a command or flash write completes with an error status.	
1 (R/W)	WRALCMPLT	Write Almost Complete Interrupt Enable.	
0 (R/W)	CMDCMPLT	Command Complete Interrupt Enable. When set, an interrupt will be generated when a command or flash write completes.	

Key

When user code must write a key to access protected features, the key value must be written to this register.

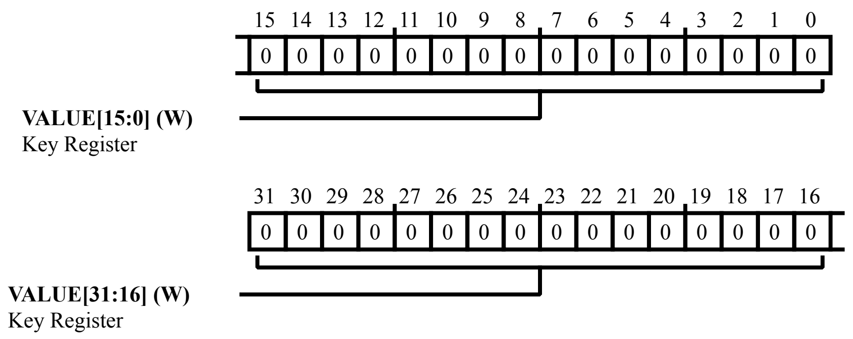


Figure 8-14: FLCC_KEY Register Diagram

Table 8-12: FLCC_KEY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:0 (RX/W)	VALUE	Key Register. Unlock protected features by writing the appropriate key value to this register	
		1735161189	USERKEY Write this field with hex value 0x676C7565 to enable certain registers to be modified or to allow certain commands to be executed. This key is used as a sanity check to prevent accidental modification of settings or flash content. It is not a security component and is not intended to be secret information.

Write Address

Write the byte-address of any byte of a 64-bit dual-word flash location to be targeted by a WRITE command.

All writes target 64-bit dual-word elements in the flash array. User code may byte-mask data to emulate byte, hword, or word writes. Flash IP specifications warn that no location should be written more than twice between erasures. When writing a location more than once, user should be aware that ECC meta-data cannot be updated appropriately; user code should disable ECC for the relevant region of flash.

(Writing any address above the valid range of flash memory will saturate the address to prevent aliasing; user code should take care to target valid flash address locations)

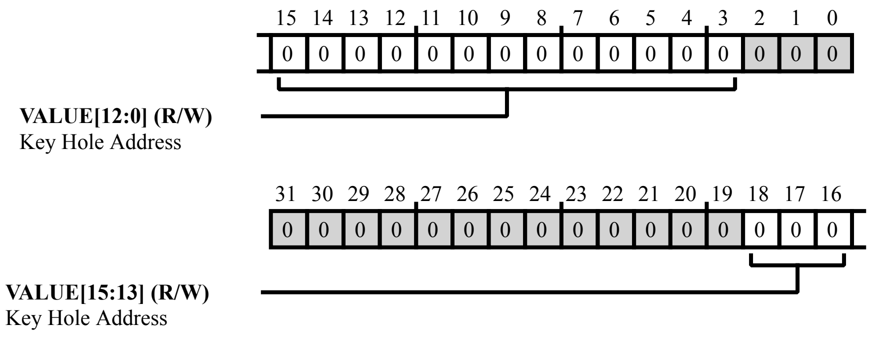


Figure 8-15: FLCC_KH_ADDR Register Diagram

Table 8-13: FLCC_KH_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18:3 (R/W)	VALUE	Key Hole Address. Address to be written on a WRITE command.

Write Lower Data

The lower half of 64-bit dualword data to be written to flash

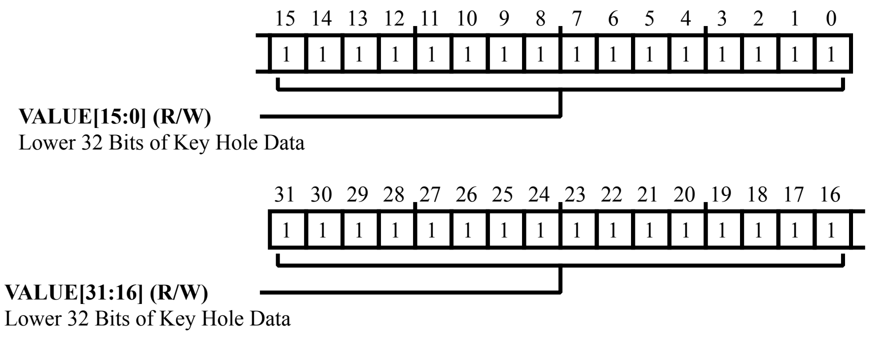


Figure 8-16: FLCC_KH_DATA0 Register Diagram

Table 8-14: FLCC_KH_DATA0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Lower 32 Bits of Key Hole Data. Lower half of 64-bit dualword data to be written on a WRITE command.

Write Upper Data

The lower half of 64-bit dualword data to be written to flash.

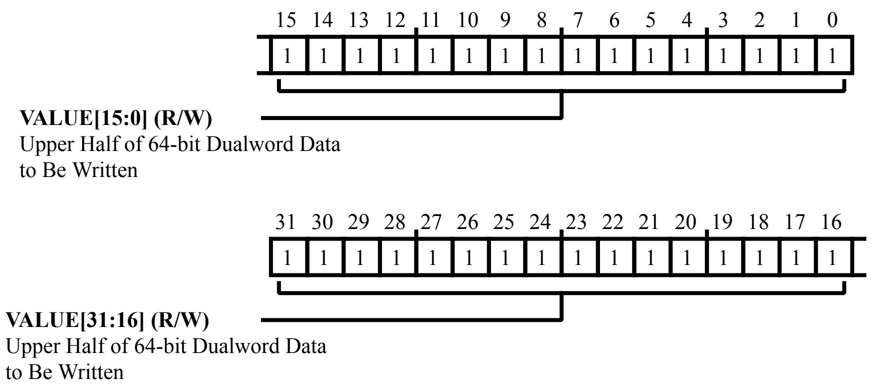


Figure 8-17: FLCC_KH_DATA1 Register Diagram

Table 8-15: FLCC_KH_DATA1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Upper Half of 64-bit Dualword Data to Be Written. Upper Half of 64-bit Dualword Data to be Written on a WRITE Command. If DMA is enabled, then this register acts as a FIFO. Writes to this register will push the old data to lower 32 bit of 64 bit data (FLCC_KH_DATA0). When this register is written twice (in DMA mode) the FIFO becomes full and a flash write command is automatically be executed.

Lower Page Address

Write a byte-address to this register to select the page in which that byte exists.

The selected page may be used for a ERASEPAGE command (selecting which page to erase) or for a SIGN command (selecting the start page for a block on which a signature should be calculated).

For commands using both `FLCC_PAGE_ADDR0` and `FLCC_PAGE_ADDR1`, user should ensure that `FLCC_PAGE_ADDR0` is always less than or equal to `FLCC_PAGE_ADDR1`, else the command is denied.

Writing any address above the valid range of flash memory saturates the address register to prevent aliasing in the flash memory space.

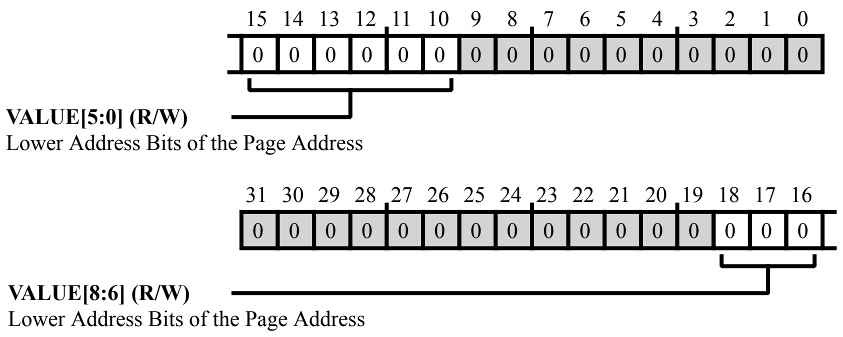


Figure 8-18: FLCC_PAGE_ADDR0 Register Diagram

Table 8-16: FLCC_PAGE_ADDR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18:10 (R/W)	VALUE	Lower Address Bits of the Page Address.

Upper Page Address

Write a byte-address to this register to select the page in which that byte exists.

The selected page may be used for a SIGN command (selecting the end page for a block on which a signature should be calculated).

For commands using both `FLCC_PAGE_ADDR0` and `FLCC_PAGE_ADDR1`, user should ensure that `FLCC_PAGE_ADDR0` is always less than or equal to `FLCC_PAGE_ADDR1`, else the command is denied.

Writing any address above the valid range of flash memory saturates the address register to prevent aliasing in the flash memory space.

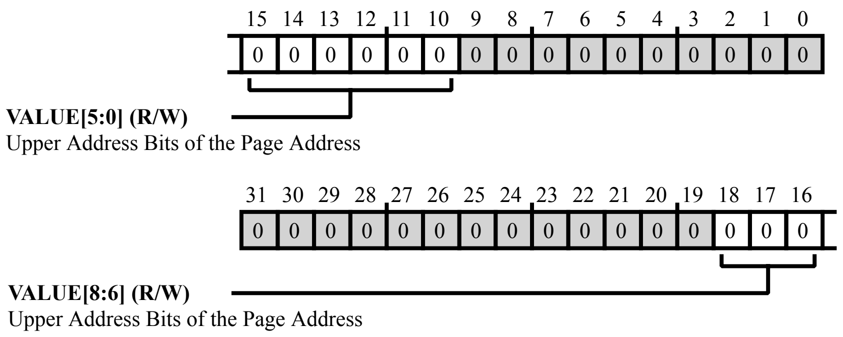


Figure 8-19: FLCC_PAGE_ADDR1 Register Diagram

Table 8-17: FLCC_PAGE_ADDR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18:10 (R/W)	VALUE	Upper Address Bits of the Page Address.

Signature

Provides read access to the most recently generated signature.

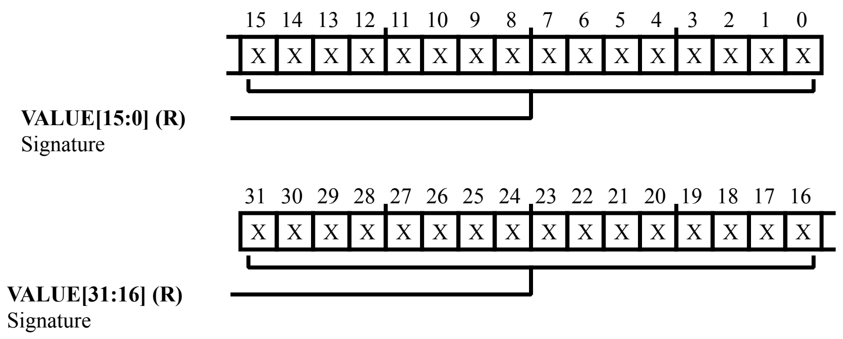


Figure 8-20: FLCC_SIGNATURE Register Diagram

Table 8-18: FLCC_SIGNATURE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Signature. Provides read access to the most recently generated signature.

Status

Provides information on current command states and error detection/correction.

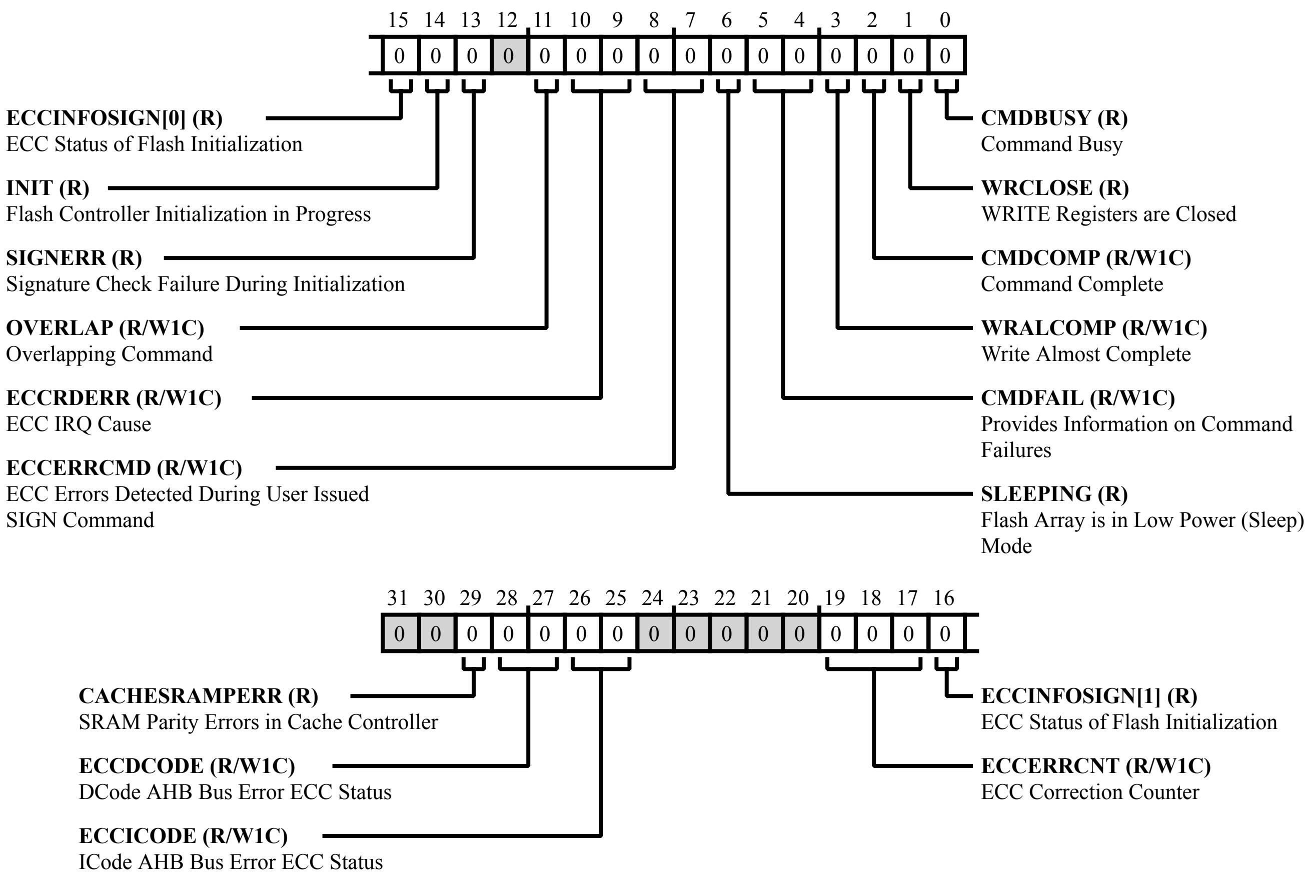


Figure 8-21: FLCC_STAT Register Diagram

Table 8-19: FLCC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/NW)	CACHESRAMPERR	SRAM Parity Errors in Cache Controller. This register provides details for AHB Errors generated due to cache SRAM parity error on the ICode bus.

Table 8-19: FLCC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
28:27 (R/W1C)	ECCDCODE	DCode AHB Bus Error ECC Status. Provides details for AHB Bus Errors generated due to ECC errors and/or corrections on the DCODE bus.	
		0	No Error No errors or corrections reported since reset or the register was last cleared
		1	2-bit Error 2-bit ECC error has been detected and reported on AHB read access
		2	1-bit Correction 1-bit ECC correction has been detected and reported on AHB read access
26:25 (R/W1C)	ECCICODE	ICode AHB Bus Error ECC Status. Provides details for AHB Bus Errors generated due to ECC errors and/or corrections on the ICODE bus.	
		0	No Error No errors or corrections reported since reset or the register was last cleared
		1	2-bit Error 2-bit ECC error has been detected and reported on AHB read access
		2	1-bit Correction 1-bit ECC correction has been detected and reported on AHB read access
19:17 (R/W1C)	ECCERRCNT	ECC Correction Counter. This counter keeps track of overlapping ECC 1-bit correction reports. When configured to generate IRQs or AHB Bus Errors in the event of an ECC correction event, this field counts the number of ECC corrections that occur after the first reported correction. The counter remains at full scale when it overflows and clears automatically when clearing either `FLCC_STAT.ECCICODE` or `FLCC_STAT.ECCDCODE` status bits.	
16:15 (R/NW)	ECCINFOSIGN	ECC Status of Flash Initialization. ECC status after the end of automatic signature check on Info space.	
		0	No Error No errors reported.
		1	2-bit error 1 or more 2-bit ECC Errors detected during signature check (signature check has failed)
		2	1-bit error 1 or more 1-bit ECC Corrections performed during signature check. (signature check will pass if checksum still matches)
		3	1- and 2-bit error At least one of each ECC event (1-bit Correction and 2-bit Error) were detected during signature check (signature check will fail).

Table 8-19: FLCC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/NW)	INIT	Flash Controller Initialization in Progress. Flash controller initialization is in progress. Until this bit de-asserts AHB accesses will stall and APB commands will be ignored.	
13 (R/NW)	SIGNERR	Signature Check Failure During Initialization. Indicates an automatic signature check has failed during flash controller initialization. The register value is valid only after the signature check has completed.	
11 (R/W1C)	OVERLAP	Overlapping Command. This bit is set when a command is requested while another command is busy (overlapping commands will be ignored)	
10:9 (R/W1C)	ECCRDERR	ECC IRQ Cause. This field reports the cause of recently generated interrupts. The controller may be configured to generate interrupts for 1- or 2-bit ECC events by writing the appropriate values to `FLCC_IEN.ECC_ERROR` and `FLCC_IEN.ECC_ERROR`. These bits are sticky high until cleared by user code.	
		0	No Error
		1	2-bit Error ECC engine detected a non-correctable 2-bit error during AHB read access
		2	1-bit Correction ECC engine corrected a 1-bit error during AHB read access
		3	1- and 2-bit Events ECC engine detected both 1- and 2-bit data corruptions which triggered IRQs (note that a single read can only report one type of event; this status indicates that a subsequent AHB read access incurred the alternate ECC error event) By default 1-bit ECC corrections are reported as IRQs and 2-bit ECC errors are reported as bus faults. It is not recommended to report both types as IRQs else the status bits become ambiguous when trying to diagnose which fault came first.

Table 8-19: FLCC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8:7 (R/W1C)	ECCERRCMD	ECC Errors Detected During User Issued SIGN Command. ECC errors, if produced during signature commands, are reported by these bits. To generate interrupts base on these bits user code should set the corresponding bits in FLCC_IEN register.	
		0	success no error, successful flash read operation during signature check
		1	2-bit error During signature commands, 2 bit error is detected on one or more flash locations, not corrected.
		2	1-bit error 1 bit error is corrected for one or more flash locations while doing signature commands
		3	1 or 2 bit error During signature commands, 1 bit error and 2 bit errors are detected on one or more flash locations
6 (R/NW)	SLEEPING	Flash Array is in Low Power (Sleep) Mode. Indicates that the flash array is in a low power (sleep) mode. The flash controller automatically wakes the flash when required for another data transaction. User may wake the flash at any time by writing the IDLE command to the FLCC_CMD register. Flash wake-up times vary but are typically ~5us. When possible, it is recommend that the user begin waking the flash ~5us before it will be used as a performance optimization.	

Table 8-19: FLCC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/W1C)	CMDFAIL	Provides Information on Command Failures. This field indicates the status of a command upon completion. If multiple commands are executed without clearing these bits then only the first error encountered is stored.	
		0	Success Successful completion of a command (e.g. WRITE)
		1	Ignored Attempted access of a protected or out of memory location (such a command is ignored).
		2	Verify Error Read verify error occurred. This status will be returned for either of 2 causes; Failed erasures and/or failed signature checks. Failed Erasure: After erasing flash page(s) the controller reads the corresponding word(s) to verify that the erasure completed successfully. If data still persists, the erasure has failed and this field reports the failure. Failed signature check: If the Sign command is executed and the resulting signature does not match the data stored in the most significant 32-bit word of the sign-checked block, the sign check has failed and this field reports the failure.
		3	Abort Indicates a command was aborted either by user code (abort command issued) or by a system interrupt (see `FLCC_ABORT_EN_HI` and `FLCC_ABORT_EN_LO` registers for details)
3 (R/W1C)	WRALCOMP	Write Almost Complete. WRITE data registers are re-opened for access as an ongoing write nears completion. Requesting another write operation* before `FLCC_STAT.CMDCOMP` asserts will result in a "burst write". Burst writes take advantage of low level protocols of the Flash memory and result in significant performance gains (~15us saved from each write operation). *Note that the performance gain of a Burst Write only applies to back-to-back writes within the same row of the flash array. See the flash IP spec for memory organization details.	
2 (R/W1C)	CMDCOMP	Command Complete. This bit asserts when a command completes. (Automatically clears when a new command is requested) NOTE: Following a power-on-reset, the flash controller performs a number of operations (e.g. verifying the integrity of code in info space). At the conclusion of this process the controller sets the `FLCC_STAT.CMDCOMP` bit to indicate that the process has completed.	

Table 8-19: FLCC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/NW)	WRCLOSE	WRITE Registers are Closed. The WRITE data/address registers and the command register are closed for access. This bit is asserted part of the time while a write is in progress. If this bit is high, the related registers are in use by the flash controller and cannot be written. This bit clears when `FLCC_STAT.WRALCOMP` flag goes high, indicating that the ongoing write command has consumed the associated data and these registers may now be overwritten with new data.
0 (R/NW)	CMDBUSY	Command Busy. This bit is asserted when the flash block is actively executing any command entered via the command register. NOTE: There is a modest delay between requesting a command and having this bit assert; user code polling for command completion should watch for `FLCC_STAT.CMDCOMP` bit rather than `FLCC_STAT.CMDBUSY`.

Time Parameter 0

User Key is required to write this register.

This register should not be modified while a flash write or erase command is in progress.

This register defines a set of parameters used to control the timing of signals driven to the Flash Memory. The default values are appropriate for a system clock of 26 MHz and a reference clock (driven by the internal oscillator) operating within 10% of 13 MHz.

The value of each timing parameter consists of a user programmable nibble (4 bits) as well as some number of hard-coded bits. User programmable bits are the most significant bits for each parameter.

Time parameters describe the number of ref-clk periods to wait when meeting the associated timing constraint of the flash memory itself. Note that clock-domain-crossings and the constraints of signals not described by these parameters will increase the effective delays by a small margin. When programming the time parameter registers the user should select a value approaching the minimum time for each constraint.

Improper programming of this register may result in damage to the flash memory during PROGRAM or ERASE operations.

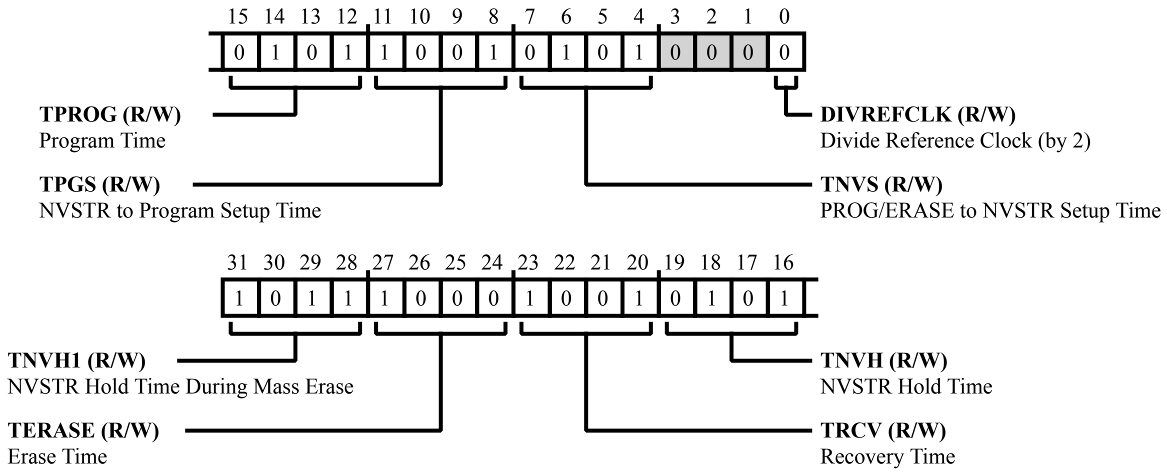


Figure 8-22: FLCC_TIME_PARAM0 Register Diagram

Table 8-20: FLCC_TIME_PARAM0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:28 (R/W)	TNVH1	NVSTR Hold Time During Mass Erase. Determines the upper 4 bits of an 11 bit value loaded into the timer. The lower bits are hard-coded to 0x14. With an ideal refclk at 13MHz: Min [0x0] = 1.5us Default [0xB] = 110us Max [0xF] = 149us

Table 8-20: FLCC_TIME_PARAM0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
27:24 (R/W)	TERASE	Erase Time. Determines the upper 4 bits of a 19 bit value loaded into the timer. The lower bits are hard-coded to 0x7370. With an ideal refclk at 13MHz: Min [0x0] = 2.3ms Default [0xB] = 30ms Max [0xF] = 40ms
23:20 (R/W)	TRCV	Recovery Time. Determines the upper 4 bits of an 8 bit value loaded into the timer. The lower bits are hard-coded to 0x2. With an ideal refclk at 13MHz: Min [0x0] = 154ns Default [0x9] = 1.1us Max [0xF] = 18.6us
19:16 (R/W)	TNVH	NVSTR Hold Time. Determines the upper 4 bits of an 8 bit value loaded into the timer. The lower bits are hard-coded to 0x1. With an ideal refclk at 13MHz: Min [0x0] = 77ns Default [0x5] = 5.5us Max [0xF] = 18.5us
15:12 (R/W)	TPROG	Program Time. Determines the upper 4 bits of a 10 bit value loaded into the timer. The lower bits are hard-coded to 0x04. With an ideal refclk at 13MHz: Min [0x0] = 308ns Default [0x6] = 30us Max [0xF] = 74.2us
11:8 (R/W)	TPGS	NVSTR to Program Setup Time. Determines the upper 4 bits of an 8 bit value loaded into the timer. The lower bits are hard-coded to 0x2. With an ideal refclk at 13MHz: Min [0x0] = 154ns Default [0x9] = 1.1us Max [0xF] = 18.6us
7:4 (R/W)	TNVS	PROG/ERASE to NVSTR Setup Time. Determines the upper 4 bits of an 8 bit value loaded into the timer. The lower bits are hard-coded to 0x1. With an ideal refclk at 13MHz: Min [0x0] = 77ns Default [0x5] = 5.5us Max [0xF] = 18.5us
0 (R/W)	DIVREFCLK	Divide Reference Clock (by 2). If '1' the reference clock is divided by 2. All time parameters are relative to the reference clock; dividing the clock enables the selection of long time periods if necessary. NOTE: It is unlikely that user code should ever need to modify the time parameters; support is provided in the unlikely event that a user finds it necessary.

Time Parameter 1

See FLCC_TIME_PARAM0 for documentation

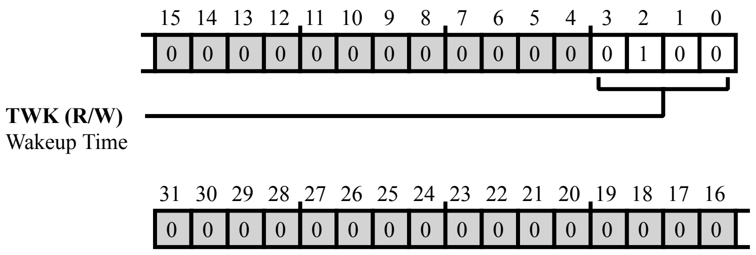


Figure 8-23: FLCC_TIME_PARAM1 Register Diagram

Table 8-21: FLCC_TIME_PARAM1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	TWK	Wakeup Time. Determines the upper 4 bits of an 8 bit value loaded into the timer. The lower bits are hard-coded to 0xB With an ideal refclk at 13MHz: Min [0x0] = 847ns Default [0x4] = 5.7us Max [0xF] = 19.3us

User Configuration

User key is required (see FLCC_KEY register for details).

Write to this register to enable user control of DMA and Auto-increment features.

When user code has finished accessing this register, garbage data should be written to the FLCC_KEY register to re-assert protection.

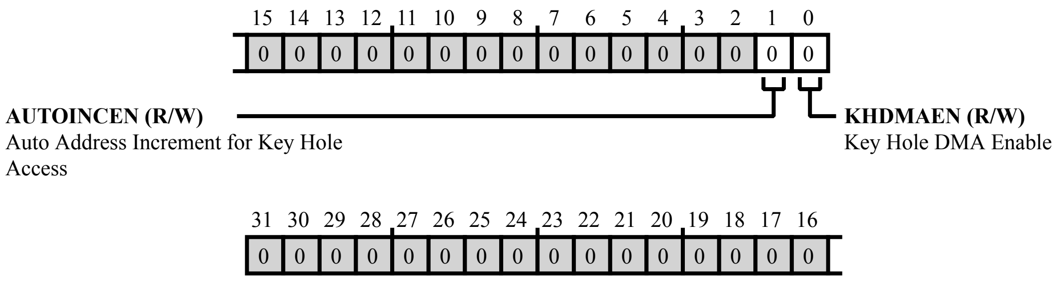


Figure 8-24: FLCC_UCFG Register Diagram

Table 8-22: FLCC_UCFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	AUTOINCEN	Auto Address Increment for Key Hole Access. When this bit is set FLCC_KH_ADDR will automatically increment by 0x8 during each WRITE command or after each READ command. This enables user code to write a series of sequential flash locations without having to manually set the flash address for each write. The FLCC_KH_ADDR is incremented, and may be observed by user code, when FLCC_STAT.WRALCOMP is asserted during a WRITE command, or when FLCC_STAT.WRALCOMP is asserted after a READ command. When this bit is set user code may not directly modify FLCC_KH_ADDR

Table 8-22: FLCC_UCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	KHDMAEN	Key Hole DMA Enable. The flash controller will interact with the DMA controller when this bit is set. Prior to setting this bit, user code should: - Write the starting address to `FLCC_KH_ADDR` - Configure the DMA controller to write data to `FLCC_KH_DATA1` (address must be DWORD aligned) - Configure the DMA controller to always write pairs of 32 bit words (R Power = 1) - Configure the DMA controller to write an integer number of data pairs (for an odd number of words user code must write one word manually without the help of DMA) Note that all DMA writes will automatically increment the target address (similar to the behavior of `FLCC_UCFG.AUTOINCEN`). The DMA controller may only be used to write sequential addresses starting from the value of `FLCC_KH_ADDR` The flash controller will automatically begin write operations each time the DMA controller provides a pair of words to write. Interaction with the DMA controller has been designed to use burst writes which may significantly reduce overall programming time.

Write Protection

User Key is required to modified this register.

The FLCC_WRPROT register may be automatically configured during device boot up; in this event the boot loader reads data from user space and loads that data into this register.

User code may affect non-volatile write protection by writing to the appropriate location in the flash memory (see chapter on Protection for details). By default, the relevant location in flash is 0x3FFF0 (the 4th most significant word in user space), but may be relocated by ADI's secure bootloader.

User code may alternatively assert protection at runtime for any unprotected blocks by directly writing this register: Blocks may have protection added but cannot have protection removed; changes will be lost on reset. This approach is suggested especially during user code development.

All write protection is cleared on a power-on-reset but note that the ADI secure bootloader will reassert write protection as defined by the FLCC_WRPROT word in user space before enabling user access to the flash array. Therefore removing write protection can only be performed by an ERASEPAGE command of the most significant page in user space (provided that page is not currently protected) or by a MASSERASE command. Following a successful MASSERASE command all protection of pages in user space is immediately cleared (user may write to user space immediately following such an erase without a device reset required).

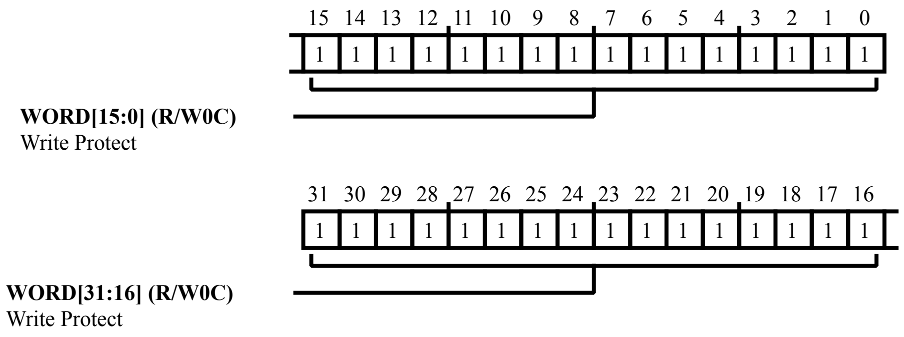


Figure 8-25: FLCC_WRPROT Register Diagram

Table 8-23: FLCC_WRPROT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W0C)	WORD	Write Protect. Clear bits to write protect related groups of user space pages. Once cleared these bits can only be set again by resetting the part. Each bit of this 32 bit word represents a 32nd of the total available user space. For 256kB parts consisting of 2kB pages (128 pages) each bit represents the write protection state of a group of 4 pages. For 128kB parts consisting of 2kB pages (64 pages) each bit represents the write protection state of a group of 2 pages. for 64kB parts consisting of 2kB pages (32 pages) each bit represents the write protection state of a single page. The most significant bit of this register corresponds to the most significant group of pages in user space.

Write Abort Address

Address of recently aborted write command. This address is only populated if the aborted write command was started; if the command is aborted early enough to have no affect on the flash IP this address will not be updated.

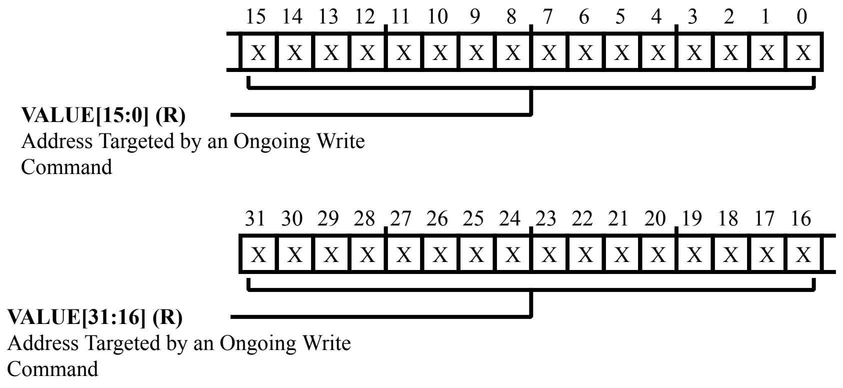

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
X X X X X X X X X X X X X X X X
VALUE[15:0] (R)
Address Targeted by an Ongoing Write Command
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
X X X X X X X X X X X X X X X X
VALUE[31:16] (R)
Address Targeted by an Ongoing Write Command

Figure 8-26: FLCC_WR_ABORT_ADDR Register Diagram

Table 8-24: FLCC_WR_ABORT_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Address Targeted by an Ongoing Write Command. Holds the address targeted by an ongoing write command and retains its value after an ABORT event. User code may read this register to determine the flash location(s) affected a write abort. The register value is not guaranteed to persist once a new flash command is requested, therefore user code should read this value immediately following an aborted WRITE.

9 Static Random Access Memory (SRAM)

This chapter provides an overview of the SRAM functionality of the ADuCM302x MCU.

SRAM Features

The SRAM used by the ADuCM302x MCU supports the following features:

- Low power controller for data SRAM, instruction SRAM and cache SRAM.
- Total available memory: 64 KB
- Maximum retained memory in hibernate mode: 32 KB
- Data SRAM is composed of 32 KB. Option to retain 8 KB or 16 KB in hibernate mode.
- Instruction SRAM is composed of 32 KB. Option to retain 16 KB in hibernate mode.
- If instruction SRAM is not enabled, the associated 32 KB can be mapped as data SRAM. In this case we have the option to retain 8 KB, 16 KB, 24 KB or 32 KB of data SRAM.
- When cache controller is enabled, 4 KB of instruction SRAM will be reserved for cache data. The 4 KB of cache data are not retained in hibernate mode.
- Parity bit error detection (optional) is available on all SRAM memories. Two parity bits are associated with each 32-bit word. Parity check can be configured to be enabled/disabled in different memory regions.
- Byte, Half-word, and Word accesses are supported.

For more information, refer to SRAM Region.

SRAM Configuration

Instruction SRAM vs Data SRAM

If the `PMG_TST_SRAM_CTL.INSTREN` bit is asserted, 32 KB of SRAM is mapped at start address 0x1000_0000 as Instruction SRAM. 32 KB of data SRAM is mapped in two sections, the first starting at 0x2000_0000 and second starting at 0x2004_0000. If cache memory feature is used, only 28 KB is available for instruction SRAM.

If PMG_TST_SRAM_CTL.INSTREN bit is 0 and cache is disabled, the 64 KB of SRAM is mapped as data SRAM. The memory is arranged in two sections, the first one (32 KB) is mapped at start address 0x2000_0000 and the second one (32 KB) at 0x2004_0000. If cache memory feature is used, the second section will only map 28 KB, so the total data SRAM available is 60 KB.

By default, at power up and hardware reset, the 32 KB of SRAM is made available as instruction SRAM. If the user needs to exercise the option of using a total of 64 KB data SRAM, the PMG_TST_SRAM_CTL.INSTREN bit must be programmed to zero at the start of the user code.

When cache controller is enabled, SRAM bank 5 is not accessible. A bus error (unmapped address) is generated if an access is attempted.

For more information about this, refer to SRAM Region.

SRAM Retention in Hibernate Mode

In terms of the amount of SRAM being retained in hibernate mode, different configurations are available to the user.

The content of the first 8 KB (Bank0) of data SRAM mapped at 0x2000_0000 is always retained. The SRAM mapped from 0x2004_0000 onwards (Bank3, Bank4, and Bank5) cannot be retained in hibernate mode.

If the PMG_TST_SRAM_CTL.BNK1EN bit is enabled, 8 KB of data SRAM mapped from 0x2000_2000 to 0x2000_3FFF (Bank1) is retained in hibernate mode.

If PMG_TST_SRAM_CTL.INSTREN = 1 and PMG_TST_SRAM_CTL.BNK2EN bit is enabled, 16 KB of instruction SRAM mapped from 0x1000_0000 to 0x1000_3FFF (Bank2) is retained.

If PMG_TST_SRAM_CTL.INSTREN = 0, 16 KB of data SRAM mapped from 0x2000_4000 to 0x2000_7FFF is retained.

SRAM Programming Model

The SRAM programming model is explained below.

Stack

The SRAM start address is set to 0x2000_0000 and the stack pointer is set at 0x2000_2000. The stack will be written from 0x2000_1FFF downwards. The covered memory region is always be retained (see SRAM Region). To reserve a given size for the stack area, the user can declare a data array of that desired size ending at position 0x2000_1FFF. This way, the stack will not be overwritten by the compiler when allocating new variables.

SRAM Parity

For robustness, parity check can be enabled on all or a user selected group of SRAM banks. Parity check can detect up to two errors per word. Parity check feature can be enabled by asserting PMG_TST_SRAM_CTL.PENBNK0 to PMG_TST_SRAM_CTL.PENBNK5 bits for each SRAM bank. User must configure these bits at the beginning of the program code.

Parity is checked when data is read and when byte or halfword data is written. Parity is not checked when word (32 bits) write is performed. If a parity error is detected, a bus error is generated. Even if parity error is detected when writing a byte or halfword, the write operation is completed and parity bits are updated according to the new data. User must manage the parity error in the bus fault interrupt routine.

SRAM Initialization

If parity check is enabled, SRAM contents have to be initialized to avoid false parity errors. A dedicated hardware can automatically initialize the selected SRAM banks and the whole process takes 1024 HCLK cycles to complete. This hardware is fully programmable by the user so initialization can be started automatically or manually.

As initialization overwrites the contents of the selected SRAM banks, this process must be performed before writing to those SRAMs. During the initialization sequence, if a write or read access to a SRAM bank being initialized is detected, bus error response is issued until the initialization sequence is completed. SRAMs banks that are not selected to be initialized can be accessed as usual during the initialization process of the rest of the banks.

The initialization for a particular SRAM bank can be monitored for its completion by polling the appropriate `PMG_TST_SRAM_INITSTAT.BNK0` to `PMG_TST_SRAM_INITSTAT.BNK5` bits. Every time a particular SRAM bank is initialized, its associated `PMG_TST_SRAM_INITSTAT.BNK0` to `PMG_TST_SRAM_INITSTAT.BNK5` bits are cleared and will remain low until initialization is completed.

After power up, SRAM bank 0 (8 KB) is automatically initialized. This memory is always retained and will contain the stack pointer and critical information. Its contents will not have to be overwritten in the future as initialization is already been performed. User must avoid initializing those SRAM banks that are already initialized as they may already contain user information.

Initialization of more SRAM banks, where parity will be enabled, can be achieved at any time by writing to `PMG_TST_SRAM_CTL` register. Here, the appropriate `PMG_TST_SRAM_CTL.BNK0EN` to `PMG_TST_SRAM_CTL.BNK5EN` bits for SRAM banks which need parity to be enabled has to be set to 1; set to 1 the `PMG_TST_SRAM_CTL.STARTINIT` bit. This bit will be auto-cleared to 0 after being written and will trigger the initialization sequence.

After hibernate mode, the contents of non-retained SRAM banks is lost. If these banks have parity enabled, initialization is required.

There are two options to initialize the required SRAM banks after hibernate mode:

1. Initialize by writing to `PMG_TST_SRAM_CTL.STARTINIT` register after coming from hibernate mode. Those SRAMs banks which `PMG_TST_SRAM_CTL.BNK0EN` to `PMG_TST_SRAM_CTL.BNK5EN` bits are set to 1 will be initialized.
2. Automatic initialization after hibernate mode. No write to `PMG_TST_SRAM_CTL` is required each time we come back from hibernate mode. To select this automatic mode, the user has to previously set the `PMG_TST_SRAM_CTL.AUTOINIT` bit. SRAMs selected for initialization in this register are automatically initialized after coming back from hibernate mode.

Initialization resets the contents of the selected memory banks. User must properly select the memory banks that are initialized so that user information is not lost.

The initialization sequence can be aborted at any time by writing to the `PMG_TST_SRAM_CTL.ABTINIT` bit. This bit is self cleared after it has been written.

Initialization in Cache

When cache memory is used, parity can also be enabled on its associated SRAM bank (bank 5). In this case (when SRAM bank 5 is used as cache memory) initialization is not required. The reason is that initialization is only required when we perform byte or halfword accesses to SRAM with parity check enabled. When SRAM bank 5 is used as cache memory, all the accesses are word accesses.

As initialization is not required for the cache memory, the cache feature will be available right after coming back from hibernate mode (there is no initialization time penalty). To prevent undesired bus errors, the controller ignores any initialization of SRAM bank 5 when cache is enabled.

ADuCM302x SRAM Register Description

Table 9-1: ADuCM302x SRAM Register List

Name	Description	Reset	Access
`PMG_TST_SRAM_INITSTAT`	Initialization Status Register	0x00000000	R/W
`PMG_TST_SRAM_CTL`	Control for SRAM Parity and Instruction SRAM	0x80000000	R/W
`PMG_SRAMRET`	Control for Retention SRAM in Hibernate Mode	0x00000000	R/W

Initialization Status Register

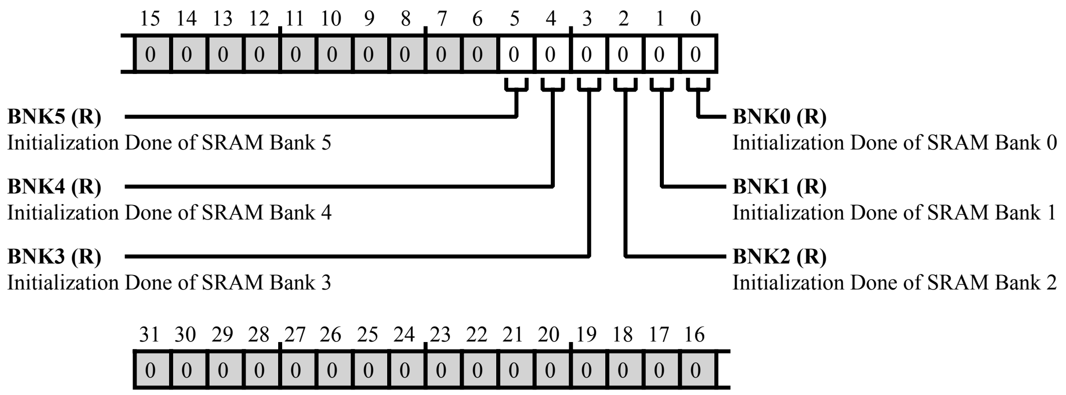

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BNK5 (R)
Initialization Done of SRAM Bank 5
BNK4 (R)
Initialization Done of SRAM Bank 4
BNK3 (R)
Initialization Done of SRAM Bank 3
BNK0 (R)
Initialization Done of SRAM Bank 0
BNK1 (R)
Initialization Done of SRAM Bank 1
BNK2 (R)
Initialization Done of SRAM Bank 2
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 9-1: PMG_TST_SRAM_INITSTAT Register Diagram

Table 9-2: PMG_TST_SRAM_INITSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/NW)	BNK5	Initialization Done of SRAM Bank 5. 0:Not initialized; 1:Initialization completed
4 (R/NW)	BNK4	Initialization Done of SRAM Bank 4. 0:Not initialized; 1:Initialization completed
3 (R/NW)	BNK3	Initialization Done of SRAM Bank 3. 0:Not initialized; 1:Initialization completed
2 (R/NW)	BNK2	Initialization Done of SRAM Bank 2. 0:Not initialized; 1:Initialization completed
1 (R/NW)	BNK1	Initialization Done of SRAM Bank 1. 0:Not initialized; 1:Initialization completed
0 (R/NW)	BNK0	Initialization Done of SRAM Bank 0. 0:Not initialized; 1:Initialization completed

Control for SRAM Parity and Instruction SRAM

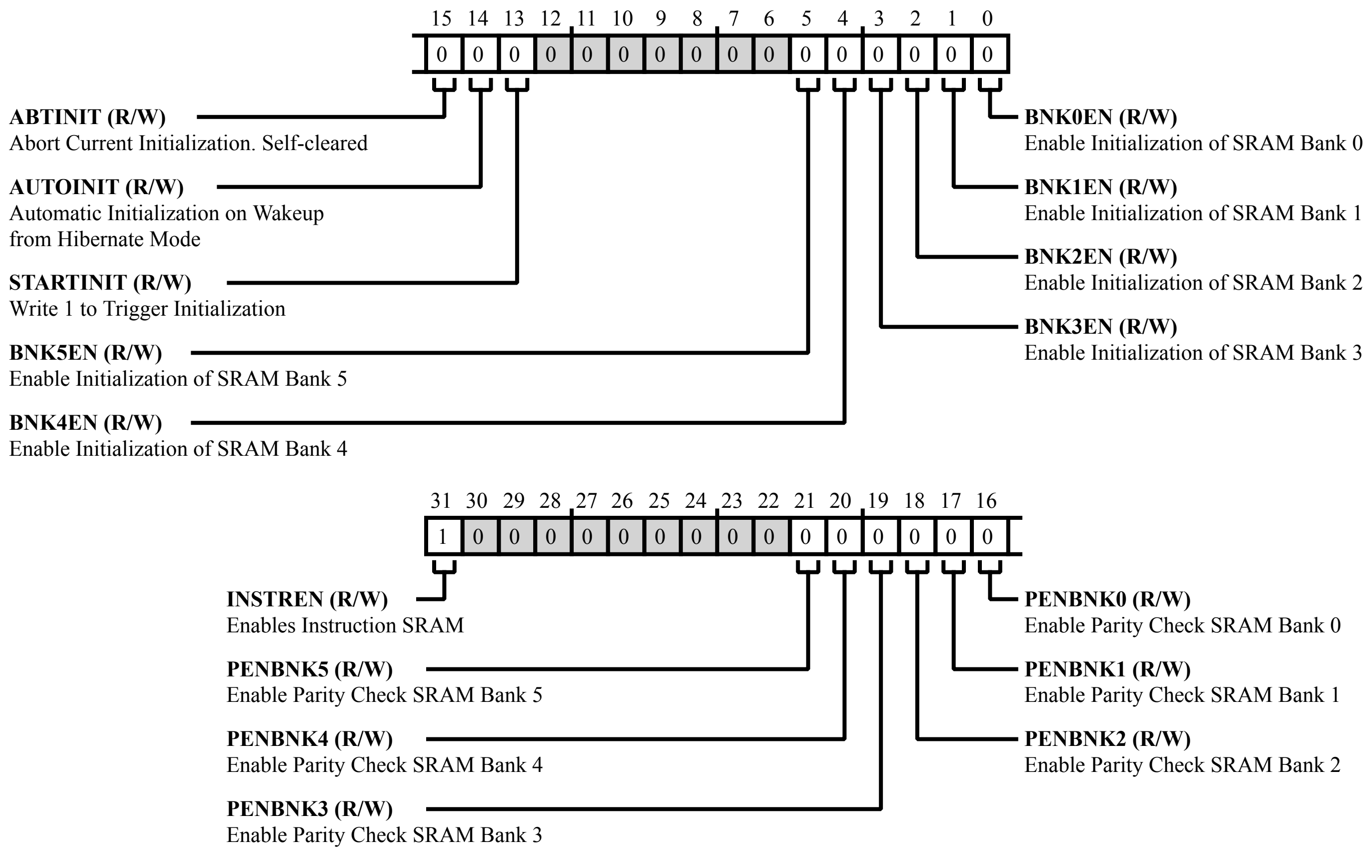

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ABTINIT (R/W)
Abort Current Initialization. Self-cleared
AUTOINIT (R/W)
Automatic Initialization on Wakeup from Hibernate Mode
STARTINIT (R/W)
Write 1 to Trigger Initialization
BNK5EN (R/W)
Enable Initialization of SRAM Bank 5
BNK4EN (R/W)
Enable Initialization of SRAM Bank 4
BNK0EN (R/W)
Enable Initialization of SRAM Bank 0
BNK1EN (R/W)
Enable Initialization of SRAM Bank 1
BNK2EN (R/W)
Enable Initialization of SRAM Bank 2
BNK3EN (R/W)
Enable Initialization of SRAM Bank 3
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INSTREN (R/W)
Enables Instruction SRAM
PENBNK5 (R/W)
Enable Parity Check SRAM Bank 5
PENBNK4 (R/W)
Enable Parity Check SRAM Bank 4
PENBNK3 (R/W)
Enable Parity Check SRAM Bank 3
PENBNK0 (R/W)
Enable Parity Check SRAM Bank 0
PENBNK1 (R/W)
Enable Parity Check SRAM Bank 1
PENBNK2 (R/W)
Enable Parity Check SRAM Bank 2

Figure 9-2: PMG_TST_SRAM_CTL Register Diagram

Table 9-3: PMG_TST_SRAM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	INSTREN	Enables Instruction SRAM. 0: Disable, 1: Enable
21 (R/W)	PENBNK5	Enable Parity Check SRAM Bank 5. 0: Disable, 1: Enable
20 (R/W)	PENBNK4	Enable Parity Check SRAM Bank 4. 0: Disable, 1: Enable
19 (R/W)	PENBNK3	Enable Parity Check SRAM Bank 3. 0: Disable, 1: Enable
18 (R/W)	PENBNK2	Enable Parity Check SRAM Bank 2. 0: Disable, 1: Enable

Table 9-3: PMG_TST_SRAM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/W)	PENBNK1	Enable Parity Check SRAM Bank 1. 0: Disable, 1: Enable
16 (R/W)	PENBNK0	Enable Parity Check SRAM Bank 0. 0: Disable, 1: Enable
15 (R/W)	ABTINIT	Abort Current Initialization. Self-cleared.
14 (R/W)	AUTOINIT	Automatic Initialization on Wakeup from Hibernate Mode.
13 (R/W)	STARTINIT	Write 1 to Trigger Initialization. Self-cleared.
5 (R/W)	BNK5EN	Enable Initialization of SRAM Bank 5. 0: Disable, 1: Enable
4 (R/W)	BNK4EN	Enable Initialization of SRAM Bank 4. 0: Disable, 1: Enable
3 (R/W)	BNK3EN	Enable Initialization of SRAM Bank 3. 0: Disable, 1: Enable
2 (R/W)	BNK2EN	Enable Initialization of SRAM Bank 2. 0: Disable, 1: Enable
1 (R/W)	BNK1EN	Enable Initialization of SRAM Bank 1. 0: Disable, 1: Enable
0 (R/W)	BNK0EN	Enable Initialization of SRAM Bank 0. 0: Disable, 1: Enable

Control for Retention SRAM in Hibernate Mode

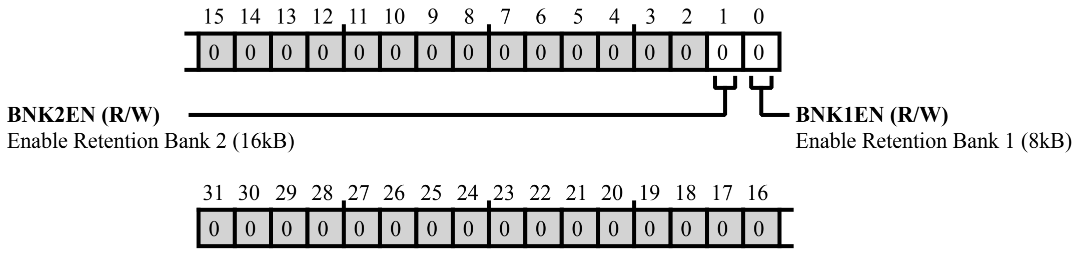


Figure 9-3: PMG_SRAMRET Register Diagram

Table 9-4: PMG_SRAMRET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	BNK2EN	Enable Retention Bank 2 (16kB).
0 (R/W)	BNK1EN	Enable Retention Bank 1 (8kB).

10 Cache

Enabling Cache increases the performance of applications executing from flash. Cache memory coexists with SRAM. When Cache is enabled, part of the SRAM is allocated to Cache. Therefore, Cache cannot be used for other purposes. Instruction Cache (I-Cache) is of size is 4 KB. On wake up from hibernate, I-Cache contents are not retained.

Cache Programming Model

Instruction Cache is disabled on power-up.

Programming Guidelines

The sequence to enable cache is as follows:

1. Read `FLCC_CACHE_STAT.ICEN` bit to make sure that Cache is disabled. Poll until this bit is cleared.
2. Write USER KEY (0xF123F456) to `FLCC_CACHE_KEY` register.
3. Set `FLCC_CACHE_SETUP.ICEN` bit to enable the cache.

ADuCM302x FLCC_CACHE Register Descriptions

Cache Controller (FLCC_CACHE) contains the following registers.

Table 10-1: ADuCM302x FLCC_CACHE Register List

Name	Description
FLCC_CACHE_KEY	Cache Key
FLCC_CACHE_SETUP	Cache Setup
FLCC_CACHE_STAT	Cache Status

Cache Key

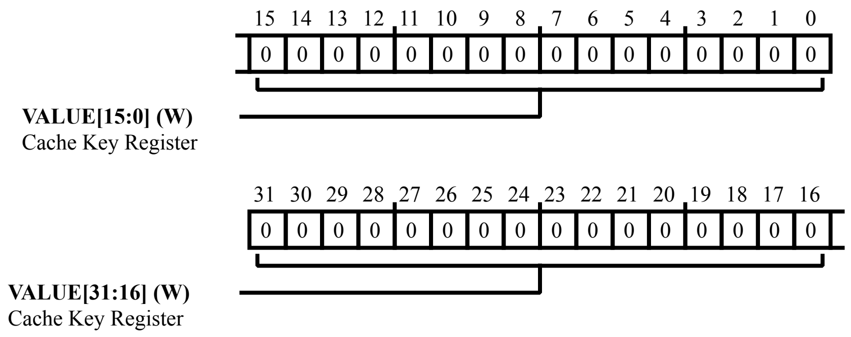


Figure 10-1: FLCC_CACHE_KEY Register Diagram

Table 10-2: FLCC_CACHE_KEY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Cache Key Register. Enter 0xF123_F456 to set the UserKey. Returns 0x0 if read. The key is cleared automatically after writing to `FLCC_CACHE_SETUP` register.

Cache Setup

Cache User key is required to enable a write to this location. Key will be cleared after a write to this register.

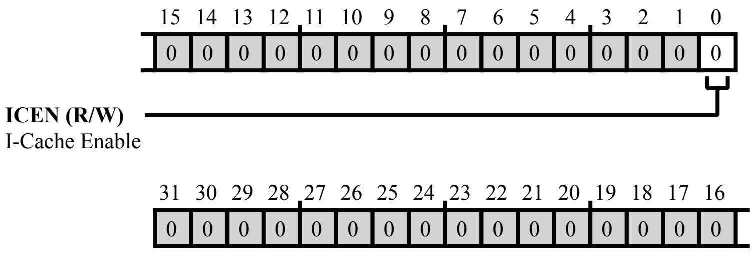


Figure 10-2: FLCC_CACHE_SETUP Register Diagram

Table 10-3: FLCC_CACHE_SETUP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	ICEN	I-Cache Enable. If this bit set, then I-Cache is enabled for AHB accesses. If 0, then I-cache is disabled, and all AHB accesses will be via Flash memory.

Cache Status

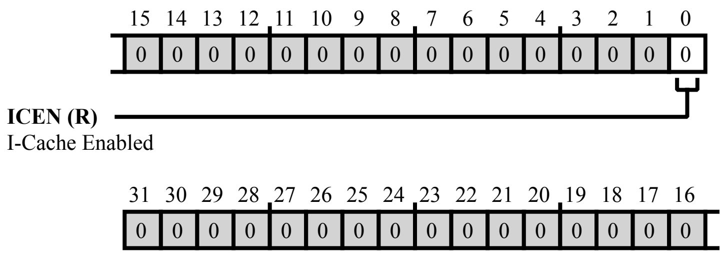


Figure 10-3: FLCC_CACHE_STAT Register Diagram

Table 10-4: FLCC_CACHE_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/NW)	ICEN	I-Cache Enabled. If this bit is set then I-Cache is enabled and when cleared I-Cache is disabled.

11 Direct Memory Access (DMA)

The direct memory access (DMA) controller is used to perform data transfer tasks and offload these from the ADuCM302x MCU. DMA is used to provide high speed data transfer between peripherals and memory. Data can be quickly moved by the DMA without any CPU actions. This keeps the CPU resources free for other operations.

DMA Features

The DMA supports the following features:

- 25 independent DMA channels
- Two programmable priority levels for each DMA channel
- Each priority level arbitrates using a fixed priority that is determined by the DMA channel number
- Each DMA channel can access a primary, and/or alternate channel control data structure
- Supports the following multiple transfer types:
  - Memory-to-memory
  - Memory-to-peripheral
  - Peripheral-to-memory
- Supports the following multiple DMA cycle types:
  - Basic
  - Auto request
  - Ping-Pong
  - Scatter-Gather
- Supports multiple DMA transfer data widths (8-bit, 16-bit, and 32-bit)
- Each DMA channel can have independent source and destination increment/decrement controls

DMA Functional Description

The DMA has 25 channels in total, each dedicated to managing memory access requests from peripherals. The ***DMA Channels*** table shows the assignments.

Each DMA channel is connected to dedicated hardware DMA requests, and the software trigger is also supported on each channel. This configuration is done by software. Each DMA channel has a programmable priority level default or high. Each priority level arbitrates using a fixed priority that is determined by the DMA channel number.

The DMA controller supports multiple DMA transfer data widths. However, the source and destination transfer sizes must be the same. Also, always align the source and destination addresses to the data transfer size. It has a single output to indicate when an error condition occurs: DMA error interrupt. An error condition can occur when a bus error happens while doing a DMA transfer, or when the DMA controller reads an invalid cycle control. The DMA controller supports memory-to-memory, peripheral-to-memory, and memory-to-peripheral transfers and has access to flash or SRAM as source and destination.

Table 11-1: DMA Channels

Channel Node	Peripheral
0	SPI2 Tx
1	SPI2 Rx
2	SPORT0A
3	SPORT0B
4	SPI0 Tx
5	SPI0 Rx
6	SPI1 Tx
7	SPI1 Rx
8	UART Tx
9	UART Rx
10	I2C Slave Tx
11	I2C Slave Rx
12	I2C Master
13	Crypto In
14	Crypto Out
15	Flash
16 to 23	Software DMA
24	ADC Rx

DMA Architectural Concepts

The DMA channel provides a means to transfer data between memory spaces or between memory and a peripheral using the system interface. The DMA channel provides an efficient means of distributing data throughout the system, freeing up the core for other operations. Each peripheral that supports DMA transfers has its own dedicated DMA channel or channels with its own register set that configures and controls the operating modes of the DMA transfers.

DMA Operating Modes

The DMA controller has two buses where one is connected to the system bus shared with the Cortex-M3 core and the other bus is connected to 16-bit peripherals. The DMA request may stop the CPU access to the system bus for some bus cycles, such as when the CPU and DMA target the same destination (memory or peripheral).

The DMA controller fetches channel control data structures located in the system memory to perform data transfers. The DMA-capable peripherals, when enabled to use DMA, can request the DMA controller for a transfer. At the end of the programmed number of DMA transfers for a channel, the DMA controller generates a single cycle `dma_done` interrupt corresponding to that channel. This interrupt indicates the completion of the DMA transfer. Separate interrupt enable bits are available in the NVIC for each of the DMA channels.

Channel Control Data Structure

Every channel has two control data structures associated with it: a primary data structure and an alternate data structure. For simple transfer modes, the DMA controller uses either the primary or alternate data structure. For more complex data transfer modes, such as Ping-Pong or Scatter-Gather, the DMA controller uses both the primary and alternate data structures. Each control data structure (primary or alternate) occupies four 32-bit locations in the memory as shown in the ***Channel Control Data Structure*** table.

Table 11-2: Channel Control Data Structure

Offset	Name	Description
0x00	SRC_END_PTR	Source End Pointer
0x04	DST_END_PTR	Destination End Pointer
0x08	CHNL_CFG	Control Data Configuration
0x0C	RESERVED	Reserved

Memory Map for Primary DMA Structure figure shows an example of the entire channel control data structure for eight DMA channel case. It uses 512 bytes of system memory.

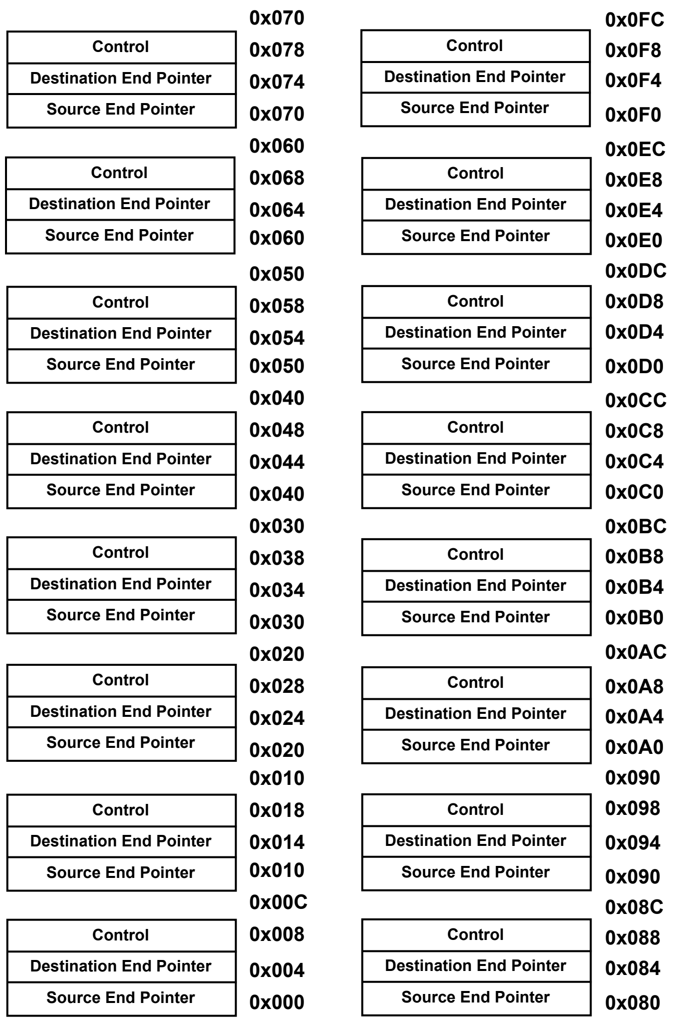


Figure 11-1: Memory Map for Primary DMA Structure

Before the controller can perform a DMA transfer, the data structure related to the DMA channel needs to be written to the designated location in system memory. The Source End Pointer memory location contains the end address of the source data. The Destination End Pointer memory location contains the end address of the destination data. The Control memory location contains the channel configuration control data. This determines the source and destination data size, number of transfers, and the number of data transfers for each arbitration.

When the DMA controller receives a request for a channel, it reads the corresponding data structure from the system memory into its internal cache. Any update to the descriptor in the system memory until `dma_done` interrupt does not guarantee expected behavior. It is recommended that the user not update the descriptor before receiving `dma_done`.

Source Data End Pointer

The `SRC_END_PTR` memory location stores the address of the last location from where data will be read as part of DMA transfer. This memory location must be programmed with the end address of the source data before the controller can perform a DMA transfer. The controller reads this memory location when it starts the first DMA data transfer. DMA controller does not write to this memory location.

Table 11-3: Source Data End Pointer

Bit	Name	Description
[31:0]	SRC_END_PTR	The end address of the source data.

Destination Data End Pointer

The `DST_END_PTR` memory location stores the address of the last location where data will be written to as part of DMA transfer. This memory location must be programmed with the end address of the destination data before the controller can perform a DMA transfer. The controller reads this memory location when it starts the first DMA data transfer. DMA controller does not write to this memory location.

Table 11-4: Destination Data End Pointer

Bit	Name	Description
[31:0]	DST_END_PTR	The end address of the destination data.

Control Data Configuration

For each DMA transfer, the control data configuration (`CHNL_CFG`) memory location provides the control information for the DMA transfer to the controller.

Table 11-5: Destination Data End Pointer

Bits	Name	Description
31:30	DST_INC	Destination address increment. The address increment depends on the source data width as follows: SRC_SIZE: Byte 00: byte 01: half-word 10: word 11: no increment. Address remains set to the value that the DST_END_PTR memory location contains. SRC_SIZE: Half-word 00: reserved 01: halfword 10: word 11: no increment. Address remains set to the value that the DST_END_PTR memory location contains. SRC_SIZE: Word 00: reserved 01: reserved 10: word 11: no increment. Address remains set to the value that the DST_END_PTR memory location contains.
29:28	RESERVED	Undefined. Write as zero

Table 11-5: Destination Data End Pointer (Continued)

Bits	Name	Description
27:26	SRC_INC	Source address increment. The address increment depends on the source data width as follows: SRC_SIZE: Byte 00: byte 01: half-word 10: word 11: no increment. Address remains set to the value that the SRC_END_PTR memory location contains. SRC_SIZE: Half-word 00: reserved 01: half-word 10: word. 11: no increment. Address remains set to the value that the SRC_END_PTR memory location contains. SRC_SIZE: Word 00: reserved 01: reserve 10: word 11: no increment. Address remains set to the value that the SRC_END_PTR memory location contains.
25:24	SRC_SIZE	Size of the source data. 00: byte 01: half-word 10: word 11: reserved
23:18	RESERVED	Undefined. Write as zero.
17:14	R_Power	R_Power arbitrates after x DMA transfers.
		0000: x = 1
		0001: x = 2
		0010: x = 4
		0011: x = 8
		0100: x = 16
		0101: x = 32
		0110: x = 64

Table 11-5: Destination Data End Pointer (Continued)

Bits	Name	Description
		0111: x = 128
		1000: x = 256
		1001: x = 512
		1010 to 1111: x = 1024
		NOTE: Software DMA transfers can use any value from 0000 to 1111. DMA transfers that involve peripherals should always use 0000 with the exception of the Crypto block. The DMA operation is indeterminate if a value other than 0000 is programmed for the DMA transfers involving peripherals.
13:4	N_minus_1	Total number of transfers in the current DMA cycle - 1. This value indicates the total number of transfers in the DMA cycle and not the total number of bytes. The possible values are 0 - 1023.
3	RESERVED	Undefined. Write as zero.
2:0	cycle_ctrl	The operating mode of the DMA cycle.
		000: Stop (Invalid)
		001: Basic
		010: Auto-Request
		011: Ping-Pong
		100: Memory Scatter-Gather Primary
		101: Memory Scatter-Gather Alternate
		110: Peripheral Scatter-Gather Primary
		111: Peripheral Scatter-Gather Alternate

During the DMA transfer process, if any error occurs during the data transfer, `CHNL_CFG` is written back to the system memory, with `N_minus_1` field updated to reflect the number of transfers yet to be completed. When a full DMA cycle is complete, `cycle_ctrl` bits are made invalid to indicate the completion of transfer.

DMA Priority

The priority of a channel is determined by its number and priority level. Each channel can have two priority levels: default or high. All channels at the high priority level have higher priority than all channels at the default priority level. At the same priority level, a channel with a lower channel number has a higher priority than a channel with a higher channel number. The DMA channel priority levels can be changed by writing into the appropriate bit in the `DMA_PRI_SET` register.

DMA Transfer Types

The DMA controller supports various types of DMA transfers based on the primary and alternate control data structure and number of requests required to complete a DMA transfer. The DMA transfer type is configured by programming the appropriate values to the `cycle_ctrl` bits in the `CHNL_CFG` location of the control data structure.

Based on the `cycle_ctrl` bits, the following DMA transfer types are supported:

- Invalid
- Basic
- Auto-request
- Ping-Pong
- Memory Scatter-Gather
- Peripheral Scatter-Gather

DMA Transfer Type Usage table shows the various DMA transfer types that should be used by the peripherals and software DMA. Use of software DMA transfer types for peripherals and vice versa is not recommended.

Table 11-6: DMA Transfer Type Usage

Software	Peripheral
Basic	Auto Request
Ping-Pong	Ping-Pong*1
Peripheral Scatter-Gather	Memory Scatter-Gather

*1 Software Ping-Pong DMA transfer operation is different from Peripheral Ping-Pong DMA transfer operation.

Invalid

This means that no DMA transfer is enabled for the channel. After the controller completes a DMA cycle, it sets the cycle type to invalid to prevent it from repeating the same DMA cycle. Reading an invalid descriptor for any channel generates a DMA error interrupt and the corresponding bit in `DMA_INVALIDDESC_CLR` is set.

Basic

In this mode, the controller can be configured to use either primary or alternate data structure by programming `DMA_ALT_CLR` register for the corresponding channel. This mode is used for peripheral DMA transfers. While using this mode for peripherals, the `R_Power` in `CHNL_CFG` must be set to 0, which means the peripheral needs to present a request for every data transfer. Once the channel is enabled, when the controller receives a request, it performs the following operations:

1. The controller performs a transfer. If the number of transfers remaining is zero, the flow continues at *Step 3.*
2. The controller arbitrates:

   a. If a higher priority channel is requesting service, the controller services that channel, or
   b. If the peripheral or software signals a request to the controller, it continues at *Step 1*.
3. At the end of the transfer, the controller interrupts the MCU using `dma_done` interrupt for the corresponding channel.

Auto-Request

In this mode, the controller must receive single request to enable the DMA controller to complete the entire DMA cycle. This allows a large data transfer to occur, without significantly increasing the latency for servicing higher priority requests, or requiring multiple requests from the MCU or peripheral. This mode is useful for a memory-to-memory copy application using software DMA requests.

In this mode, the controller can be configured to use either primary or alternate data structure by programming the `DMA_ALT_CLR` register for the corresponding channel.

After the channel is enabled, when the controller receives a request, it performs the following operations.

1. The controller performs, min ($2^{R - Power}$, N) transfers for the channel. If the number of transfers remaining is zero, the flow continues at *Step 3*.
2. A request for the channel is automatically generated. The controller arbitrates. If the channel has the highest priority, the DMA cycle continues at *Step 1*.
3. At the end of the transfer, the controller interrupts the MCU using the `dma_done` interrupt for that channel.

Ping-Pong

In this mode, the controller performs a DMA cycle using one of the data structures and then performs a DMA cycle using the other data structure. The controller continues to switch between primary and alternate, until it either reads a data structure that is basic/auto-request, or the MCU disables the channel.

This mode is useful for transferring data using different buffers in the memory. In a typical application, the MCU is required to configure both primary and alternate data structure before starting the transfer. As the transfer progresses, the host can subsequently configure primary or alternate control data structures in the interrupt service routine when corresponding transfer ends.

The DMA controller interrupts the MCU using the `dma_done` interrupt after the completion of transfers associated with each control data structure. The individual transfers using either the primary or the alternate control data structures work exactly the same as a basic DMA transfer.

Software Ping-Pong DMA Transfer

In this mode, if the DMA request is from software, request is generated automatically after each arbitration cycle until completion of primary or alternate descriptor tasks. This final descriptor must use an auto-request transfer type.

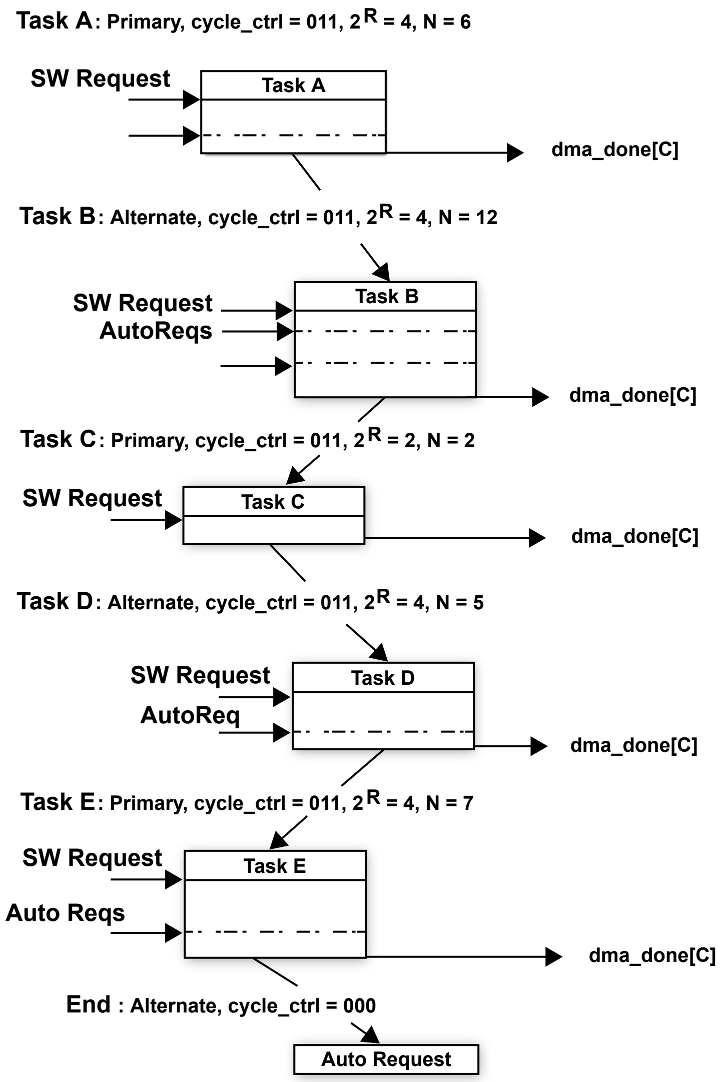

Task A: Primary, cycle_ctrl = 011, 2^R = 4, N = 6
SW Request
Task A
dma_done[C]
Task B: Alternate, cycle_ctrl = 011, 2^R = 4, N = 12
SW Request
AutoReqs
Task B
dma_done[C]
Task C: Primary, cycle_ctrl = 011, 2^R = 2, N = 2
SW Request
Task C
dma_done[C]
Task D: Alternate, cycle_ctrl = 011, 2^R = 4, N = 5
SW Request
AutoReq
Task D
dma_done[C]
Task E: Primary, cycle_ctrl = 011, 2^R = 4, N = 7
SW Request
Auto Reqs
Task E
dma_done[C]
End : Alternate, cycle_ctrl = 000
Auto Request

Figure 11-2: Software Ping-Pong DMA Transfer

Peripheral Ping-Pong DMA Transfer

In this mode, if the DMA request is from a peripheral, it needs to send DMA requests after every data transfer to complete primary or alternate descriptor tasks and the final descriptor must be programmed to use a basic transfer type.

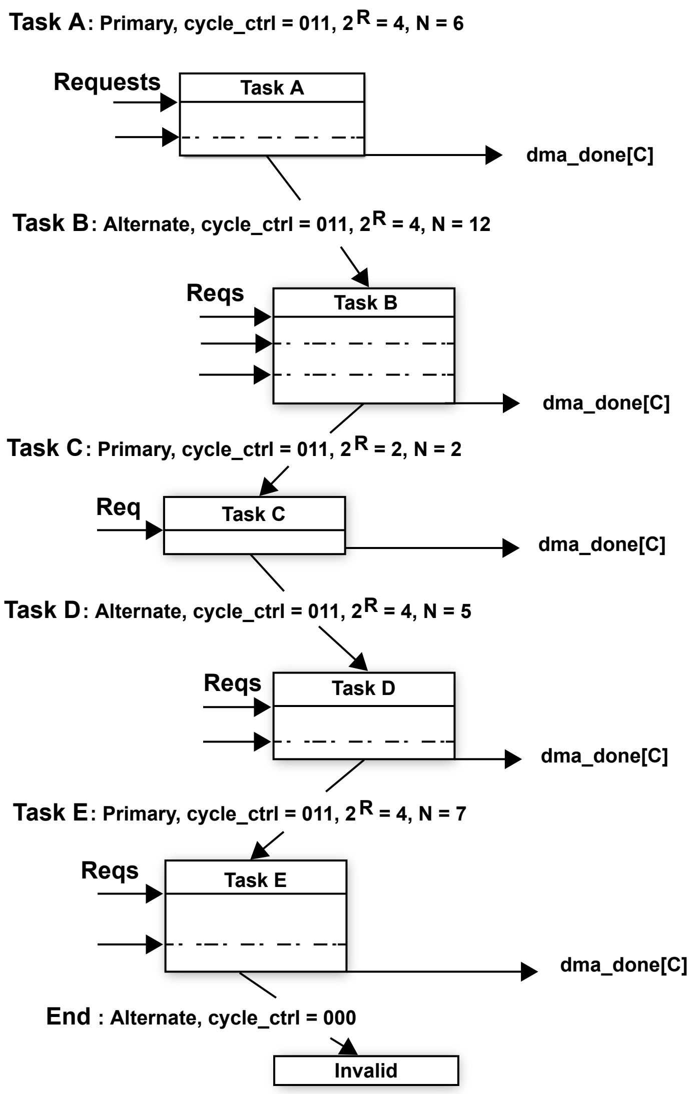

Task A: Primary, cycle_ctrl = 011, 2^R = 4, N = 6
Requests
Task A
dma_done[C]
Task B: Alternate, cycle_ctrl = 011, 2^R = 4, N = 12
Reqs
Task B
dma_done[C]
Task C: Primary, cycle_ctrl = 011, 2^R = 2, N = 2
Req
Task C
dma_done[C]
Task D: Alternate, cycle_ctrl = 011, 2^R = 4, N = 5
Reqs
Task D
dma_done[C]
Task E: Primary, cycle_ctrl = 011, 2^R = 4, N = 7
Reqs
Task E
dma_done[C]
End : Alternate, cycle_ctrl = 000
Invalid

Figure 11-3: Peripheral Ping-Pong DMA Transfer

Memory Scatter-Gather

In this mode, the DMA controller needs to be configured to use both primary and alternate data structures. It uses the primary data structure to program the control configuration for alternate data structure. The alternate data structure is used for data transfers using a transfer similar to an auto-request DMA transfer. The controller arbitrates after every primary transfer. The controller only needs one request to complete the entire transfer. This mode is used when performing multiple memory-to-memory copy tasks. The MCU can configure all of them together and is not required to intervene in-between. The DMA controller interrupts the MCU using the `dma_done` interrupt when the entire scatter-gather transaction is completed using a basic cycle.

In this mode, the controller receives an initial request and then performs four DMA transfers using the primary data structure to program the control structure of the alternate data structure. After this transfer is completed, it starts a

DMA cycle using the alternate data structure. After the cycle is completed, the controller performs another four DMA transfers using the primary data structure.

The controller continues to switch from primary to alternate to primary, until:

- The MCU configures the alternate data structure for a basic cycle,

  Or

- The DMA reads an invalid data structure.

Table 11-7: CHNL_CFG for Primary Data Structure in Memory Scatter-Gather Mode

Bit	Name	Value	Description
31:30	DST_INC	10	Configures the controller to use word increments for the address.
29:28	RESERVED	00	Undefined. Write as zero.
27:26	SRC_INC	10	Configures the controller to use word increments for the address.
25:24	SRC_SIZE	10	Configures the controller to use word transfers.
23:18	RESERVED	00	Undefined. Write as zero.
17:14	R_power	0010	Indicates that the DMA controller will perform four transfers.
13:4	N_minus_1	User	Configures the controller to perform N DMA transfers, where N is a multiple of 4.
3	RESERVED	00	Undefined. Write as zero.
2:0	cycle_ctrl	100	Configures the controller to perform a memory scatter-gather DMA cycle.

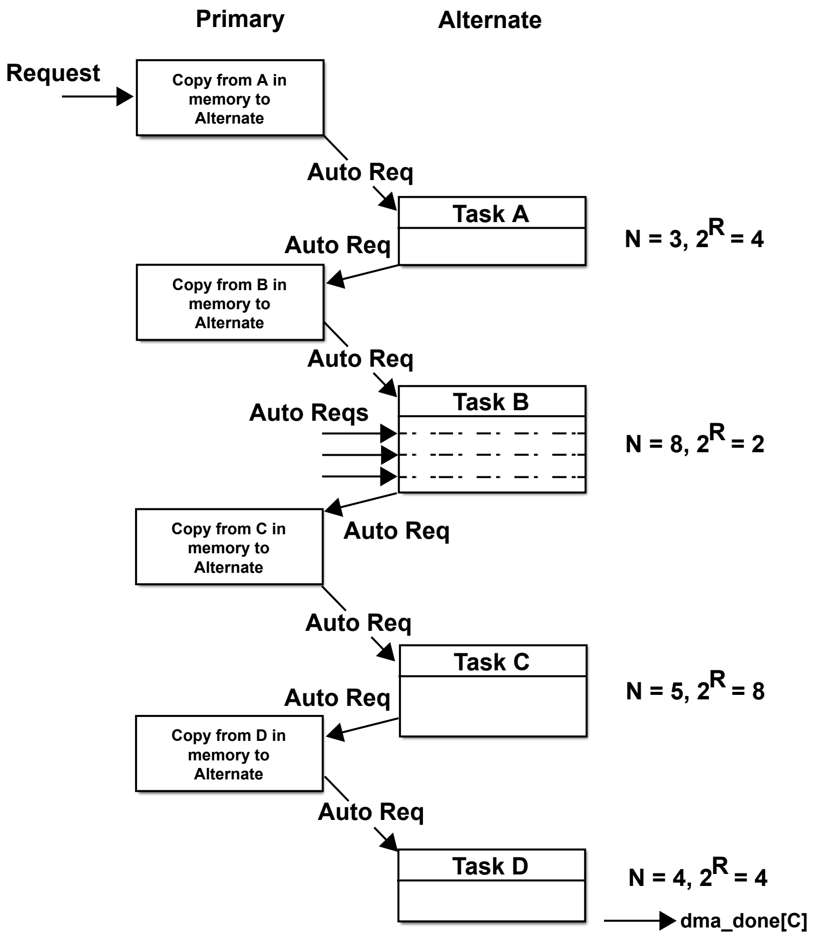


Figure 11-4: Memory Scatter-Gather Transfer Process

Peripheral Scatter-Gather

In this mode, the DMA controller should be configured to use both primary and alternate data structures. The controller uses the primary data structure to program the control structure of the alternate data structure. The alternate data structure is used for actual data transfers and each transfer takes place using the alternate data structure with a basic DMA transfer. The controller does not arbitrate after every primary transfer. This mode is used when there are multiple peripheral-to-memory DMA tasks to be performed. The MCU can configure all of them at the same time and need not intervene in between.

This mode is similar to memory scatter-gather mode except for arbitration and request requirements. The DMA controller interrupts the MCU using the `dma_done` interrupt when the entire scatter-gather transaction is completed using a basic cycle.

In peripheral scatter-gather mode, the controller receives an initial request from a peripheral and then performs four DMA transfers using the primary data structure to program the alternate control data structure. It then immediately starts a DMA cycle using the al-ternate data structure, without re-arbitrating.

After this cycle is completed, the controller re-arbitrates and if it receives a request from the peripheral and this has the highest priority then it performs another four DMA transfers using the primary data structure. It then immediately starts a DMA cycle using the al-ternate data structure without re-arbitrating.

The controller continues to switch from primary to alternate to primary until:

- The MCU configures the alternate data structure for a basic cycle,

  Or

- The DMA reads an invalid data structure

Table 11-8: CHNL_CFG for Primary Data Structure in Peripheral Scatter-Gather Mode

Bit	Name	Value	Description
31:30	DST_INC	10	Configures the controller to use word increments for the address.
29:28	RESERVED	00	Undefined. Write as zero.
27:26	SRC_INC	10	Configures the controller to use word increments for the address.
25:24	SRC_SIZE	10	Configures the controller to use word transfers.
23:18	RESERVED	00	Undefined. Write as zero.
17:14	R_power	0010	Indicates that the DMA controller will perform four transfers.
13:4	N_minus_1	user	Configures the controller to perform N DMA transfers, where N is a multiple of four.
3	RESERVED	00	Undefined. Write as zero.
2:0	cycle_ctrl	110	Configures the controller to perform a memory scatter-gather DMA cycle.

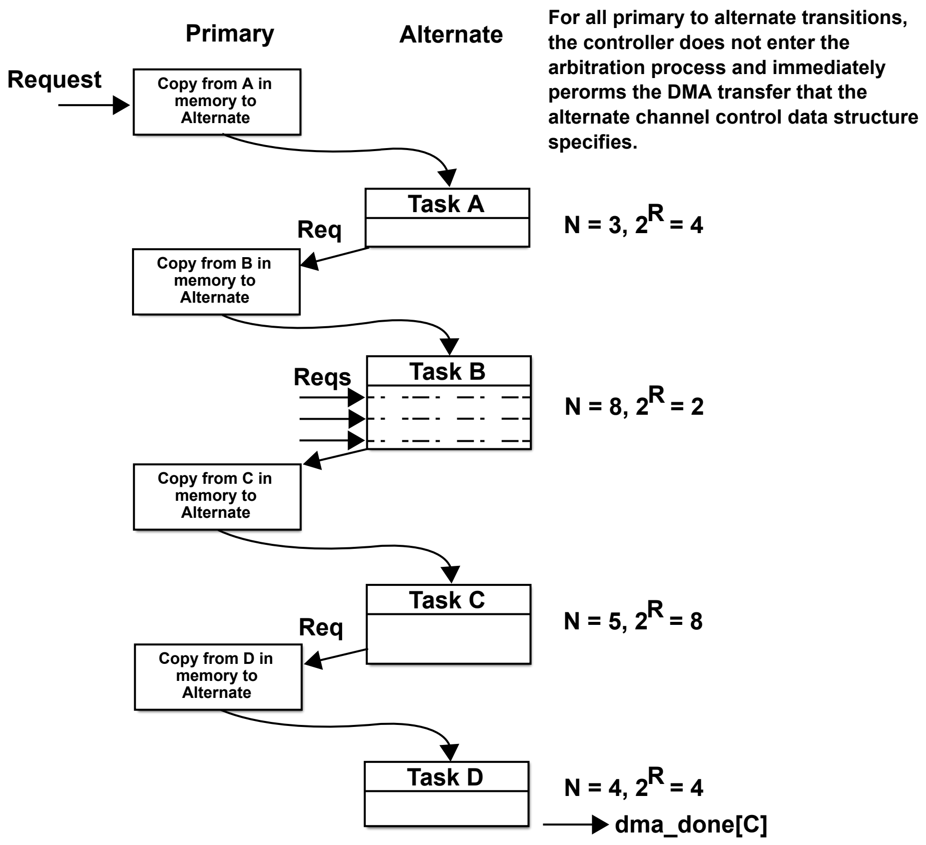

Primary
Alternate
For all primary to alternate transitions, the controller does not enter the arbitration process and immediately peroms the DMA transfer that the alternate channel control data structure specifies.
Request
Copy from A in memory to Alternate
Task A
Req
N = 3, 2^R = 4
Copy from B in memory to Alternate
Task B
Reqs
N = 8, 2^R = 2
Copy from C in memory to Alternate
Task C
Req
N = 5, 2^R = 8
Copy from D in memory to Alternate
Task D
N = 4, 2^R = 4
dma_done[C]

Figure 11-5: Peripheral Scatter-Gather Transfer Process

DMA Interrupts and Exceptions

Error Management

The DMA controller generates an error interrupt to the core when a DMA error occurs. A DMA error can occur due to a bus error or an invalid descriptor fetch. A bus error can occur while fetching the descriptor or performing a data transfer. A bus error can occur when a read from or write to a reserved address location happens.

When a bus error occurs, the faulty channel is automatically disabled, and the corresponding status bit in the DMA_ERRCHNL_CLR is set. If the DMA error is enabled in the NVIC, the error will also generate an interrupt. In addition, the CHNL_CFG data structure for the corresponding channel will be updated with the latest n_count. User can check this to know how many successful data transfers before the bus error.

When the controller fetches an invalid descriptor, the faulty channel is automatically disabled, and the corresponding status bit in the DMA_INVALIDDESC_CLR register is set. If the DMA error is enabled in NVIC, the error also generates an interrupt.

Address Calculation

The DMA controller calculates the source read address based on the SRC_END_PTR, the source address increment setting in CHNL_CFG, and the current value of the N_minus_1.

$$\text{Source read address} = \begin{cases} \text{SRC\_END\_PTR} - (\text{N\_minus\_1} \ll (\text{SRC\_INC})) & \text{for SRC\_INC} = 0, 1, 2 \\ \text{SRC\_END\_PTR} & \text{for SRC\_INC} = 3 \end{cases}$$

Similarly, the destination write address is calculated based on the `DST_END_PTR`, destination address increment setting in `CHNL_CFG`, and the current value of the `N_minus_1`.

$$\text{Destination write address} = \begin{cases} \text{DST\_END\_PTR} - (\text{N\_minus\_1} \ll (\text{DST\_INC})) & \text{for DST\_INC} = 0, 1, 2 \\ \text{DST\_END\_PTR} & \text{for DST\_INC} = 3 \end{cases}$$

`N_minus_1` is the current count of transfers to be done.

Address Decrement

Address decrement can be enabled to source and destination addresses. Source address decrement can be enabled for channels by setting appropriate bits in the `DMA_SRCADDR_SET` register. Similarly, destination address decrement can be enabled for channels by setting the required bits in the `DMA_DSTADDR_SET` register. The values written into the source data end pointer `(SRC_END_PTR)` and destination data end pointer (`DST_END_PTR`) are still used as the addresses for the last transfer as part of the DMA cycle. However, the start address will be computed differently to the address increment scheme for either source read or destination write.

If the source decrement bit is set in the `DMA_SRCADDR_SET` register for a channel, its source address is computed as follows:

$$\text{Source read address} = \begin{cases} \text{SRC\_END\_PTR} + (\text{N\_minus\_1} \ll (\text{SRC\_INC})) & \text{for SRC\_INC} = 0, 1, 2 \\ \text{SRC\_END\_PTR} & \text{for SRC\_INC} = 3 \end{cases}$$

If the destination decrement bit is set in the `DMA_DSTADDR_SET` register for a channel, its source address is computed as follows:

$$\text{Destination write address} = \begin{cases} \text{DST\_END\_PTR} + (\text{N\_minus\_1} \ll (\text{DST\_INC})) & \text{for DST\_INC} = 0, 1, 2 \\ \text{DST\_END\_PTR} & \text{for DST\_INC} = 3 \end{cases}$$

NOTES:

`N_minus_1` is the current count of transfers to be done.

Byte swap and address decrement should not be used together for any channel. If used together, DMA data transfer operation is unpredictable.

Image Decrement figure shows all the combinations of source and destination decrement with their data movement direction.

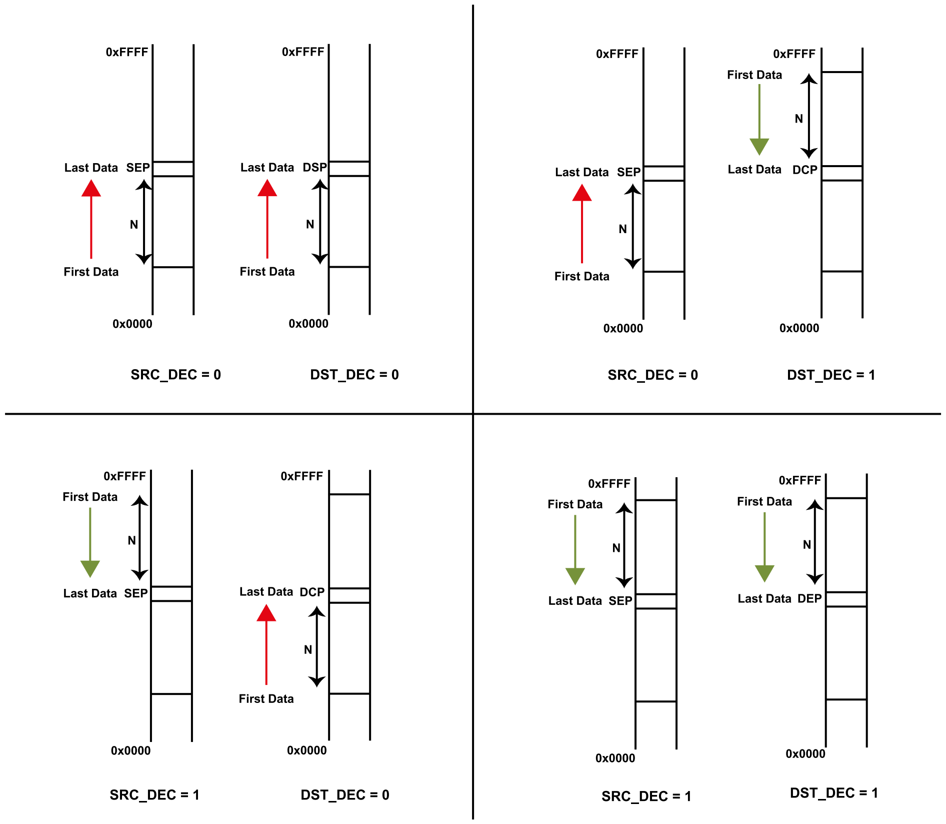


Figure 11-8: Image Decrement

Endian Operation

The DMA controller does the transfer by default using a little-endian approach; however, this default behavior can be changed by setting the corresponding channel bit in the `DMA_BS_SET` register. The endian operation is referred to as byte swap.

Byte Swap Disabled

Byte swap is disabled by default. In this case, the data transfer is considered to be little-endian. Data arriving from a peripheral is placed in sequence starting from the LSB of a 32-bit word.

For example, if 16 bytes of data arrive at the SPI as 0x01(start), 0x02, 0x03, 0x04 ... 0x0F, 0x10, it is stored by the DMA in memory as follows:

1. `04_03_02_01`
2. `08_07_06_05`
3. `0C_0B_0A_09`
4. `10_0F_0E_0D`

Byte Swap Enabled

Byte swap is enabled on any channel by setting the corresponding bit in the `DMA_BS_SET` register. By setting the bit, big-endian data transfers will occur. Data arriving from the peripheral will be placed in sequence starting from the MSB of a 32-bit word.

For example, if 16 bytes of data arrive at the SPI as 0x01(start), 0x02, 0x03, 0x04 ... 0x0F, 0x10, it will be stored by the DMA in memory as follows:

1. `01_02_03_04`
2. `05_06_07_08`
3. `09_0A_0B_0C`
4. `0D_0E_0F_10`

NOTES:

Byte swap happens on 32-bit data boundaries. The transfer size must be a multiple of 4.

Byte swap and address decrement should not be used together for any channel. If used together, DMA data transfer operation is unpredictable.

When using byte swap, care must be taken to ensure that the source data address is constant for the full data transfer. For example, SPI where the data source is always from the same location (a FIFO).

Byte swap functionality is independent of DMA transfer size and can be 8-bit, 16-bit, or 32-bit.

DMA Channel Enable/Disable

Before issuing a DMA request, the DMA channel should be enabled, otherwise, the DMA request for the corresponding channel is driven as DONE interrupt. Any DMA channel can be enabled by writing the corresponding bit in the `DMA_EN_SET` register. DMA controller disables the channel when the corresponding dma_done interrupt is generated. However, you can also disable any channel by writing to the corresponding bit in the `DMA_EN_CLR` register.

Whenever a channel is disabled, based on the current state of the DMA controller, it does the following:

- If the user disables the channel and there is no request pending for that channel, it is disabled immediately.
- If the user disables the channel that is not getting serviced, but its request is posted, its pending request will be cleared, and the channel is disabled immediately.
- If the user disables a channel that has been selected after arbitration but yet to start transfers, the controller completes the arbitration cycle and then disables channel.
- If the user disables the channel when it is getting serviced, controller completes the current arbitration cycle and then disables the channel.

DMA Master Enable

The `DMA_CFG.MEN` bit acts as a soft reset to the DMA controller. Any activity in the DMA controller can be performed only when this bit is set to 1. Clearing this bit to 0, clears all cached descriptors within the controller and resets the controller.

Power-down Considerations

Complete all on-going DMA transfers before powering down the chip to hibernate mode. However, if the user decides to hibernate as early as possible (current data transfers are ignored), the DMA controller must be disabled by clearing the `DMA_CFG.MEN` bit before going into hibernate mode.

NOTE: If hibernate mode is selected when a DMA transfer is in progress, the transfer discontinues. The DMA returns to the disabled state.

After hibernate (or POR), the DMA must be enabled again by setting the `DMA_CFG.MEN` bit.

The following registers are retained in hibernate mode:

- `DMA_PDBPTR`
- `DMA_ADBPTR`
- `DMA_RMSK_SET`
- `DMA_RMSK_CLR`
- `DMA_PRI_SET`
- `DMA_PRI_CLR`
- `DMA_BS_SET`
- `DMA_BS_CLR`
- `DMA_SRCADDR_SET`
- `DMA_SRCADDR_CLR`
- `DMA_DSTADDR_SET`
- `DMA_DSTADDR_CLR`

DMA Programming Model

The following section describes the programming sequence to configure the DMA.

Programming Guidelines

1. Enable the DMA controller in the `DMA_CFG` register.
2. Enable the desired DMA channel.

3. Setup the DMA base pointer.
4. Setup the DMA descriptor for data transmission.
5. Generate the software DMA request in the `DMA_SWREQ` register.

ADuCM302x DMA Register Descriptions

DMA contains the following registers.

Table 11-9: ADuCM302x DMA Register List

Name	Description
DMA_ADBPTR	DMA Channel Alternate Control Database Pointer
DMA_ALT_CLR	DMA Channel Primary Alternate Clear
DMA_ALT_SET	DMA Channel Primary Alternate Set
DMA_BS_CLR	DMA Channel Bytes Swap Enable Clear
DMA_BS_SET	DMA Channel Bytes Swap Enable Set
DMA_CFG	DMA Configuration
DMA_DSTADDR_CLR	DMA Channel Destination Address Decrement Enable Clear
DMA_DSTADDR_SET	DMA Channel Destination Address Decrement Enable Set
DMA_EN_CLR	DMA Channel Enable Clear
DMA_EN_SET	DMA Channel Enable Set
DMA_ERRCHNL_CLR	DMA per Channel Error Clear
DMA_ERR_CLR	DMA Bus Error Clear
DMA_INVALIDDESC_CLR	DMA per Channel Invalid Descriptor Clear
DMA_PDBPTR	DMA Channel Primary Control Database Pointer
DMA_PRI_CLR	DMA Channel Priority Clear
DMA_PRI_SET	DMA Channel Priority Set
DMA_REVID	DMA Controller Revision ID
DMA_RMSK_CLR	DMA Channel Request Mask Clear
DMA_RMSK_SET	DMA Channel Request Mask Set
DMA_SRCADDR_CLR	DMA Channel Source Address Decrement Enable Clear
DMA_SRCADDR_SET	DMA Channel Source Address Decrement Enable Set
DMA_STAT	DMA Status
DMA_SWREQ	DMA Channel Software Request

DMA Channel Alternate Control Database Pointer

The DMA_ADBPTR read-only register returns the base address of the alternate channel control data structure. This register removes the necessity for application software to calculate the base address of the alternate data structure. This register cannot be read when the DMA controller is in the reset state.

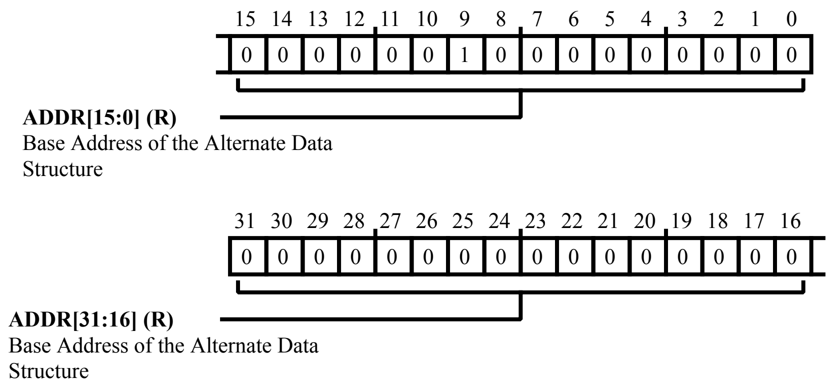


Figure 11-7: DMA_ADBPTR Register Diagram

Table 11-10: DMA_ADBPTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Base Address of the Alternate Data Structure.

DMA Channel Primary Alternate Clear

The DMA_ALT_CLR write-only register enables the user to configure the appropriate DMA channel to use the primary control data structure. Each bit of the register represents the corresponding channel number in the DMA controller.

Note: The DMA controller sets/clears these bits automatically as necessary for ping-pong, memory scatter-gather and peripheral scatter-gather transfers.

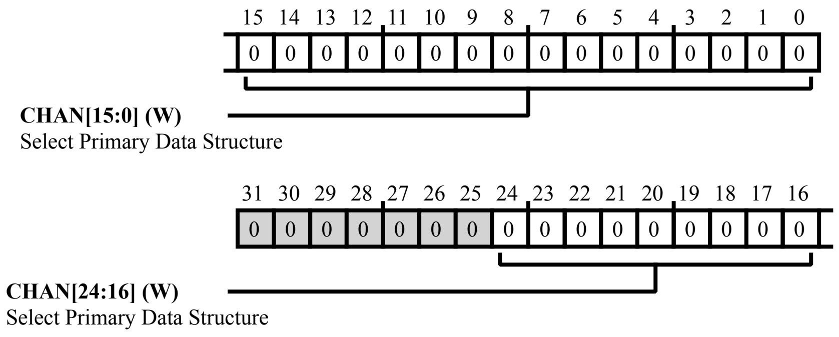


Figure 11-8: DMA_ALT_CLR Register Diagram

Table 11-11: DMA_ALT_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:0 (RX/W)	CHAN	Select Primary Data Structure. Set the appropriate bit to select the primary data structure for the corresponding DMA channel. Bit 0 corresponds to DMA channel 0 Bit M-1 corresponds to DMA channel M-1 When Written: DMA_ALT_CLR.CHAN[C] = 0, No effect. Use the DMA_ALT_SET register to select the alternate data structure. DMA_ALT_CLR.CHAN[C] = 1, Selects the primary data structure for channel C.

DMA Channel Primary Alternate Set

The DMA_ALT_SET register enables the user to configure the appropriate DMA channel to use the alternate control data structure. Reading the register returns the status of which data structure is in use for the corresponding DMA channel. Each bit of the register represents the corresponding channel number in the DMA controller.

Note: The DMA controller sets/clears these bits automatically as necessary for ping-pong, memory scatter-gather and peripheral scatter-gather transfers.

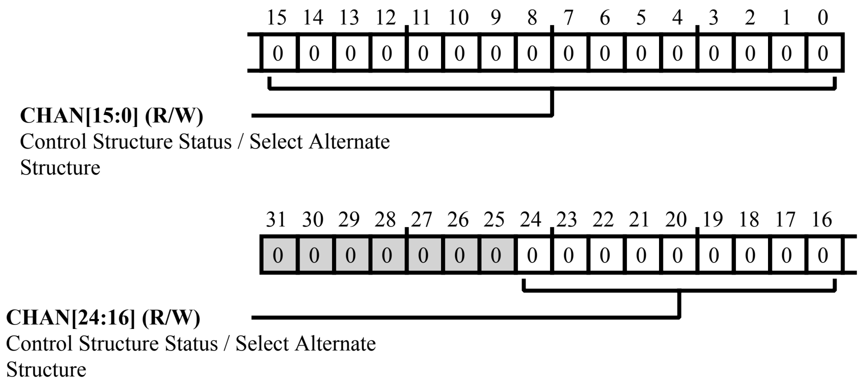


Figure 11-9: DMA_ALT_SET Register Diagram

Table 11-12: DMA_ALT_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:0 (R/W)	CHAN	Control Structure Status / Select Alternate Structure. Returns the channel control data structure status, or selects the alternate data structure for the corresponding DMA channel. Bit 0 corresponds to DMA channel 0 Bit M-1 corresponds to DMA channel M-1 When Read: DMA_ALT_SET.CHAN[C] = 0, DMA channel C is using the primary data structure. DMA_ALT_SET.CHAN[C] = 1, DMA channel C is using the alternate data structure. When Written: DMA_ALT_SET.CHAN[C] = 0, No effect. Use the DMA_ALT_CLR register to set bit [C] to 0. DMA_ALT_SET.CHAN[C] = 1, Selects the alternate data structure for channel C.

DMA Channel Bytes Swap Enable Clear

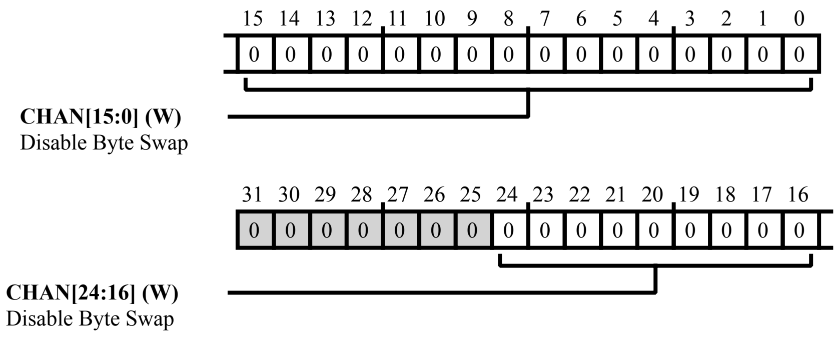


Figure 11-10: DMA_BS_CLR Register Diagram

Table 11-13: DMA_BS_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:0 (RX/W)	CHAN	Disable Byte Swap. The `DMA_BS_CLR` write-only register enables the user to configure a DMA channel to not use byte swapping and use the default operation. Each bit of the register represents the corresponding channel number in the DMA controller. Bit 0 corresponds to DMA channel and bit M-1 corresponds to DMA channel M-1. When Written: `DMA_BS_CLR.CHAN`[C] = 0, No effect. Use the `DMA_BS_SET` register to enable byte swap on channel C. `DMA_BS_CLR.CHAN`[C] = 1, Disables Byte Swap on channel C.

DMA Channel Bytes Swap Enable Set

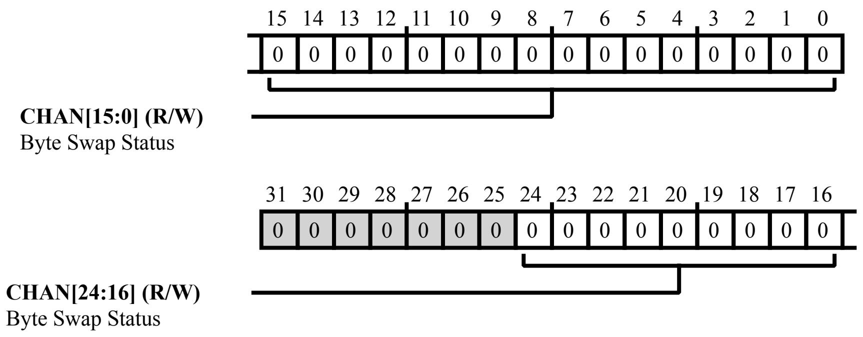


Figure 11-11: DMA_BS_SET Register Diagram

Table 11-14: DMA_BS_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:0 (R/W)	CHAN	Byte Swap Status. This register is used to configure a DMA channel to use byte swap. Each bit of the register represents the corresponding channel number in the DMA controller. Bit 0 corresponds to DMA channel 0 and bit M-1 corresponds to DMA channel M-1. When Read: `DMA_BS_SET.CHAN`[C] = 0, Channel C Byte Swap is disabled. `DMA_BS_SET.CHAN`[C] = 1, Channel C Byte Swap is enabled. When Written: `DMA_BS_SET.CHAN`[C] = 0, No effect. Use the `DMA_BS_CLR` register to disable byte swap on channel C `DMA_BS_SET.CHAN`[C] = 1, Enables Byte Swap on channel C.

DMA Configuration

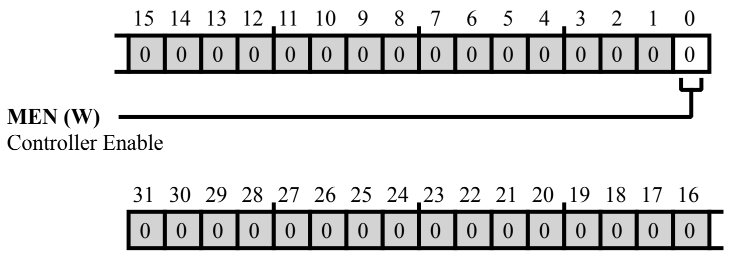


Figure 11-12: DMA_CFG Register Diagram

Table 11-15: DMA_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (RX/W)	MEN	Controller Enable.	
		0	Disable controller
		1	Enable controller

DMA Channel Destination Address Decrement Enable Clear

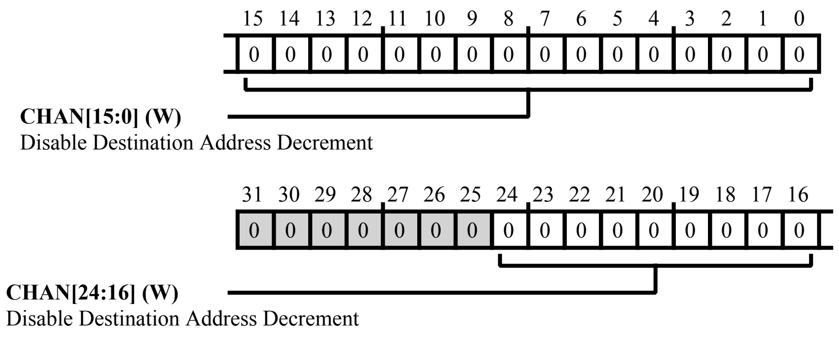


Figure 11-13: DMA_DSTADDR_CLR Register Diagram

Table 11-16: DMA_DSTADDR_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:0 (RX/W)	CHAN	Disable Destination Address Decrement. The `DMA_DSTADDR_CLR` write-only register enables the user to configure a DMA channel to use the default destination address in increment mode. Each bit of the register represents the corresponding channel number in the DMA controller. Bit 0 corresponds to DMA channel 0 Bit M-1 corresponds to DMA channel M-1 When Written: `DMA_DSTADDR_CLR.CHAN`[C] = 0, No effect. Use the `DMA_DSTADDR_SET` register to enable destination address decrement on channel C. `DMA_DSTADDR_CLR.CHAN`[C] = 1, Disables destination address decrement on channel C.

DMA Channel Destination Address Decrement Enable Set

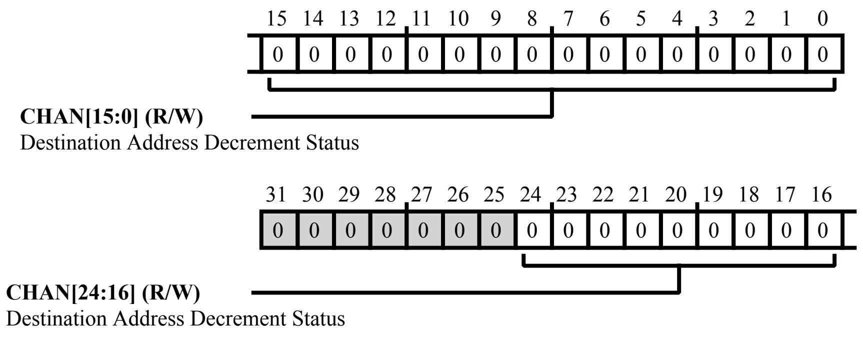


Figure 11-14: DMA_DSTADDR_SET Register Diagram

Table 11-17: DMA_DSTADDR_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:0 (R/W)	CHAN	Destination Address Decrement Status. The DMA_DSTADDR_SET register is used to configure the destination address of a DMA channel to decrement the address instead of incrementing the address after each access. Each bit of the register represents the corresponding channel number in the DMA controller. Bit 0 corresponds to DMA channel and bit M-1 corresponds to DMA channel M-1. When Read: DMA_DSTADDR_SET.CHAN[C] = 0, Channel C destination address decrement is disabled. DMA_DSTADDR_SET.CHAN[C] = 1, Channel C destination address decrement is enabled. When Written: DMA_DSTADDR_SET.CHAN[C] = 0, No effect. Use the DMA_DSTADDR_CLR register to disable destination address decrement on channel C. DMA_DSTADDR_SET.CHAN[C] = 1, Enables destination address decrement on channel C.

DMA Channel Enable Clear

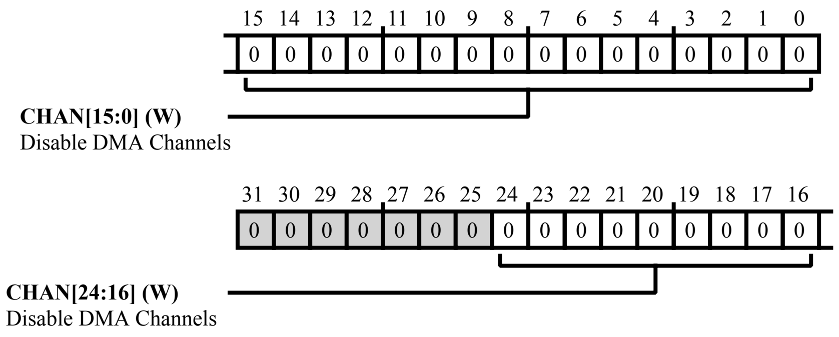


Figure 11-15: DMA_EN_CLR Register Diagram

Table 11-18: DMA_EN_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:0 (RX/W)	CHAN	Disable DMA Channels. This register allows for the disabling of DMA channels. This register is write only. Each bit of the register represents the corresponding channel number in the DMA controller. Note: The controller disables a channel automatically, by setting the appropriate bit, when it completes the DMA cycle. Set the appropriate bit to disable the corresponding channel. Bit 0 corresponds to DMA channel 0 Bit M-1 corresponds to DMA channel M-1 When Written: DMA_EN_CLR.CHAN[C] = 0, No effect. Use the DMA_EN_SET register to enable the channel. DMA_EN_CLR.CHAN[C] = 1, Disables channel C.

DMA Channel Enable Set

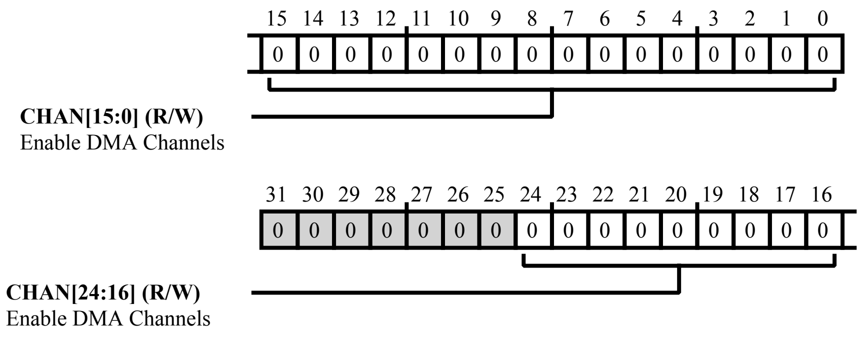


Figure 11-16: DMA_EN_SET Register Diagram

Table 11-19: DMA_EN_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:0 (R/W)	CHAN	Enable DMA Channels. This register allows for the enabling of DMA channels. Reading the register returns the enable status of the channels. Each bit of the register represents the corresponding channel number in the DMA controller. Set the appropriate bit to enable the corresponding channel. Bit 0 corresponds to DMA channel 0 Bit M-1 corresponds to DMA channel M -1 When Read: `DMA_EN_SET.CHAN`[C] = 0, Channel C is disabled. `DMA_EN_SET.CHAN`[C] = 1, Channel C is enabled. When Written: `DMA_EN_SET.CHAN`[C] = 0, No effect. Use the `DMA_EN_CLR` register to disable the channel. `DMA_EN_SET.CHAN`[C] = 1, Enables channel C.

DMA per Channel Error Clear

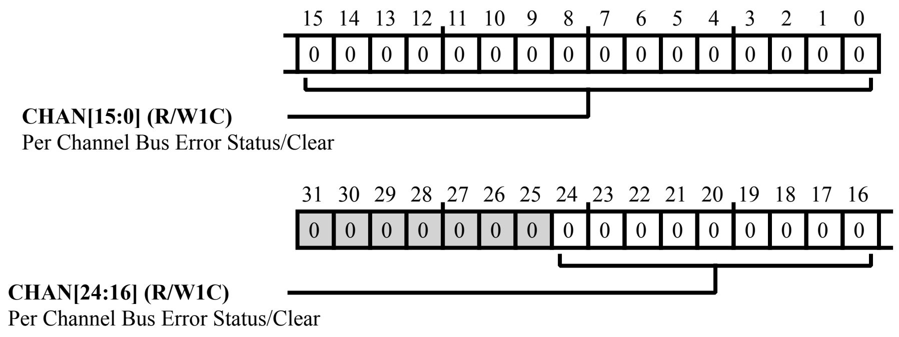


Figure 11-17: DMA_ERRCHNL_CLR Register Diagram

Table 11-20: DMA_ERRCHNL_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:0 (R/W1C)	CHAN	Per Channel Bus Error Status/Clear. This register is used to read and clear the per channel DMA bus error status. The error status is set if the controller encountered a bus error while performing a transfer. If a bus error occurs on a channel, that channel is automatically disabled by the controller. The other channels are unaffected. Write one to clear bits. When Read: 0: No bus error occurred. 1: A bus error control is pending. When Written: 0: No effect. 1: Bit is cleared.

DMA Bus Error Clear

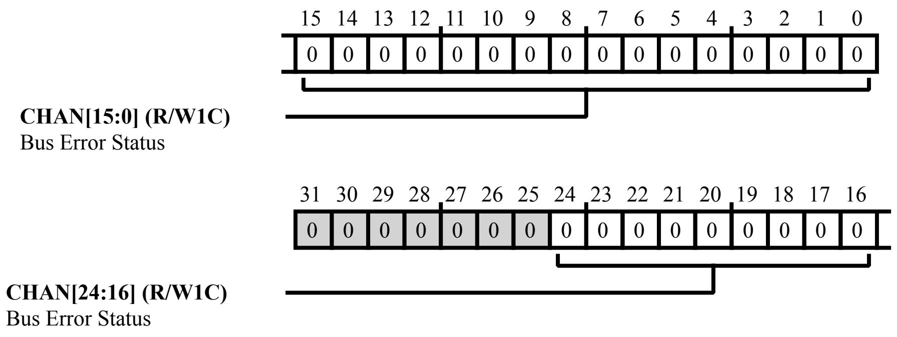


Figure 11-18: DMA_ERR_CLR Register Diagram

Table 11-21: DMA_ERR_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:0 (R/W1C)	CHAN	Bus Error Status. This register is used to read and clear the DMA bus error status. The error status is set if the controller encountered a bus error while performing a transfer or when it reads an invalid descriptor (whose cycle control is 3'b000). If a bus error occurs or invalid cycle control is read on a channel, that channel is automatically disabled by the controller. The other channels are unaffected. Write one to clear bits. When Read: 0: No bus_error/invalid cycle control occurred. 1: A bus_error/invalid cycle control is pending. When Written: 0: No effect. 1: Bit is cleared.

DMA per Channel Invalid Descriptor Clear

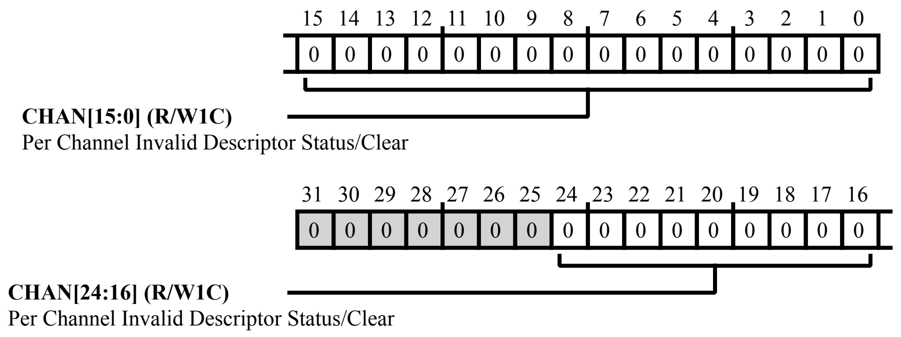


Figure 11-19: DMA_INVALIDDESC_CLR Register Diagram

Table 11-22: DMA_INVALIDDESC_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:0 (R/W1C)	CHAN	Per Channel Invalid Descriptor Status/Clear. This register is used to read and clear the per channel DMA invalid descriptor status. The per channel invalid descriptor status is set if the controller reads an invalid descriptor (cycle control is 3'b000). If it reads invalid cycle control for a channel, that channel is automatically disabled by the controller. The other channels are unaffected. Write one to clear bits. When Read: 0: No invalid cycle control occurred. 1: An invalid cycle control is pending. When Written: 0 No effect. 1 Bit is cleared.

DMA Channel Primary Control Database Pointer

The DMA_PDBPTR register must be programmed to point to the primary channel control base pointer in the system memory. The amount of system memory that must be assigned to the DMA controller depends on the number of DMA channels used and whether the alternate channel control data structure is used. This register cannot be read when the DMA controller is in the reset state.

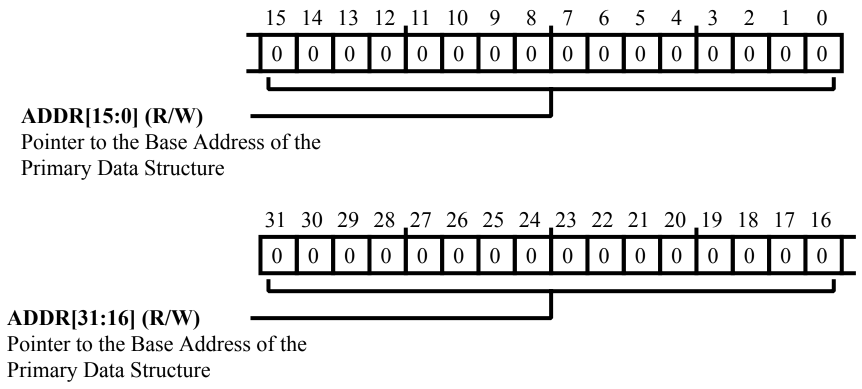


Figure 11-20: DMA_PDBPTR Register Diagram

Table 11-23: DMA_PDBPTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDR	Pointer to the Base Address of the Primary Data Structure. 5 + log(2) M LSBs are reserved and must be written 0. M is number of channels.

DMA Channel Priority Clear

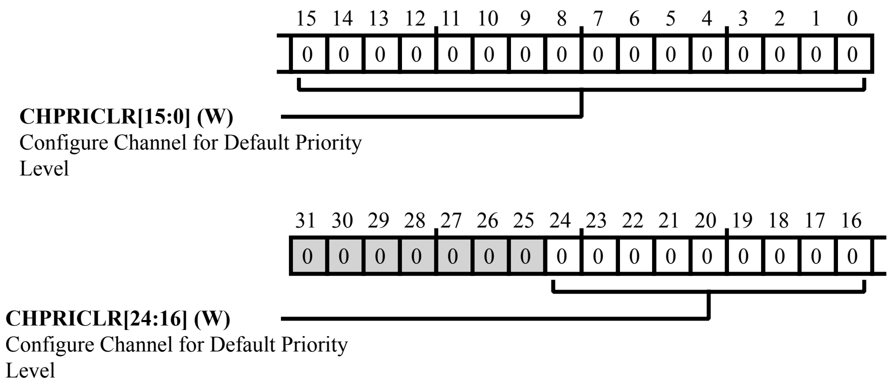


Figure 11-21: DMA_PRI_CLR Register Diagram

Table 11-24: DMA_PRI_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:0 (RX/W)	CHPRICLR	Configure Channel for Default Priority Level. The DMA_PRI_CLR write-only register enables the user to configure a DMA channel to use the default priority level. Each bit of the register represents the corresponding channel number in the DMA controller. Set the appropriate bit to select the default priority level for the specified DMA channel. Bit 0 corresponds to DMA channel and bit M-1 corresponds to DMA channel M-1. When Written: DMA_PRI_CLR.CHPRICLR[C] = 0, No effect. Use the DMA_PRI_SET register to set channel C to the high priority level. DMA_PRI_CLR.CHPRICLR[C] = 1, Channel C uses the default priority level.

DMA Channel Priority Set

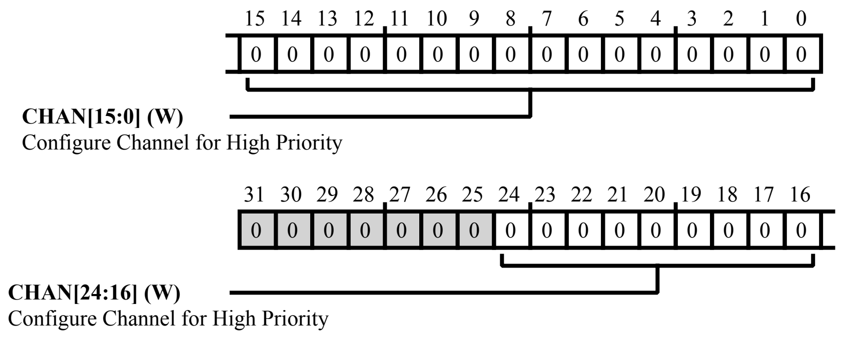


Figure 11-22: DMA_PRI_SET Register Diagram

Table 11-25: DMA_PRI_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:0 (RX/W)	CHAN	Configure Channel for High Priority. This register enables the user to you to configure a DMA channel to use the high priority level. Reading the register returns the status of the channel priority mask. Each bit of the register represents the corresponding channel number in the DMA controller. Returns the channel priority mask status, or sets the channel priority to high. Bit 0 corresponds to DMA channel 0, bit M-1 corresponds to DMA channel M-1. When Read: `DMA_PRI_SET.CHAN`[C] = 0, DMA channel C is using the default priority level. `DMA_PRI_SET.CHAN`[C] = 1, DMA channel C is using a high priority level. When Written: `DMA_PRI_SET.CHAN`[C] = 0, No effect. Use the `DMA_PRI_CLR` register to set channel C to the default priority level. `DMA_PRI_SET.CHAN`[C] = 1, Channel C uses the high priority level.

DMA Controller Revision ID

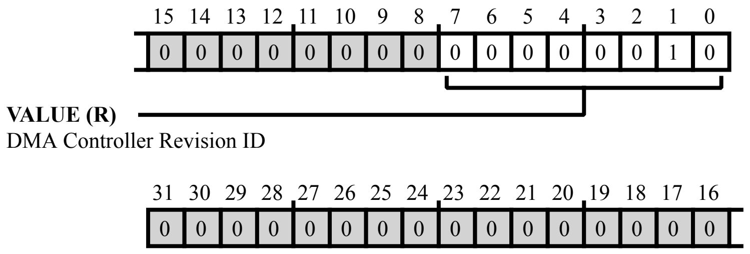


Figure 11-23: DMA_REVID Register Diagram

Table 11-26: DMA_REVID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	VALUE	DMA Controller Revision ID.

DMA Channel Request Mask Clear

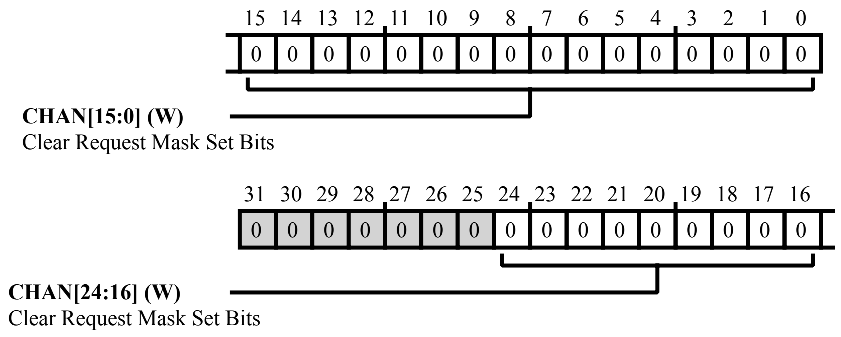


Figure 11-24: DMA_RMSK_CLR Register Diagram

Table 11-27: DMA_RMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:0 (RX/W)	CHAN	Clear Request Mask Set Bits. This register enables DMA requests from peripherals by clearing the mask set in `DMA_RMSK_SET` register. Each bit of the register represents the corresponding channel number in the DMA controller. Set the appropriate bit to clear the corresponding `DMA_RMSK_SET.CHAN` bit. Bit 0 corresponds to DMA channel 0 Bit M-1 corresponds to DMA channel M-1 When Written: `DMA_RMSK_CLR.CHAN`[C] = 0, No effect. Use the `DMA_RMSK_SET` register to disable DMA requests. `DMA_RMSK_CLR.CHAN`[C] = 1, Enables peripheral associated with channel C to generate DMA requests.

DMA Channel Request Mask Set

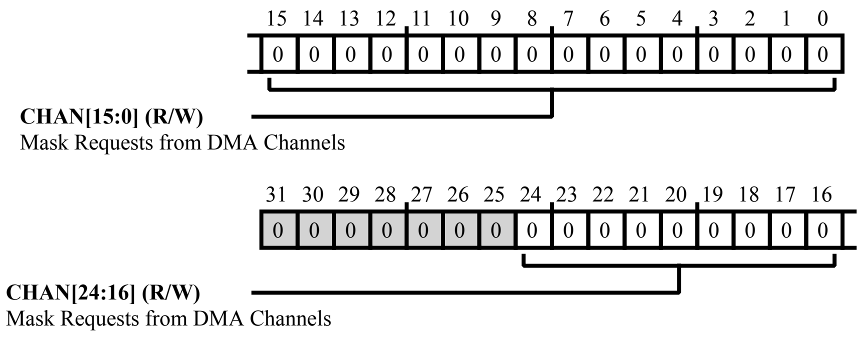


Figure 11-25: DMA_RMSK_SET Register Diagram

Table 11-28: DMA_RMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:0 (R/W)	CHAN	Mask Requests from DMA Channels. This register disables DMA requests from peripherals. Each bit of the register represents the corresponding channel number in the DMA controller. Set the appropriate bit to mask the request from the corresponding DMA channel. Bit 0 corresponds to DMA channel 0 Bit M-1 corresponds to DMA channel M-1 When Read: `DMA_RMSK_SET.CHAN`[C] = 0, Requests are enabled for channel C. `DMA_RMSK_SET.CHAN`[C] = 1, Requests are disabled for channel C. When Written: `DMA_RMSK_SET.CHAN`[C] = 0, No effect. Use the `DMA_RMSK_SET` register to enable DMA requests. `DMA_RMSK_SET.CHAN`[C] = 1, Disables peripheral associated with channel C from generating DMA requests.

DMA Channel Source Address Decrement Enable Clear

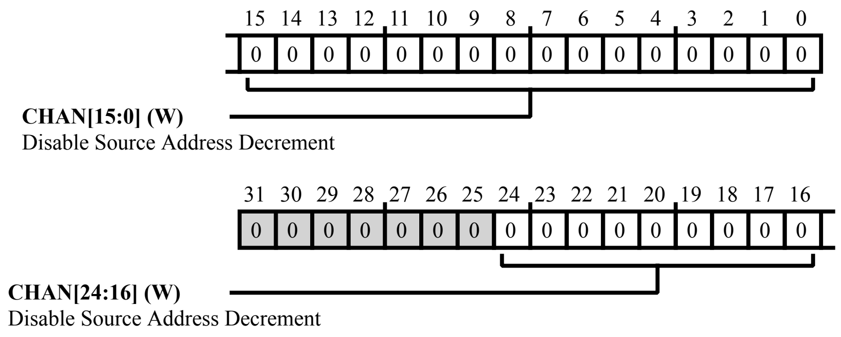


Figure 11-26: DMA_SRCADDR_CLR Register Diagram

Table 11-29: DMA_SRCADDR_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:0 (RX/W)	CHAN	Disable Source Address Decrement. The DMA_SRCADDR_CLR write-only register enables the user to configure a DMA channel to use the default source address in increment mode. Each bit of the register represents the corresponding channel number in the DMA controller. Bit 0 corresponds to DMA channel 0 Bit M-1 corresponds to DMA channel M-1 When Written: DMA_SRCADDR_CLR.CHAN[C] = 0, No effect. Use the DMA_SRCADDR_SET register to enable source address decrement on channel C. DMA_SRCADDR_CLR.CHAN[C] = 1, Disables Address source decrement on channel C.

DMA Channel Source Address Decrement Enable Set

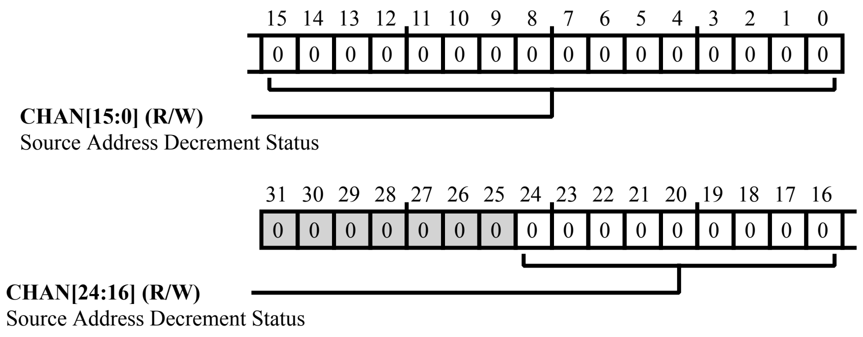


Figure 11-27: DMA_SRCADDR_SET Register Diagram

Table 11-30: DMA_SRCADDR_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:0 (R/W)	CHAN	Source Address Decrement Status. The `DMA_SRCADDR_SET` register is used to configure the source address of a DMA channel to decrement the address instead of incrementing the address after each access. Each bit of the register represents the corresponding channel number in the DMA controller. Bit 0 corresponds to DMA channel and bit M-1 corresponds to DMA channel M-1. When Read: `DMA_SRCADDR_SET.CHAN`[C] = 0, Channel C Source Address decrement is disabled. `DMA_SRCADDR_SET.CHAN`[C] = 1, Channel C Source Address decrement is enabled. When Written: `DMA_SRCADDR_SET.CHAN`[C] = 0, No effect. Use the `DMA_SRCADDR_CLR` register to disable source address decrement on channel C. `DMA_SRCADDR_SET.CHAN`[C] = 1, Enables source address decrement on channel C.

DMA Status

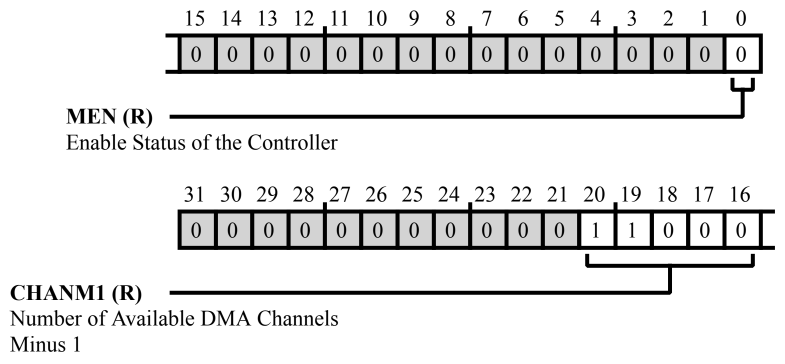


Figure 11-28: DMA_STAT Register Diagram

Table 11-31: DMA_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20:16 (R/NW)	CHANM1	Number of Available DMA Channels Minus 1. With 25 channels available, the register will read back 0x18.	
0 (R/NW)	MEN	Enable Status of the Controller.	
		0	Controller is disabled
		1	Controller is enabled

DMA Channel Software Request

The DMA_SWREQ register enables the generation of software DMA request. Each bit of the register represents the corresponding channel number in the DMA controller. M is the number of DMA channels

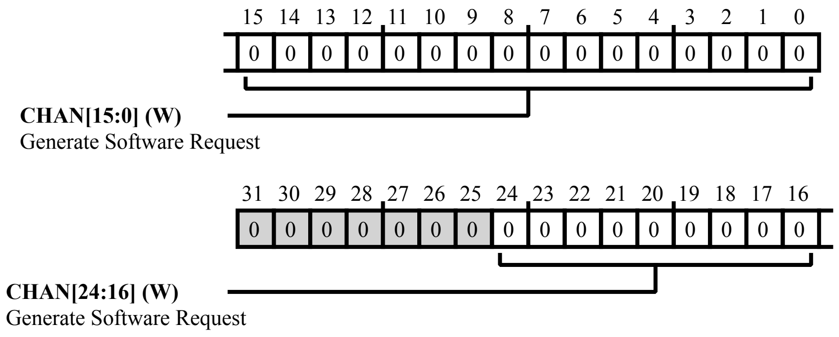


Figure 11-29: DMA_SWREQ Register Diagram

Table 11-32: DMA_SWREQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:0 (RX/W)	CHAN	Generate Software Request. Set the appropriate bit to generate a software DMA request on the corresponding DMA channel. Bit 0 corresponds to DMA channel 0 Bit M-1 corresponds to DMA channel M-1 When Written: DMA_SWREQ.CHAN[C] = 0, Does not create a DMA request for channel C. DMA_SWREQ.CHAN[C] = 1, Generates a DMA request for channel C. These bits are automatically cleared by the hardware after the corresponding software request completes.

12 Cryptography (CRYPTO)

Crypto is an accelerator block that supports the AES Cipher, NIST Block modes of operation and the SHA hash function generator.

Advanced Encryption Standard (AES) is the specification for a symmetric key algorithm to encrypt electronic data announced by the National Institute of Standards and Technology (NIST) in the year 2001 to replace the older DES Standard. The AES algorithm operates on a fixed data block of 128 bits.

Secure Hash Algorithm (SHA) is a hashing algorithm that works on messages parsed into 512-bit data blocks and generates a digest.

This document assumes familiarity with the operation of the AES standard, NIST block cipher modes of operation, and SHA algorithms. For more information, refer to the following publications.

Table 12-1: NIST Publication List

AES standard	Federal Information Processing Standard (FIPS) 197
NIST Block Modes of Operation	Special Publication-800-38A, B & C
SHA	FIPS 180-4

Crypto Features

The ADuCM302x MCU supports the following features:

- 32-bit APB Slave
- Two 128-bit buffers:
    - Input buffer
    - Output buffer
- Two DMA channels:
    - Input buffer
    - Output buffer
- AES Cipher Key Lengths supported:

- 128-bit
- 256-bit

- AES Cipher Key Source
    - Programmable through MMR
- Supported Modes:
    - AES Cipher Encryption and Decryption in ECB Mode
    - AES Cipher Encryption and Decryption in CBC Mode
    - AES Cipher Encryption and Decryption in CTR Mode
    - AES Cipher MAC Generation Mode
    - AES Cipher Encryption and Decryption in CCM Mode
    - AES Cipher Encryption and Decryption in CCM* Mode
    - SHA - 256 Mode

Crypto Functional Description

This section provides information on the function of the Crypto block used by the ADuCM302x MCU.

Crypto Block Diagram

The figure shows the Crypto block used by the ADuCM302x MCU.

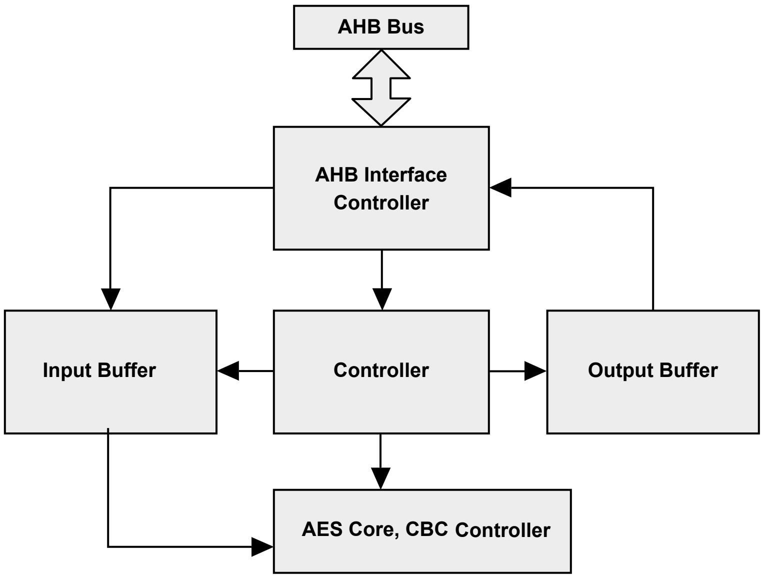


Figure 12-1: Crypto Block

Crypto is a 32-bit APB DMA capable peripheral. There are two buffers provided for data I/O operations. These buffers are 32-bit wide and will read in or read out 128-bits in four data accesses. Big Endian and Little Endian data formats are supported.

When enabled, the block takes the data from its input buffer and gives out the processed data through the output buffer. DMA can be used for performing these data operations. It is important for the programmer to ensure that the configuration bits, keys and other relevant registers are set to the desired values before the block is enabled and data transfer is initiated.

Crypto Operating Modes

The following block modes of operation are supported:

- CTR mode: Counter mode
- CBC mode: Cipher Block Chaining mode
- MAC mode: Message Authentication Code mode
- CCM mode: Cipher Block Chaining-Message Authentication Code mode
- ECB mode: Electronic Code Book mode

Mode enable bit fields in the `CRYPT_CFG` register must be set to enable the desired mode of operation. A particular block mode is enabled by setting the appropriate bit field to 1.

The programmer should enable only one block mode of operation at a given time. The bit fields for enabling different modes of operation are: `CRYPT_CFG.ECBEN`, `CRYPT_CFG.CTREN`, `CRYPT_CFG.CBCEN`, `CRYPT_CFG.CCMEN`, and `CRYPT_CFG.CMACEN`.

Enabling multiple block modes will cause unpredictable results. No error or warning is flagged.

Crypto Data Transfer

The input buffer and the output buffer hold 128-bits of data. Being a peripheral on a 32-bit APB bus, the block is accessed by a 32-bit data bus. Therefore, four reads/writes are required to empty/fill the output/input buffer.

In SHA mode, the results are updated in the `CRYPT_SHAH0.SHAHASH0` to `CRYPT_SHAH7.SHAHASH7` registers, and are not given out in the output buffer. Results for block modes of operation are streamed out of the output buffer.

Crypto Data Rate

Depending on the Block Mode and the Key length chosen for the AES Cipher, the rate at which data is processed changes. The following list describes the computation time involved in different block modes of operation. The numbers below do not include the buffer read and write time which are controlled by the DMA or core data rates. The rates are written for 128/256 key length cases.

- ECB Mode: 40/56 clock cycles

- CBC Mode: 40/56 clock cycles
- CTR Mode: 40/56 clock cycles
- MAC Mode: 40/56 clock cycles
- CCM Mode: 80/102 clock cycles

In block mode of operation, as soon as 128 bits of data (say block B1) is written into the input buffer, the data is copied internally for computation and the next 128 bits of data (say block B2) may be written into the input buffer. By the time block B2 is written, if computation on the block B1 is not complete, the input buffer will remain full and further writes will not be permitted. Appropriate interrupts and status registers will be updated. As soon as block B1 processing is completed the results are moved to the output buffer and computation begins on the B2. If B1 is not read out of the output buffer by the time processing on B2 is completed, the block stalls. Appropriate interrupt and status registers will be set.

DMA Capability

The block supports use of DMA to transfer data to or from the buffers. The DMA is expected to do four word transfers per DMA request on both the input and output buffers. The DMA requesting for the buffers can be enabled by setting the `CRYPT_CFG.INDMAEN` and `CRYPT_CFG.OUTDMAEN` bits. The block raises the input buffer DMA request line high till the input buffer is filled. Similarly, the DMA request on the output buffer is held high till the output buffer is emptied.

The programmer must not change the configuration bits in the middle of a data transaction. Else, it may result in wrong output, and no error or warning is raised.

Core Transfer

If DMA is not used, the software is expected to keep track of the number of words that are to be transferred to the block and read back from the block. The bit fields that indicate the status and error conditions are available in the `CRYPT_STAT` register. The interrupts for different conditions can be enabled in the `CRYPT_INTEN` register.

Data Formating

This section explains how to format the input data and keys, and the format in which data can be expected from the output buffer.

Consider the following key, input data, and output data. The key, plain text, and cipher text are taken from the data mentioned in the NIST standard.

```
Key[0:15]: 2b7e151628aed2a6abf7158809cf4f3c
```

```
Plain text[0:15]: 6bc1bee22e409f96e93d7e117393172a
```

```
Cipher Text[0:15]: 3ad77bb40d7a3660a89ecaf32466ef97
```

This 28-bit key is mapped to the AESKEY registers as follows:

```
CRYPT_AESKEYi=Key[i*4+3:i*4];
```

For the above example,

```
CRYPT_AESKEY0[31:0] = 0x16157e2b;

CRYPT_AESKEY1[31:0] = 0xa6d2ae28;

CRYPT_AESKEY2[31:0] = 0x8815f7ab;

CRYPT_AESKEY3[31:0]=3c4fcf09;
```

The plain text must be fed to the input buffer via APB writes using DMA/Core as follows:

```
for (i=0; i<4; i++)

{

Input_buffer = plain_text[i*4+3:i*4];
}
```

The example plain text can be mapped to input buffer writes as follows:

```
write(Input_buffer, 0xe2bec16b);
write(Input_buffer, 0x 969f402e);
write(Input_buffer, 0x117e3de9);
write(Input_buffer, 0x2a179373);
```

The ciphertext block can be read from the output buffer in four consecutive reads.

```
for (i=0; i<4; i++)
{
 cipher_text[ i*4+3:i*4] = read(output_buffer);
}
```

Byte-Swap Configuration

Three configuration bits have been provided for byte swapping the data written to the crypto buffers and key registers.

A byteswap operation is defined as:

```
Input String: 0xB3B2B1B0
Ouput String: 0xB0B1B2B3
```

Endian

When this bit is set, it byte swaps the data written to the input buffer for block modes of operation and expected data out of the output buffer.

Interrupts

INPUT_READY_INT

This interrupt indicates that the input buffer is not full. If enabled, this triggers interrupts as long as the Input Buffer is not filled. As this interrupt is used when the data buffer is being written by the core, fill the input buffer from the ISR.

This interrupt is to be used with AES block cipher modes only and not with SHA.

OUTPUT_READY_INT

This interrupt indicates that the output buffer holds data and is waiting to be read. This will remain set as long as the output buffer is not empty. As this interrupt is used when the data buffer is being read by the core, read the entire 128-bits in the ISR.

This interrupt is to be used with AES block cipher modes only and not with SHA.

SHA_DONE

This interrupt indicates that the SHA computation is completed. New data may be written into the input buffer or that the hash results may be read out of the `CRYPT_SHAH0` to `CRYPT_SHAH7` registers.

IN_OVF_INT

This interrupt indicates indicates that the input buffer overflow event has occurred. The causes for this may be:

- The output buffer has not been read. This stalls the block.
- The input data rate is higher than the rate the Crypto block can process.

When the DMA or the core program is programmed as recommended in this manual, this interrupt must not be raised.

Crypto Error Conditions

If a read is attempted from output buffer when the output data buffer is empty, there is an underflow error. The block does not generate a DMA request on output buffer channel when output buffer is empty.

If the Crypto block is disabled at any stage during the computation, the data output is no longer reliable. The state machine is reset.

If a write is attempted to input buffer when it is full, there will be an overflow error. The block does not generate a DMA request on input buffer channel when input buffer is full.

The input and output buffers can be flushed by writing 1 to the `CRYPT_CFG.INFLUSH` and `CRYPT_CFG.OUTFLUSH` bits respectively.

Interrupts can be enabled by writing 1 into the corresponding bits in the `CRYPT_INTEN` register. Clearing interrupts is to be done by writing 1 to the respective bits in the `CRYPT_STAT` register.

Crypto Status Bits

IN_DATA_CNT/OUT_DATA_CNT

These fields hold information about the number of words the input and output buffers contain at that moment. This information can be used in situations when input/output buffers are not empty/full to know the number of words which need to be written or read.

HASH_READY

This field indicates that the hash computation on the currently provided data is complete and the block is ready to receive new data.

HASH_BUSY

This field indicates that hash computation is ongoing. When this signal goes low, it is an indication that hash computation is complete and that the user can read out the hash result.

IN_OVF_INT

This field indicates that an overflow event has occurred on the input buffer. The status bit is sticky and can be cleared by writing 1 into the bit field.

Crypto Keys

The user must program the key used for the AES Cipher in the key registers from `CRYPT_AESKEY0` to `CRYPT_AESKEY7`.

Key Length

The AES Cipher can be used with two different key lengths: 128 and 256. The selection is made by setting the `CRYPT_CFG.KEYLEN` bit. The registers `CRYPT_AESKEY0` to `CRYPT_AESKEY7` are used to hold the key to be used in the AES Cipher.

Key Programming

The following list shows how the key (K) used for the AES Cipher operations are constructed from the `CRYPT_AESKEYx` registers:

`CRYPT_AESKEY0` = K[31:0]

`CRYPT_AESKEY1` = K[63:32]

`CRYPT_AESKEY2` = K[95:64]

`CRYPT_AESKEY3` = K[127:96]

`CRYPT_AESKEY4` = K[159:128]. Used only in case of Key Length 256.

`CRYPT_AESKEY5` = K[191:160]. Used only in case of Key Length 256.

`CRYPT_AESKEY6` = K[223:192]. Used only in case of Key Length 256.

`CRYPT_AESKEY7` = K[255:224]. Used only in case of Key Length 256.

NOTE: If the programmer changes the configuration bits (for example, encrypt), the key register must be reprogrammed to the desired key value. Else, it results in wrong outputs. No warning or error is flagged.

Key registers cannot be programmed partially. If any of the 8 registers is overwritten, all the others register must be reset to zero and programmed again.

Crypto Power Saver Mode

Enabling the Power Saver mode in the `CRYPT_CFG.BLKEN` register enables an automatic clock gating function. This reduces the clocking activity in the block to the barest minimum possible. It is recommended to keep this on because there is not performance hit by having this enabled.

Registers with Mode Specific Roles

DATALEN

The `CRYPT_DATALEN` register is used to specify the length of the payload data. This information is used in CCM and MAC modes of operation.

- CCM Mode: Program the number of 128-bit data blocks in the resulting aligned payload. `CRYPT_DATALEN [15:0]` is provided for this purpose. CCM supports data lengths which are not a multiples of 16 bytes. The number of blocks must be rounded off to the next integer. For example, if the number of blocks in data is 4.5, the data length must be programmed to 5.
- CMAC Mode: Pad the payload data to ensure that the total number of bits is an integral multiple of 128. Program the number of 128-bit blocks in the aligned payload data. `CRYPT_DATALEN [19:0]` is provided for this purpose. For example, if the data length is 56 bytes, the data length must be programmed to 4.

CCM_NUM_VALID_BYTES

This field is to be programmed with the numberof valid bytes in the last data block for a CCM operation. Crypto block operates on 128-bit data at a time. User must pad the last data block with zeros and load it into the input buffer. These padded zeros do not form a part of data, and must be excluded from the data using the `CRYPT_CCM_NUM_VALID_BYTES` register. For example, if the data length is 56 bytes, `CRYPT_CCM_NUM_VALID_BYTES` must be programmed to 8.

PREFIXLEN

The `CRYPT_PREFIXLEN` register is used to specify the length of the associated data that is only authenticated. This information is used in the CCM mode of operation.

CCM Mode: Pad the associated data enough zeroes with to ensure that the total number of bits is an integral multiple of 128. Then, program the number of 128-bit blocks in the resulting aligned payload data. `CRYPT_PREFIXLEN.VALUE` bits are provided for this purpose.

NONCEx

The `CRYPT_NONCE0` to `CRYPT_NONCE3` registers are used to program the nonce for CTR, CBC, and CCM Modes of operation.

The Nonce is constructed in the following way:

Nonce[127:0] = {CRYPT_NONCE3[31:0], CRYPT_NONCE2[31:0], CRYPT_NONCE1[31:0], CRYPT_NONCE0[31 :0]}

Different lengths of nonce are used in different block modes of operation:

- CTR mode: This mode uses a 108-bit long nonce. In this mode, the encryption operation is performed in the following format:

  Ciphertext_Block = Ciph ({Nonce[107:0], CRYPT_CNTRINIT[19:0]) xor Payload_Block

- CBC mode: In this mode, 128 bits of the nonce are used. The nonce is used as the initialization vector.

  Initialization_Vector = Nonce[127:0]

- CCM mode: In this mode, 112 bits are used. The nonce is used in counter mode computation. CTR mode data is constructed in the following format:

  Ciphertext_Block = Ciph ({Nonce[111:0], Counter[15:0]}) xor Payload_Block

CNTRINIT

The value is used to initialize the counter generating function. The counter generating function implemented in this block is an incrementing function, i.e. this function counts up by 1 for every new data block. The initialization value for the counters can be programmed in the CRYPT_CNTRINIT register.

Counter widths for the CTR mode and CCM modes are described below:

- CTR mode: 20-bit counter. If counter starts from zero, then 16 GB of data will be processed before the counter overflows. A 20-bit register for initializing the counter is provided.
- CCM mode: 16-bit counter. If counter starts from zero, then 1 GB of data will be processed before the counter overflows. A 16-bit counter initialization register is provided.
- The fixed bit width of the counter implies that there is a limit to the maximum length of data that can be encrypted without repeating the same counter values. This is 16 GB of payload data in CTR mode and 1 GB of confidentiality data in CCM mode.

NOTE: When the counter exhausts all unique values it starts all over again from the init value. Ensure that the limit is not crossed if repletion of counter value needs to be avoided.

SHAINIT

This is an auto-clear register field. The control signal is to be used to initialize the SHA computation registers to reset value when a new computation is initiated

SHA_LAST_WORD

SHA256 module is updated to include a length counter and padding mask. Software does not need to append the padding and bit length at the end of a message.

This register has the following fields:

- CRYPT_SHA_LAST_WORD.O_BITS_VALID [4:0]

- `CRYPT_SHA_LAST_WORD.O_LAST_WORD`

Software loads the input data as a word (32 bits) to the crypto accelerator input buffer. The input data may or may not be a multiple of 32 bits. User must set the last_word bit, bits_valid bit to number of bits valid in the last word (0-31), and write the last word to the crypto accelerator. The LSBs of this word are ignored (replaced with padding) according to the SHA standard. If data length is a multiple of 32 bits, user must write a dummy word to complete the final operation (all bits are replaced with padding).

NOTE: For the last word in the message, the `CRYPT_SHA_LAST_WORD.O_LAST_WORD` must be asserted and `CRYPT_SHA_LAST_WORD.O_BITS_VALID` must reflect the number of valid bits in the `CRYPT_SHA_LAST_WORD`. If all the bits are valid, the word must be written first, and a dummy word (which is the last word) is written with no bits valid. Invalid bits are ignored.

Crypto Programming Model

The following section details the programming flow description for block modes of operation.

In general, all block modes of operation require the software to perform the following operations:

- Crypto Block Configuration
- Input Data Preparation
- Data Retrieval

The remaining part of this section lists the different registers that need to be programmed in different modes and additional information that would be necessary for the programmers.

Enabling Crypto

Set to 1 the `CRYPT_CFG.BLKEN` field.

Key Programming

1. Write the key to the `CRYPT_AESKEY0` to `CRYPT_AESKEY7` registers.
2. Set the `CRYPT_CFG.KEYLEN` field.

Data Transfer

1. Select the endianness in the `CRYPT_CFG.ENDIAN` field.
2. Enable the DMA channel request for input and ouput buffers by setting the `CRYPT_CFG.INDMAEN`/ `CRYPT_CFG.OUTDMAEN` fields

Block Mode of Operation

The Crypto block supports the following block modes of operation:

- AES Cipher Encryption and Decryption in ECB mode

- AES Cipher Encryption and Decryption in CBC mode
- AES Cipher Encryption and Decryption in CTR mode
- AES Cipher MAC Generation mode
- AES Cipher Encryption and Decryption in CCM mode
- AES Cipher Encryption and Decryption in CCM* mode

NOTE: Only one mode must be enabled at a time.

Mode Specific Parameters

CTR mode:

- `CRYPT_CNTRINIT`
- `CRYPT_NONCE0` to `CRYPT_NONCE3`

CCM/CCM* mode:

- `CRYPT_DATALEN`
- `CRYPT_PREFIXLEN`
- `CRYPT_CNTRINIT`
- `CRYPT_NONCE0` to `CRYPT_NONCE3`
- `CRYPT_CCM_NUM_VALID_BYTES`

CBC mode:

- Initialization Vector in the `CRYPT_NONCE0` to `CRYPT_NONCE3` registers.

MAC mode:

- `CRYPT_DATALEN`: This information is used by the block to determine when the MAC results must be provided.

SHA mode:

- `CRYPT_SHA_LAST_WORD`: Indicates the termination of SHA data input and number of valid bytes in the last input.

Payload and Associated Data Formatting

In every mode of operation, it is necessary to pad the payload and/or associated data with sufficient number of zeroes so as to make the length a multiple of 128 bits.

Mode Specific Data Format

- CBC mode: Initialize the `CRYPT_NONCE0` to `CRYPT_NONCE3` registers with the initialization vector.

- MAC mode: The block always gives out a 128-bit MAC result on the output buffer in the selected data format (Big_Endian or Litte_Endian). The software needs to ensure that the desired number of bits is used.
- Subkey generation: The values of K1 and K2 needs to be used to generate the final message DataBlock. The generation of K1, K2, padding the higher bits with ‘b10j (j = number of zeroes that on padding make the final message DataBlock 128-bit long) and the XOR operation involved in generating the last message DataBlock is to be handled in software.
- CCM/CCM* mode: Any formatting function that satisfies the following three properties can be used:
    - The first DataBlock uniquely determines the nonce. Note that the nonce will have to be programmed by the software into the `CRYPT_NONCE0` to `CRYPT_NONCE3` registers. The block will not pick it up from the incoming data stream directly.
    - The formatted data uniquely determines Payload and Associated Data. The lengths of the Payload and Associated Data must be programmed into the `CRYPT_DATALEN` and `CRYPT_PREFIXLEN` registers respectively.
    - The counter must be initialized to a unique value distinct from any other invocation with the same key.

NOTE: CCM* has additional formatting constraints which is left to the software to implement. Once the formatting is complete, the registers `CRYPT_DATALEN` and `CRYPT_PREFIXLEN` must contain the following information:

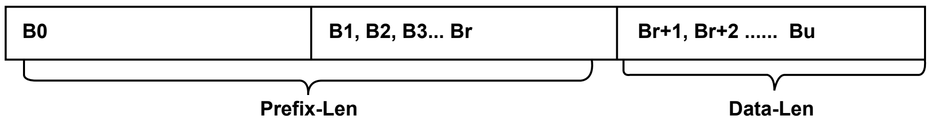


Figure 12-2: Data Lengths to be Programmed

Programming Flow for SHA-Only Operation

Core Mode

Configuration

1. Enable `CRYPT_CFG.SHA256EN`.
2. Enable the `CRYPT_STAT.SHADONE` interrupt. Do not enable `CRYPT_STAT.INRDY`.
3. Set `CRYPT_CFG.BLKEN` =1 and enable the Crypto.

Data Input

1. Program the first 512-bit block of data to the input buffer.
2. Wait for the `CRYPT_STAT.SHADONE` interrupt (Core can sleep until an interrupt occurs).
3. Wait until `CRYPT_STAT.SHABUSY` goes low. Clear the `CRYPT_STAT.SHADONE` bit.

4. Program the next 512-bit data block. If the next block is the last data block:
    a. Send the integral number of words from the last block.
    b. Before the last word in the data block, set the `CRYPT_SHA_LAST_WORD.O_LAST_WORD` bit and program the number of valid bits in that word. For example, if 12 bits are valid, the `CRYPT_SHA_LAST_WORD.O_BITS_VALID` bit must be loaded with 12.
    c. If all the bits are valid, the last write must be a dummy write.

SHA Hash Retrieval

1. The `CRYPT_SHA_LAST_WORD.O_LAST_WORD` bit auto clears once the current data block is processed.
2. Wait until `CRYPT_STAT.SHADONE` interrupt occurs. Read the digest from the `CRYPT_SHAHx` registers (x = 0 to 7).
3. For a fresh SHA calculation, initialized SHA using the `CRYPT_CFG.SHAINIT` bit or reset the `CRYPT_CFG.BLKEN` bit.

DMA Mode

Configuration

1. Enable `CRYPT_CFG.SHA256EN` and `CRYPT_CFG.INDMAEN`.
2. Configure the DMA descriptor for the required number of words and set up the DMA transfer.

Data Input

1. Wait until the DMA transfer of integral number of words is complete. If the message length is a multiple of 512-bits, wait for `CRYPT_STAT.SHABUSY` to go low.
2. Clear the `CRYPT_STAT.SHADONE` bit.
3. Before the last word in the data block, set the `CRYPT_SHA_LAST_WORD.O_LAST_WORD` bit and program the number of valid bits in that word.
4. If all the bits are valid, the last write must be a dummy write.

SHA Hash Retrieval

1. Enable `CRYPT_STAT.SHADONE` interrupt.
2. Wait until `CRYPT_STAT.SHADONE` interrupt occurs.
3. Read the digest from the `CRYPT_SHAHx` registers (x = 0 to 7).

Software Operation in CCM Mode

The diagrams use the following color code.

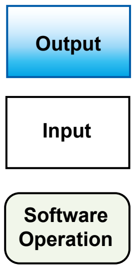


Figure 12-3: Color Codes

The NIST Standard for CCM mode has the following description of the CCM operation.

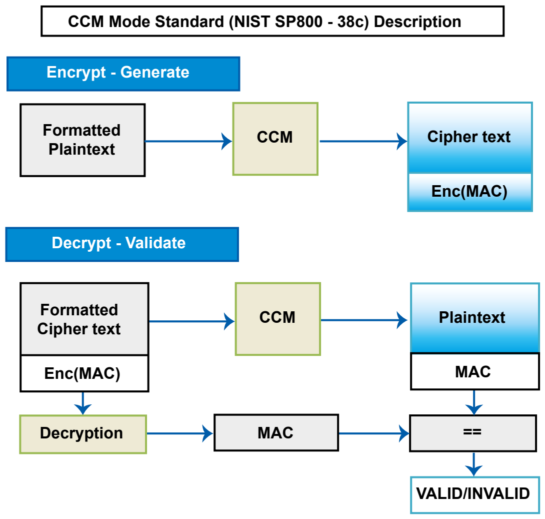


Figure 12-4: CCM Mode Standard Description

The hardware implementation for the CCM mode is shown in the figure.

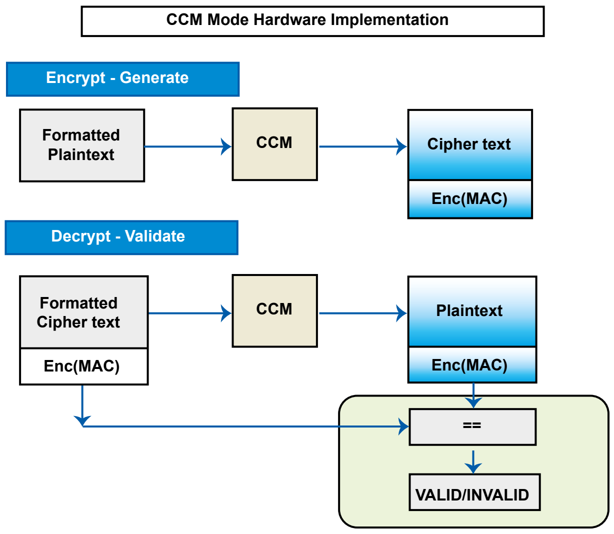
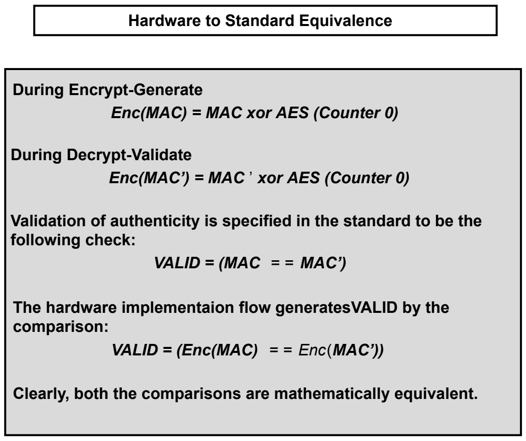


Figure 12-5: CCM Mode Hardware Implementation



Hardware to Standard Equivalence

During Encrypt-Generate

Enc(MAC) = MAC xor AES (Counter 0)

During Decrypt-Validate

Enc(MAC') = MAC ' xor AES (Counter 0)

Validation of authenticity is specified in the standard to be the following check:

VALID = (MAC = = MAC')

The hardware implementaion flow generatesVALID by the comparison:

VALID = (Enc(MAC) = = Enc(MAC'))

Clearly, both the comparisons are mathematically equivalent.

Figure 12-6: Hardware to Standard Equivalence

The following are the reasons for the difference in the implementation flow adopted for generation of VALID signal in Decrypt-Validate phase:

- CMAC mode provides the MAC results and does not check the result against the received reasonable to do the same in CCM mode.
- This approach is faster: When the system receives data that needs decryption, it receives an encrypted MAC. In out implementation it is not necessary to decrypt the received MAC.
- The length of the MAC is selectable. This means that if the hardware were to do the comparison internally, information regarding the length of the MAC output must be programmed. This adds an additional control parameter and also increases the hardware complexity by having a variable length comparator.
- The comparison operation is simple and involves very few operations. It does not require a hardware accelerator.
- MAC is not expected as an output from CCM operation. Only a VALID or INVALID signal is necessary to be generated by software. The crypto block computes the MAC and passes it to software via registers. The matching of this MAC and corresponding pass/fail status has to be generated within software. Therefore. it does not matter if we give out Enc(MAC) or MAC.

ADuCM302x CRYPT Register Descriptions

Crypto block contains the following registers.

Table 12-2: ADuCM302x CRYPT Register List

Name	Description
CRYPT_AESKEY0	AES Key Bits [31:0]
CRYPT_AESKEY1	AES Key Bits [63:32]
CRYPT_AESKEY2	AES Key Bits [95:64]

Table 12-2: ADuCM302x CRYPT Register List (Continued)

Name	Description
CRYPT_AESKEY3	AES Key Bits [127:96]
CRYPT_AESKEY4	AES Key Bits [159:128]
CRYPT_AESKEY5	AES Key Bits [191:160]
CRYPT_AESKEY6	AES Key Bits [223:192]
CRYPT_AESKEY7	AES Key Bits [255:224]
CRYPT_CCM_NUM_VALID_BYTES	NUM_VALID_BYTES
CRYPT_CFG	Configuration Register
CRYPT_CNTRINIT	Counter Initialization Vector
CRYPT_DATALEN	Payload Data Length
CRYPT_INBUF	Input Buffer
CRYPT_INTEN	Interrupt Enable Register
CRYPT_NONCE0	Nonce Bits [31:0]
CRYPT_NONCE1	Nonce Bits [63:32]
CRYPT_NONCE2	Nonce Bits [95:64]
CRYPT_NONCE3	Nonce Bits [127:96]
CRYPT_OUTBUF	Output Buffer
CRYPT_PREFIXLEN	Authentication Data Length
CRYPT_SHAH0	SHA Bits [31:0]
CRYPT_SHAH1	SHA Bits [63:32]
CRYPT_SHAH2	SHA Bits [95:64]
CRYPT_SHAH3	SHA Bits [127:96]
CRYPT_SHAH4	SHA Bits [159:128]
CRYPT_SHAH5	SHA Bits [191:160]
CRYPT_SHAH6	SHA Bits [223:192]
CRYPT_SHAH7	SHA Bits [255:224]
CRYPT_SHA_LAST_WORD	SHA Last Word and Valid Bits Information
CRYPT_STAT	Status Register

AES Key Bits [31:0]

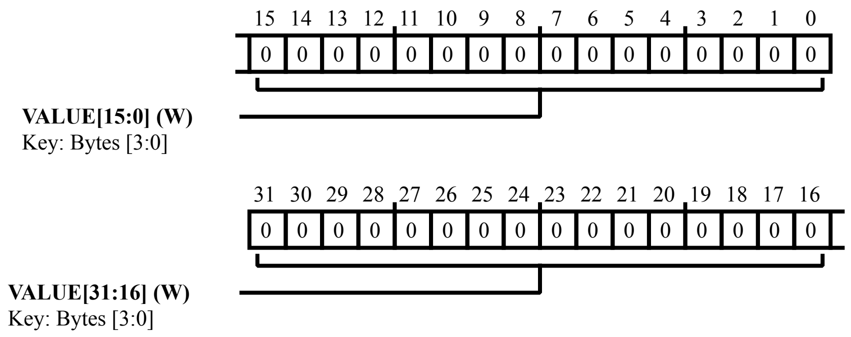


Figure 12-7: CRYPT_AESKEY0 Register Diagram

Table 12-3: CRYPT_AESKEY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Key: Bytes [3:0].

AES Key Bits [63:32]

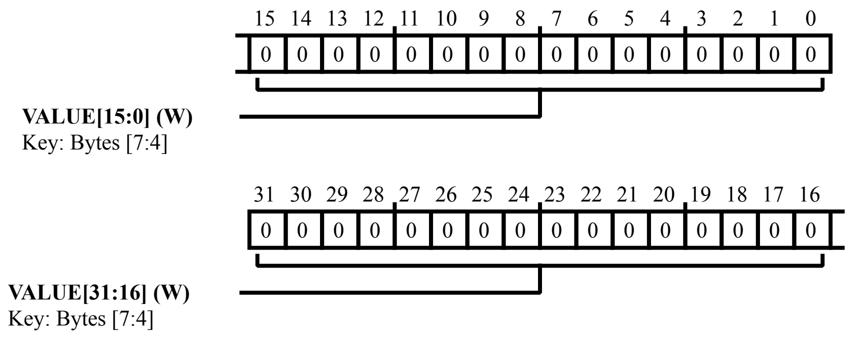


Figure 12-8: CRYPT_AESKEY1 Register Diagram

Table 12-4: CRYPT_AESKEY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Key: Bytes [7:4].

AES Key Bits [95:64]

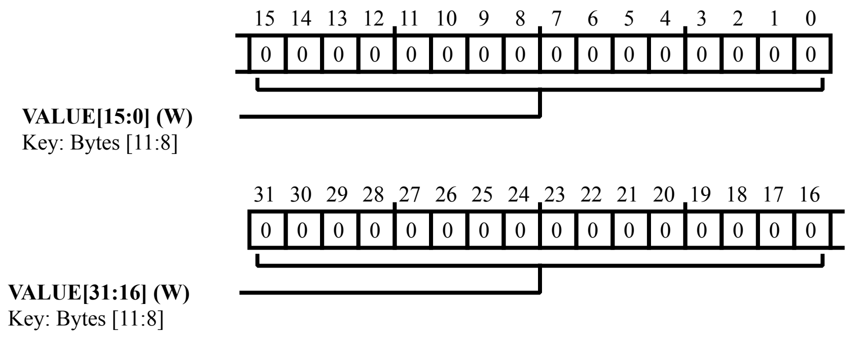


Figure 12-9: CRYPT_AESKEY2 Register Diagram

Table 12-5: CRYPT_AESKEY2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Key: Bytes [11:8].

AES Key Bits [127:96]

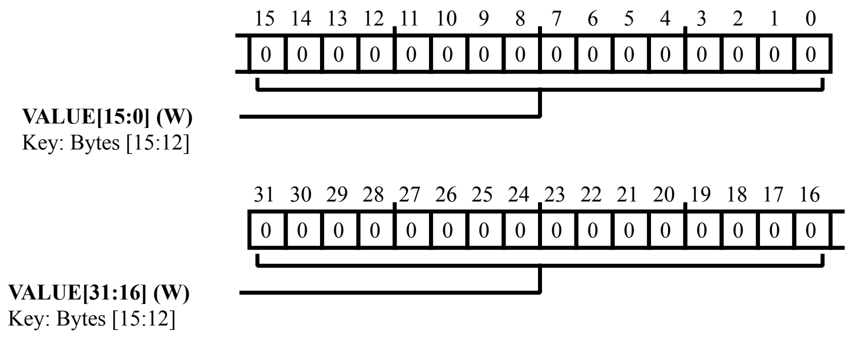


Figure 12-10: CRYPT_AESKEY3 Register Diagram

Table 12-6: CRYPT_AESKEY3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Key: Bytes [15:12].

AES Key Bits [159:128]

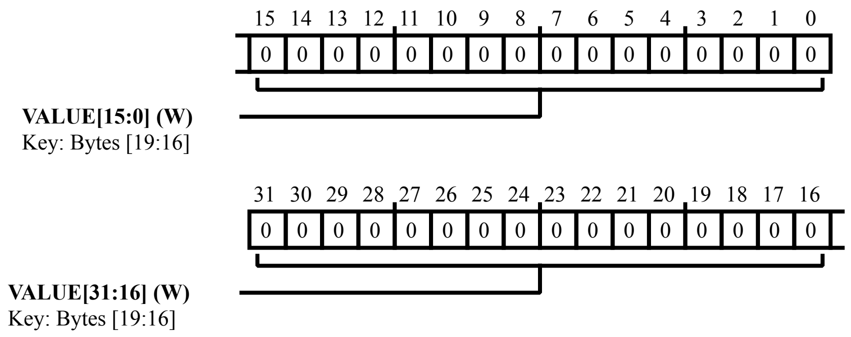


Figure 12-11: CRYPT_AESKEY4 Register Diagram

Table 12-7: CRYPT_AESKEY4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Key: Bytes [19:16].

AES Key Bits [191:160]

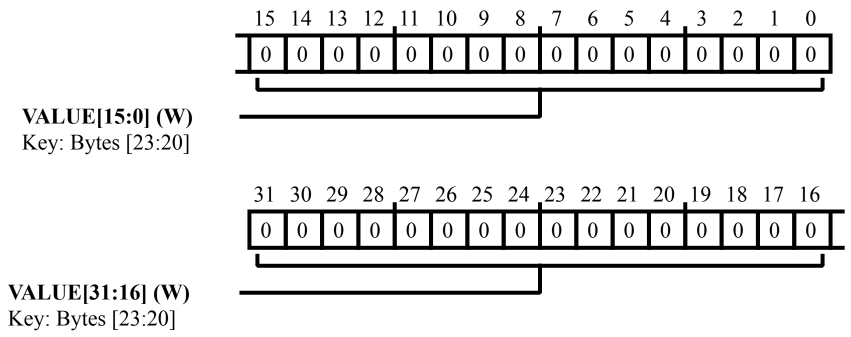


Figure 12-12: CRYPT_AESKEY5 Register Diagram

Table 12-8: CRYPT_AESKEY5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Key: Bytes [23:20].

AES Key Bits [223:192]

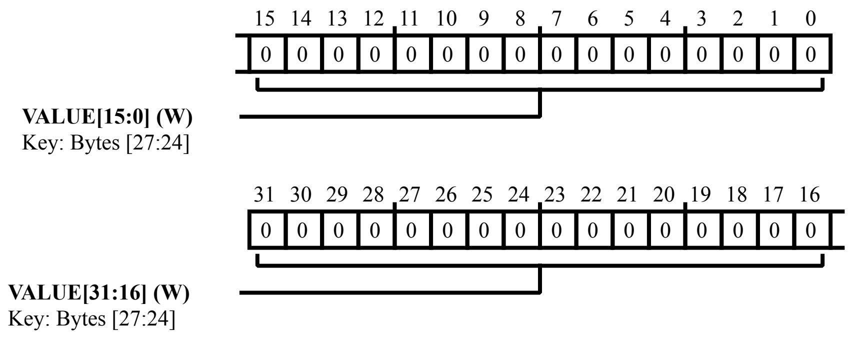


Figure 12-13: CRYPT_AESKEY6 Register Diagram

Table 12-9: CRYPT_AESKEY6 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Key: Bytes [27:24].

AES Key Bits [255:224]

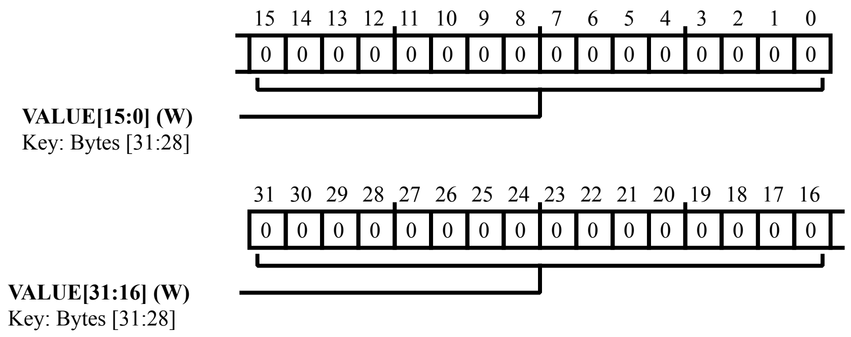


Figure 12-14: CRYPT_AESKEY7 Register Diagram

Table 12-10: CRYPT_AESKEY7 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Key: Bytes [31:28].

NUM_VALID_BYTES

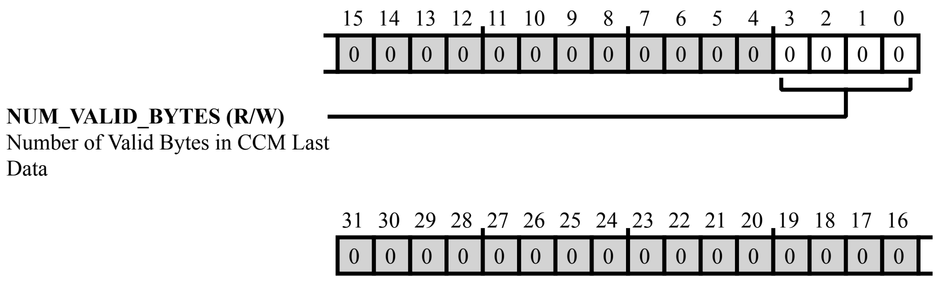


Figure 12-15: CRYPT_CCM_NUM_VALID_BYTES Register Diagram

Table 12-11: CRYPT_CCM_NUM_VALID_BYTES Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	NUM_VALID_BYTES	Number of Valid Bytes in CCM Last Data.

Configuration Register

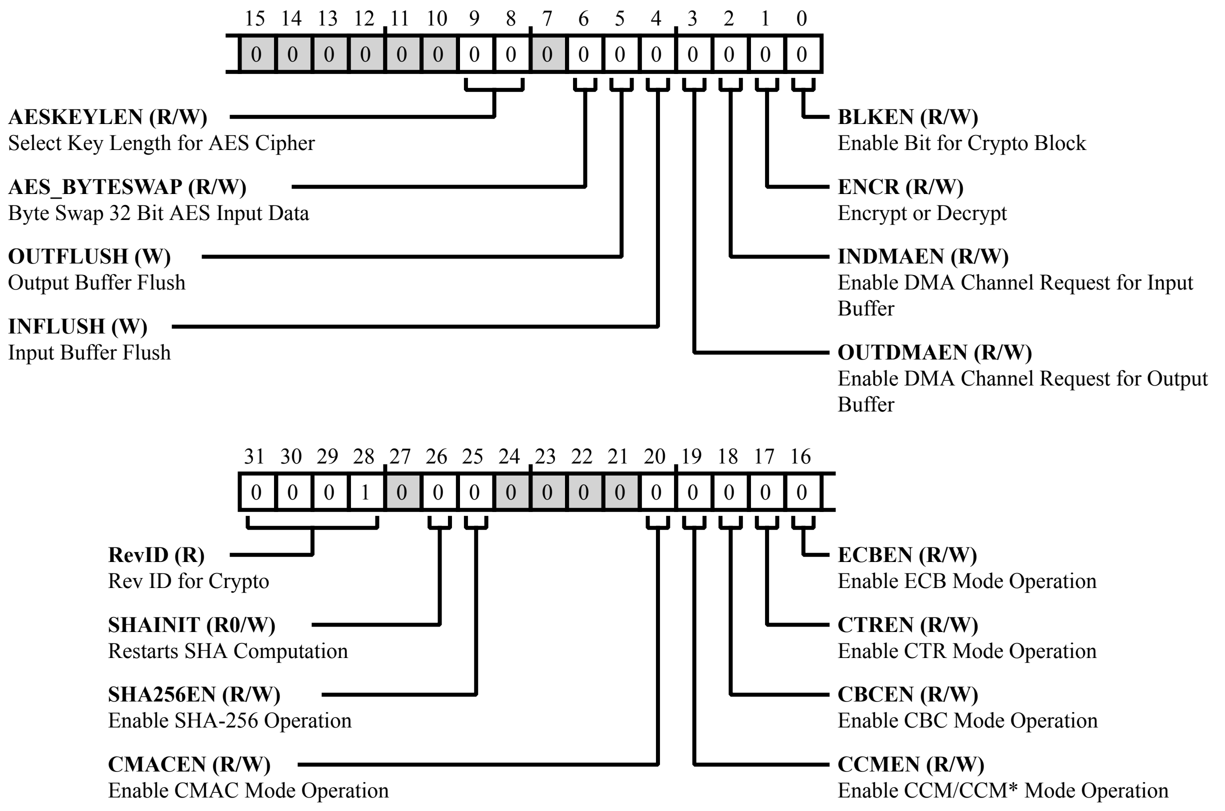


Figure 12-16: CRYPT_CFG Register Diagram

Table 12-12: CRYPT_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:28 (R/NW)	REVID	Rev ID for Crypto. The revision id is set to 0x2 for this version.
26 (R0/W)	SHAINIT	Restarts SHA Computation. Initialize SHA module for fresh SHA calculation on a new data block. This bit resets the SHA result registers to their default state. This auto-clears after reseting the SHA result registers.
25 (R/W)	SHA256EN	Enable SHA-256 Operation.
20 (R/W)	CMACEN	Enable CMAC Mode Operation.
19 (R/W)	CCMEN	Enable CCM/CCM* Mode Operation.

Table 12-12: CRYPT_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	CBCEN	Enable CBC Mode Operation.	
17 (R/W)	CTREN	Enable CTR Mode Operation.	
16 (R/W)	ECBEN	Enable ECB Mode Operation.	
9:8 (R/W)	AESKEYLEN	Select Key Length for AES Cipher. Specify the length of AES key. Supported key lengths are 128 and 256 bits. Any change in the key length will reset the AES key.	
		0	Uses 128-bit long key
		2	Uses 256-bit long key
6 (R/W)	AES_BYTESWAP	Byte Swap 32 Bit AES Input Data. Enable Key Wrap	
5 (RX/W)	OUTFLUSH	Output Buffer Flush.	
4 (RX/W)	INFLUSH	Input Buffer Flush.	
3 (R/W)	OUTDMAEN	Enable DMA Channel Request for Output Buffer.	
		0	Disable DMA Requesting for Output Buffer
		1	Enable DMA Requesting for Output Buffer
2 (R/W)	INDMAEN	Enable DMA Channel Request for Input Buffer.	
		0	Disable DMA Requesting for Input Buffer
		1	Enable DMA Requesting for Input Buffer
1 (R/W)	ENCR	Encrypt or Decrypt. Select Encryption/Decryption for a given mode. This bit need not be set/reset for Wrap/Unwrap operation in Protected Key Storage.	
		0	Decrypt
		1	Encrypt

Table 12-12: CRYPT_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	BLKEN	Enable Bit for Crypto Block. Enable Crypto hardware accelerator. This bit is intended to be used as a start of operation for all Crypto modes. Unless this bit is enabled, no Crypto mode shall function.	
		0	Enable Crypto Block
		1	Disable Crypto Block

Counter Initialization Vector

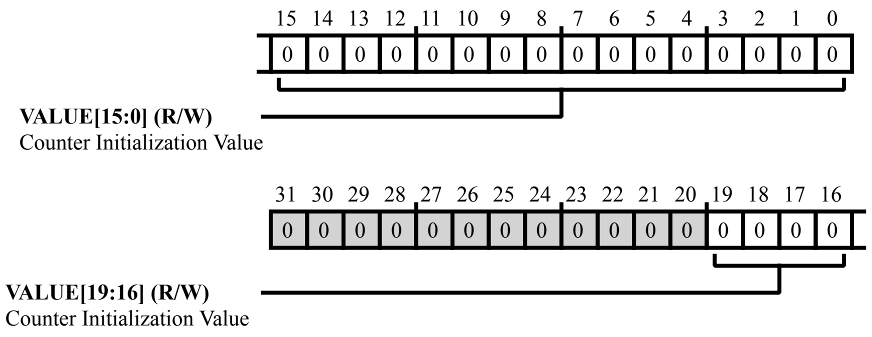


Figure 12-17: CRYPT_CNTRINIT Register Diagram

Table 12-13: CRYPT_CNTRINIT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:0 (R/W)	VALUE	Counter Initialization Value. This is the initialization value used in the internal counter generating functions that are used in some modes of operation. 1. CCM/CCM* Mode: Only `CRYPT_CNTRINIT.VALUE`[15:0] are used in this mode for initializing the counter initialization vector. The higher bits are ignored. 2. CTR Mode: `CRYPT_CNTRINIT.VALUE`[19:0] are used in to initialize the counter initialization vector. When the block is reset, the counters are initialized to the value programmed in this register.

Payload Data Length

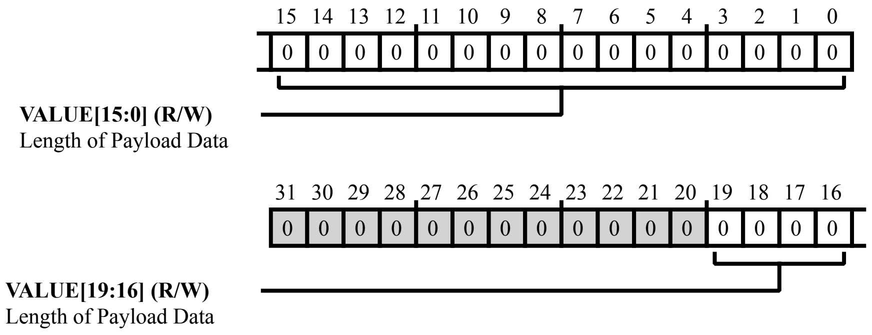


Figure 12-18: CRYPT_DATALEN Register Diagram

Table 12-14: CRYPT_DATALEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:0 (R/W)	VALUE	Length of Payload Data. Program the length of the Payload that needs to be encrypted with the following values: 1. MAC Mode: `CRYPT_DATALEN.VALUE`[19:0] should be used to program the number of 128-bit blocks in the Payload. 2. CCM/CCM* Mode: `CRYPT_DATALEN.VALUE`[15:0] should be used to program the number of 128-bit blocks in the Payload.

Input Buffer

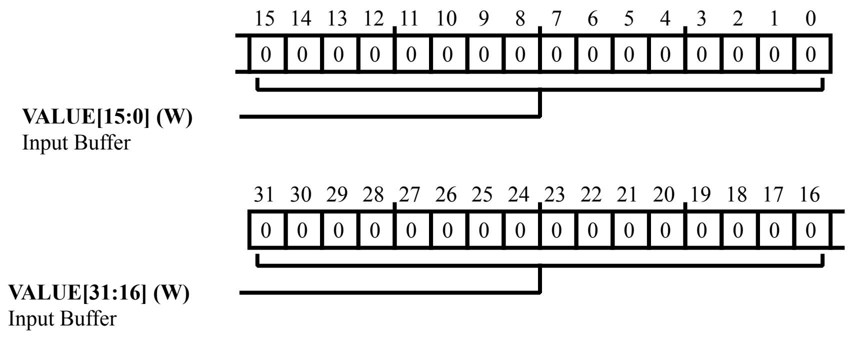


Figure 12-19: CRYPT_INBUF Register Diagram

Table 12-15: CRYPT_INBUF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Input Buffer.

Interrupt Enable Register

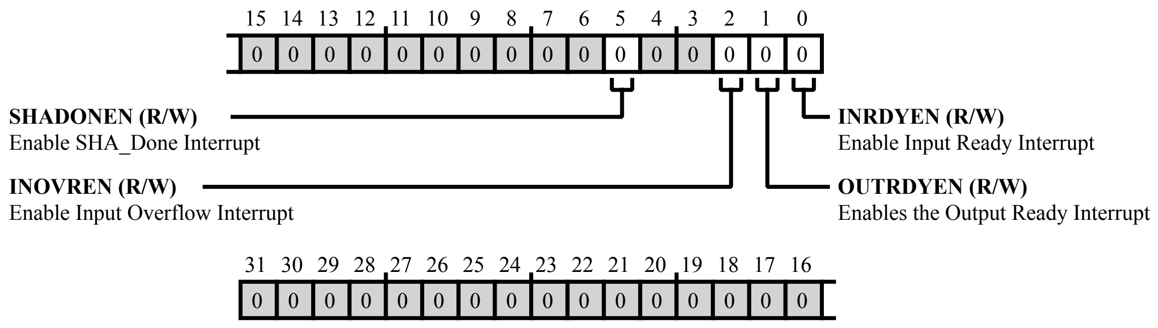


Figure 12-20: CRYPT_INTEN Register Diagram

Table 12-16: CRYPT_INTEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W)	SHADONEN	Enable SHA_Done Interrupt. This bit when set interrupts the core when SHA has finished processing the current data block.
2 (R/W)	INOVREN	Enable Input Overflow Interrupt. This bit when set interrupts the core in case input buffer overflows.
1 (R/W)	OUTRDYEN	Enables the Output Ready Interrupt. This bit when set interrupts the core when output buffer is ready to provide more data. Once the output buffer is empty, this bit resets.
0 (R/W)	INRDYEN	Enable Input Ready Interrupt. This bit when set interrupts the core when input buffer is ready for more data. Once the input buffer is full, this bit resets.

Nonce Bits [31:0]

Nonce is used in some modes of operations. Depending on the mode, different nonce lengths will be used.

1. CTR mode: This takes a 108-bit nonce. This nonce is formed as follows:

{CRYPT_NONCE3[11:0], CRYPT_NONCE2, CRYPT_NONCE1, CRYPT_NONCE0}

2. CBC mode: This takes a 128-bit nonce. This nonce is formed as follows:

{CRYPT_NONCE3, CRYPT_NONCE2, CRYPT_NONCE1, CRYPT_NONCE0}

3. CTR mode: This takes a 108-bit nonce. This nonce is formed as follows:

{CRYPT_NONCE3[15:0], CRYPT_NONCE2, CRYPT_NONCE1, CRYPT_NONCE0}

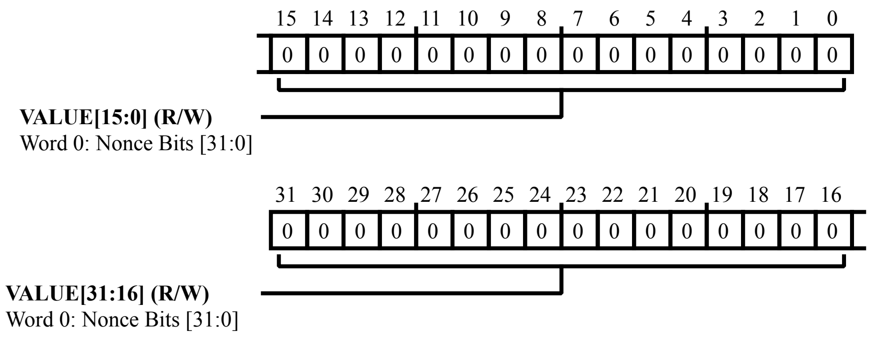


Figure 12-21: CRYPT_NONCE0 Register Diagram

Table 12-17: CRYPT_NONCE0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Word 0: Nonce Bits [31:0].

Nonce Bits [63:32]

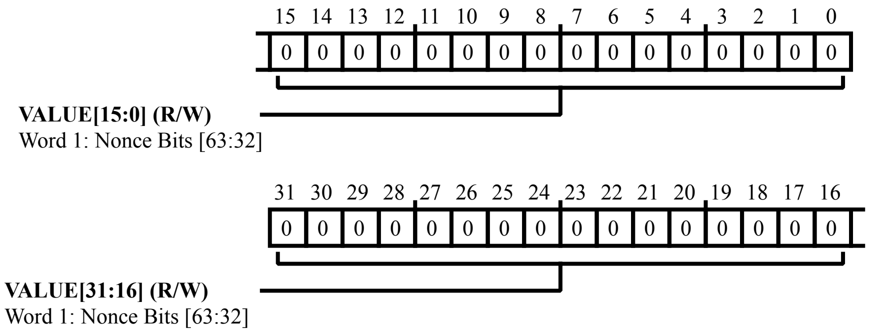


Figure 12-22: CRYPT_NONCE1 Register Diagram

Table 12-18: CRYPT_NONCE1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Word 1: Nonce Bits [63:32].

Nonce Bits [95:64]

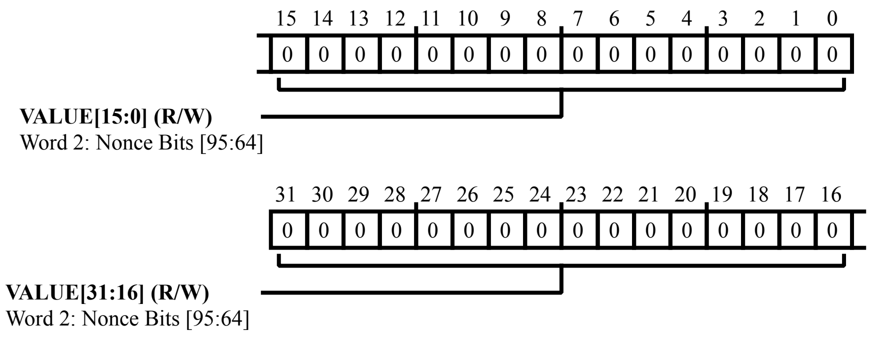


Figure 12-23: CRYPT_NONCE2 Register Diagram

Table 12-19: CRYPT_NONCE2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Word 2: Nonce Bits [95:64].

Nonce Bits [127:96]

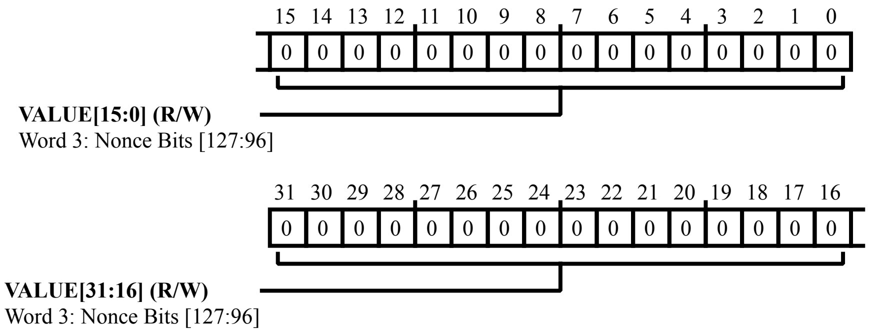


Figure 12-24: CRYPT_NONCE3 Register Diagram

Table 12-20: CRYPT_NONCE3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Word 3: Nonce Bits [127:96].

Output Buffer

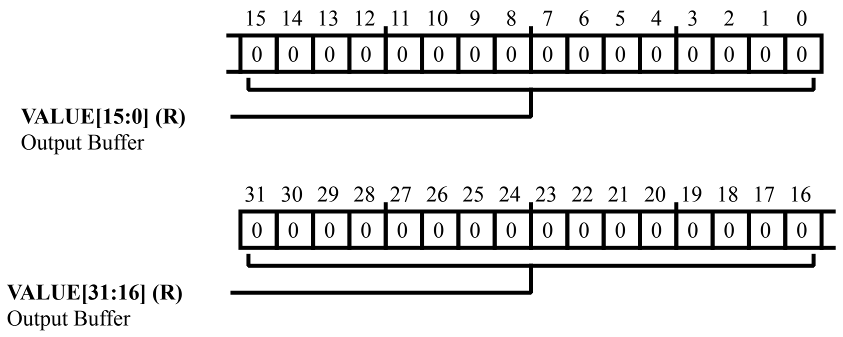


Figure 12-25: CRYPT_OUTBUF Register Diagram

Table 12-21: CRYPT_OUTBUF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Output Buffer.

Authentication Data Length

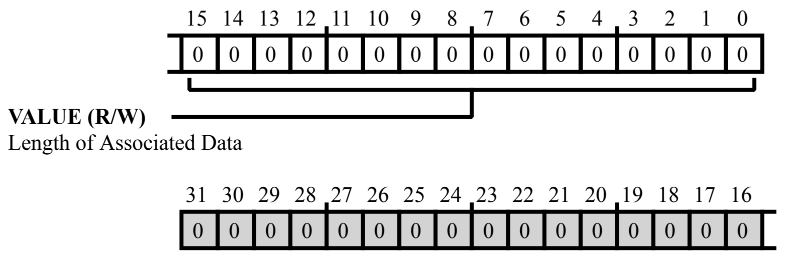


Figure 12-26: CRYPT_PREFIXLEN Register Diagram

Table 12-22: CRYPT_PREFIXLEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Length of Associated Data. Program the length of the Associated that needs to be encrypted with the following values: CCM/CCM* Mode: 20 bits are provided to program the number of 128-bit blocks of the associated data. The higher 4 bits are be ignored.

SHA Bits [31:0]

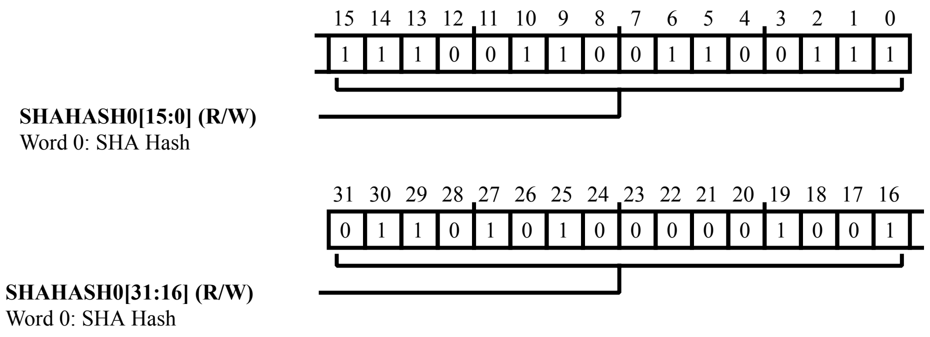


Figure 12-27: CRYPT_SHAH0 Register Diagram

Table 12-23: CRYPT_SHAH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	SHAHASH0	Word 0: SHA Hash.

SHA Bits [63:32]

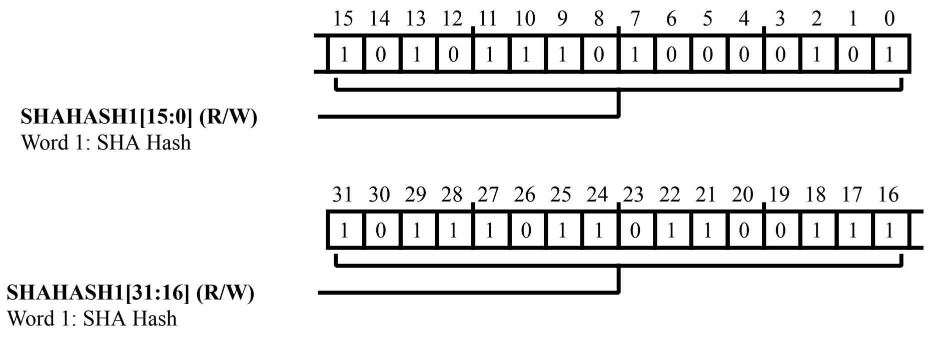


Figure 12-28: CRYPT_SHAH1 Register Diagram

Table 12-24: CRYPT_SHAH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	SHAHASH1	Word 1: SHA Hash.

SHA Bits [95:64]

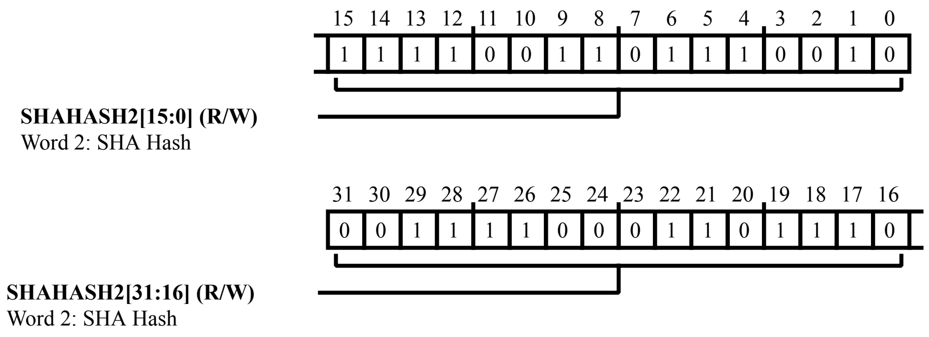


Figure 12-29: CRYPT_SHAH2 Register Diagram

Table 12-25: CRYPT_SHAH2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	SHAHASH2	Word 2: SHA Hash.

SHA Bits [127:96]

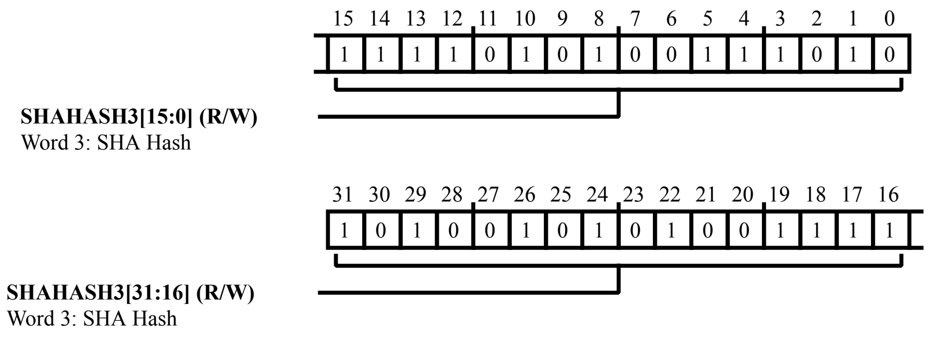


Figure 12-30: CRYPT_SHAH3 Register Diagram

Table 12-26: CRYPT_SHAH3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	SHAHASH3	Word 3: SHA Hash.

SHA Bits [159:128]

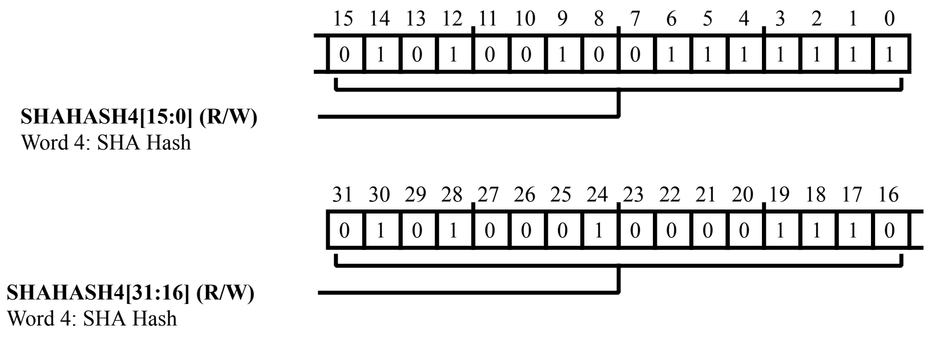


Figure 12-31: CRYPT_SHAH4 Register Diagram

Table 12-27: CRYPT_SHAH4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	SHAHASH4	Word 4: SHA Hash.

SHA Bits [191:160]

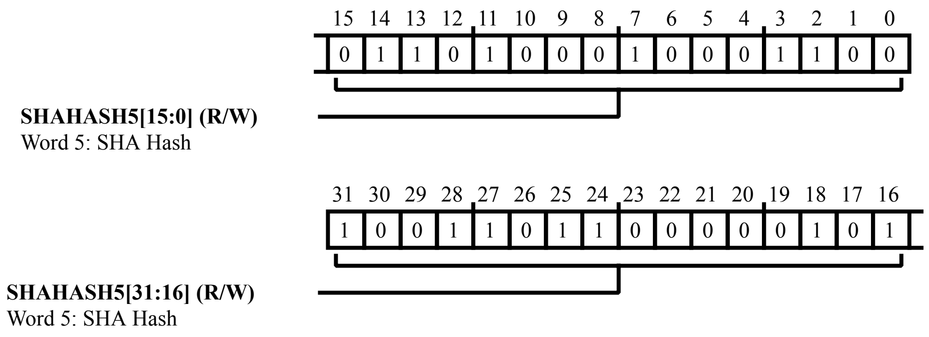


Figure 12-32: CRYPT_SHAH5 Register Diagram

Table 12-28: CRYPT_SHAH5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	SHAHASH5	Word 5: SHA Hash.

SHA Bits [223:192]

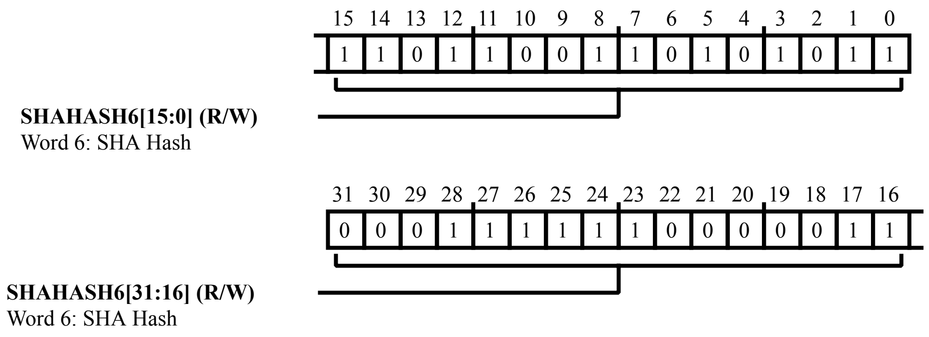


Figure 12-33: CRYPT_SHAH6 Register Diagram

Table 12-29: CRYPT_SHAH6 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	SHAHASH6	Word 6: SHA Hash.

SHA Bits [255:224]

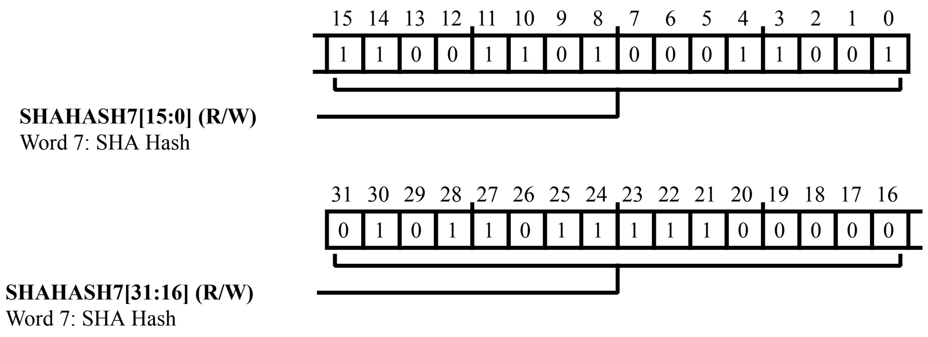


Figure 12-34: CRYPT_SHAH7 Register Diagram

Table 12-30: CRYPT_SHAH7 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	SHAHASH7	Word 7: SHA Hash.

SHA Last Word and Valid Bits Information

This register is to be written before writing the last word to SHA input register. This is to inform the SHA engine that last word is about to be written by writing to the `CRYPT_SHA_LAST_WORD.O_LAST_WORD`. Also, the number of valid bits has to be programmed in the `CRYPT_SHA_LAST_WORD.O_BITS_VALID`.

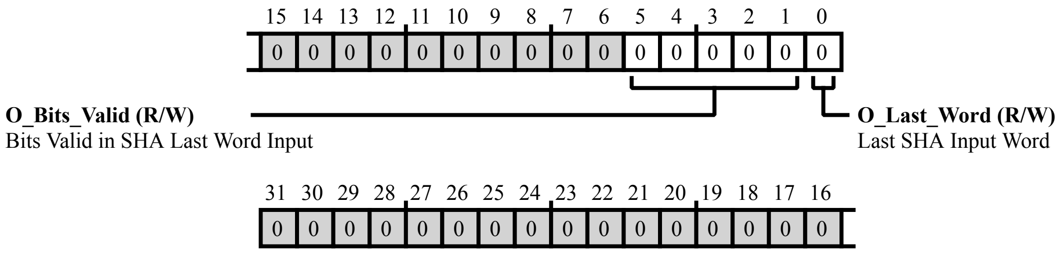


Figure 12-35: CRYPT_SHA_LAST_WORD Register Diagram

Table 12-31: CRYPT_SHA_LAST_WORD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5:1 (R/W)	O_BITS_VALID	Bits Valid in SHA Last Word Input. Indicates the number of valid bits in the last input word to SHA. This should auto clear when `CRYPT_STAT.SHADONE` is set
0 (R/W)	O_LAST_WORD	Last SHA Input Word. This should be set to indicate that last word is about to be written to the SHA engine. This should auto clear when `CRYPT_STAT.SHADONE` is set.

Status Register

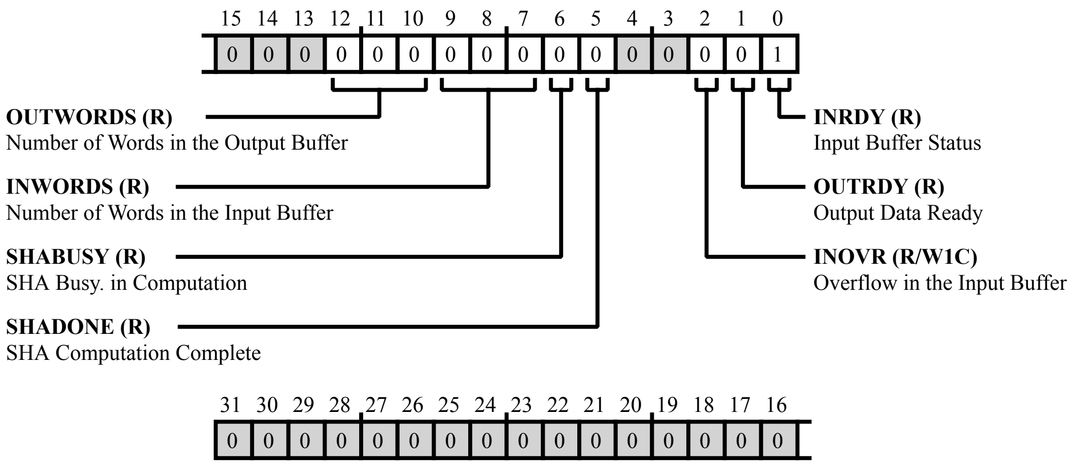


Figure 12-36: CRYPT_STAT Register Diagram

Table 12-32: CRYPT_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:10 (R/NW)	OUTWORDS	Number of Words in the Output Buffer.
9:7 (R/NW)	INWORDS	Number of Words in the Input Buffer.
6 (R/NW)	SHABUSY	SHA Busy. in Computation. Indicates that computation is ongoing for recent data. While this is set, values in the CRYPT_SHAHx registers are invalid.
5 (R/NW)	SHADONE	SHA Computation Complete. Indicates that hash computation is complete for current data. When hash computation is complete, the current value of the hash may be read out or new data may be written into the input buffer to proceed with further computation.
2 (R/W1C)	INOVR	Overflow in the Input Buffer.
1 (R/NW)	OUTRDY	Output Data Ready. Output Data ready to be read
0 (R/NW)	INRDY	Input Buffer Status. Input buffer requires more data before computation can begin. Remains set till the buffer is filled.

13 True Random Number Generator (TRNG)

True Random Number Generator (TRNG) is used during operations where non-deterministic values are required. This may include generating challenges for secure communication or keys used for an encrypted communication channel. The generator can be run multiple times to generate a sufficient number of bits for the strength of the intended operation. The true random number generator can be used to seed a deterministic random bit generator such as one described by NIST CRC SP-800-90A.

TRNG Features

The following are the features of TRNG:

- Programmable length to obtain sufficient entropy.
- Includes an oscillator counter to characterize the sampling jitter.

TRNG Functional Description

This section provides information on the function of the TRNG used by the ADuCM302x MCU.

TRNG Block Diagram

The figure shows the block diagram of the TRNG used by the ADuCM302x MCU.

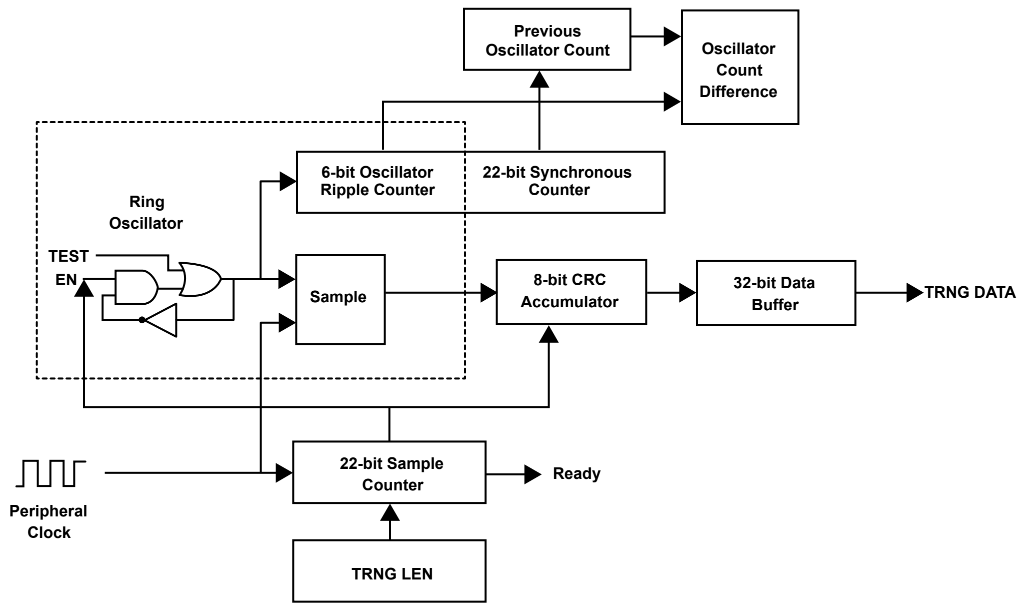


Figure 13-1: TRNG Block Diagram

The TRNG is based on a sampled asynchronous clock (in this implementation a ring oscillator). A CRC is used as a compaction function to reduce a large amount of low entropy bits into a smaller amount of high entropy bits. The CRC will accumulate a programmable number of samples determined by the sample length register.

The TRNG is enabled through the control register. The length register specifies the number of samples for which the TRNG should run to accumulate sufficient entropy. One sample is obtained on each peripheral clock and compacted by the CRC. Once the TRNG has accumulated the programmed number of samples, the accumulated 8-bit CRC result can be read. Software can poll a status bit (`RNG_STAT.RNRDY`) or be interrupted when the TRNG is ready.

Any number of iterations can be run. For example, 112 bits of entropy can be obtained by reading the 8-bit TRNG result at least 14 times.

TRNG Oscillator Counter

The TRNG peripheral includes an oscillator counter. This on-chip counter can be used to characterize the sampling jitter which is difficult to measure off-chip. The sample jitter is the source of entropy for the random number generator.

By counting the number of ring oscillator clocks over a given sampling period, the frequency ratio of the ring oscillator to peripheral sampling clock can be determined (and thus the ring oscillator frequency can be determined if the peripheral clock frequency is known).

$$f_{OSCCLK} = f_{PCLK} \times \frac{OSCCNT}{SAMPCNT}$$

where,

OSCCNT is `RNG_OSCCNT.VALUE`. Length of sampling time (SAMPCNT) used by the sample counter can be determined from the `RNG_LEN` register.

The sample jitter can be determined by calculating the standard deviation of multiple oscillator count values.

A pseudo-code for an efficient loop across N values to determine the average oscillator count and jitter is shown below:

```
sum=0; sum_sqr=0;
for(i=0;i<N;i++)
{
gen_rng();
sum += osc_cnt;
sum_sqr += osc_cnt*osc_cnt;
}
avg=sum/N;
std=sqrt((sum_sqr-avg*sum)/(N-1));
```

The code has been simplified for clarity.

If implemented in C:

- The sum and sum_sqr variables must be integer data types. Floating point variables could lose precision if the sums accumulate for a sufficient length to overflow the fractional portion of the variable truncating the result. It is desirable to keep the entire fractional portion, so the exponent of a floating point data type can be excluded.
- The sum variable needs $\log_2$(osc_cnt) + $\log_2$(N) bits of storage.
- The sum_sqr variable needs 2 × $\log_2$(osc_cnt) + $\log_2$(N) bits of storage.
- The squaring operation (osc_cnt × osc_cnt and avg × avg) requires a cast to a type with size at least 2 × $\log_2$(osc_cnt).
- The avg and std variables can take fractional values (real or fixed point data types).
- The division needs a cast to a fractional data type.

For accurate jitter measurements, the standard deviation of `RNG_OSCCNT` should be at least one. If it is less than one, then SAMPCNT must be increased.

The jitter of the oscillator counter encapsulates both jitter of the ring oscillator and jitter of the peripheral clock. This can be used to determine sample jitter. If the jitter of the peripheral clock is known, the jitter of the oscillator clock can be determined using the following equation.

$$\frac{\sigma_{OSCCNT}}{\sqrt{SAMPCNT}} \frac{SAMPCNT}{OSCCNT} \frac{1}{f_{PCLK}} = \sigma_{SAMPLE} = \sqrt{\sigma^2_{PCLK} + \sigma^2_{OSCCLK} \frac{OSCCNT}{SAMPCNT}}$$

TRNG Entropy and Surprisal

The jitter of the sampling clocks is the source of entropy for the random number generator. Noise in the transistors of the oscillator contributes to jitter. Accumulated long-term jitter increases with the square-root of time or the number of clocks.

$$\sigma_{TOTAL} = \sigma_{CLK}\sqrt{N}$$

The number of samples that need to be accumulated to achieve ideal entropy of 1.0/bit depends on the amount of jitter in the system. Entropy is calculated as follows:

$$average\ entropy = -\sum p_i log_2(p_i)$$

$$minimum\ entropy = min(-log_2(p_i))$$

The probability of each number occurring should be the same or uniform for all numbers for an ideal generator. A deficient generator would output some numbers more frequently than others. This random number generator was designed to retain no state such that entropy can be measured and quantified in this manner. For each random number generated, the CRC is reset and the ring oscillator starts up in the same phase. If there is insufficient entropy in the system, then the generator will output the same number more frequently. This is by design and makes entropy assessment possible. Insufficient entropy can be observed by setting SAMPCNT to a low value in which case some random numbers will appear more frequently than others. SAMPCNT should be increased until the probability of seeing each number is uniform.

The minimum entropy of the TRNG can be computed by tallying a histogram of generated random numbers and calculating the probability of the most frequent number occurring. The minimum entropy equation should be used when determining how many true random numbers to seed to a deterministic random number generator.

The minimum value for SAMPCNT (set by `RNG_LEN` register) needed to obtain ideal minimum surprisal for the 8-bit accumulator can be determined using the following equation. A conservative design will set SAMPCNT at least as large as this, if not greater (to account for variations in jitter across various operating conditions).

$$SAMPCNT_{min} = \frac{1}{\sigma^2_{SAMPLE} f_{OSC}^2}$$

If an attacker can physically tamper the system and has control of the peripheral clock, the jitter due to the peripheral clock must be removed from the sample jitter in the previous calculation. This assumes an attacker can replace the crystal with a low jitter clock source. A physical attacker may also use better voltage supplies that can minimize the amount of noise in the system. If an attacker can remove or minimize the peripheral clock jitter, then the entropy source of the random number generator is reduced. The system should rely solely on the jitter of the ring oscillator, and conservatively, assume the jitter of the peripheral clock to be zero.

Oscillator Count Difference

There is logic built into the random number generator to calculate the difference between subsequent oscillator count values. This can be used to calculate the variance and quantify the amount of jitter and thus entropy in the system. This provides a measure of health of the random number generator entropy source.

The statistical sample variance is typically calculated as follows:

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

The on-chip circuit calculates the difference in oscillator count between the current and previous sample. Since the samples are independent and identically generated, this removes the mean.

$$E[X] - E[X] = 0$$

$$E[(X - X)^2] = 2\sigma^2$$

The variance can be computed by averaging half of the square of the difference between oscillator count values.

$$OSCVAR = AVERAGE\left(\frac{(OSCCNT_i - OSCCNT_{i-1})^2}{2}\right)$$

The average can be replaced with a low pass IIR filter to track changes in the variance over time.

TRNG Interrupts and Exceptions

The TRNG block cannot generate any interrupts.

ADuCM302x RNG Register Descriptions

Random Number Generator (RNG) contains the following registers.

Table 13-1: ADuCM302x RNG Register List

Name	Description
RNG_CTL	RNG Control Register
RNG_DATA	RNG Data Register

Table 13-1: ADuCM302x RNG Register List (Continued)

Name	Description
RNG_LEN	RNG Sample Length Register
RNG_OSCCNT	Oscillator Count
RNG_OSCDIFF[n]	Oscillator Difference
RNG_STAT	RNG Status Register

RNG Control Register

The RNG_CTL register is used to enable the random number generator.

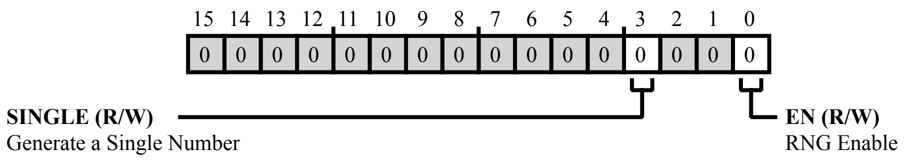


Figure 13-2: RNG_CTL Register Diagram

Table 13-2: RNG_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	SINGLE	Generate a Single Number. By default the RNG will generate and buffer four 8-bit values to provide a 32-bit random number. Setting this bit will cause the RNG to only generate a single 8-bit random number.	
		0	Buffer Word
		1	Single Byte
0 (R/W)	EN	RNG Enable. When RNG_CTL.EN is set and RNG_STAT.RNRDY is clear, the ring oscillator will be powered up and the number of samples defined by RNG_LEN will be accumulated in the RNG_DATA register.	
		0	Disable the RNG
		1	Enable the RNG

RNG Data Register

RNG_DATA register provides the CPU with read-only access of the entropy accumulator (8-bit CRC) and data buffer. When the data buffer is not enabled, an 8-bit result is provided. When the data buffer is enabled, 32-bits (four 8-bit values) are provided. The contents of this register are valid when the RNG_STAT.RNRDY bit is set. This register is reset when the RNG_STAT.RNRDY bit is cleared. The RNG_STAT.RNRDY bit is automatically cleared when this register is read and the CPU is not in debug halt. Reading this register by the CPU when RNG_CTL.EN is set will cause a new random number to be generated.

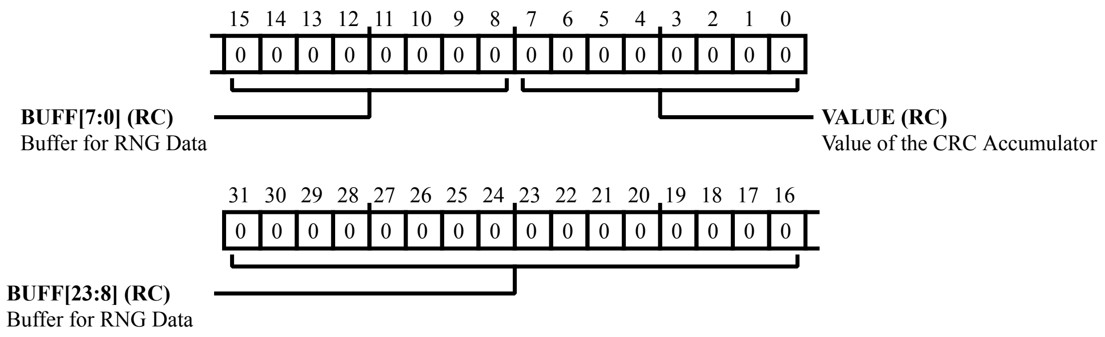


Figure 13-3: RNG_DATA Register Diagram

Table 13-3: RNG_DATA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:8 (RC/NW)	BUFF	Buffer for RNG Data. When configured to generate 32-bit values, RNG_DATA.BUFF stores the previous three numbers generated.
7:0 (RC/NW)	VALUE	Value of the CRC Accumulator. This register provides data from the entropy compaction function (8-bit CRC accumulator).

RNG Sample Length Register

The RNG_LEN register defines the number of samples to accumulate in the CRC register when generating a random number. The number of samples accumulated is RNG_LEN.RELOAD scaled by 2^RNG_LEN.PRESCALE.

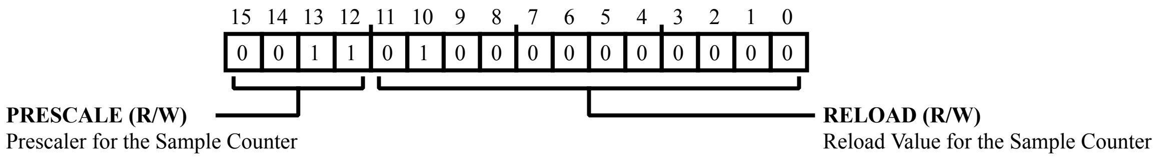


Figure 13-4: RNG_LEN Register Diagram

Table 13-4: RNG_LEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:12 (R/W)	PRESCALE	Prescaler for the Sample Counter. The sample counter reload value RNG_LEN.RELOAD is scaled by 2^RNG_LEN.PRESCALE. The prescaler is a 10-bit counter. Valid values for the prescaler are 0 to 10. Values greater than 10 will saturate at the maximum prescaler value.
11:0 (R/W)	RELOAD	Reload Value for the Sample Counter. Defines the number of samples to accumulate in the CRC when generating a random number.

Oscillator Count

The oscillator counter counts the number of ring oscillator cycles which occur during the generation of a random number. The oscillator counter is 28-bits. The oscillator counter will saturate at the maximum value to prevent overflow.

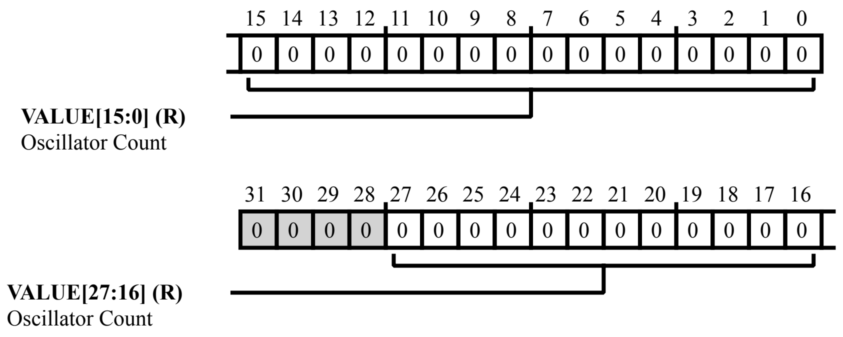


Figure 13-5: RNG_OSCCNT Register Diagram

Table 13-5: RNG_OSCCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:0 (R/NW)	VALUE	Oscillator Count. This register is only valid when `RNG_STAT.RNRDY` is set.

Oscillator Difference

The oscillator difference register stores the difference in RNG_OSCCNT from the current value compared to the previous value (RNG_OSCCNT[n] - RNG_OSCCNT[n-1]). This difference is represented as a signed 8-bit value. It saturates at the maximum and minimum values. This can be used to reconstruct RNG_OSCCNT for the values currently in the RNG_DATA buffer. This information can be used to compute the RNG_OSCCNT variance to check the health of the random number generator and ensure there is adequate entropy.

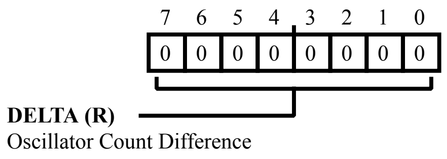


Figure 13-6: RNG_OSCDIFF[n] Register Diagram

Table 13-6: RNG_OSCDIFF[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	DELTA	Oscillator Count Difference.

RNG Status Register

The RNG_STAT register indicates when the RNG has finished generating a random number.

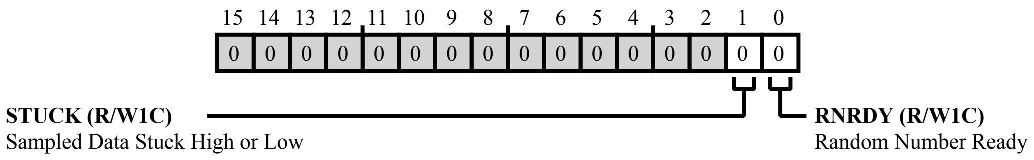


Figure 13-7: RNG_STAT Register Diagram

Table 13-7: RNG_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	STUCK	Sampled Data Stuck High or Low. When a random number is generated, a circuit monitors the output of the sampled ring oscillator. This circuit checks to ensure the ring oscillator output has been sampled both high and low and is not stuck at a constant value. This bit is sticky once set and is write one to clear.	
0 (R/W1C)	RNRDY	Random Number Ready. This bit indicates when the value in RNG_DATA is ready to be read. An interrupt is generated when this bit is set. The ring oscillator is stopped when this bit is set to conserve power. This bit is automatically cleared when the RNG_DATA register is read and the CPU is not stopped in Debug Halt. This bit can also be written with one to clear.	
		0	Data register not ready
		1	Data register is ready to be read

14 Cyclic Redundancy Check (CRC)

The CRC accelerator is used to compute the CRC for a block of memory locations. The exact memory location can be in the SRAM, flash, or any combination of memory mapped registers. The CRC accelerator generates a checksum that can be used to compare with an expected signature. The final CRC comparison is the responsibility of the MCU.

CRC Features

The CRC used by the ADuCM302x MCU supports the following features:

- Generate a CRC signature for a block of data.
- Programmable polynomial length of up to 32 bits.
- Operates on 32 bits of data at a time.
- MSB-first and LSB-first CRC implementations.
- Various data mirroring capabilities.
- Initial seed to be programmed by user.
- DMA controller (using software DMA) can be used for data transfer to offload the MCU.

CRC Functional Description

This section provides information on the function of the CRC accelerator used by the ADuCM302x MCU. Control for address decrement/increment options for computing the CRC on a block of memory is in the DMA controller.

For more information about these options, refer to Direct Memory Access (DMA).

CRC Block Diagram

The CRC block diagram is shown below.

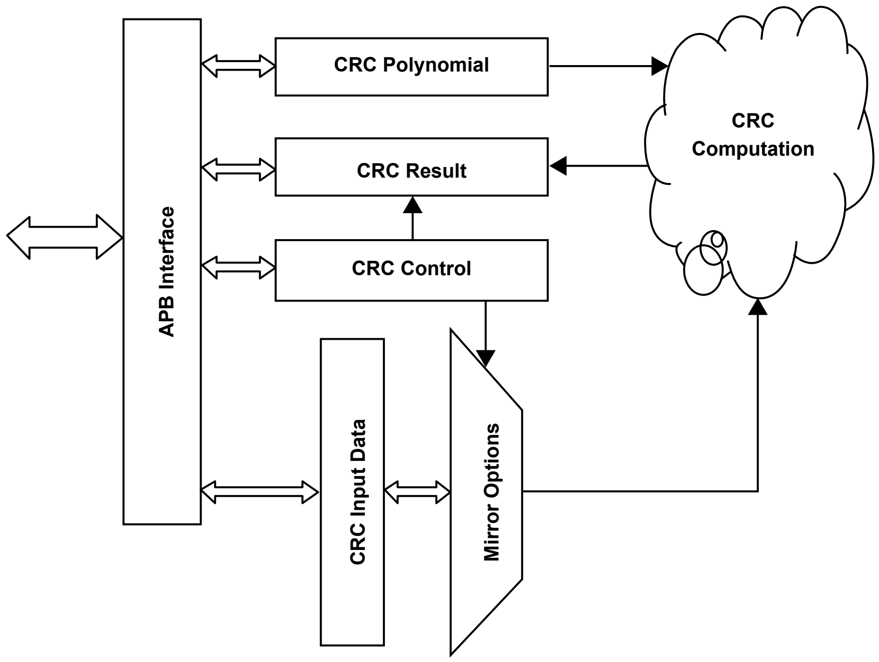


Figure 14-1: CRC Block Diagram

CRC Architectural Concepts

The CRC accelerator works on 32-bit data words which are fed to the block through the DMA channel dedicated to the CRC accelerator or directly by the MCU, and the CRC accelerator guarantees immediate availability of the CRC output.

CRC Operating Modes

The accelerator calculates CRC on the data stream it receives, 32 bits at a time, which is written into the block either using the DMA engine or MCU directly.

The CRC works on 32-bit data-words. For data-words less than 32 bit in size, it is the responsibility of the MCU to pack the data into 32-bit data units. Data mirroring on the input data can be performed at bit, byte, and word level (only for 32-bit data) before the CRC engine uses it by setting the `CRC_CTL.BITMIRR`, `CRC_CTL.BYTMIRR`, and `CRC_CTL.W16SWP` bits respectively.

When operating, the CRC algorithm runs on the incoming data stream written to the `CRC_IPDATA` register. For every new word of data received, the CRC is computed and the `CRC_RESULT` register is updated with the calculated CRC. The CRC Accelerator guarantees the immediate availability of CRC result up to the current data in the CRC Result register and the same can be read.

The CRC engine uses the current `CRC_RESULT` for generating the next `CRC_RESULT` when a new data-word is received. The `CRC_RESULT` register can be programmed with an initial seed. The bit-width of the seed value for an n-bit polynomial must be n. The seed should be justified in the `CRC_RESULT` register.

Polynomial

The CRC Accelerator supports the calculation of the CRC using any length polynomial. The polynomial has to be written to the `CRC_POLY` register. For MSB first implementation the highest power is omitted while programming the CRC polynomial register and the polynomial is left justified. For LSB, first implementation the polynomial is right justified and the LSB is omitted. The `CRC_RESULT` register contains n-bit MSBs as checksum for an n-bit CRC polynomial.

The following examples illustrate the CRC polynomial.

16-bit Polynomial Programming for MSB First Calculation

Polynomial: CRC-16-CCITT, $x^{16}+x^{12}+x^5+1$ = (1) 0001 0000 0010 0001 = 0x1021

The largest exponent (x^{16} term) is implied, so it is 0001 0000 0010 0001

When left justified in the polynomial register, this becomes

CRC Polynomial Register (`CRC_POLY`)

0001 0000	0010 0001	8b0	8b0

CRC Result Register (`CRC_RESULT`)

CRC	Result	8b0	8b0

Initial seed programmed in CRC Result Register (`CRC_RESULT`)

CRC	SEED	8b0	8b0

16-bit Polynomial Programming for LSB First Calculation

Polynomial: CRC-16-CCITT, $x^{16}+x^{12}+x^5+1$ = 1000 0100 0000 1000 (1) = 0x8408

The smallest exponent (x^0 term) is implied, so it is 1000 0100 0000 1000

When right justified in the polynomial register, this becomes

CRC Polynomial Register (`CRC_POLY`)

8b0	8b0	1000 0100	0000 1000

CRC Result Register (`CRC_RESULT`)

8b0	8b0	CRC	Result

Initial seed programmed in CRC Result Register (`CRC_RESULT`)

8b0	8b0	CRC	SEED

8-bit Polynomial Programming for MSB First Calculation

Polynomial: CRC-8-ATM, $x^8 + x^2 + x + 1$ = (1) 0000 0111= 0x07

The largest exponent (x^8 term) is implied, so it is 0000 0111

When left justified in the polynomial register, this becomes

CRC Polynomial Register (`CRC_POLY`)

0000 0111	8b0	8b0	8b0

CRC Result Register (`CRC_RESULT`)

CRC RESULT	8b0	8b0	8b0

Initial seed programmed in CRC Result Register (`CRC_RESULT`)

CRC SEED	8b0	8b0	8b0

8-bit Polynomial Programming for LSB First Calculation

Polynomial: CRC-8-ATM, $x^8 + x^2 + x + 1$ = 1000 0011 (1) = 0x83

The smallest exponent (x^0 term) is implied, so it is 1000 0011

When right justified in the polynomial register, this becomes

CRC Polynomial Register (`CRC_POLY`)

8b0	8b0	8b0	1000 0011

CRC Result Register (`CRC_RESULT`)

8b0	8b0	8b0	CRC RESULT

Initial seed programmed in CRC Result Register (`CRC_RESULT`)

8b0	8b0	8b0	CRC SEED

The CRC engine uses the following 32-bit CRC polynomial as default (IEEE 802.3):

$g(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

This is programmed for MSB First Calculation by default as shown below

0x04	0xC1	0x1D	0xB7

Reset and Hibernate Modes

1. The CRC configuration bits are retained except block enable bit (`CRC_CTL.EN`). The block needs to be enabled again after coming out of hibernate mode.
2. The CRC polynomial and CRC result registers are retained after coming out of hibernate mode.

Table 14-1: Reset and Hibernate Modes

Register	Reset	Hibernate
`CRC_CTL`	0x0	Apart from BLKEN, all other bits retained
`CRC_POLY`	0x04C11DB7	Retained
`CRC_IPDATA`	0x0	Not retained (0x0)
`CRC_RESULT`	0x0	Retained

CRC Data Transfer

The data stream can be written to the block using DMA controller or by the MCU directly.

CRC Interrupts and Exceptions

DMA channel generates an interrupt upon completion of data transfer to the CRC block.

CRC Programming Model

This block is provided to calculate CRC signature over block of data in the background while the core can perform other tasks.

The CRC block supports two modes of CRC calculation: Core access and DMA access.

Core Access

1. Program the `CRC_POLY` register with the required polynomial justified as shown in the examples in the Polynomial section.
2. Program the `CRC_RESULT` register with initial seed. The seed must be justified and written to the `CRC_RESULT` register, as described in Polynomial.
3. Kick in the CRC Accelerator block by writing into the `CRC_CTL` register:
   a. Set the `CRC_CTL.EN` bit high.
   b. Modify the `CRC_CTL.W16SWP`, `CRC_CTL.BITMIRR`, and `CRC_CTL.BITMIRR` bits, which configure the application with different mirror options. For more information about this, refer to Mirroring Options.
   c. Set/Reset the `CRC_CTL.LSBFIRST` bit to indicate LSB/MSB first CRC calculation.

NOTE: The sub-steps in *Step 3* require only a single write to the `CRC_CTL` register.

The core can now start sending data to the CRC block by writing into the `CRC_IPDATA` register. The CRC accelerator continues to calculate the CRC as long as data is written to the `CRC_IPDATA` register. It is the responsibility of the application to count the number of words written to the CRC block. Once all the words are written, the application can read the `CRC_RESULT` register.

4. Read the `CRC_RESULT` register. It contains the n-bit result in n MSB bits for MSB first and in n LSB bits for LSB first CRC calculation.
5. Calculate CRC on the next data block. To calculate the CRC on the next block of data, repeat *Steps 1-4*.
6. Disable the CRC accelerator block by clearing the `CRC_CTL.EN` bit. This ensure that the block is in the low-power state.

DMA Access

The CRC accelerator block supports software DMA.

1. Program the `CRC_POLY` register with the required polynomial left justified as shown in the example in the Polynomial section.
2. Program the `CRC_RESULT` register with initial seed value. The seed should be justified and written to the `CRC_RESULT` register, as described in Polynomial.
3. Enable accelerator function by writing into `CRC_CTL` register.
   a. Set the `CRC_CTL.EN` bit high.
   b. Modify the `CRC_CTL.W16SWP`, `CRC_CTL.BITMIRR`, and `CRC_CTL.BITMIRR` bits, which configure the application with different mirror options. For more information about this, refer to Mirroring Options.
   c. Set/Reset the `CRC_CTL.LSBFIRST` bit to indicate LSB/MSB first CRC calculation.

   NOTE: The sub-steps in *Step 3* require only a single write to the `CRC_CTL` register.

   The DMA starts sending the CRC data by writing into the `CRC_IPDATA`. The CRC Accelerator block continues to calculate the CRC as long as the data is written to the `CRC_IPDATA`.
4. Setup the DMA channels as required. DST_END_PNTR value is `CRC_IPDATA` register address. Data size is word. Use the destination no increment option for the channel used. For more information about programming the DMA, refer to Programming Guidelines.
5. A `dma_done` interrupt signal of the DMA channel indicates the completion of data transfer to the CRC block.
6. Repeat *Steps 1-4* until all the data has been sent to the accelerator block.
7. Read the `CRC_RESULT` register. It contain the n-bit result in n MSB bits for MSB first and in n LSB bits for LSB first CRC calculations.

8. Calculate CRC on the next data block. To calculate the CRC on the next block of data, repeat *Steps 1-5*.
9. Disable the CRC accelerator block by clearing the `CRC_CTL.EN` bit. This ensures that the block is in low-power state.

Mirroring Options

The `CRC_CTL.W16SWP`, `CRC_CTL.BITMIRR`, and `CRC_CTL.BYTMIRR` bits determine the sequence of the bits in which the CRC is calculated.

Mirroring Options for 32-bit Input Data with 32-bit Polynomial table shows the details of all the mirroring options used within this block for a 32-bit polynomial.

Assume that DIN[31:0] is the data being written to the `CRC_IPDATA` register, and CIN[31:0] is the data after the mirroring of the data. The serial engine calculates CIN[31:0] starting with the MSB bit and ending with LSB bit in sequence, that is, CIN[31], CIN[30], ... CIN[1], CIN[0] in order.

Table 14-2: Mirroring Options for 32-bit Input Data with 32-bit Polynomial

W16SWP	BYTMIRR	BITMIRR	Input Data DIN[31:0]	CRC Input Data (CIN[31:0])
0	0	0	DIN[31:0]	CIN[31:0] = DIN[31:0]
0	0	1	DIN[31:0]	CIN[31:0] = DIN[24:31], DIN[16:23], DIN[8:15], DIN[0:7]
0	1	0	DIN[31:0]	CIN[31:0] = DIN[23:16], DIN[31:24], DIN[7:0], DIN[15:8]
0	1	1	DIN[31:0]	CIN[31:0] = DIN[16:23], DIN[24:31], DIN[0:7], DIN[8:15]
1	0	0	DIN[31:0]	CIN[31:0] = DIN[15:0], DIN[31:16]
1	0	1	DIN[31:0]	CIN[31:0] = DIN[8:15], DIN[0:7], DIN[24:31], DIN[16:23]
1	1	0	DIN[31:0]	CIN[31:0] = DIN[7:0], DIN[15:8], DIN[23:16], DIN[31:24]
1	1	1	DIN[31:0]	CIN[31:0] = DIN[0:7], DIN[8:15], DIN[16:23], DIN[24:31]

ADuCM302x CRC Register Descriptions

CRC Accelerator (CRC) contains the following registers.

Table 14-3: ADuCM302x CRC Register List

Name	Description
`CRC_CTL`	CRC Control
`CRC_IPBITS[n]`	Input Data Bits
`CRC_IPBYTE`	Input Data Byte
`CRC_IPDATA`	Input Data Word
`CRC_POLY`	Programmable CRC Polynomial

Table 14-3: ADuCM302x CRC Register List (Continued)

Name	Description
CRC_RESULT	CRC Result

CRC Control

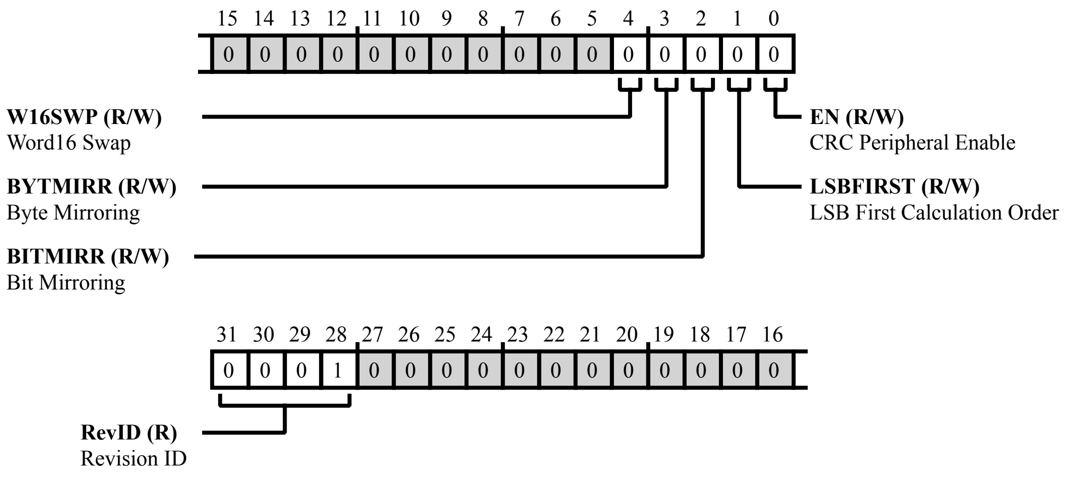


Figure 14-2: CRC_CTL Register Diagram

Table 14-4: CRC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:28 (R/NW)	REVID	Revision ID.	
4 (R/W)	W16SWP	Word16 Swap. This bit will swap 16-bit half-words within a 32-bit word.	
		0	Word16 Swap disabled
		1	Word16 Swap enabled
3 (R/W)	BYTMIRR	Byte Mirroring. This bit will swap 8-bit bytes within each 16-bit half-word.	
		0	Byte Mirroring is disabled
		1	Byte Mirroring is enabled
2 (R/W)	BITMIRR	Bit Mirroring. This bit will swap bits within each byte.	
		0	Bit Mirroring is disabled
		1	Bit Mirroring is enabled
1 (R/W)	LSBFIRST	LSB First Calculation Order.	
		0	MSB First CRC calculation is done
		1	LSB First CRC calculation is done

Table 14-4: CRC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	CRC Peripheral Enable.	
		0	CRC peripheral is disabled
		1	CRC peripheral is enabled

Input Data Bits

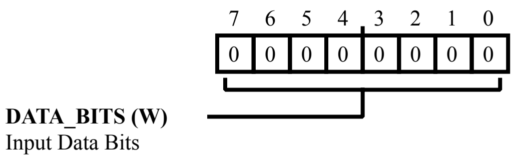


Figure 14-3: CRC_IPBITS[n] Register Diagram

Table 14-5: CRC_IPBITS[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (RX/W)	DATA_BITS	Input Data Bits. These fields are used to calculate CRC on partial data byte from 1-7 bits of input data. Computing CRC on n bits of input data can be achieved by writing byte to n bit of `CRC_IPBITS[n].DATA_BITS`.

Input Data Byte

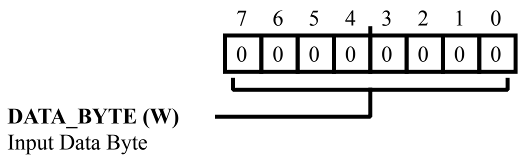


Figure 14-4: CRC_IPBYTE Register Diagram

Table 14-6: CRC_IPBYTE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (RX/W)	DATA_BYTE	Input Data Byte. Writing data to this field calculates CRC on a byte of data.

Input Data Word

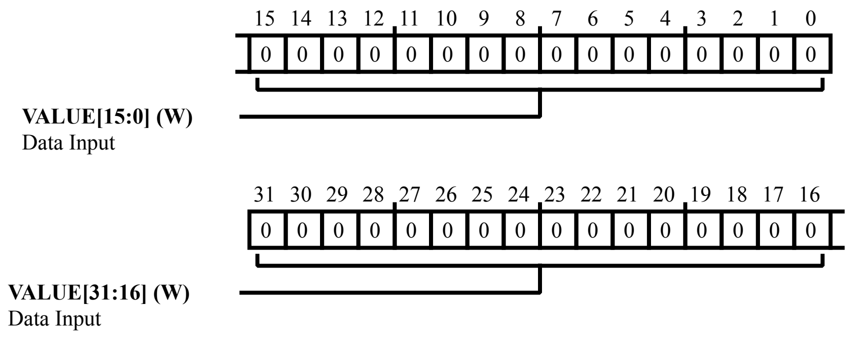


Figure 14-5: CRC_IPDATA Register Diagram

Table 14-7: CRC_IPDATA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Data Input.

Programmable CRC Polynomial

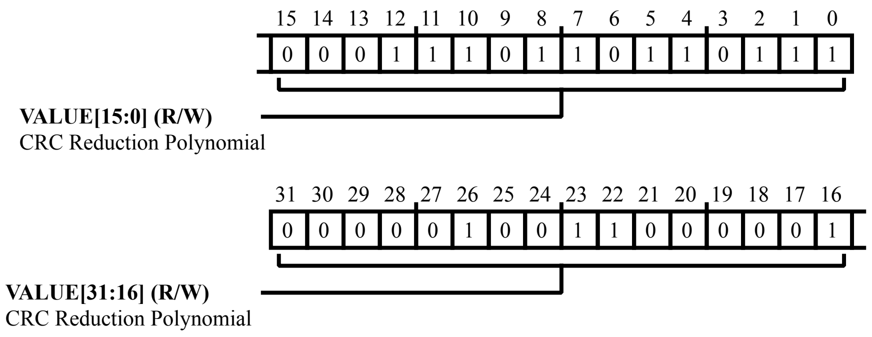


Figure 14-6: CRC_POLY Register Diagram

Table 14-8: CRC_POLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	CRC Reduction Polynomial.

CRC Result

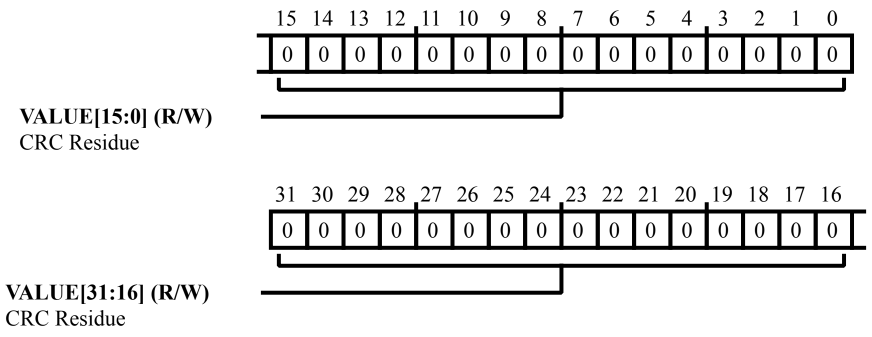


Figure 14-7: CRC_RESULT Register Diagram

Table 14-9: CRC_RESULT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	CRC Residue.

15 Serial Peripheral Interface (SPI)

The Serial Peripheral Interface (SPI) is an industry-standard synchronous serial link that supports communication with multiple SPI-compatible devices. The baseline SPI peripheral is a synchronous, four-wire interface consisting of two data pins, one device select pin, and a gated clock pin. The two data pins allow full duplex operation to other SPI compatible devices. Enhanced modes of operation such as flow control, fast mode, and read command mode (half duplex operation) are also supported. In addition, a DMA mode allows for transferring several words with minimal CPU interaction.

With a range of configurable options, the SPI ports provide a glueless hardware interface with other SPI compatible devices in master mode, slave mode, and multislave environments. The SPI peripheral includes programmable baud rates, clock phase, and clock polarity. The peripheral can operate in a multislave environment by interfacing with several other devices, acting as either a master device or a slave device. In a multislave environment, the SPI peripheral uses open drain outputs to avoid data bus contention. The flow control features enable slow slave devices to interface with fast master devices by providing an SPI ready pin which flexibly controls the transfers.

SPI Features

The SPI module supports the following features:

- Serial clock phase mode and serial clock polarity mode
- Loopback mode
- Continuous transfer mode
- Wired-OR output mode
- Read-command mode for half-duplex operation
- Flow control
- Multiple CS line
- CS software override
- Support for 3-pin SPI Master or Slave mode
- LSB-first transfer option

- Interrupt mode: interrupt after 1, 2, 3, 4, 5, 6, 7 or 8 bytes

SPI Functional Description

During an SPI transfer, data is simultaneously transmitted (shifted out serially) and received (shifted in serially). A serial clock line synchronizes shifting and sampling of the information on the two serial data lines. During a data transfer, one SPI system acts as the link master which controls the data flow, while the other system acts as the slave, which has data shifted into and out of it by the master. Different devices can take turn being masters, and one master may simultaneously shift data into multiple slaves (broadcast mode).

However, only one slave may drive its output to write data back to the master at any given time. This must be enforced in the broadcast mode, where several slaves can be selected to receive data from the master, but only one slave can be enabled to send data back to the master.

The SPI port can be configured for master or slave operation and consists of four pins: MISO, MOSI, SCLK, and CS. Note that the GPIOs used for SPI communication must be configured in SPI mode before enabling the SPI peripheral. The peripheral should be enabled by setting the `SPI_CTL.SPIEN` bit. If not used, the peripheral must be turned off by clearing this bit.

SPI Block Diagram

The figure shows the SPI block diagram.

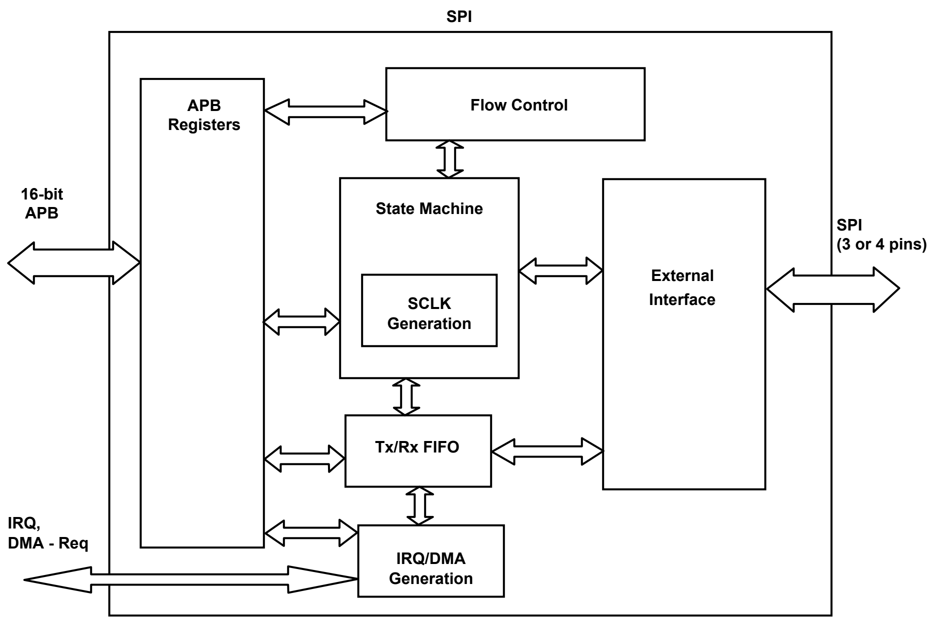


Figure 15-1: SPI Block Diagram

MISO (Master In, Slave Out) Pin

The MISO pin is configured as an input line in master mode and an output line in slave mode. The MISO line on the master (data in) should be connected to the MISO line in the slave device (data out). The data is transferred as byte wide (8-bit) serial data.

MOSI (Master Out, Slave In) Pin

The MOSI pin is configured as an output line in master mode and an input line in slave mode. The MOSI line on the master (data out) should be connected to the MOSI line in the slave device (data in). The data is transferred as byte wide (8-bit) serial data.

SCLK (Serial Clock I/O) Pin

The master serial clock (SCLK) synchronizes the data being transmitted and received during the MOSI SCLK period. Therefore, a byte is transmitted/received after eight SCLK periods. The SCLK pin is configured as an output in master mode and as an input in slave mode.

In the master mode, the polarity and phase of the clock are controlled by the `SPI_CTL` register, and the bit rate is defined in the `SPI_DIV` register as follows:

$$f_{SERIALCLOCK} = \frac{PCLK}{2 \times (1+ SPIx\_DIV)}$$

The maximum data rate is 13 Mbps for a maximum PCLK frequency of 26 MHz.

In the slave mode, the `SPI_CTL` register must be configured with the phase and polarity of the expected input clock. The slave accepts data from an external master up to 13 Mbps.

In both master and slave modes, data is transmitted on one edge of the SCLK signal and sampled on the other. Therefore, the polarity and phase are configured the same for the master and slave devices.

Chip Select (CS I/O) Pin

In SPI slave mode, a transfer is initiated by the assertion of CS, which is an active low input signal. The SPI port then transmits and receives 8-bit data until the transfer is concluded by deassertion of CS. In slave mode, CS is always an input.

In SPI master mode, the CS is an active low output signal. It asserts itself automatically at the beginning of a transfer and deasserts itself upon completion.

In a multi-slave environment, we would support up to 4 different slaves. The SPI master can be configured to drive up to four CS lines (CS0, CS1, CS2, and CS3) using the `SPI_CS_CTL` register. Multiple CS lines can be driven simultaneously for a broadcast access. There are also override fields to enable software driving of 0 or 1 on the active CS line(s), which may be required for some special use cases. All other SPI pins are shared across by the slaves.

SPI Operating Modes

The SPI supports the following features.

Wired-OR Mode (WOM)

To prevent contention when the SPI is used in multislave system, the data output pins, MOSI and MISO, can be configured to behave as open-circuit drivers. An external pull-up resistor is required when this feature is selected. The `SPI_CTL.WOM` bit in the control register controls the pad enable outputs for the data lines.

General Instructions

1. SPI module operates on PCLK. So, if PCLK is stopped, this block does not work.
2. Whenever SPI configuration needs to be changed, ensure that the CS is de-asserted. This avoids any abrupt breaks in the SPI transfers. If the configuration changes while SPI transfer is actively going on, the behavior of the peripheral is nondeterministic.
3. While changing configurations, program `SPI_CNT` register only after changing `SPI_CTL` and `SPI_IEN` registers. Else, transfers may get started even before the configuration is changed.

Power-down Mode

In master mode, before entering power-down mode, the user should ensure that the transfers are completed by checking the appropriate interrupts/status bits. The SPI block should then be disabled by clearing the `SPI_CTL.SPIEN` bit. Only then, the power-down entry would be clean.

In slave mode, in either mode of operation (interrupt driven or DMA), the CS line level must be checked using the `SPI_STAT.CS` bit to ensure that the SPI is not communicating. The SPI block must be disabled only when the CS line is high.

While being powered down, the following fields are retained:

- All the bit fields of the `SPI_CTL` register except `SPI_CTL.SPIEN`. During power-up, the `SPI_CTL.SPIEN` bit is reset. This allows a fresh start of the peripheral at wake up.
- `SPI_IEN.IRQMODE` bit
- `SPI_DIV.VALUE` bit
- `SPI_RD_CTL.THREEPIN` bit
- `SPI_FLOW_CTL.RDYPOL` bit

All the other fields are not retained. They are reset on power-up. On exiting the power-down mode, the software must reprogram all the non-retained registers as required. The SPI block must be re-enabled by setting the `SPI_CTL.SPIEN` bit.

SPIH vs SPI0/SPI1

From an SPI programming and users model perspective, SPI0, SPI1, and SPI2 are identical. The main difference between SPI2 and SPI0/1 is the internal bus interface, which they are connected to. SPI2 is connected to a higher performance Advanced Peripheral Bus (APB) which is always clocked at the higher system clock rate (HCLK) and contains fewer modules requiring arbitration. Therefore, SPI2 is also know as SPIH, from SPI High performance. SPI0 and SPI1 are connected to the main APB which selectively can be clocked at a lower rate (PCLK) and whose latency is more uncertain due to a greater number of modules requiring arbitration. This means that under higher data rates, SPIH can move data more efficiently and with lower latency. SPIH is recommended for use with high data rate peripherals.

Interfacing with SPI Slaves

Though the SPI transfers are usually full duplex, in many cases, the slave works on a protocol which consists of Command, Address, and Data Read/Write. The write command is always unidirectional. However, the read command needs a Tx transfer followed by an Rx transfer in a single CS frame. An example protocol is shown below.

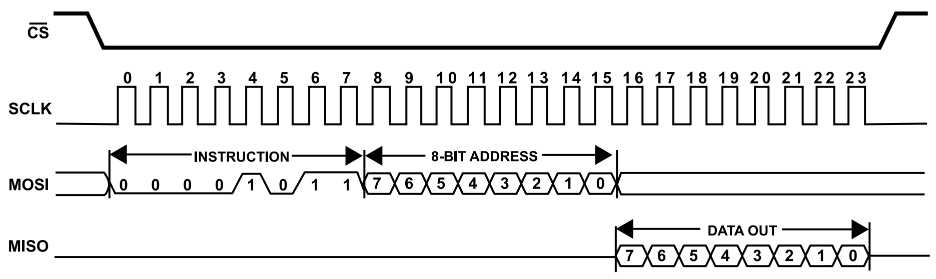


Figure 15-2: SPI Register Read

To support transfers like the above one, which resembles a half duplex operation, `SPI_RD_CTL.CMDEN` bit must be set. The number of bytes to be transmitted should be programmed in `SPI_RD_CTL.TXBYTES` field (which is 1 for the above example). The number of bytes to be received (after completing the transmission) would be specified by the `SPI_CNT.VALUE` bits. In this case, `SPI_CNT.VALUE` must be 1.

The `SPI_RD_CTL.OVERLAP` bit specifices if the bytes received while transmitting the command and address bytes must be stored or ignored. If this bit is set, the `SPI_CNT.VALUE` bit refers to the total number of bytes in the entire frame. Some SPI read examples are given below.

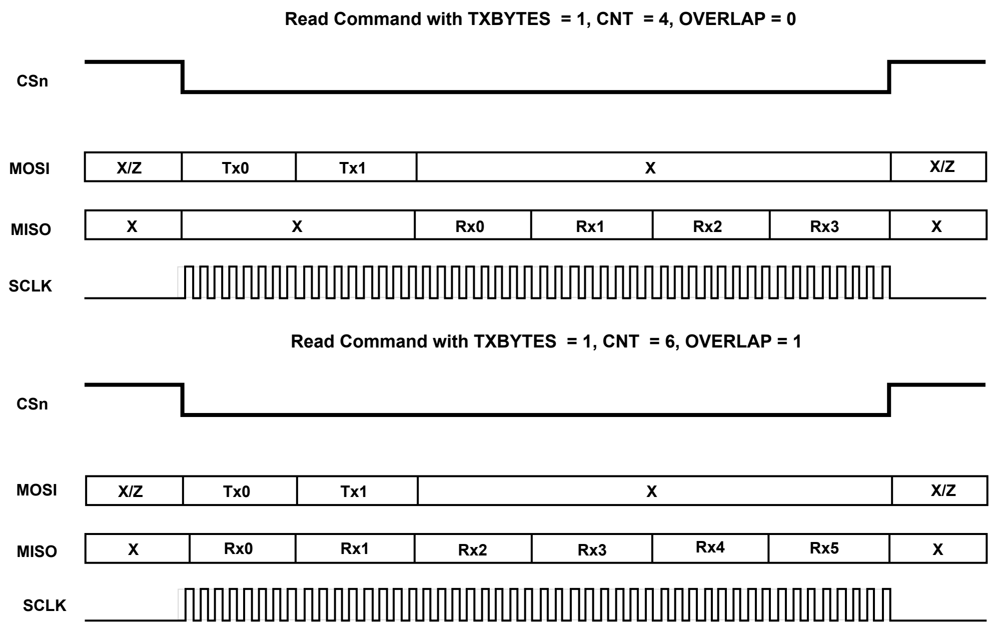

Read Command with TXBYTES = 1, CNT = 4, OVERLAP = 0
CSn
MOSI
X/Z
Tx0
Tx1
X
X/Z
MISO
X
X
Rx0
Rx1
Rx2
Rx3
X
SCLK
Read Command with TXBYTES = 1, CNT = 6, OVERLAP = 1
CSn
MOSI
X/Z
Tx0
Tx1
X
X/Z
MISO
X
Rx0
Rx1
Rx2
Rx3
Rx4
Rx5
X
SCLK

Figure 15-3: SPI Read Examples

Pseudo Code Example for Read Command Mode

```
SPI0_DIV = 0x0001          //SPI serial clk frequency = ¼ of PCLK freq
SPI0_CTL = 0x0883         //Enable SPI in master mode, ZEN=1,
                        //Continuous mode, TIM=0
SPI0_CNT = 0x0040       //64 bytes to be Rxed
SPI0_DMA = 0x0005      //Enable DMA mode and Rx-DMA request
SPI0_RD_CTL = 0x000D  //TXBYTES=3, CMDEN=1, Write 4 Tx bytes
                    //in two 16-bit writes (DMA mode)
SPI0_TX = 0xB6A5
SPI0_TX = 0xD8C7
read_data = SPI0_RX  //Do a dummy read of the Rx FIFO to
                   //initiate SPI transfers
```

In this case, only 4 bytes are transmitted as the `SPI_RD_CTL.TXBYTES` field is 3. However, zeroes are sent on MOSI for the next 64 bytes that are being received. In total, 4 bytes are transmitted and 64 bytes are received in a non-overlapping mode.

Flow Control

Several converters use a flow control mechanism to match the required data/sample rate. The SPI master supports the following types of flow control:

- Using a 16-bit timer clocked at the serial clock rate to introduce wait states while reading data. The master waits until the timer ends and then reads `SPI_FLOW_CTL.RDBURSTSZ`+1 number of bytes. It goes back to wait state and restarts the timer. This continues until `SPI_CNT.VALUE` number of bytes are received.

- Using a separate RDY pin which is connected through one of the GPIOs. The master waits until it sees an active level in this pin. Once it detects the transition, it reads `SPI_FLOW_CTL.RDBURSTSZ`+1 number of bytes and then goes back to wait state until another active level is detected. This continues until `SPI_CNT.VALUE` number of bytes are received.
- Using the MISO pin. In this mode, the master waits for an active level on the MISO pin. Once it detects the transition, it reads `SPI_FLOW_CTL.RDBURSTSZ`+1 number of bytes and then goes back to wait state until another active level is detected. This continues until `SPI_CNT.VALUE` number of bytes are received.

In all the above cases, `SPI_CNT.VALUE` must be an integer multiple of `SPI_FLOW_CTL.RDBURSTSZ+1`. Some example flow controlled transfers are shown below.

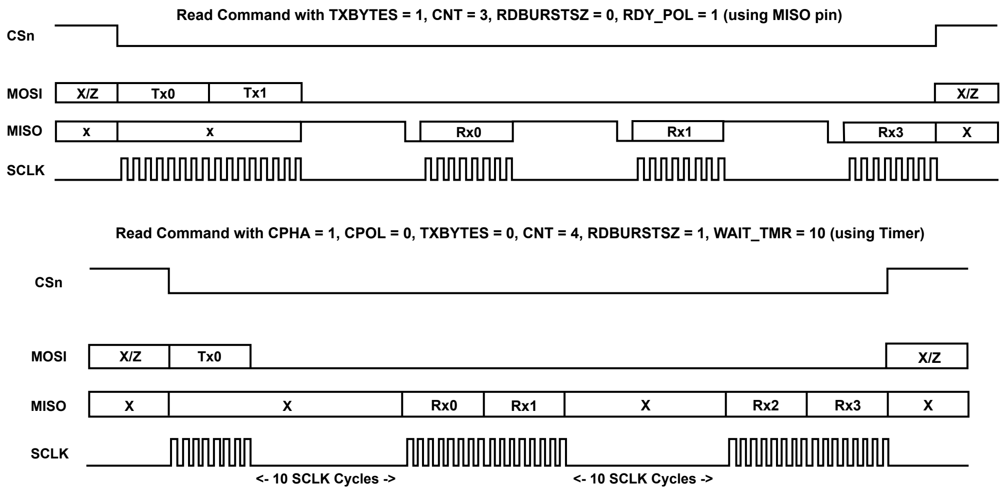

Read Command with TXBYTES = 1, CNT = 3, RDBURSTSZ = 0, RDY_POL = 1 (using MISO pin)
CSn
MOSI
X/Z
Tx0
Tx1
X/Z
MISO
X
X
Rx0
Rx1
Rx3
X
SCLK
Read Command with CPHA = 1, CPOL = 0, TXBYTES = 0, CNT = 4, RDBURSTSZ = 1, WAIT_TMR = 10 (using Timer)
CSn
MOSI
X/Z
Tx0
X/Z
MISO
X
X
Rx0
Rx1
X
Rx2
Rx3
X
SCLK
<- 10 SCLK Cycles ->
<- 10 SCLK Cycles ->

Figure 15-4: Flow Controlled Transfers

NOTE: While stalling the SCLK output for flow control or FIFO data/space unavailability, the last SCLK edge is always a sampling edge. This is to avoid a driving edge before the stall period. That way, the slave will have an SCLK driving edge to proceed with, after the stall period.

Though the SCLK signal idles low for a `SPI_CTL.CPOL` =0 and idles high for a `SPI_CTL.CPOL` = 1, the `SPI_CTL.CPHA` bit determines the sequence of sampling and driving edges. If `SPI_CTL.CPHA` = 1, the SCLK signal is stalled at the same level as the idle level. If `SPI_CTL.CPHA` = 0, the SCLK is stalled at the opposite level to the idle level. At the end of a transfer (when CS is deasserted), the SCLK is always idled as per `SPI_CTL.CPOL`. The ***Flow Control*** table explains the same.

Table 15-1: Flow Control

CPHA	CPOL	SCLK Idle level	SCLK Stalled level
0	0	0	1
0	1	1	0
1	0	0	0
1	1	1	1

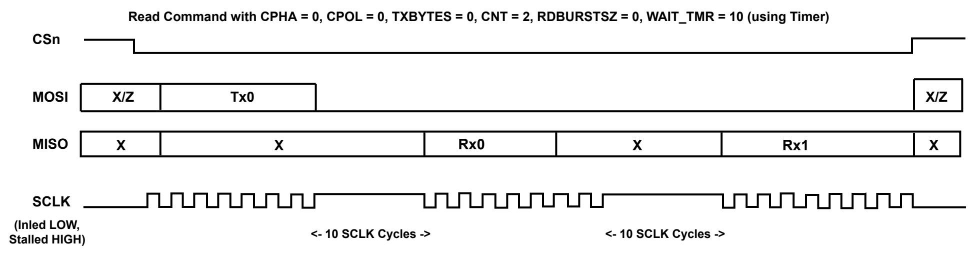

Read Command with CPHA = 0, CPOL = 0, TXBYTES = 0, CNT = 2, RDBURSTSZ = 0, WAIT_TMR = 10 (using Timer)
CSn
MOSI
X/Z
Tx0
X/Z
MISO
X
X
Rx0
X
Rx1
X
SCLK
(Inled LOW, Stalled HIGH)
<- 10 SCLK Cycles ->
<- 10 SCLK Cycles ->

Figure 15-5: Sample Transfer

Three-Pin Mode

SPI can be used in a 3-pin mode where the data transmission/reception occurs over a single bidirectional line. The SPI master supports this for read command mode, where the MOSI line is used for transmission of *Command + Address* bytes and the same line is used for receiving the read data. To enable this, `SPI_RD_CTL.THREEPIN` and `SPI_RD_CTL.CMDEN` fields must be set. Inherently, there is a half SCLK cycle between the last sampling edge of transmit phase and the first driving edge of receive phase. However, if the slave needs a larger turn around time, then it should enable timer based flow control and program the `SPI_WAIT_TMR.VALUE` as required. An example 3-pin SPI transfer is shown below.

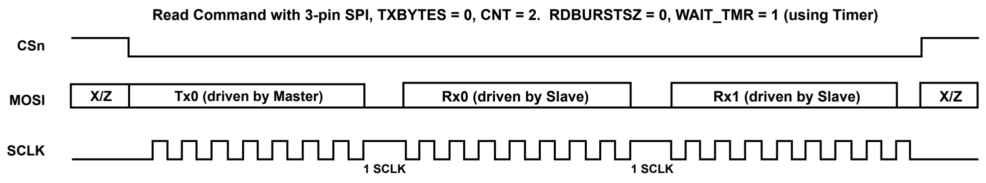


Figure 15-6: 3-Pin SPI Transfer

SPI Data Transfer

In master mode, the transfer and interrupt mode bit (`SPI_CTL.TIM`) determines the manner in which an SPI serial transfer is initiated. If the `SPI_CTL.TIM` bit is set, a serial transfer is initiated after a write to the Tx FIFO occurs. If the `SPI_CTL.TIM` bit is cleared, a serial transfer is initiated after a read of the Rx FIFO. The read must be done while the SPI interface is idle. A read done during an active transfer will not initiate another transfer.

For any setting of the master mode enable (`SPI_CTL.MASEN`) and `SPI_CTL.TIM` bits, the SPI simultaneously receives and transmits data (if `SPI_RD_CTL.CMDEN` bit is 0). Therefore, during data transmission, the SPI is also receiving data and filling up the Rx FIFO. If the data is not read from the Rx FIFO, the overflow interrupt occurs once the FIFO starts to overflow. If the user does not want to read the Rx data or receive overflow interrupts, the Rx FIFO flush enable (`SPI_CTL.RFLUSH`) bit can be set, and the receive data will not be saved to the Rx FIFO. Else, if the user just wants to read the Rx data (not concerned about the overflow condition), this interrupt can be disabled by clearing the `SPI_IEN.RXOVR` bit. Similarly, when the user only wants to receive data and does not want to write data to the Tx FIFO, the Tx FIFO flush enable (`SPI_CTL.TFLUSH`) bit can be set to avoid getting underflow interrupts from the Tx FIFO. Alternatively, if the user wants to send the FIFO data, but, does not want underflow interrupts, the `SPI_IEN.TXUNDR` bit must be cleared.

Transmit Initiated Transfer

For transfers initiated by a write to the transmit FIFO, the SPI transmits as soon as the first byte is written to the FIFO. The SPI transfer of the first byte happens immediately.

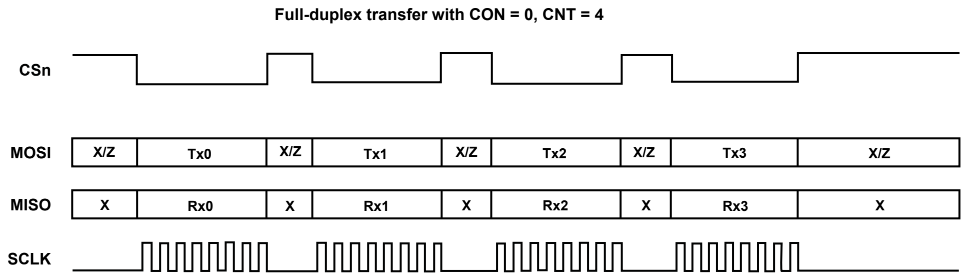
If the continuous transfer enable bit (`SPI_CTL.CON`) is set, the transfer continues until it is complete. This completion is either the end of `SPI_CNT.VALUE` number of bytes (if `SPI_CNT.VALUE` > 0) or when no valid data is available in the transmit FIFO (if `SPI_CNT.VALUE` = 0). Chip Select remains asserted for the duration of the complete transfer. If `SPI_CNT.FRAMECONT` is cleared and `SPI_CNT.VALUE` > 0, the transfer stops when all the `SPI_CNT.VALUE` bytes have been transferred. If `SPI_CNT.FRAMECONT` is set, a new frame starts after every `SPI_CNT.VALUE` number of bytes. Therefore, multiples of `SPI_CNT.VALUE` bytes are transferred. If there is no data/space in FIFO, the transfer stalls until it is available. Conversely, the transfer continues when there is valid data in the FIFO.

If `SPI_CTL.CON` is cleared, each transfer consists of a single 8-bit serial transfer. If valid data exists in the transmit FIFO, a new transfer is initiated after a stall period, where Chip Select is deasserted.

Receive Initiated Transfer

Transfers initiated by a read of the receive FIFO depend on the number of bytes to be received in the FIFO. If `SPI_IEN.IRQMODE` is set to 7 and a read to the receive FIFO occurs, the SPI master initiates an 8-byte transfer. If continuous mode is set (`SPI_CTL.CON`), the 8 bytes happen continuously with no deassertion of Chip Select between bytes. If continuous mode is not set, the 8 bytes will happen with stall periods between transfers, where the chip select will be deasserted. However, in continuous mode, if `SPI_CNT.VALUE` > 0, then CS will be asserted for the entire frame duration. SPI will introduce stall periods by not clocking SCL until FIFO space is available.

If `SPI_IEN.IRQMODE` is set to 6, then a read of the receive FIFO will initiate a 7 byte transfer similar to above. If `SPI_IEN.IRQMODE` is set to 1, then a read of the receive FIFO will initiate a 2 byte transfer. Finally, a read of the FIFO with `SPI_IEN.IRQMODE` set to 0 will initiate a single byte transfer. A read of the receive FIFO while the SPI is receiving data will not initiate another transfer after the present transfer is complete. In continuous mode, if `SPI_CNT.VALUE` > 0 and `SPI_CNT.FRAMECONT` = 1, read of receive FIFO at the end of an SPI frame (to get the last set of bytes received) will always initiate a new SPI frame. Therefore, to stop SPI transfers at any given frame, `SPI_CNT.FRAMECONT` bit should be cleared before reading the final set of receive bytes.



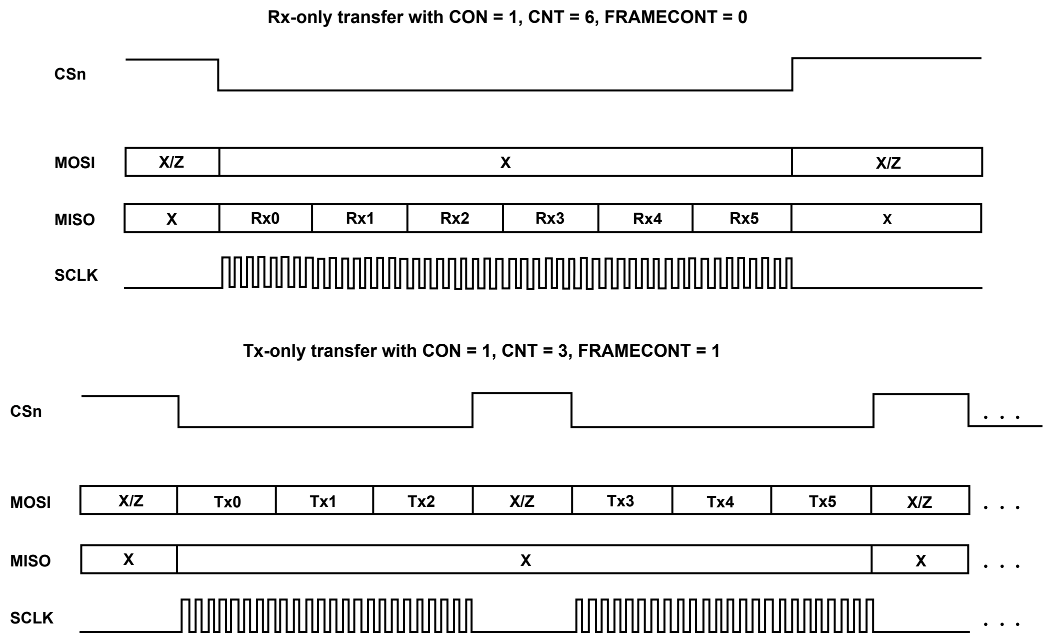

Rx-only transfer with CON = 1, CNT = 6, FRAMECONT = 0
CSn
MOSI
X/Z
X
X/Z
MISO
X
Rx0
Rx1
Rx2
Rx3
Rx4
Rx5
X
SCLK
Tx-only transfer with CON = 1, CNT = 3, FRAMECONT = 1
CSn
MOSI
X/Z
Tx0
Tx1
Tx2
X/Z
Tx3
Tx4
Tx5
X/Z
MISO
X
X
X
SCLK

Figure 15-7: SPI Transfers

Transfers in Slave Mode

In slave mode, a transfer is initiated by the assertion of the Chip Select of the device.

Though the master can support upto four CS output lines, only one CS input, CS0, is used in slave mode.

The device as a slave transmits and receives 8-bit data until the transfer is concluded by the deassertion of Chip Select.

The following SPI transfer protocol figures show the data transfer protocol for SPI and the effects of `SPI_CTL.CPHA` and `SPI_CTL.CPOL` bits in the control register on that protocol, as indicated by T1, T2, and T3 (See Figures ***SPI Transfer Protocol CPHA = 0*** and ***SPI Transfer Protocol CPHA = 1***).

NOTE: Chip Select must not be tied to the ground.

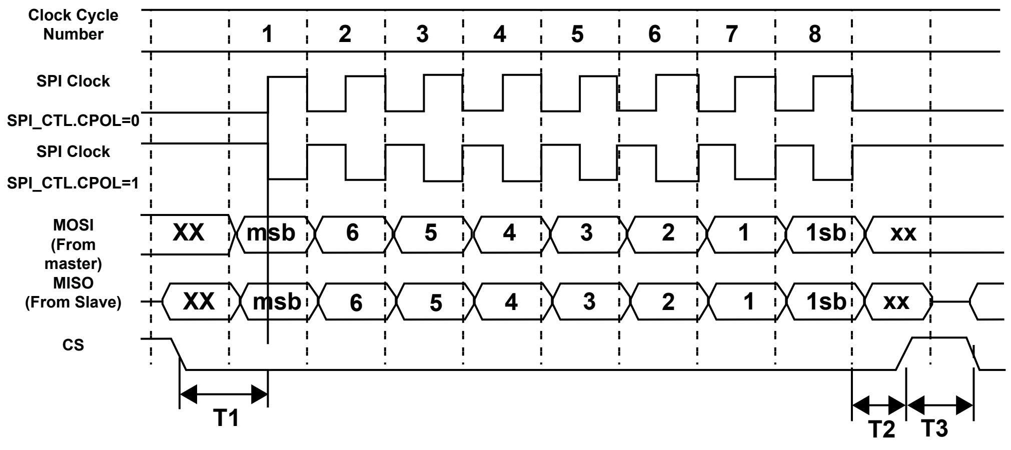


Figure 15-8: SPI Transfer Protocol CPHA = 0

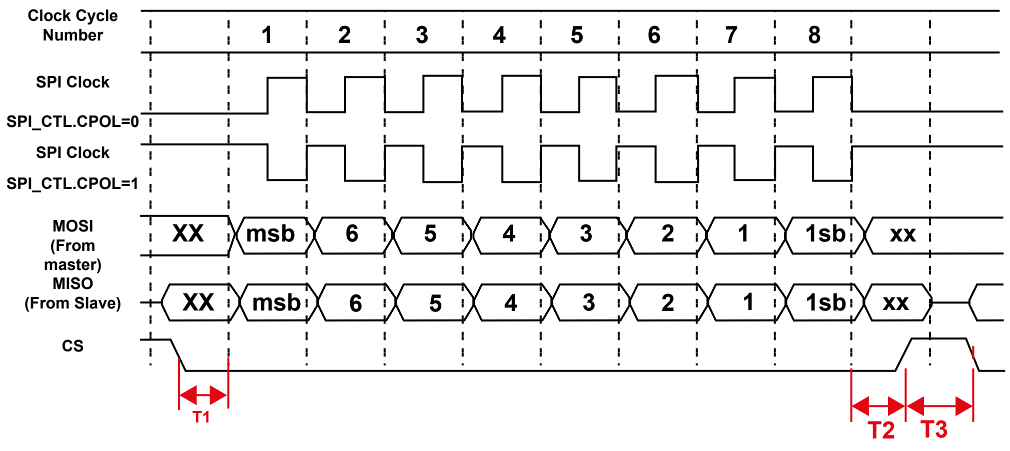


Figure 15-9: SPI Transfer Protocol CPHA = 1

SPI Data Underflow and Overflow

If the Tx transmit zeroes underflow mode bit (`SPI_CTL.ZEN`) is cleared, the last stale byte is shifted out when a transfer is initiated with no valid data in the FIFO. If `SPI_CTL.ZEN` is set, zeros are transmitted when a transfer is initiated with no valid data in the FIFO.

If the Rx overflow overwrite enable bit (`SPI_CTL.RXOF`) is set, the valid data in the Rx FIFO is overwritten by the new serial byte received when there is no space left in the FIFO. If `SPI_CTL.RXOF` is cleared, the new serial byte received is discarded when there is no space left in the FIFO.

When valid data is being overwritten in the Rx FIFO, the oldest byte is overwritten first followed by the next oldest byte and so on.

SPI Interrupts and Exceptions

There is one interrupt line per SPI and eleven sources of interrupts.

The SPI_STAT.IRQ bit reflects the state of the interrupt line.

The SPI_STAT[15:12] and SPI_STAT[7:1] bits reflect the state of the eleven sources.

The SPI generates either Transmit Interrupt Requests (TIRQ) or Receive Interrupt Request (RIRQ). Both interrupts cannot be enabled at the same time. The appropriate interrupt is enabled using the SPI_CTL.TIM bit. If SPI_CTL.TIM = 1, TIRQ is enabled. If SPI_CTL.TIM = 0, RIRQ is enabled.

All interrupts are sticky and are cleared only when the appropriate interrupt bits in SPI_STAT register are written with a 1. The interrupt line from the device is cleared only after all the interrupt sources are cleared.

Transmit Interrupt

If the SPI_CTL.TIM bit is set, the transmit FIFO status causes the interrupt. The SPI_IEN.IRQMODE bit control when the interrupt occurs.

When the SPI_IEN.IRQMODE bits are set to:

000: An interrupt is generated after each byte that is transmitted. The interrupt occurs when the byte is read from the FIFO and written to the shift register.

001: An interrupt is generated after every two bytes that are transmitted.

...

110: An interrupt occurs after every seven bytes that are transmitted.

111: An interrupt occurs after every eight bytes that are transmitted.

The interrupts are generated depending on the number of bytes transmitted and not on the number of bytes in the FIFO. This is different from receive interrupt, which depends on the number of bytes in the receive FIFO and not the number of bytes received.

The status of this interrupt can be read by reading the SPI_STAT.TXIRQ bit. The interrupt is disabled if transmit FIFO flush enable (SPI_CTL.TFLUSH) is high.

NOTE: Write to the SPI_CTL register resets the transmitted byte counter back to zero. For example, when the SPI_IEN.IRQMODE bit is set to 0x3 and after three bytes have been transmitted, the SPI_CTL register is re-initialized, the transmit interrupt does not occur until another 4 bytes have been transmitted.

Receive Interrupt

If SPI_CTL.TIM is cleared, the receive FIFO status causes the interrupt. Again, SPI_IEN.IRQMODE controls when the interrupt will occur. The status of this interrupt can be read by reading the SPI_STAT.RXIRQ bit.

Interrupts are only generated when data is written to the FIFO. For example, if SPI_IEN.IRQMODE is set to 0x0, an interrupt is generated after the first byte is received. If the interrupt was cleared by a write-1 and the data byte was not read from the receive FIFO, the interrupt will not be regenerated. Another interrupt is generated only when another byte is received into the FIFO.

The interrupt depends on the number of valid bytes in FIFO when the SPI is receiving data. It does not depend on the number of bytes received over the SPI.

The interrupt is disabled if `SPI_CTL.RFLUSH` is left high.

Underflow/Overflow Interrupts

When a transfer starts with no data in the transmit FIFO, the `SPI_STAT.TXUNDR` bit of the Status register gets set to indicate an underflow condition. This causes an interrupt if `SPI_STAT.TXUNDR` is set. This interrupt occurs irrespective of what `SPI_CTL.TIM` bit is set to. This interrupt is disabled if `SPI_CTL.TFLUSH` bit is set.

If data is received when the receive FIFO is already full, this causes the `SPI_STAT.RXOVR` bit to go high indicating an overflow condition. This causes an interrupt if `SPI_IEN.RXOVR` is set. This interrupt occurs irrespective of what `SPI_CTL.TIM` bit is set to. This interrupt is disabled if `SPI_CTL.RFLUSH` bit is set.

The receive and transmit interrupts are cleared if the relevant flush bits are asserted or if the SPI is disabled. Otherwise, the interrupts remain active even when the SPI is reconfigured.

SPI Programming Model

The following section provides the SPI DMA details.

SPI DMA

Two DMA channels are dedicated to SPI, transmit and receive. The two SPI DMA channels must be configured in the DMA controller.

It is possible to enable DMA request on 1 or 2 channels at the same time, by setting the DMA request bits for receive or transmit in the `SPI_DMA` register. If only the DMA transmit request (`SPI_DMA.TXEN`) is enabled, the receive FIFO will overflow during SPI transfer unless received data is read by user code, and an overflow interrupt will be generated. To avoid generating overflow interrupts, the receive FIFO flush bit (`SPI_CTL.RFLUSH`) should be set, or the `SPI_IEN.RXOVR` bit should be cleared, or the SPI interrupt be disabled in the NVIC of the core. If only the DMA receive request (`SPI_DMA.RXEN`) is enabled, the transmit FIFO underflows. Again, to avoid underflow interrupt, the `SPI_IEN.TXUNDR` bit must be cleared or the SPI interrupt must be disabled in the NVIC.

The SPI transmit and receive interrupts are not generated when DMA is used. `SPI_IEN.IRQMODE` is not used in transmit mode and must be set to 3'b000 in receive mode.

The DMA bit (`SPI_DMA.TXEN`) controls the start of a DMA transfer. DMA requests are only generated when `SPI_DMA.EN` = 1. At the end of a DMA transfer, that is, when receiving a DMA SPI transfer interrupt, this bit needs to be cleared to prevent extra DMA requests to the DMA controller. The data still present in the transmit FIFO is transmitted if in transmit mode.

All DMA data transfers are 16-bit transfers, and the DMA should be programmed accordingly. For example, if 16 bytes of data are to be transferred over the SPI, the DMA should be programmed to perform eight half-word (16-bit) transfers. If 17 bytes are to be transferred, nine half-word transfers are required, the additional byte is discarded. Data errors occur if the DMA transfers are programmed as byte-wide transfers.

In DMA mode, the transmit/receive FIFOs are 2 bytes wide. Bits[7:0] are first accessed by the SPI followed by the bits[15:8]. This is irrespective of count or `SPI_CTL.LSB` settings. For example, if `SPI_CNT.VALUE` = 3, the order of transmission/reception is as follows (Byte-3 is ignored).

Byte-1	Byte-0
Byte-3	Byte-2

NOTE: The `SPI_CTL.LSB` bit does not affect the FIFO access order in the DMA mode. It only affects how each byte is transferred over SPI.

DMA Master Transmit Configuration

The DMA SPI transmit channel must be configured. The NVIC must be configured to enable DMA transmit master interrupt.

The SPI block must be configured as follows:

```
SPI_DIV = SPI_SERIAL_FREQ;  //configures serial clock frequency.
SPI_CTL = 0x1043;          //enables SPI in master mode and transmit mode, receive
FIFO
                          //flush enabled.

SPI_CNT.VALUE = TxBytes; //sets the number of bytes to transfer.
SPI_DMA = 0x1;          //(optional) enables FIFO to accept 16-bit
                       //core data writes.
SPI_TX = 0xXXXX;      //(optional) up to four 16-bit core writes can be performed
                     //to preload FIFO.
SPI_DMA = 0x3;      //enable DMA mode, enable transmit DMA request.
```

All DMA transfers are expected to be 16-bit transfers. When all data present in the DMA buffer are transmitted, the DMA generates an interrupt. User code should disable DMA request. Data will still be in the transmit FIFO as the DMA request is generated each time there is free space in the transmit FIFO, to keep the FIFO always full.

DMA Master Receive Configuration

The `SPI_CNT` register sets the number of receive bytes required by the SPI master. When the required number of bytes has been received, no more transfers are initiated. To initiate a DMA master receive transfer, a dummy read should be done by user code. This dummy read must not be added to the `SPI_CNT` number.

The counter counting the bytes as they are received is reset when SPI is disabled using the `SPI_CTL.SPIEN` bit, or if the `SPI_CNT` register is modified by user code.

To perform SPI DMA master receive, the DMA SPI receive channel must be configured. The NVIC of the core must be configured to enable DMA receive master interrupt.

The SPI block must be configured as follows:

```
SPI_DIV = SPI_SERIAL_FREQ; //configures serial clock frequency.
SPI_CTL = 0x2003;         //enable SPI in master mode and
```

```
                              //receive mode, 1 byte transfer.
SPI_DMA = 0x5;              //enable DMA mode, enable receive DMA request.
SPI_CNT.VALUE = XXX;       //number of bytes to be received.
A = SPI_RX;               //dummy read.
```

The DMA transfer stops when the appropriate number of clock cycles has been generated. All DMA data transfers are 16-bit transfers, and the DMA should be programmed accordingly. For example, if 16 bytes of data are to be received over the SPI, the DMA should be programmed to perform eight 16-bit transfers. If 17 bytes are to be received, nine 16-bit transfers are required. The additional byte is padded for the final DMA transfer. Data errors occur if the DMA transfers are programmed as byte wide transfers.

NOTE: DMA buffer must be of the same size as `SPI_CNT.VALUE` (or same size plus one if `SPI_CNT.VALUE` is odd) to generate a DMA interrupt when the transfer is complete.

ADuCM302x SPI Register Descriptions

Serial Peripheral Interface (SPI) contains the following registers.

Table 15-2: ADuCM302x SPI Register List

Name	Description
SPI_CNT	Transfer Byte Count
SPI_CS_CTL	Chip Select Control for Multi-slave Connections
SPI_CS_OVERRIDE	Chip Select Override
SPI_CTL	SPI Configuration
SPI_DIV	SPI Baud Rate Selection
SPI_DMA	SPI DMA Enable
SPI_FIFO_STAT	FIFO Status
SPI_FLOW_CTL	Flow Control
SPI_IEN	SPI Interrupts Enable
SPI_RD_CTL	Read Control
SPI_RX	Receive
SPI_STAT	Status
SPI_TX	Transmit
SPI_WAIT_TMR	Wait Timer for Flow Control

Transfer Byte Count

This register is only used in master mode.

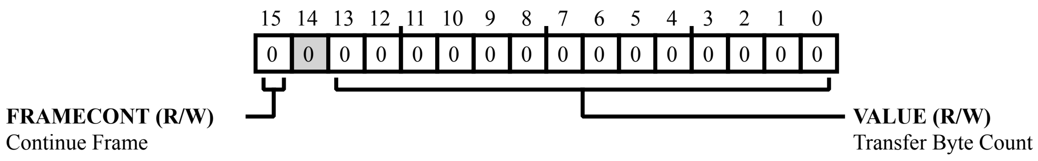


Figure 15-10: SPI_CNT Register Diagram

Table 15-3: SPI_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	FRAMECONT	Continue Frame. This bit should be used in conjunction with `SPI_CTL.CON` and `SPI_CNT.VALUE` fields. It is used to control SPI data framing. If this bit is cleared, the SPI master transfers only one frame of `SPI_CNT.VALUE` bytes. If set, the SPI master will transfer data in frames of `SPI_CNT.VALUE` bytes each. Note: If `SPI_CNT.VALUE` = 0, this field has no effect as the SPI master will continue with transfers as long as Tx/Rx FIFO is ready. If `SPI_CTL.CON` = 0, this field has no effect as all SPI frames are single byte wide irrespective of other control fields.	
		0	If `SPI_CNT.VALUE` > 0, stop SPI transfers after `SPI_CNT.VALUE` number of bytes.
		1	Continue SPI transfers as long as Tx/Rx FIFO is ready.
13:0 (R/W)	VALUE	Transfer Byte Count. This field specifies the number of bytes to be transferred. It is used in both receive and transmit transfer types. This value assures that a master mode transfer terminates at the proper time and that 16-bit `SPI_DMA` transfers are byte padded or discarded as required to match odd transfer counts.	

Chip Select Control for Multi-slave Connections

This register is only used in master mode.

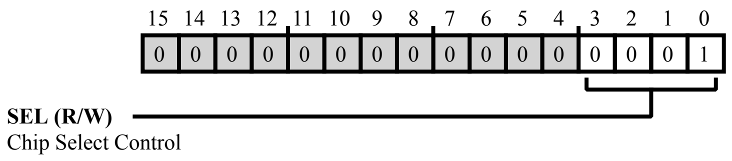


Figure 15-11: SPI_CS_CTL Register Diagram

Table 15-4: SPI_CS_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	SEL	Chip Select Control. This field specifies the CS line to be used for the current SPI transfer. It is useful in a multi-slave setup where only the CS lines are unique across the various slaves. The SPI master can support up to 4 different CS lines. By default, CS0 is used if none of the bits are set. Note: If multiple bits are set, the respective CS lines are active simultaneously.

Chip Select Override

This register is only used in master mode.

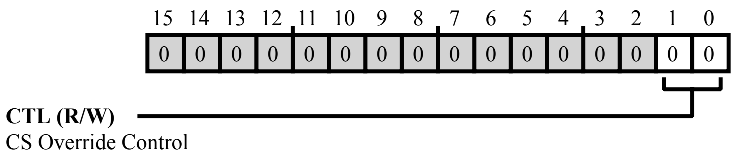


Figure 15-12: SPI_CS_OVERRIDE Register Diagram

Table 15-5: SPI_CS_OVERRIDE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	CTL	CS Override Control. This bit overrides the actual CS output from the master state machine. It may be needed for special SPI transfers. Note: Use it with precaution.	
		0	CS is not forced.
		1	CS is forced to drive 1'b1.
		2	CS is forced to drive 1'b0.
		3	CS is not forced.

SPI Configuration

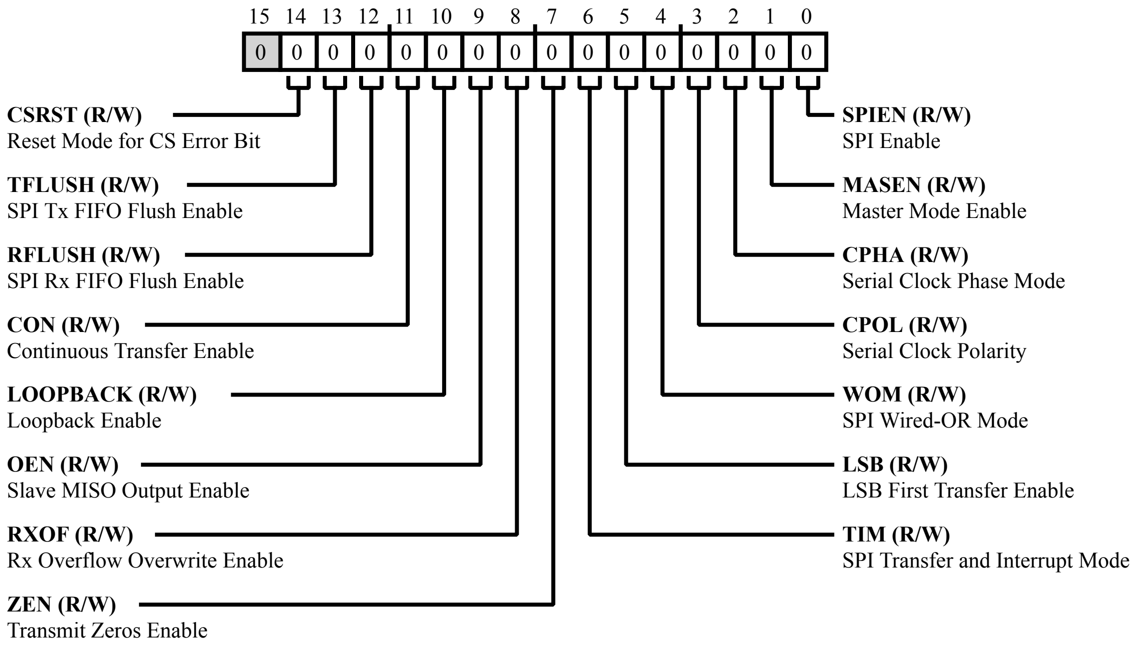


Figure 15-13: SPI_CTL Register Diagram

Table 15-6: SPI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W)	CSRST	Reset Mode for CS Error Bit. If this bit is set, the bit counter will be reset after a CS error condition and the Cortex is expected to clear the `SPI_CTL.SPIEN`. If this bit is clear, the bit counter will continue from where it stopped. SPI can receive the remaining bits when CS gets asserted and Cortex has to ignore the `SPI_STAT.CSERR` interrupt. However, it is recommended to set this bit for recovery after a CS error.
13 (R/W)	TFLUSH	SPI Tx FIFO Flush Enable. Set this bit to flush the Tx FIFO. This bit does not clear itself and should be toggled if a single flush is required. If this bit is left high, then either the last transmitted value or 0x00 is transmitted depending on the `SPI_CTL.ZEN` bit. Any writes to the Tx FIFO are ignored while this bit is set. Clear this bit to disable Tx FIFO flushing.

Table 15-6: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	RFLUSH	SPI Rx FIFO Flush Enable. Set this bit to flush the Rx FIFO. This bit does not clear itself and should be toggled if a single flush is required. If this bit is set all incoming data is ignored and no interrupts are generated. If set and `SPI_CTL.TIM` = 0, a read of the Rx FIFO will initiate a transfer. Clear this bit to disable Rx FIFO flushing.	
11 (R/W)	CON	Continuous Transfer Enable. Set by user to enable continuous transfer. In master mode, the transfer continues until no valid data is available in the `SPI_TX` register (`SPI_CTL.TIM`=1) or until the Rx FIFO is full (`SPI_CTL.TIM`=0). CS is asserted and remains asserted for the duration of each 8-bit serial transfer until Tx FIFO is empty or Rx FIFO is full. Cleared by user to disable continuous transfer. Each SPI frame then consists of a single 8-bit serial transfer. If valid data exists in the `SPI_TX` register (`SPI_CTL.TIM`=1) or Rx FIFO is not full (`SPI_CTL.TIM`=0), a new transfer is initiated after a stall period of 1 serial clock cycle.	
10 (R/W)	LOOPBACK	Loopback Enable. Set by user to connect MISO to MOSI and test software. Cleared by user to be in normal mode.	
9 (R/W)	OEN	Slave MISO Output Enable. Set this bit for MISO to operate as normal. Clear this bit to disable the output driver on the MISO pin. The MISO pin will be Open-Circuit when this bit is clear.	
8 (R/W)	RXOF	Rx Overflow Overwrite Enable. Set by user, the valid data in the `SPI_RX` register is overwritten by the new serial byte received. Cleared by user, the new serial byte received is discarded.	
7 (R/W)	ZEN	Transmit Zeros Enable. Set this bit to transmit 0x00 when there is no valid data in the Tx FIFO. Clear this bit to transmit the last transmitted value when there is no valid data in the Tx FIFO.	
6 (R/W)	TIM	SPI Transfer and Interrupt Mode. Set by user to initiate transfer with a write to the `SPI_TX` register. Interrupt only occurs when `SPI_IEN.IRQMODE`+1 number of bytes have been transmitted. Cleared by user to initiate transfer with a read of the `SPI_RX` register. Interrupt only occurs when Rx-FIFO has `SPI_IEN.IRQMODE`+1 number of bytes or more.	
5 (R/W)	LSB	LSB First Transfer Enable.	
		0	MSB transmitted first
		1	LSB transmitted first

Table 15-6: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	WOM	SPI Wired-OR Mode.	
		0	Normal output levels
		1	Enables open circuit data output enable. External pull-ups required on data out pins
3 (R/W)	CPOL	Serial Clock Polarity.	
		0	Serial clock idles low
		1	Serial clock idles high
2 (R/W)	CPHA	Serial Clock Phase Mode.	
		0	Serial clock pulses at the end of each serial bit transfer
		1	Serial clock pulses at the beginning of each serial bit transfer
1 (R/W)	MASEN	Master Mode Enable. Note: Clearing this bit issues a synchronous reset to the design and most status bits, while other MMRs are unaffected.	
		0	Enable slave mode
		1	Enable master mode
0 (R/W)	SPIEN	SPI Enable.	
		0	Disable the SPI
		1	Enable the SPI

SPI Baud Rate Selection

This register is only used in master mode.

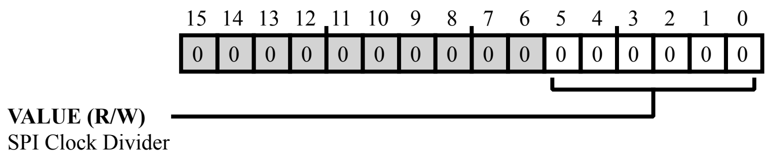


Figure 15-14: SPI_DIV Register Diagram

Table 15-7: SPI_DIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5:0 (R/W)	VALUE	SPI Clock Divider. `SPI_DIV.VALUE` is the factor used to divide PCLK to generate the serial clock.

SPI DMA Enable

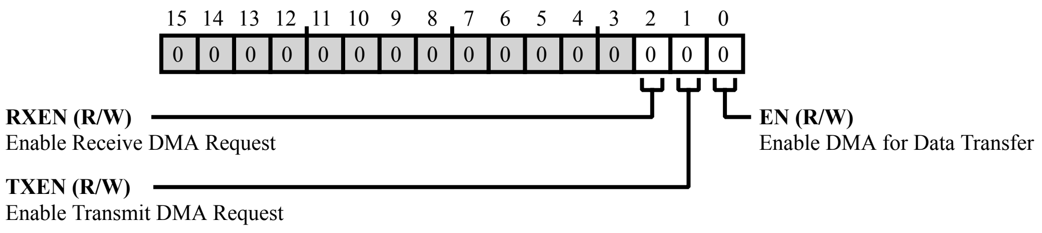


Figure 15-15: SPI_DMA Register Diagram

Table 15-8: SPI_DMA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	RXEN	Enable Receive DMA Request. If this bit is set when DMA is enabled, then a Rx DMA request is raised as long there is valid data in Rx FIFO.
1 (R/W)	TXEN	Enable Transmit DMA Request. If this bit is set and DMA is enabled, a Tx DMA request is raised as long there is space in Tx FIFO. Note: This bit needs to be cleared as soon as a Tx DMA DONE interrupt is received to prevent extra Tx DMA requests to the DMA controller.
0 (R/W)	EN	Enable DMA for Data Transfer. Set by user code to start a DMA transfer. Cleared by user code at the end of DMA transfer.

FIFO Status

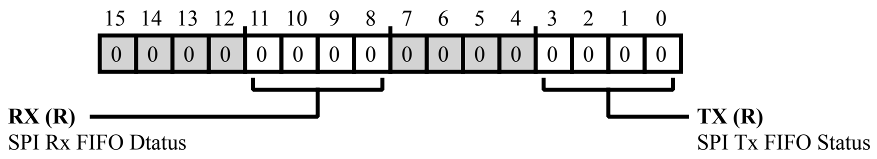


Figure 15-16: SPI_FIFO_STAT Register Diagram

Table 15-9: SPI_FIFO_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:8 (R/NW)	RX	SPI Rx FIFO Dtatus. This field specifies the number of bytes in Rx FIFO when DMA is disabled. In DMA mode, it refers to the number of half-words in Rx FIFO.	
		0	Rx FIFO empty
		1	1 valid byte/half-word in Rx FIFO
		2	2 valid bytes/half-words in Rx FIFO
		3	3 valid bytes/half-words in Rx FIFO
		4	4 valid bytes/half-words in Rx FIFO
		5	5 valid bytes/half-words in Rx FIFO
		6	6 valid bytes/half-words in Rx FIFO
		7	7 valid bytes/half-words in Rx FIFO
		8	8 valid bytes/half-words in Rx FIFO (Rx FIFO full)
3:0 (R/NW)	TX	SPI Tx FIFO Status. This field specifies the number of bytes in Tx FIFO when DMA is disabled. In DMA mode, it refers to the number of half-words in Tx FIFO.	
		0	Tx FIFO empty
		1	1 valid byte/half-word in Tx FIFO
		2	2 valid bytes/half-words in Tx FIFO
		3	3 valid bytes/half-words in Tx FIFO
		4	4 valid bytes/half-words in Tx FIFO
		5	5 valid bytes/half-words in Tx FIFO
		6	6 valid bytes/half-words in Tx FIFO
		7	7 valid bytes/half-words in Tx FIFO
		8	8 valid bytes/half-words in Tx FIFO (Tx FIFO full)

Flow Control

This register is only used in master mode.

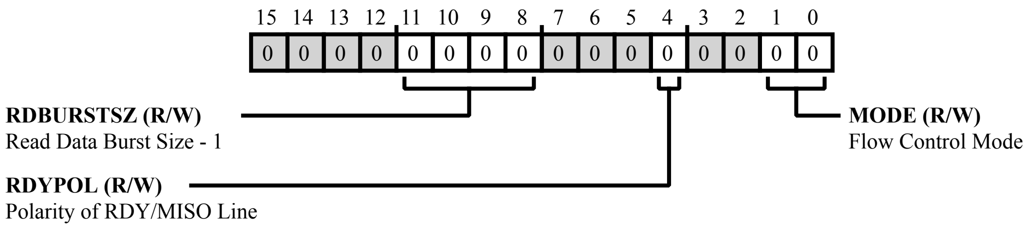


Figure 15-17: SPI_FLOW_CTL Register Diagram

Table 15-10: SPI_FLOW_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:8 (R/W)	RDBURSTSZ	Read Data Burst Size - 1. Specifies number of bytes to be received - 1 in a single burst from a slave before waiting for flow-control. This is not valid if `SPI_FLOW_CTL.MODE` is 2'b00. For all other values of `SPI_FLOW_CTL.MODE`, this field is valid. It takes values from 0 to 1023 implying a read burst of 1 to 1024 bytes respectively. Note: This mode is useful for reading fixed-width conversion results periodically.	
4 (R/W)	RDYPOL	Polarity of RDY/MISO Line. Specifies the polarity of the RDY/MISO pin, which indicates that the slave's read data is ready. If `SPI_FLOW_CTL.MODE`= 2'b10, it refers to the polarity of RDY pin. If `SPI_FLOW_CTL.MODE`=2'b11, it refers to the polarity of MISO (DOUT) line. For all other values of `SPI_FLOW_CTL.MODE`, this bit is ignored.	
		0	Polarity is active high. SPI master waits until RDY/MISO becomes high.
		1	Polarity is active low. SPI master waits until RDY/MISO becomes low.

Table 15-10: SPI_FLOW_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	MODE	Flow Control Mode. Flow control configuration for data reads. Note: When RDY signal is used for flow-control, it could be any signal which is tied to this RDY input of SPI module. For example, it can be an off-chip input, on-chip timer output, or any other control signal.	
		0	Flow control is disabled.
		1	Flow control is based on timer (WAIT_TMR).
		2	Flow control is based on RDY signal.
		3	Flow control is based on MISO pin.

SPI Interrupts Enable

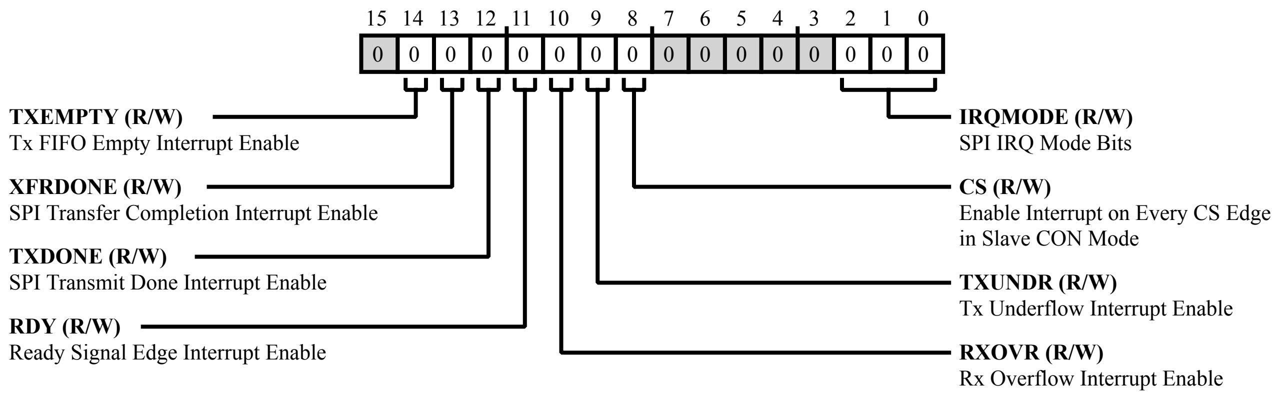

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TXEMPTY (R/W)
Tx FIFO Empty Interrupt Enable
XFRDONE (R/W)
SPI Transfer Completion Interrupt Enable
TXDONE (R/W)
SPI Transmit Done Interrupt Enable
RDY (R/W)
Ready Signal Edge Interrupt Enable
IRQMODE (R/W)
SPI IRQ Mode Bits
CS (R/W)
Enable Interrupt on Every CS Edge in Slave CON Mode
TXUNDR (R/W)
Tx Underflow Interrupt Enable
RXOVR (R/W)
Rx Overflow Interrupt Enable

Figure 15-18: SPI_IEN Register Diagram

Table 15-11: SPI_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	TXEMPTY	Tx FIFO Empty Interrupt Enable. This bit enables the `SPI_STAT.TXEMPTY` interrupt whenever Tx FIFO gets emptied.	
		0	TXEMPTY interrupt is disabled.
		1	TXEMPTY interrupt is enabled.
13 (R/W)	XFRDONE	SPI Transfer Completion Interrupt Enable. This bit enables the `SPI_STAT.XFRDONE` interrupt.	
		0	XFRDONE interrupt is disabled.
		1	XFRDONE interrupt is enabled.
12 (R/W)	TXDONE	SPI Transmit Done Interrupt Enable. This bit enables the `SPI_STAT.TXDONE` interrupt in read command mode. Note: This can be used to signal the change of SPI transfer direction in read command mode.	
		0	TXDONE interrupt is disabled.
		1	TXDONE interrupt is enabled.

Table 15-11: SPI_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	RDY	Ready Signal Edge Interrupt Enable. Enables the `SPI_STAT.RDY` interrupt whenever an active edge occurs on RDY/ MISO signals. If `SPI_FLOW_CTL.MODE` = 2'b10, edge detection happens on RDY signal. If `SPI_FLOW_CTL.MODE` = 2'b11, MISO signal is used instead of RDY. For all other values of `SPI_FLOW_CTL.MODE`, this bit has no effect. The active edge (rising/falling) is determined by the `SPI_FLOW_CTL.RDYPOL` bit.	
		0	RDY signal edge interrupt is disabled.
		1	RDY signal edge interrupt is enabled.
10 (R/W)	RXOVR	Rx Overflow Interrupt Enable.	
		0	Rx overflow interrupt is disabled.
		1	Rx overflow interrupt is enabled.
9 (R/W)	TXUNDR	Tx Underflow Interrupt Enable.	
		0	Tx underflow interrupt is disabled.
		1	Tx underflow interrupt is enabled.
8 (R/W)	CS	Enable Interrupt on Every CS Edge in Slave CON Mode. If this bit is set and the SPI module is configured as a slave in continuous mode, any edge on CS will generate an interrupt and the corresponding status bits (`SPI_STAT.CSRISE`, `SPI_STAT.CSFALL`) will be asserted. If this bit is not set, then no interrupt will be generated and the status bits will not be asserted. This bit has no effect if the SPI is not in continuous mode or if it is a master.	
2:0 (R/W)	IRQMODE	SPI IRQ Mode Bits. These bits configure when the Tx/Rx interrupts occur in a transfer. For DMA Rx transfer, these bits must be 3'b000.	
		0	Tx interrupt occurs when 1 byte has been transferred. Rx interrupt occurs when 1 or more bytes have been received into the FIFO.
		1	Tx interrupt occurs when 2 bytes have been transferred. Rx interrupt occurs when 2 or more bytes have been received into the FIFO.
		2	Tx interrupt occurs when 3 bytes have been transferred. Rx interrupt occurs when 3 or more bytes have been received into the FIFO.

Table 15-11: SPI_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		3	Tx interrupt occurs when 4 bytes have been transferred. Rx interrupt occurs when 4 or more bytes have been received into the FIFO.
		4	Tx interrupt occurs when 5 bytes have been transferred. Rx interrupt occurs when 5 or more bytes have been received into the FIFO.
		5	Tx interrupt occurs when 6 bytes have been transferred. Rx interrupt occurs when 6 or more bytes have been received into the FIFO.
		6	Tx interrupt occurs when 7 bytes have been transferred. Rx interrupt occurs when 7 or more bytes have been received into the FIFO.
		7	Tx interrupt occurs when 8 bytes have been transferred. Rx interrupt occurs when the FIFO is full.

Read Control

This register is only used in master mode.

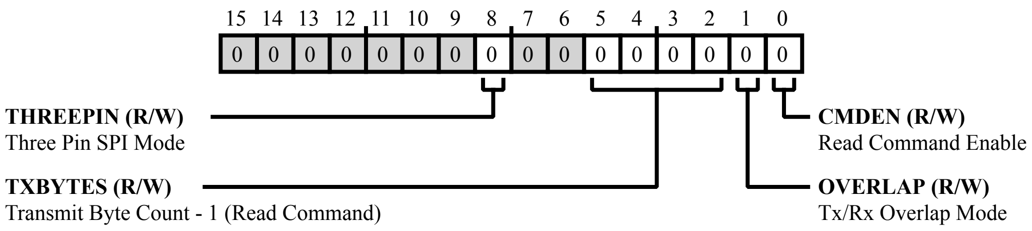


Figure 15-19: SPI_RD_CTL Register Diagram

Table 15-12: SPI_RD_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	THREEPIN	Three Pin SPI Mode. Specifies if the SPI interface has a bidirectional data pin (3-pin interface) or dedicated unidirectional data pins for Tx and Rx (4-pin interface). This is only valid in Read-command mode and when `SPI_FLOW_CTL.MODE` = 2'b01. If 3-pin mode is selected, MOSI pin will be driven by master during transmit phase and after a wait time of `SPI_WAIT_TMR` SCLK cycles, the slave is expected to drive the same MOSI pin. Note: `SPI_FLOW_CTL.MODE` should be programmed to 2'b01 to introduce wait states for allowing turnaround time. Else, the slave only has a turn-around time of half SCLK period (between sampling and driving edges of SCLK). If this mode is used, set `SPI_RD_CTL.OVERLAP` = 0.	
		0	SPI is a 4-pin interface.
		1	SPI is a 3-pin interface.
5:2 (R/W)	TXBYTES	Transmit Byte Count - 1 (Read Command). Specifies the number of bytes to be transmitted - 1 before reading data from a slave. This field can take values from 0 to 15 corresponding to 1 to 16 Tx bytes . This includes all the bytes that need to be sent out to the slave viz., command and address (if required). The design does not differentiate between these two. It just transmits the specified number of bytes from the Tx FIFO. Note: If there is a latency between the command transmission and data reception, then the number of Tx bytes (mostly 0's) to be padded for that delay should also be accounted for in this count.	

Table 15-12: SPI_RD_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	OVERLAP	Tx/Rx Overlap Mode. This bit specifies if the start of Tx and Rx overlap. In most of the slaves, the read data starts only after the master completes the transmission of 'command+address'. This is a non-overlapping transfer. In some slaves, there might be status bytes sent out while the command is being received. So, user may need to start receiving the bytes from the beginning of the CS frame. This is overlapping mode. Note: In case of overlapping mode, SPI_CNT.VALUE refers to the total number of bytes to be received. So, the extra status bytes (which are in addition to the actual read bytes) should be accounted for while programming SPI_CNT.VALUE.	
		0	Tx/Rx overlap is disabled.
		1	Tx/Rx overlap is enabled.
0 (R/W)	CMDEN	Read Command Enable. SPI read command mode where a command + address is transmitted and read data is expected in the same CS frame. If this bit is cleared, all other fields of SPI_RD_CTL, SPI_FLOW_CTL and SPI_WAIT_TMR registers do not have any effect.	
		0	Read command mode is disabled.
		1	Read command mode is enabled.

Receive

This register allows access to the 8-deep receive FIFO.

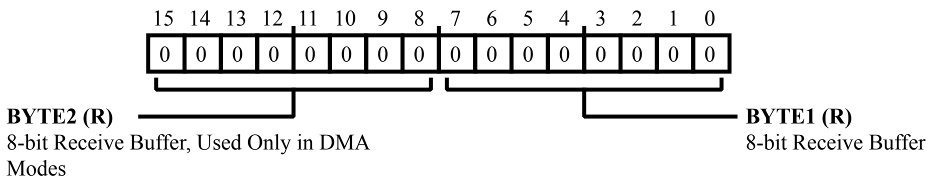


Figure 15-20: SPI_RX Register Diagram

Table 15-13: SPI_RX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/NW)	BYTE2	8-bit Receive Buffer, Used Only in DMA Modes. These 8-bits are used only in the SPI_DMA mode, where all FIFO accesses happen as half-word access. They return zeros if SPI_DMA is disabled.
7:0 (R/NW)	BYTE1	8-bit Receive Buffer.

Status

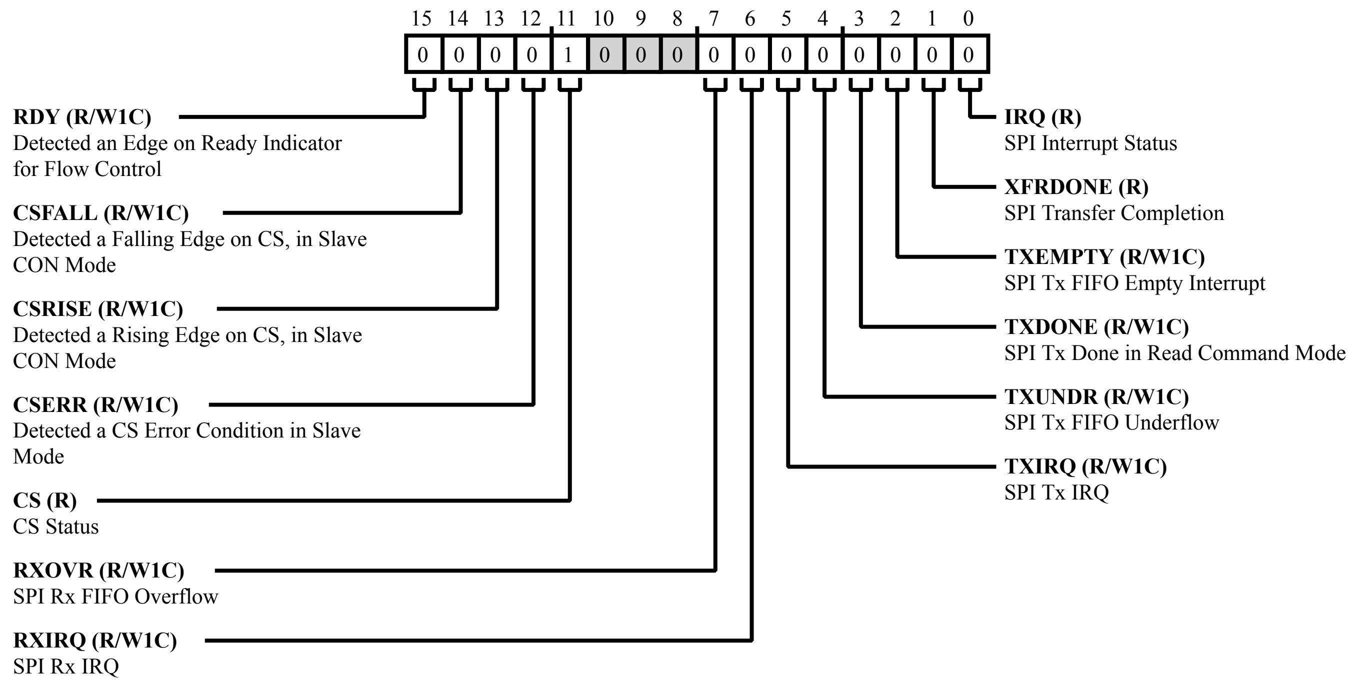


Figure 15-21: SPI_STAT Register Diagram

Table 15-14: SPI_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	RDY	Detected an Edge on Ready Indicator for Flow Control. This bit indicates that there was an active edge on the RDY/MISO line depending on the flow control mode. If `SPI_FLOW_CTL.MODE` = 2'b10, this bit is set whenever an active edge is detected on RDY signal. If `SPI_FLOW_CTL.MODE` = 2'b11, this bit is set if an active edge is detected on MI-SO signal. For all other values of MODE, this bit is always 0. The active edge (rising/falling) is determined by the `SPI_FLOW_CTL.RDYPOL`. When `SPI_IEN.RDY` is set, this bit will cause an interrupt, and it is cleared only when 1 is written to this bit. Note: This can be used for staggered flow-control on transmit side, along with a CS override if required.

Table 15-14: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W1C)	CSFALL	Detected a Falling Edge on CS, in Slave CON Mode. This bit indicates that there was a falling edge in CS line, when the device was a slave in continuous mode and `SPI_IEN.CS` was asserted. This bit will cause an interrupt. It is cleared when 1 is written to this bit or when `SPI_CTL.SPIEN` is cleared. This can be used to identify the start of an SPI data frame.	
13 (R/W1C)	CSRISE	Detected a Rising Edge on CS, in Slave CON Mode. This bit indicates that there was a rising edge in CS line, when the device was a slave in continuous mode and `SPI_IEN.CS` was asserted. This bit will cause an interrupt. It is cleared when 1 is written to this bit or when `SPI_CTL.SPIEN` is cleared. This can be used to identify the end of an SPI data frame.	
12 (R/W1C)	CSERR	Detected a CS Error Condition in Slave Mode. This bit indicates that the CS line was de-asserted abruptly by an external master, even before the full-byte of data was transmitted completely. This bit will cause an interrupt. It is cleared when 1 is written to this bit or when `SPI_CTL.SPIEN` is cleared.	
11 (R/NW)	CS	CS Status. This bit reflects the actual CS status as seen by the SPI module. Note: This uses SCLK-PCLK synchronization. So, there would be a slight delay when CS changes state.	
		0	CS line is low.
		1	CS line is high.
7 (R/W1C)	RXOVR	SPI Rx FIFO Overflow. Set when the Rx FIFO was already full when new data was loaded to the FIFO. This bit generates an interrupt if `SPI_IEN.RXOVR` is set, except when `SPI_CTL.RFLUSH` is set. It is cleared when 1 is written to this bit or `SPI_CTL.SPIEN` is cleared.	
6 (R/W1C)	RXIRQ	SPI Rx IRQ. Set when a receive interrupt occurs. Not available in DMA mode. This bit is set when `SPI_CTL.TIM` is cleared and the required number of bytes have been received. It is cleared when 1 is written to this bit or `SPI_CTL.SPIEN` is cleared.	
5 (R/W1C)	TXIRQ	SPI Tx IRQ. SPI Tx IRQ Status Bit. Not available in DMA mode. Set when a transmit interrupt occurs. This bit is set when `SPI_CTL.TIM` is set and the required number of bytes have been transmitted. It is cleared when 1 is written to this bit or `SPI_CTL.SPIEN` is cleared.	

Table 15-14: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W1C)	TXUNDR	SPI Tx FIFO Underflow. This bit is set when a transmit is initiated without any valid data in the Tx FIFO. This bit generates an interrupt if `SPI_IEN.TXUNDR` is set, except when `SPI_CTL.TFLUSH` is set. It is cleared when 1 is written to this bit or `SPI_CTL.SPIEN` is cleared.
3 (R/W1C)	TXDONE	SPI Tx Done in Read Command Mode. This bit is set when the entire transmit is completed in a read command. This bit generates an interrupt if `SPI_IEN.TXDONE` is set. This bit is valid only if `SPI_RD_CTL.CMDEN` is set. It is cleared only when 1 is written to this bit.
2 (R/W1C)	TXEMPTY	SPI Tx FIFO Empty Interrupt. This bit is set when the Tx-FIFO is empty and `SPI_IEN.TXEMPTY` is set, except when `SPI_CTL.TFLUSH` is set. This bit generates an interrupt. It is cleared when 1 is written to this bit or `SPI_CTL.SPIEN` is cleared.
1 (R/NW)	XFRDONE	SPI Transfer Completion. This bit indicates the status of SPI transfer completion in master mode. It is set when the transfer of `SPI_CNT.VALUE` number of bytes have been finished. In slave mode or if `SPI_CNT.VALUE` = 0, this bit is invalid. If `SPI_IEN.XFRDONE` is set, this bit generates an interrupt. It uses the state of the master state machine to determine the completion of a SPI transfer. Therefore, a CS override does not affect this bit. It is cleared only when 1 is written to this bit.
0 (R/NW)	IRQ	SPI Interrupt Status. Set to 1 when an SPI based interrupt occurs. Cleared when all the interrupt sources are cleared.

Transmit

This register allows access to the 8-deep transmit FIFO.

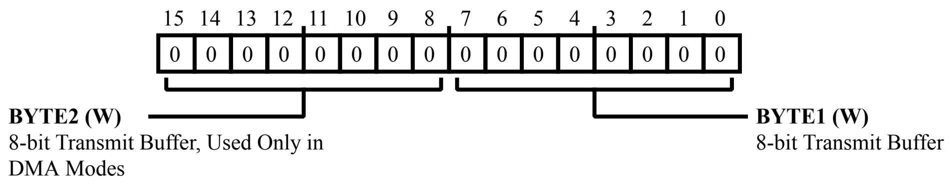


Figure 15-22: SPI_TX Register Diagram

Table 15-15: SPI_TX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (RX/W)	BYTE2	8-bit Transmit Buffer, Used Only in DMA Modes. These 8-bits are used only in the `SPI_DMA` mode, where all FIFO accesses happen as half-word access.
7:0 (RX/W)	BYTE1	8-bit Transmit Buffer.

Wait Timer for Flow Control

This register is only used in master mode.

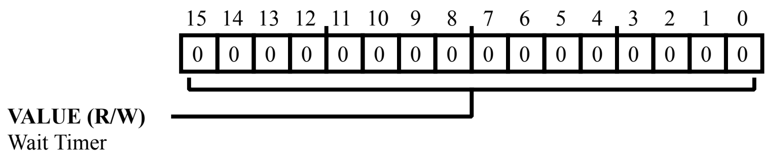


Figure 15-23: SPI_WAIT_TMR Register Diagram

Table 15-16: SPI_WAIT_TMR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Wait Timer. This field specifies the number of SCLK cycles to wait before continuing the SPI read. This field can take values of 0 to 65535. This field is only valid if `SPI_FLOW_CTL.MODE` = 2'b01. For all other values of `SPI_FLOW_CTL.MODE`, this field is ignored. A value of 0 implies a wait time of 1 SCLK cycle.

16 Serial Port (SPORT)

The ADuCM302x MCU has a serial port (SPORT) that support a variety of serial data communication protocols. In addition, the SPORTs provide a glueless hardware interface to several industry-standard data converters, codecs and other processors, including DSPs. The SPORT top module comprises of two half SPORTs (HSPORT) with identical functionality SPORT_A and SPORT_B. Each half SPORT can be independently configured as either a transmitter or receiver and can be coupled with the other HSPORT within the same SPORT. Each half SPORT has the same capabilities and is programmed in the same way.

SPORT Features

Each HSPORT supports the following features:

- One bidirectional data line, configurable as either transmitter or receiver. The two half SPORT can be combined to enable full duplex operation.
- Operation mode:
  - Standard DSP serial mode
  - Timer-enabled mode
- Serial data words between 4 and 32 bits in length.
- Improved granularity for internal clock generation, allowing even PCLK to SPT_CLK ratios. The SPORTs can also accept an input clock from an external source.
- Configurable rising or falling edge of the SPT_CLK for driving or sampling data and frame sync.
- Gated clock mode support for internal clock mode.
- Frame Sync options:
  - Unframed mode
  - Internal/external frame sync options
  - Programmable polarity
  - Early/Late Frame Sync

- Status flagging and optional interrupt generation for prematurely received external frame syncs.
- External frame sync signal configured as level-sensitive signal.
- Programmable bit order - MSB/LSB first.
- Programmable transfer count of 1-4095 words.
- Optional 16-bit to 32-bit word or 8-bit to 32-bit packing when SPORT is configured as receiver and 32-bit to16-bit or 32-bit to 8-bit word unpacking when configured as transmitter.
- Status flagging and optional interrupt generation for transmit underflow or receive overflow.
- Interrupt driven transfer support.
- Dedicated DMA channel for each half SPORT.
- Transfer Finish Interrupt (TFI): An interrupt is generated when the last bit of programmed number of transfers is transmitted out.
- Ability to route and share the clocks and/or frame sync between the SPORT halves of the SPORT module.
- SPT_CNV signal generation in timer-enable mode.
- Interrupt control
- One DMA channel per half SPORT.
- Each half SPORT has its own set of control registers and data buffers.
- Core can also access the data buffers.

Signal Description

Each half SPORT module has four dedicated pins, as described in the following table.

Table 16-1: SPORT Pin Descriptions

Internal Node	Direction	Description
SPT_CLK	I/O	Transmit/Receive Serial Clock. Data and Frame Sync are driven/sampled with respect to this clock. This signal can be either internally or externally generated.
SPT_FS	I/O	Transmit/Receive Frame Sync. The frame sync pulse initiates shifting of serial data. This signal is either generated internally or externally.
SPT_D0	I/O	Transmit/receive Data channel. Bidirectional data pin. This signal can be configured as an output to transmit serial data, or as an input to receive serial data.
SPT_CNV	Output	This is an additional control signal used only in the case of Timer enable mode for peripherals which require two signals for control information.

- The clock and frame sync signals can be interconnected between the half SPORT pair.
- The SPT_CNV signal is used only in timer enable mode.

SPORT Functional Description

The following section provides general information about functionality of the serial ports of the MCU.

SPORT Block Diagram

The figure shows the top-level block diagram of the SPORT.

It shows the interfacing with the APB and DMA channel. It also shows the connection with the peripheral.

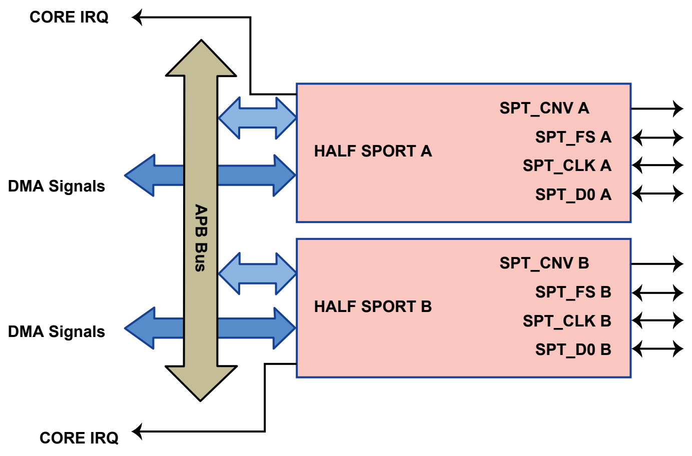


Figure 16-1: Top-level Block Diagram of SPORT

The following figure shows the block diagram of a half SPORT.

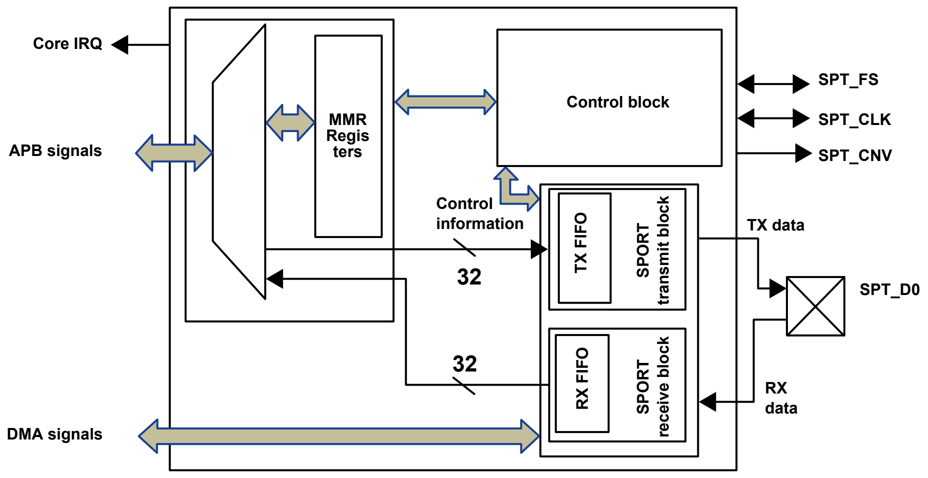


Figure 16-2: Half SPORT Block Diagram

Each SPORT module consists of two separate blocks, known as half SPORT (HSPORT) A and B, with identical functionality. These blocks can be independently configured as a transmitter or receiver, as shown in the top-level block diagram.

Each HSPORT has its own set of control registers and data buffers. The two HSPORTs must be combined to achieve full-duplex operation.

SPORT Transfer

The SPORT is configured in transmit mode, if SPORT_CTL_A.SPTRAN or SPORT_CTL_B.SPTRAN control bit is set. If this bit is cleared, serial port is configured in receive mode. Once a path is activated, data is shifted in response to a frame sync at the rate of serial clock. Receive data buffers are inactive when SPORT is operating in transmit mode and transmit data buffers when in receive mode. Inactive data buffers are not used and should not be accessed. An application program must use the appropriate data buffers.

Transmit Path

The SPORT_CTL_A.SPTRAN or SPORT_CTL_B.SPTRAN control bit, when set, configures the SPORT in transmit mode.

The SPORT_TX_A and SPORT_TX_B registers are used as the transmit data buffer (which is a three deep FIFO).

The data to be transmitted is written to the SPORT_TX_A and SPORT_TX_B registers to transmit data through the APB bus. This data is then transferred to the transmit shift register. The shift register, clocked by SPT_CLK signal, then serially shifts out this data on SPT_D0, synchronously. If framing signal is used, the SPT_FS signal indicates the start of the serial word transmission.

When SPORT is configured as transmitter, the enabled SPORT data pins SPT_D0 are always driven.

NOTE: If next frame sync is assigned after a time duration greater than serial word length, then during inactive serial clock cycles (clock cycles after data transmission in the current frame), the SPORT drives the first bit of next word to be transmitted on the data pin. This does not cause any problem at the receiver end, as it starts sampling the data pin only after detecting a valid frame sync. Note that this is not the case in gated clock mode as no clock is present during inactive frame sync.

The serial port provides status of transmit data buffers and error detection logic for transmit errors such as underrun. For more information, refer to Error Detection.

When a serial port is configured in transmit mode, the receive data path is deactivated and do not respond to serial clock or frame sync signals. So, reading from an empty Receive Data Buffer is not recommended in transmit mode.

Receive Path

The SPORT_CTL_A.SPTRAN bit, when cleared, configures the SPORT in receive mode.

The SPORT_RX_A and SPORT_RX_B registers, which are the receive data buffers (three deep FIFO), are accessible over APB.

In receive mode, the input shift register shifts in data bits on the SPT_D0, synchronous to the SPT_CLK. If framing signal is used, the SPT_AFS signal indicates the beginning of the serial word being received. When an entire word is shifted in on the channel, the data is available to be read from SPORT_RX_x.

The serial port provides the status of Receive Data buffers and error detection logic for receive errors such as overflow. For more information, refer to Error Detection.

When a serial port is configured in receive mode, the transmit path is deactivated and does not respond to serial clock or frame sync signals. Therefore, accessing the transmit data buffer is not recommended in receive mode.

Multiplexer Logic

There is a multiplexing block, known as SPMUX that is integrated between the SPORT and the PinMux logic. It allows flexibility to route and share the clock and frame sync signals between the two half SPORT pairs. This feature can be used to reduce the total number of pins for the interface and is efficient when the half SPORT pair is used for full-duplex operation.

`SPORT_CTL_A.CKMUXSEL` and `SPORT_CTL_A.FSMUXSEL` are used to configure this feature.

The figure below shows the operation of this multiplexer connected between half SPORTs clocks. Half SPORT A can be programmed to obtain clock from the neighboring half SPORT B.

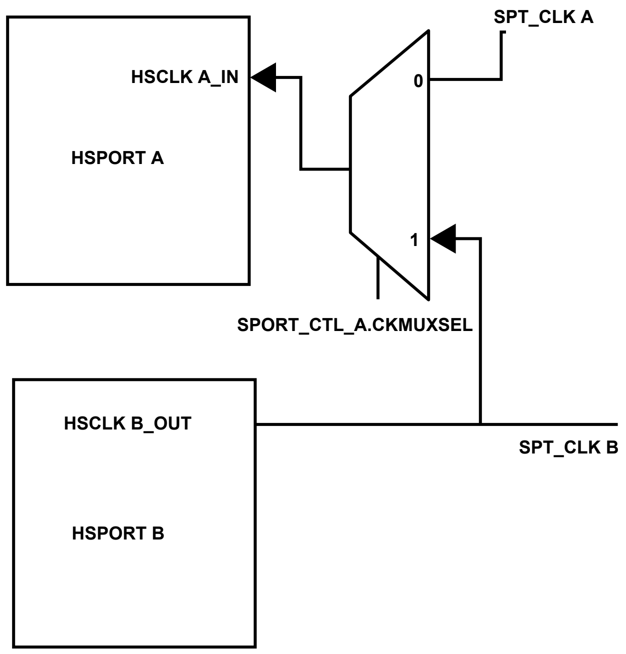


Figure 16-3: Multiplexer Logic When HSPORT A Uses HSPORT B's Internal Clock

If `SPORT_CTL_A.CKMUXSEL` is 1, it receives an internal clock of B when `SPORT_CTL_B.ICLK` value is 1.

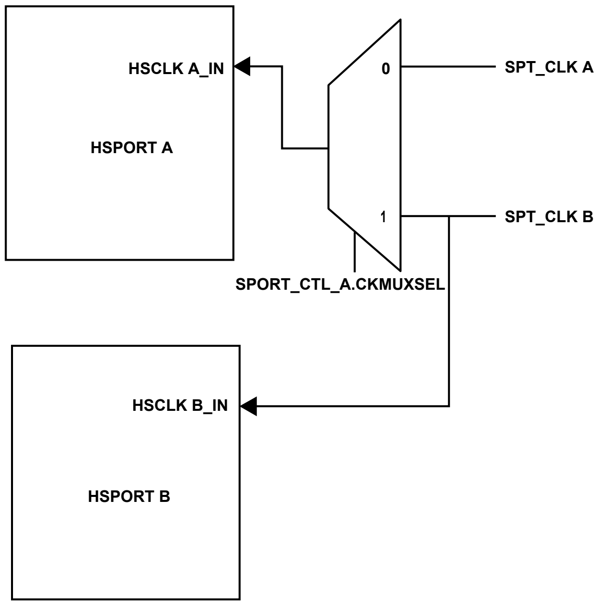


Figure 16-4: Multiplexer Logic When HSPORT A and HSPORT B Use Same External Clock

If `SPORT_CTL_A.CKMUXSEL` is 1 and `SPORT_CTL_B.ICLK` is 0, both half SPORTs get the same external clock.

If `SPORT_CTL_A.CKMUXSEL` is 0, normal operation of clock is carried out for both the half SPORTs.

The figure below shows the operation of the multiplexer connected between half SPORTs frame sync. Half SPORT A can be programmed to obtain frame sync from the neighboring half SPORT B.

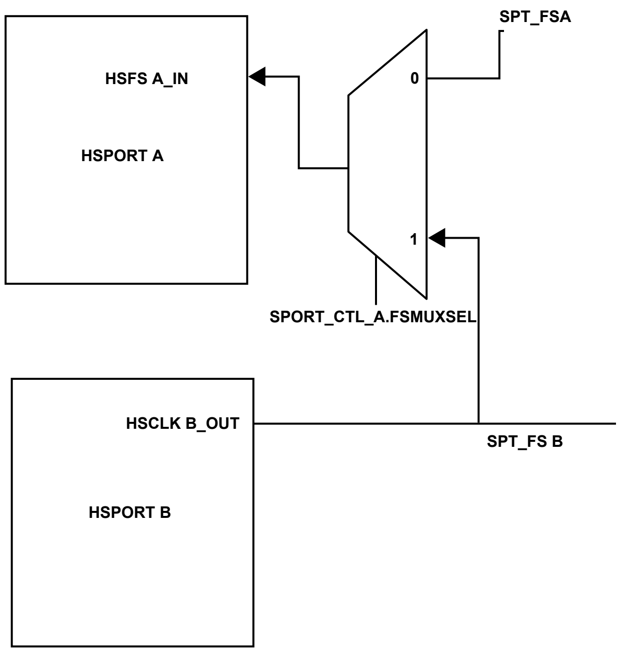


Figure 16-5: Multiplexer Logic When HSPORT A Uses HSPORT B's Internal Frame Sync

If SPORT_CTL_A.FSMUXSEL is 1, it gets internal frame sync of B when SPORT_CTL_B.IFS value is 1.

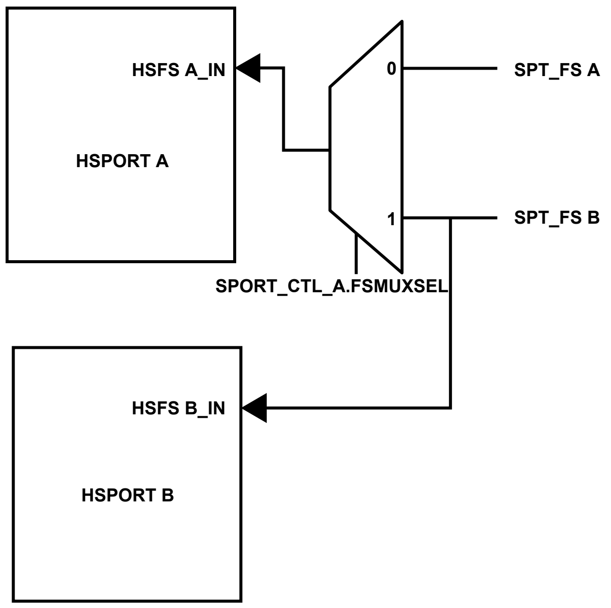


Figure 16-6: Multiplexer Logic When HSPORT A and HSPORT B Use Same External Frame Sync

If SPORT_CTL_A.FSMUXSEL is 1 and SPORT_CTL_B.IFS is 0, both half SPORTs get the same external frame sync.

If SPORT_CTL_A.FSMUXSEL is 0, normal operation of frame sync is carried out for both the half SPORTs.

Polarity bits such as SPORT_CTL_A.CKRE, SPORT_CTL_B.CKRE, SPORT_CTL_A.LFS, and SPORT_CTL_B.LFS must have identical settings when using muxing between two SPORT halves. The number of transfers (SPORT_NUMTRAN_A, SPORT_NUMTRAN_B) must also be programmed with the same number in both half SPORTs when same internal clock/frame sync is used in both halves.

NOTE: HSPORT A can import serial clock signal from HSPORT B, only when it is configured in external clock mode. Similarly, it can import frame sync signal, only when it is configured in external frame sync mode. SPORT_CTL_A register programming (for SPORT_CTL_A.CKMUXSEL and SPORT_CTL_A.FSMUXSEL) is required only for HSPORT A (not for HSPORT B) to enable this sharing. HSPORTB cannot import serial clock sgnal and frame sync signal from HSPORT A.

Serial Clock

The serial clock (SPT_CLK) signal controls how the data bits are driven or sampled. The frame sync signal is also driven (in internal frame sync mode) or sampled (in external frame sync mode) with respect to the serial clock signal. The serial clock can be internally generated from the MCU's system clock or externally provided, based on the SPORT_CTL_x.ICLK bit settings. If a SPORT is configured in internal clock mode (SPORT_CTL_x.ICLK= 1), the SPORT_DIV_x.CLKDIV field specifies the divider to generate serial port clock signal from its fundamental clock, PCLK. This divisor is a 16-bit value, allowing a wide range of serial clock rates.

The following equation is used to calculate the serial clock frequency:

$$f_{SPT\_CLK} = \frac{PCLK}{2x(1+SPORT\_DIV\_xCLKDIV)}$$

The maximum serial data rate is 13 Mbps for a maximum PCLK frequency of 26 MHz.

In gated clock mode, the serial port can be configured to generate gated clock which is active only for the duration of valid data.

If a SPORT is configured in external clock mode (`SPORT_CTL_x.ICLK` = 0), then serial clock is an input signal and the SPORT operates in slave mode. The `SPORT_DIV_x.CLKDIV` is ignored.

The externally supplied serial clock is always considered as an asynchronous clock with respect to the MCU system clock. For exact AC timing specifications, refer to the *ADuCM3027/ADuCM3029 Ultra Low Power ARM Cortex-M3 MCU with Integrated Power Management Data Sheet*.

Frame Sync

Frame sync is a control signal, used to determine the start of new word or frame. Upon detecting this signal, the serial port starts shifting in or out the data bits serially based on the direction selected. The frame sync signal can be internally generated from its serial clock (SPT_CLK) or externally provided, based on the `SPORT_CTL_x.IFS` bit.

If SPORT is configured for internal frame sync mode (`SPORT_CTL_x.IFS` = 1), the `SPORT_DIV_x.FSDIV` field specifies the divider to generate SPT_FS signal from the serial clock. This divisor is a 8-bit value, allowing a wide range of frame sync rates to initiate periodic transfers.

Number of serial clocks between frame syncs = (`SPORT_DIV_x.FSDIV` + 1).

`SPORT_DIV_x.FSDIV` is used for the internally generated SPT_CNV signal when it is operating in timer enable mode. Therefore, for timer enable mode, this field refers to the number of serial clock cycles between SPT_CNV pulses.

The value of `SPORT_DIV_x.FSDIV` must not be less than the value of the `SPORT_CTL_x.SLEN` bit field (the serial word length minus one, as this may cause an external device to abort the current operation or cause other unpredictable results.

NOTE: After enabling the SPORT, the first internal frame sync (or SPT_CNV in case of Timer enable mode) appears after a delay of (`SPORT_DIV_x.FSDIV` + 1) serial clocks.

If a SPORT is configured in external frame sync mode (`SPORT_CTL_x.IFS` = 0), then SPT_FS is an input signal and the `SPORT_DIV_x.FSDIV` field of the `SPORT_DIV_x` register is ignored. By default, this external signal is level sensitive. The frame sync is expected to be synchronous with the serial clock. If not, it must meet the timing requirements mentioned in the *ADuCM3027/ADuCM3029 Ultra Low Power ARM Cortex-M3 MCU with Integrated Power Management Data Sheet*.

NOTE: When the SPORT is enabled, frame sync must be deasserted.

Frame Sync Options

The framing signals may be active high or low. The SPORT_CTL_x.LFS bit selects the logic level of the frame sync signals.

- When SPORT_CTL_x.LFS = 0, the corresponding frame sync signal is active high. Default polarity of frame sync signal.
- When SPORT_CTL_x.LFS = 1, the corresponding frame sync signal is active low.

The following sections describe how frame sync signal is used by the serial port in an operating mode.

Data Dependent Frame Sync and Data Independent Frame Sync

When a SPORT is configured as a transmitter, that is, SPORT_CTL_x.SPTRAN bit is 1, and if data independent frame sync select, that is SPORT_CTL_x.DIFS bit is 0, then an internally generated transmit frame sync is only driven when a new data word has been loaded into the transmit buffer of the SPORT. In other words, frame sync signal generation and therefore data transmission is data dependent. This mode of operation allows for wait states when data is not ready.

When SPORT is configured as receiver (SPORT_CTL_x.SPTRAN = 0) and if SPORT_CTL_x.DIFS= 0, then a receive frame sync signal is generated only when receive data buffer status is not full.

The data independent frame sync mode allows the continuous generation of the framing signal, regardless of the buffer status. Setting SPORT_CTL_x.DIFS activates this mode. When SPORT_CTL_x.DIFS = 1, a transmit/receive frame sync signal is generated regardless of the transmit/receive data buffer status respectively.

NOTE: The DMA typically keeps the transmit buffer full. The application is responsible for filling the transmit buffers with data. Else, an underflow/overflow case would arise and it would be flagged.

Early Frame Sync and Late Frame Sync

The frame sync signals can be early or late. The SPORT_CTL_x.LAFS bit of the serial port control register configures this option.

By default, when SPORT_CTL_x.LAFS is cleared, the frame sync signal is configured as early framing signal. This is the normal mode of operation. In this mode, the first bit of the transmit data word is available (and the first bit of the receive data word is latched) in the next serial clock cycle after the frame sync is asserted. The frame sync is not checked again until the entire word has been transmitted (or received).

If data transmission is continuous in early framing mode (in other words, the last bit of each word is immediately followed by the first bit of the next word), then frame sync signal occurs during the last bit of each word. Internally generated frame syncs are asserted for one clock cycle in early framing mode.

When SPORT_CTL_x.LAFS is set, late frame syncs are configured; this is the alternate mode of operation. In this mode, the first bit of the transmit data word is available (and the first bit of the receive data word is latched) in the same serial clock cycle that the frame sync is asserted. Receive data bits are latched by serial clock edges. Internally-generated frame syncs remain asserted for the entire length of the data word in late framing mode.

NOTE: In the case of late frame sync, externally generated frame syncs are expected to be asserted for the entire word length of the data transfer. Else, unpredictable results can occur.

The figure shows the two modes of frame signal timing.

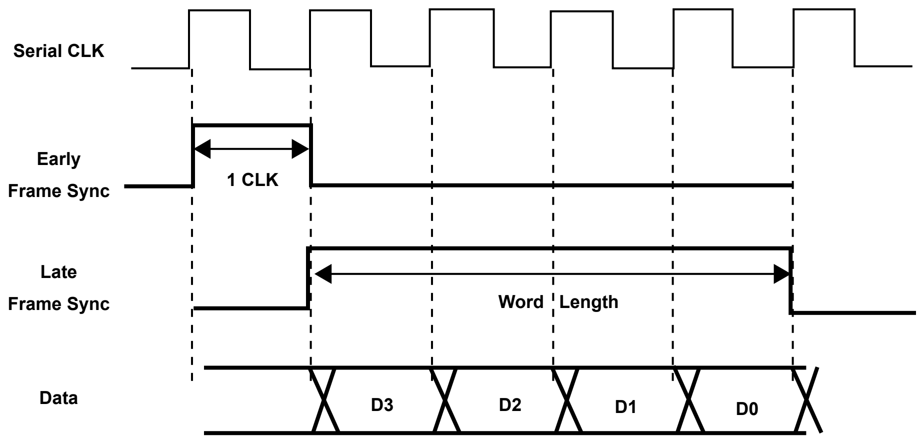


Figure 16-7: Normal Framing (Early Frame Sync) and Alternate Framing (Late Frame Sync)

Framed Sync and Unframed Frame Sync

The use of frame sync signal is optional in serial port communications. The `SPORT_CTL_x.FSR` bit determines if framing signal is required.

When `SPORT_CTL_x.FSR` bit is set, a frame sync signal is required for every data word.

When `SPORT_CTL_x.FSR` is cleared, the corresponding frame sync signal is not required. A single frame sync is required to initiate communications but it is ignored after the first bit is transferred. Data words are then transferred continuously in what is referred to as an unframed mode. Unframed mode is appropriate for continuous reception/transmission.

The following figure shows the framed versus unframed mode of serial port operation.

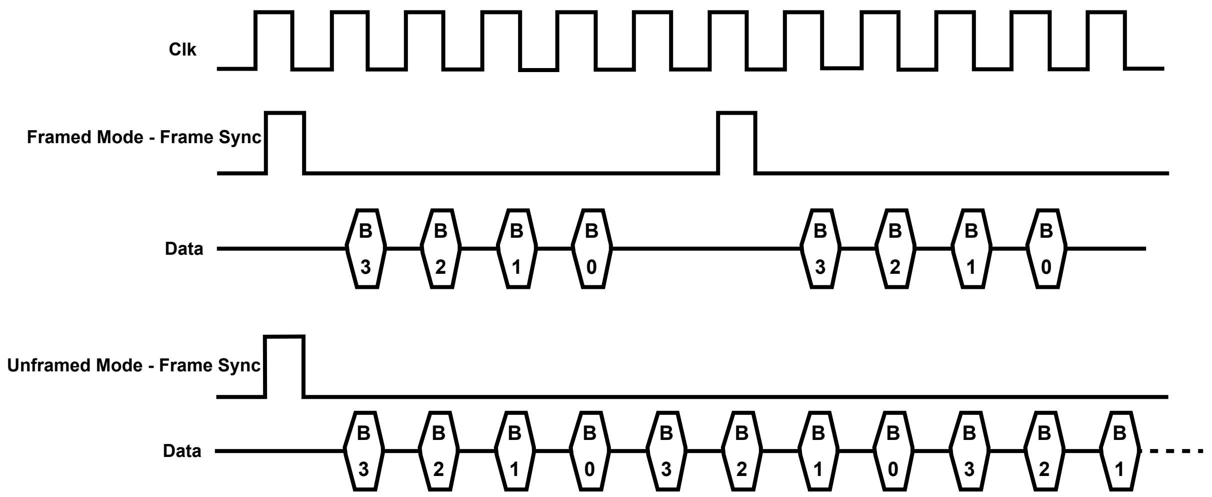


Figure 16-8: Framed Data Stream and Unframed Data Stream

NOTE: In unframed mode of operation, the `SPORT_CTL_x.DIFS` bit must always be set to 1. `SPORT_CTL_x.DIFS`=0 is not supported for unframed mode. Also, the `SPORT_CTL_x.LAFS` bit must always be set as zero. Late frame sync is not supported for unframed mode.

Sampling Edge

The serial port uses two control signals to sample or drive the serial data:

- Serial clock (SPT_CLK) applies the bit clock for each serial data.
- Frame sync (SPT_FS) divides the data stream into frames.

These control signals can be internally generated or externally provided, determined by the `SPORT_CTL_x.ICLK` and `SPORT_CTL_x.IFS` bit settings.

Data and frame syncs can be sampled on the rising or falling edges of the serial port clock signals. The `SPORT_CTL_x.CKRE` bit controls the sampling edge. By default, when `SPORT_CTL_x.CKRE` = 0, the MCU selects the falling edge of SPT_CLK signal for sampling receive data and external frame sync. The receive data and frame sync are sampled on the rising edge of SPT_CLK when `SPORT_CTL_x.CKRE` = 1.

Transmit data and internal frame sync signals are driven (change their state) on the serial clock edge that is not selected. By default, (`SPORT_CTL_x.CKRE` = 0) the SPORTs drive data and frame sync signals on the rising edge of the SPT_CLK signal and drives on falling edge when `SPORT_CTL_x.CKRE`= 1.

Therefore, transmit and receive functions of any two serial ports connected together should always select the same value for `SPORT_CTL_x.CKRE` so that internally generated signals are driven on one edge, and received signals are sampled on the opposite edge.

The following figure shows the typical SPORT signals on two sides of serial communication for `SPORT_CTL_x.CKRE` = 0.

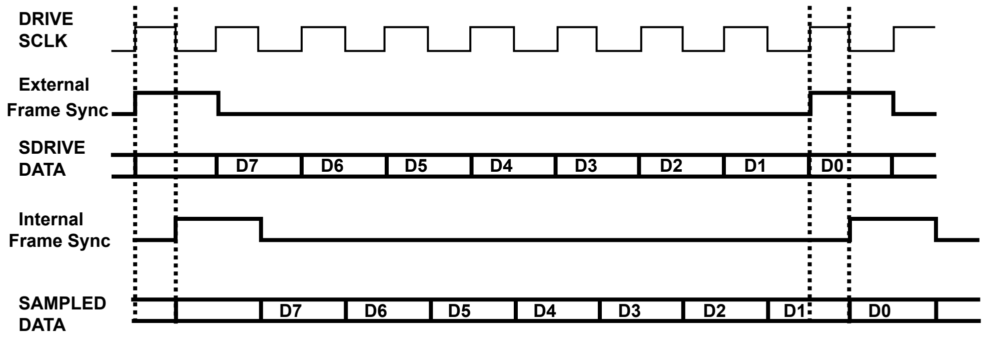


Figure 16-9: Frame Sync and Data Driven on Rising Edge

When the slave samples the Frame Sync signal, the word counter is reloaded to the maximum setting. Each SPT_CLK decrements this word counter until the full frame is received.

Therefore, if the transmitter drives the internal frame sync and data on the rising edge of serial clock, the falling edge should be used by receiver to sample the external frame sync and data, and vice versa.

Premature Frame Sync Error Detection

A SPORT framing signal is used to synchronize transmit or receive data. In external FS mode, if a frame sync received when an active frame is in progress or the frame sync truncates during an on-going transfer when using late frame sync, it is called premature frame sync and is invalid.

If premature frame sync is received, the `SPORT_STAT_x.FSERR` bit is flagged to indicate this framing error. An optional error interrupt can be generated for this event by setting the `SPORT_IEN_x.FSERRMSK` bits.

As shown in the following figure, the frame sync error (which sets the error bit) is triggered when an early frame sync occurs during data transfer (transmission or reception) or for late frame sync if the period of the frame sync is smaller than the serial word length, `SPORT_CTL_x.SLEN`.

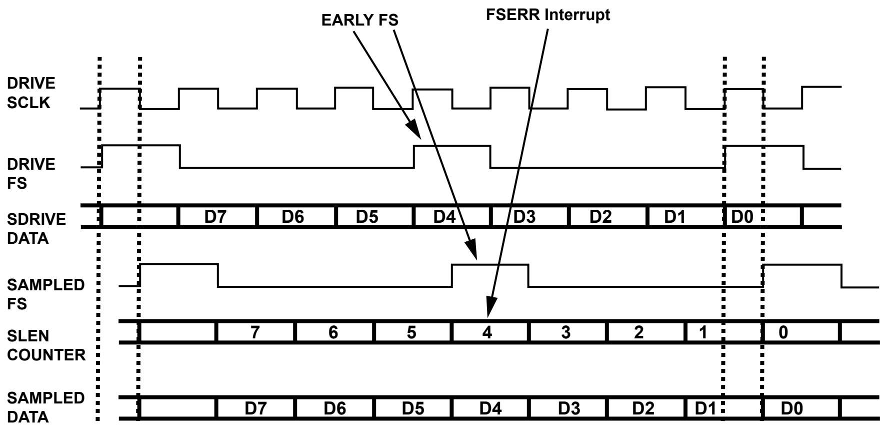


Figure 16-10: Frame Sync Error Detection

When a frame sync error occurs, the `SPORT_CTL_x_FSERMODE` bit decides the way this error is handled only in case of receive operation. If this bit is set, the receive data is discarded. When the next frame sync comes, fresh receive transfer is carried out.

If this bit is zero, then the frame sync error is flagged and the transfer is continued. No action is taken with respect to the transfer.

Serial Word Length

The SPORT_CTL_x.SLEN field of serial port control register determines the word length of serial data to transmit and receive. Each half SPORT can independently handle word lengths up to 32 bits. Words smaller than 32 bits are right-justified during transmit or receive buffers to least significant bit (LSB) position. However, data can be shifted in or out in MSB first or LSB first format based on SPORT_CTL_x.LSBF bit setting.

The value of the SPORT_CTL_x.SLEN field can be calculated as:

SLEN = Serial port word length - 1

The range of valid word length in standard serial mode is 4-32.

Number of Transfers

The SPORT_NUMTRAN_x register determines the number of word transfers. The word length is specified in the SPORT_CTL_x.SLEN field. Each half SPORT has this field and can be used for receive or transmit depending on the SPORT_CTL_x.SPTRAN bit. This field can be programmed to the required number of word transfers to be transmitted/received. Once the programmed number of transfers are done, the SPORT disables all the operations. That is, clock is not generated if in internal clock mode and not latched if in external clock mode. The frame sync is not generated and no operation is performed for received frame syncs.

For example, if 20 transfers are programmed and word length programmed is 20 bits, after 400 bits of transfer the SPORT would disable all the operations.

This register has 12 programmable bits to decide the number of transfers.

To start the next set of transfers, the SPORT_NUMTRAN_CNT_x register must be started with the desired number of transfers. If the same number of transfers need to be programmed, the same value must be reprogrammed.

Also, once the number of transfers are finished, in external clock mode, no extra frame syncs are expected to be sent to the SPORT. For new transfers, the frame sync must be sent only once the SPORT_NUMTRAN_x is reprogrammed.

NOTE: In unframed mode, the number of transfers cannot be reprogrammed to perform the next set of transfers. SPORT must be disabled and re-enabled to perform the operation for the next set of transfers in case of unframed mode.

SPORT Power Management

When the chip goes into hibernate mode, the configuration values are retained. The following registers are retained:

- SPORT_CTL_A (except SPORT_CTL_A.DMAEN and SPORT_CTL_A.SPEN bit fields)
- SPORT_DIV_A
- SPORT_CNVT_A

- `SPORT_CTL_B` (except `SPORT_CTL_B.DMAEN` and `SPORT_CTL_B.SPEN` bit fields)
- `SPORT_DIV_B`
- `SPORT_CNVT_B`

NOTE: If any transfer is going on before the chip goes into hibernate, that transfer does not resume after returning from hibernate. A fresh start must be made when SPORT comes back from hibernate for the new transfer. Program the `SPORT_NUMTRAN_x` register appropriately and then program the `SPORT_CTL_x.SPEN` bit to enable the SPORT operation.

SPORT Operating Modes

The ADuCM302x MCU has SPORT that supports the following operating modes:

- Standard serial mode
- Timer enable mode

The half SPORTs within a SPORT can be independently configured if they are not coupled using SPMUX logic.

NOTE:

- These modes support either, cases where the clock and the frame sync are both internal or both are external. Modes with external frame sync, internal clock and internal frame sync, external clock is not supported by the SPORT. Only for the unframed mode, external clock (`SPORT_CTL_x.ICLK` = 0) and internal frame sync pulse (`SPORT_CTL_x.IFS` = 1), generated for the start of transfer is supported.
- To change the SPORT configuration (for example, changing `SPORT_CTL_x.SLEN` or `SPORT_CTL_x.LSBF`), clear the `SPORT_CTL_x.SPEN` bit and re-enable again by setting this bit.
- When SPORT is operating in external clock mode, SPORT needs three serial clocks after the SPORT is enabled before the normal operation can begin.

Mode Selection

The serial port operating mode is configured in the `SPORT_CTL_x.OPMODE`.

Standard Serial Mode

The SPORT can be configured in standard DSP serial mode by clearing the `SPORT_CTL_x.OPMODE`. The standard serial mode lets programs configure serial ports to connect to a variety of serial devices such as serial data converters and audio codecs. In order to connect to these devices, a variety of clocking, framing, and data formatting options are available. All of these options can be used in the standard serial mode.

Timer Enable Mode

Few ADCs/DACs require two control signals for their conversion process. To interface with such devices, an extra signal (SPT_CNV) is required. The timer enable mode must be enabled to use this signal by programming the

`SPORT_CTL_x.OPMODE` to 1. A PWM timer inside the module is used to generate the programmable SPT_CNV signal.

The width of the SPT_CNV signal is programmed in the `SPORT_CNVT_x.WID` field. The delay between SPT_CNV assertion and frame sync assertion is programmed in the `SPORT_CNVT_x.CNVT2FS`. This programmability provides flexibility in the use of these signals. The following figure shows these values.

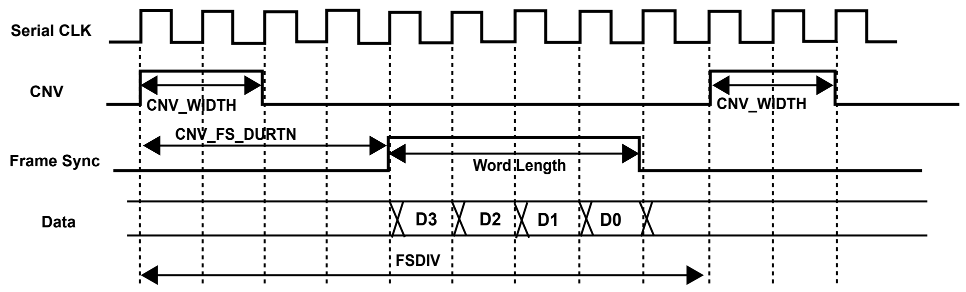


Figure 16-11: Signals Interfaced in Timer Enable Mode

The polarity of the SPT_CNV signal can be programmed in the `SPORT_CNVT_x.POL` bit field.

The number of serial clock cycles between SPT_CNV pulse is given by the `SPORT_DIV_x.FSDIV` value. All the other programmable options defined for frame sync in the DSP serial mode can be used in this mode except the following options:

- External frame sync and external clock cannot be used.
- Early frame sync cannot be used.
- Unframed mode cannot be used.

NOTE: SPT_CNV signal is not used when DSP serial mode is in use.

SPORT Data Transfer

The SPORT uses either a core driven or DMA driven data transfers. DMA transfers can be set up to transfer a configurable number of serial words between the serial port transmit or receive data buffers and internal memory automatically. Core-driven transfers use SPORT interrupts to signal the MCU core to perform single word transfers to/ from the serial port data buffers.

Single Word (Core) Transfers

Individual data words may also be transmitted or received by the serial ports, with interrupts occurring as each data word is transmitted or received. When a serial port is enabled and corresponding DMA is disabled, the SPORT interrupts are generated whenever a complete word has been received in the receive data buffer, or whenever the transmit data buffer is not full.

When serial port data packing is enabled, transmit and receive interrupts are generated for 32-bit packed words, not for each 16-bit or 8-bit word. When performing core driven transfers, write to the buffer designated by the `SPORT_CTL_x.SPTRAN` bit setting. To avoid undetermined behavior, check the status of appropriate data buffers when the MCU core tries to read a word from a serial port's receive buffer or writes a word to its transmit buffer. The full/empty status can be read using the `SPORT_STAT_x.DXS` bit.

DMA Transfers

DMA provides a mechanism for receiving or transmitting an entire block of serial data before an interrupt is generated. When SPORT DMA is not enabled, the SPORT generates an interrupt every time it receives or starts to transmit a data word. The MCU's on-chip DMA controller handles the DMA transfer, allowing the MCU core to either go to sleep or continue running other tasks until the entire block of data is transmitted or received. Service routines can then operate on the block of data rather than on single words, significantly reducing overhead.

Therefore, set the direction bit, DMA enable bit, and serial port enable bit before initiating any operations on the SPORT data buffers. Do not access data buffers when the associated DMA channel of serial port is enabled. Each half SPORT has a dedicated DMA channel for transmit and receive data paths. In transmit mode, the DMA controller writes to the transmit data buffer. Similarly, in receive mode, the DMA controller reads from the receive data buffer. The DMA controller generates an interrupt at the end of the completion of a DMA transfer.

Though DMA transfers are performed with 32-bit words, the SPORTs can handle word sizes from 4 to 32 bits (as defined by `SPORT_CTL_x.SLEN` field). If serial data length is 16 bits or smaller, two data bytes can be packed into 32-bit words for each DMA transfer. If serial data length is 8 bits or smaller, four data can be packed into 32-bit words for each DMA transfer. When serial port data packing is enabled, transmit and receive DMA requests are generated for the 32-bit packed words, not for each 16-bit or 8-bit word.

Depending on whether packing is enabled, appropriate DMA transfer width needs to be selected in the DMA controller. If packing is enabled or greater than 16-bit transfers are desired, use word transfers in DMA. If packing is not enabled and transfer is less than 16 bits but greater than 8 bits, use half word transfers. If packing is not enabled and transfer length is less than 8 bits, use byte transfers within the DMA controller.

NOTE: DMA requests may not be serviced frequently to guarantee continuous data flow in case of continuous transmission/reception. Ensure that the DMA channel has the highest priority when performing such transfers.

SPORT Data Buffers

Data Buffer Status

The SPORT provides status information about data buffers. Depending on the `SPORT_CTL_x.SPTRAN` bit setting, these bits reflect the status of transmit data buffers or receive data buffers. The `SPORT_STAT_x.DXS` bits indicate if the buffers are full, partially full, or empty.

To avoid undetermined conditions, always check the buffer status to determine if the access can be made. The `SPORT_STAT_x.DXS` field always reflect the FIFO status.

Three complete 32-bit words can be stored in the receive buffer while the fourth word is being shifted in. Therefore, four complete words can be received without the receive buffer being read, before an overflow occurs. After receiving the fourth word completely, the shift register contents overwrite the third word if the first word has not been read out (by the MCU core or DMA controller). When this happens, the receive overflow status is flagged through the error status bits of the `SPORT_STAT_x.DERR` register. The overflow status is generated after the last bit of the fourth word is received.

Data Buffer Packing

When the SPORT is configured as a receiver with a serial data word length of 16 or less or even 8 or less, then received data words may be packed into a 32-bit word. Similarly, if the SPORT is configured as transmitter with a serial data word length of 16 or less, then 32-bit words being transmitted may be unpacked into two 16-bit words or four 8 bit words. This feature is selected by the `SPORT_CTL_x.PACK` bit.

NOTE: If packing is not enabled for lower length transfers, bus bandwidth as well as FIFO buffers are not efficiently used.

When `SPORT_CTL_x.PACK`= 01, four successive words received are packed into a single 32-bit word, or each 32-bit word is unpacked and transmitted as four 8-bit words. The words with less than 16-bit are packed as [SLEN: 0], [(SLEN+8):8], [(SLEN+16):16] and [(SLEN+24):24], , where SLEN is `SPORT_CTL_x.SLEN`.

This applies to both receive (packing) and transmit (unpacking) operations. Following diagram shows the 8 bit packing. The dark region corresponds to the packed bits.

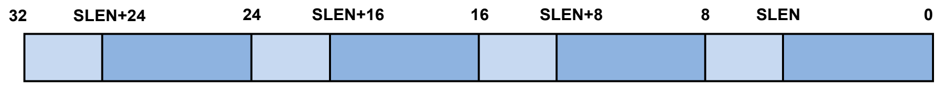


Figure 16-12: Packed Data (8-bit Packing)

When `SPORT_CTL_x.PACK` is assigned 10, two successive words received are packed into a single 32-bit word, or each 32-bit word is unpacked and transmitted as two 16-bit words. The words with less than 16-bit are packed as [SLEN: 0] and [(SLEN+16):16]. This applies to both receive (packing) and transmit (unpacking) operations. Following figure shows the 16 bit packing.

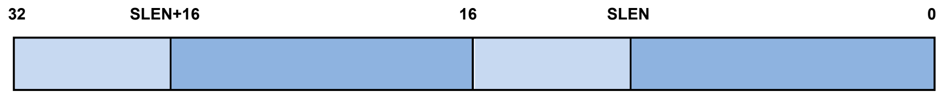


Figure 16-13: Packed Data (16-bit Packing)

In packing, transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word or 8 bit word. The packing feature enables efficient utilization of the system bandwidth.

NOTE: When packing is enabled, the word length must be programmed lesser than the programmed packing. For instance, if 8-bit packing is programmed, `SPORT_CTL_x.SLEN` must be less than or equal to 7. If 16-bit packing is programmed, `SPORT_CTL_x.SLEN` must be less than or equal to 15.

SPORT Interrupts and Exceptions

Each half SPORT has an interrupt associated with it. To determine the source of an interrupt, applications should check the interrupt status register. This section describes the various interrupt sources. In core mode, SPORT is able to generate data interrupts for receive or transmit operations. Moreover, the SPORT modules generate error conditions which have a separate status bit for each interrupt source.

NOTE: To clear the interrupt signal, the corresponding status bit is written by 1. This clears the value of the status bit, that is, the corresponding status bit becomes zero.

Error Detection

When the SPORT is configured as a transmitter, the `SPORT_STAT_x.DERR` bit provides transmit data buffer underflow status for the data path (indicates that frame sync signal occurred when the transmit data buffer was empty). The SPORT transmits data whenever it detects a framing signal.

Similarly, if the SPORT configured as a receiver, the `SPORT_STAT_x.DERR` bit provides receive overflow status of receive data buffer. In other words, the SPORT indicates that a channel has received new data when the receive buffer is full, so new data overwrites existing data. The SPORT receives data when it detects a framing signal.

These error conditions occur in case of external frame sync or when the DIFS (data independent frame sync) is set.

The `SPORT_IEN_x.DERRMSK` (data error interrupt enable) bit can be used to unmask the status interrupt for data errors.

In addition to the data underflow and data overflow errors, the status interrupt is also triggered optionally when frame sync error is detected. The `SPORT_IEN_x.FSERRMSK` (frame sync error interrupt enable) bit unmasks the status interrupt for this frame sync error. This frame sync error is generated due to premature frame sync. For more information about this, refer to Premature Frame Sync Error Detection.

NOTE: A frame sync error is not detected when there is no active data transmit/receive and the frame sync pulse occurs due to noise in the input signal.

System Transfer Interrupts

When the SPORT is configured to transmit, an interrupt is generated when an attempt is made by the core to write into a transmit FIFO which is full. The `SPORT_STAT_x.SYSDATERR` bit indicates this transmit FIFO overflow error.

Similarly, when the SPORT is configured to receive, an interrupt is generated when an attempt is made by the core to read from an empty receive FIFO. The `SPORT_STAT_x.SYSDATERR` bit now indicates receive FIFO underflow error.

NOTE: In DMA mode, SPORT does not produce a DMA request when transmit FIFO is full or receive FIFO is empty. Interrupt is not generated for DMA transfers.

Transfer Finish Interrupt (TFI)

The Transmit Finish Interrupt (TFI) feature is used to signal the end of the transmission/reception. This feature can be enabled by setting the `SPORT_IEN_x` bit. When the number of transfers programmed in the `SPORT_NUMTRAN_x` field is finished, SPORT waits until all the data in the FIFO is transmitted out (including the transmit shift register) or received completely in the receive registers and then generates the TFI. This feature allows the user to determine that all data corresponding to understand that data corresponding to the programmed number of transfers have been transmitted out or received completely by the SPORT.

NOTE: Once the TFI is obtained, the user can either reprogram the number of transfers or disable the SPORT and re-enabled (if needed). To reprogram the number of transfers, refer to Number of Transfers.

To re-enable SPORT, refer to SPORT Programming Model.

SPORT Programming Model

The following are the SPORT programming guidelines:

- Write all the registers of SPORT with the desired values of configuration except the control register.
- After configuration, write the control register (`SPORT_CTL_x`) with desired configuration for the transfer. The `SPORT_CTL_x.SPEN` field must be programmed as 1.
- When the `SPORT_CTL_x.SPEN` field is written to 1, normal operation of the SPORT can start based on the programmed configuration values.

The SPORT registers that need to be written with configuration values are:

- Write `SPORT_DIV_x` to program the desired CLKDIV and FSDIV values.
- The `SPORT_CNVT_x` register must only be written in timer enable mode.
- EN register is written based on the interrupts. For example, if an interrupt is desired for an underflow/overflow error, `SPORT_IEN_x` must be set.
- Write `SPORT_NUMTRAN_x` with the desired number of transfers. This must not be programmed to zero.

ADuCM302x SPORT Register Descriptions

Serial Port (SPORT) contains the following registers.

Table 16-2: ADuCM302x SPORT Register List

Name	Description
SPORT_CTL_A	Half SPORT 'A' Control
SPORT_CTL_B	Half SPORT 'B' Control
SPORT_DIV_A	Half SPORT 'A' Divisor
SPORT_DIV_B	Half SPORT 'B' Divisor
SPORT_IEN_A	Half SPORT A's Interrupt Enable
SPORT_IEN_B	Half SPORT B's Interrupt Enable
SPORT_NUMTRAN_A	Half SPORT A Number of Transfers
SPORT_NUMTRAN_B	Half SPORT B Number of Transfers

Table 16-2: ADuCM302x SPORT Register List (Continued)

Name	Description
SPORT_RX_A	Half SPORT 'A' Rx Buffer
SPORT_RX_B	Half SPORT 'B' Rx Buffer
SPORT_STAT_A	Half SPORT A's Status
SPORT_STAT_B	Half SPORT B's Status
SPORT_CNVT_A	Half SPORT 'A' CNV Width
SPORT_CNVT_B	Half SPORT 'B' CNV Width
SPORT_TX_A	Half SPORT 'A' Tx Buffer
SPORT_TX_B	Half SPORT 'B' Tx Buffer

Half SPORT 'A' Control

This register contains transmit and receive control bits for SPORT half 'A', including serial port mode selection.

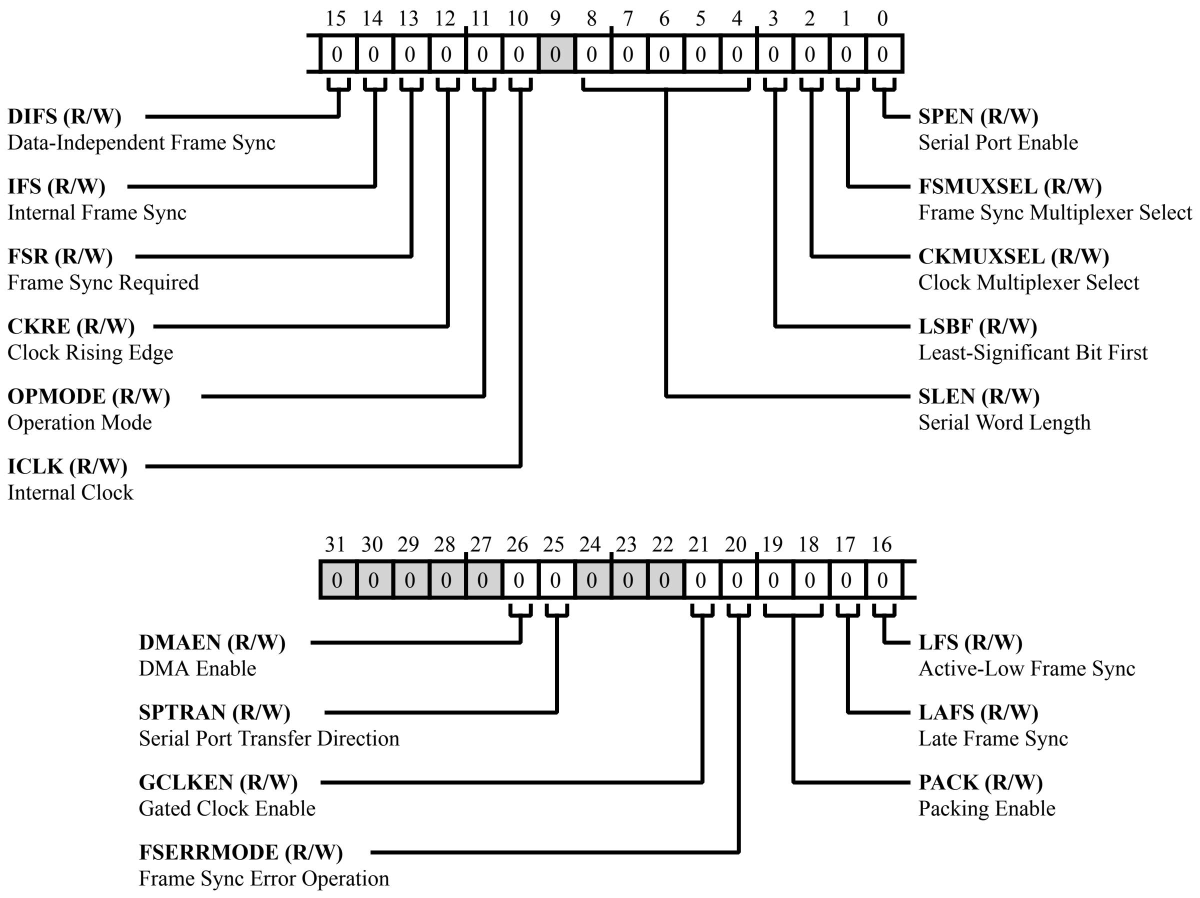


Figure 16-14: SPORT_CTL_A Register Diagram

Table 16-3: SPORT_CTL_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/W)	DMAEN	DMA Enable. This bit tells whether the half SPORT would send DMA request signals when the Transmit FIFO needs any data or Receive FIFO wants to send any data to DMA.	
		0	DMA requests are diabled
		1	DMA requests are enabled

Table 16-3: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
25 (R/W)	SPTRAN	Serial Port Transfer Direction. This bit selects the transfer direction (receive or transmit) for the half SPORT's channels.	
		0	Receive
		1	Transmit
21 (R/W)	GCLKEN	Gated Clock Enable. The `SPORT_CTL_A.GCLKEN` bit enables gated clock operation for the half SPORT. When `SPORT_CTL_A.GCLKEN` is enabled, the SPORT clock is active when the SPORT is transferring data or when the frame sync changes (transitions to active state).	
		0	Disable
		1	Enable
20 (R/W)	FSERRMODE	Frame Sync Error Operation. This bit decides the way the SPORT responds when a frame sync error occurs.	
		0	Flag the Frame Sync error and continue normal operation
		1	When frame Sync error occurs, discard the receive data
19:18 (R/W)	PACK	Packing Enable. This bit enables the half SPORT to perform 16- to 32-bit or 8- to 32-bit packing on received data and to perform 32- to 16-bit or 32- to 8-bit unpacking on transmitted data.	
		0	Disable
		1	8-bit packing enable
		2	16-bit packing enable
		3	Reserved
17 (R/W)	LAFS	Late Frame Sync. The `SPORT_CTL_A.LAFS` bit selects whether the half SPORT generates a late frame sync (`SPORT_AFS` during first data bit) or generates an early frame sync signal (`SPORT_AFS` during serial clock cycle before first data bit).	
		0	Early frame sync
		1	Late frame sync

Table 16-3: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	LFS	Active-Low Frame Sync. This bit selects whether the half SPORT uses active low or active high frame sync.	
		0	Active high frame sync
		1	Active low frame sync
15 (R/W)	DIFS	Data-Independent Frame Sync. This bit selects whether the half SPORT uses a data-independent or data-dependent frame sync. When using a data-independent frame sync, the half SPORT generates the sync at the interval selected by `SPORT_DIV_A.FSDIV`. When using a data-dependent frame sync, the half SPORT generates the sync on the selected interval when the transmit buffer is not empty or when the receive buffer is not full.	
		0	Data-dependent frame sync
		1	Data-independent frame sync
14 (R/W)	IFS	Internal Frame Sync. This bit selects whether the half SPORT uses an internal frame sync or uses an external frame sync. Note that the externally-generated frame sync does not need to be synchronous with the processor's system clock.	
		0	External frame sync
		1	Internal frame sync
13 (R/W)	FSR	Frame Sync Required. This bit selects whether or not the half SPORT requires frame sync for data transfer.	
		0	No frame sync required
		1	Frame sync required
12 (R/W)	CKRE	Clock Rising Edge. This bit selects the rising or falling edge of the `SPORT_ACLK` clock for the half SPORT to sample receive data and frame sync.	
		0	Clock falling edge
		1	Clock rising edge
11 (R/W)	OPMODE	Operation Mode. This bit selects whether the half SPORT operates in DSP standard mode or in Timer enable mode.	
		0	DSP standard
		1	Timer_enable mode

Table 16-3: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W)	ICLK	Internal Clock. This bit selects whether the half SPORT uses an internal or external clock.	
		0	External clock
		1	Internal clock
8:4 (R/W)	SLEN	Serial Word Length. This bits selects word length in bits for the half SPORT's data transfers. Word may be from 4- to 32-bits in length. The formula for selecting the word length in bits is: SPORT_CTL_A.SLEN = (serial word length in bits) - 1.	
3 (R/W)	LSBF	Least-Significant Bit First. This bit selects whether the half SPORT transmits or receives data LSB first or MSB first.	
		0	MSB first sent/received
		1	LSB first sent/received
2 (R/W)	CKMUXSEL	Clock Multiplexer Select. This bit enables multiplexing of the half SPORT' serial clock. In this mode, the serial clock of the related half SPORT is used instead of the half SPORT's own serial clock. For example, if SPORT_CTL_A.CKMUXSEL is enabled, half SPORT 'A' uses SPORT_BCLK instead of SPORT_ACLK.	
		0	Disable serial clock multiplexing
		1	Enable serial clock multiplexing
1 (R/W)	FSMUXSEL	Frame Sync Multiplexer Select. The SPORT_CTL_A.FSMUXSEL bit enables multiplexing of the half SPORT' frame sync. In this mode, the frame sync of the related half SPORT is used instead of the half SPORT's own frame sync. For example, if SPORT_CTL_A.FSMUXSEL is enabled, half SPORT 'A' uses SPORT_BFS instead of SPORT_AFS.	
		0	Disable frame sync multiplexing
		1	Enable frame sync multiplexing
0 (R/W)	SPEN	Serial Port Enable. The SPORT_CTL_A.SPEN bit enables the half SPORT's data channel. Note: When this bit is cleared (changes from =1 to =0), the half SPORT automatically flushes the channel's data buffers and disables the clock and frame sync and the counters inside SPORT.	
		0	Disable
		1	Enable

Half SPORT 'B' Control

This register contains transmit and receive control bits for SPORT half 'B', including serial port mode selection for the channels. The function of some bits in this register vary, depending on the SPORT's operating mode. For more information, see the SPORT operating modes description. If reading reserved bits, the read value is the last written value to these bits or is the reset value of these bits.

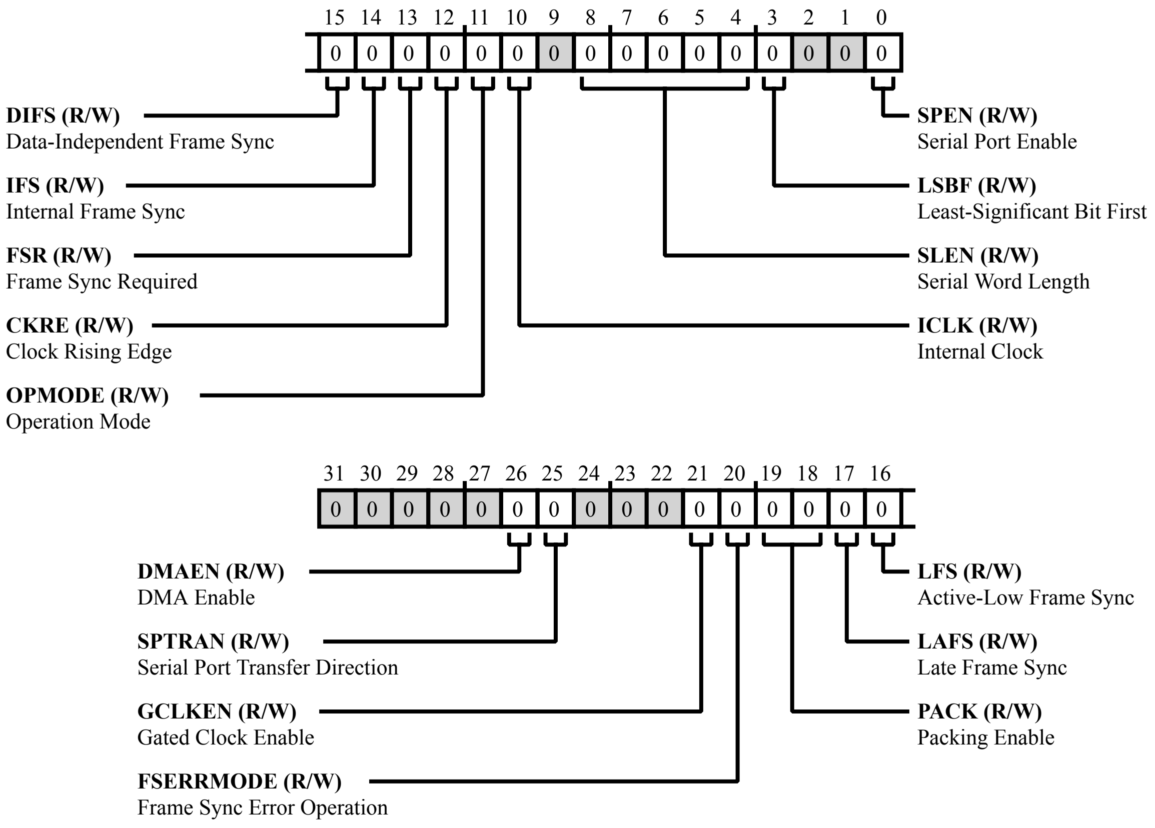


Figure 16-15: SPORT_CTL_B Register Diagram

Table 16-4: SPORT_CTL_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/W)	DMAEN	DMA Enable. This bit tells whether the half SPORT would send DMA request signals when the Transmit FIFO needs any data or Receive FIFO wants to send any data to DMA	
		0	DMA requests are diabled
		1	DMA requests are enabled

Table 16-4: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
25 (R/W)	SPTRAN	Serial Port Transfer Direction. The `SPORT_CTL_B.SPTRAN` bit selects the transfer direction (receive or transmit) for the half SPORT's channels.	
		0	Receive
		1	Transmit
21 (R/W)	GCLKEN	Gated Clock Enable. The `SPORT_CTL_B.GCLKEN` bit enables gated clock operation for the half SPORT when in DSP serial mode. When `SPORT_CTL_B.GCLKEN` is enabled, the SPORT clock is active when the SPORT is transferring data or when the frame sync changes (transitions to active state).	
		0	Disable
		1	Enable
20 (R/W)	FSERRMODE	Frame Sync Error Operation. The `SPORT_CTL_B.FSERRMODE` bit decides the way the SPORT responds when a frame sync error occurs.	
		0	Flag the Frame Sync error and continue normal operation
		1	When frame Sync error occurs, discard the receive data
19:18 (R/W)	PACK	Packing Enable. The `SPORT_CTL_B.PACK` bit enables the half SPORT to perform 16- to 32-bit or 8- to 32- bit packing on received data and to perform 32- to 16-bit or 32- to 8- bit unpacking on transmitted data.	
		0	Disable
		1	8-bit packing enable
		2	16-bit packing enable
		3	Reserved
17 (R/W)	LAFS	Late Frame Sync. When the half SPORT is in DSP standard mode (`SPORT_CTL_B.OPMODE` =0) , the `SPORT_CTL_B.LAFS` bit selects whether the half SPORT generates a late frame sync (`SPORT_BFS` during first data bit) or generates an early frame sync signal (`SPORT_BFS` during serial clock cycle before first data bit).	
		0	Early frame sync
		1	Late frame sync

Table 16-4: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	LFS	Active-Low Frame Sync. When the half SPORT is in DSP standard mode (`SPORT_CTL_B.OPMODE` =0), the `SPORT_CTL_B.LFS` bit selects whether the half SPORT uses active low or active high frame sync.	
		0	Active high frame sync
		1	Active low frame sync
15 (R/W)	DIFS	Data-Independent Frame Sync. The `SPORT_CTL_B.DIFS` bit selects whether the half SPORT uses a data-independent or data-dependent frame sync. When using a data-independent frame sync, the half SPORT generates the sync at the interval selected by `SPORT_DIV_B.FSDIV`. When using a data-dependent frame sync, the half SPORT generates the sync on the selected interval when the transmit buffer is not empty or when the receive buffer is not full.	
		0	Data-dependent frame sync
		1	Data-independent frame sync
14 (R/W)	IFS	Internal Frame Sync. The `SPORT_CTL_B.IFS` bit selects whether the half SPORT uses an internal frame sync or uses an external frame sync. Note that the externally-generated frame sync does not need to be synchronous with the processor's system clock.	
		0	External frame sync
		1	Internal frame sync
13 (R/W)	FSR	Frame Sync Required. The `SPORT_CTL_B.FSR` selects whether or not the half SPORT requires frame sync for data transfer.	
		0	No frame sync required
		1	Frame sync required
12 (R/W)	CKRE	Clock Rising Edge. The `SPORT_CTL_B.CKRE` selects the rising or falling edge of the `SPORT_BCLK` clock for the half SPORT to sample receive data and frame sync.	
		0	Clock falling edge
		1	Clock rising edge

Table 16-4: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	OPMODE	Operation Mode. The `SPORT_CTL_B.OPMODE` bit selects whether the half SPORT operates in DSP standard mode or timer enable mode	
		0	DSP standard mode
		1	Timer Enable mode
10 (R/W)	ICLK	Internal Clock. When the half SPORT is in DSP standard mode (`SPORT_CTL_B.OPMODE` =0), the `SPORT_CTL_B.ICLK` bit selects whether the half SPORT uses an internal or external clock. For internal clock enabled, the half SPORT generates the `SPORT_BCLK` clock signal, and the `SPORT_BCLK` is an output. The `SPORT_DIV_B.CLKDIV` serial clock divisor value determines the clock frequency. For internal clock disabled, the `SPORT_BCLK` clock signal is an input, and the serial clock divisor is ignored. Note that the externally-generated serial clock does not need to be synchronous with the processor's system clock.	
		0	External clock
		1	Internal clock
8:4 (R/W)	SLEN	Serial Word Length. The `SPORT_CTL_B.SLEN` bits selects word length in bits for the half SPORT's data transfers. Word may be from 4- to 32-bits in length. The formula for selecting the word length in bits is: `SPORT_CTL_B.SLEN` = (serial word length in bits) - 1	
3 (R/W)	LSBF	Least-Significant Bit First. The `SPORT_CTL_B.LSBF` bit selects whether the half SPORT transmits or receives data LSB first or MSB first.	
		0	MSB first sent/received
		1	LSB first sent/received
0 (R/W)	SPEN	Serial Port Enable. The `SPORT_CTL_B.SPEN` bit enables the half SPORT's data channel. Note: When this bit is cleared (changes from =1 to =0), the half SPORT automatically flushes the channel's data buffers and disables the clock and frame sync and the counters inside SPORT.	
		0	Disable
		1	Enable

Half SPORT 'A' Divisor

This register contains divisor values that determine frequencies of internally generated clocks and frame syncs for half SPORT 'A'.

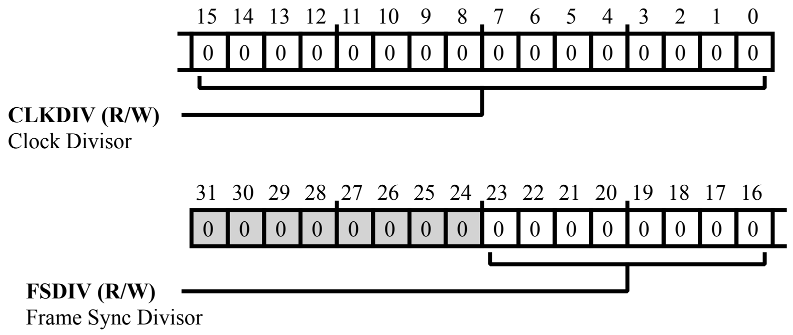


Figure 16-16: SPORT_DIV_A Register Diagram

Table 16-5: SPORT_DIV_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	FSDIV	Frame Sync Divisor. This bitfield selects the number of transmit or receive clock cycles that the half SPORT counts before generating a frame sync pulse. The half SPORT counts serial clock cycles whether these are from an internally- or an externally-generated serial clock. This field is used to measure the number of serial clock cycles before generating SPT_CNV signal in timer enable mode.
15:0 (R/W)	CLKDIV	Clock Divisor. This bitfield selects the divisor that the half SPORT uses to calculate the serial clock (`SPORT_ACLK`) from the processor system clock (PCLK).

Half SPORT 'B' Divisor

This register contains divisor values that determine frequencies of internally generated clocks and frame syncs for SPORT half 'B'.

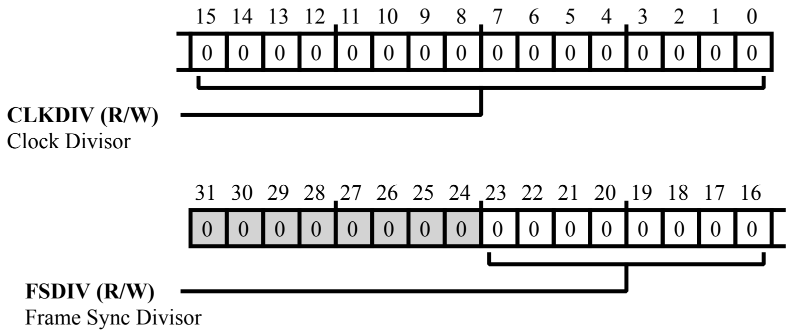


Figure 16-17: SPORT_DIV_B Register Diagram

Table 16-6: SPORT_DIV_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	FSDIV	Frame Sync Divisor. This bitfield selects the number of transmit or receive clock cycles that the half SPORT counts before generating a frame sync pulse. The half SPORT counts serial clock cycles whether these are from an internally- or an externally-generated serial clock. This field is used to measure the number of serial clock cycles before generating SPT_CNV signal in timer enable mode.
15:0 (R/W)	CLKDIV	Clock Divisor. This bitfield selects the divisor that the half SPORT uses to calculate the serial clock (`SPORT_BCLK`) from the processor system clock (PCLK).

Half SPORT A's Interrupt Enable

This register contains all the fields related to the Enable given for the various interrupts related to errors and data requests present in the half SPORT A.

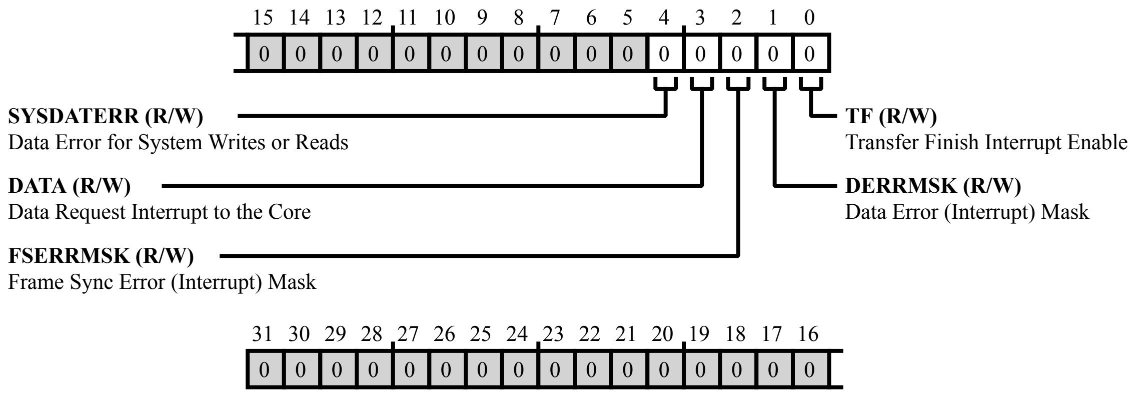


Figure 16-18: SPORT_IEN_A Register Diagram

Table 16-7: SPORT_IEN_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	SYSDATERR	Data Error for System Writes or Reads. This field enables the half SPORT to generate the data error interrupt for the system write resulting in TX FIFO overflow or system read resulting in a RX FIFO underflow.	
		0	Disable System data error interrupt
		1	Enable System data error interrupt
3 (R/W)	DATA	Data Request Interrupt to the Core. This bit enables interrupt given to the core for a data write into transmit FIFO for transmit or data read from the Receive FIFO.	
		0	Data request interrupt disable
		1	Data request interrupt enable
2 (R/W)	FSERRMSK	Frame Sync Error (Interrupt) Mask. This bit unmasks (enables) the half SPORT to generate the frame sync error interrupt.	
		0	Mask (disable)
		1	Unmask (enable)

Table 16-7: SPORT_IEN_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	DERRMSK	Data Error (Interrupt) Mask. This bit unmasks (enables) the half SPORT to generate the data error interrupt for the data channel.	
		0	Mask (disable)
		1	Unmask (enable)
0 (R/W)	TF	Transfer Finish Interrupt Enable. This bit selects when the half SPORT issues its transmission complete interrupt once the programmed number of transfers are finished. When enabled (`SPORT_IEN_A.TF` =1), when the last bit of last word of the programmed number of transfers is shifted out or received completely, an interrupt is generated. When disabled (`SPORT_IEN_A.TF` =0), no interrupt is generated by SPORT.	
		0	Transfer finish Interrupt is disabled
		1	Transfer Finish Interrupt is Enabled

Half SPORT B's Interrupt Enable

This register contains all the fields related to the Enable given for the various interrupts related to errors and data requests present in the half SPORT B.

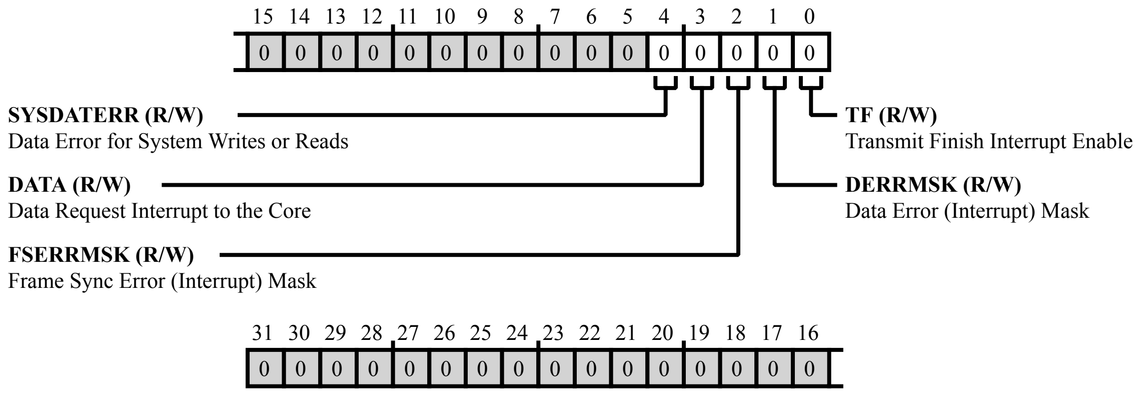


Figure 16-19: SPORT_IEN_B Register Diagram

Table 16-8: SPORT_IEN_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	SYSDATERR	Data Error for System Writes or Reads. This field enables the half SPORT to generate the data error interrupt for the system write resulting in TX FIFO overflow or system read resulting in a RX FIFO underflow.	
		0	Disable System data error interrupt
		1	Enable System data error interrupt
3 (R/W)	DATA	Data Request Interrupt to the Core. This bit enables interrupt given to the core for a data write into transmit FIFO for transmit or data read from the Receive FIFO.	
		0	Data request interrupt disable
		1	Data request interrupt enable
2 (R/W)	FSERRMSK	Frame Sync Error (Interrupt) Mask. This bit unmasks (enables) the half SPORT to generate the frame sync error interrupt.	
		0	Mask (disable)
		1	Unmask (enable)

Table 16-8: SPORT_IEN_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	DERRMSK	Data Error (Interrupt) Mask. This bit unmasks (enables) the half SPORT to generate the data error interrupt for the data channel.	
		0	Mask (disable)
		1	Unmask (enable)
0 (R/W)	TF	Transmit Finish Interrupt Enable. This bit selects when the half SPORT issues its transmission complete interrupt once the programmed number of transfers are finished. When enabled (`SPORT_IEN_B.TF` =1), when the last bit of last word of the programmed number of transfers is shifted out or received completely, an interrupt is generated. When disabled (`SPORT_IEN_B.TF` =0), no interrupt is generated by SPORT.	
		0	Transfer Finish Interrupt is disabled
		1	Transfer Finish Interrupt is Enabled

Half SPORT A Number of Transfers

This register specifies the number of transfers of words to transfer or receive depending on `SPORT_CTL_A.SPTRAN`.

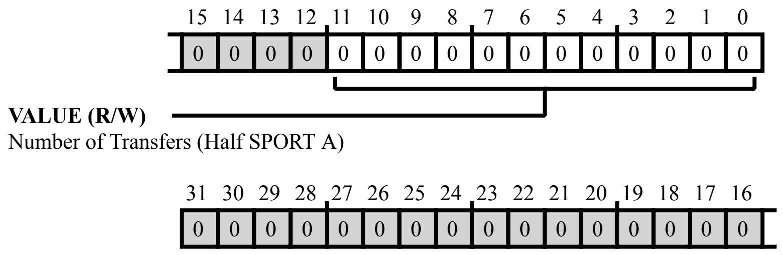


Figure 16-20: SPORT_NUMTRAN_A Register Diagram

Table 16-9: SPORT_NUMTRAN_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/W)	VALUE	Number of Transfers (Half SPORT A).

Half SPORT B Number of Transfers

This register specifies the number of transfers of the words to transfer or receive depending on `SPORT_CTL_B.SPTRAN`.

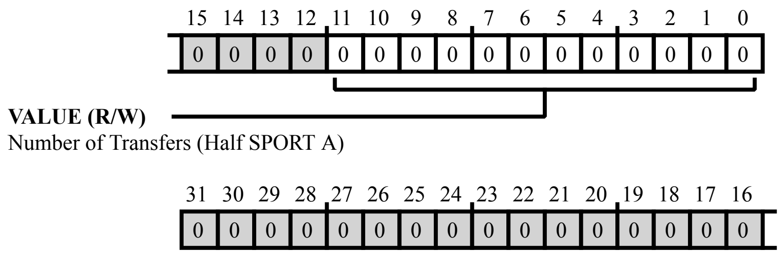


Figure 16-21: SPORT_NUMTRAN_B Register Diagram

Table 16-10: SPORT_NUMTRAN_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/W)	VALUE	Number of Transfers (Half SPORT A).

Half SPORT 'A' Rx Buffer

This register buffers the half SPORT's receive data. This buffer becomes active when the half SPORT is configured to receive data. After a complete word has been received in receive shifter, it is placed into this register. This data can be read in core mode (in interrupt-based or polling-based mechanism) or directly DMA'd into processor memory using DMA controller.

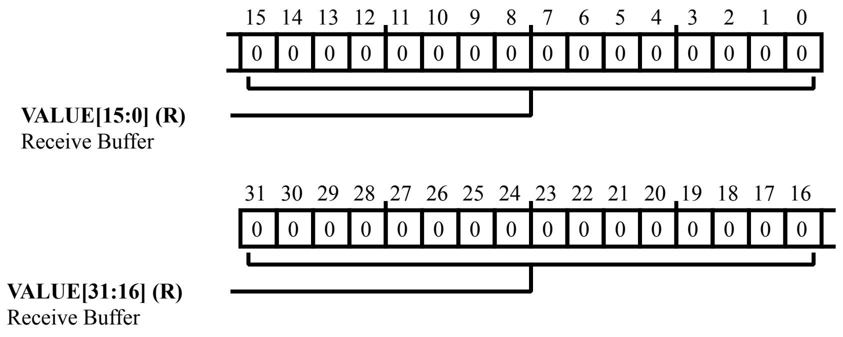


Figure 16-22: SPORT_RX_A Register Diagram

Table 16-11: SPORT_RX_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Receive Buffer. These bits hold the half SPORT's channel receive data.

Half SPORT 'B' Rx Buffer

This register buffers the half SPORT's channel receive data. This buffer becomes active when the half SPORT is configured to receive data. After a complete word has been received in receive shifter, it is placed into this register. This data can be read in core mode or directly DMA'd into processor memory using DMA controller.

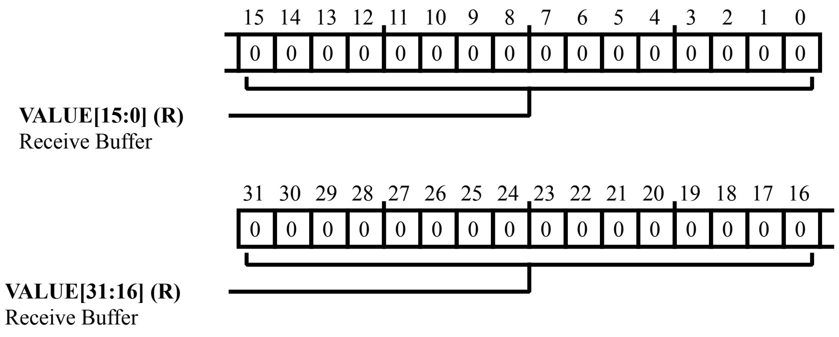


Figure 16-23: SPORT_RX_B Register Diagram

Table 16-12: SPORT_RX_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Receive Buffer. These bits hold the half SPORT's receive data.

Half SPORT A's Status

This register contains all the status fields in the half SPORT A. Detected errors are frame sync violations or buffer over/underflow conditions.

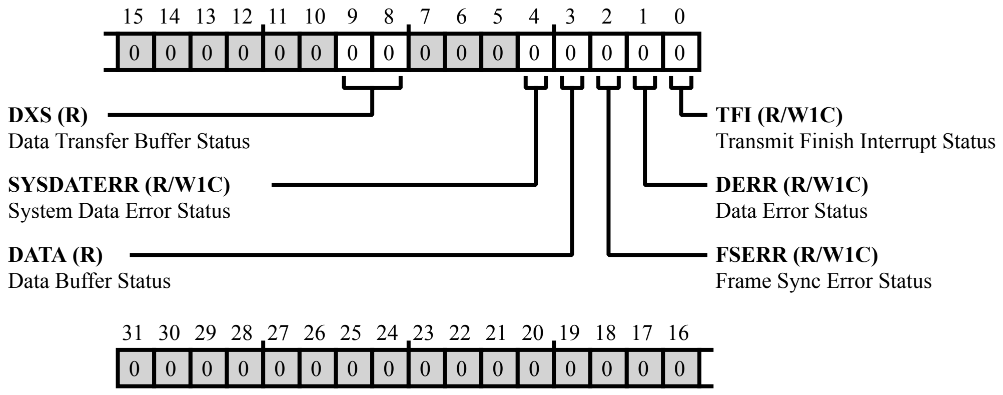


Figure 16-24: SPORT_STAT_A Register Diagram

Table 16-13: SPORT_STAT_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/NW)	DXS	Data Transfer Buffer Status. This bit indicates the status of the half SPORT A's data buffer.	
		0	Empty
		1	Reserved
		2	Partially full
		3	Full
4 (R/W1C)	SYSDATERR	System Data Error Status. This bit indicates the error status for the half SPORT's data buffers during system transfer of data. For transmit (`SPORT_CTL_A.SPTRAN` =1), `SPORT_STAT_A.SYSDATERR` indicates transmit overflow status. For receive (`SPORT_CTL_A.SPTRAN` =0), `SPORT_STAT_A.SYSDATERR` indicates receive underflow status.	
		0	No Error
		1	System Transfer overflow for TXFIFO or System Transfer underflow for RXFIFO
3 (R/NW)	DATA	Data Buffer Status. This field indicates only the status of the data buffers in Half SPORT A.	
		0	Transmit FIFO is full or receive FIFO is empty
		1	Transmit FIFO is not full or receive FIFO is not empty

Table 16-13: SPORT_STAT_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	FSERR	Frame Sync Error Status. This bit indicates that the half SPORT has detected a frame sync when the bit count (bits remaining in the frame) is non-zero or it has seen frame sync active for less than the word length in case of late frame sync.i	
		0	No error
		1	Frame Sync Error occurred
1 (R/W1C)	DERR	Data Error Status. This bit indicates the error status for the half SPORT's data buffers. During transmit (`SPORT_CTL_A.SPTRAN` =1), this bit indicates transmit underflow status. During receive (`SPORT_CTL_A.SPTRAN` =0), this bit indicates receive overflow status.	
		0	No error
		1	Error (transmit underflow or receive overflow)
0 (R/W1C)	TFI	Transmit Finish Interrupt Status. This bit shows when the half SPORT issues its transmission complete interrupt once the programmed number of transfers are finished. When it is 1, the last bit of last word of the programmed number of transfers is shifted out or received completely. When 0, the total number of transfers are not finished.	
		0	Last bit is not transmitted/received
		1	Last bit Transmitted/received

Half SPORT B's Status

This register contains all the status fields present in the half SPORT B. Detected errors are frame sync violations or buffer over/underflow conditions.

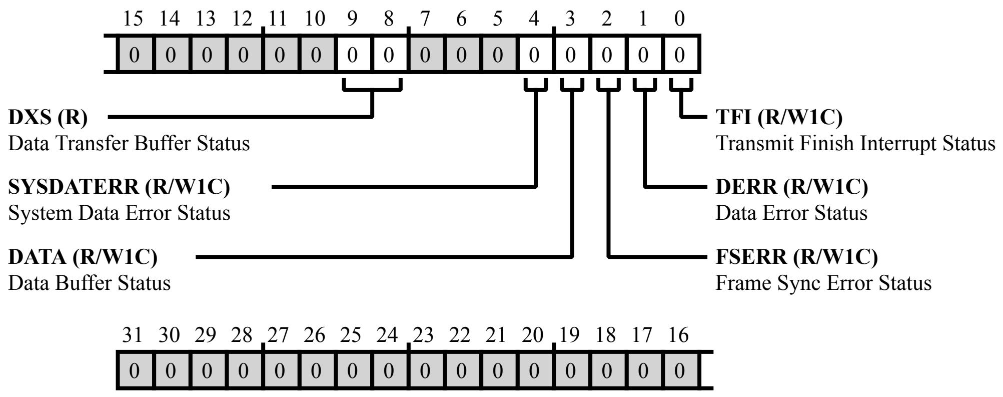


Figure 16-25: SPORT_STAT_B Register Diagram

Table 16-14: SPORT_STAT_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/NW)	DXS	Data Transfer Buffer Status. This bit indicates the status of the half SPORT B's data buffer.	
		0	Empty
		1	Reserved
		2	Partially full
		3	Full
4 (R/W1C)	SYSDATERR	System Data Error Status. This bit indicates the error status for the half SPORT's data buffers during system transfer of data. For transmit (`SPORT_CTL_A.SPTRAN` =1), `SPORT_STAT_B.SYSDATERR` indicates transmit overflow status. For receive (`SPORT_CTL_A.SPTRAN` =0), `SPORT_STAT_B.SYSDATERR` indicates receive underflow status.	
		0	No Error
		1	System Transfer overflow for TXFIFO or System Transfer underflow for RXFIFO
3 (R/W1C)	DATA	Data Buffer Status. This field indicates only the status of the data buffers in Half SPORT B.	
		0	Transmit FIFO is full or receive FIFO is empty
		1	Transmit FIFO is not full or receive FIFO is not empty

Table 16-14: SPORT_STAT_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	FSERR	Frame Sync Error Status. This bit indicates that the half SPORT has detected a frame sync when the bit count (bits remaining in the frame) is non-zero or it has seen frame sync active for less than the word length in case of late frame sync.	
		0	No error
		1	Error (non-zero bit count at frame sync)
1 (R/W1C)	DERR	Data Error Status. This bit indicates the error status for the half SPORT B's data buffers. During transmit (`SPORT_CTL_B.SPTRAN` =1), this bit indicates transmit underflow status. During receive (`SPORT_CTL_B.SPTRAN` =0), this bit indicates receive overflow status.	
		0	No error
		1	Error (transmit underflow or receive overflow)
0 (R/W1C)	TFI	Transmit Finish Interrupt Status. This bit shows when the half SPORT issues its transmission complete interrupt once the programmed number of transfers are finished. When it is 1, the last bit of last word of the programmed number of transfers is shifted out or received completely. When 0, the total number of transfers are not finished.	
		0	Last bit is not transmitted/received
		1	Last bit Transmitted/received

Half SPORT 'A' CNV Width

This register contains the settings related to the SPT_CNV signal for Half SPORT A

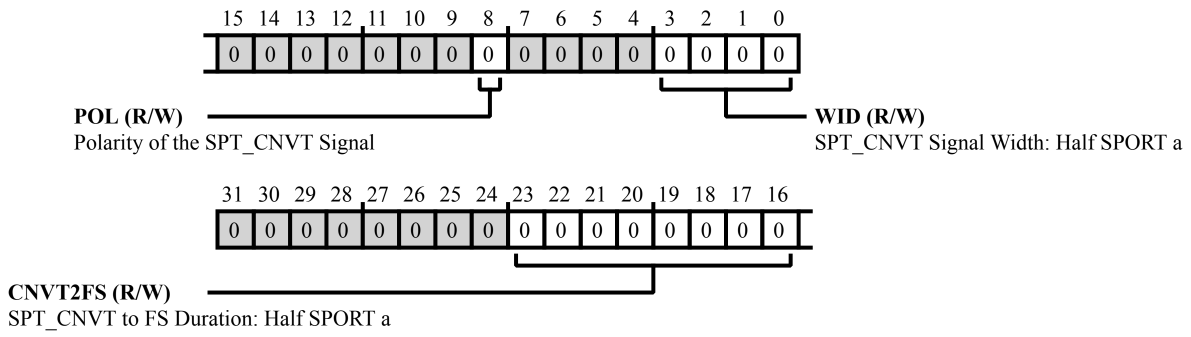


Figure 16-26: SPORT_CNVT_A Register Diagram

Table 16-15: SPORT_CNVT_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:16 (R/W)	CNVT2FS	SPT_CNVT to FS Duration: Half SPORT a. This field contains the value of the number of clocks which would be programmed corresponding to the desired time duration from assertion of SPT_CNV signal to Frame sync signal for Half SPORT A	
8 (R/W)	POL	Polarity of the SPT_CNVT Signal. This bit decides the polarity of the SPT_CNV signal.	
		0	Active High Polarity
		1	Active low Polarity
3:0 (R/W)	WID	SPT_CNVT Signal Width: Half SPORT a. This field contains the value of the number of serial clocks for which SPT_CNV signal should be active for Half SPORT A.	

Half SPORT 'B' CNV Width

This register contains the settings related to the SPT_CNV signal for Half SPORT B

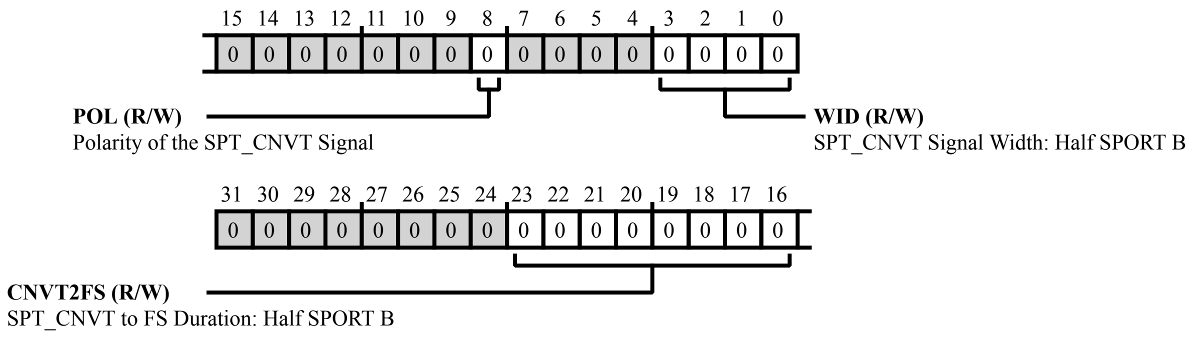


Figure 16-27: SPORT_CNVT_B Register Diagram

Table 16-16: SPORT_CNVT_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:16 (R/W)	CNVT2FS	SPT_CNVT to FS Duration: Half SPORT B. This field contains the value of the number of clocks which would be programmed corresponding to the desired time duration from assertion of SPT_CNV signal to Frame sync signal for Half SPORT B.	
8 (R/W)	POL	Polarity of the SPT_CNVT Signal. This bit decides the polarity of the SPT_CNV signal.	
		0	Active High Polarity
		1	Active low Polarity
3:0 (R/W)	WID	SPT_CNVT Signal Width: Half SPORT B. This field contains the value of the number of clocks which would be programmed corresponding to the desired width of the SPT_CNV signal for Half SPORT B.	

Half SPORT 'A' Tx Buffer

This register buffers the half SPORT's transmit data. This register must be loaded with the data to be transmitted if the half SPORT is configured to transmit. Either a program running on the processor core may load the data into the buffer (word-by-word process) or the DMA controller may automatically load the data into the buffer (DMA process).

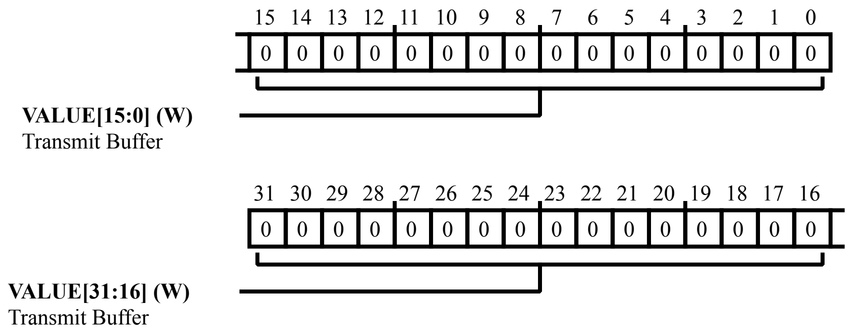


Figure 16-28: SPORT_TX_A Register Diagram

Table 16-17: SPORT_TX_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Transmit Buffer. The `SPORT_TX_A.VALUE` bits hold the half SPORT's channel transmit data.

Half SPORT 'B' Tx Buffer

This register buffers the half SPORT's channel transmit data. This register must be loaded with the data to be transmitted. Either a program running on the processor core may load the data into the buffer (word-by-word process) or the DMA controller may automatically load the data into the buffer (DMA process).

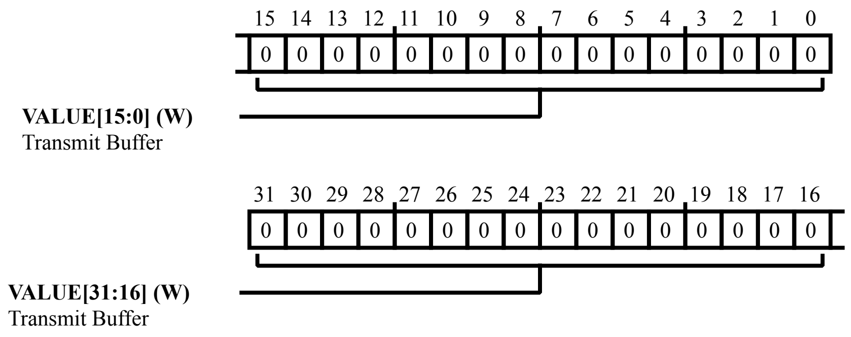


Figure 16-29: SPORT_TX_B Register Diagram

Table 16-18: SPORT_TX_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Transmit Buffer. The `SPORT_TX_B.VALUE` bits hold the half SPORT's transmit data.

17 Universal Asynchronous Receiver/Transmitter (UART)

The UART peripheral is a serial full duplex universal asynchronous receiver/transmitter, compatible with the industry standard 16450/16550. The serial communication follows an asynchronous protocol supporting various word lengths, stop bits, and parity generation options. The ADuCM302x MCU has one UART peripheral.

UART Features

This UART also contains interrupt handling hardware and provides a fractional divider that facilitates high accuracy baud rate generation.

Interrupts may be generated from a number of unique events, such as data buffer full/empty, transfer error detection, and break detection.

While the UART implementation supports modem control signals, only serial transmit and receive functionality is supported.

The following modem inputs are tied high internally:

- `UART_MSR.DCD`
- `UART_MSR.RI`
- `UART_MSR.DSR`
- `UART_MSR.CTS`

The following modem outputs are not connected:

- `UART_MCR.RTS`
- `UART_MCR.DTR`
- `UART_MCR.OUT1`
- `UART_MCR.OUT2`

UART Functional Description

The UART peripheral supports industry standard asynchronous serial communication and can transfer data through the transmit and receive pins. It supports the word lengths from 7 to 12 bits. Transmit operation is initiated by writing to the transmit holding register (`UART_TX`). Receive operation uses the same data format as the transmit configuration, except for the number of stop bits, which is always one.

UART Block Diagram

The UART block diagram is shown below.

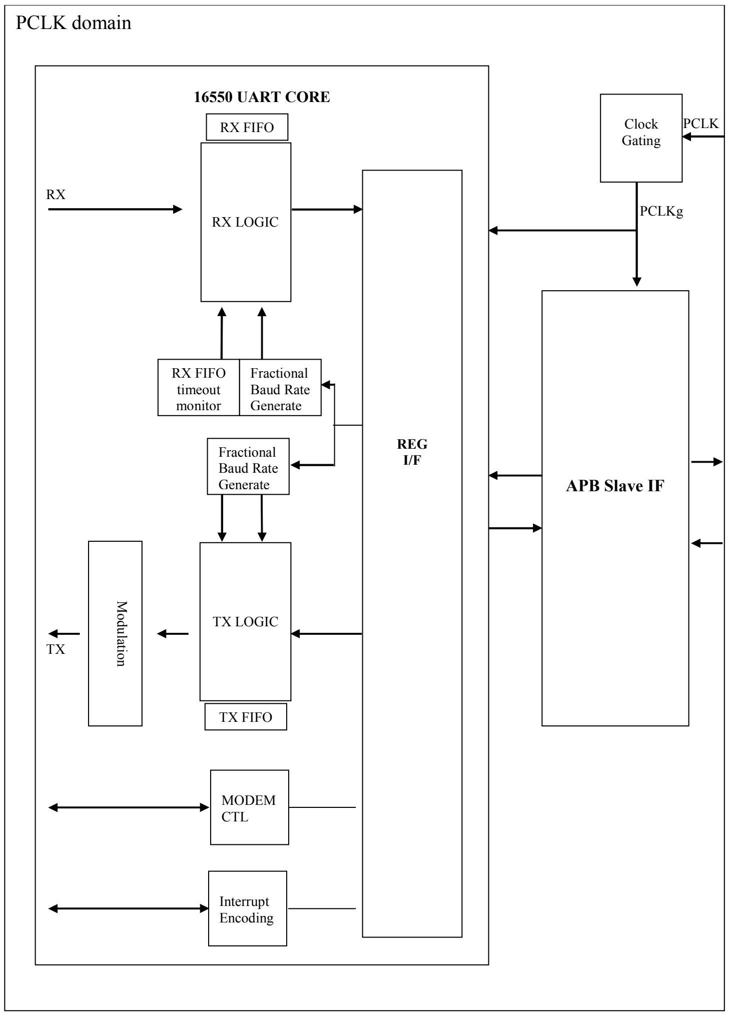


Figure 17-1: UART Block Diagram

UART Operations

Serial Communications

The asynchronous serial communication protocol supports the following options:

- 5 to 8 data bits
- 1, 2 or 1 and 1/2 stop bits
- None, or even or odd parity
- Programmable over sample rate by 4, 8, 16, 32
- Baud rate = PCLK / ((M + N/2048) × 2^{OSR+2} × DIV)

  where,

  OSR (`UART_LCR2.OSR`) = 0 to 3

  DIV (`UART_DIV`) = 1 to 65535

  M (`UART_FBR.DIVM`) = 1 to 3

  N (`UART_FBR.DIVN`) = 0 to 2047

All data words require a start bit and at least one stop bit. This creates a range from 7 bits to 12 bits for each word.

Transmit operation is initiated by writing to the transmit holding register (`UART_TX`). After a synchronization delay, the data is moved to the transmit shift register where it will be shifted out at a baud (bit) rate equal to PCLK / ((M + N/2048) × 2^{OSR+2} × DIV) with start, stop, and parity bits appended as required. All data words begin with a low going start bit. The transfer of the transmit holding register to the transmit shift register causes the transmit register empty status bit (`UART_LSR.THRE`) to be set.

Receive operation uses the same data format as the transmit configuration, except for the number of stop bits, which is always one. After detecting the start bit, the received word is shifted to the receive shift register. After the appropriate number of bits (including stop bits) are received, the receive shift register is transferred to the receive buffer register, after the appropriate synchronization delay, and the receive buffer register full status flag (`UART_IIR.STAT`) is updated.

A sampling clock equal to 2^{OSR+2} times the baud rate is used to sample the data as close to the midpoint of the bit as possible. A receive filter is also present that removes spurious pulses less than the sampling clock period.

NOTE: Data is transmitted and received least significant bit first, that is, transmit shift register, bit 0.

Baud Rate Generator

To bypass the fractional divider, set `UART_FBR.FBEN` = 0.

This results in a baud rate = PCLK / (2^{OSR+2} × DIV).

To estimate the `UART_FBR.DIVN` and `UART_FBR.DIVM` values, the fractional baud rate generator fine tunes the baud rate if the integer `UART_DIV` gives an unreasonable error to the rate.

NOTE: Setting `UART_DIV` to 0 disables the UART logic.

The following tables show example baud rates assuming a 26 MHz and 16 MHz input clock.

Table 17-1: Baud Rate Examples Based on 26 MHz PCLK

Baud Rate	OSR	DIV	DIVM	DIVN	Actual	Error
9600	3	24	3	1078	9600.29	0.0031%
19200	3	12	3	1078	19200.59	0.0031%
38400	3	8	2	1321	38397.64	–0.0062%
57600	3	4	3	1078	57601.77	0.0031%
115200	3	4	1	1563	115203.5	0.0031%
230400	3	2	1	1563	230407.1	0.0031%
460800	3	1	1	1563	460814.2	0.0031%
921,600	2	1	1	1563	921628.4	0.0031%
1,000,000	2	1	1	1280	1000000	0.0%
1,500,000	2	1	1	171	1499775	-0.0150%

Table 17-2: Baud Rate Examples Based on a 16 MHz PCLK

Baud Rates	OSR	DIV	DIVM	DIVN	Actual	% Error
9600	3	17	3	131	9599.25	–0.0078%
19200	3	8	3	523	19199.04	–0.0050%
38400	3	4	3	523	38398.08	–0.0050%
57600	3	8	1	174	57605.76	0.0100%
115200	3	2	2	348	115211.5	0.0100%
230400	3	2	1	174	230423	0.0100%
460800	3	1	1	174	460846.1	0.0100%
921,600	2	1	1	174	921692.2	0.0100%
1,000,000	2	1	1	0	1000000	0.0000%
1,500,000	1	1	1	683	1499816.9	-0.012%

UART Operating Modes

The UART used by the ADuCM302x MCU supports the following modes.

IO Mode

In this mode, the software moves the data to and from the UART. This is accomplished by interrupt service routines that respond to the transmit and receive interrupts by either reading or writing data as appropriate. In this mode, the software must respond within a certain time to prevent overrun errors in the receive channel.

The IO mode also requires polling the status flags to determine when it is okay to move data.

This mode is core intensive and used only if the system can tolerate the overhead. Interrupts can be disabled using the UART interrupt enable register (`UART_IEN`).

Writing to the transmit holding register when it is not empty, or reading from the receive buffer register when it is not full produces an incorrect result. In the former case, the transmit holding register is overwritten by the new word and the previous word is never transmitted, and in the latter case, the previously received word is read again. These errors must be avoided in software by correctly using either interrupts or status register polling. These errors are not detected in the hardware.

DMA Mode

In this mode, user code does not move data to and from the UART. The DMA request signals to the DMA block are generated indicating that the UART is ready to transmit or receive data. These DMA request signals can be disabled in the `UART_IEN` register.

UART Interrupts

The UART peripheral has one output signal to the core interrupt controller representing all Rx and Tx interrupts. The UART interrupt identification register (`UART_IIR`) must be read by the software to determine the cause of the interrupt.

In the IO mode, interrupts may be generated for the following cases:

- Receive buffer register full
- Receive overrun error
- Receive parity error
- Receive framing error
- Receive FIFO timeout if FIFO (16550) is enabled
- Break interrupt (SIN held low)
- Modem status interrupt (changes to `UART_MSR.DCD`, `UART_MSR.RI`, `UART_MSR.DSR`, or `UART_MSR.CTS`)
- Transmit holding register empty

FIFO Mode (16550)

The 16-byte deep transmit FIFO and receive FIFOs are implemented so that the UART is compatible with the industry standard 16550.

By default, the FIFOs are disabled. They are enabled by setting the `UART_FCR.FIFOEN` bit. When enabled, the internal FIFOs are activated, allowing 16 bytes (and 3 bits of error data per byte in receive FIFO) to be stored in both receive and transmit modes. This minimizes the system overhead and maximizes the system efficiency.

The interrupt and/or DMA trigger level of receive FIFO are programmed by the `UART_FCR.RFTRIG` bit. The DMA requests are programmed by the `UART_FCR.FDMAMD` bit. DMA mode 1 only works (in burst mode) when FIFO is enabled.

Auto Baud Rate Detection

The Auto Baud Detection (ABD) block is used to match the baud rates of two UART devices automatically without pre-assumptions. The receiver must be enabled to detect the mode before a common baud rate is configured.

The `UART_ACR.ABE` bit enables the receiver to work in the Auto Baud Detection mode. A 20-bit counter logic counts the number of cycles between the programmed rising or falling edge and another rising or falling edge. An interrupt is generated once the expected edges are reached. The counter may overflow and generate timeout interrupt (for example, continuous break condition, or no expected edges).

For example, if the data byte being received is 0x0D (8'b00001101, carriage return), in 8-bit mode without parity bit, LSB first

DATA0 = 1, DATA1 = 0, DATA2 = 1, DATA3 = 1

DATA4 = DATA5 = DATA6 = DATA7 = 0

There are three falling edges and three rising edges. The `UART_ACR.SEC` can be written to 1 (second edge, or first rising edge), and `UART_ACR.EEC` to 5 (sixth edge, or third rising edge) to count between first rising edge and second rising edge.

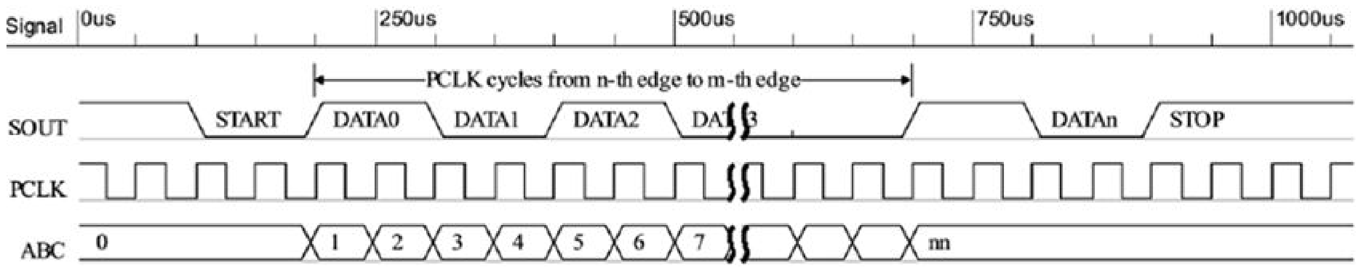


Figure 17-2: Auto Baud Rate Example

Similarly, for 0x7F (8'b01111111, Delete key) , `UART_ACR.SEC` =1 and `UART_ACR.EEC` = 3 to count between first rising edge and second rising edge.

Auto baud rate must be disabled to clear the internal counter and re-enabled for another run (if required).

Based on the UART baud rate configuration, the Auto Baud Detection result can be calculated in the following way:

CNT[19:0] = CountedBits × 2^{OSR+2} × `UART_DIV` × (`UART_FBR.DIVM` + `UART_FBR.DIVN` ÷ 2048)

if CNT < 8 × CountedBits, OSR = 0, `UART_DIV` = 1, `UART_FBR.DIVN` = 512 × CNT ÷ CountedBits - 2048,

else if CNT < 16 × CountedBits, OSR = 1, `UART_DIV` = 1, `UART_FBR.DIVN` = 256 ×CNT ÷ CountedBits - 2048, else if CNT < 32 × CountedBits, OSR = 2, `UART_DIV` = 1, `UART_FBR.DIVN` = 128 ×CNT ÷ CountedBits - 2048,

else if CNT > = 32 × CountedBits, OSR = 3,

if CNT exactly divided by (32 × CountedBits), `UART_DIV` = CNT ÷ 32 ÷ CountedBits,

else `UART_DIV` = $2^{\log_2(\text{CNT} \div 32 \div \text{CountedBits})}$

`UART_FBR.DIVN` = 64 × CNT ÷ `UART_DIV` ÷ CountedBits - 2048

NOTE: To reduce truncation error, the `UART_FBR.DIVM` field is set to 1. The `UART_DIV` field is set to nearest power of 2.

CountedBits is the effective number of bits between an active starting edge and ending edge. It is determined by the application code on selected edges and character used for Auto Baud Detection.

RS485 Half Duplex Mode

To support RS485 half duplexing, SOUT_EN is implemented. The transmit driver must be enabled when SOUT_EN is asserted.

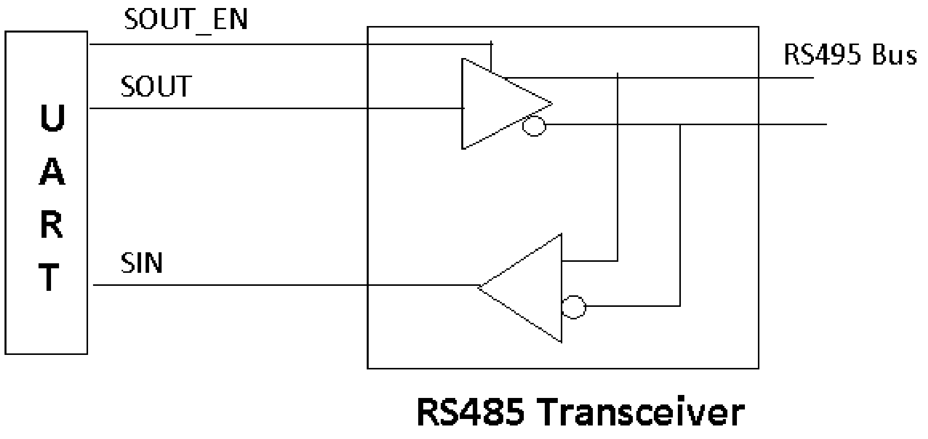


Figure 17-3: UART with RS485 Transceiver

Receive Line Inversion

For specific applications such as UART communication through optical link, the receive line may work at the opposite level (idling at low level). The `UART_CTL.RXINV` field can invert the receive line for this purpose.

NOTE: Do not use Receive line inversion with RS485 applications.

Configure the `UART_CTL.RXINV` field before configuring the UART and enabling Auto Baud.

Clock Gating

The clock driving the UART logic is automatically gated off when idle, and not accessed. This automatic clock gating cannot be disabled by the `UART_CTL.FORCECLK` field.

UART and Power-down Modes

Complete the on-going UART transfers before powering down the chip into hibernate mode. Else, disable the UART by clearing the `UART_DIV` register before going into hibernate.

NOTE: If hibernate mode is selected while a UART transfer is on, the transfer does not continue on returning from hibernate mode. All the intermediate data, states, status logic in the UART are cleared. However, the transmit pads (SOUT and SOUT_EN, if pin mux is selected) may remain active in hibernate mode while transmitting.

After hibernate, the UART can be enabled by setting the `UART_DIV` register (if previously cleared). If DMA mode is needed, `UART_IEN` [5:4] must be configured.

For a clean wake up,

- Prior to hibernate: Disable the UART block by clearing the `UART_DIV` register.

  Or

- After waking from hibernate: Before any UART transaction, apply a break which is at least one frame period longer than the system wake-up time (typically, 10 µs). This ensures that the UART identifie the transaction start condition.

  The wake-up time depends on the clock sources and memory used for instruction code.

The following registers are retained through hibernate mode. Other registers and internal logic are cleared to hardware default value.

- `UART_IEN.ELSI`, `UART_IEN.ERBFI`
- `UART_LCR.BRK`, `UART_LCR.SP`, `UART_LCR.EPS`, `UART_LCR.PEN`, `UART_LCR.STOP`, `UART_LCR.WLS`
- `UART_FCR.RFTRIG`, `UART_FCR.FDMAMD`, `UART_FCR.FIFOEN`
- `UART_FBR.FBEN`, `UART_FBR.DIVM`, `UART_FBR.DIVN`
- `UART_DIV.DIV`
- `UART_LCR2.OSR`
- `UART_CTL.RXINV`, `UART_CTL.FORCECLK`
- `UART_RSC.DISTX`, `UART_RSC.DISRX`, `UART_RSC.OENSP`, `UART_RSC.OENP`

ADuCM302x UART Register Descriptions

UART contains the following registers.

Table 17-3: ADuCM302x UART Register List

Name	Description
UART_ACR	Auto Baud Control
UART_ASRH	Auto Baud Status (High)
UART_ASRL	Auto Baud Status (Low)
UART_CTL	UART Control Register
UART_DIV	Baud Rate Divider
UART_FBR	Fractional Baud Rate
UART_FCR	FIFO Control
UART_IEN	Interrupt Enable
UART_IIR	Interrupt ID
UART_LCR	Line Control
UART_LCR2	Second Line Control
UART_LSR	Line Status
UART_MCR	Modem Control
UART_MSR	Modem Status
UART_RFC	RX FIFO Byte Count
UART_RSC	RS485 Half-duplex Control
UART_RX	Receive Buffer Register
UART_SCR	Scratch Buffer
UART_TFC	TX FIFO Byte Count
UART_TX	Transmit Holding Register

Auto Baud Control

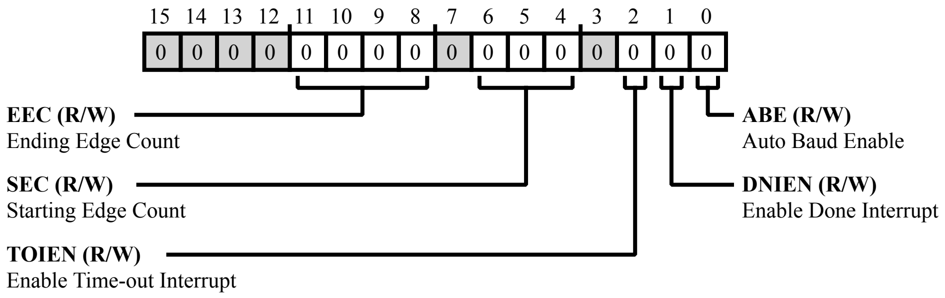


Figure 17-4: UART_ACR Register Diagram

Table 17-4: UART_ACR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:8 (R/W)	EEC	Ending Edge Count.	
		0	First edge
		1	Second edge
		2	Third edge
		3	Fourth edge
		4	Fifth edge
		5	Sixth edge
		6	Seventh edge
		7	Eighth edge
		8	Ninth edge
6:4 (R/W)	SEC	Starting Edge Count.	
		0	First edge (always the falling edge of START bit)
		1	Second edge
		2	Third edge
		3	Fourth edge
		4	Fifth edge
		5	Sixth edge
		6	Seventh edge
		7	Eighth edge
2 (R/W)	TOIEN	Enable Time-out Interrupt.	

Table 17-4: UART_ACR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	DNIEN	Enable Done Interrupt.
0 (R/W)	ABE	Auto Baud Enable.

Auto Baud Status (High)

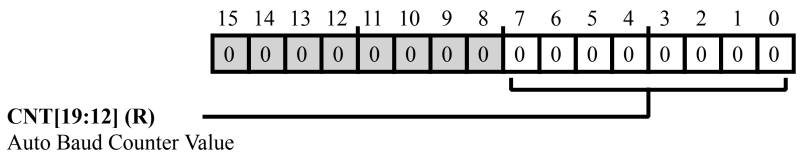


Figure 17-5: UART_ASRH Register Diagram

Table 17-5: UART_ASRH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	CNT	Auto Baud Counter Value.

Auto Baud Status (Low)

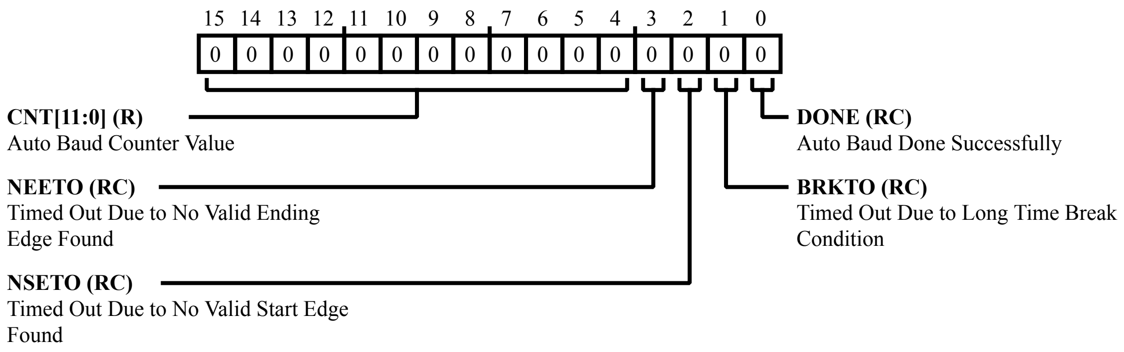

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[11:0] (R)
Auto Baud Counter Value
NEETO (RC)
Timed Out Due to No Valid Ending Edge Found
NSETO (RC)
Timed Out Due to No Valid Start Edge Found
DONE (RC)
Auto Baud Done Successfully
BRKTO (RC)
Timed Out Due to Long Time Break Condition

Figure 17-6: UART_ASRL Register Diagram

Table 17-6: UART_ASRL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:4 (R/NW)	CNT	Auto Baud Counter Value.
3 (RC/NW)	NEETO	Timed Out Due to No Valid Ending Edge Found.
2 (RC/NW)	NSETO	Timed Out Due to No Valid Start Edge Found.
1 (RC/NW)	BRKTO	Timed Out Due to Long Time Break Condition.
0 (RC/NW)	DONE	Auto Baud Done Successfully.

UART Control Register

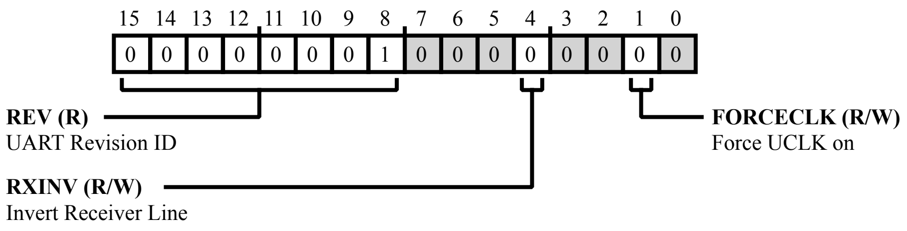


Figure 17-7: UART_CTL Register Diagram

Table 17-7: UART_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:8 (R/NW)	REV	UART Revision ID.	
4 (R/W)	RXINV	Invert Receiver Line.	
		0	Don't invert receiver line (idling high).
		1	Invert receiver line (idling low).
1 (R/W)	FORCECLK	Force UCLK on.	
		0	UCLK automatically gated
		1	UCLK always working

Baud Rate Divider

Internal UART baud generation counters are restarted whenever UART_DIV register accessed by writing, regardless same or different value.

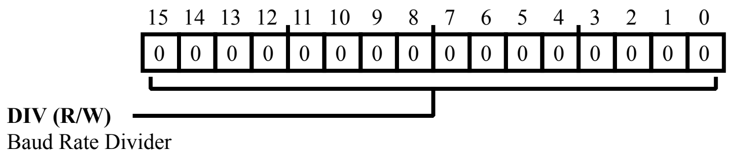


Figure 17-8: UART_DIV Register Diagram

Table 17-8: UART_DIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DIV	Baud Rate Divider. The UART_DIV register should not be 0, which is not specified. Allowed range is 1 to 65535.

Fractional Baud Rate

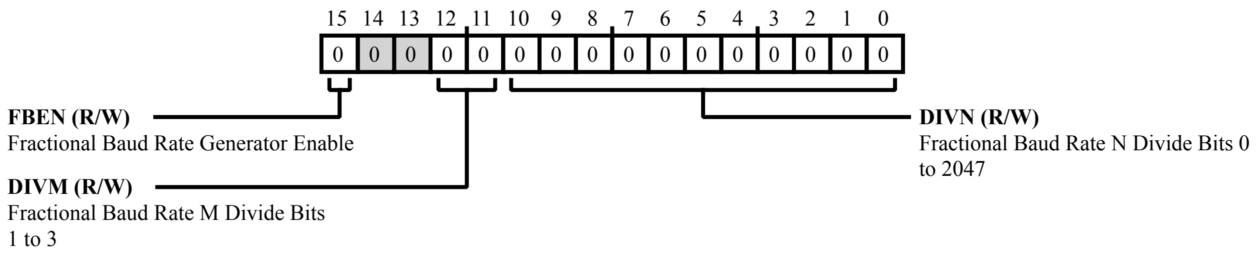

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FBEN (R/W)
Fractional Baud Rate Generator Enable
DIVM (R/W)
Fractional Baud Rate M Divide Bits 1 to 3
DIVN (R/W)
Fractional Baud Rate N Divide Bits 0 to 2047

Figure 17-9: UART_FBR Register Diagram

Table 17-9: UART_FBR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	FBEN	Fractional Baud Rate Generator Enable. The generating of fractional baud rate can be described by the following formula and the final baud rate of UART operation is calculated as: Baudrate = (UCLK / (2 * (`UART_FBR.DIVM` + `UART_FBR.DIVN`/2048) * 16 * `UART_DIV`))
12:11 (R/W)	DIVM	Fractional Baud Rate M Divide Bits 1 to 3. This bit should not be 0 when Fractional Baud Rate enabled.
10:0 (R/W)	DIVN	Fractional Baud Rate N Divide Bits 0 to 2047.

FIFO Control

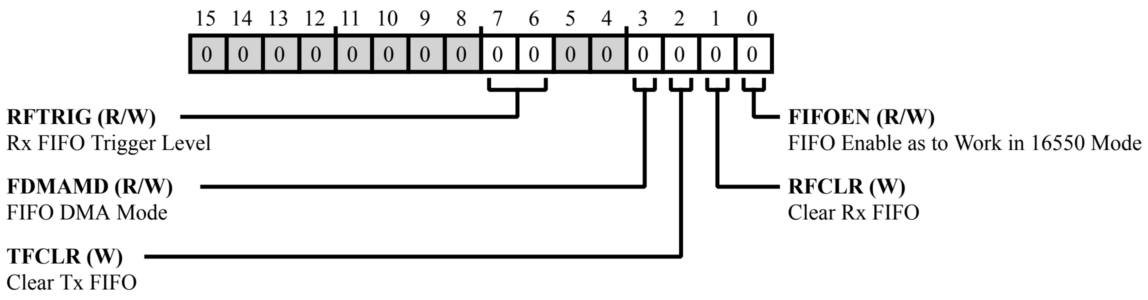


Figure 17-10: UART_FCR Register Diagram

Table 17-10: UART_FCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/W)	RFTRIG	Rx FIFO Trigger Level.	
		0	1 byte to trig RX interrupt
		1	4 byte to trig RX interrupt
		2	8 byte to trig RX interrupt
		3	14 byte to trig RX interrupt
3 (R/W)	FDMAMD	FIFO DMA Mode.	
		0	In DMA mode 0, RX DMA request will be asserted whenever there's data in RBR or RX FIFO and de-assert whenever RBR or RX FIFO is empty; TX DMA request will be asserted whenever THR or TX FIFO is empty and de-assert whenever data written to.
		1	in DMA mode 1, RX DMA request will be asserted whenever RX FIFO trig level or time out reached and de-assert thereafter when RX FIFO is empty; TX DMA request will be asserted whenever TX FIFO is empty and de-assert thereafter when TX FIFO is completely filled up full.
2 (RX/W)	TFCLR	Clear Tx FIFO.	
1 (RX/W)	RFCLR	Clear Rx FIFO.	
0 (R/W)	FIFOEN	FIFO Enable as to Work in 16550 Mode.	

Interrupt Enable

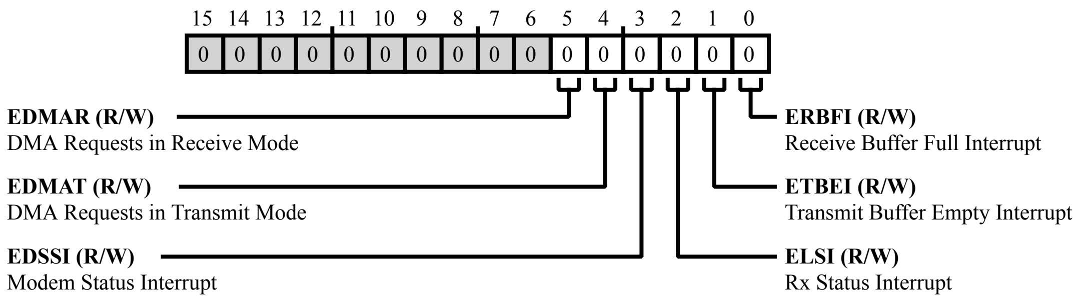


Figure 17-11: UART_IEN Register Diagram

Table 17-11: UART_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	EDMAR	DMA Requests in Receive Mode.	
		0	DMA requests disabled
		1	DMA requests enabled
4 (R/W)	EDMAT	DMA Requests in Transmit Mode.	
		0	DMA requests are disabled
		1	DMA requests are enabled
3 (R/W)	EDSSI	Modem Status Interrupt. Interrupt is generated when any of `UART_MSR.DDCD`, `UART_MSR.TERI`, `UART_MSR.DDSR`, or `UART_MSR.DCTS` are set.	
		0	Interrupt disabled
		1	Interrupt enabled
2 (R/W)	ELSI	Rx Status Interrupt.	
		0	Interrupt disabled
		1	Interrupt enabled
1 (R/W)	ETBEI	Transmit Buffer Empty Interrupt.	
		0	Interrupt disabled
		1	Interrupt enabled
0 (R/W)	ERBFI	Receive Buffer Full Interrupt.	
		0	Interrupt disabled
		1	Interrupt enabled

Interrupt ID

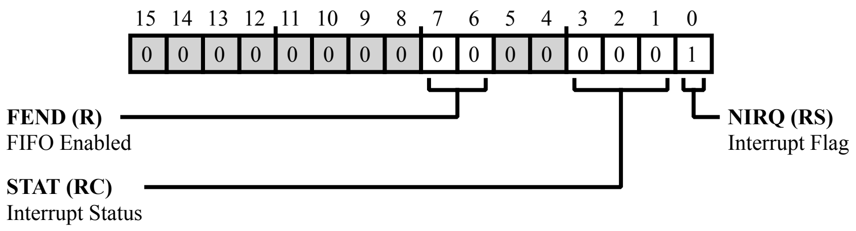


Figure 17-12: UART_IIR Register Diagram

Table 17-12: UART_IIR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/NW)	FEND	FIFO Enabled.	
		0	FIFO not enabled, 16450 mode
		3	FIFO enabled, 16550 mode
3:1 (RC/NW)	STAT	Interrupt Status. When `UART_IIR.NIRQ` is low (active-low), this indicates an interrupt and the `UART_IIR.STAT` bit decoding below is used.	
		0	Modem status interrupt (Read MSR register to clear)
		1	Transmit buffer empty interrupt (Write to Tx register or read IIR register to clear)
		2	Receive buffer full interrupt (Read Rx register to clear)
		3	Receive line status interrupt (Read LSR register to clear)
		6	Receive FIFO time-out interrupt (Read Rx register to clear)
0 (RS/NW)	NIRQ	Interrupt Flag.	

Line Control

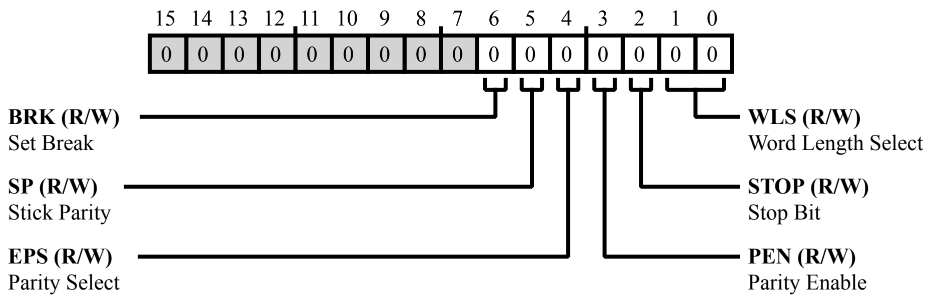


Figure 17-13: UART_LCR Register Diagram

Table 17-13: UART_LCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	BRK	Set Break.	
		0	Normal TxD operation
		1	Force TxD to 0
5 (R/W)	SP	Stick Parity. Used to force parity to defined values. When set, the parity will be based on the following bit settings : `UART_LCR.EPS` = 1 and `UART_LCR.PEN` = 1, parity will be forced to 0. `UART_LCR.EPS` = 0 and `UART_LCR.PEN` = 1, parity will be forced to 1. `UART_LCR.EPS` = X and `UART_LCR.PEN` = 0, no parity will be transmitted	
		0	Parity will not be forced based on Parity Select and Parity Enable bits.
		1	Parity forced based on Parity Select and Parity Enable bits.
4 (R/W)	EPS	Parity Select. This bit only has meaning if parity is enabled (`UART_LCR.PEN` set).	
		0	Odd parity will be transmitted and checked.
		1	Even parity will be transmitted and checked.
3 (R/W)	PEN	Parity Enable. Used to control of the parity bit transmitted and checked. The value transmitted and the value checked will be based on the settings of `UART_LCR.EPS` and `UART_LCR.SP`.	
		0	Parity will not be transmitted or checked.
		1	Parity will be transmitted and checked.

Table 17-13: UART_LCR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	STOP	Stop Bit. Used to control the number of stop bits transmitted. In all cases only the first stop bit will be evaluated on data received.	
		0	Send 1 stop bit regardless of the Word Length Select bit
		1	Send a number of stop bits based on the word length as follows: WLS = 00, 1.5 stop bits transmitted (5-bit word length) WLS = 01 or 10 or 11, 2 stop bits transmitted (6 or 7 or 8-bit word length)
1:0 (R/W)	WLS	Word Length Select. Selects the number of bits per transmission.	
		0	5 bits
		1	6 bits
		2	7 bits
		3	8 bits

Second Line Control

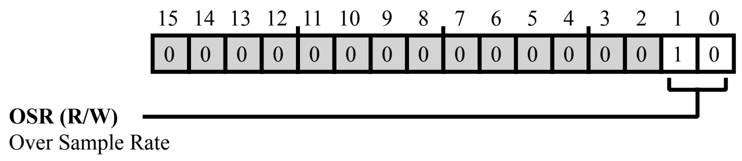


Figure 17-14: UART_LCR2 Register Diagram

Table 17-14: UART_LCR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	OSR	Over Sample Rate.	
		0	Over sample by 4.
		1	Over sample by 8.
		2	Over sample by 16.
		3	Over sample by 32.

Line Status

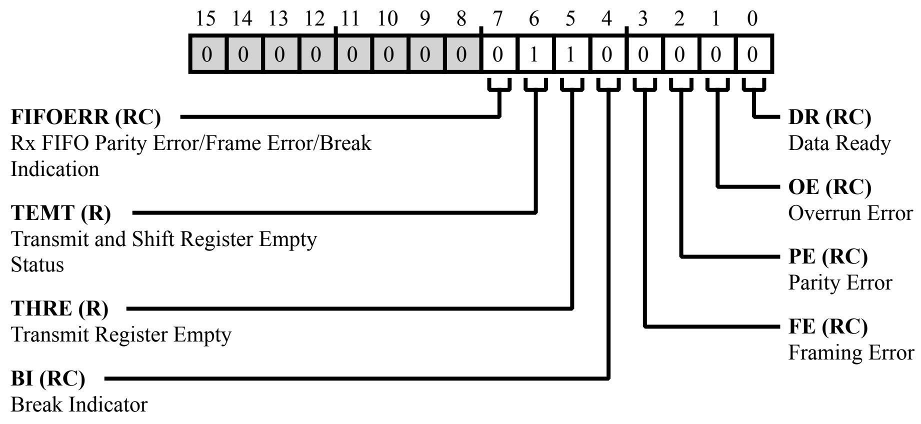


Figure 17-15: UART_LSR Register Diagram

Table 17-15: UART_LSR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (RC/NW)	FIFOERR	Rx FIFO Parity Error/Frame Error/Break Indication. Data byte(s) in RX FIFO have either parity error, frame error or break indication. only used in 16550 mode. Read-clear if no more error in RX FIFO.	
6 (R/NW)	TEMT	Transmit and Shift Register Empty Status.	
		0	Tx register has been written to and contains data to be transmitted. Care should be taken not to overwrite its value.
		1	Tx register and the transmit shift register are empty and it is safe to write new data to the Tx Register. Data has been transmitted.
5 (R/NW)	THRE	Transmit Register Empty. THRE is cleared when UART_RX is read.	
		0	Tx register has been written to and contains data to be transmitted. Care should be taken not to overwrite its value.
		1	Tx register is empty and it is safe to write new data to Tx register The previous data may not have been transmitted yet and can still be present in the shift register.

Table 17-15: UART_LSR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (RC/NW)	BI	Break Indicator. If set, this bit will self clear after UART_LSR is read.	
		0	SIN was not detected to be longer than the maximum word length.
		1	SIN was held low for more than the maximum word length.
3 (RC/NW)	FE	Framing Error. If set, this bit will self clear after UART_LSR is read.	
		0	No invalid Stop bit was detected.
		1	An invalid Stop bit was detected on a received word.
2 (RC/NW)	PE	Parity Error. If set, this bit will self clear after UART_LSR is read.	
		0	No parity error was detected.
		1	A parity error occurred on a received word.
1 (RC/NW)	OE	Overrun Error. If set, this bit will self clear after UART_LSR is read.	
		0	Receive data has not been overwritten.
		1	Receive data was overwritten by new data before Rx register was read.
0 (RC/NW)	DR	Data Ready. This bit is cleared only by reading UART_RX. This bit will not self clear.	
		0	Rx register does not contain new receive data.
		1	Rx register contains receive data that should be read.

Modem Control

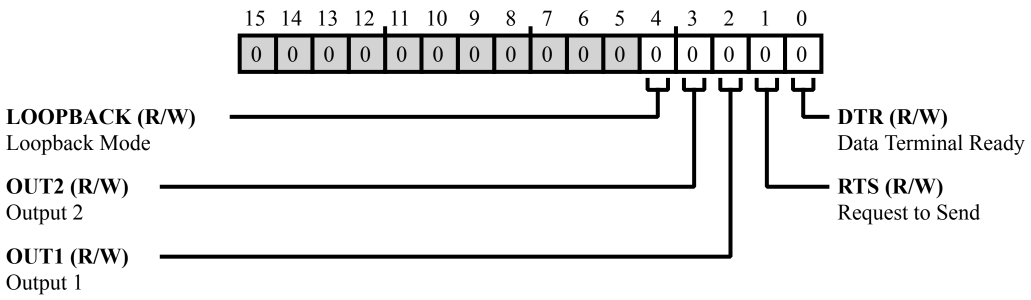


Figure 17-16: UART_MCR Register Diagram

Table 17-16: UART_MCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	LOOPBACK	Loopback Mode. In loop back mode, the SOUT is forced high. The modem signals are also directly connected to the status inputs (`UART_MCR.RTS` to `UART_MSR.CTS`, `UART_MCR.DTR` to `UART_MSR.DSR`, `UART_MCR.OUT1` to `UART_MSR.RI`, and `UART_MCR.OUT2` to `UART_MSR.DCD`).	
		0	Normal operation - loopback disabled
		1	Loopback enabled
3 (R/W)	OUT2	Output 2.	
		0	Force nOUT2 to a logic 1
		1	Force nOUT2 to a logic 0
2 (R/W)	OUT1	Output 1.	
		0	Force nOUT1 to a logic 1
		1	Force nOUT1 to a logic 0
1 (R/W)	RTS	Request to Send.	
		0	Force nRTS to a logic 1
		1	Force nRTS to a logic 0
0 (R/W)	DTR	Data Terminal Ready.	
		0	Force nDTR to a logic 1
		1	Force nDTR to a logic 0

Modem Status

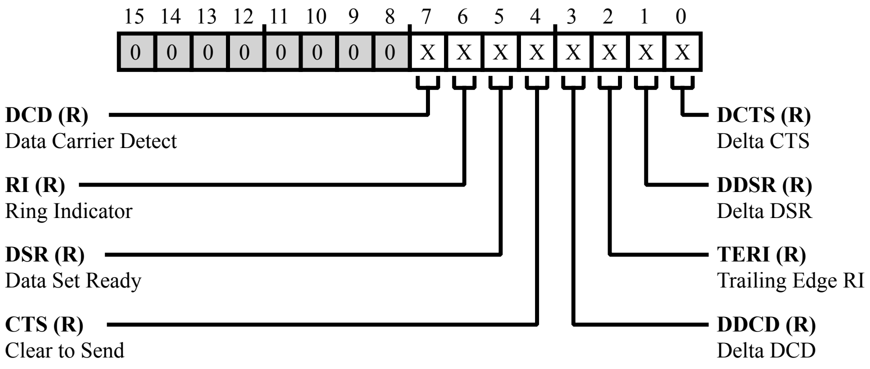


Figure 17-17: UART_MSR Register Diagram

Table 17-17: UART_MSR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	DCD	Data Carrier Detect. This bit reflects the direct status complement of the nDCD pin.	
		0	nDCD is currently logic high.
		1	nDCD is currently logic low.
6 (R/NW)	RI	Ring Indicator. This bit reflects the direct status complement of the nRI pin.	
		0	nRI is currently logic high.
		1	nRI is currently logic low.
5 (R/NW)	DSR	Data Set Ready. This bit reflects the direct status complement of the nDSR pin	
		0	nDSR is currently logic high
		1	nDSR is currently logic low
4 (R/NW)	CTS	Clear to Send. Reflects the direct status complement of the nCTS pin	
		0	nCTS is currently logic high
		1	nCTS is currently logic low

Table 17-17: UART_MSR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/NW)	DDCD	Delta DCD. If set, this bit will self clear after UART_MSR is read.	
		0	Data Carrier Detect bit has not changed state since UART_MSR was last read
		1	Data Carrier Detect bit changed state since UART_MSR last read
2 (R/NW)	TERI	Trailing Edge RI. If set, this bit will self clear after UART_MSR is read.	
		0	Ring Indicator bit has not changed from 0 to 1 since UART_MSR last read
		1	Ring Indicator bit changed from 0 to 1 since UART_MSR last read
1 (R/NW)	DDSR	Delta DSR. If set, this bit will self clear after UART_MSR is read.	
		0	Data Set Ready bit has not changed state since UART_MSR was last read
		1	Data Set Ready bit changed state since UART_MSR last read
0 (R/NW)	DCTS	Delta CTS. If set, this bit will self clear after UART_MSR is read.	
		0	Clear to send bit has not changed state since UART_MSR was last read
		1	Clear to send bit changed state since UART_MSR last read

RX FIFO Byte Count

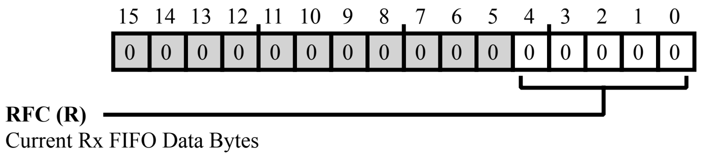


Figure 17-18: UART_RFC Register Diagram

Table 17-18: UART_RFC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/NW)	RFC	Current Rx FIFO Data Bytes.

RS485 Half-duplex Control

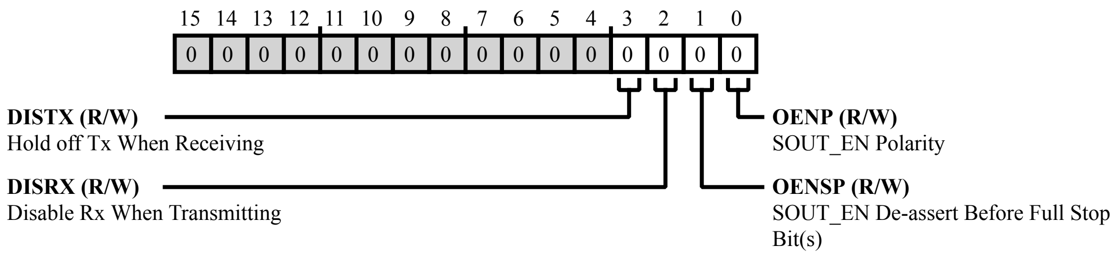


Figure 17-19: UART_RSC Register Diagram

Table 17-19: UART_RSC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	DISTX	Hold off Tx When Receiving.	
2 (R/W)	DISRX	Disable Rx When Transmitting.	
1 (R/W)	OENSP	SOUT_EN De-assert Before Full Stop Bit(s).	
		0	SOUT_EN de-assert same time as full stop bit(s)
		1	SOUT_EN de-assert half-bit earlier than full stop bit(s)
0 (R/W)	OENP	SOUT_EN Polarity.	
		0	High active.
		1	Low active.

Receive Buffer Register

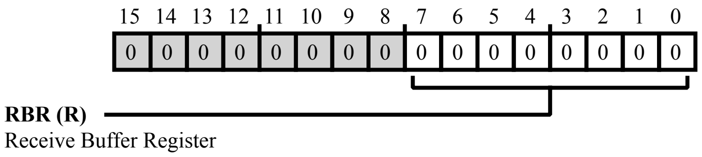


Figure 17-20: UART_RX Register Diagram

Table 17-20: UART_RX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	RBR	Receive Buffer Register.

Scratch Buffer

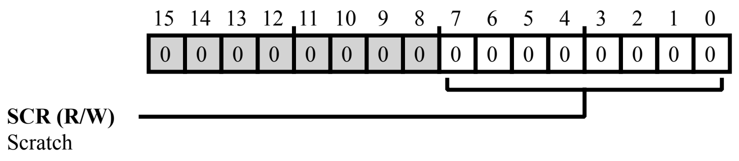


Figure 17-21: UART_SCR Register Diagram

Table 17-21: UART_SCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	SCR	Scratch. An 8-bit register used to store intermediate results. The value contained in UART_SCR does not affect the UART functionality or performance. Only 8 bits of this register are implemented. Bits [15:8] are read only and always return 0x00 when read. Writable with any value from 0 to 255. A read will return the last value written.

TX FIFO Byte Count

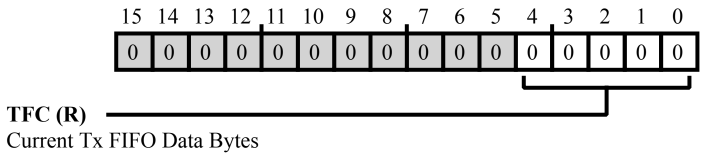


Figure 17-22: UART_TFC Register Diagram

Table 17-22: UART_TFC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/NW)	TFC	Current Tx FIFO Data Bytes.

Transmit Holding Register

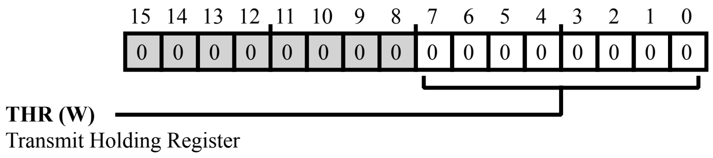


Figure 17-23: UART_TX Register Diagram

Table 17-23: UART_TX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (RX/W)	THR	Transmit Holding Register.

18 Inter-Integrated Circuit (I2C) Interface

The ADuCM302x MCU provides an Inter-Integrated Circuit (I2C) interface with both master and slave functionalities. The peripheral complies with the *I2C Bus Specification, Version 2.1*.

I2C Features

The I2C interface in the ADuCM302x MCU supports the following features:

- 2-byte transmit and receive FIFOs for the master and slave.
- Support for repeated starts.
- Support for 10-bit addressing.
- Master arbitration is supported.
- Continuous read mode for the master or up to 512 bytes fixed read.
- Clock stretching supported for the slave and the master.
- Slave address setting: support for four 7-bit addresses or one 10-bit address and two 7-bit adddresses.
- Support for internal and external loopback.
- Support for DMA.
- Support for bus clear.

I2C Functional Description

I2C Block Diagram

The I2C block diagram is shown below.

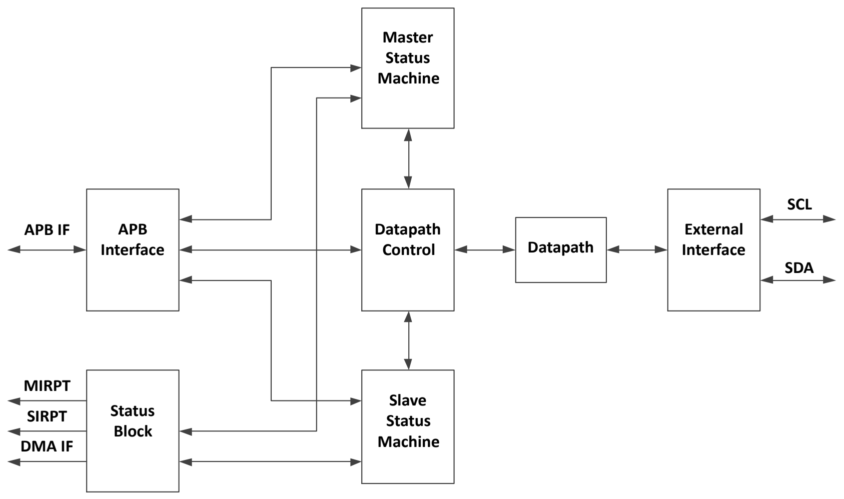


Figure 18-1: I2C Block Diagram

The I2C bus peripheral has two pins used for data transfer. SCL is a serial clock, and SDA is a serial data pin. The pins are configured in a wired-AND format that allows arbitration in a multimaster system.

A master device can be configured to generate the serial clock. The frequency is programmed by the user in the serial clock divisor register. The master channel can be set to operate in fast mode (400 kHz) or standard mode (100 kHz).

The I2C bus peripherals address in the I2C bus system is programmed by the user. This ID may be changed at any time while a transfer is not in progress. The user can set up to four slave addresses that are recognized by the peripheral. The peripheral is implemented with a 2-byte FIFO for each transmit and receive shift register. IRQ pins and status bits in the control registers are available to signal to the MCU core when the FIFO's need to be serviced.

I2C Operating Modes

The GPIOs used for I2C communication must be configured in I2C mode before enabling the I2C peripheral.

As shown in the ***Signal Muxing - Port 0*** table, P0_04 and P0_05 must use the multiplexed function 1. For this, the `GPIO_CFG.PIN04` and `GPIO_CFG.PIN05` fields must be set to 0x1. The drive strength for these pins must be enabled by setting bit 4 and bit 5 of the `GPIO_DS` register.

Master Transfer Initiation

If the master enable bit (`I2C_MCTL.MASEN`) is set, a master transfer sequence is initiated by writing valid value to the `I2C_ADDR1` and `I2C_ADDR2` registers. If there is valid data in the `I2C_MTX` register, it will be the first byte transferred in the sequence after the address byte during a write sequence.

Slave Transfer Initiation

If the slave enable bit (`I2C_SCTL.SLVEN`) is set, a slave transfer sequence is monitored for the device address in the `I2C_ID0`, `I2C_ID1`, `I2C_ID2`, or `I2C_ID3` registers. If the device address is recognized, the part participates in the slave transfer sequence.

A slave operation always starts with the assertion of one of three interrupt sources (`I2C_MSTAT.MRXREQ`/ `I2C_SSTAT.SRXREQ`, `I2C_MSTAT.MTXREQ`/`I2C_SSTAT.STXREQ`, or `I2C_SSTAT.GCINT`). The software looks for a stop interrupt to ensure that the transaction has completed correctly and to deassert the stop interrupt status bit.

Rx/Tx Data FIFOs

The transmit datapath for both master and slave consists of Tx FIFOs 2 bytes deep, MTX and STX, and a transmit shifter. The transmit status bits, `I2C_MSTAT.MTXF` and `I2C_SSTAT.STXFSEREQ` indicate if there is valid data in the Tx FIFO. Data from the Tx FIFO is loaded into the Tx shifter when a serial byte begins transmission. If the Tx FIFO is not full during an active transfer sequence, the transmit request bit (`I2C_MSTAT.MTXREQ` or `I2C_SSTAT.STXREQ`) will assert.

In the slave, if there is no valid data to transmit when the Tx shifter is loaded, the transmit underflow status bit (`I2C_SSTAT.STXUNDR`) will assert.

The master generates a stop condition if there is no data in the transmit FIFO and the master is writing data.

The receive datapath consists of a master and slave Rx FIFO, each two bytes deep, `I2C_MRX` and `I2C_SRX`. The receive request interrupt bits (`I2C_MSTAT.MRXREQ` or `I2C_SSTAT.SRXREQ`) indicate if there is valid data in the Rx FIFO. Data is loaded into the Rx FIFO after each byte is received.

If valid data in the Rx FIFO is overwritten by the Rx shifter, the receive overflow status bit (`I2C_MSTAT.MRXOVR` or `I2C_SSTAT.SRXOVR`) will assert.

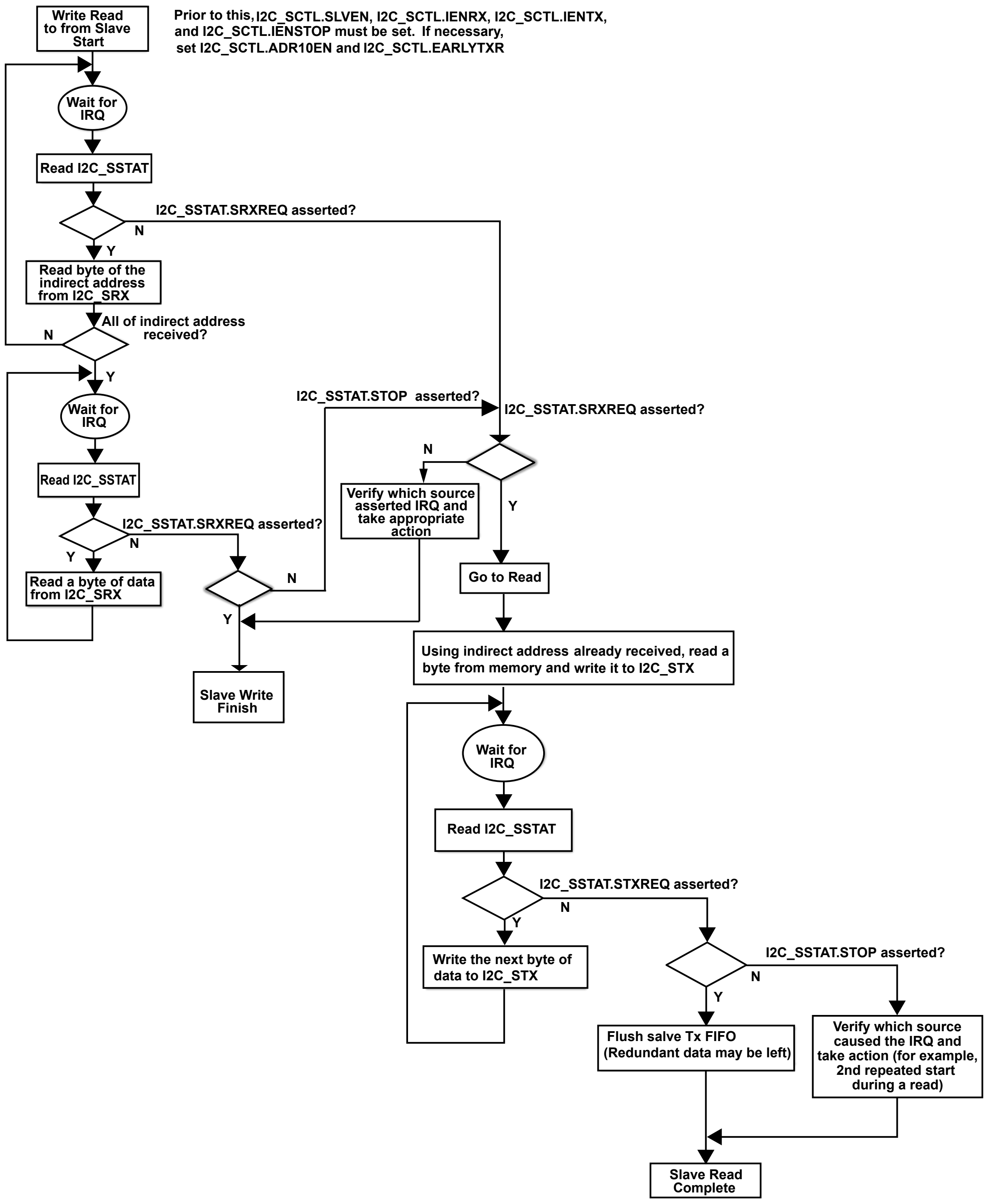

Write Read to from Slave Start
Prior to this, I2C_SCTL.SLVEN, I2C_SCTL.IENRX, I2C_SCTL.IENTX, and I2C_SCTL.IENSTOP must be set. If necessary, set I2C_SCTL.ADR10EN and I2C_SCTL.EARLYTXR
Wait for IRQ
Read I2C_SSTAT
I2C_SSTAT.SRXREQ asserted?
N
Y
Read byte of the indirect address from I2C_SRX
All of indirect address received?
N
Y
Wait for IRQ
Read I2C_SSTAT
I2C_SSTAT.SRXREQ asserted?
N
Y
Read a byte of data from I2C_SRX
N
Y
Slave Write Finish
I2C_SSTAT.STOP asserted?
I2C_SSTAT.SRXREQ asserted?
N
Verify which source asserted IRQ and take appropriate action
Y
Go to Read
Using indirect address already received, read a byte from memory and write it to I2C_STX
Wait for IRQ
Read I2C_SSTAT
I2C_SSTAT.STXREQ asserted?
N
Y
Write the next byte of data to I2C_STX
I2C_SSTAT.STOP asserted?
N
Y
Flush salve Tx FIFO (Redundant data may be left)
Verify which source caused the IRQ and take action (for example, 2nd repeated start during a read)
Slave Read Complete

Figure 18-2: Slave Read/Write Flow

No Acknowledge from Master

When receiving data, the master responds with a NACK if its FIFO is full and an attempt is made to write another byte to the FIFO. This last byte received is not written to the FIFO and is lost.

No Acknowledge from Slave

If the slave does not want to acknowledge a read access, simply not writing data into the slave transmit FIFO results in a NACK.

If the slave does not want to acknowledge a master write, assert the `I2C_SCTL.NACK` bit in the slave control register.

Normally, the slave will ACK all bytes written into the receive FIFO. If the receive FIFO fills up, the slave cannot write further bytes to it, and it will not acknowledge the byte that it was not written to the FIFO. Then, the master must stop the transaction.

The slave does not acknowledge a matching device address if the direction bit is 1 (read) and the transmit FIFO is empty. Therefore, the microcontroller has less time to respond to a slave transmit request and the assertion of ACK. It is recommended to assert the `I2C_SCTL.EARLYTXR` bit.

General Call

If the general call enable bit (`I2C_SCTL.GCEN`) and slave enable bit (`I2C_SCTL.SLVEN`) are set, the device responds to a general call. If the second byte of the general call is 0x06, the I2C interface (master and slave) is reset. The general call interrupt status bit (`I2C_SSTAT.GCINT`) is assertered and general call ID bits (`I2C_SSTAT.GCID`) are set to 0x1. User code must reset the entire system or re-enable the I2C interface.

If the second byte is 0x04 (write programmable part of slave address by hardware), the general call interrupt status bit (`I2C_SSTAT.GCINT`) is asserted and general call ID (`I2C_SSTAT.GCID`) is set to 0x2.

The general call interrupt status bit (`I2C_SSTAT.GCINT`) gets set on any general call after the second byte is received. User code must ensure correct actions such as reprogramming the device address and so on.

If `I2C_SCTL.GCEN` is asserted, the slave always acknowledges the first byte of a general call. It acknowledges the second byte of a general call if the second byte is 0x04 or 0x06, or if the second byte is a hardware general call, and `I2C_SCTL.HGCEN` is asserted.

The `I2C_ALT` register contains an alternate device ID for the hardware general call sequence. If the `I2C_SCTL.HGCEN`, `I2C_SCTL.GCEN`, and `I2C_SCTL.SLVEN` bits are set, the device recognizes a hardware general call. When a general call sequence is issued, and the second byte of the sequence is identical to ALT, the hardware call sequence is recognized for the device.

Generation of Repeated Starts by Master

The master generates a repeated start if the first master address byte register is written while the master is still busy with a transaction. Once the state machine has started to transmit the device address, it is then safe to write to the first master address byte register.

For instance, if a write-repeated start-read/write transaction is required, write to the first master address byte register after the state machine starts to transmit the device address, or after the first TXREQ interrupt is received. When the transmit FIFO empties, a repeated start is generated.

Similarly, if a read-repeated start-read/write transaction is required, write to the first master address byte register after the state machine starts to transmit the device address, or after the first RXREQ interrupt is received. When the requested receive count is reached, a repeated start is generated.

DMA Requests

Four DMA channels (two for the master - MAS_DMA_RX_REQ and MAS_DMA_TX_REQ, and two for the slave - SLV_DMA_RX_REQ and SLV_DMA_TX_REQ) are available. The DMA enable bits are provided in the `I2C_SCTL.STXDMA`, `I2C_SCTL.SRXDMA`, `I2C_MCTL.MTXDMA`, and `I2C_MCTL.MRXDMA` registers.

I2C Reset Mode

The slave state machine is reset when `I2C_SCTL.SLVEN` is written to 0, and the master state machine is reset when `I2C_MCTL.MASEN` is written to 0.

I2C Test Modes

The device can be placed in an internal loopback mode by setting the `I2C_MCTL.LOOPBACK` bit. There are four FIFOs (master Tx and Rx and slave Tx and Rx); therefore in effect, the I2C peripheral can be setup to talk to itself. External loopback can be performed if the master is setup to address the address of the slave.

I2C Low-Power Mode

If the master and slave are both disabled (`I2C_MCTL.MASEN` = `I2C_SCTL.SLVEN` = 0), the device is in its lowest power mode.

Auto Clock Stretching

Automatic clock stretching pauses a transaction by holding the SCL line low. The transaction cannot continue until the line is high.

An I2C slave may receive data at a faster rate, but might need more time to store the received byte or prepare another byte for transmission. Slaves can then hold the SCL line low after reception and acknowledgment of a byte. This forces the master into a wait state until the slave is ready for the next byte transfer.

In the case where the MCU runs at a low frequency or the I2C interrupt has low priority, the automatic clock stretching feature can be used to hold the I2C bus until the I2C interrupt has been processed.

I2C Bus Clear Operation

In the scenario where the external slave holds SDA low to transmit a 0 (or ACK), it does not release SDA until it gets another falling edge on SCL. As a result, the bus hangs. If the I2C master initiates a new transfer, it hits an arbitration lost condition as SDA does not match the address sent.

If the master lost arbitration when the `I2C_MCTL.BUSCLR` bit is set, the master sends out extra nine SCL cycles so that the slave holding the SDA line can release it anywhere before those nine SCL cycles. If the `I2C_MCTL.STOPBUSCLR` bit is asserted, the master stops sending the SCL clocks once SDA is released.

Power-down Considerations

Consider the following when the part is being powered down to hibernate mode. If the master/slave is IDLE (which can be known from the respective status registers), it can be immediately disabled by clearing `I2C_MCTL.MASEN`/`I2C_SCTL.SLVEN` bits in the master/slave control registers respectively.

If they are active, there are four cases:

- I2C is a master and it does Rx:

  In this case, the device receives data based on the count programmed in the `I2C_MRXCNT` register. It would be in continuous read mode if the `I2C_MRXCNT.EXTEND` bit is set. To stop the read transfer, clear the `I2C_MRXCNT.EXTEND` bit and assign the `I2C_MRXCNT` register with `I2C_MCRXCNT.VALUE` + 1, where `I2C_MCRXCNT.VALUE` gives the current read count.

  "+1" signifies that there must be some room for the completion. If the newly programmed value is less than the current count, it receives until the current count overflows and reaches the programmed count. This ends the transfer after receiving the next byte. Once the transaction complete interrupt is received, the core must disable the master by clearing the `I2C_MCTL.MASEN` bit.

- I2C is a master and it does Tx:

  The software must flush the Tx FIFO by setting the `I2C_STAT.MFLUSH` bit, and disable Tx request by clearing the `I2C_MCTL.IENMTX` bit. This ends the current transfer after transmitting the byte in progress. When the transaction complete interrupt is received, it must clear `I2C_MCTL.MASEN` bit.

  NOTE: Disabling the master before completion can cause the bus to hang indefinitely.

- I2C is a slave and it does Rx:

  The software must set the `I2C_SCTL.NACK` bit. This gives a NACK for the next communication, after which, the external master has to STOP. On receiving the STOP interrupt, the core must disable the slave by clearing `I2C_SCTL.SLVEN` bit.

- I2C is a slave and it does Tx:

  Once the Slave transmit starts, it cannot NACK any further transaction (ACK is driven only by the master). So, it has to wait until the external master issues a STOP condition. After receiving the STOP interrupt, the slave can be disabled. This is a clean way to exit. However, if the slave has to be disabled immediately, then it can be done only at the cost of wrong data getting transmitted (all FFs). This is because the SDA line is not driven anymore and pulled up during the data phase. In this case, the bus does not hang.

I2C Data Transfer

The I2C peripheral can be programmed for data transfer through both core and DMA modes. There are dedicated DMA channels for master and slave functionalities. Refer to the Direct Memory Access (DMA) for the DMA channel numbers.

I2C Interrupts and Exceptions

The I2C block can generate interrupts under various conditions in master and slave modes.

ADuCM302x I2C Register Descriptions

I2C Master/Slave (I2C) contains the following registers.

Table 18-1: ADuCM302x I2C Register List

Name	Description
I2C_ADDR1	Master Address Byte 1
I2C_ADDR2	Master Address Byte 2
I2C_ALT	Hardware General Call ID
I2C_ASTRETCH_SCL	Automatic Stretch SCL
I2C_BYT	Start Byte
I2C_DIV	Serial Clock Period Divisor
I2C_ID0	First Slave Address Device ID
I2C_ID1	Second Slave Address Device ID
I2C_ID2	Third Slave Address Device ID
I2C_ID3	Fourth Slave Address Device ID
I2C_MCTL	Master Control
I2C_MCRXCNT	Master Current Receive Data Count
I2C_MRX	Master Receive Data
I2C_MRXCNT	Master Receive Data Count
I2C_MSTAT	Master Status
I2C_MTX	Master Transmit Data
I2C_SCTL	Slave Control
I2C_SHCTL	Shared Control
I2C_SRX	Slave Receive
I2C_SSTAT	Slave I2C Status/Error/IRQ
I2C_STAT	Master and Slave FIFO Status
I2C_STX	Slave Transmit
I2C_TCTL	Timing Control Register

Master Address Byte 1

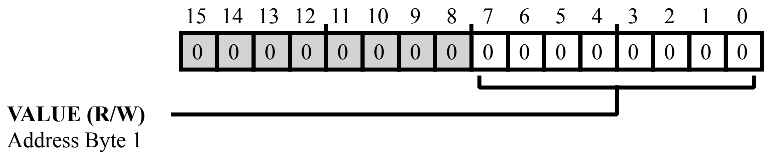


Figure 18-3: I2C_ADDR1 Register Diagram

Table 18-2: I2C_ADDR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Address Byte 1. If a 7-bit address is required, Bit 7 to Bit 1 of `I2C_ADDR1.VALUE` are programmed with the address, and Bit 0 of `I2C_ADDR1.VALUE` is programmed with the direction (read or write). If a 10-bit address is required, Bit 7 to Bit 3 of `I2C_ADDR1.VALUE` are programmed with 11110, Bit 2 to Bit 1 of `I2C_ADDR1.VALUE` are programmed with the 2 MSBs of the address, and Bit 0 of `I2C_ADDR1.VALUE` is programmed to 0.

Master Address Byte 2

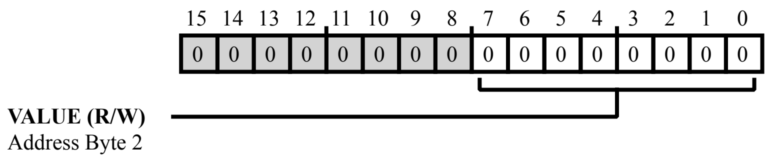


Figure 18-4: I2C_ADDR2 Register Diagram

Table 18-3: I2C_ADDR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Address Byte 2. This register is only required when addressing a slave with a 10-bit address. Bit 7 to Bit 0 of `I2C_ADDR2.VALUE` are programmed with the lower 8 bits of the address.

Hardware General Call ID

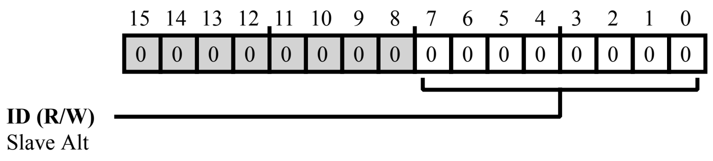


Figure 18-5: I2C_ALT Register Diagram

Table 18-4: I2C_ALT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	ID	Slave Alt. This register is used in conjunction with `I2C_SCTL.HGCEN` to match a master generating a hardware general call. It is used when a master device cannot be programmed with a slave address and the slave has to recognize the master address.

Automatic Stretch SCL

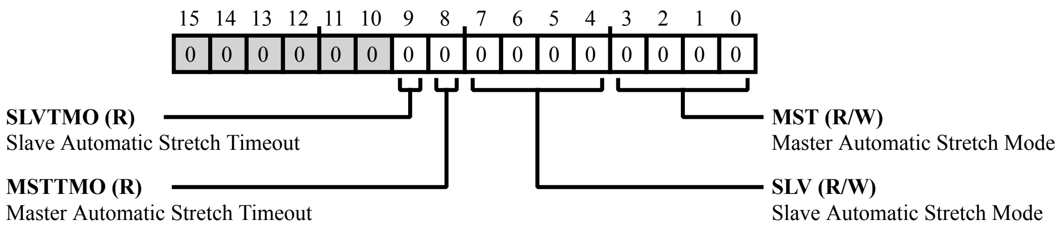

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SLVTMO (R)
Slave Automatic Stretch Timeout
MSTTMO (R)
Master Automatic Stretch Timeout
MST (R/W)
Master Automatic Stretch Mode
SLV (R/W)
Slave Automatic Stretch Mode

Figure 18-6: I2C_ASTRETCH_SCL Register Diagram

Table 18-5: I2C_ASTRETCH_SCL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/NW)	SLVTMO	Slave Automatic Stretch Timeout. Asserts when slave automatic stretch timeout occurs and gets cleared when the bit is read.
8 (R/NW)	MSTTMO	Master Automatic Stretch Timeout. Asserts when master automatic stretch timeout occurs and gets cleared when the bit is read."
7:4 (R/W)	SLV	Slave Automatic Stretch Mode. It defines the automatic stretch mode for slave. As a slave transmitter, SCL clock is automatically stretched start from the negative edge of SCL, when slave Tx FIFO is empty, before send ACK/NACK for address byte, or before send data for data byte. Stretching stops when slave Tx FIFO is no longer empty or timeout occurs. As a slave receiver, SCL clock is automatically stretched start from the negative edge of SCL, when slave Rx FIFO is overflow, before send ACK/NACK. Stretching stops when slave Rx FIFO is no longer overflow or timeout occurs. Slave clock stretch mode is automatically disabled when slv_txint_clr is asserted. Stretch mode will restore the old value after the master transmit interrupt is cleared. Note: When this bit is 4b0000, `I2C_SCTL.EARLYTXR` must be set to 1. Otherwise, CPU has no time to service slave transmit interrupt when receive address and read request.

Table 18-5: I2C_ASTRETCH_SCL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	MST	Master Automatic Stretch Mode. It defines the automatic stretch mode for master. Stretching means hold SCL line low, then theres more time to service the interrupt. And use a timeout to avoid bus lockup. 0000: no SCL clock stretching 0001: stretch SCL up to 2^1 = 2 bit-times 0010: stretch SCL up to 2^2 = 4 bit-times 1110: stretch SCL up to 2^14 bit-times 1111: stretch SCL up to infinity (no timeout) Bit time is decided by `I2C_DIV.HIGH` + `I2C_DIV.LOW`, and count by PCLK. Maximum timeout = 2^14/400 kHz = 40 ms. Compatible with SMBus, which has a timeout of 35 ms. As a master transmitter, SCL clock is automatically stretched start from the negative edge of SCL, when master Tx FIFO is empty, before send data. And stretching will stop when master Tx FIFO is no longer empty or timeout occurs. As a master receiver, SCL clock is automatically stretched start from the negative edge of SCL, when master Rx FIFO is overflow, before ACK/NACK. And stretching will stop when master Rx FIFO is no longer overflow or timeout occurs. Master clock stretch mode will be automatically disabled when mas_txint_clr is asserted or a new address is written. Stretch mode restores the old value after mas_txint_clr is cleared or the address is loaded (RESTART is sent).

Start Byte

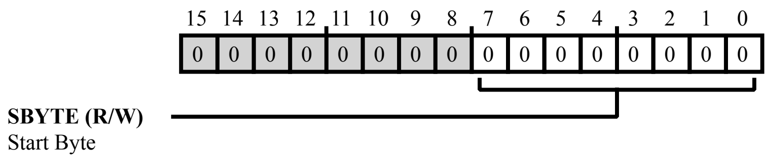


Figure 18-7: I2C_BYT Register Diagram

Table 18-6: I2C_BYT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	SBYTE	Start Byte. Used to generate a start byte at the start of a transaction. To generate a start byte followed by a normal address, first write to `I2C_BYT.SBYTE` then write to the address register (`I2C_ADDR1`). This will drive the byte written in `I2C_BYT.SBYTE` on to the bus followed by a repeated start. This register can be used to drive any byte on to the I2C bus followed by a repeated start (not just a start byte 00000001).

Serial Clock Period Divisor

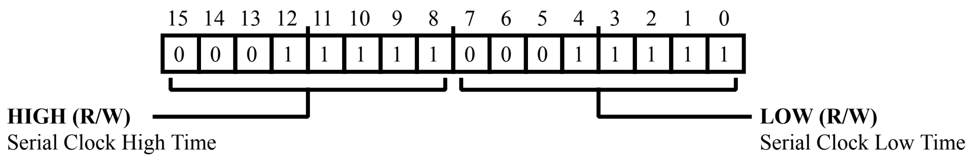


Figure 18-8: I2C_DIV Register Diagram

Table 18-7: I2C_DIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	HIGH	Serial Clock High Time. This register controls the clock high time. The timer is driven by the core clock (UCLK). Use the following equation to derive the required high time. `I2C_DIV.HIGH` = (REQD_HIGH_TIME/PCLK_PERIOD) - 2 For example, to generate a 400 kHz SCL with a low time of 1300ns and a high time of 1200ns, with a core clock frequency 50 MHz: LOTIME = 1300 ns/20 ns - 1 = 0x40 (64 decimal) `I2C_DIV.HIGH` = 1200 ns/20 ns - 2 = 0x3A (58 decimal). This register is reset to 0x1F which gives an SCL high time of 33 PCLK ticks. tHD: STA is also determined by the `I2C_DIV.HIGH`. tHD: STA = (HIGH-1) x pclk_period. As tHD:STA must be 600 ns, with UCLK = 50 MHz the minimum value for `I2C_DIV.HIGH` is 31. This gives an SCL high time of 660 ns.
7:0 (R/W)	LOW	Serial Clock Low Time. This register controls the clock low time. The timer is driven by the core clock (UCLK). Use the following equation to derive the required low time. `I2C_DIV.LOW` = (REQD_LOW_TIME/PCLK_PERIOD) - 1 This register is reset to 0x1F, which gives an SCL low time of 32 PCLK ticks.

First Slave Address Device ID

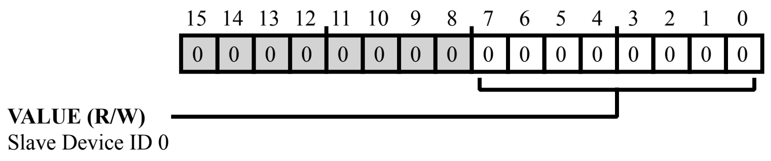


Figure 18-9: I2C_ID0 Register Diagram

Table 18-8: I2C_ID0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Slave Device ID 0. `I2C_ID0.VALUE`[7:1] is programmed with the device ID. `I2C_ID0.VALUE`[0] is don't care. See the `I2C_SCTL.ADR10EN` bit to see how this register is programmed with a 10-bit address.

Second Slave Address Device ID

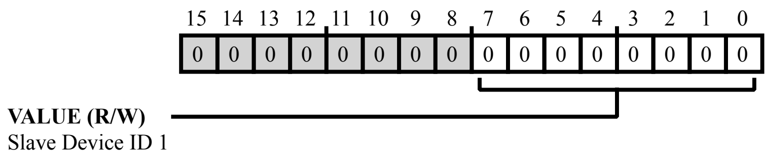


Figure 18-10: I2C_ID1 Register Diagram

Table 18-9: I2C_ID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Slave Device ID 1. `I2C_ID1.VALUE`[7:1] is programmed with the device ID. `I2C_ID1.VALUE`[0] is don't care. See the `I2C_SCTL.ADR10EN` bit to see how this register is programmed with a 10-bit address.

Third Slave Address Device ID

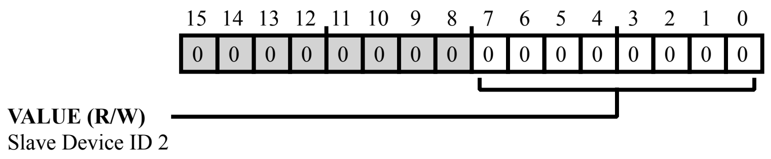


Figure 18-11: I2C_ID2 Register Diagram

Table 18-10: I2C_ID2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Slave Device ID 2. `I2C_ID2.VALUE`[7:1] is programmed with the device ID. `I2C_ID2.VALUE`[0] is don't care. See the `I2C_SCTL.ADR10EN` bit to see how this register is programmed with a 10-bit address.

Fourth Slave Address Device ID

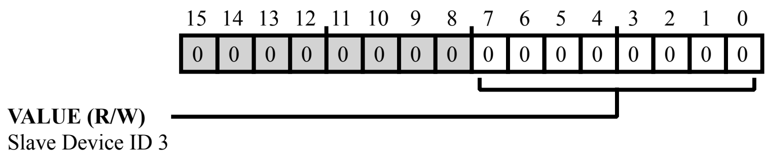


Figure 18-12: I2C_ID3 Register Diagram

Table 18-11: I2C_ID3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Slave Device ID 3. `I2C_ID3.VALUE`[7:1] is programmed with the device ID. `I2C_ID3.VALUE`[0] is don't care. See the `I2C_SCTL.ADR10EN` bit to see how this register is programmed with a 10-bit address.

Master Control

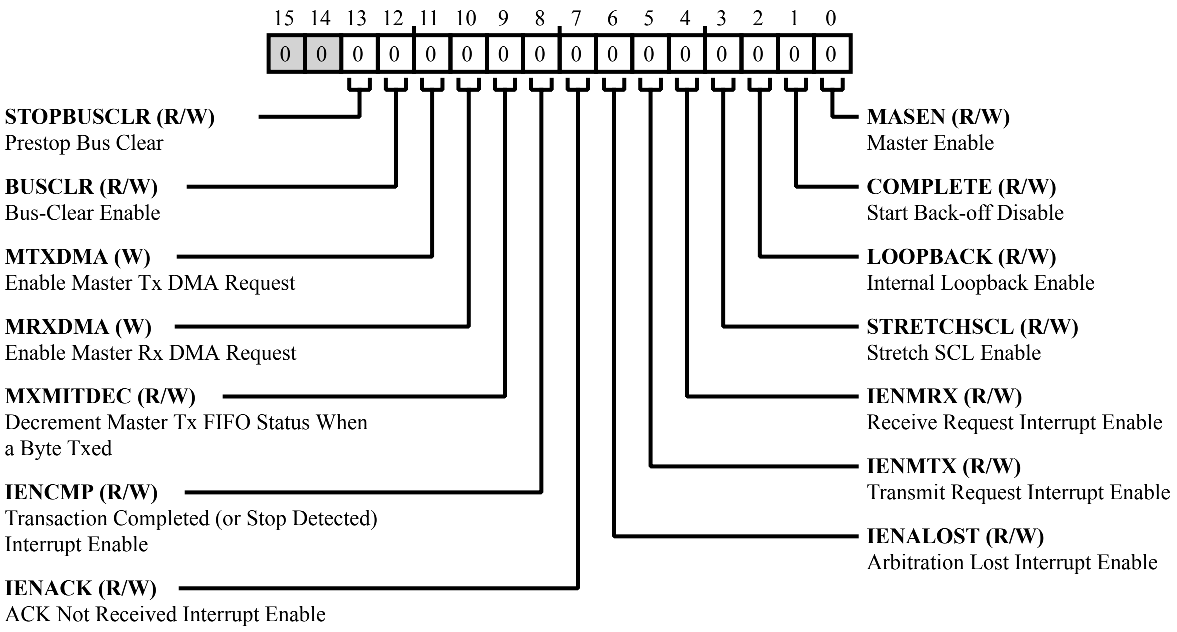


Figure 18-13: I2C_MCTL Register Diagram

Table 18-12: I2C_MCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W)	STOPBUSCLR	Prestop Bus Clear. This bit should be used in conjunction with `I2C_MCTL.BUSCLR`. If this bit is set, the master will stop sending any more SCL clocks if SDA is released before 9 SCL cycles.
12 (R/W)	BUSCLR	Bus-Clear Enable. If this bit is set, the master will initiate a Bus-Clear operation by sending up to 9 extra SCL cycles if the arbitration was lost. This bit is added to come out of a SDA-stuck situation due to a misbehaving slave and hence it should be used with caution. It should be set only when there is no other active I2C master.
11 (RX/W)	MTXDMA	Enable Master Tx DMA Request. Set to 1 by user code to enable I2C master DMA Tx requests. Cleared by user code to disable DMA mode.
10 (RX/W)	MRXDMA	Enable Master Rx DMA Request. Set to 1 by user code to enable I2C master DMA Rx requests. Cleared by user code to disable DMA mode.

Table 18-12: I2C_MCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W)	MXMITDEC	Decrement Master Tx FIFO Status When a Byte Txed. Decrement master TX FIFO status when a byte has been transmitted. If set to 1, the master transmit FIFO status is decremented when a byte has been transmitted. If set to 0, the master transmit FIFO status is decremented when the byte is unloaded from the FIFO into a shadow register at the start of byte transmission.
8 (R/W)	IENCMP	Transaction Completed (or Stop Detected) Interrupt Enable. Transaction completed interrupt enable. When asserted an interrupt is generated when a STOP is detected.
7 (R/W)	IENACK	ACK Not Received Interrupt Enable.
6 (R/W)	IENALOST	Arbitration Lost Interrupt Enable.
5 (R/W)	IENMTX	Transmit Request Interrupt Enable.
4 (R/W)	IENMRX	Receive Request Interrupt Enable.
3 (R/W)	STRETCHSCL	Stretch SCL Enable. Setting this bit tells the device if SCL is 0 hold it at 0; or if SCL is 1 then when it next goes to 0 hold it at 0.
2 (R/W)	LOOPBACK	Internal Loopback Enable. When this bit is set, SCL and SDA lines are muxed onto their corresponding inputs. Note that is also possible for the master to loop back a transfer to the slave as long as the device address corresponds, i.e. external loopback.
1 (R/W)	COMPLETE	Start Back-off Disable. Setting this bit enables the device to compete for ownership even when another device is driving a start condition.
0 (R/W)	MASEN	Master Enable. When the bit is 1 the master is enabled. When the bit is 0 all master state machine flops are held in reset and the master is disabled. The master should be disabled when not in use as this will gate the clock to the master and save power. This bit should not be cleared until a transaction has completed, see the `I2C_MSTAT.TCOMP`. APB writable register bits are not reset by this bit. Cleared by default.

Master Current Receive Data Count

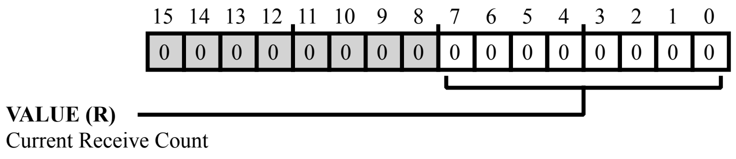


Figure 18-14: I2C_MCRXCNT Register Diagram

Table 18-13: I2C_MCRXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	VALUE	Current Receive Count. This register gives the total number of bytes received. If 256 bytes are requested, this register will read 0 when the transaction is completed.

Master Receive Data

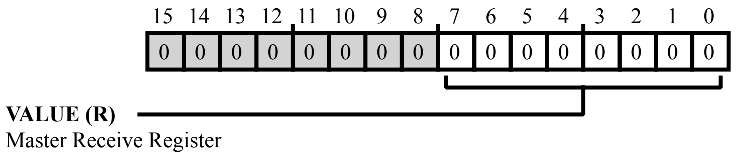


Figure 18-15: I2C_MRX Register Diagram

Table 18-14: I2C_MRX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	VALUE	Master Receive Register. This register allows access to the receive data FIFO. The FIFO can hold 2 bytes.

Master Receive Data Count

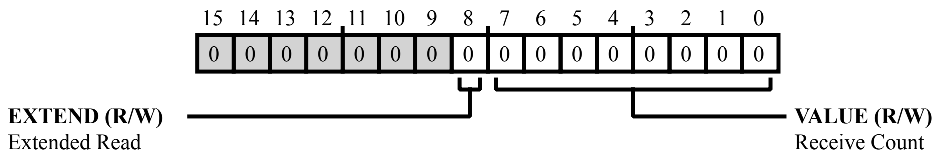


Figure 18-16: I2C_MRXCNT Register Diagram

Table 18-15: I2C_MRXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W)	EXTEND	Extended Read. Use this bit if greater than 256 bytes are required for a read. For example, to receive 412 bytes write 1 to `I2C_MRXCNT.EXTEND`. Wait for the first byte to be received, then check the `I2C_MCRXCNT` register for every byte received. When `I2C_MRXCNT.VALUE` returns to 0, 256 bytes have been received. Then write 0x09C to the `I2C_MRXCNT` register.
7:0 (R/W)	VALUE	Receive Count. Program the number of bytes required minus one to this register. If just 1 byte is required write 0 to this register. If greater than 256 bytes are required then use `I2C_MRXCNT.EXTEND`.

Master Status

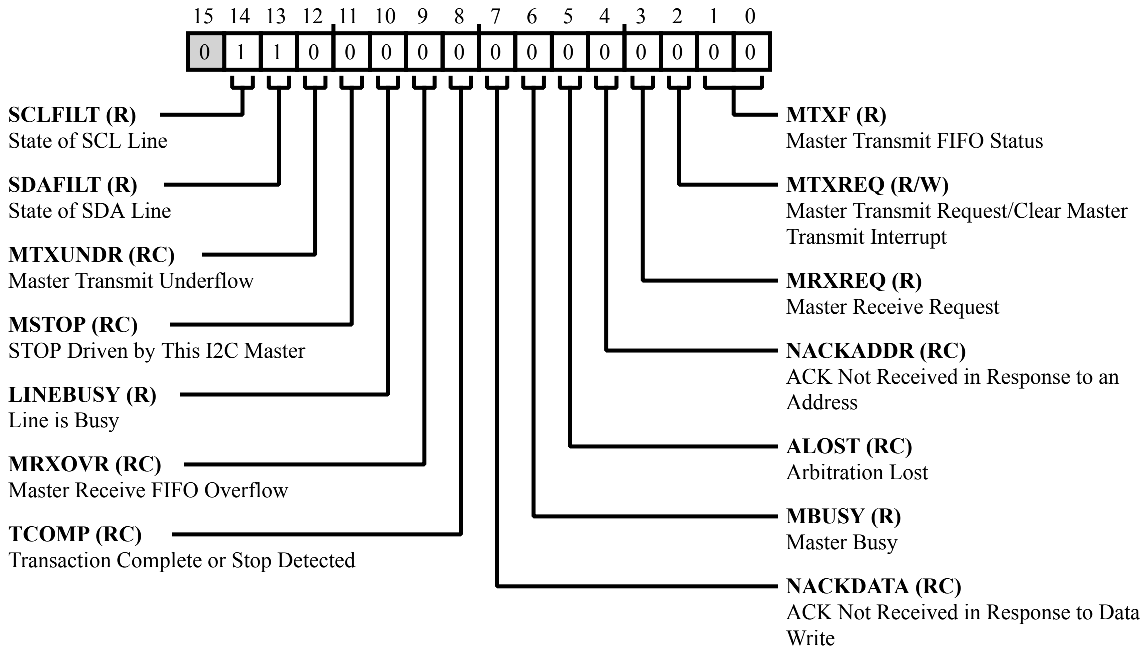


Figure 18-17: I2C_MSTAT Register Diagram

Table 18-16: I2C_MSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/NW)	SCLFILT	State of SCL Line. This bit is the output of the glitch-filter on SCL. SCL is always pulled high when un-driven.
13 (R/NW)	SDAFILT	State of SDA Line. This bit is the output of the glitch-filter on SDA. SDA is always pulled high when un-driven.
12 (RC/NW)	MTXUNDR	Master Transmit Underflow. Asserts when the I2C master ends the transaction due to Tx FIFO empty condition. This bit is asserted only when the `I2C_MCTL.IENMTX` bit is set.
11 (RC/NW)	MSTOP	STOP Driven by This I2C Master. Asserts when this I2C master drives a STOP condition on the I2C bus. This bit, when asserted, can indicate a Transaction completion, Tx-underflow, Rx-overflow or a NACK by the slave. This is different from the `I2C_MSTAT.TCOMP` as this bit is not asserted when the STOP condition occurs due to any other I2C master. No interrupt is generated for the assertion of this bit. However, if `I2C_MCTL.IENCMP` = 1, every STOP condition will generate an interrupt and this bit can be read. When this bit is read, it will clear status.

Table 18-16: I2C_MSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/NW)	LINEBUSY	Line is Busy. Asserts when a START is detected on the I2C bus. De-asserts when a STOP is detected on the I2C bus.
9 (RC/NW)	MRXOVR	Master Receive FIFO Overflow. Asserts when a byte is written to the receive FIFO when the FIFO is already full. When the bit is read it will clear status.
8 (RC/NW)	TCOMP	Transaction Complete or Stop Detected. Transaction complete. This bit will assert when a STOP condition is detected on the I2C bus. If `I2C_MCTL.IENCMP` = 1, an interrupt will be generated when this bit asserts. This bit will only assert if the master is enabled (`I2C_MCTL.MASEN` = 1). This bit should be used to determine when it is safe to disable the master. It can also be used to wait for another master transaction to complete on the I2C bus when this master looses arbitration. When this bit is read it will clear status. This bit can drive an interrupt.
7 (RC/NW)	NACKDATA	ACK Not Received in Response to Data Write. This bit will assert when an ACK is not received in response to a data write transfer. If `I2C_MCTL.IENACK` = 1, an interrupt will be generated when this bit asserts. This bit can drive an interrupt. This bit is cleared on a read of the `I2C_MSTAT` register.
6 (R/NW)	MBUSY	Master Busy. This bit indicates that the master state machine is servicing a transaction. It will be clear if the state machine is idle or another device has control of the I2C bus.
5 (RC/NW)	ALOST	Arbitration Lost. This bit will assert if the master loses arbitration. If `I2C_MCTL.IENALOST` = 1, an interrupt will be generated when this bit asserts. This bit is cleared on a read of the `I2C_MSTAT` register. This bit can drive an interrupt.
4 (RC/NW)	NACKADDR	ACK Not Received in Response to an Address. This bit will assert if an ACK is not received in response to an address. If `I2C_MCTL.IENACK` = 1, an interrupt will be generated when this bit asserts. This bit is cleared on a read of the `I2C_MSTAT` register. This bit can drive an interrupt.
3 (R/NW)	MRXREQ	Master Receive Request. This bit will assert when there is data in the receive FIFO. If `I2C_MCTL.IENMRX` = 1, an interrupt will be generated when this bit asserts. This bit can drive an interrupt.
2 (R/W)	MTXREQ	Master Transmit Request/Clear Master Transmit Interrupt. When read is master transmit request; when write is clear master transmit interrupt (mas_txirq_clr) bit. This bit will assert when the direction bit is 0 and transmit FIFO is not full. If `I2C_MCTL.IENMTX` = 1, an interrupt will be generated when this bit asserts. When this bit is written to 1, master transmit interrupt and clock stretching will be cleared.

Table 18-16: I2C_MSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/NW)	MTXF	Master Transmit FIFO Status. These 2 bits show the master transmit FIFO status.	
		0	FIFO Empty.
		2	1 byte in FIFO.
		3	FIFO Full.

Master Transmit Data

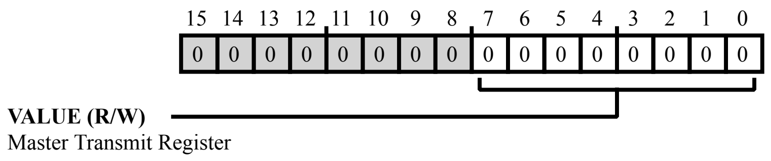


Figure 18-18: I2C_MTX Register Diagram

Table 18-17: I2C_MTX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Master Transmit Register. For test and debug purposes, when read, this register returns the byte that is currently being transmitted by the master. That is a byte written to the transmit register can be read back some time later when that byte is being transmitted on the line. This register allows access to the transmit data FIFO. The FIFO can hold 2 bytes.

Slave Control

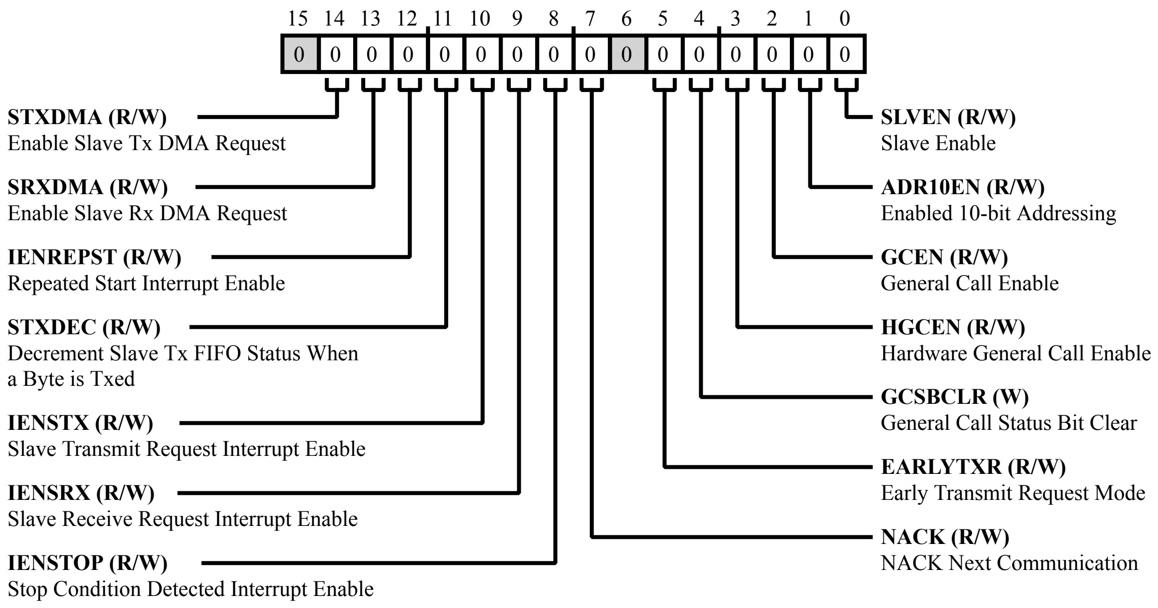

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
STXDMA (R/W)
Enable Slave Tx DMA Request
SRXDMA (R/W)
Enable Slave Rx DMA Request
IENREPST (R/W)
Repeated Start Interrupt Enable
STXDEC (R/W)
Decrement Slave Tx FIFO Status When a Byte is Txed
IENSTX (R/W)
Slave Transmit Request Interrupt Enable
IENSRX (R/W)
Slave Receive Request Interrupt Enable
IENSTOP (R/W)
Stop Condition Detected Interrupt Enable
SLVEN (R/W)
Slave Enable
ADR10EN (R/W)
Enabled 10-bit Addressing
GCEN (R/W)
General Call Enable
HGCEN (R/W)
Hardware General Call Enable
GCSBCLR (W)
General Call Status Bit Clear
EARLYTXR (R/W)
Early Transmit Request Mode
NACK (R/W)
NACK Next Communication

Figure 18-19: I2C_SCTL Register Diagram

Table 18-18: I2C_SCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W)	STXDMA	Enable Slave Tx DMA Request. Set to 1 by user code to enable I2C slave DMA Rx requests. Cleared by user code to disable DMA mode.
13 (R/W)	SRXDMA	Enable Slave Rx DMA Request. Set to 1 by user code to enable I2C slave DMA Rx requests. Cleared by user code to disable DMA mode.
12 (R/W)	IENREPST	Repeated Start Interrupt Enable. If 1 an interrupt will be generated when the `I2C_SSTAT.REPSTART` status bit asserts. If 0 an interrupt will not be generated when the `I2C_SSTAT.REPSTART` status bit asserts.
11 (R/W)	STXDEC	Decrement Slave Tx FIFO Status When a Byte is Txed. Decrement Slave Tx FIFO status when a byte has been transmitted. If set to 1, the transmit FIFO status is decremented when a byte has been transmitted. If set to 0, the transmit FIFO status is decremented when the byte is unloaded from the FIFO into a shadow register at the start of byte transmission.
10 (R/W)	IENSTX	Slave Transmit Request Interrupt Enable.

Table 18-18: I2C_SCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W)	IENSRX	Slave Receive Request Interrupt Enable.
8 (R/W)	IENSTOP	Stop Condition Detected Interrupt Enable.
7 (R/W)	NACK	NACK Next Communication. If this bit is set the next communication will be NACK 'ed. This could be used for example if during a 24xx style access, an attempt was made to write to a 'read only' or nonexisting location in system memory. That is the indirect address in a 24xx style write pointed to an unwritable memory location.
5 (R/W)	EARLYTXR	Early Transmit Request Mode. Setting this bit enables a transmit request just after the positive edge of the direction bit SCL clock pulse.
4 (RX/W)	GCSBCLR	General Call Status Bit Clear. The `I2C_SSTAT.GCINT` and `I2C_SSTAT.GCID` bits are cleared when a '1' is written to this bit. The `I2C_SSTAT.GCINT` and `I2C_SSTAT.GCID` bits are not reset by anything other than a write to this bit or a full reset.
3 (R/W)	HGCEN	Hardware General Call Enable. When this bit and the `I2C_SCTL.GCEN` bit are set the device after receiving a general call, address 00h and a data byte checks the contents of the `I2C_ALT` against the receive shift register. If they match the device has received a 'hardware general call'. This is used if a device needs urgent attention from a master device without knowing which master it needs to turn to. This is a call "to whom it may concern". The device that requires attention embeds its own address into the message. The LSB of the `I2C_ALT` register should always be written to a 1, as per I2C January 2000 specification.
2 (R/W)	GCEN	General Call Enable. This bit enables the I2C slave to ACK an I2C general call, address 0x00 (Write).
1 (R/W)	ADR10EN	Enabled 10-bit Addressing. If this bit is clear, the slave can support four slave addresses, programmed in `I2C_ID0` to `I2C_ID0`. When this bit is set, 10 bit addressing is enabled. One 10 bit address is supported by the slave and is stored in `I2C_ID0` and `I2C_ID1`, where `I2C_ID0` contains the first byte of the address and the upper 5 bits must be programmed to 11110. `I2C_ID2` and `I2C_ID3` can be programmed with 7 bit addresses at the same time.
0 (R/W)	SLVEN	Slave Enable. When '1' the slave is enabled. When '0' all slave state machine flops are held in reset and the slave is disabled. Note that APB writable register bits are not reset.

Shared Control

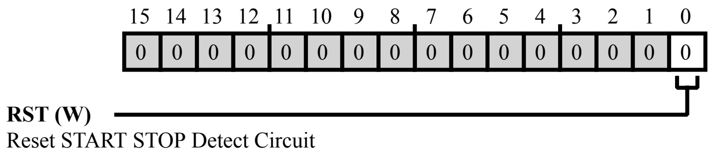


Figure 18-20: I2C_SHCTL Register Diagram

Table 18-19: I2C_SHCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (RX/W)	RST	Reset START STOP Detect Circuit. Writing a 1 to this bit will reset the SCL and SDA synchronisers, the START and STOP detect circuit and the LINEBUSY detect circuit. These circuits are not reset when both the master and slave are disabled as LINEBUSY needs to assert even when the master is not enabled. It should only be necessary to reset these circuits after a power on reset in case SCL/SDA do not power up cleanly. Reading this bit will always read back '0'.

Slave Receive

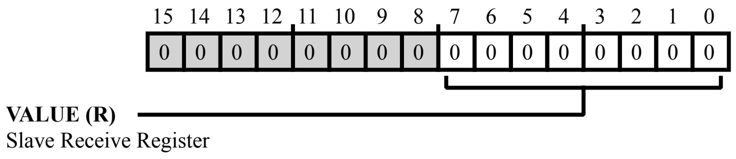


Figure 18-21: I2C_SRX Register Diagram

Table 18-20: I2C_SRX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	VALUE	Slave Receive Register.

Slave I2C Status/Error/IRQ

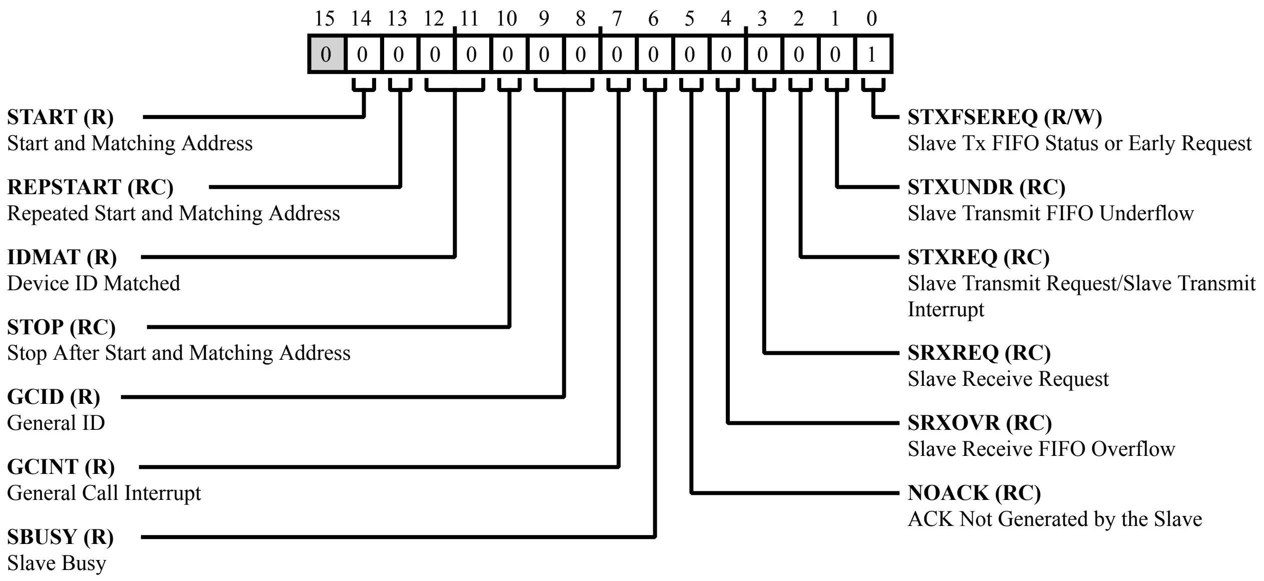


Figure 18-22: I2C_SSTAT Register Diagram

Table 18-21: I2C_SSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/NW)	START	Start and Matching Address. This bit is asserted if a start is detected on SCL/SDA and the device address matched, or a general call(GC - 0000_0000) code is received and GC is enabled, or a High Speed (HS - 0000_1XXX) code is received, or a start byte (0000_0001) is received. It is cleared on receipt of either a stop or start condition.	
13 (RC/NW)	REPSTART	Repeated Start and Matching Address. This bit is asserted if `I2C_SSTAT.START` is already asserted and then a repeated start is detected. It is cleared when read or on receipt of a STOP condition. This bit can drive an interrupt.	
12:11 (R/NW)	IDMAT	Device ID Matched.	
		0	Received address matched ID register 0
		1	Received address matched ID register 1
		2	Received address matched ID register 2
		3	Received address matched ID register 3

Table 18-21: I2C_SSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (RC/NW)	STOP	Stop After Start and Matching Address. Gets set by hardware if the slave device received a STOP condition after a previous START condition and a matching address. Cleared by a read of the I2C_SSTAT register. If I2C_SCTL.IENSTOP in the slave control register is asserted, then the slave interrupt request will assert when this bit is set. This bit can drive an interrupt.	
9:8 (R/NW)	GCID	General ID. I2C_SSTAT.GCID is cleared when the I2C_SCTL.GCSBCLR is written to 1. These status bits will not be cleared by a General call reset.	
		0	No general call
		1	General call reset and program address
		2	General call program address
		3	General call matching alternative ID
7 (R/NW)	GCINT	General Call Interrupt. This bit always drives an interrupt. The bit is asserted if the slave device receives a general call of any type. To clear, write 1 to the I2C_SCTL.GCSBCLR in the slave control register. If it was a general call reset, all registers will be at their default values. If it was a hardware general call, the Rx FIFO holds the second byte of the general call, and this can be compared with the I2C_ALT register.	
6 (R/NW)	SBUSY	Slave Busy. Set by hardware if the slave device receives a I2C START condition. Cleared by hardware when the address does not match an ID register, the slave device receives a I2C STOP condition, or if a repeated start address does not match.	
5 (RC/NW)	NOACK	ACK Not Generated by the Slave. When asserted, it indicates that the slave responded to its device address with a NOACK. It is asserted if there was no data to transmit and sequence was a slave read or if the I2C_SCTL.NACK bit was set and the device was addressed. This bit is cleared on a read of the I2C_SSTAT register.	
4 (RC/NW)	SRXOVR	Slave Receive FIFO Overflow. Asserts when a byte is written to the slave receive FIFO when the FIFO is already full.	
3 (RC/NW)	SRXREQ	Slave Receive Request. I2C_SSTAT.SRXREQ asserts whenever the slave receive FIFO is not empty. Read or flush the slave receive FIFO to clear this bit. This bit will assert on the falling edge of the SCL clock pulse that clocks in the last data bit of a byte. This bit can drive an interrupt.	

Table 18-21: I2C_SSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (RC/NW)	STXREQ	Slave Transmit Request/Slave Transmit Interrupt. When read is slave transmit request; when write is clear slave transmit interrupt (slv_txint_clr) bit. This bit is read only. If `I2C_SCTL.EARLYTXR` = 0, this bit is set when the direction bit for a transfer is received high. There after, as long as the transmit FIFO is not full this bit will remain asserted. Initially it is asserted on the negative edge of the SCL pulse that clocks in the direction bit (if the device address matched also). If `I2C_SCTL.EARLYTXR` = 1, this bit is set when the direction bit for a transfer is received high. There after, as long as the transmit FIFO is not full this bit will remain asserted. Initially it is asserted after the positive edge of the SCL pulse that clocks in the direction bit (if the device address matched also). This bit is cleared on a read of the `I2C_SSTAT` register. slv_txint_clr, write only. When this bit is 1, slave transmit interrupt and clock stretching will be cleared. And this bit will be cleared, when STOP or (RE)START is received (transfer is done); TX FIFO is written (software decided it now has more data to send); TX FIFO is empty and an ACK is received instead of a NACK.
1 (RC/NW)	STXUNDR	Slave Transmit FIFO Underflow. Is set if a master requests data from the device, and the Tx FIFO is empty for the rising edge of SCL.
0 (R/W)	STXFSEREQ	Slave Tx FIFO Status or Early Request. If `I2C_SCTL.EARLYTXR` = 0, this bit is asserted whenever the slave Tx FIFO is empty. If `I2C_SCTL.EARLYTXR` = 1, this bit is set when the direction bit for a transfer is received high. It asserts on the positive edge of the SCL clock pulse that clocks in the direction bit (if the device address matched also). It only asserts once for a transfer. It is cleared when read if `I2C_SCTL.EARLYTXR` is asserted.

Master and Slave FIFO Status

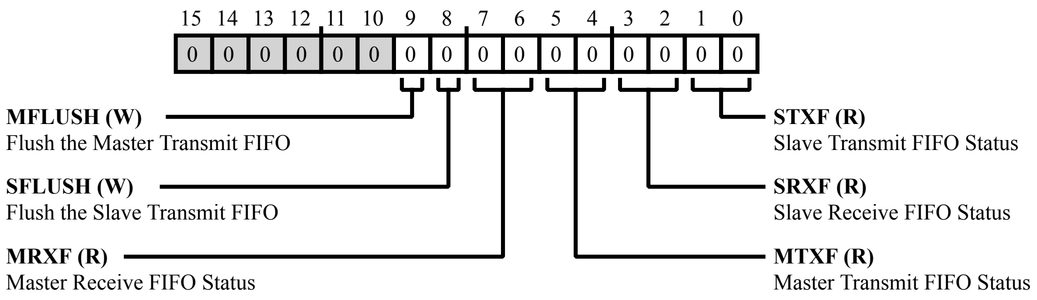

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MFLUSH (W)
Flush the Master Transmit FIFO
SFLUSH (W)
Flush the Slave Transmit FIFO
MRXF (R)
Master Receive FIFO Status
STXF (R)
Slave Transmit FIFO Status
SRXF (R)
Slave Receive FIFO Status
MTXF (R)
Master Transmit FIFO Status

Figure 18-23: I2C_STAT Register Diagram

Table 18-22: I2C_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (RX/W)	MFLUSH	Flush the Master Transmit FIFO. Writing a '1' to this bit flushes the master transmit FIFO. The master transmit FIFO will have to flushed if arbitration is lost or a slave responds with a NACK.	
8 (RX/W)	SFLUSH	Flush the Slave Transmit FIFO. Writing a '1' to this bit flushes the slave transmit FIFO.	
7:6 (R/NW)	MRXF	Master Receive FIFO Status. The status is a count of the number of bytes in a FIFO.	
		0	FIFO empty
		1	1 bytes in the FIFO
		2	2 bytes in the FIFO
		3	Reserved
5:4 (R/NW)	MTXF	Master Transmit FIFO Status. The status is a count of the number of bytes in a FIFO.	
		0	FIFO empty
		1	1 bytes in the FIFO
		2	2 bytes in the FIFO
		3	Reserved

Table 18-22: I2C_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:2 (R/NW)	SRXF	Slave Receive FIFO Status. The status is a count of the number of bytes in a FIFO.	
		0	FIFO empty
		1	1 bytes in the FIFO
		2	2 bytes in the FIFO
		3	Reserved
1:0 (R/NW)	STXF	Slave Transmit FIFO Status. The status is a count of the number of bytes in a FIFO.	
		0	FIFO empty
		1	1 bytes in the FIFO
		2	2 bytes in the FIFO
		3	Reserved

Slave Transmit

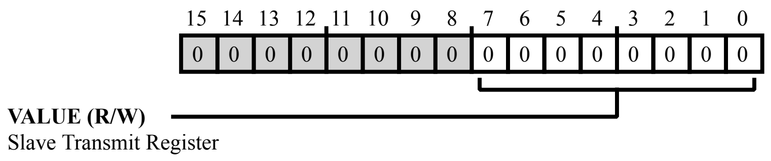


Figure 18-24: I2C_STX Register Diagram

Table 18-23: I2C_STX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Slave Transmit Register.

Timing Control Register

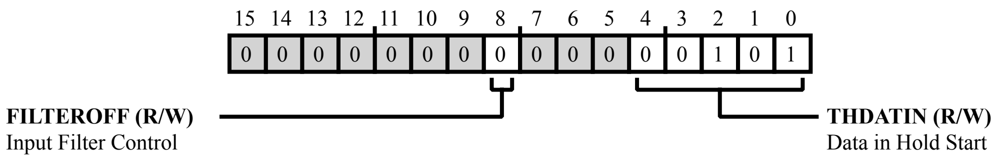


Figure 18-25: I2C_TCTL Register Diagram

Table 18-24: I2C_TCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	FILTEROFF	Input Filter Control. Input Filter Control. It may be desirable to disable the digital filter if PCLK rate becomes < 16MHz.	
		0	Digital filter is enabled and is equal to 1 PCLK
		1	Digital filter is disabled. Filtering in the pad is unaffected
4:0 (R/W)	THDATIN	Data in Hold Start. `I2C_TCTL.THDATIN` determines the hold time requirement that must be met before a start or stop condition is recognized. The hold time observed between SDA and SCL fall must exceed the programmed value to be recognized as a valid Start. `I2C_TCTL.THDATIN` = (tHD;STA) / (Clock Period). For example, at 16MHz PCLK (62.5ns) and setting a minimum tHD;STA limit of 300ns. (300ns)/(62.5ns) = 5.	

19 Beeper Driver (BEEP)

The beeper driver module generates a differential square wave of programmable frequency. It is used to drive an external piezoelectric sound component whose two terminals connect to the differential square wave output. A standard APB interface connects this module to the system bus.

Beeper Features

The beeper driver present in the MCU includes the following features:

- A module that can deliver frequencies from 8 kHz to ~0.25 kHz.
- It operates on a fixed, independent 32 kHz clock source that is unaffected by changes in system clocks.
- It allows a programmable tone duration from 4 ms to 1.02 s in 4 ms increments.
- Single-tone (pulse) and multitone (sequence) modes provide versatile playback options.
- In sequence mode, the beeper may be programmed to play any number of tone pairs from 1 to 254 (2 to 508 tones) or be programmed to play forever (until stopped by the user).
- Interrupts are available to indicate the start or end of any beep, the end of a sequence, or that the sequence is nearing completion.

Beeper Functional Description

This section provides information on the function of the beeper driver used by the ADuCM302x MCU.

Beeper Block Diagram

The figure shows the block diagram of the beeper driver used by the ADuCM302x MCU.

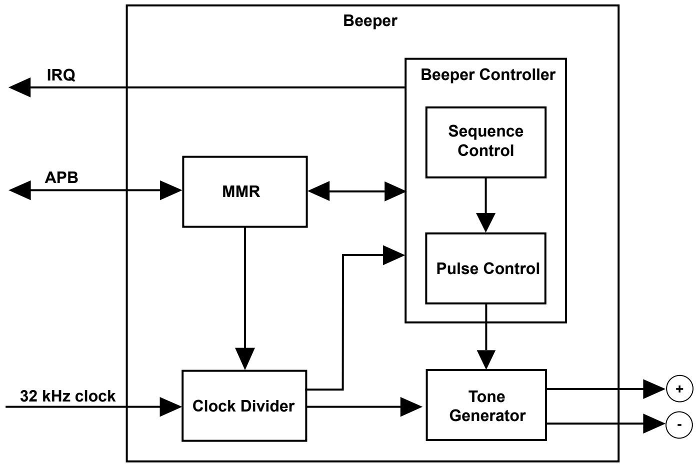


Figure 19-1: Beeper Driver

The clock divider divides the 32 kHz input clock down to frequencies between 8 kHz and ~0.25 kHz for driving the off-chip piezoelectric component. The beeper controller block controls the tone generator, enabling and disabling its operation, and selecting the appropriate frequency output from the clock divider.

The beeper controller also tracks the durations and repeating sequences of tones. The beeper module can interrupt the microcontroller with one interrupt line. Its off-chip output lines are held low when the module is disabled (no voltage across the piezoelectric component), and drive a differential square wave while tones are being played.

Beeper Operating Modes

The basic operation of the beeper driver consists of writing tone data into one or both tone registers, followed by enabling the beeper by setting the `BEEP_CFG.EN` bit. Once enabled, the module plays one or more tones depending on the operating mode. During playback, the module outputs a differential square wave (VSS to VCC) with frequency and duration based on the `BEEP_TONEA` or `BEEP_TONEB` register settings.

Beeper has two modes of operation. If the `BEEP_CFG.SEQREPEAT` bit field has a non-zero value when the beeper is enabled, the beeper operates in sequence mode. If the beeper is enabled with `BEEP_CFG.SEQREPEAT` set to 0x0, the beeper operates in pulse mode.

Pulse mode operation completes after playing exactly one tone. Sequence mode operation plays a configurable number of tones before completion. While operating in pulse or sequence mode, the `BEEP_STAT.BUSY` bit is asserted. This bit is cleared on completion of the operation.

A currently playing tone may be prematurely terminated by writing 0x0 to the `BEEP_TONEA.DUR` and `BEEP_TONEB.DUR` bits. When terminated, the `BEEP_STAT.AENDED` and `BEEP_STAT.BENDED` bits are set, and generates an interrupt if configured using the `BEEP_CFG.AENDIRQ` and `BEEP_CFG.BENDIRQ` bits. Alternatively, all playback may be immediately terminated by disabling the beeper (clearing the `BEEP_CFG.EN` bit). Disabling the beeper prevents any events from occurring and, therefore, prevents an interrupt from being generated.

An interrupt is generated when an event is requested. The BEEP_STAT register provides feedback on the six tracked events that has occurred since the register was last cleared. Any of these six events may be used for generating interrupts by selecting them in the BEEP_CFG register.

The operating mode of the beeper is set when the beeper is enabled and cannot be changed until the beeper is again disabled. The beeper may be disabled by writing 0x0 to the BEEP_CFG.EN bit or by waiting for the beeper to disable itself after the current pulse or sequence completes.

Pulse Mode

In pulse mode, the beeper plays back exactly one tone. Pulse mode operation begins by enabling the beeper with a zero value in the BEEP_CFG.SEQREPEAT bit field. Once enabled, the beeper plays back a single tone as described in the BEEP_TONEA register. Once the playback of this tone has been completed, the BEEP_STAT.BUSY bit is cleared.

Interrupts are generated as requested in the BEEP_CFG register. Note that the only events that may occur in this mode (and used to generate meaningful interrupts) are for the start (BEEP_CFG.ASTARTIRQ) and end (BEEP_CFG.AENDIRQ) of BEEP_TONEA play back.

Sequence Mode

In sequence mode, the beeper plays back a programmable number of tones. Sequence mode operation begins by enabling the beeper with a non-zero value in the BEEP_CFG.SEQREPEAT bit field. Once enabled, the beeper plays back a series of two-tone sequences as described in the BEEP_TONEA and BEEP_TONEB registers. Each two-tone iteration starts by playing BEEP_TONEA and ends by playing BEEP_TONEB.

The sequence is repeated as configured in the BEEP_CFG.SEQREPEAT bit field. 0xFF is a special case used to run the sequencer until terminated by the user code. Once all iterations have been completed, the BEEP_STAT.BUSY bit is cleared.

The number of remaining sequence iterations may be read from the BEEP_STAT.SEQREMAIN bit field. When running an infinite sequence, the BEEP_STAT.SEQREMAIN bit field always returns 0xFF. Writing to the BEEP_CFG.SEQREPEAT while the beeper is running in sequence mode, restarts the iteration counter to that value and immediately updates the value of the BEEP_STAT.SEQREMAIN bit field.

Sequence mode may be prematurely terminated by writing 0x0 to the BEEP_CFG.SEQREPEAT bit field. Setting the BEEP_CFG.SEQREPEAT to 0x0 causes the beeper to terminate playback and disable itself after the completion of the current two-tone sequence. When terminated, the BEEP_STAT.SEQENDED bit is set, and generates an interrupt if configured using the BEEP_CFG.SEQATENDIRQ bit. Alternatively, all playback may be immediately terminated by disabling the beeper (clearing the BEEP_CFG.EN bit); disabling the beeper prevents any events from occurring and, therefore, prevents an interrupt from being generated.

Interrupts are generated in sequence mode as requested in the BEEP_CFG register. All beeper events are valid for interrupt generation in this mode.

Tones

The frequency and duration of tones played by the beeper depend on the values stored in the `BEEP_TONEA` and `BEEP_TONEB` registers. Each of these registers describes a single independent tone. Pulse mode plays tones only from register `BEEP_TONEA`, while sequence mode plays tones from both tone registers.

The tone registers are broken into three fields: a duration field (`BEEP_TONEA.DUR`, `BEEP_TONEB.DUR`), a frequency field (`BEEP_TONEA.FREQ`, `BEEP_TONEB.FREQ`), and a disable field (`BEEP_TONEA.DIS`, `BEEP_TONEB.DIS`).

Tone registers may be written at any time. Writing 0x0 to the duration field (`BEEP_TONEA.DUR`, `BEEP_TONEB.DUR`) or setting/clearing the disable bit (`BEEP_TONEA.DIS`, `BEEP_TONEB.DIS`) for a currently playing tone will take effect immediately. All other modifications to the `BEEP_TONEA` and `BEEP_TONEB` registers take effect only the next time the tone is played; for most use cases, this allows the next tone data to be written to while the current tone is being played.

The duration field is used to program how long a tone will play when selected for playback. Durations are measured in units of 4 ms. Any value from 0 through 255 may be stored in the tone register. A duration of 0x0 will cause the tone playback to end immediately. A value of 255 (0xFF) is a special case value used to program a tone for infinite duration; once started, the tone will play continuously until user code terminates the tone (writes 0x0 to `BEEP_TONEA.DUR`, `BEEP_TONEB.DUR`) or disables the beeper (clears the `BEEP_CFG.EN` bit).

The frequency field is used to program the relative frequency of the tone with respect to source clock (32.768 kHz). Writing values 0, 1, 2, or 3 into the frequency field all have the same effect: the output will not oscillate during playback (also known as rest tone). Any value from 4 through 127 (0x7F) may be used to divide the source clock down to the desired tone frequency. This provides a playback range from 8 kHz down to ~0.25 kHz.

The disable field is used to provide further control over the output pins of the beeper during playback. When playing a tone with the `BEEP_TONEA.DIS`, `BEEP_TONEB.DIS` bit set, the output pins behave as though the beeper were disabled; both pins rest at logic 0 with no DC potential between them and no oscillations. This feature is intended for use when long periods of silence exist in a programmed sequence. It may be used to prevent damage to the piezoelectric component during these periods of silence.

Clocking and Power

The frequency and duration timers of the beeper driver are based on a 32.768 kHz input clock. The clock source is configured system wide, selecting between an external crystal or an internal oscillator by writing to the appropriate clock control module register (`CLKG_OSC_CTL.LFCLKMUX`). The selected clock only provides a stable reference for the timers; the internal logic of the beeper is clocked by the peripheral clock (PCLK) of the system. For this reason, the beeper cannot run while the system is in a low power mode that gates the peripheral clock (PCLK).

Timer start events are synchronized with the 32 kHz clock. When enabling the beeper, its output may therefore be delayed by as much as one 32 kHz clock period (30,520 ns). For a 4 MHz PCLK, this results in up to 122 PCLK periods between starting the beeper and actually producing audio output. Any changes to the `BEEP_TONEA` and

BEEP_TONEB registers during this time will affect the pending audio. User code must ensure that the tone is playing before modifying the tone registers, either by polling the BEEP_STAT.ASTARTED and BEEP_STAT.BSTARTED bits or by setting an interrupt for the same.

User code must disable the beeper prior to entering a low power state where the peripheral clock would be gated. Damage to the piezoelectric component may occur if the system enters a low power mode while the beeper is playing a tone due to constant, long term DC potential across the terminals of the piezoelectric component.

Power-down Considerations

There is the possibility of irreversible damage to the external piezo beeper device if the beeper is enabled and driving the tone outputs under the following conditions:

- Entering hibernate power mode
- Entering shutdown power mode
- Turning off PCLK

The pins do not disengage automatically from the beeper module in these modes.

Beeper Interrupts and Events

There are six tracked events that may occur while the beeper is running: Tone A may start or end, Tone B may start or end, and the sequencer may end or be one step away from ending. These six events are always monitored, and when they occur, a sticky bit is set in the BEEP_STAT register to be read by the user at some future time. These bits remain set until cleared by user code.

Any of these six events may also be used to generate an interrupt. Selecting which events will generate interrupts is done by setting the respective bits in the BEEP_CFG register. When an event triggers an interrupt, user code must clear the event bit or disable the interrupt selection bit when servicing the interrupt.

When servicing a beeper interrupt, user code should check and eventually clear all event bits in the BEEP_STAT register; multiple events may have triggered the interrupt (if enabled). Writing 0xFF to BEEP_STAT clears all events.

All tracked events are independently selectable for interrupt generation.

Beeper Programming Model

The following sections provide general programming guidelines and procedures.

Timing Diagram

The figure shows the behavior of the beeper when programmed to generate a tone.

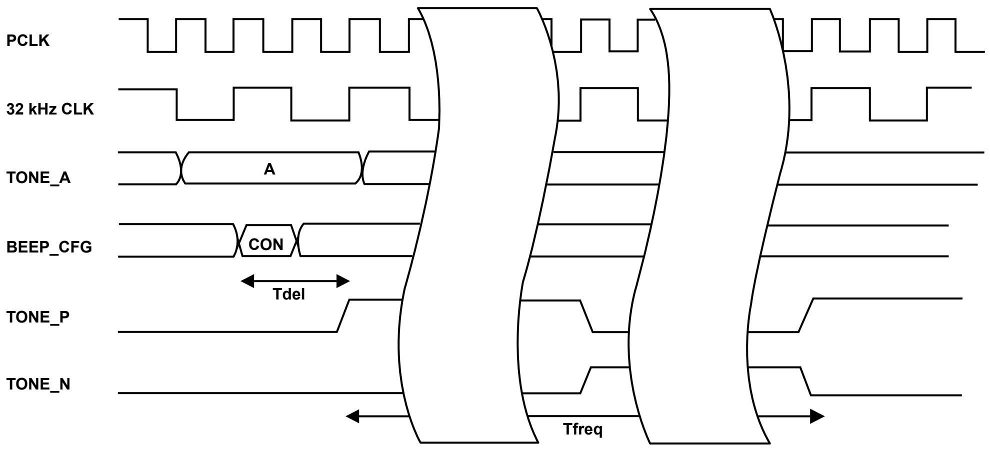


Figure 19-2: Timing Diagram

In this example, the tone is first written to the `BEEP_TONEA` register, followed by enabling the beeper by writing to the `BEEP_CFG` register. Tdel illustrates the maximum one 32 kHz clock period delay between enabling the beeper and the actual start of playback. Tfreq illustrates the output period of the two differential pins; for a 1 kHz (1024 Hz) tone, Tfreq = 0.9766 ms.

Programming Guidelines

Programming Examples

The beeper driver is programmed for a single 1 kHz beep of 1 second length with an interrupt when the tone is done playing.

1. Set Tone A duration to 1 s. `BEEP_TONEA.DUR` = 0xFA // (250 × 4 ms).
2. Set Tone A frequency to 1 kHz. `BEEP_TONEA.FREQ` = 0x20 // (32 kHz/32).
3. Set interrupt at the end of Tone A. `BEEP_CFG.AENDIRQ` = 0x1.
4. Set the sequence repeat to zero. `BEEP_CFG.SEQREPEAT` = 0x0.
5. Enable the beeper. `BEEP_CFG.EN` = 0x1.

The register access sequence is `BEEP_TONEA` = 0x20FA and `BEEP_CFG` = 0x0900.

The beeper driver is programmed for a melody of 32 two-tone sequences of the note G# for 500 ms and F# for 1000 ms with an interrupt at the end of the melody.

1. Set Tone A duration to 500 ms. `BEEP_TONEA.DUR` = 0x7D // (125 × 4 ms).
2. Set Tone A frequency to G#. `BEEP_TONEA.FREQ` = 0x27 // (32 kHz/39 = 840 Hz).
3. Set Tone B duration to 1000 ms. `BEEP_TONEB.DUR` = 0xFA // (250 × 4 ms).
4. Set Tone B frequency to F#. `BEEP_TONEB.FREQ` = 0x2C //(32 kHz/44 = 745 Hz).

5. Set interrupt at end of sequence. `BEEP_CFG.SEQATENDIRQ` = 0x1.
6. Set sequence repeat to 32. `BEEP_CFG.SEQREPEAT` = 0x20.
7. Enable the beeper. `BEEP_CFG.EN` = 0x1.

The register access sequence is `BEEP_TONEA` = 0x277D, `BEEP_TONEB` = 0x2CFA, and `BEEP_CFG` = 0x8120.

ADuCM302x BEEP Register Descriptions

Beeper Driver (BEEP) contains the following registers.

Table 19-1: ADuCM302x BEEP Register List

Name	Description
BEEP_CFG	Beeper Configuration
BEEP_STAT	Beeper Status
BEEP_TONEA	Tone A Data
BEEP_TONEB	Tone B Data

Beeper Configuration

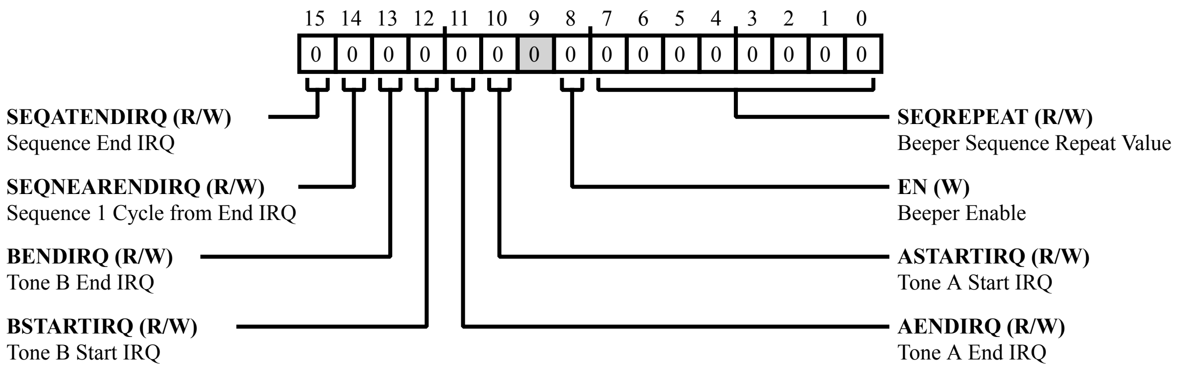


Figure 19-3: BEEP_CFG Register Diagram

Table 19-2: BEEP_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	SEQATENDIRQ	Sequence End IRQ. Set this bit to request an interrupt on the event the sequencer has stopped playing.	
		0	IRQ not generated at end of Sequence
		1	IRQ generated at end of Sequence
14 (R/W)	SEQNEARENDIRQ	Sequence 1 Cycle from End IRQ. Set this bit to request an interrupt on the event the sequencer is currently running and has just one repetition remaining before completion.	
		0	IRQ not generated 1 tone pair before Sequence ends
		1	IRQ generated 1 tone pair before sequence ends
13 (R/W)	BENDIRQ	Tone B End IRQ. Set this bit to request an interrupt on the event BEEP_TONEB stops playing	
		0	IRQ not generated at end of Tone B
		1	IRQ generated at end of Tone B
12 (R/W)	BSTARTIRQ	Tone B Start IRQ. Set this bit to request an interrupt on the event BEEP_TONEB starts playing	
		0	IRQ not generated at start of Tone B
		1	IRQ generated at start of Tone B

Table 19-2: BEEP_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	AENDIRQ	Tone A End IRQ. Set this bit to request an interrupt on the event Tone A stops playing.	
		0	IRQ not generated at end of Tone A
		1	IRQ generated at end of Tone A
10 (R/W)	ASTARTIRQ	Tone A Start IRQ. Set this bit to request an interrupt on the event Tone A starts playing.	
		0	IRQ not generated at start of Tone A
		1	IRQ generated at start of Tone A
8 (RX/W)	EN	Beeper Enable. Write 0x1 to this bit to start playing Tone A. Plays a single tone if `BEEP_CFG.SEQREPEAT` = 0, else plays a series of two-tone sequences. Write 0x0 to this bit at any time to end audio playback. Reading this bit always returns 0x0, read `BEEP_STAT.BUSY` to determine if the beeper is currently enabled.	
		0	Disable module
		1	Enable module
7:0 (R/W)	SEQREPEAT	Beeper Sequence Repeat Value. When the beeper transitions to an enabled state, the value of this field selects whether the beeper runs in Pulse mode or Sequence mode. If 0x0, the beeper runs in Pulse mode and plays back only one tone (Tone A) before being disabled. If a non-zero value is written to this field, the beeper runs in Sequence mode and plays the two-tone sequence (Tone A and Tone B) the requested number of times before being disabled. Updates to this field have no affect while running in Pulse mode. Updates to this field take immediate affect when running in Sequence mode.	
		0	Enabling will start in Pulse mode.
		1-254	Enabling will start in Sequence mode and play a finite number of notes.
		255	Enabling will start in Sequence mode and play forever until stopped by user code.

Beeper Status

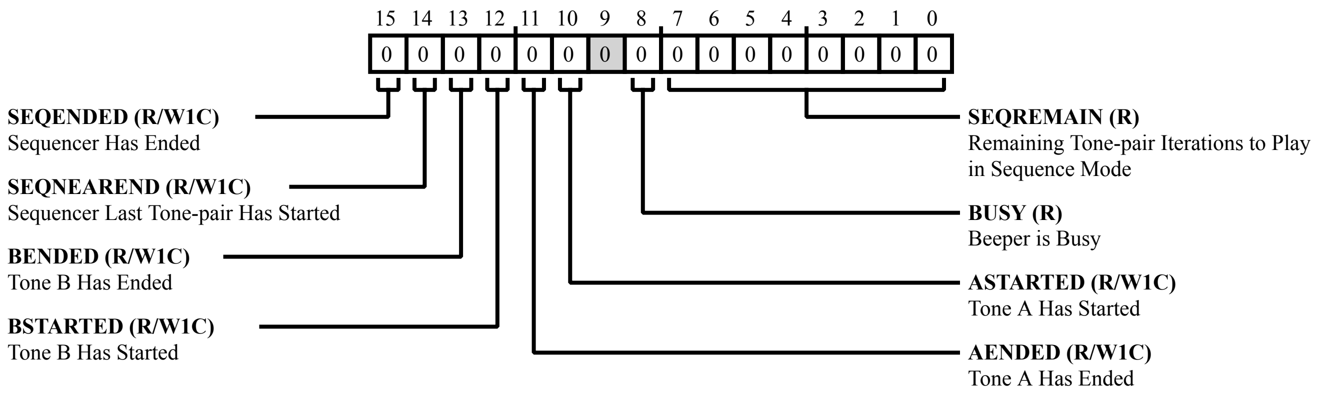


Figure 19-4: BEEP_STAT Register Diagram

Table 19-3: BEEP_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	SEQENDED	Sequencer Has Ended. This bit is asserted whenever the beeper stops running in Sequence mode. It is cleared only by user code writing 0x1 to this location.
14 (R/W1C)	SEQNEAREND	Sequencer Last Tone-pair Has Started. This bit is asserted whenever the beeper is running in Sequence mode and has only one iteration of sequences left to play. It is cleared only by user code writing 0x1 to this location.
13 (R/W1C)	BENDED	Tone B Has Ended. This bit is asserted whenever the beeper stops playing Tone B. It is cleared only by user code writing 0x1 to this location.
12 (R/W1C)	BSTARTED	Tone B Has Started. This bit is asserted whenever the beeper begins playing Tone B. It is cleared only by user code writing 0x1 to this location.
11 (R/W1C)	AENDED	Tone A Has Ended. This bit is asserted whenever the beeper stops playing Tone A. It is cleared only by user code writing 0x1 to this location.
10 (R/W1C)	ASTARTED	Tone A Has Started. This bit is asserted whenever the beeper begins playing Tone A. It is cleared only by user code writing 0x1 to this location.

Table 19-3: BEEP_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	BUSY	Beeper is Busy. This bit is asserted after enabling the beeper module by writing 0x1 to `BEEP_CFG.EN`. The bit is cleared when the module completes its operation or when user code writes 0x0 to `BEEP_CFG.EN`.	
		0	Beeper is currently disabled
		1	Beeper is currently enabled
7:0 (R/NW)	SEQREMAIN	Remaining Tone-pair Iterations to Play in Sequence Mode. After a sequence has ended, this field resets to `BEEP_CFG.SEQREPEAT`. This field is updated as the beeper plays back audio in Sequence mode. Each two-tone iteration starts by playing Tone A and ends by playing Tone B. This field is updated at the conclusion of Tone B playback and therefore during the last iteration will return 0x1 during Tone A and Tone B playback. When playing an infinite sequence this field always returns 0xFF.	

Tone A Data

Tone A is the first tone to play in Sequence Mode, and the only tone to play in Pulse Mode. Writing 0x0 to the `BEEP_TONEA.DUR` field while Tone A is playing will immediately terminate the tone. All other writes to `BEEP_TONEA` affect the next play back of the tone only and do not affect the currently playing tone.

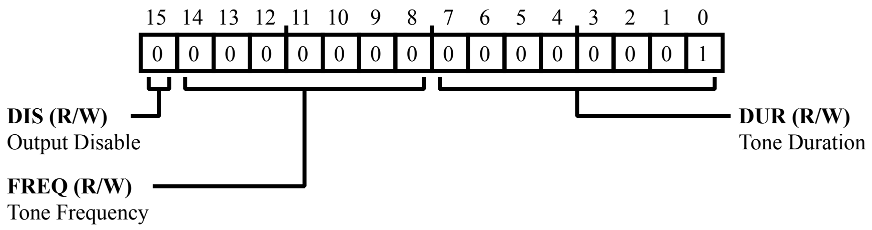


Figure 19-5: BEEP_TONEA Register Diagram

Table 19-4: BEEP_TONEA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	DIS	Output Disable. This bit is set to disable the beeper output while Tone A is playing. The beeper holds both of its two output pins at logic 0 when disabled. Writes to this bit take effect immediately if Tone A is currently playing.	
		0	Output enabled
		1	Output disabled, no DC potential across output pins
14:8 (R/W)	FREQ	Tone Frequency. This field defines the frequency for Tone A as integer divisions of the 32.768kHz clock. The values 0 through 3 are interpreted as 'rest-tone' and will result in no oscillations of the beeper output pins. All other values are directly used to divide the 32kHz input clock. Writes to this field immediately affect the output of Tone A if currently playing.	
		0-3	Rest tone (duration but no oscillation)
		4-127	Oscillation at 32kHz/(FREQ)
7:0 (R/W)	DUR	Tone Duration. This field defines the duration for Tone A in 4ms increments. Writing a value of 0x0 will immediately terminate Tone A if currently playing. Writing a value of 0xFF will cause Tone A to play forever once started, or until user code terminates the tone. Only a write of 0x0 will affect a currently playing tone, all other values written will be used only the next time Tone A is played.	
		0-254	Tone will play for (DUR)*4ms period
		255	Tone will play for infinite duration

Tone B Data

Tone B is the second tone to play in Sequence Mode, and is not played Pulse Mode. Writing 0x0 to the `BEEP_TONEB.DUR` field while Tone B is playing will immediately terminate the tone. All other writes to `BEEP_TONEB` affect the next play back of the tone only and do not affect the currently playing tone.

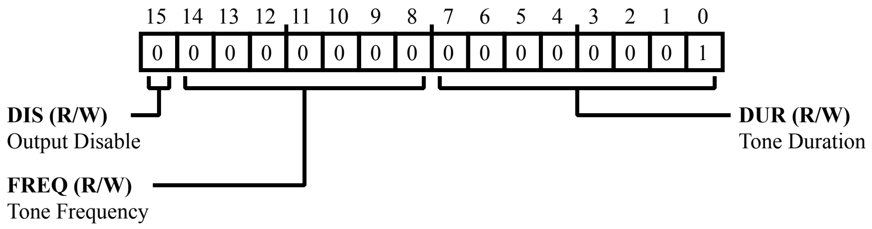


Figure 19-6: BEEP_TONEB Register Diagram

Table 19-5: BEEP_TONEB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	DIS	Output Disable. This bit is set to disable the beeper output while Tone B is playing. The beeper holds both of its two output pins at logic 0 when disabled. Writes to this bit take effect immediately if Tone B is currently playing.	
		0	Output enabled
		1	Output disabled, no DC potential across output pins
14:8 (R/W)	FREQ	Tone Frequency. This field defines the frequency for Tone B as integer divisions of the 32.768kHz clock. The values 0 through 3 are interpreted as 'rest-tone' and will result in no oscillations of the beeper output pins. All other values are directly used to divide the 32kHz input clock. Writes to this field immediately affect the output of Tone B if currently playing.	
		0-3	Rest tone (duration but no oscillation)
		4-127	Oscillation at 32kHz/(FREQ)
7:0 (R/W)	DUR	Tone Duration. This field defines the duration for Tone B in 4ms increments. Writing a value of 0x0 will immediately terminate Tone B if currently playing. Writing a value of 0xFF will cause Tone B to play forever once started, or until user code terminates the tone. Only a write of 0x0 will affect a currently playing tone, all other values written will be used only the next time Tone B is played.	
		0-254	Tone will play for (DUR)*4ms period
		255	Tone will play for infinite duration

20 ADC Subsystem

The ADuCM302x MCU incorporates a fast, multichannel, 12-bit analog-to-digital converter (ADC) capable of operating up to a maximum of 1.8 MSPS. The ADC controller can be setup to perform a series of conversions and transfer data to the system using a dedicated DMA channel. This allows the MCU to be in Flexi mode (minimizing the overall device power consumption) or perform other tasks.

ADC Features

The ADC has the following features:

- 12-bit resolution.
- Programmable ADC update rate up to 1.8 MSPS.
- Integrated input mux that supports up to eight channels.
- Supports temperature sensing.
- Supports battery monitoring.
- Software selectable on-chip reference voltage generation: 1.25 V and 2.5 V.
- Software selectable internal or external reference.
- Auto cycle mode: Ability to automatically select a sequence of input channels for conversion.
- Averaging function: Converted data on single or multiple channels can be averaged over up to 256 samples.
- Alert function: Internal digital comparator for AIN0, AIN1, AIN2, and AIN3 channels. An interrupt can be generated if the digital comparator detects an ADC result above or below the user-defined threshold.
- Supports dedicated DMA channel.
- Each channel, including temperature sensor and battery monitoring, has a dedicated data register for conversion result.

ADC Functional Description

This section provides information on the function of the ADC used by the ADuCM302x MCU.

ADC Subsystem Block Diagram

The figure shows the ADC subsystem.

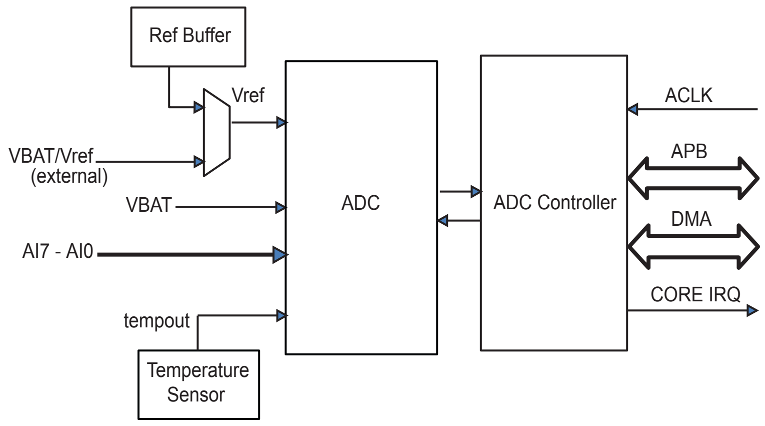


Figure 20-1: ADC Subsystem

ADC Subsystem Components

The ADC subsystem consists of the following components:

- ADC core
- ADC digital controller
- Reference buffer
- Temperature sensor

ADC Voltage Reference Selection

The ADC can use an internal voltage reference or external voltage reference for its operation.

Internal Sources

The ADuCM302x MCU integrates a reference buffer that can generate either 1.25 V or 2.5 V as reference using the integrated bandgap reference. This internal reference buffer is enabled by setting the `ADC_CFG.REFBUFEN` bit.

Depending on the battery range indicated by the power supply monitor, use either 2.5 V or 1.25 V as reference. The `ADC_CFG.VREFSEL` bit selects the reference. If the battery voltage is above 2.75 V, select 2.5 V or 1.25 V as reference. If the battery voltage is less than 2.75 V, select 1.25 V as reference.

Low-Power Mode

For conversion rate less than 1.6 MSPS, the reference buffer offers a capability where it consumes less current when compared with the regular operation. This mode of operation can be enabled by setting the `ADC_CFG1.RBUFLP` bit.

Sink Enable

If an external bias voltage is generated that requires an inverting configuration, the reference buffer can sink current, as shown in ***ADC Subsystem Reference Buffer with Current Sink*** figure. A 50 µA sink current capability at 1.25 V reference, and 100 µA sink current capability at 2.5 V reference is provided in the ADC subsystem. This can be enabled by setting the `ADC_CFG.SINKEN` bit.

The figure shows the reference buffer with a current sink.

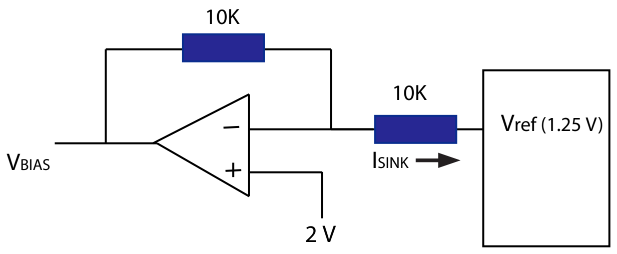


Figure 20-2: ADC Subsystem Reference Buffer with Current Sink

Fast Discharge of Internal Reference Buffer

For fast switching from a higher to lower reference voltage, write `ADC_CFG.FAST_DISCH` to 1.

For example, while using internal reference buffer at 2.5 V, if the battery discharges to 2.75 V, VREF must be switched to 1.25 V. For this, write `ADC_CFG.FAST_DISCH` and `ADC_CFG.VREFSEL` to 1.

External Voltage Reference

To select an external reference voltage source as ADC reference, clear the `ADC_CFG.REFBUFEN` bit and connect the external voltage reference across VREF_ADC and GND_VREFADC pins. To use VBAT as a reference voltage, externally connect VBAT_ADC to VREF_ADC.

The ***Reference Voltage Selection*** table shows the programming of the various bit fields in the `ADC_CFG` register to perform reference voltage selection.

Table 20-1: Reference Voltage Selection

Voltage	REFBUFEN	VREFSEL	VREFVBAT
1.25 V	1	1	0
2.5 V	1	0	0

Table 20-1: Reference Voltage Selection (Continued)

Voltage	REFBUFEN	VREFSEL	VREFVBAT
External Voltage Reference	0	X	0

NOTE: External Voltage Reference is connected to the ADC using VREF_ADC pin. This pin must not be grounded or connected to any voltage supply. It must be floating when internal reference and VBAT are used.

If the ADC and reference buffer are not used, VREF_ADC pin can be left floating (without connecting it to an external capacitor). If reference buffer is powered up without an external capacitor, VREF_ADC pin oscillates.

Digital Offset Calibration

An offset correction block is used to measure and correct the offset error of the reference voltage of the ADC. For accuracy, calibration is required.

After the ADC is powered up, to start a new calibration cycle, the `ADC_CFG.STARTCAL` bit must be set after ADC is ready. The `ADC_STAT.CALDONE` bit is set when calibration is done. Interrupt can be enabled by setting the `ADC_IRQ_EN.CALDONE` bit. The `ADC_CAL_WORD` register is loaded with a new calibration word at the end of calibration. This register is retained in hibernate mode. The `ADC_CAL_WORD` register can also be programmed by the user.

NOTE: If the `ADC_CAL_WORD` register is programmed during conversion, the new calibration word is considered from the next conversion cycle.

Powering the ADC

The ADC, reference buffer, and temperature sensor are powered down at the reset. The user has to explicitly power up each of these blocks. The power-up sequence depends on the internal or external reference used.

Using External Reference as Reference

To use the external reference as the ADC reference, the following sequence must be employed at power-up:

1. Set the `ADC_CFG.PWRUP` bit to power up the ADC.
2. Set `ADC_PWRUP.WAIT` bits as 526/(`CLKG_CLK_CTL1.PCLKDIVCNT` field).
3. Assert the `ADC_CFG.EN` bit and enable the ADC.
4. Wait for interrupt and write 1 to clear the `ADC_STAT.RDY` bit. Interrupt must be enabled by setting the `ADC_IRQ_EN.RDY` bit.
5. Wait for interrupt and write 1 to clear the `ADC_STAT.CALDONE` bit. Interrupt must be enabled by setting the `ADC_IRQ_EN.CALDONE` bit.
6. Set the `ADC_CFG.STARTCAL` bit to start the calibration cycle.

The ADC subsystem is ready for operation.

Using Internal Reference Buffer

To use the internal reference buffer, the following sequence must be employed at power-up:

1. Set the `ADC_CFG.PWRUP` bit to power up the ADC.
2. Set `ADC_PWRUP.WAIT` bits as 526/(`CLKG_CLK_CTL1.PCLKDIVCNT` field).
3. Select 1.25 V or 2.5 V as reference voltage using the `ADC_CFG.VREFSEL` bit.
4. Assert the `ADC_CFG.REFBUFEN` bit.
5. Assert the `ADC_CFG.EN` bit.
6. Wait for 3.5 ms.

   One of the general purpose timers can be used to wait for 3.5 ms. During this wait period, the device can be put into Flexi mode to save system power, and woken up by the general purpose timer interrupt.
7. Write 1 to clear the `ADC_STAT.RDY` bit.
8. Set the `ADC_CFG.STARTCAL` bit to start the calibration cycle.

The ADC is ready for conversion.

Hibernate

All components of the ADC subsystem are automatically powered down when the part goes to hibernate mode. This is done to keep the power consumption minimum. After waking up from hibernate mode, all components must be explicitly powered up and ready before a conversion can take place, as explained in Powering the ADC section.

However, while the ADC is in hibernate mode, the calibration coefficients are retained in the internal registers of the ADC. In this way, the ADC can wake up from hibernate, and a calibrated conversion cycle can be started immediately after the wake-up time has elapsed. A new calibration cycle, if required, can be run by asserting the `ADC_CFG.STARTCAL` bit.

A new calibration cycle is recommended at wake-up if the system is in hibernate mode for an extended period, and the operating conditions of the ADC, temperature, and reference voltage may have changed since the last calibration cycle.

The following register fields are retained when the chip goes into hibernate mode:

- `ADC_CFG.VREFSEL`
- `ADC_CFG.SINKEN`
- `ADC_PWRUP.WAIT`
- `ADC_CAL_WORD.VALUE`
- `ADC_AVG_CFG.FACTOR`
- `ADC_AVG_CFG.OS`

- ADC_LIMx_LO.VALUE
- ADC_LIMx_HI.VALUE
- ADC_HYSx.VALUE
- ADC_HYSx.MONCYC
- ADC_CFG1.RBUFLP

NOTE: If a conversion is ongoing prior to the part going into hibernate mode, that conversion does not resume after the part wakes up.

The ADC must be disabled before going to hibernate or shutdown mode by clearing the ADC_CFG.EN bit.

Sampling and Conversion Time

The ADC can be in one of the following phases:

- *Acquisition Phase*: During the acquisition phase, the capacitor arrays are connected directly to the inputs to get fully charged. The minimum required acquisition time depends on the output impedance of the source. The timing for this phase is programmed in the ADC_CNV_TIME.SAMPTIME bit in terms of number of clock cycles.

  Acquisition phase is (ADC_CNV_TIME.SAMPTIME + 1) ACLK cycles.

  The time depends on the ADC_CNV_TIME.SAMPTIME value and selected clock frequency (ACLK).

  ACLK frequency = 26 MHz / ACLKDIV

  ACLKDIV can be programmed in the CLKG_CLK_CTL1.ACLKDIVCNT bits (8 bits).

- *Conversion Phase*: At the end of the acquisition phase, the conversion phase is initiated. The conversion is completed by successive approximation.

  NOTE: The time between two conversions must not exceed 100 µs. If the ADC is idle for more than 100 µs, a dummy conversion must be performed.

Operation

The ADC controller supports several use cases.

Single Channel Mode

The ADC can be configured to convert on a particular channel by selecting one of the channels using the ADC_CNV_CFG.SEL bits. The ADC converts analog input, provides the CNV_DONE interrupt, and stores the result in the corresponding CHx_OUT register. A channel can be enabled by writing to the specific bit in the ADC_CNV_CFG.SEL bits. For example, to enable channel 2, set ADC_CNV_CFG.SEL[2]. Ensure that only one bit (out of ADC_CNV_CFG.SEL[7:0], ADC_CNV_CFG.BAT, ADC_CNV_CFG.TMP, ADC_CNV_CFG.TMP2) is set for conversion on single channel (ADC_CNV_CFG.AUTOMODE = 0).

To perform multiple conversions on a selected channel, set the `ADC_CNV_CFG.MULTI` bit. An interrupt is generated if enabled, and the corresponding `ADC_STAT` bit is set after each conversion. The conversion output is stored in the `ADC_CHx_OUT` register.

If the result is not read from the output register and the status bit is not cleared before the next conversion ends, the conversion output overflows and new result is stored in the `ADC_CHx_OUT` register, resulting in data loss.

It is recommended to use DMA for multiple conversions. Delay can also be introduced between successive conversions.

NOTE: After the desired number of conversions are complete, the `ADC_CNV_CFG.MULTI` bit must be cleared to stop the ADC subsystem from converting.

Auto Cycle Mode

Auto cycle mode reduces the MCU overhead of sampling and reading individual channel registers. It allows the user to select a sequence of ADC input channels and provides a single interrupt when conversions on all channels end. Temperature sensing and battery monitoring cannot be included in the auto cycle mode. Auto cycle mode is enabled by setting the `ADC_CNV_CFG.AUTOMODE` bit.

Conversions are performed starting from the lowest numbered channel to the highest numbered channel of the channels enabled. Output for each channel is stored in the respective `CHx_OUT` register, and `ADC_STAT` bits are set after conversion on all channels (one sequence) is completed.

To initiate multiple conversions on selected channels, enable the `ADC_CNV_CFG.MULTI` bit. An interrupt is generated after each sequence completes. Delay can be introduced between successive sequences. It is recommended to use DMA for multiple conversions due to the amount of data that can be generated.

NOTE: After the desired number of conversions are complete, the `ADC_CNV_CFG.MULTI` bit must be cleared to stop the ADC subsystem from converting. `ADC_CNV_CFG.MULTI` must be low for a minimum of one ACLK cycle. Status bits for conversion done, alert, and overflow must be cleared before starting a next conversion.

Delay Between Conversions

The `ADC_CNV_TIME.DLY` bits can be programmed to set the delay between completing one sequence of channels and starting another sequence, or multiple conversions on a single channel. This delay is in terms of number of ACLK cycles.

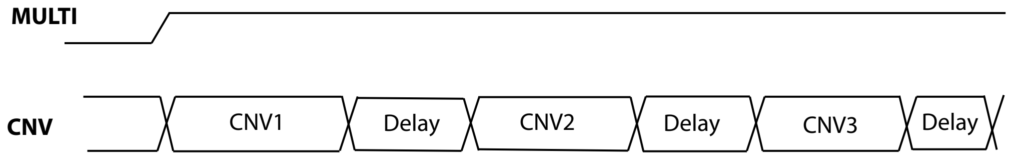


Figure 20-3: Delay Between Conversions on Single Channel in Multiple Conversions Mode

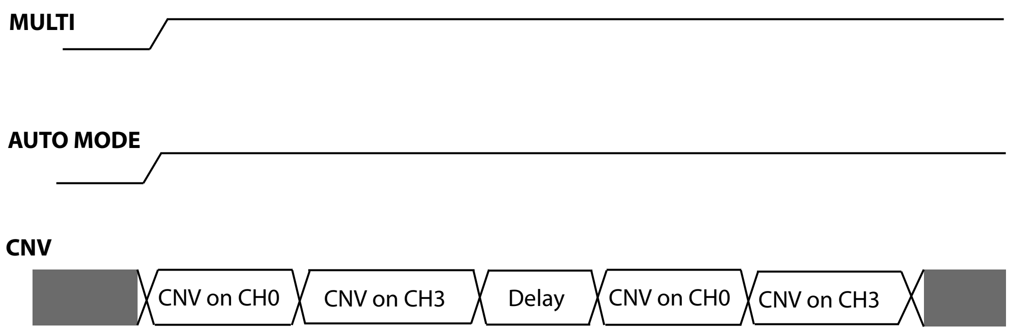


Figure 20-4: Delay Between Sequences in Auto Cycle Mode

DMA

A DMA channel can be used to reduce the processor overhead by moving the ADC results directly into SRAM with a single interrupt asserted after the required number of ADC conversions are completed. DMA can be enabled using the `ADC_CNV_CFG.DMAEN` bit. The conversion result must be read from the `ADC_DMA_OUT.RESULT` bits. The DMA channel can be programmed for a given number of conversions. The MCU can be in Flexi mode during this period, until the programmed number of conversions is completed and the DMA done interrupt is received. For more information, refer to the Programming Flow section.

Interrupts

The ADC controller has an interrupt associated with it. The interrupt status register indicates the source of an interrupt. When DMA is not used, the ADC controller generates data interrupts after a conversion.

NOTE: All interrupts are Write One to Clear.

Conversion

An interrupt can be generated after each conversion is complete by setting the `ADC_IRQ_EN.CNVDONE` bit. Corresponding *DONE* bit is set in the `ADC_STAT` register after the completion of a conversion.

Overflow

If output data is not read from the output register by the user or DMA before the next conversion is performed on the channel, it is overwritten. The overflow bit is set for the respective channel in the `ADC_OVF` register and interrupt is generated. It remains set until it is cleared by the user. This interrupt must be enabled by the user by setting the `ADC_IRQ_EN.OVF` bit.

NOTE: The `ADC_CNV_CFG.SEL` bits must remain unchanged when a conversion is being done.

Programming Flow

After powering up and calibrating (if required), the ADC is ready for conversion.

Single Conversion on Single Channel

This mode is used to perform a single conversion on a selected channel. The `ADC_CNV_CFG.SINGLE` bit must set to 1 to perform the conversion. This bit works as a write one to action bit and reads as zero. DMA is not recommended for this mode, as only a single data needs to be read. The conversion result can be read from the RESULT bits in the corresponding Channel Output register (ADC_CHx_OUT).

The following steps describe how to program the ADC for a single conversion on a single channel (say AIN2):

1. Set ADC_CNV_CFG.SEL = 0x2 and select channel 2 for conversion.
2. Set ADC_IRQ_EN.CNVDONE = 0x1 to enable interrupt when conversion is done.
3. Set ADC_CNV_CFG.SINGLE = 0x1 to start the conversion.
4. Interrupt is generated and ADC_STAT[2] is set when conversion is done.
5. Read ADC_CH0_OUT.RESULT to read the conversion result.
6. Set ADC_STAT.DONE2 = 0x1 to clear the interrupt.

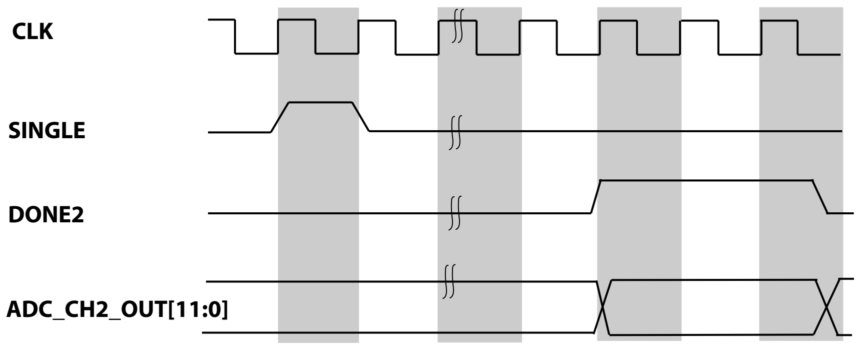


Figure 20-5: Single Conversion on Single Channel

Multiple Conversion on Single Channel

The ADC_CNV_CFG.MULTI bit is used to perform multiple conversions on the selected channel. The conversions continue until this bit is deasserted. Use DMA for this mode. This mode is similar to performing multiple back-to-back single conversions on the channel with the added overhead of reading the data (else, it is overwritten) and re-starting the conversion. When using DMA, the number of conversions to be done must be programmed in the count descriptor of the DMA.

The following steps describe how to program the ADC for a multiple conversion on a single channel (say AIN2):

1. Set ADC_CNV_CFG.SEL = 0x2 and select channel 2 for conversion.
2. Set the following DMA configurations:
   a. DMA count = 2 for three conversions (DMA count = number of conversions – 1).
   b. Source Address = Address of ADC_DMA_OUT register.
   c. Source Size = Half-word.
   d. Destination address of DMA as SRAM memory location address to store the conversion result.

    e. Program the required increment in the destination address.

3. Set `ADC_CNV_CFG.DMAEN` = 0x1 and enable DMA.
4. Set `ADC_CNV_TIME.DLY` = 0x14 and set the delay between two conversions.
5. Set `ADC_CNV_CFG.MULTI` = 0x1 to start the conversion.

   dma_done is generated after three conversions (count = 2).
6. Set `ADC_CNV_CFG.MULTI` = 0 and disable further conversions in ISR.

The conversion results can be read from the SRAM.

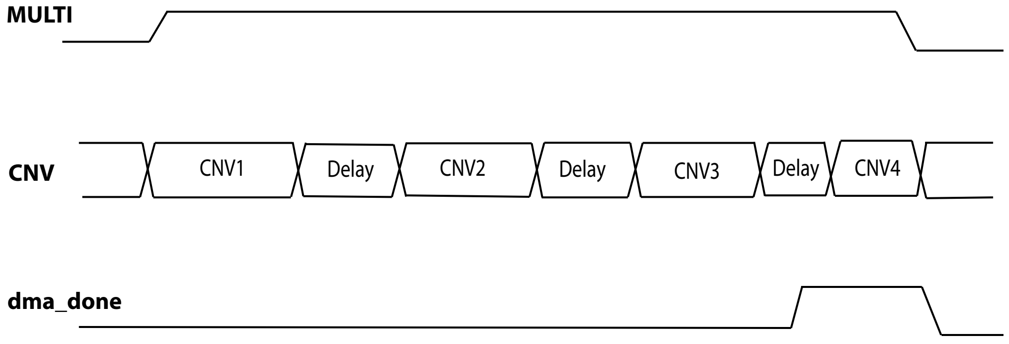


Figure 20-6: Multiple Conversion on Single Channel

Single Conversion in Auto Cycle Mode

Set the appropriate `ADC_CNV_CFG.SEL` channel bits to enable the channels included in the sequence.

The following steps describe how to program the ADC for a single conversion on a series of channels in auto cycle mode (say AIN2, AIN4, and AIN7):

1. Set `ADC_CNV_CFG.SEL` = 0x94.
2. Set the following DMA configurations:
    a. DMA count = 2, for three conversions (DMA count = number of conversions – 1).
    b. Source Address = Address of `ADC_DMA_OUT` register.
    c. Source Size = Half-word.
    d. Destination address of DMA as SRAM memory location address to store the conversion result.
    e. Program the required increment in the destination address.
3. Set `ADC_CNV_CFG.DMAEN` = 0x1 and enable DMA (if used).
4. Set `ADC_CNV_CFG.AUTOMODE` = 0x1.
5. Set `ADC_CNV_TIME.DLY` = 0x0.

6. Set ADC_CNV_CFG.SINGLE = 0x1 to start the conversion.

   dma_done is generated after conversion.

The conversion results can be read from the SRAM or respective Channel Out registers.

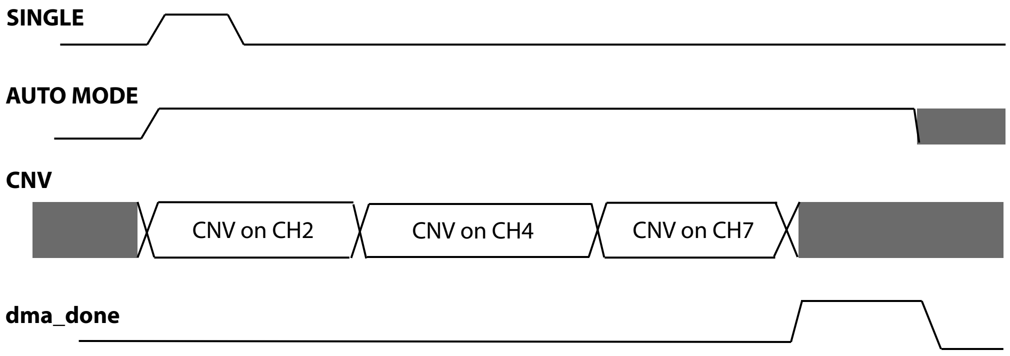


Figure 20-7: Single Conversion in Auto Cycle Mode

Multiple Conversions in Auto Cycle Mode

Program the number of sequences multiplied by number of channels selected to be run in the count descriptor of the DMA.

The following steps describe how to program the ADC for a multiple conversion on a series of channels in auto cycle mode (say AIN0, AIN2, and AIN3):

1. Set ADC_CNV_CFG.SEL = 0x0D.
2. Set ADC_CNV_TIME.DLY = 0x7E.
3. Set the following DMA configurations:
   a. Source Address = Address of ADC_DMA_OUT register.
   b. Source Size = Half-word.
   c. Destination address of DMA as SRAM memory location address to store the conversion result.
   d. Program the required increment in the destination address.
   e. Count = 5, for two sequences.
4. Set ADC_CNV_CFG.DMAEN = 0x1.
5. Set ADC_CNV_CFG.MULTI = 0x1 to start the conversion.

   dma_done is generated after conversion.
6. Set ADC_CNV_CFG.MULTI = 0 to disable further conversions in ISR.

The conversion results can be read from the SRAM.

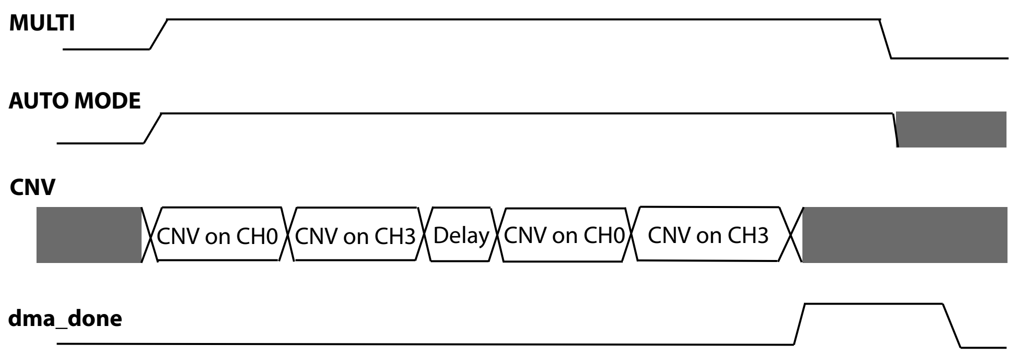


Figure 20-8: Multiple Conversions in Auto Cycle Mode

Temperature Sensor

The ADC subsystem provides a temperature sensor that measures die temperature. To enable the temperature sensing, set the `ADC_CFG.TMPEN` bit. Wait for 300 μs before starting the conversion. The temperature sensor can be enabled along with the ADC or the reference buffer to save time.

This temperature sensor is not designed for use as an absolute ambient temperature calculator. It is used as an approximate indicator of temperature of the ADuCM302x die.

Program the `ADC_CNV_TIME.SAMPTIME` bit to provide an acquisition time of 65 μs.

The following steps describe how to measure temperature:

1. Set the `ADC_CNV_CFG.TMP` bit to 1.
2. Set the `ADC_CNV_CFG.SINGLE` bit.
3. The `ADC_STAT.TMPDONE` bit is set after conversion.

   Interrupt is given if the `ADC_IRQ_EN.CNVDONE` bit is set.
4. Read the output, ***code1***, from the `ADC_TMP_OUT` register.
5. Set the `ADC_CNV_CFG.TMP2` bit to 1.
6. Set the `ADC_CNV_CFG.SINGLE` bit to 1.
7. The `ADC_STAT.TMP2DONE` bit is set after conversion.

   Interrupt is given if the `ADC_IRQ_EN.CNVDONE` bit is set.
8. Read the output, ***code2***, from `ADC_TMP2_OUT` register after conversion.

Auto mode can also be used.

Set the `ADC_CNV_CFG.TMP` and `ADC_CNV_CFG.TMP2` bits. Start the conversion by setting the `ADC_CNV_CFG.SINGLE` bit or `ADC_CNV_CFG.MULTI` bit (for multiple conversions).

Temperature can be calculated as:

T (^{0}C) = [[code1/(code2 + RG * code1)] * [Rvirtual$_{reference}$/ideal$_{sensitivity}$]] - 273.15

where,

- RG = 1.18
- Rvirtual$_{reference}$=1.223331
- ideal$_{sensitivity}$ = 0.001392736

For example, if:

- TMP = 1: Code1 = 1632
- TMP2 = 1: Code2 = 2080

T (^{0}C) = 84.64 ^{0}C

Battery Monitoring

To enable battery monitoring, the `ADC_CNV_CFG.BAT` bit must be set. The `ADC_CNV_TIME.SAMPTIME` bit must be set to obtain an acquisition time of 500 ns. Set the `ADC_CNV_CFG.SINGLE` bit to start conversion. The ADC converted output can be read from the `ADC_BAT_OUT` register or SRAM, depending on the conversion mode.

Conversion done interrupt can be enabled for getting an interrupt after conversion is done. Clear the `ADC_CNV_CFG.BAT` bit after conversion is done to reduce power consumption.

To convert ADC result to battery voltage:

VBAT = 4 * adc_out * Vref / (2^{12} – 1)

where,

- VBAT = battery voltage
- Vref = reference voltage selected (preferred, 1.25 V)
- adc_out = ADC converted output

Over Sampling

Resolution greater than 12-bit can be achieved by oversampling. Oversampling can be enabled by writing the `ADC_AVG_CFG.OS` and `ADC_AVG_CFG.EN` bits to 1. When oversampling is enabled, the ADC samples multiple times and averages the result to provide the required output in the `CHx_OUT` register.

The value to be programmed in the `ADC_AVG_CFG.FACTOR` field for a required resolution is given in the ***Factor for Enhanced Resolution*** table.

Table 20-2: Factor for Enhanced Resolution

Resolution Required	AVG_CFG_FACTOR to be programmed	Number of Samples Used
13-bit	h02	4
14-bit	h08	16
15-bit	h20	64
16-bit	h80	256

Averaging Function

When performing multiple conversions, averaging can be enabled on all selected channels. Averaging can be performed over a maximum of 256 samples, in the steps of power of 2 (that is, 2, 4, 8, 16, …256). Averaging can be enabled by writing the `ADC_AVG_CFG.OS` bit to 0 and the `ADC_AVG_CFG.EN` bit to 1.

Averaging factor is to be programmed as factor/2 in the `ADC_AVG_CFG.FACTOR` bits.

For example, to average 64 samples, program `ADC_AVG_CFG.FACTOR` as 32 (0x20). An interrupt is generated after averaging is complete.

In case of auto cycle mode, DMA can be enabled.

For example, to average over 16 samples, 16 values are converted, added, and then divided by 16.

If averaging is enabled in auto mode (for multiple channels), programmed number of conversions are done on one channel (and averaged) before moving to the next channel.

Averaging for Multiple Conversions on Single Channel

The following steps describe how to perform averaging over multiple conversions on a single channel (say AIN2):

1. Set `ADC_CNV_CFG.SEL` = 0x2 and select Channel 2 for conversion.
2. Set `ADC_IRQ_EN.CNVDONE` = 0x1 to enable interrupt when conversion is done.
3. Set `ADC_AVG_CFG.OS` = 0.
4. Set `ADC_AVG_CFG.EN` = 0x1 to enable averaging.
5. Set `ADC_AVG_CFG.FACTOR` = 0x20 to average 64 samples.
6. Set `ADC_CNV_CFG.SINGLE` = 0x1 to start the conversion.
7. Interrupt is generated and `ADC_STAT[2]` is set when conversion is done.
8. Set cnv_result = `ADC_CH2_OUT.RESULT` to read the conversion output.
9. Set `ADC_STAT.DONE2` = 0x1 to clear the interrupt.

Averaging for Multiple Conversions in Auto Cycle Mode

The following steps describe how to perform averaging over multiple conversion on a series of channels in auto cycle mode:

1. Set ADC_CNV_CFG.SEL = 0x30 and select Channel 5 and Channel 4 for conversion.
2. Set the following DMA configurations:
    a. DMA count = 1, for averaging ones over two channels (DMA count = number of conversions – 1).
    b. Source Address = Address of ADC_DMA_OUT register.
    c. Source Size = Half-word.
    d. Destination address of DMA as SRAM memory location address to store the conversion result.
    e. Program the required increment in the destination address.
3. Set ADC_CNV_CFG.DMAEN = 0x1 and enable DMA (if used).
4. Set ADC_CNV_CFG.AUTOMODE = 0x1 to start the conversion.
5. Set ADC_AVG_CFG.OS = 0.
6. Set ADC_AVG_CFG.EN = 0x1.
7. Set ADC_AVG_CFG.FACTOR = 0x08 to average 16 samples.
8. Set ADC_CNV_CFG.SINGLE = 0x1 to start the conversion.

    dma_done is generated after conversion.

The conversion result can be read from the SRAM or respective Channel Out registers.

NOTE: Averaging/over sampling and monitoring cannot be enabled together. They are mutually exclusive.

ADC Digital Comparator

A digital comparator is provided to allow an interrupt to be triggered if the ADC input is above or below a programmable threshold. Input channels AIN0, AIN1, AIN2, and AIN3 can be used with the digital comparator. This can be used to continuously monitor if any (or a set) of these channels stay within normal range of values. Comparators can be enabled only in multiple conversion mode, either on single channel or in auto cycle mode. Overflow indication is disabled, as the user is not expected to read the results during monitoring.

To set up the ADC digital comparator:

- Lower threshold value must be written to 12-bit ADC_LIMx_LO.VALUE field. To enable comparison with lower limit, the ADC_LIMx_LO.EN bit of the corresponding register must be set.
- Higher threshold value must be written to 12-bit ADC_LIMx_HI.VALUE field. To enable comparison, the ADC_LIMx_HI.EN bit of the corresponding register must be set.
- Hysteresis value can also be given for each of these channels in the ADC_HYSx.VALUE bit . It must be enabled by setting the ADC_HYSx.EN bit.

- An alert is indicated if the ADC result crosses the threshold (`ADC_LIMx_LO.VALUE` or `ADC_LIMx_HI.VALUE`) and does not return to its normal value within number of clock cycles programmed in the `ADC_HYSx.MONCYC` bit.

  The normal value is defined as:

  - For lower threshold: Above `ADC_LIMx_LO.VALUE` + `ADC_HYSx.VALUE`
  - For higher threshold: Below `ADC_LIMx_HI.VALUE` – `ADC_HYSx.VALUE`

  If the converted result is above `ADC_LIMx_HI.VALUE`, it is monitored for the next `ADC_HYSx.MONCYC` conversions.

  If the converted results during monitoring remain above `ADC_LIMx_HI.VALUE` - `ADC_HYSx.VALUE`, an alert is indicated.

  If the converted result during monitoring is below `ADC_LIMx_HI.VALUE` – `ADC_HYSx.VALUE`, alert is not indicated and monitoring stops.

  Similarly, if the converted result is below `ADC_LIMx_LO.VALUE`, it is monitored for the next `ADC_HYSx.MONCYC` conversions.

  If the converted results during monitoring remain below `ADC_LIMx_LO.VALUE` + `ADC_HYSx.VALUE`, an alert is indicated.

  If the converted result during monitoring is above `ADC_LIMx_LO.VALUE` + `ADC_HYSx.VALUE`, alert is not indicated and monitoring stops.

The `ADC_ALERT` register can be read to determine the source of alert. An interrupt is generated if the `ADC_IRQ_EN.ALERT` bit is set.

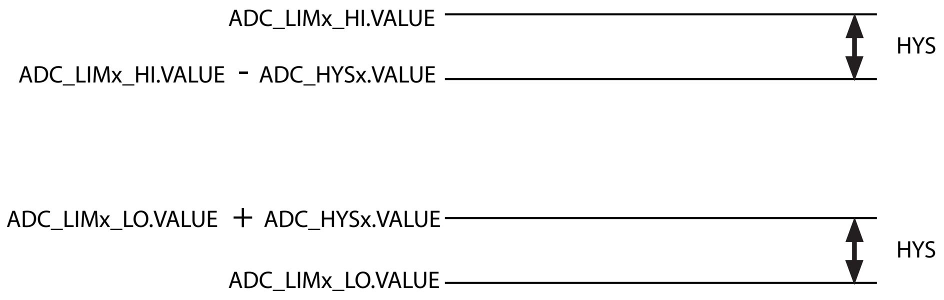

ADC_LIMx_HI.VALUE
ADC_LIMx_HI.VALUE - ADC_HYSx.VALUE
HYS
ADC_LIMx_LO.VALUE + ADC_HYSx.VALUE
ADC_LIMx_LO.VALUE
HYS

Figure 20-9: Low-High Limits and Hysteresis

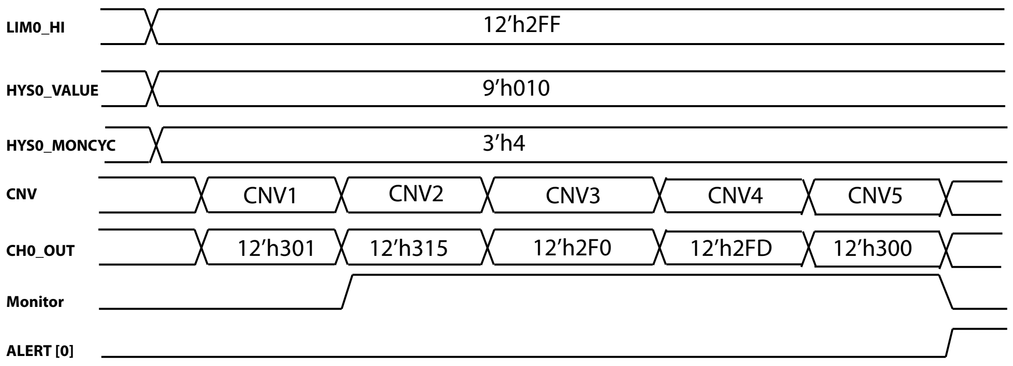


Figure 20-10: Alert Functionality

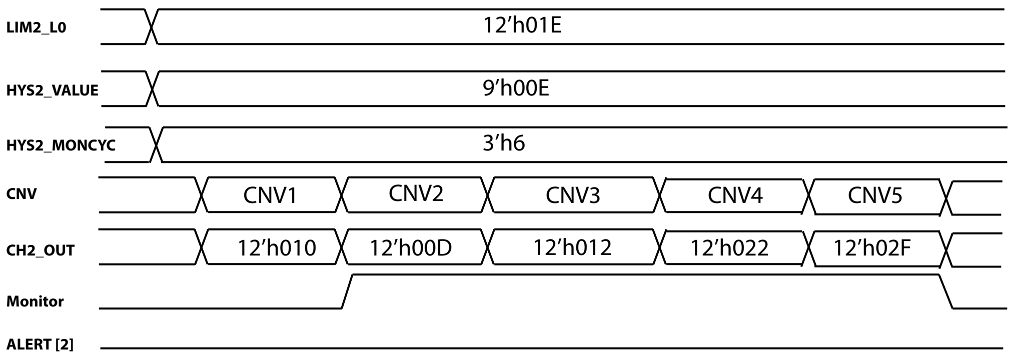


Figure 20-11: No Alert when Conversion Result Returns to Expected Value within MONCYC Cycles

In auto cycle mode, channels are converted sequentially. If any channel crosses the threshold, it is monitored before moving to the next channel in sequence. Monitoring stops if the channel input returns to a normal value within `ADC_HYSx.MONCYC` cycles or an alert is raised after `ADC_HYSx.MONCYC` cycles.

NOTE: Hysteresis is not supported with DMA enabled.

The flowchart explains the flow of conversion when comparison is enabled on three channels in auto cycle mode.

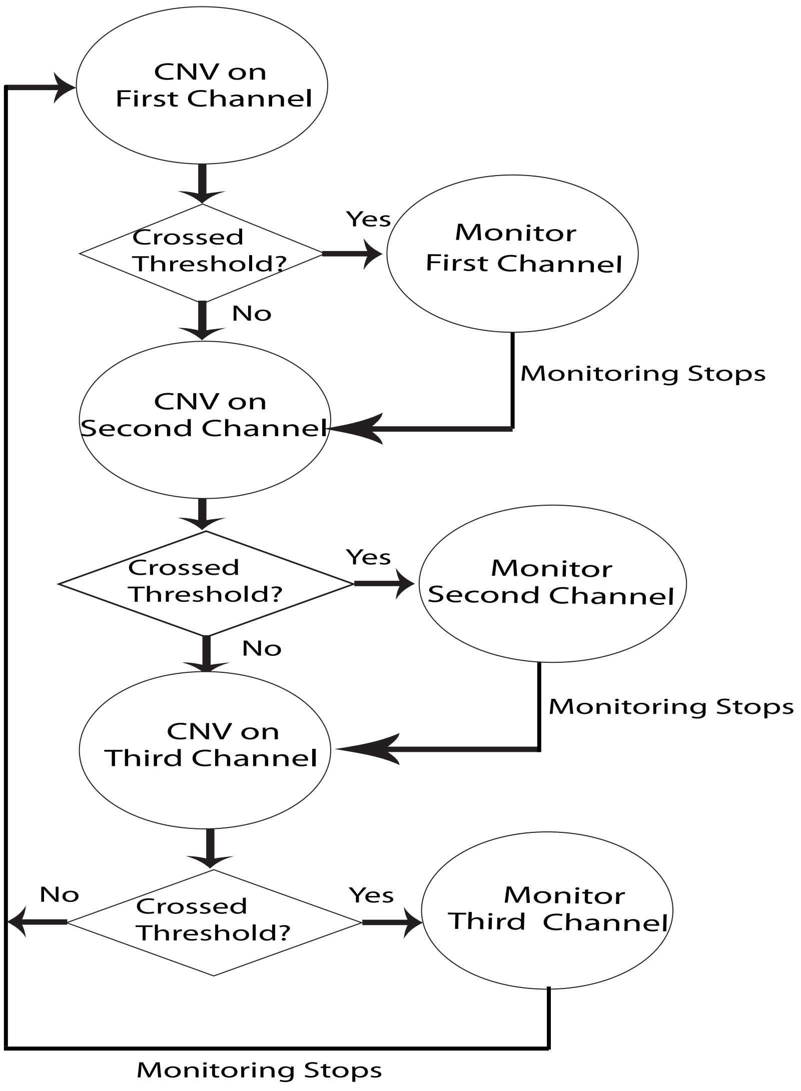


Figure 20-12: Conversion Flow for Comparison Enabled on Three Channels in Auto Cycle Mode

ADuCM302x ADC Register Descriptions

ADC contains the following registers.

Table 20-3: ADuCM302x ADC Register List

Name	Description
ADC_ALERT	Alert Indication
ADC_AVG_CFG	Averaging Configuration
ADC_BAT_OUT	Battery Monitoring Result
ADC_CAL_WORD	Calibration Word
ADC_CFG	ADC Configuration

Table 20-3: ADuCM302x ADC Register List (Continued)

Name	Description
ADC_CFG1	Reference Buffer Low Power Mode
ADC_CH0_OUT	Conversion Result Channel 0
ADC_CH1_OUT	Conversion Result Channel 1
ADC_CH2_OUT	Conversion Result Channel 2
ADC_CH3_OUT	Conversion Result Channel 3
ADC_CH4_OUT	Conversion Result Channel 4
ADC_CH5_OUT	Conversion Result Channel 5
ADC_CH6_OUT	Conversion Result Channel 6
ADC_CH7_OUT	Conversion Result Channel 7
ADC_CNV_CFG	ADC Conversion Configuration
ADC_CNV_TIME	ADC Conversion Time
ADC_DMA_OUT	DMA Output Register
ADC_HYS0	Channel 0 Hysteresis
ADC_HYS1	Channel 1 Hysteresis
ADC_HYS2	Channel 2 Hysteresis
ADC_HYS3	Channel 3 Hysteresis
ADC_IRQ_EN	Interrupt Enable
ADC_LIM0_HI	Channel 0 High Limit
ADC_LIM0_LO	Channel 0 Low Limit
ADC_LIM1_HI	Channel 1 High Limit
ADC_LIM1_LO	Channel 1 Low Limit
ADC_LIM2_HI	Channel 2 High Limit
ADC_LIM2_LO	Channel 2 Low Limit
ADC_LIM3_HI	Channel 3 High Limit
ADC_LIM3_LO	Channel 3 Low Limit
ADC_OVF	Overflow of Output Registers
ADC_PWRUP	ADC Power-up Time
ADC_STAT	ADC Status
ADC_TMP2_OUT	Temperature Result 2
ADC_TMP_OUT	Temperature Result

Alert Indication

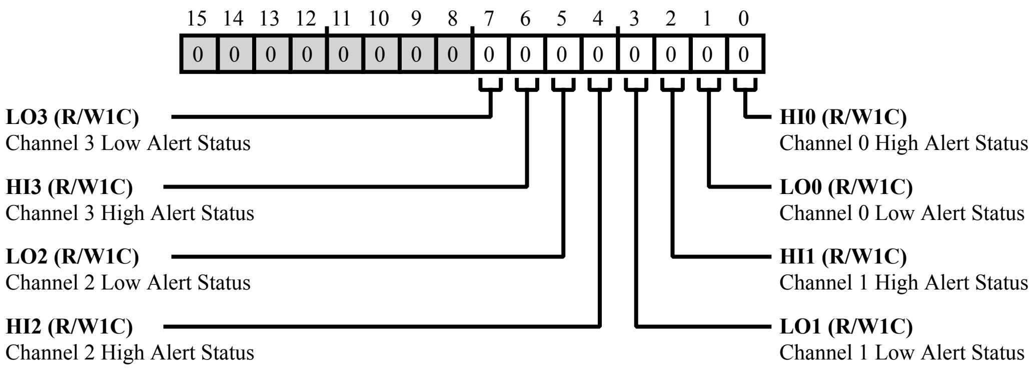

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LO3 (R/W1C)
Channel 3 Low Alert Status
HI3 (R/W1C)
Channel 3 High Alert Status
LO2 (R/W1C)
Channel 2 Low Alert Status
HI2 (R/W1C)
Channel 2 High Alert Status
HI0 (R/W1C)
Channel 0 High Alert Status
LO0 (R/W1C)
Channel 0 Low Alert Status
HI1 (R/W1C)
Channel 1 High Alert Status
LO1 (R/W1C)
Channel 1 Low Alert Status

Figure 20-13: ADC_ALERT Register Diagram

Table 20-4: ADC_ALERT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W1C)	LO3	Channel 3 Low Alert Status.
6 (R/W1C)	HI3	Channel 3 High Alert Status.
5 (R/W1C)	LO2	Channel 2 Low Alert Status.
4 (R/W1C)	HI2	Channel 2 High Alert Status.
3 (R/W1C)	LO1	Channel 1 Low Alert Status.
2 (R/W1C)	HI1	Channel 1 High Alert Status.
1 (R/W1C)	LO0	Channel 0 Low Alert Status.
0 (R/W1C)	HI0	Channel 0 High Alert Status.

Averaging Configuration

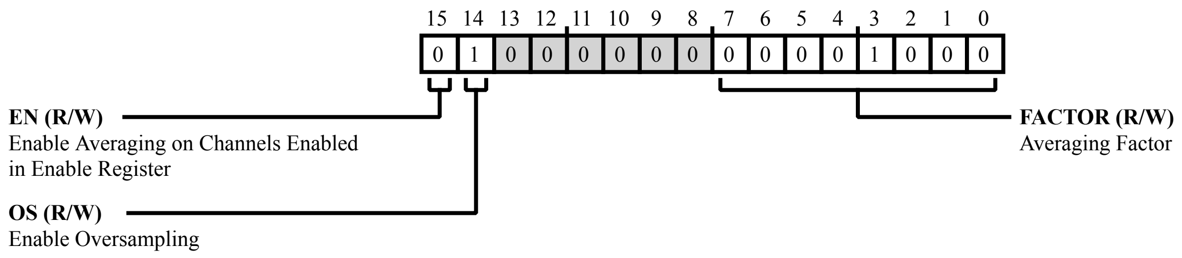


Figure 20-14: ADC_AVG_CFG Register Diagram

Table 20-5: ADC_AVG_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	EN	Enable Averaging on Channels Enabled in Enable Register.
14 (R/W)	OS	Enable Oversampling.
7:0 (R/W)	FACTOR	Averaging Factor. Program averaging factor for averaging enabled channels (1-256).

Battery Monitoring Result

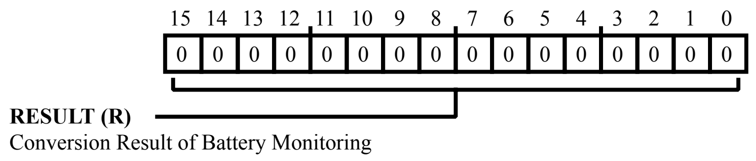


Figure 20-15: ADC_BAT_OUT Register Diagram

Table 20-6: ADC_BAT_OUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	RESULT	Conversion Result of Battery Monitoring.

Calibration Word

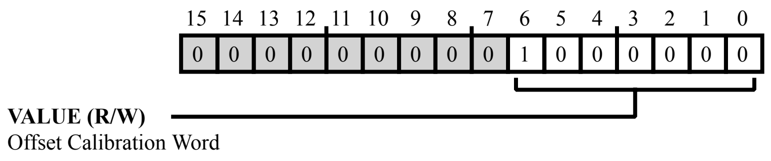


Figure 20-16: ADC_CAL_WORD Register Diagram

Table 20-7: ADC_CAL_WORD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Offset Calibration Word.

ADC Configuration

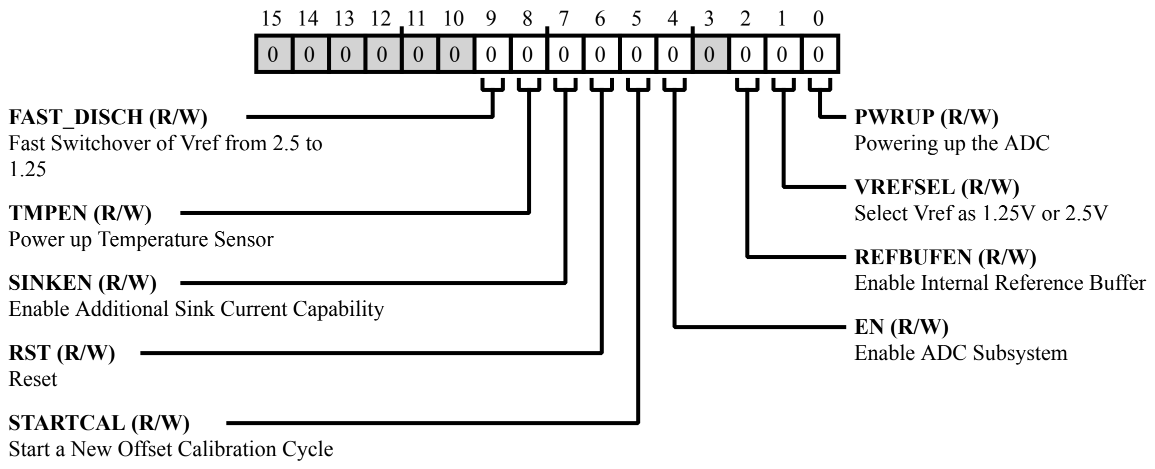


Figure 20-17: ADC_CFG Register Diagram

Table 20-8: ADC_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	FAST_DISCH	Fast Switchover of Vref from 2.5 to 1.25. For fast switchover of vref from 2.5 V to 1.25 V.	
8 (R/W)	TMPEN	Power up Temperature Sensor.	
7 (R/W)	SINKEN	Enable Additional Sink Current Capability. To enable additional 50uA sink current capability @1.25V, 100uA current capability @2.5V.	
6 (R/W)	RST	Reset. Resets internal buffers and registers when high.	
5 (R/W)	STARTCAL	Start a New Offset Calibration Cycle.	
4 (R/W)	EN	Enable ADC Subsystem.	
2 (R/W)	REFBUFEN	Enable Internal Reference Buffer.	
		0	External reference is used
		1	Reference buffer is enabled

Table 20-8: ADC_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	VREFSEL	Select Vref as 1.25V or 2.5V.	
		0	Vref = 2.5V
		1	Vref = 1.25V
0 (R/W)	PWRUP	Powering up the ADC.	

Reference Buffer Low Power Mode

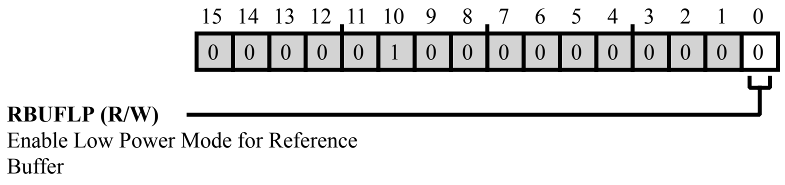


Figure 20-18: ADC_CFG1 Register Diagram

Table 20-9: ADC_CFG1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	RBUFLP	Enable Low Power Mode for Reference Buffer.

Conversion Result Channel 0

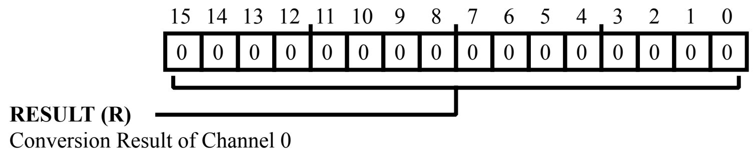


Figure 20-19: ADC_CH0_OUT Register Diagram

Table 20-10: ADC_CH0_OUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	RESULT	Conversion Result of Channel 0.

Conversion Result Channel 1

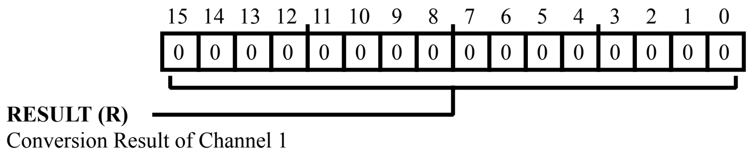


Figure 20-20: ADC_CH1_OUT Register Diagram

Table 20-11: ADC_CH1_OUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	RESULT	Conversion Result of Channel 1.

Conversion Result Channel 2

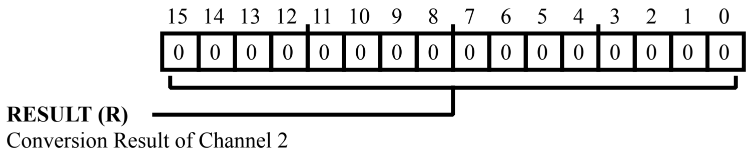


Figure 20-21: ADC_CH2_OUT Register Diagram

Table 20-12: ADC_CH2_OUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	RESULT	Conversion Result of Channel 2.

Conversion Result Channel 3

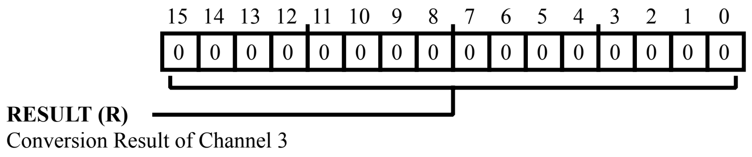


Figure 20-22: ADC_CH3_OUT Register Diagram

Table 20-13: ADC_CH3_OUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	RESULT	Conversion Result of Channel 3.

Conversion Result Channel 4

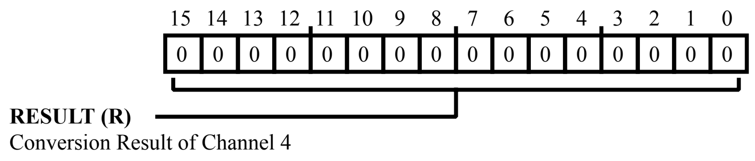


Figure 20-23: ADC_CH4_OUT Register Diagram

Table 20-14: ADC_CH4_OUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	RESULT	Conversion Result of Channel 4.

Conversion Result Channel 5

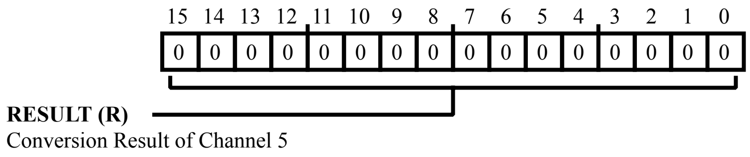


Figure 20-24: ADC_CH5_OUT Register Diagram

Table 20-15: ADC_CH5_OUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	RESULT	Conversion Result of Channel 5.

Conversion Result Channel 6

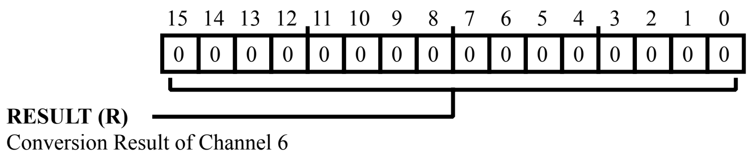


Figure 20-25: ADC_CH6_OUT Register Diagram

Table 20-16: ADC_CH6_OUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	RESULT	Conversion Result of Channel 6.

Conversion Result Channel 7

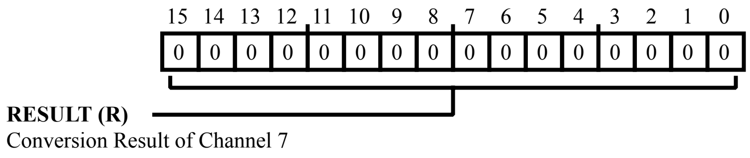


Figure 20-26: ADC_CH7_OUT Register Diagram

Table 20-17: ADC_CH7_OUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	RESULT	Conversion Result of Channel 7.

ADC Conversion Configuration

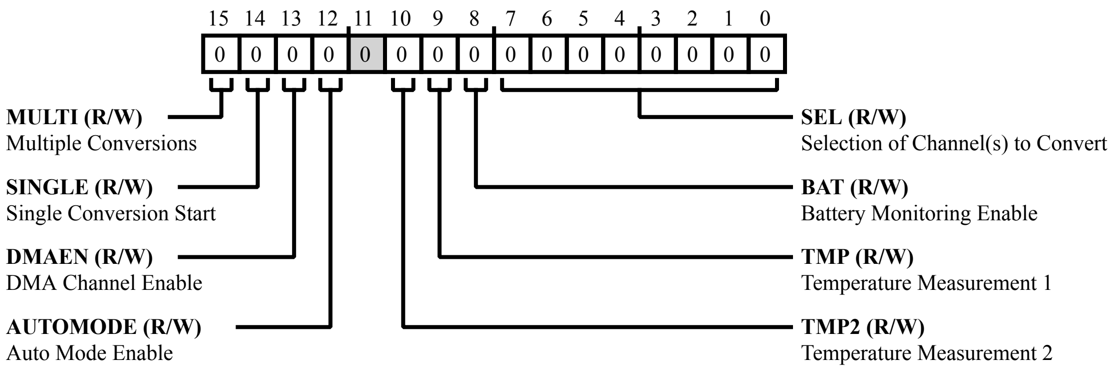


Figure 20-27: ADC_CNV_CFG Register Diagram

Table 20-18: ADC_CNV_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	MULTI	Multiple Conversions. Set to start multiple conversions.
14 (R/W)	SINGLE	Single Conversion Start. Set to start single conversion.
13 (R/W)	DMAEN	DMA Channel Enable.
12 (R/W)	AUTOMODE	Auto Mode Enable.
10 (R/W)	TMP2	Temperature Measurement 2.
9 (R/W)	TMP	Temperature Measurement 1.
8 (R/W)	BAT	Battery Monitoring Enable.
7:0 (R/W)	SEL	Selection of Channel(s) to Convert.

ADC Conversion Time

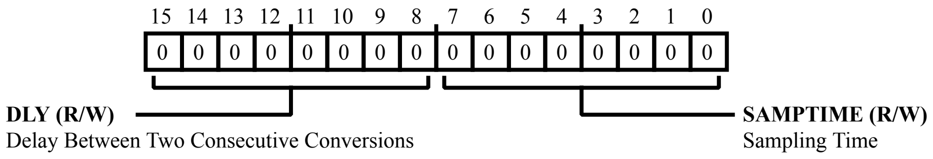


Figure 20-28: ADC_CNV_TIME Register Diagram

Table 20-19: ADC_CNV_TIME Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	DLY	Delay Between Two Consecutive Conversions. Delay between two consecutive conversions in terms of number of ACLK cycles.
7:0 (R/W)	SAMPTIME	Sampling Time. Number of clock cycles (ACLK) required for sampling.

DMA Output Register

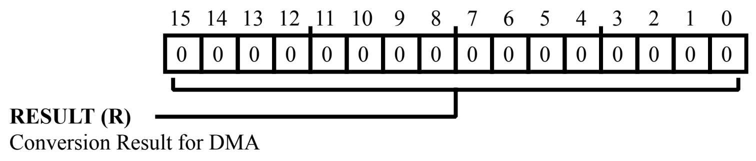


Figure 20-29: ADC_DMA_OUT Register Diagram

Table 20-20: ADC_DMA_OUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	RESULT	Conversion Result for DMA.

Channel 0 Hysteresis

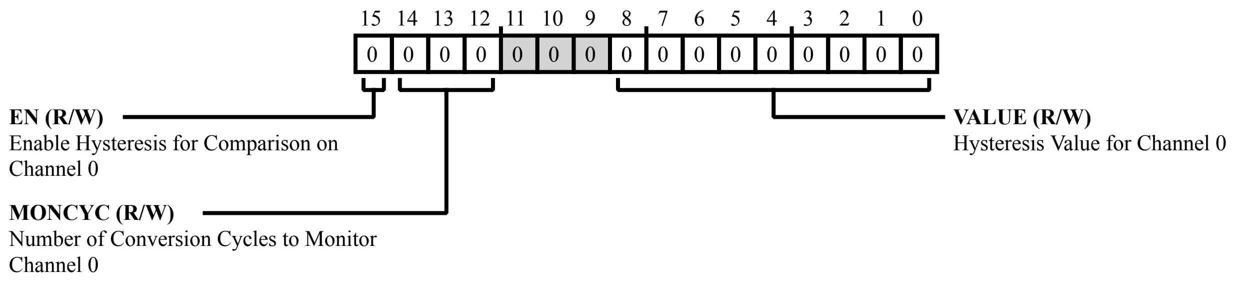
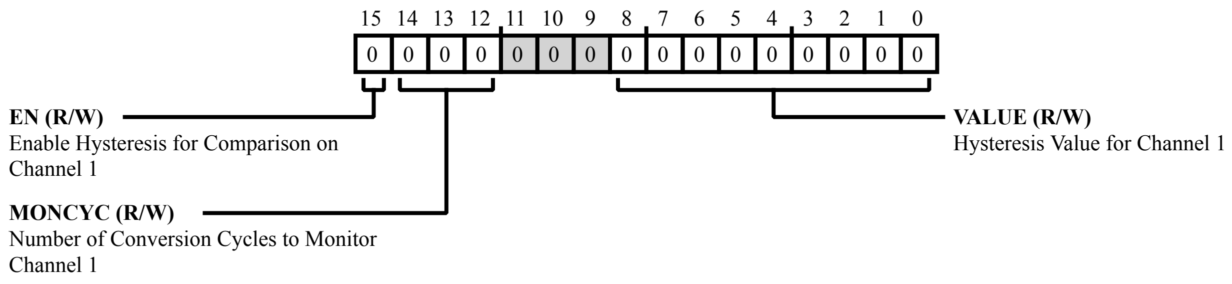

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN (R/W)
Enable Hysteresis for Comparison on Channel 0
MONCYC (R/W)
Number of Conversion Cycles to Monitor Channel 0
VALUE (R/W)
Hysteresis Value for Channel 0

Figure 20-30: ADC_HYS0 Register Diagram

Table 20-21: ADC_HYS0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	EN	Enable Hysteresis for Comparison on Channel 0.
14:12 (R/W)	MONCYC	Number of Conversion Cycles to Monitor Channel 0. Program number of conversion cycles to monitor channel 0 before raising alert.
8:0 (R/W)	VALUE	Hysteresis Value for Channel 0.

Channel 1 Hysteresis



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
EN (R/W)
Enable Hysteresis for Comparison on Channel 1
MONCYC (R/W)
Number of Conversion Cycles to Monitor Channel 1
VALUE (R/W)
Hysteresis Value for Channel 1

Figure 20-31: ADC_HYS1 Register Diagram

Table 20-22: ADC_HYS1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	EN	Enable Hysteresis for Comparison on Channel 1.
14:12 (R/W)	MONCYC	Number of Conversion Cycles to Monitor Channel 1. Program number of conversion cycles to monitor channel 1 before raising alert.
8:0 (R/W)	VALUE	Hysteresis Value for Channel 1.

Channel 2 Hysteresis

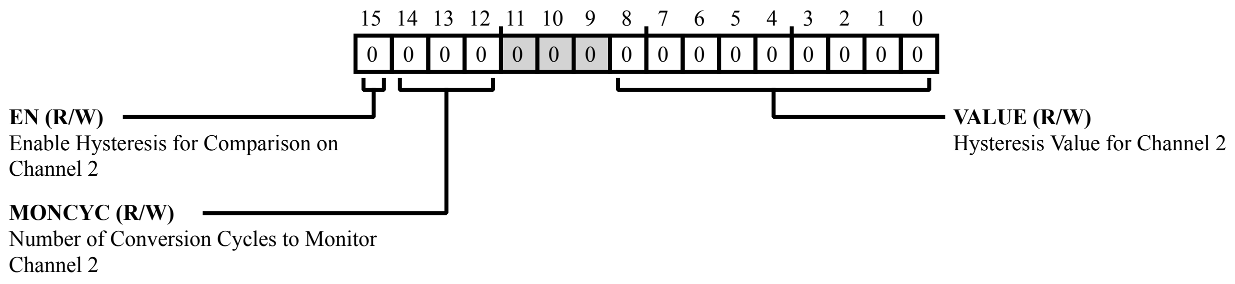

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN (R/W)
Enable Hysteresis for Comparison on Channel 2
MONCYC (R/W)
Number of Conversion Cycles to Monitor Channel 2
VALUE (R/W)
Hysteresis Value for Channel 2

Figure 20-32: ADC_HYS2 Register Diagram

Table 20-23: ADC_HYS2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	EN	Enable Hysteresis for Comparison on Channel 2.
14:12 (R/W)	MONCYC	Number of Conversion Cycles to Monitor Channel 2. Program number of conversion cycles to monitor channel 2 before raising alert.
8:0 (R/W)	VALUE	Hysteresis Value for Channel 2.

Channel 3 Hysteresis

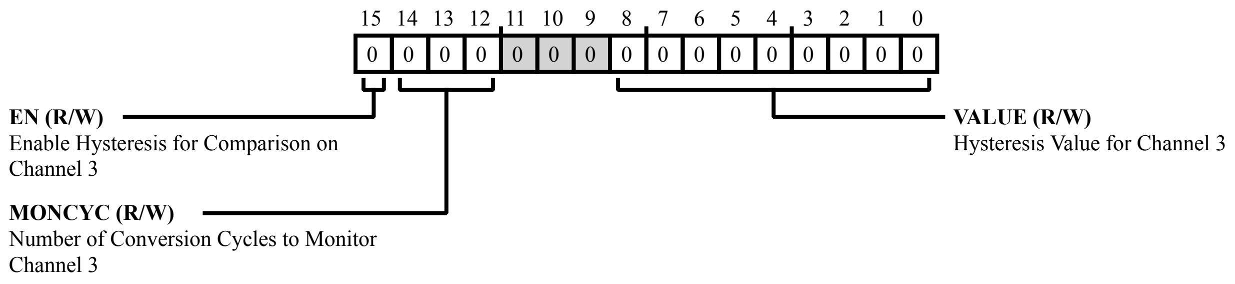

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN (R/W)
Enable Hysteresis for Comparison on Channel 3
MONCYC (R/W)
Number of Conversion Cycles to Monitor Channel 3
VALUE (R/W)
Hysteresis Value for Channel 3

Figure 20-33: ADC_HYS3 Register Diagram

Table 20-24: ADC_HYS3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	EN	Enable Hysteresis for Comparison on Channel 3.
14:12 (R/W)	MONCYC	Number of Conversion Cycles to Monitor Channel 3.
8:0 (R/W)	VALUE	Hysteresis Value for Channel 3.

Interrupt Enable

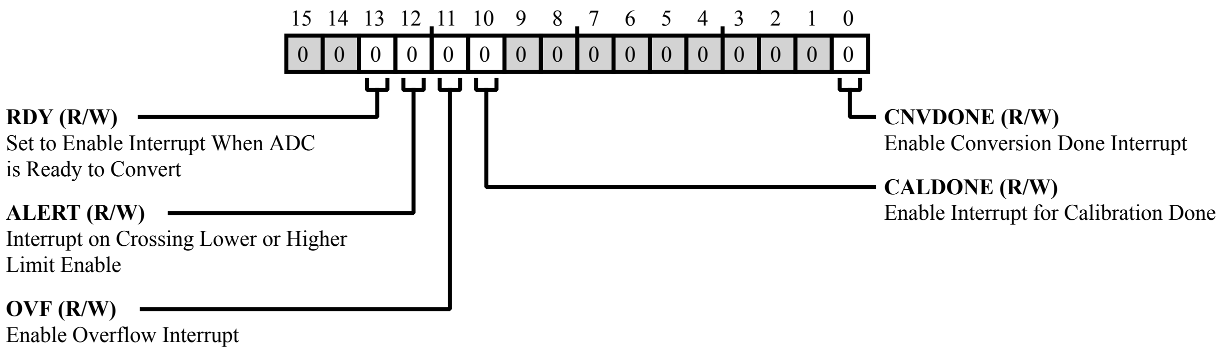

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RDY (R/W)
Set to Enable Interrupt When ADC is Ready to Convert
ALERT (R/W)
Interrupt on Crossing Lower or Higher Limit Enable
OVF (R/W)
Enable Overflow Interrupt
CNVDONE (R/W)
Enable Conversion Done Interrupt
CALDONE (R/W)
Enable Interrupt for Calibration Done

Figure 20-34: ADC_IRQ_EN Register Diagram

Table 20-25: ADC_IRQ_EN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W)	RDY	Set to Enable Interrupt When ADC is Ready to Convert.
12 (R/W)	ALERT	Interrupt on Crossing Lower or Higher Limit Enable.
11 (R/W)	OVF	Enable Overflow Interrupt. Set to enable interrupt in case of overflow.
10 (R/W)	CALDONE	Enable Interrupt for Calibration Done.
0 (R/W)	CNVDONE	Enable Conversion Done Interrupt. Set it to enable interrupt after conversion is done.

Channel 0 High Limit

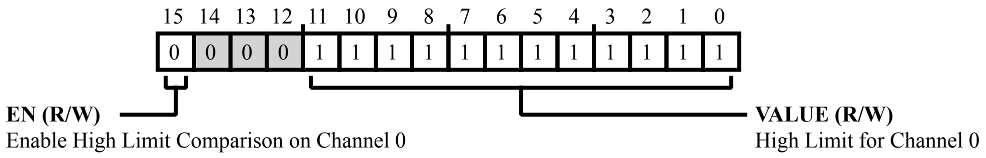


Figure 20-35: ADC_LIM0_HI Register Diagram

Table 20-26: ADC_LIM0_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	EN	Enable High Limit Comparison on Channel 0.
11:0 (R/W)	VALUE	High Limit for Channel 0.

Channel 0 Low Limit

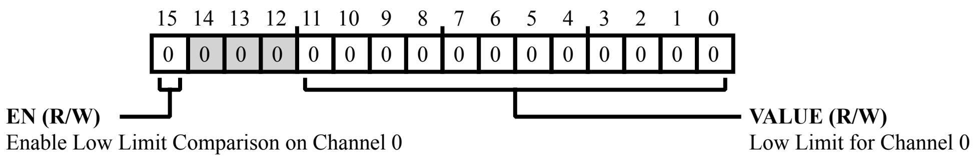


Figure 20-36: ADC_LIM0_LO Register Diagram

Table 20-27: ADC_LIM0_LO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	EN	Enable Low Limit Comparison on Channel 0.
11:0 (R/W)	VALUE	Low Limit for Channel 0.

Channel 1 High Limit

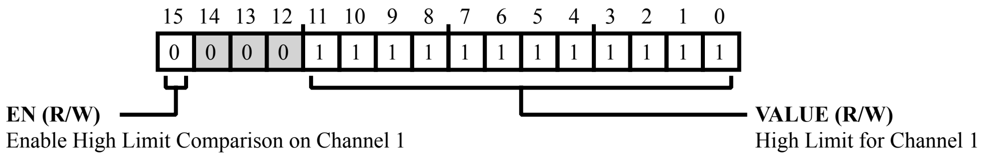


Figure 20-37: ADC_LIM1_HI Register Diagram

Table 20-28: ADC_LIM1_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	EN	Enable High Limit Comparison on Channel 1.
11:0 (R/W)	VALUE	High Limit for Channel 1.

Channel 1 Low Limit

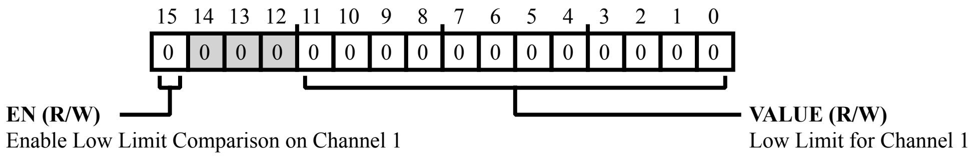


Figure 20-38: ADC_LIM1_LO Register Diagram

Table 20-29: ADC_LIM1_LO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	EN	Enable Low Limit Comparison on Channel 1.
11:0 (R/W)	VALUE	Low Limit for Channel 1.

Channel 2 High Limit

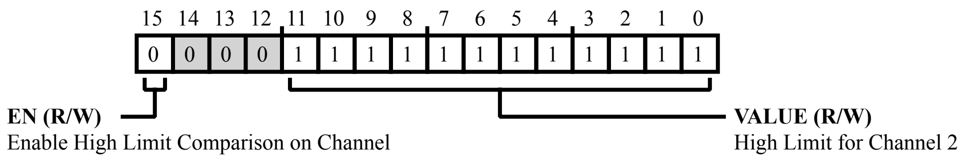


Figure 20-39: ADC_LIM2_HI Register Diagram

Table 20-30: ADC_LIM2_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	EN	Enable High Limit Comparison on Channel.
11:0 (R/W)	VALUE	High Limit for Channel 2.

Channel 2 Low Limit

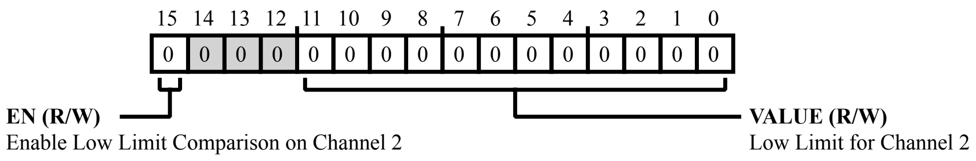


Figure 20-40: ADC_LIM2_LO Register Diagram

Table 20-31: ADC_LIM2_LO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	EN	Enable Low Limit Comparison on Channel 2.
11:0 (R/W)	VALUE	Low Limit for Channel 2.

Channel 3 High Limit

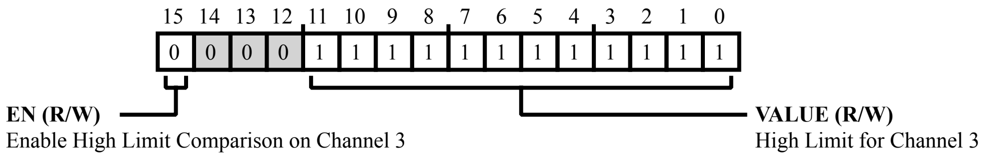


Figure 20-41: ADC_LIM3_HI Register Diagram

Table 20-32: ADC_LIM3_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	EN	Enable High Limit Comparison on Channel 3.
11:0 (R/W)	VALUE	High Limit for Channel 3.

Channel 3 Low Limit

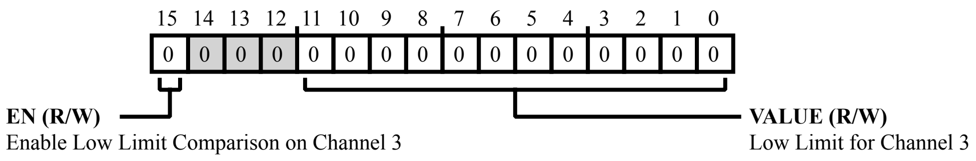


Figure 20-42: ADC_LIM3_LO Register Diagram

Table 20-33: ADC_LIM3_LO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	EN	Enable Low Limit Comparison on Channel 3.
11:0 (R/W)	VALUE	Low Limit for Channel 3.

Overflow of Output Registers

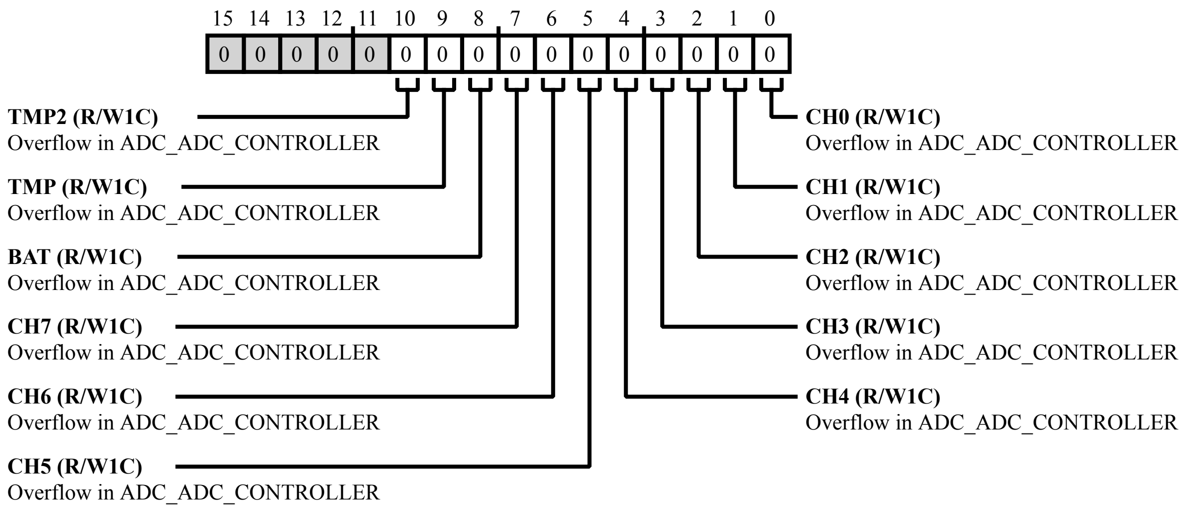

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TMP2 (R/W1C)
Overflow in ADC_ADC_CONTROLLER
TMP (R/W1C)
Overflow in ADC_ADC_CONTROLLER
BAT (R/W1C)
Overflow in ADC_ADC_CONTROLLER
CH7 (R/W1C)
Overflow in ADC_ADC_CONTROLLER
CH6 (R/W1C)
Overflow in ADC_ADC_CONTROLLER
CH5 (R/W1C)
Overflow in ADC_ADC_CONTROLLER
CH0 (R/W1C)
Overflow in ADC_ADC_CONTROLLER
CH1 (R/W1C)
Overflow in ADC_ADC_CONTROLLER
CH2 (R/W1C)
Overflow in ADC_ADC_CONTROLLER
CH3 (R/W1C)
Overflow in ADC_ADC_CONTROLLER
CH4 (R/W1C)
Overflow in ADC_ADC_CONTROLLER

Figure 20-43: ADC_OVF Register Diagram

Table 20-34: ADC_OVF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/W1C)	TMP2	Overflow in ADC_TMP2_OUT.
9 (R/W1C)	TMP	Overflow in ADC_TMP_OUT.
8 (R/W1C)	BAT	Overflow in ADC_BAT_OUT. Indicates overflow in battery monitoring output register.
7 (R/W1C)	CH7	Overflow in ADC_CH7_OUT.
6 (R/W1C)	CH6	Overflow in ADC_CH6_OUT.
5 (R/W1C)	CH5	Overflow in ADC_CH5_OUT.
4 (R/W1C)	CH4	Overflow in ADC_CH4_OUT.
3 (R/W1C)	CH3	Overflow in ADC_CH3_OUT.

Table 20-34: ADC_OVF Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	CH2	Overflow in ADC_CH2_OUT.
1 (R/W1C)	CH1	Overflow in ADC_CH1_OUT.
0 (R/W1C)	CH0	Overflow in ADC_CH0_OUT.

ADC Power-up Time

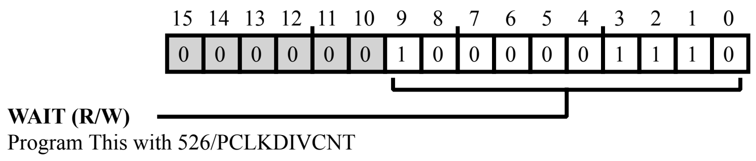


Figure 20-44: ADC_PWRUP Register Diagram

Table 20-35: ADC_PWRUP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	WAIT	Program this with 526/PCLKDIVCNT.

ADC Status

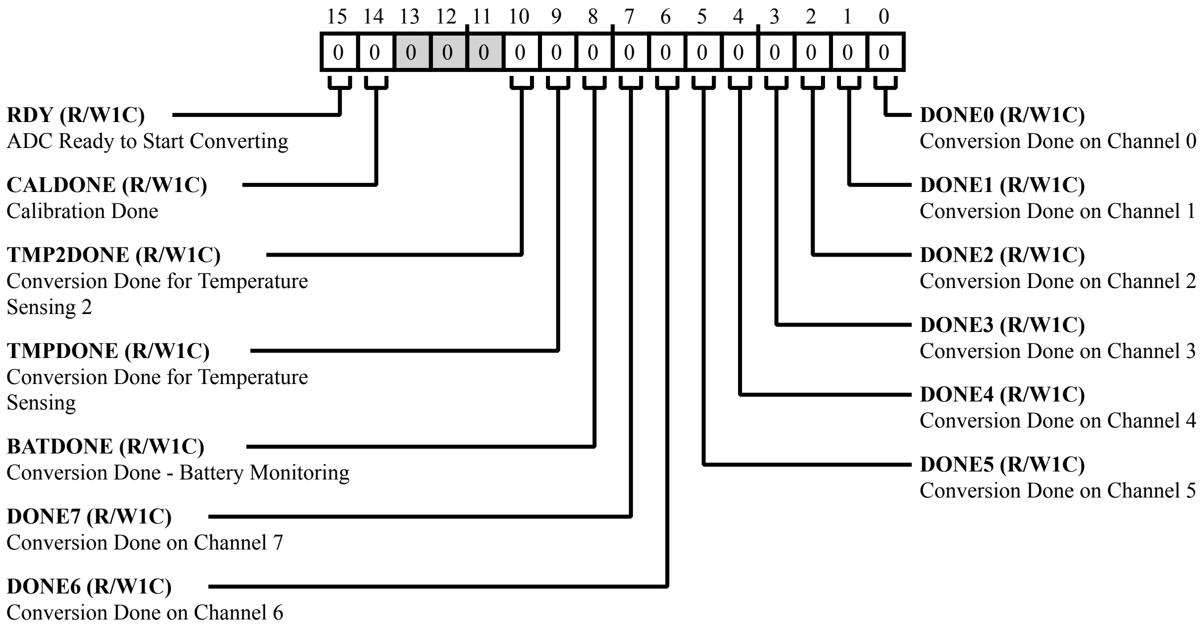

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RDY (R/W1C)
ADC Ready to Start Converting
CALDONE (R/W1C)
Calibration Done
TMP2DONE (R/W1C)
Conversion Done for Temperature Sensing 2
TMPDONE (R/W1C)
Conversion Done for Temperature Sensing
BATDONE (R/W1C)
Conversion Done - Battery Monitoring
DONE7 (R/W1C)
Conversion Done on Channel 7
DONE6 (R/W1C)
Conversion Done on Channel 6
DONE0 (R/W1C)
Conversion Done on Channel 0
DONE1 (R/W1C)
Conversion Done on Channel 1
DONE2 (R/W1C)
Conversion Done on Channel 2
DONE3 (R/W1C)
Conversion Done on Channel 3
DONE4 (R/W1C)
Conversion Done on Channel 4
DONE5 (R/W1C)
Conversion Done on Channel 5

Figure 20-45: ADC_STAT Register Diagram

Table 20-36: ADC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	RDY	ADC Ready to Start Converting. Indicates ADC is ready to start converting, when using external reference buffer.
14 (R/W1C)	CALDONE	Calibration Done. Indicates calibration is done.
10 (R/W1C)	TMP2DONE	Conversion Done for Temperature Sensing 2.
9 (R/W1C)	TMPDONE	Conversion Done for Temperature Sensing.
8 (R/W1C)	BATDONE	Conversion Done - Battery Monitoring. Indicates conversion done for battery monitoring.
7 (R/W1C)	DONE7	Conversion Done on Channel 7.
6 (R/W1C)	DONE6	Conversion Done on Channel 6.

Table 20-36: ADC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W1C)	DONE5	Conversion Done on Channel 5.
4 (R/W1C)	DONE4	Conversion Done on Channel 4.
3 (R/W1C)	DONE3	Conversion Done on Channel 3.
2 (R/W1C)	DONE2	Conversion Done on Channel 2.
1 (R/W1C)	DONE1	Conversion Done on Channel 1.
0 (R/W1C)	DONE0	Conversion Done on Channel 0.

Temperature Result 2

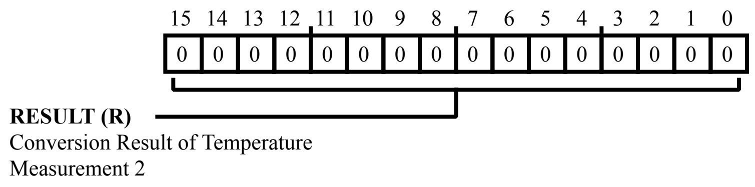


Figure 20-46: ADC_TMP2_OUT Register Diagram

Table 20-37: ADC_TMP2_OUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	RESULT	Conversion Result of Temperature Measurement 2.

Temperature Result

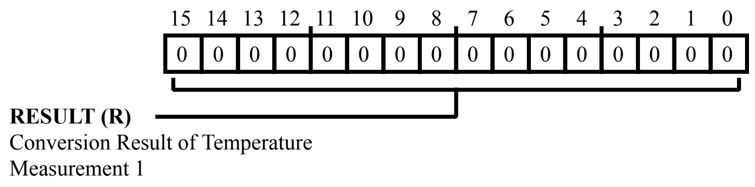


Figure 20-47: ADC_TMP_OUT Register Diagram

Table 20-38: ADC_TMP_OUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	RESULT	Conversion Result of Temperature Measurement 1. Conversion result of Temperature measurement 1 is stored here.

21 Real-Time Clock (RTC)

The ADuCM302x MCU uses the real-time clock (RTC) that keeps track of elapsed time, in configurable time units, using an externally attached 32,768 Hz crystal to generate a time base. When enabled, the RTC maintains a cumulative count of elapsed time from the counts most recent redefinition (re-initialization) by the CPU or power-up value as applicable. The RTC can count from any configured initial value and roll-overs of the RTC count are supported. The RTC is a 16-bit peripheral, but contains triple registers for any 47-bit quantities such as the elapsed time count and alarm values.

The RTC has two configurable alarm features which permit the generation of absolute (exact time match) or periodic (every 60 increments of the RTC count) alarms.

Software is responsible for enabling and configuring the RTC and for interpreting its count value to turn this into the time of day. The RTC logic also has a digital trim capability that is calibrated to achieve higher PPM accuracy in tracking time.

RTC Features

The ADuCM302x MCU supports the following features:

- An RTC count register with 32 integer bits and 15 fractional bits of elapsed time in configurable time units from a programmable reference point (initial value). This register is programmed under software control.
    - The RTC can count time in units of a divided-down period of the RTC base clock (nominally 32,768 Hz), where the division can be any power of two from zero to fifteen. The range of RTC time units is thus 30.52 µs to 1s.
- A prescaler that divides down the RTC base clock (nominally a 32,768 Hz crystal input) by a power of two which can be any integer from 0 to 15. Note that when programming (initializing) or re-enabling the RTC count, or when changing the prescale division ratio, the prescaler is automatically zeroed. This is so that the RTC count value is positioned on exact, coincident boundaries of both the start of the prescale sequence and the modulo-60 count roll over.
    - CPU redefinition of the RTC count (elapsed time) can be deposited on coincident 1 second and 1 minute boundaries, when using a time base of 1 second. The same capability is supported by the RTC for any prescaled time base.

- Two optionally enabled, independent alarm features (one at absolute time and the other at modulo-60, periodic time) that cause a processor interrupt when the RTC count equals the alarm values. Note that such an interrupt wakes up the processor if the latter is in a sleep state.
- A digital trim capability whereby a positive or negative adjustment (in units of configured RTC time units) can be added to the RTC count at a fixed interval to keep the RTC PPM time accuracy within target. The values for both the adjustment and interval are calculated by running a calibration algorithm on the CPU.
- The RTC can take and preserve a snapshot of its elapsed real-time count when prompted to do so by the CPU. This allows the CPU to associate a time stamp with an incoming data packet. The RTC preserves the snapshot for read back by the CPU. The snapshot is persistent and is only overwritten when the CPU issues a request to capture a new value.
- RTC has a SensorStrobe mechanism, which is an alarm function. It sends an output pulse to an external device via GPIO, and instructs the device to measure or perform some action at a specific time. The SensorStrobe events are scheduled by the CPU on the ADuCM302x MCU by instructing the RTC to activate the SensorStrobe events at a specific target time relative to the RTC real-time count.
- RTC has an input capture feature. Input capture is the process of taking a snapshot of the RTC real-time count when an external device signals an event via a transition on one of the GPIO inputs to the ADuCM302x MCU. An input capture event is triggered by an autonomous measurement or action on a device, which then signals the ADuCM302x MCU that the RTC must take a snapshot of time corresponding to the event.

RTC Functional Description

This section provides information on the function of the RTC used by the ADuCM302x MCU.

RTC Block Diagram

A high-level block diagram of the RTC is shown below. All functionality for counting, alarm, trim, snapshot and wake-up interrupts is located in a dedicated 32 kHz timed, always On RTC power domain. The APB interface with the CPU, which comprises queuing and dispatch logic for posted register writes, along with interrupts to the Cortex NVIC are all located in a PCLK/FCLK timed (synchronous clocks) section of the main power gated core domain.

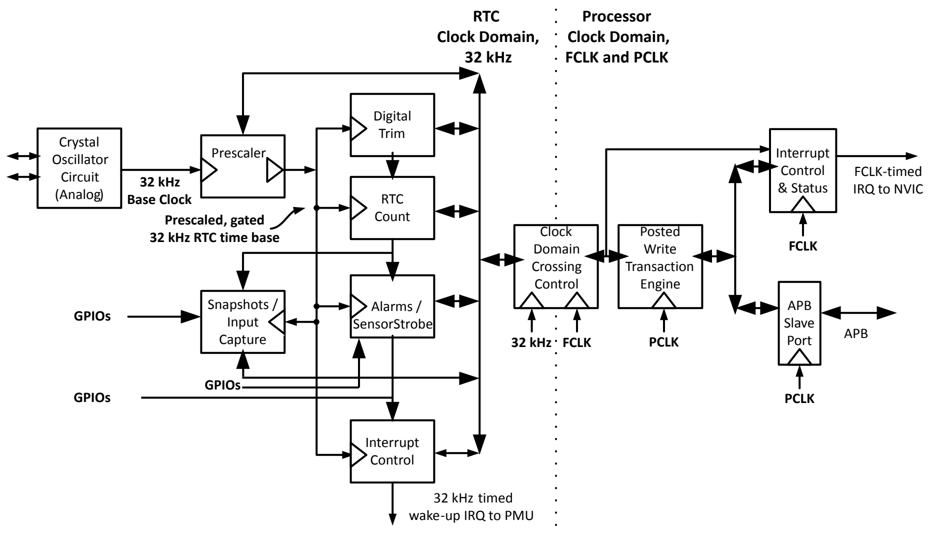

RTC Clock Domain, 32 kHz
Processor Clock Domain, FCLK and PCLK
Crystal Oscillator Circuit (Analog)
32 kHz Base Clock
Prescaler
Prescaled, gated 32 kHz RTC time base
Digital Trim
RTC Count
Snapshots / Input Capture
Alarms / SensorStrobe
GPIOs
GPIOs
GPIOs
Interrupt Control
32 kHz timed wake-up IRQ to PMU
Clock Domain Crossing Control
32 kHz
FCLK
Posted Write Transaction Engine
PCLK
Interrupt Control & Status
FCLK-timed IRQ to NVIC
FCLK
APB Slave Port
APB
PCLK

Figure 21-1: RTC High-level Block Diagram

The key use of the RTC is to provide the time keeping function and maintain the time and date in an accurate and reliable manner with minimal power consumption. In addition to time keeping, it also provides the stopwatch and alarm features. The RTC uses the internal counters to keep the time of the day in terms of seconds, minutes, hours, and days. This data is sufficient for the user application to extract the date and time information from RTC. Interrupts can be issued periodically.

RTC Operating Modes

The RTC used by the ADuCM302x MCU supports the following operations.

Initial RTC Power-Up

The RTC operates in a dedicated voltage domain which, under normal (configurable) conditions, is continuously powered. However, when a battery is attached for the first time or replaced, a power-on reset occurs which resets all RTC registers.

Upon detecting an RTC failure, the CPU reprograms the RTCs count and digital trim registers and clear the fail flag in the RTC control register. The CPU can optionally program the alarm registers of the RTC to generate an interrupt when the alarm and count values match.

Persistent, Sticky RTC Wake-Up Events

Note that there is no loss of any RTC alarm event which happens when the part is in a power down mode. The resulting interrupt due to the alarm, assuming it is enabled, is maintained asserted by the RTC so that the NVIC will subsequently see it (an FCLK-timed version of the interrupt) when power is restored to the processor. To facilitate this, the RTC sends a 32 kHz-timed version of the same interrupt to the wake-up controller in the PMU which causes the digital core to be repowered. Once the CPU is woken up, it can inspect both the PMU and RTC to understand the cause of the interrupt event for the wake-up.

RTC Capacity to Accommodate Posted Writes by CPU

If a posted write by the CPU to a 32 kHz-sourced MMR in the RTC is pending dispatch (in the RTC) due to a queue of other, similar register writes to the 32 Hz domain, a second or subsequent write by the CPU to the same RTC register cannot be stacked up or overwrite the pending transaction. Any such attempts will be rejected by the RTC. These result in `RTC_SR0.WPNDERRINT` interrupt events in the RTC (see the MMR details of `RTC_SR0`).

Realignment of RTC Count to Packet Defined Time Reference

The CPU can instruct the RTC to take a snapshot of its elapsed time count by writing a software key of 0x7627 to the `RTC_GWY` MMR. This causes the combined three snap registers (`RTC_SNAP2`, `RTC_SNAP1`, and `RTC_SNAP0`) to update to the current value of the three count registers (`RTC_CNT2`, `RTC_CNT1`, and `RTC_CNT0`) and to maintain this snapshot until subsequently told by the CPU to overwrite it.

RTC Recommendations: Clocks and Power

The following recommendations apply for using the RTC.

Stopping PCLK

Before entering any mode which causes PCLK to stop, the CPU should first wait until there is confirmation from the RTC that no previously posted writes have yet to complete. The CPU can check this by reading both the `RTC_SR0` and `RTC_SR2` registers.

Ensuring No Communication across RTC Power Boundary when Powering Down

When the CPU has advance knowledge about a power down, it must take the following action to ensure the integrity of always-on half of the RTC.

- Either :

  The CPU must check and satisfy itself that there are no posted writes in the RTC awaiting execution.

- Or :

  Cancel all queued and executing posted writes in the RTC. This is achieved by writing a cancellation key of 0xA2C5 to the `RTC_GWY` register which takes immediate effect.

- And :

  Do not post any further register writes to the RTC until power is lost by the core.

These steps ensure that no communication between the CPU and the RTC is happening and thus liable to corruption when the RTC power domains isolation barrier is subsequently activated.

RTC Interrupts and Exceptions

The RTC block can generate interrupts from multiple sources which can be unmasked by programming the Control register. The source of the interrupt is reflected in the Status register.

RTC Programming Model

The following section shows the programming sequence to configure RTC for an alarm event.

Programming Guidelines

The following are the programming guidelines:

1. Reset the `RTC_CNT` registers to 0.
2. Configure the prescaler to divide the RTC base clock in the `RTC_CR1` register.
3. Poll for the RTC synchronization bit to be set in the `RTC_SR1` register, as the MMR write happens in the slower RTC domain.
4. Program the `RTC_ALM0`, `RTC_ALM1`, `RTC_ALM2` registers with the intended alarm time.
5. Enable the interrupt for alarm by setting the `RTC_CR0.ALMINTEN` bit.
6. Set the `RTC_CR0.ALMEN` and `RTC_CR0.CNTEN` bits.
7. Wait for the RTC alarm interrupt which is triggered when the `RTC_CNT` matches with the `RTC_ALM` value.

RTC SensorStrobe

SensorStrobe is an alarm mechanism in the RTC. It sends an output pulse to an external device via GPIO, and instructs the device to measure or perform some action at a specific time. The SensorStrobe events are scheduled by the CPU on the ADuCM302x MCU by instructing the RTC to activate the SensorStrobe events at a specific target time relative to the RTC real-time count.

SensorStrobe channel prompts the external sensors to perform a periodic action at scheduled alarm time. The alarm time is programmed in the RTC by the CPU, typically before going into hibernate mode. When a SensorStrobe event occurs, the RTC can optionally cause the CPU to be woken up and interrupted to examine the results of the sensor stimulated via the GPIO.

The real-time clock (RTC1) on the ADuCM302x MCU has two SensorStrobe channels—SS0 (alarm only) and SS1. These channels act as alarms scheduled in the RTC by the CPU. When a SensorStrobe event (alarm) occurs, the channel concerned asserts an output control line high for one 32 kHz cycle. This output is routed through a fixed GPIO pin for the channel concerned and exported off chip.

Currently, the ADuCM302x MCU has GPIO support for SS1. SS0 is the 47-bit alarm feature of the RTC without GPIO connectivity.

The figure shows the SensorStrobe channels and how they are align to the main RTC count.

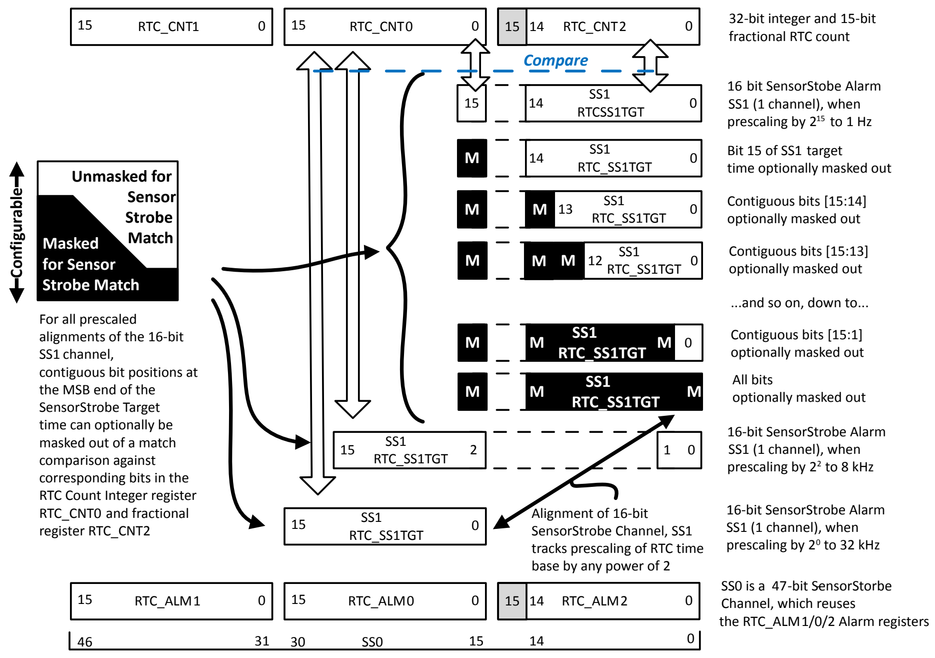

15 RTC_CNT1 0
15 RTC_CNT0 0
15 14 RTC_CNT2 0
32-bit integer and 15-bit fractional RTC count
Compare
15 14 SS1 RTCSS1TGT 0
16 bit SensorStrobe Alarm SS1 (1 channel), when prescaling by 2^{15} to 1 Hz
M 14 SS1 RTC_SS1TGT 0
Bit 15 of SS1 target time optionally masked out
M M 13 SS1 RTC_SS1TGT 0
Contiguous bits [15:14] optionally masked out
M M M 12 SS1 RTC_SS1TGT 0
Contiguous bits [15:13] optionally masked out
...and so on, down to...
M M SS1 RTC_SS1TGT M 0
Contiguous bits [15:1] optionally masked out
M M SS1 RTC_SS1TGT M
All bits optionally masked out
Configurable
Unmasked for Sensor Strobe Match
Masked for Sensor Strobe Match
For all prescaled alignments of the 16-bit SS1 channel, contiguous bit positions at the MSB end of the SensorStrobe Target time can optionally be masked out of a match comparison against corresponding bits in the RTC Count Integer register RTC_CNT0 and fractional register RTC_CNT2
15 SS1 RTC_SS1TGT 2
1 0
16-bit SensorStrobe Alarm SS1 (1 channel), when prescaling by 2^2 to 8 kHz
15 SS1 RTC_SS1TGT 0
Alignment of 16-bit SensorStrobe Channel, SS1 tracks prescaling of RTC time base by any power of 2
16-bit SensorStrobe Alarm SS1 (1 channel), when prescaling by 2^0 to 32 kHz
15 RTC_ALM1 0
15 RTC_ALM0 0
15 14 RTC_ALM2 0
SS0 is a 47-bit SensorStorbe Channel, which reuses the RTC_ALM1/0/2 Alarm registers
46 31 30 SS0 15 14 0

Figure 21-2: RTC SensorStrobe Channel

The 47-bit SensorStrobe channel (SS0) is a synonym of the 47-bit alarm feature in the RTC defined by the `RTC_ALM1`, `RTC_ALM0`, and `RTC_ALM2` registers. This is an absolute-time alarm, unmaskable in any of its bit positions and whose time span before the alarm repeats is equal to the length of the RTC count.

To use SS0 for frequent SensorStrobe activations, the CPU must reprogram the alarm time in the `RTC_ALM1`/`RTC_ALM0`/`RTC_ALM2` registers each time it is woken up and interrupted by the RTC upon an SS0 event. SS0 does not have GPIO activation associated with it.

The 16-bit SensorStrobe channel (SS1) has a shorter time span, but more versatile than SS0. SS1 aligns the 16 usable bits for prescaling the RTC integer count time base. It masks the contiguous MSB bits of its target time to reduce the periodicity of repeating alarms. It supports optional auto-reloading for alarm periods which are not powers of 2 with respect to the 32 kHz clock period. SS1 has a fixed GPIO associated with it to export a one-cycle activation of its SensorStrobe line.

RTC SensorStrobe Channels figure shows three example degrees of prescaling (2^{15}, 2^2, and 2^0) of the RTC time base. It shows how the 16 bits of the alarm time for SS1 align with the degree of prescaling of the 32 kHz clock.

All the fractional bits in the `RTC_CNT2` register consistent with the configurable prescaling are compared with the corresponding bit positions of the alarm time specified by SS1 when checking for an SensorStrobe match. The remaining bits are compared with the LSB end of the `RTC_CNT0` register. This self-aligning of SS1 with the `RTC_CNT2` and `RTC_CNT0` registers based on prescaling is supported for all prescale powers of 2 between 0 and 15.

SensorStrobe for two types of periodicity of alarms (SensorStrobe events):

- ***Periods which are a power of 2 with respect to the 32 kHz clock***: Comparisons are made repeatedly between the value in SS1 and bits in the RTC count, suitably aligned for prescaling.
- ***Periods which are not a power of 2***: Comparison is done by auto-reloading (cumulatively adding to the existing content in SS1) an offset value in `RTC_SS1ARL` into SS1 every time a SensorStrobe event occurs.

The target time for an upcoming event on the SS1 channel can be read by the CPU via the `RTC_SS1TGT` register. For SensorStrobe that does not use auto reloading, the live value in the `RTC_SS1TGT` register is same as the value of SS1, as the step size between events is defined by SS1. When auto-reloading is used, the live value in the `RTC_SS1TGT` register reflects the repeated, cumulative, modulo-16 addition of `RTC_SS1ARL` to a starting value of SS1. `RTC_SS1TGT` always allows the CPU to know when a SensorStrobe event is due, in terms of a match between `RTC_SS1TGT` and RTC count.

All types of SensorStrobes on the SS1 channel can be partially masked. Masking allows shortening of the period of recurrence of SensorStrobe events by considering the bit positions of the target time and RTC count as Don't Cares. When a 16-bit SensorStrobe match check is carried out for SS1 against the RTC count, contiguous bits at the MSB end of SS1 and `RTC_SS1TGT` can optionally be masked, thereby becoming Don't Cares (shown as M-bit positions in the figure). The number of contiguous bits masked out in SS1 and `RTC_SS1TGT` is specified by the `RTC_SSMSK`.

The following are the SensorStrobe capabilities of the hibernate real-time clock (RTC1) on the ADuCM302x MCU:

- Supports one SensorStrobe channel (SS1) with GPIO activation:
  - 16-bit channel. The target time for the 16-bit channel is specified as:

    ```
    {least_significant_integer_bits_to_bring_total_to_16,
    fractional_bits_due_to_prescaling}
    ```

    The total number of bits in this concatenated capture time is 16.

    The number of fractional bits in the target depends on the degree of prescaling configured for the 32 kHz base clock. The number of fractional bits used in the input capture function tracks with the prescaling. The remaining bits in the 16-bit target time consist of integers from the LSB end of the integer count of the RTC. The number of integer bits combined with the fractional bits (due to prescaling) must be 16.
  - There is a 47-bit alarm feature in the RTC. However, there is no GPIO associated with it to act as a SensorStrobe channel. The target time for this 47-bit alarm is specified as:

    ```
    {32_integer_bits, 15_fractional_bits}
    ```
- For SensorStrobe channel, the target time is always specified down to an individual 32 kHz clock cycle as all relevant fractional bits are included in the target time.
- For the 16-bit SensorStrobe channel (SS1), contiguous bits in the 16-bit target time can be thermometer-code masked out, on a per-channel basis, from the MSB end of the target.

As a result of this optional masking, the contiguous bits at the MSB end of the target time are treated as Don't Cares, which results in dividing the periodicity of the repeating SensorStrobe event by 2 for every bit masked out.

In other words, if more bits are masked out from the MSB end of the target time, the modular sequence of repeating SensorStrobe events is shortened by a factor of 2 for that channel.

- Auto Reload: Each time an event triggers, a configurable 16-bit delta (the reload value) is added to its target time to calculate the revised target for the next event. This allows the periodicity of repeating events to be created on this channel which is not a power of two with respect to the 32 kHz clock.

  The reloaded target time can be read by the CPU to confirm the accumulation due to reloading with each event on that SensorStrobe channel. The reloaded target can also be contiguously masked from the MSB end.
- The SensorStrobe channel has a readable, sticky interrupt source bit, which activates (sticks high) whenever a SensorStrobe event occurs. This interrupt source bit can optionally be enabled to fan into the interrupt and wake-up lines from the RTC.
- When a SensorStrobe event triggers on SS1, a one-cycle pulse of length 32 kHz period is exported by the RTC via the GPIOs, to an external device, prompting the device to act.

  The exported pulses are on a dedicated-channel basis from the RTC to external device connected to the channel. There is no broadcast function, where a SensorStrobe event on one channel can be routed to all SensorStrobe channels and external devices.
- The SensorStrobe channel can be reconfigured and enabled/disabled on the fly, with no interruption to channels that are not part of the reconfiguration.
- A SensorStrobe channel is independent of the input capture channels in the RTC.

RTC SensorStrobe Channel Waveforms show how the SensorStrobe for the SS1 channel works in the RTC.

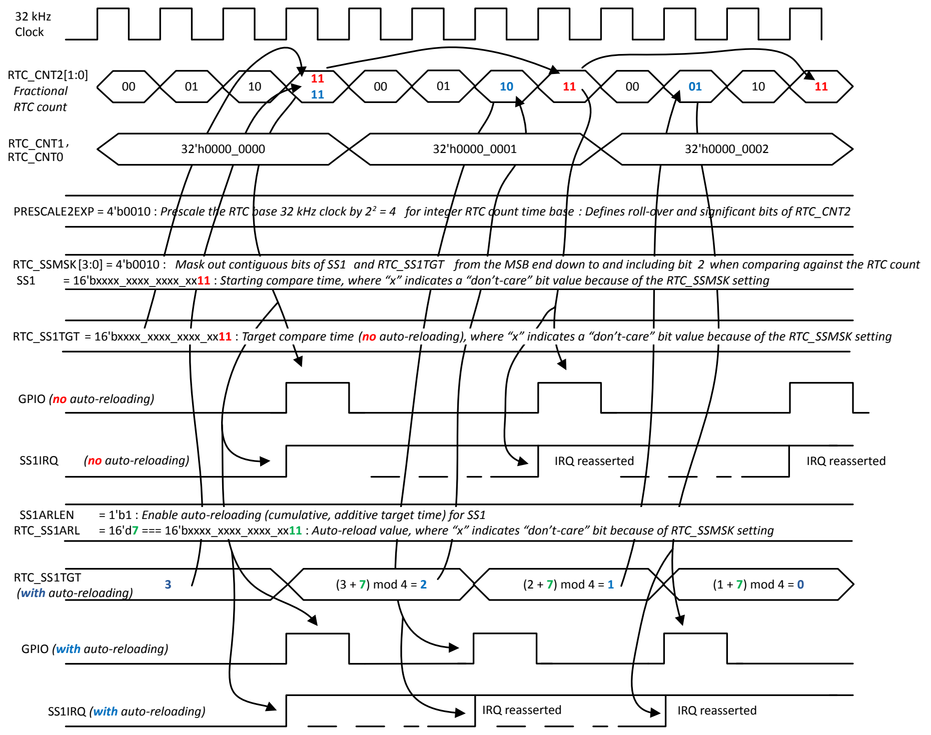

32 kHz Clock
RTC_CNT2[1:0] Fractional RTC count
00
01
10
11 11
00
01
10
11
00
01
10
11
RTC_CNT1, RTC_CNT0
32'h0000_0000
32'h0000_0001
32'h0000_0002
PRESCALE2EXP = 4'b0010 : Prescale the RTC base 32 kHz clock by 2² = 4 for integer RTC count time base : Defines roll-over and significant bits of RTC_CNT2
RTC_SSMSK[3:0] = 4'b0010 : Mask out contiguous bits of SS1 and RTC_SS1TGT from the MSB end down to and including bit 2 when comparing against the RTC count
SS1 = 16'bxxxx_xxxx_xxxx_xx11 : Starting compare time, where "x" indicates a "don't-care" bit value because of the RTC_SSMSK setting
RTC_SS1TGT = 16'bxxxx_xxxx_xxxx_xx11 : Target compare time (no auto-reloading), where "x" indicates a "don't-care" bit value because of the RTC_SSMSK setting
GPIO (no auto-reloading)
SS1IRQ (no auto-reloading)
IRQ reasserted
IRQ reasserted
SS1ARLEN = 1'b1 : Enable auto-reloading (cumulative, additive target time) for SS1
RTC_SS1ARL = 16'd7 === 16'bxxxx_xxxx_xxxx_xx11 : Auto-reload value, where "x" indicates "don't-care" bit because of RTC_SSMSK setting
RTC_SS1TGT (with auto-reloading)
3
(3 + 7) mod 4 = 2
(2 + 7) mod 4 = 1
(1 + 7) mod 4 = 0
GPIO (with auto-reloading)
SS1IRQ (with auto-reloading)
IRQ reasserted
IRQ reasserted

Figure 21-3: RTC SensorStrobe Channel Waveforms

The figure shows the occurrence of compare events when the RTC prescales the base 32 kHz clock by 2^2 (giving two meaningful, fractional bits in the `RTC_CNT2` register). SensorStrobe is supported for all prescaling powers of 2 between 0 and 15, selected via `RTC_CR1.PRESCALE2EXP`.

The settings in red show a SensorStrobe event occurring on the SS1 channel for every four cycles of the 32 kHz clock. The event has a period which is a power of two cycles due to the absence of auto reloading, disabled via `RTC_CR4SS.SS1ARLEN`. The value in the SS1 register defines the target time. Only two least significant bits of SS1 are compared with the RTC count due to the mask setting in the `RTC_SSMSK` register. The number of unmasked bits (2) defines the periodicity of the compare events, as the unmasked target time 2'b11 at the LSB end of SS1 is matched with every four increments of the `RTC_CNT2` register.

The settings in blue show an equivalent sequence of SensorStrobe events with auto-reloading enabled. The effects of the cumulative addition on the target time can be seen in the `RTC_SS1TGT` register. When auto reloading is enabled, the initial target time is the value of SS1, but on activation of each event, the value of `RTC_SS1ARL` is added to the content existing in the `RTC_SS1TGT` register, allowing roll overs in the addition. As the SensorStrobe match considers the masking specified in the `RTC_SSMSK` register, cumulative addition when auto-reloading undergoes modulo operation (modular addition). Modulo operation is $2^{\text{Number of unmasked bits in the comparison}}$.

The figure shows two unmasked LSBs of the auto reload value (2'b11) being added cumulatively to the unmasked bits of `RTC_SS1TGT`. The result is subjected to a modulo-4 operation. The period of the compare events is specified by the unmasked bits of the auto-reload value.

Masking and auto reloading are optional features of the RTC.

There is an automatic alignment to the LSB end of the RTC count by the 16 bits (considering prescaling) in the following:

- Initial target time in SS1
- Cumulative target time in `RTC_SS1TGT`
- Any possible mask decoded from `RTC_SSMSK`
- Any enabled reload value in `RTC_SS1ARL`

When a SensorStrobe event occurs, a GPIO output is activated for one 32 kHz cycle along with the assertion of a sticky interrupt source (`RTC_CR3SS.SS1IRQEN`), to record the occurrence of a new compare event since the CPU last cleared this interrupt source bit. The CPU can optionally enable `RTC_CR3SS.SS1IRQEN` to contribute to the activation of the RTC interrupt lines to the wake-up controller and NVIC. This is used to wake up the CPU from hibernate and examine the results from an external sensor prompted by an RTC SensorStrobe event to perform an action.

RTC Input Capture

Input capture is the process of taking a snapshot of the RTC real-time count when an external device signals an event via a transition on one of the GPIO inputs to the ADuCM302x MCU. An input capture event is triggered by an autonomous measurement or action on a device, which then signals to the ADuCM302x MCU that the RTC must take a snapshot of time corresponding to the event. Taking a snapshot can wake up the ADuCM302x MCU and interrupt the CPU. The CPU can subsequently obtain information from the RTC on the exact 32 kHz cycle and exact time of the input capture event.

The real-time clock (RTC1) on the ADuCM302x MCU has four input capture channels—RTCIC0, RTCIC2, RTCIC3, and RTCIC4. RTCIC0 is a 47-bit wide channel. RTCIC2, RTCIC3, and RTCIC4 are 16-bit input capture channels.

Each of these independent channels can take a snapshot of the RTC count when a rising edge or falling edge event (one of these edge types is selected per channel) occurs on a GPIO associated with that channel. The snapshots are used to time stamp the inputs from external sensors for subsequent examination and processing by the CPU (once woken up).

Input capture snapshots are accurate down to a single 32 kHz clock period for all prescaling by the RTC of this clock. Prescaling is the process of dividing the 32 kHz clock by any power of 2 (between 0 and 15), to create a slower time base and count the integer time in the `RTC_CNT1` and `RTC_CNT0` registers. Fractional time for this time base is simultaneously counted in the `RTC_CNT2` register in the units of 32 kHz clock periods. Each input capture channel has a fixed GPIO pin to prompt the snapshot of RTC time to be taken when an edge (rising or falling) occurs on the GPIO.

The RTCIC0 channel, unlike its counterparts, can also take the software initiated snapshot when the CPU writes a special key to the RTC.

The 16-bit input capture channels takes snapshots based on the degree of prescaling in the RTC for the integer count time base, and align themselves accordingly. The 16-bit snapshots contain fractional bits in `RTC_CNT2` register that are relevant for prescaling, and additional bits from the LSB end of the `RTC_CNT0` register.

RTCIC0 snapshots all the bits from the `RTC_CNT1`, `RTC_CNT0`, and `RTC_CNT2` registers. This gives a full, absolute-time snapshot for an event on the RTCIC0 channel. Other 16-bit input capture channels only have a time span (uniqueness of the snapshot) equivalent to 2^{16} periods of the 32 kHz clock.

The alignment of snapshots with the main RTC count is shown in the ***Input Capture Channels: 16-bit and 47-bit Snapshots of RTC Count*** figure. It contains three example degrees of prescaling (2^{15}, 2^2, and 2^0) of the RTC time base. All powers of 2 between 0 and 15 used in prescaling are supported for the 16-bit input capture channels.

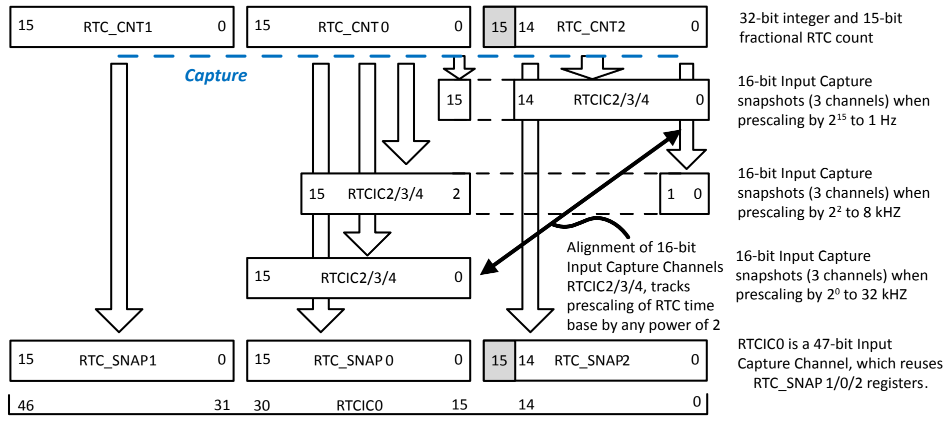

15 RTC_CNT1 0
15 RTC_CNT0 0
15 14 RTC_CNT2 0
32-bit integer and 15-bit fractional RTC count
Capture
15
14 RTCIC2/3/4 0
16-bit Input Capture snapshots (3 channels) when prescaling by 2^15 to 1 Hz
15 RTCIC2/3/4 2
1 0
16-bit Input Capture snapshots (3 channels) when prescaling by 2^2 to 8 kHZ
15 RTCIC2/3/4 0
Alignment of 16-bit Input Capture Channels RTCIC2/3/4, tracks prescaling of RTC time base by any power of 2
16-bit Input Capture snapshots (3 channels) when prescaling by 2^0 to 32 kHZ
15 RTC_SNAP1 0
15 RTC_SNAP0 0
15 14 RTC_SNAP2 0
RTCIC0 is a 47-bit Input Capture Channel, which reuses RTC_SNAP 1/0/2 registers.
46 31 30 RTCIC0 15 14 0

Figure 21-4: RTC Input Capture Channels: 16-bit and 47-bit Snapshots of RTC Count

Each input capture channel has an independent control of the following features and can be changed on-the-fly without affecting the other channels:

- Enable/disable channel
- Polarity of the active-going edge, rising or falling, of the GPIO to prompt snapshots
- Enable to interrupt upon a capture event
- Sticky interrupt source for a capture event
- Read/unread status of a capture snapshot by the CPU

A common configurable setting `RTC_CR2IC.RTCICOWUSEN` is applied to all the input capture channels. It is used to select between allowing new snapshots overwrite unread previous ones on a per-channel basis, and having the input capture snapshots stick until read by the CPU. While the same policy on overwriting snapshots applies to all channels, each channel enforces it independently.

The following are the input capture capabilities of RTC1 on the ADuCM302x MCU:

- Supports four independent input capture channel (three 16-bit channels and one 47-bit channel):
    - The three 16-bit channels snapshot the RTC elapsed count using 16 bits as follows:

```
{least_significant_integer_bits_to_bring_total_to_16,
fractional_bits_due_to_prescaling}
```

The total number of bits in this concatenated capture time is 16.

The number of fractional bits in the target depends on the degree of prescaling configured for the 32 kHz base clock. The number of fractional bits used in the input capture function tracks with the prescaling. The remaining bits in the 16-bit target time consist of integers from the LSB end of the integer count of the RTC. The number of integer bits combined with the fractional bits (due to prescaling) must be 16.

  - The 47-bit channel captures the absolute time for that input capture channel, where the capture time is given as `{32_integer_bits, 15_fractional_bits}`.

- For each of the four input capture channels, the snapshot always has a resolution down to an individual 32 kHz clock cycle as all relevant fractional bits are included in the capture time.
- When an input capture event is detected by the RTC, the accuracy of the snapshot time for the event is as follows:
  - For the three 16-bit input capture channels, each channel captures a snapshot which is exact in time down to the 32 kHz cycle in which the event occurred.
  - For the one 47-bit input capture channel, there is a fixed latency (delay) of one 32 kHz cycle of fractional time in the snapshot value captured for that event.

    This latency of one LSB of the fractional count can be subtracted later by the software.
- An input capture event can be configured as a low-to-high or high-to-low transition on the GPIO input for that channel. The polarity of this transition can be configured on a per-channel basis. The input capture channels can individually specify the type of transition that must cause an event for it.
- Each of the four input capture channels has a readable, sticky interrupt source bit, which activates (sticks high) whenever an input capture event occurs. This interrupt source bit can optionally be enabled to fan into (be a contributory term to) the interrupt and wake-up lines from the RTC.
- User can configure one of the following global overwrite policy for all the input capture channels in the RTC:
  - When a new input capture event occurs on a given channel, the new snapshot value overwrites the previously captured value for that individual channel.
  - For a given channel, snapshot can be overwritten by a new event only if the CPU has already read the existing snapshot value for that channel.

    NOTE: In addition to the readable interrupt sources, which stick active upon reception of new input capture events, the RTC also provides flags to the CPU confirming the read/unread status of the input capture snapshots. This aids the CPU if it is uses a policy of not allowing the unread snapshots to be overwritten by new events on the input capture channels.
- The input capture channels can be reconfigured and enabled/disabled on-the-fly, with no interruption to the channels not part of the reconfiguration.

- The input capture channels operate independently of each other and independently of SensorStrobe channel in the RTC.

RTC Input Capture Channels Waveforms show the input capture operation in the RTC. It shows the occurrence of the capture events while the RTC is prescales the 32 kHz base clock by 2^2 (giving two meaningful, fractional bits in RTC_CNT2), but the input capture is supported for all prescaling powers of two between 0 and 15 (selected via RTC_CR1.PRESCALE2EXP).

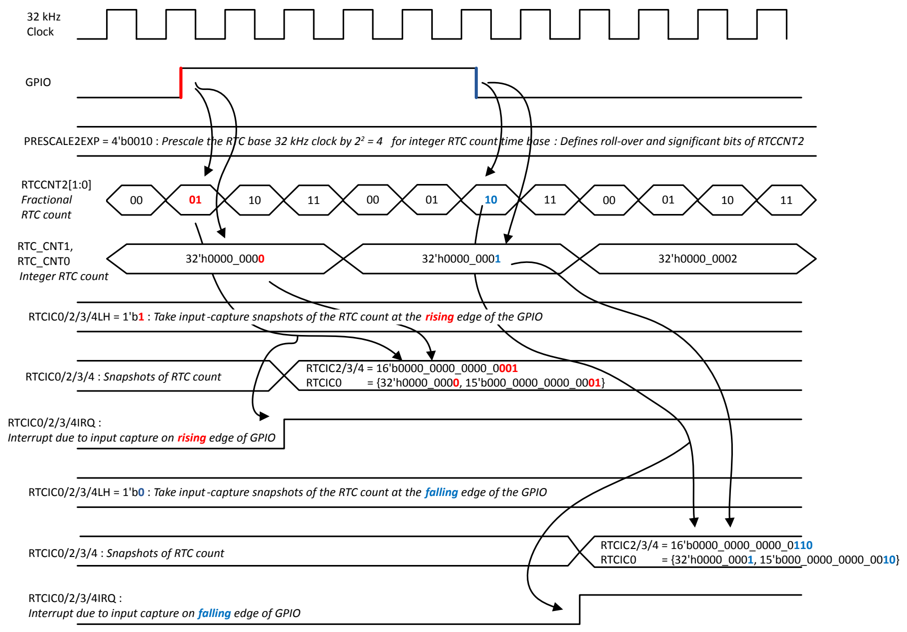

32 kHz Clock
GPIO
PRESCALE2EXP = 4'b0010 : Prescale the RTC base 32 kHz clock by 2² = 4 for integer RTC count time base : Defines roll-over and significant bits of RTCCNT2
RTCCNT2[1:0] Fractional RTC count
00
01
10
11
00
01
10
11
00
01
10
11
RTC_CNT1, RTC_CNT0 Integer RTC count
32'h0000_0000
32'h0000_0001
32'h0000_0002
RTCIC0/2/3/4LH = 1'b1 : Take input-capture snapshots of the RTC count at the rising edge of the GPIO
RTCIC0/2/3/4 : Snapshots of RTC count
RTCIC2/3/4 = 16'b0000_0000_0000_0001
RTCIC0 = {32'h0000_0000, 15'b000_0000_0000_0001}
RTCIC0/2/3/4IRQ : Interrupt due to input capture on rising edge of GPIO
RTCIC0/2/3/4LH = 1'b0 : Take input-capture snapshots of the RTC count at the falling edge of the GPIO
RTCIC0/2/3/4 : Snapshots of RTC count
RTCIC2/3/4 = 16'b0000_0000_0000_0110
RTCIC0 = {32'h0000_0001, 15'b000_0000_0000_0010}
RTCIC0/2/3/4IRQ : Interrupt due to input capture on falling edge of GPIO

Figure 21-5: RTC Input Capture Channels Waveforms

For a given input capture channel, the RTC_CR2IC.RTCIC2LH/RTC_CR2IC.RTCIC3LH/ RTC_CR2IC.RTCIC4LH fields allow the user to take the snapshot of the RTC time on the rising edge or falling edge of the GPIO (not both edges). When an input capture event occurs, a 16-bit input capture channel stores the current value of the 16 meaningful LSBs (accounting for prescaling) from the RTC_CNT2 and RTC_CNT0 registers necessary to identify the individual 32 kHz cycle in which the GPIO edge has occurred. The 47-bit input capture channel (RTCIC0) stores all the bits from the RTC_CNT1, RTC_CNT0, and RTC_CNT2 registers, giving its snapshot a longer span of uniquely identifying a 32 kHz cycle.

If RTC_CR2IC.RTCICOWUSEN does not allows overwriting of unread snapshots due to new GPIO edges, the snapshots of the RTC time are sticky (not overwritten) until read by the CPU. When a snapshot is taken, a sticky interrupt source (RTC_SR3.RTCIC0IRQ/RTC_SR3.RTCIC2IRQ/RTC_SR3.RTCOC3IRQ/ RTC_SR3.RTCIC4IRQ) goes high to record the occurrence of a new input capture event since the CPU last cleared the interrupt source bit. Optionally, the CPU can enable such sources to contribute to the interrupt lines from the RTC to the wake-up controller and NVIC.

Any number of channels, ranging from 0 to 5, of the four input capture and one SensorStrobe channels can operate simultaneously.

ADuCM302x RTC Register Descriptions

Real-Time Clock (RTC) contains the following registers.

Table 21-1: ADuCM302x RTC Register List

Name	Description
RTC_ALM0	RTC Alarm 0
RTC_ALM1	RTC Alarm 1
RTC_ALM2	RTC Alarm 2
RTC_CNT0	RTC Count 0
RTC_CNT1	RTC Count 1
RTC_CNT2	RTC Count 2
RTC_CR0	RTC Control 0
RTC_CR1	RTC Control 1
RTC_CR2IC	RTC Control 2 for Configuring Input Capture Channels
RTC_CR3SS	RTC Control 3 for Configuring SensorStrobe Channel
RTC_CR4SS	RTC Control 4 for Configuring SensorStrobe Channel
RTC_FRZCNT	RTC Freeze Count
RTC_GWY	RTC Gateway
RTC_IC2	RTC Input Capture Channel 2
RTC_IC3	RTC Input Capture Channel 3
RTC_IC4	RTC Input Capture Channel 4
RTC_MOD	RTC Modulo
RTC_SS1	RTC SensorStrobe Channel 1
RTC_SS1ARL	RTC Auto-Reload for SensorStrobe Channel 1
RTC_SS1TGT	RTC SensorStrobe Channel 1 Target
RTC_SSMSK	RTC Mask for SensorStrobe Channel
RTC_SNAP0	RTC Snapshot 0
RTC_SNAP1	RTC Snapshot 1
RTC_SNAP2	RTC Snapshot 2
RTC_SR0	RTC Status 0
RTC_SR1	RTC Status 1
RTC_SR2	RTC Status 2
RTC_SR3	RTC Status 3
RTC_SR4	RTC Status 4

Table 21-1: ADuCM302x RTC Register List (Continued)

Name	Description
RTC_SR5	RTC Status 5
RTC_SR6	RTC Status 6
RTC_TRM	RTC Trim

RTC Alarm 0

`RTC_ALM0` contains the lower 16 bits of the non-fractional (prescaled) RTC alarm target time value .

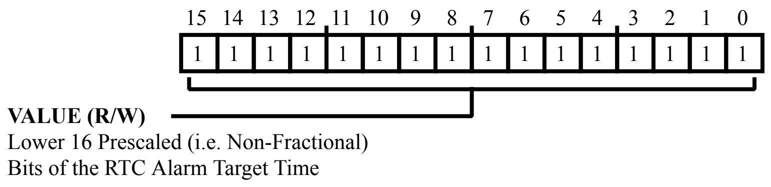


Figure 21-6: RTC_ALM0 Register Diagram

Table 21-2: RTC_ALM0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Lower 16 Prescaled (i.e. Non-Fractional) Bits of the RTC Alarm Target Time. Note that the alarm register has a different reset value to the RTC count to avoid spurious alarms.

RTC Alarm 1

RTC_ALM1 contains the upper 16 bits of the non-fractional (prescaled) RTC alarm target time value.

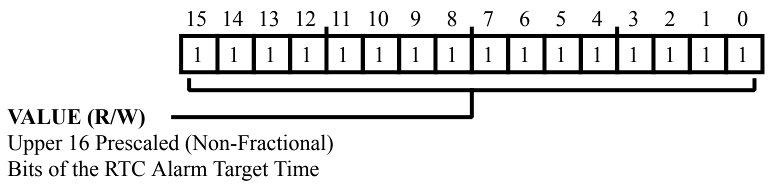


Figure 21-7: RTC_ALM1 Register Diagram

Table 21-3: RTC_ALM1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Upper 16 Prescaled (Non-Fractional) Bits of the RTC Alarm Target Time. Note that the alarm register has a different reset value to the RTC count to avoid spurious alarms.

RTC Alarm 2

RTC_ALM2 specifies the fractional (non-prescaled) bits of the RTC alarm target time value, down to an individual 32 kHz clock cycle.

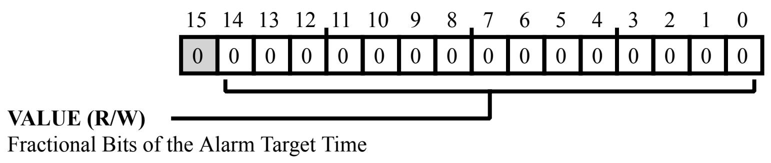


Figure 21-8: RTC_ALM2 Register Diagram

Table 21-4: RTC_ALM2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/W)	VALUE	Fractional Bits of the Alarm Target Time. Fractional (non-prescaled) bits of the RTC alarm target time.

RTC Count 0

RTC_CNT0 contains the lower 16 bits of the RTC counter which maintains a real-time count in elapsed prescaled RTC time units.

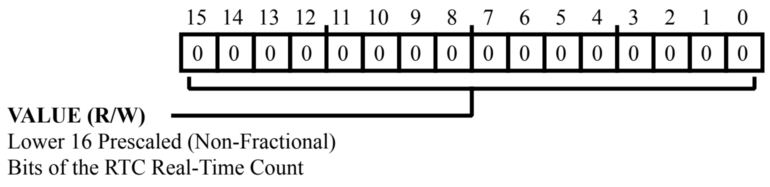


Figure 21-9: RTC_CNT0 Register Diagram

Table 21-5: RTC_CNT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Lower 16 Prescaled (Non-Fractional) Bits of the RTC Real-Time Count.

RTC Count 1

RTC_CNT1 contains the upper 16 bits of the RTC counter, which maintains a real-time count in elapsed prescaled RTC time units.

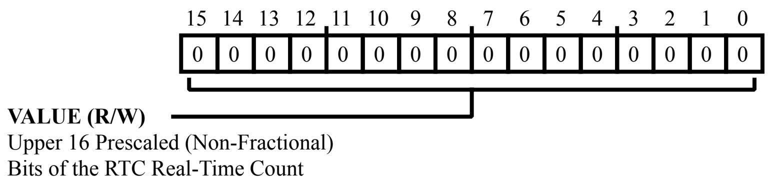


Figure 21-10: RTC_CNT1 Register Diagram

Table 21-6: RTC_CNT1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Upper 16 Prescaled (Non-Fractional) Bits of the RTC Real-Time Count.

RTC Count 2

RTC_CNT2 contains the fractional part of the RTC count. The overall resolution of the real-time count, including the fractional bits in RTC_CNT2, is one 32 kHz clock period.

Note that the RTC_CNT2 register only exists in RTC1. In RTC0, the fractional part of the RTC count cannot be read by the CPU.

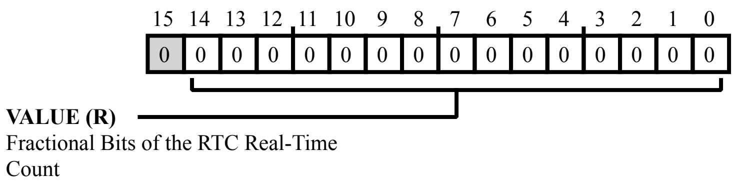


Figure 21-11: RTC_CNT2 Register Diagram

Table 21-7: RTC_CNT2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/NW)	VALUE	Fractional Bits of the RTC Real-Time Count. RTC_CNT2 is zeroed when one of the following events occurs: (i) The CPU writes a new pair of values to the RTC_CNT1 and RTC_CNT0 registers to redefine the elapsed time units count while the RTC is enabled and the posted twin write is executed. (ii) The CPU enables the RTC from a disabled state using the RTC_CR0.CNTEN field. (iii) When the RTC is enabled, the degree of prescaling in the RTC is changed via the RTC_CR1.PRESCALE2EXP field.

RTC Control 0

RTC_CR0 is the primary of two control registers for the RTC, the other being RTC_CR1.

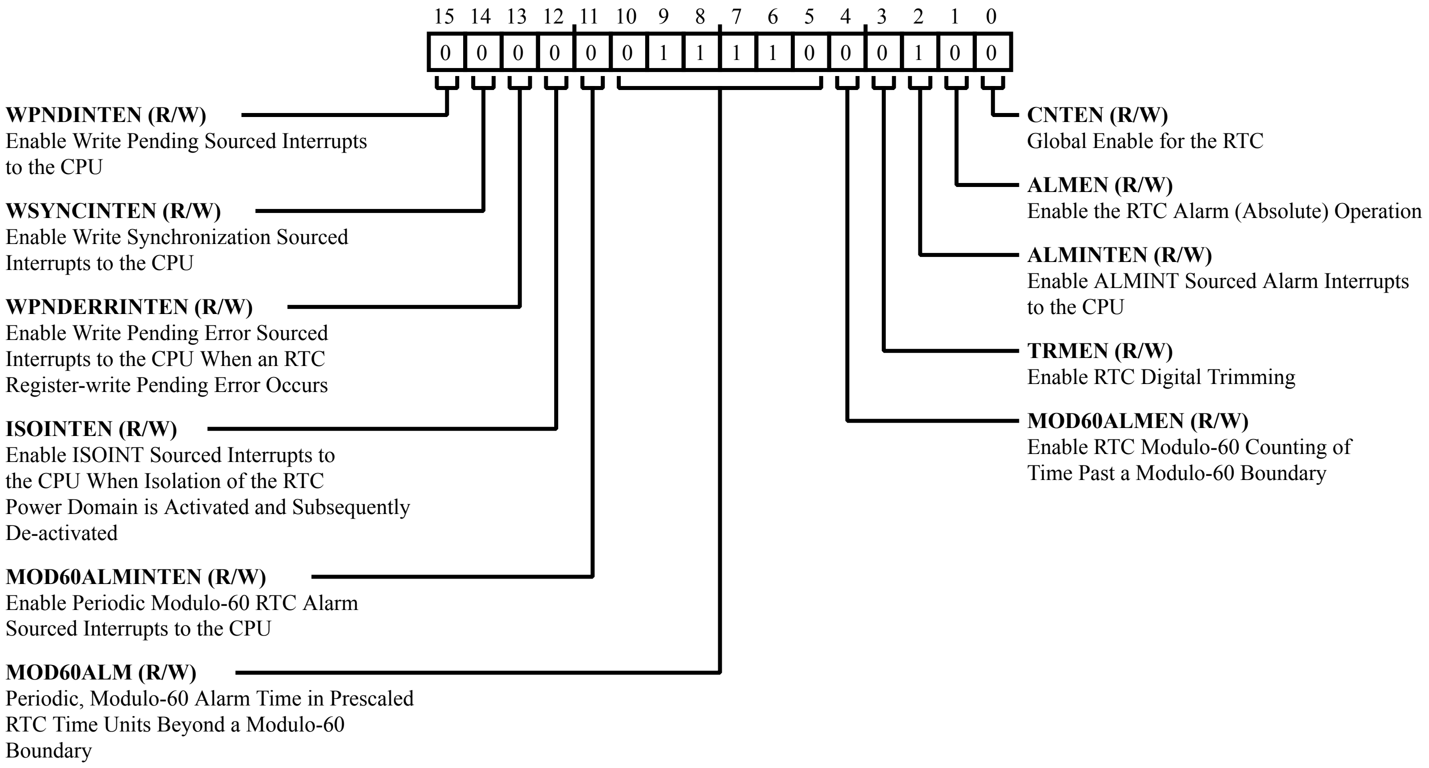

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 1 1 1 1 0 0 0 1 0 0
WPNDINTEN (R/W)
Enable Write Pending Sourced Interrupts to the CPU
WSYNCINTEN (R/W)
Enable Write Synchronization Sourced Interrupts to the CPU
WPNDERRINTEN (R/W)
Enable Write Pending Error Sourced Interrupts to the CPU When an RTC Register-write Pending Error Occurs
ISOINTEN (R/W)
Enable ISOINT Sourced Interrupts to the CPU When Isolation of the RTC Power Domain is Activated and Subsequently De-activated
MOD60ALMINTEN (R/W)
Enable Periodic Modulo-60 RTC Alarm Sourced Interrupts to the CPU
MOD60ALM (R/W)
Periodic, Modulo-60 Alarm Time in Prescaled RTC Time Units Beyond a Modulo-60 Boundary
CNTEN (R/W)
Global Enable for the RTC
ALMEN (R/W)
Enable the RTC Alarm (Absolute) Operation
ALMINTEN (R/W)
Enable ALMINT Sourced Alarm Interrupts to the CPU
TRMEN (R/W)
Enable RTC Digital Trimming
MOD60ALMEN (R/W)
Enable RTC Modulo-60 Counting of Time Past a Modulo-60 Boundary

Figure 21-12: RTC_CR0 Register Diagram

Table 21-8: RTC_CR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	WPNDINTEN	Enable Write Pending Sourced Interrupts to the CPU. This field is an enable for RTC interrupts to the CPU based on the RTC_SR0.WPNDINT sticky interrupt source field.	
		0	Disable RTC_SR0.WPNDINT-sourced interrupts to the CPU.
		1	Enable RTC_SR0.WPNDINT-sourced interrupts to the CPU.

Table 21-8: RTC_CR0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	WSYNCINTEN	Enable Write Synchronization Sourced Interrupts to the CPU. This field is an enable for RTC interrupts to the CPU based on the `RTC_SR0.WSYNCINT` sticky interrupt source field.	
		0	Disable `RTC_SR0.WSYNCINT`-sourced interrupts to the CPU.
		1	Enable `RTC_SR0.WSYNCINT`-sourced interrupts to the CPU.
13 (R/W)	WPNDERRINTEN	Enable Write Pending Error Sourced Interrupts to the CPU When an RTC Register-write Pending Error Occurs. This field is an enable for RTC interrupts to the CPU based on the `RTC_SR0.WPNDINT` sticky interrupt source field.	
		0	Don't enable interrupts if write pending errors occur in the RTC.
		1	Enable interrupts for write pending errors in the RTC.
12 (R/W)	ISOINTEN	Enable ISOINT Sourced Interrupts to the CPU When Isolation of the RTC Power Domain is Activated and Subsequently De-activated. This field enables interrupts to the CPU based on the `RTC_SR0.ISOINT` sticky interrupt source. Note that this bit field only exists in RTC1. In RTC0, the field is reserved and reads back as zero.	
		0	Disable `RTC_SR0.ISOINT`-sourced interrupts to the CPU.
		1	Enable `RTC_SR0.ISOINT`-sourced interrupts to the CPU.
11 (R/W)	MOD60ALMINTEN	Enable Periodic Modulo-60 RTC Alarm Sourced Interrupts to the CPU. This field enables interrupts to the CPU based on the `RTC_SR0.MOD60ALMINT` sticky interrupt source. Note that this bit field only exists in RTC1. In RTC0, the field is reserved and reads back as zero.	
		0	Disable periodic interrupts due to modulo-60 RTC elapsed time.
		1	Enable periodic interrupts due to modulo-60 RTC elapsed time.

Table 21-8: RTC_CR0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10:5 (R/W)	MOD60ALM	Periodic, Modulo-60 Alarm Time in Prescaled RTC Time Units Beyond a Modulo-60 Boundary. This field allows the CPU to position a periodic (repeating) alarm interrupt from the RTC at any integer number of prescaled RTC time units from a modulo-60 boundary (roll-over event) of the RTC integer count in `RTC_CNT1` and `RTC_CNT0`. Values of 0 to 59 are allowed. If a greater value is configured, it is treated as zero prescaled RTC time units. Boundaries are defined whenever any of the following events occurs : (i) the CPU writes a new pair of values to the `RTC_CNT1`and `RTC_CNT0` registers. (ii) the CPU enables the RTC from a disabled state using the `RTC_CR0.CNTEN` field. (iii) when the RTC is enabled by `RTC_CR0.CNTEN`, the degree of prescaling in the RTC is changed via the `RTC_CR1.PRESCALE2EXP` field. This bit field only exists in RTC1. In RTC0, the field is reserved and reads back as all zeros.	
4 (R/W)	MOD60ALMEN	Enable RTC Modulo-60 Counting of Time Past a Modulo-60 Boundary. `RTC_CR0.MOD60ALMEN` enables the RTC to detect and record until cleared by the CPU the periodic interrupt condition of `RTC_CNT1` and `RTC_CNT0` having reached `RTC_CR0.MOD60ALM` number of prescaled RTC time units past a modulo-60 boundary. Note that this bit field only exists in RTC1. In RTC0, the field is reserved and reads back as zero.	
		0	Disable determination of modul0-60 RTC elapsed time.
		1	Enable determination of modulo-60 RTC elapsed time.
3 (R/W)	TRMEN	Enable RTC Digital Trimming. Trimming of the RTC allows the real-time count in prescaled RTC time units to be adjusted on a periodic basis to track time with better accuracy. `RTC_CR0.TRMEN` enables this adjustment, provided the RTC is enabled via `RTC_CR0.CNTEN`. Note that if `RTC_CR0.TRMEN` is activated from a disabled state while `RTC_CR0.CNTEN` is also active, this will cause a trim interval boundary to occur and a new trim interval will begin. No trim adjustment of the RTC count occurs on such `RTC_CR0.TRMEN` activation. A whole, enabled trim interval must have elapsed before any adjustment is made.	
		0	Digital trimming of the RTC count value is disabled.
		1	Trim is enabled.

Table 21-8: RTC_CR0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	ALMINTEN	Enable ALMINT Sourced Alarm Interrupts to the CPU. `RTC_CR0.ALMINTEN` enables interrupts to the CPU based on the `RTC_SR0.ALMINT` sticky interrupt source.	
		0	Disable alarm interrupts.
		1	Enable an interrupt if RTC alarm and count values match.
1 (R/W)	ALMEN	Enable the RTC Alarm (Absolute) Operation. `RTC_CR0.ALMEN` must be set active for the alarm logic to function and for any alarm event to be detected. Such an event is defined as a match between the values of the RTC count and alarm registers, namely `RTC_CNT1`, `RTC_CNT0`, `RTC_CNT2`, `RTC_ALM1`, `RTC_ALM0`, and `RTC_ALM2`.	
		0	Disable detection of alarm events.
		1	Enable detection of alarm events.
0 (R/W)	CNTEN	Global Enable for the RTC. `RTC_CR0.CNTEN` enables counting of elapsed real time and acts as a master enable for the RTC. All interrupt sources can be cleared by the CPU irrespective of the value of CNTEN. If the RTC is enabled by activating `RTC_CR0.CNTEN` , this event causes a realignment of the prescaler, the trim interval and the modulo-60 counter used by the RTC to generate `RTC_SR0.MOD60ALMINT`-sourced interrupts.	
		0	Disable the RTC.
		1	Enable the RTC.

RTC Control 1

RTC_CR1 register expands the granularity of RTC control which is already available via RTC_CR0.

Note that RTC_CR1 is only configurable in RTC1, whereas in RTC0 it is a read-only register with fixed (reset) settings.

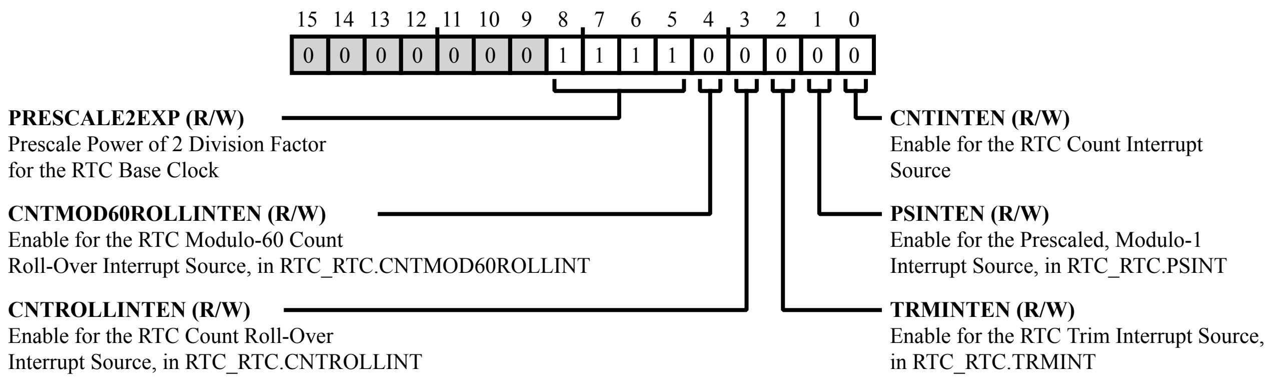

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0
PRESCALE2EXP (R/W)
Prescale Power of 2 Division Factor for the RTC Base Clock
CNTMOD60ROLLINTEN (R/W)
Enable for the RTC Modulo-60 Count Roll-Over Interrupt Source, in RTC_RTC.CNTMOD60ROLLINT
CNTROLLINTEN (R/W)
Enable for the RTC Count Roll-Over Interrupt Source, in RTC_RTC.CNTROLLINT
CNTINTEN (R/W)
Enable for the RTC Count Interrupt Source
PSINTEN (R/W)
Enable for the Prescaled, Modulo-1 Interrupt Source, in RTC_RTC.PSINT
TRMINTEN (R/W)
Enable for the RTC Trim Interrupt Source, in RTC_RTC.TRMINT

Figure 21-13: RTC_CR1 Register Diagram

Table 21-9: RTC_CR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8:5 (R/W)	PRESCALE2EXP	Prescale Power of 2 Division Factor for the RTC Base Clock. RTC_CR1.PRESCALE2EXP defines the power of two by which the RTC base clock (32 kHz) is prescaled (divided in frequency) before being used to count time by advancing the contents of the RTC_CNT1 and RTC_CNT0 registers. Note that this bit field is only configurable in RTC1. In RTC0, the field is read-only and contains a value of 0xF, signifying a fixed prescaling of 2 to the power of 15. This is because RTC0 always counts real time at 1Hz. In contrast, RTC1 can prescale the clock by any power of two in the interval [15,0] to use as its time base.	
		0	Prescale the RTC base clock by 2^0 = 1.
		1	Prescale the RTC base clock by 2^1 = 2.
		2	Prescale the RTC base clock by 2^2 = 4.
		3	Prescale the RTC base clock by 2^3 = 8.
		4	Prescale the RTC base clock by 2^4 = 16.
		5	Prescale the RTC base clock by 2^5 = 32.
		6	Prescale the RTC base clock by 2^6 = 64.
		7	Prescale the RTC base clock by 2^7 = 128.
		8	Prescale the RTC base clock by 2^8 = 256.
		9	Prescale the RTC base clock by 2^9 = 512.

Table 21-9: RTC_CR1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		10	Prescale the RTC base clock by 2^10 = 1024.
		11	Prescale the RTC base clock by 2^11 = 2048.
		12	Prescale the RTC base clock by 2^12 = 4096.
		13	Prescale the RTC base clock by 2^13 = 8192.
		14	Prescale the RTC base clock by 2^14 = 16384.
		15	Prescale the RTC base clock by 2^15 = 32768.
4 (R/W)	CNTMOD60ROLLINT-EN	Enable for the RTC Modulo-60 Count Roll-Over Interrupt Source in `RTC_SR2.CNTMOD60ROLLINT`. This bit field only exists in RTC1. In RTC0, the field is reserved and reads back as zero.	
		0	Disable `RTC_SR2.CNTMOD60ROLLINT` as a fan-in term of the RTC peripheral interrupt.
		1	Enable `RTC_SR2.CNTMOD60ROLLINT` as a fan-in term of the RTC peripheral interrupt.
3 (R/W)	CNTROLLINTEN	Enable for the RTC Count Roll-Over Interrupt Source, in `RTC_SR2.CNTROLLINT`. This bit field only exists in RTC1. In RTC0, the field is reserved and reads back as zero.	
		0	Disable `RTC_SR2.CNTROLLINT` as a fan-in term of the RTC peripheral interrupt.
		1	Enable `RTC_SR2.CNTROLLINT` as a fan-in term of the RTC peripheral interrupt.
2 (R/W)	TRMINTEN	Enable for the RTC Trim Interrupt Source, in `RTC_SR2.TRMINT`. Note that this bit field only exists in RTC1. In RTC0, the field is reserved and reads back as zero.	
		0	Disable `RTC_SR2.TRMINT` as a fan-in term of the RTC peripheral interrupt.
		1	Enable `RTC_SR2.TRMINT` as a fan-in term of the RTC peripheral interrupt.

Table 21-9: RTC_CR1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	PSINTEN	Enable for the Prescaled, Modulo-1 Interrupt Source, in `RTC_SR2.PSINT`. Note that this bit field only exists in RTC1. In RTC0, the field is reserved and reads back as zero.	
		0	Disable `RTC_SR2.PSINT` as a fan-in term of the RTC peripheral interrupt.
		1	Enable `RTC_SR2.PSINT` as a fan-in term of the RTC peripheral interrupt.
0 (R/W)	CNTINTEN	Enable for the RTC Count Interrupt Source. Note that this bit field only exists in RTC1. In RTC0, the field is reserved and reads back as zero.	
		0	Disable `RTC_SR2.CNTINT` as a fan-in term of the RTC peripheral interrupt.
		1	Enable `RTC_SR2.CNTINT` as a fan-in term of the RTC peripheral interrupt.

RTC Control 2 for Configuring Input Capture Channels

RTC_CR2IC is a control register for configuring enables related to input-capture channels. RTC_CR2IC contains enables for both the input-capture function itself for each channel, as well as enables for whether an event on a channel should contribute to the interrupt lines from the RTC to both the CPU and the wake-up controller.

RTC input capture channels and GPIO pads: Input Capture channel [0,2,3,4] are connected to SYS_WAKE [0,1,2,3] respectively.

Note that this register only exists in RTC1. In RTC0, the register address is reserved and reads back as the register's reset value.

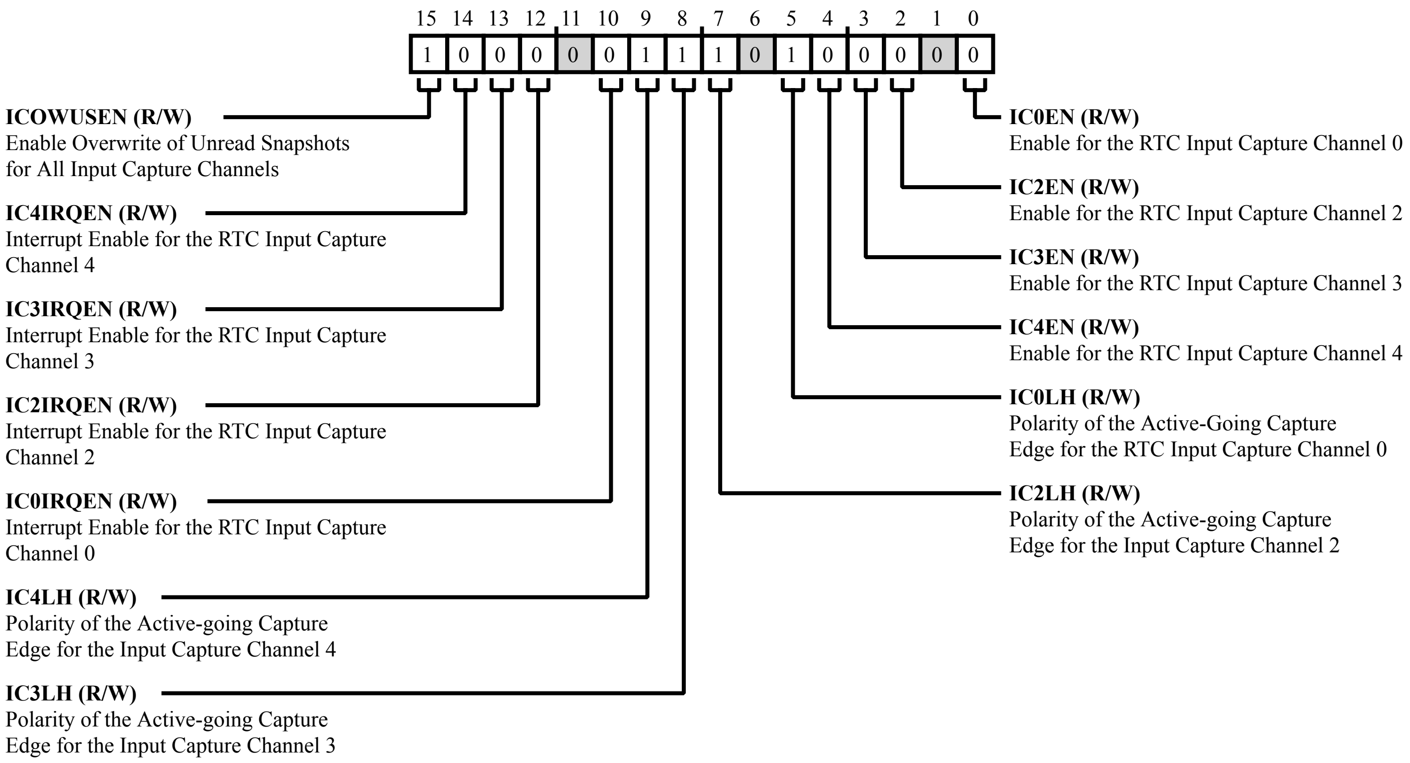

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0 0 0 1 1 1 0 1 0 0 0 0 0
ICOWUSEN (R/W)
Enable Overwrite of Unread Snapshots for All Input Capture Channels
IC4IRQEN (R/W)
Interrupt Enable for the RTC Input Capture Channel 4
IC3IRQEN (R/W)
Interrupt Enable for the RTC Input Capture Channel 3
IC2IRQEN (R/W)
Interrupt Enable for the RTC Input Capture Channel 2
IC0IRQEN (R/W)
Interrupt Enable for the RTC Input Capture Channel 0
IC4LH (R/W)
Polarity of the Active-going Capture Edge for the Input Capture Channel 4
IC3LH (R/W)
Polarity of the Active-going Capture Edge for the Input Capture Channel 3
IC0EN (R/W)
Enable for the RTC Input Capture Channel 0
IC2EN (R/W)
Enable for the RTC Input Capture Channel 2
IC3EN (R/W)
Enable for the RTC Input Capture Channel 3
IC4EN (R/W)
Enable for the RTC Input Capture Channel 4
IC0LH (R/W)
Polarity of the Active-Going Capture Edge for the RTC Input Capture Channel 0
IC2LH (R/W)
Polarity of the Active-going Capture Edge for the Input Capture Channel 2

Figure 21-14: RTC_CR2IC Register Diagram

Table 21-10: RTC_CR2IC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	ICOWUSEN	Enable Overwrite of Unread Snapshots for All Input Capture Channels. This field controls whether snapshots for input capture channels are automatically overwritten whenever a new capture event occurs, even if a previously-captured snapshot has not yet been read by the CPU.	
		0	Snapshots persist until first read and then subject to a capture event. Overwrites of unread snapshots are not enabled.
		1	Snapshots persist until new capture events occur. Overwrites of unread snapshots are enabled.
14 (R/W)	IC4IRQEN	Interrupt Enable for the RTC Input Capture Channel 4. This field, active high, determines whether the sticky interrupt source `RTC_SR3.IC4IRQ` is enabled to fan in as a contributory term to the RTC IRQ interrupt line to the CPU.	
		0	Disable `RTC_SR3.IC4IRQ` from contributing to the RTC interrupts to both the CPU and the wake-up controller.
		1	Enable `RTC_SR3.IC4IRQ` as a contributory fan-in term to the RTC interrupts to both the CPU and the wake-up controller.
13 (R/W)	IC3IRQEN	Interrupt Enable for the RTC Input Capture Channel 3. This field, active high, determines whether the sticky interrupt source `RTC_SR3.IC3IRQ` is enabled to fan in as a contributory term to the RTC IRQ interrupt line to the CPU.	
		0	Disable IC3 Interrupts Disable `RTC_SR3.IC3IRQ` from contributing to the RTC interrupts to both the CPU and the wake-up-controller.
		1	Enable IC3 Interrupts Enable `RTC_SR3.IC3IRQ` as a contributory fan-in term to the RTC interrupts to both the CPU and the wake-up controller.

Table 21-10: RTC_CR2IC Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	IC2IRQEN	Interrupt Enable for the RTC Input Capture Channel 2. This field, active high, determines whether the sticky interrupt source `RTC_SR3.IC2IRQ` is enabled to fan in as a contributory term to the RTC IRQ interrupt line to the CPU.	
		0	Disable `RTC_SR3.IC2IRQ` from contributing to the RTC interrupts to both the CPU and the wake-up controller.
		1	Enable `RTC_SR3.IC2IRQ` as a contributory fan-in term to the RTC interrupts to both the CPU and the wake-up controller.
10 (R/W)	IC0IRQEN	Interrupt Enable for the RTC Input Capture Channel 0. This field, active high, determines whether the sticky interrupt source `RTC_SR3.IC0IRQ` enabled to fan in as a contributory term to the RTC IRQ interrupt line to the CPU.	
		0	Disable `RTC_SR3.IC0IRQ` from contributing to the RTC interrupt.
		1	Enable `RTC_SR3.IC0IRQ` as a contributory fan-in term to the RTC interrupt.
9 (R/W)	IC4LH	Polarity of the Active-going Capture Edge for the Input Capture Channel 4. This field selects whether an input capture event on the IC4 channel is defined as a low-to-high (LH) or high-to-low transition of the GPIO input for that channel.	
		0	The `RTC_IC4` channel uses a high-to-low transition on its GPIO pin to signal an input capture event.
		1	The `RTC_IC4` channel uses a low-to-high transition on its GPIO pin to signal an input capture event.
8 (R/W)	IC3LH	Polarity of the Active-going Capture Edge for the Input Capture Channel 3. This field selects whether an input capture event on the IC3 channel is defined as a low-to-high (LH) or high-to-low transition of the GPIO input for that channel.	
		0	The `RTC_IC3` channel uses a high-to-low transition on its GPIO pin to signal an input capture event.
		1	The `RTC_IC3` channel uses a low-to-high transition on its GPIO pin to signal an input capture event.

Table 21-10: RTC_CR2IC Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	IC2LH	Polarity of the Active-going Capture Edge for the Input Capture Channel 2. This field selects whether an input capture event on the IC2 channel is defined as a low-to-high (LH) or high-to-low transition of the GPIO input for that channel.	
		0	The RTC_IC2 channel uses a high-to-low transition on its GPIO pin to signal an input capture event.
		1	The RTC_IC2 channel uses a low-to-high transition on its GPIO pin to signal an input capture event.
5 (R/W)	IC0LH	Polarity of the Active-Going Capture Edge for the RTC Input Capture Channel 0. This field selects whether an input capture event on the IC0 channel is defined as a low-to-high (LH) or high-to-low transition of the GPIO input for that channel.	
		0	The IC0 channel uses a high-to-low transition on its GPIO pin to signal an input capture event.
		1	The IC0 channel uses a low-to-high transition on its GPIO pin to signal an input capture event.
4 (R/W)	IC4EN	Enable for the RTC Input Capture Channel 4. This field, active high, is a global enable for the Input Capture 4 channel RTC_IC4.	
		0	Disable the 16-bit input-capture channel RTC_IC4.
		1	Enable the 16-bit input-capture channel RTC_IC4.
3 (R/W)	IC3EN	Enable for the RTC Input Capture Channel 3. This field, active high, is a global enable for the Input Capture 3 channel RTC_IC3.	
		0	Disable the 16-bit input-capture channel RTC_IC3.
		1	Enable the 16-bit Input-capture channel RTC_IC3.
2 (R/W)	IC2EN	Enable for the RTC Input Capture Channel 2. This field, active high, is a global enable for the Input Capture 2 channel RTC_IC2.	
		0	Disable the 16-bit input-capture channel RTC_IC2.
		1	Enable the 16-bit input-capture channel RTC_IC2.

Table 21-10: RTC_CR2IC Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	IC0EN	Enable for the RTC Input Capture Channel 0. Active high, global enable for the IC0 functionality related only to GPIO prompting of input capture into the 47-bit snapshot registers, RTC_SNAP1, RTC_SNAP0 and RTC_SNAP2. Note that RTC_CR2IC.IC0EN has no effect on CPU-originated snapshots, which are prompted by a write of the specific key value of 0x7627 to the RTC_GWY register.	
		0	Disable externally-requested (GPIO, non-CPU) snapshots for the 47-bit input-capture channel IC0.
		1	Enable externally-requested (GPIO, non-CPU) snapshots for the 47-bit input-capture channel IC0.

RTC Control 3 for Configuring SensorStrobe Channel

RTC_CR3SS is a control register for configuring enables related to 16-bit SensorStrobe channels. The 47-bit SensorStrobe channel 0 is not controlled by RTC_CR3SS. This is because SensorStrobe channel 0 is a synonym for the main 47-bit RTC alarm (whose interrupt source is RTC_SR0.ALMINT), which is controlled by RTC_CR0.ALMEN and RTC_CR0.ALMINTEN. Note that RTC_CR3SS only exists in RTC1. In RTC0, the register address is reserved and reads back as the register's reset value.

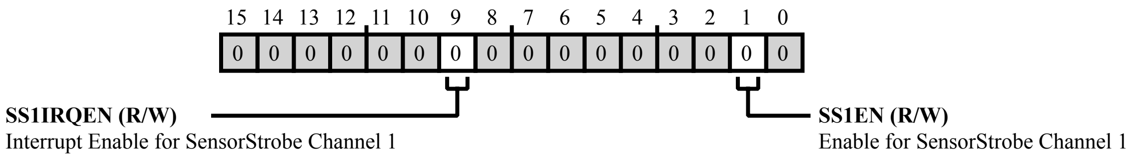


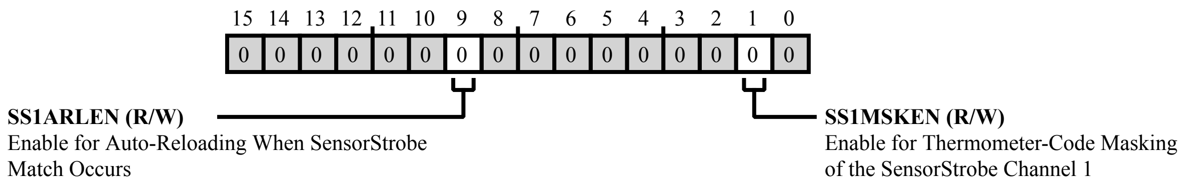
Figure 21-15: RTC_CR3SS Register Diagram

Table 21-11: RTC_CR3SS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	SS1IRQEN	Interrupt Enable for SensorStrobe Channel 1. Active high, determines whether the sticky interrupt source RTC_SR3.SS1IRQ is enabled to fan in as a contributory term to the RTC IRQ interrupt lines to both the CPU and the wake-up controller.	
		0	Disable RTC_SR3.SS1IRQ from contributing to the RTC interrupts to both the CPU and the wake-up controller.
		1	Enable RTC_SR3.SS1IRQ as a contributory fan-in term to the RTC interrupts to both the CPU and the wake-up controller.
1 (R/W)	SS1EN	Enable for SensorStrobe Channel 1. Active high, global enable for the SensorStrobe channel 1 (scheduled alarm), RTC_SS1.	
		0	Disable the 16-bit SensorStrobe channel 1
		1	Enable the 16-bit SensorStrobe channel 1

RTC Control 4 for Configuring SensorStrobe Channel

RTC_CR4SS is a control register for configuring enables related to masking and auto-reloading of the 16-bit SensorStrobe channel RTC_SS1. Note that this register only exists in RTC1. In RTC0, the register address is reserved and reads back as the register's reset value.



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
SS1ARLEN (R/W)
Enable for Auto-Reloading When SensorStrobe Match Occurs
SS1MSKEN (R/W)
Enable for Thermometer-Code Masking of the SensorStrobe Channel 1

Figure 21-16: RTC_CR4SS Register Diagram

Table 21-12: RTC_CR4SS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	SS1ARLEN	Enable for Auto-Reloading When SensorStrobe Match Occurs. If auto-reloading of RTC_SS1 is enabled via this bitfield, the contents of RTC_SS1ARL are added, modulo 16, to the value of RTC_SS1 whenever an optionally masked match occurs for the SensorStrobe channel 1. This is expressed as RTC_SS1 = RTC_CR4SS.SS1ARLEN ? RTC_SS1 + RTC_SS1ARL : RTC_SS1. In such circumstances, RTC_SS1 acts as a repeating alarm, whereby those bits which are not masked in the reload value effectively define the step size (offset) from the current time to the next SensorStrobe alarm.	
		0	Disable auto-reloading of RTC_SS1 and instead maintain its target value whenever an enabled SensorStrobe event occurs on that channel.
		1	Enable auto-reloading of RTC_SS1 from RTC_SS1ARL whenever an enabled SensorStrobe event occurs on that channel.

Table 21-12: RTC_CR4SS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	SS1MSKEN	Enable for Thermometer-Code Masking of the SensorStrobe Channel 1. This field is used to optionally enable masking of matches between RTC_SS1 and the RTC count, when determining if a scheduled alarm should be activated for SensorStrobe channel 1. When enabled via this field, a four-bit code, embedded in RTC_SSMSK, is decoded out to a 16-bit thermometer-code mask and applied to bit positions in both RTC_SS1 and 16 least significant integer and fractional bits in the lower end of the RTC count given by RTC_CNT0 and RTC_CNT2.	
		0	Do not apply a mask to SensorStrobe Channel 1 Register
		1	Apply thermometer decoded mask Apply a thermometer-decoded mask to RTC_SS1, provided the channel is enabled via RTC_CR3SS.SS1EN.

RTC Freeze Count

RTC Freeze Count MMR allows a coherent, triple 16-bit read of the 47-bit RTC count contained in RTC_CNT2, RTC_CNT1 and RTC_CNT0.

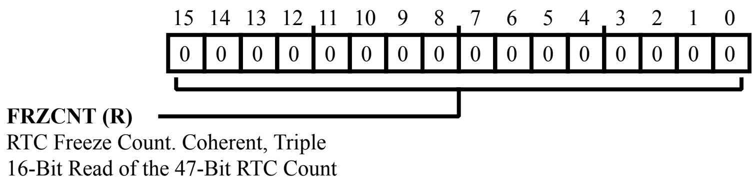


Figure 21-17: RTC_FRZCNT Register Diagram

Table 21-13: RTC_FRZCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	FRZCNT	RTC Freeze Count. Coherent, Triple 16-Bit Read of the 47-Bit RTC Count. This field is always read in sequences of three reads, such that the first read in the sequence returns the current value of RTC_CNT0. Simultaneously, with this first read, a snapshot is taken and frozen of the value of RTC_CNT1 and RTC_CNT2 so that in the second and third reads in the sequence of RTC_FRZCNT.FRZCNT, the snapshot values of RTC_CNT1 and RTC_CNT2 are returned respectively. The sequence number for such a triplet of reads is visible via RTC_SR6.FRZCNTPTR.

RTC Gateway

RTC_GWY is a gateway MMR address through which the CPU can order actions to be taken within the RTC. The CPU does this by writing specific keys to RTC_GWY. Note that RTC_GWY reads back as all zeros.

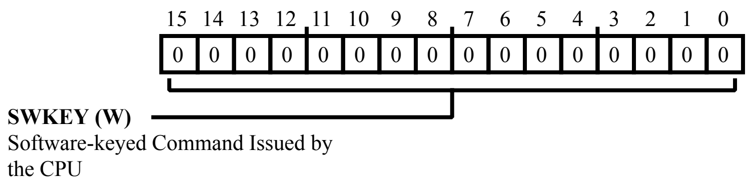


Figure 21-18: RTC_GWY Register Diagram

Table 21-14: RTC_GWY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (RX/W)	SWKEY	Software-keyed Command Issued by the CPU. The RTC_GWY register is the write target for activating software-keyed commands issued by the CPU to the RTC. The supported keys are as follows: (i) A FLUSH_RTC software key (delivered via a register write to RTC_GWY) of value 0xa2c5 causes the RTC to flush (discard) all posted write transactions and to immediately stop any transaction that is currently executing. (ii) A SNAPSHOT_RTC key of value 0x7627 (delivered via a register write to RTC_GWY) causes the RTC to take a sticky snapshot of the value of RTC_CNT1, RTC_CNT0 and RTC_CNT2 and store it in RTC_SNAP1, RTC_SNAP0 and RTC_SNAP2. (iii) A key of value 0x9376, when written to RTC_GWY, zeroes the RTC_SR6.FRZCNTPTR pointer and thus re-initialises the 0-1-2 sequence for reads of RTC_FRZCNT.

RTC Input Capture Channel 2

RTC_IC2 is a read-only snapshot of the 16 lowest {integer_bits, fractional_bits} with meaning of the main 47-bit RTC count at the most recent event on input capture channel 2. Note that this register only exists in RTC1. In RTC0, the register address is reserved and reads back as the register's reset value.

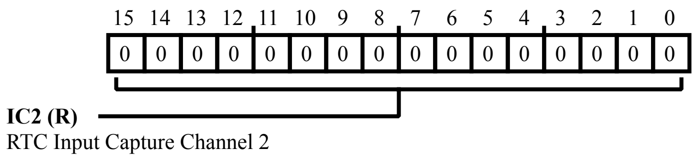


Figure 21-19: RTC_IC2 Register Diagram

Table 21-15: RTC_IC2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	IC2	RTC Input Capture Channel 2. A snaphot of RTC time is placed with an alignment determined by prescaling into `RTC_IC2.IC2` when prompted to do so by an external requesting input into the RTC on input capture channel 2. Such an event overwrites (if allowed by `RTC_CR2IC.ICOWUSEN`) any value captured previously in `RTC_IC2.IC2`.

RTC Input Capture Channel 3

RTC_IC3is a read-only snapshot of the 16 lowest {integer_bits, fractional_bits} with meaning of the main 47-bit RTC count at the most recent event on input capture channel 3. Note that this register only exists in RTC1. In RTC0, the register address is reserved and reads back as the register's reset value.

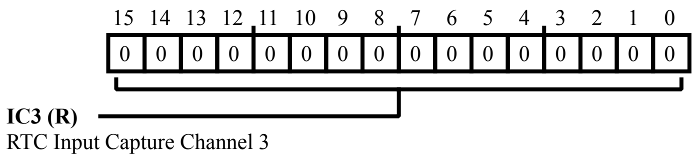


Figure 21-20: RTC_IC3 Register Diagram

Table 21-16: RTC_IC3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	IC3	RTC Input Capture Channel 3. A snaphot of RTC time is placed with an alignment determined by prescaling into `RTC_IC3.IC3` when prompted to do so by an external requesting input into the RTC on input capture channel 3. Such an event overwrites (if allowed by `RTC_CR2IC.ICOWUSEN`) any value captured previously in `RTC_IC3.IC3`.

RTC Input Capture Channel 4

RTC_IC4 is a read-only snapshot of the 16 lowest {integer_bits, fractional_bits} with meaning of the main 47-bit RTC count at the most recent event on input capture channel 4. Note that this register only exists in RTC1. In RTC0, the register address is reserved and reads back as the register's reset value.

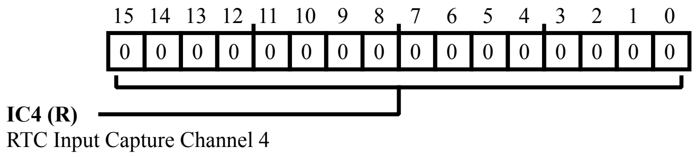


Figure 21-21: RTC_IC4 Register Diagram

Table 21-17: RTC_IC4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	IC4	RTC Input Capture Channel 4. A snaphot of RTC time is placed with an alignment determined by prescaling into `RTC_IC4.IC4` when prompted to do so by an external requesting input into the RTC on input capture channel 4. Such an event overwrites (if allowed by `RTC_CR2IC.ICOWUSEN`) any value captured previously in `RTC_IC4.IC4`.

RTC Modulo

RTC_MOD is a read-only register which makes available RTC_MOD.CNTMOD60, which is the modulo-60 equivalent of the count value in RTC_CNT1 and RTC_CNT0. This modulo-60 value is equal to the displacement in prescaled RTC time units past the most recent modulo-60 roll-over event. A roll-over is a synonym for a modulo-60 boundary.

Boundaries are defined in the following way. The RTC realigns itself to create coincident modulo-60 and modulo-1 boundaries whenever any of the following events occurs :

(i) the CPU writes a new pair of values to the RTC_CNT1 and RTC_CNT0 registers to redefine the elapsed time units count while the RTC is enabled and this posted twin write is subsequently executed.

(ii) the CPU enables the RTC from a disabled state using the RTC_CR0.CNTEN field.

(iii) while the RTC is enabled by RTC_CR0.CNTEN, the degree of prescaling in the RTC is changed via the RTC_CR1.PRESCALE2EXP field.

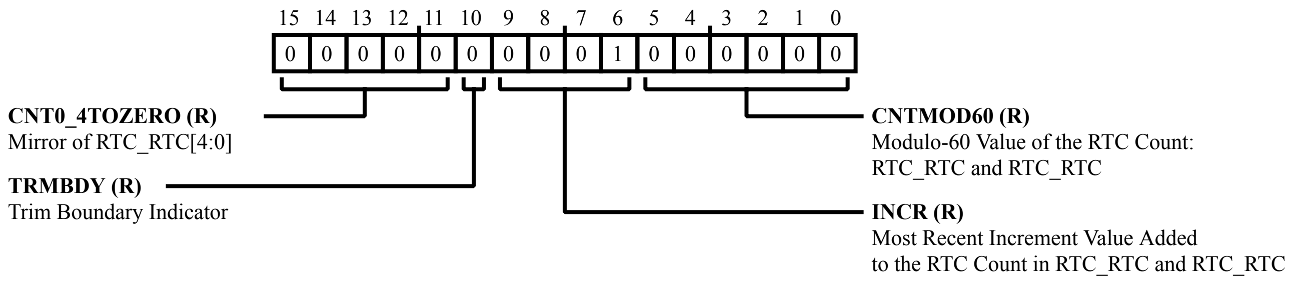


Figure 21-22: RTC_MOD Register Diagram

Table 21-18: RTC_MOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:11 (R/NW)	CNT0_4TOZERO	Mirror of RTC_CNT0[4:0]. RTC_MOD.CNT0_4TOZERO is a mirror of RTC_CNT0[4:0], available for simultaneous read-back along with the RTC_MOD.CNTMOD60 field. Having this mirror available at the same time allows the relationship between the modulo-60 and absolute versions of the RTC count to be better understood and debugged.	
10 (R/NW)	TRMBDY	Trim Boundary Indicator. Trim boundary indicator that the most recent RTC count increment has coincided with trimming of the count value.	
		0	Trimming has not occurred at the most recent increment of the RTC count.
		1	Trimming has occurred at the most recent increment of the RTC count

Table 21-18: RTC_MOD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9:6 (R/NW)	INCR	Most Recent Increment Value Added to the RTC Count in RTC_CNT1 and RTC_CNT0. RTC_MOD.INCR is a read-only value by which the RTC count has most recently been incremented. Under normal circumstances, when the RTC is enabled, this value will be one. However, when trimming occurs, RTC_MOD.INCR can be any integer value between zero and eight, depending on the trim configuration in the RTC_TRM register.
5:0 (R/NW)	CNTMOD60	Modulo-60 Value of the RTC Count: RTC_CNT1 and RTC_CNT0. RTC_MOD.CNTMOD60 counts from 0 to 59, and then rolls over to 0 again. It advances (and is trimmed) in tandem with the main RTC count in RTC_CNT1, RTC_CNT0 and RTC_CNT2. RTC_MOD.CNTMOD60 is zeroed whenever any of the following events occurs: (i) A normal roll-over from a value of 59 when advancing at a prescaled time unit. (ii) whenever the CPU writes a new pair of values to the RTC_CNT1 and RTC_CNT0 registers to redefine the elapsed time count while the RTC is enabled and this posted twin write is executed. (iii) the CPU enables the RTC from a disabled state using the RTC_CR0.CNTEN field. (iv) while the RTC is enabled via RTC_CR0.CNTEN, the degree of prescaling in the RTC is changed via the RTC_CR1.PRESCALE2EXP field. This bit field only exists in RTC1. In RTC0, the field is reserved and reads back as all zeros.

RTC SensorStrobe Channel 1

RTC_SS1 is the scheduled alarm time for SensorStrobe channel 1 with respect to the 16 lowest {integer_bits, fractional_bits} with meaning of the main 47-bit RTC count. The upper bits of the main RTC count, beyond these 16 bit positions, are don't cares for the purposes of the 16-bit RTC_SS1 SensorStrobe channel.

Note that this register only exists in RTC1. In RTC0, the register address is reserved and reads back as the register's reset value.

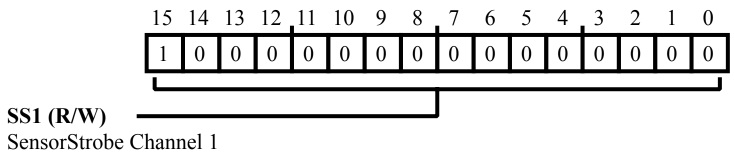


Figure 21-23: RTC_SS1 Register Diagram

Table 21-19: RTC_SS1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	SS1	SensorStrobe Channel 1. Scheduled alarm target time for SensorStrobe channel 1, with optional auto-reloading (i.e. cumulative incrementing by RTC_SS1ARL.SS1ARL upon matches) and masking.

RTC Auto-Reload for SensorStrobe Channel 1

RTC_SS1ARL contains the 16-bit reload value which is optionally (enabled by RTC_CR4SS.SS1ARLEN) added to the cumulative value of RTC_SS1, visible in the RTC_SS1TGT register, whenever an enabled SensorStrobe event occurs on that channel. Only RTC_SS1 has this reload capability. The use of RTC_SS1ARL allows a repeating alarm whose periodicity either is or is not a power of 2 to be put into effect for RTC_SS1. If reloading is not enabled, the read-back values of RTC_SS1 and RTC_SS1TGT are the same, namely the starting (and not reloaded because not enabled) SensorStrobe value in RTC_SS1.

Note that this register only exists in RTC1. In RTC0, the register address is reserved and reads back as the register's reset value.

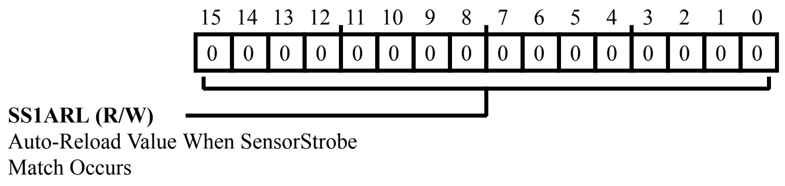


Figure 21-24: RTC_SS1ARL Register Diagram

Table 21-20: RTC_SS1ARL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	SS1ARL	Auto-Reload Value When SensorStrobe Match Occurs. If auto-reloading of RTC_SS1 is enabled via RTC_CR4SS.SS1ARLEN, the contents of this register field are added, modulo 16, to the value of RTC_SS1 whenever an optionally masked match occurs for the SensorStrobe channel 1.

RTC SensorStrobe Channel 1 Target

Reflects the current, cumulative target alarm time for the SensorStrobe channel 1. This register only exists in RTC1. In RTC0, the register address is reserved and reads back as the register's reset value.

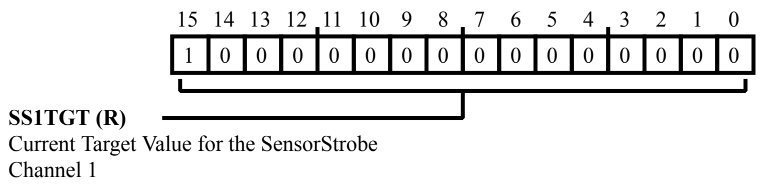


Figure 21-25: RTC_SS1TGT Register Diagram

Table 21-21: RTC_SS1TGT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	SS1TGT	Current Target Value for the SensorStrobe Channel 1. Provides Visibility to the CPU of the Current Target Value for the SensorStrobe Channel 1, taking account of any possible auto-reloading. If auto-reloading is disabled by `RTC_CR4SS.SS1ARLEN`, the value returned by this field is identical to the starting value of the `RTC_SS1` register. If auto-reloading is enabled, a read-back of this field returns the current, cumulative target value for the SensorStrobe channel 1, having started the SensorStrobe sequence from the value contained in the `RTC_SS1` register and then reloaded, i.e. additively accumulated a new target time (offset by the value in `RTC_SS1ARL.SS1ARL`) every time an SensorStrobe event occurs.

RTC Mask for SensorStrobe Channel

RTC_SSMSK contains a 4-bit encoded mask, which is decoded out to a 16-bit thermometer-code mask to define contiguous don't care bit positions for target alarm times in the 16-bit SensorStrobe channel 1. Note that this register only exists in RTC1. In RTC0, the register address is reserved and reads back as the register's reset value.

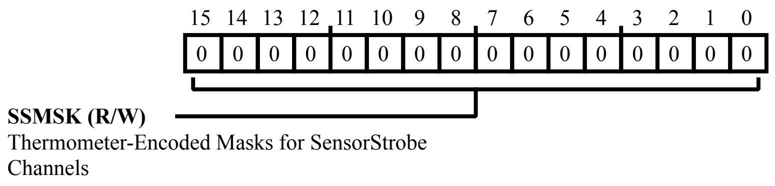


Figure 21-26: RTC_SSMSK Register Diagram

Table 21-22: RTC_SSMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	SSMSK	Thermometer-Encoded Masks for SensorStrobe Channels. Bits [3:0] of this field, when thermometer-decoded out to 16-bit values, act as an optional mask, enabled by RTC_CR4SS.SS1MSKEN, used in the determination of matches with the RTC count for SensorStrobe channel RTC_SS1. Bits [3:0] specify from what bit position upwards, contiguous bits of the SensorStrobe channel should be masked out as don't-care values for matches. Thus, for example, a value of "2" in bits [3:0] of RTC_SSMSK.SSMSK specifies that all bit positions from 2 upwards are masked out in comparisons. Masks can therefore be constructed the following series : 0x0000 (no bits masked), 0x8000 (MSBit masked), 0xC000 (2 MSBits masked), 0xE000, 0xF000, ..., 0xFFF0, 0xFFF8, 0xFFFC (only the 2 LSBits unmasked), 0xFFFE (only the LSBit unmasked), 0xFFFF (all bits masked, implying continuous SensorStrobe matches).

RTC Snapshot 0

RTC_SNAP0 is a sticky snapshot of the value of RTC_CNT0. It is updated, along with its counterparts RTC_SNAP1 and RTC_SNAP2, thereby overwriting any previous value, whenever either of the following two events occurs:

(i) the CPU writes a snapshot request key of 16'h7627 to the RTC_GWY MMR.

(ii) an input capture event occurs on the Input Capture channel 0 when enabled, provided the setting RTC_CR2IC.ICOWUSEN allows such overwriting.

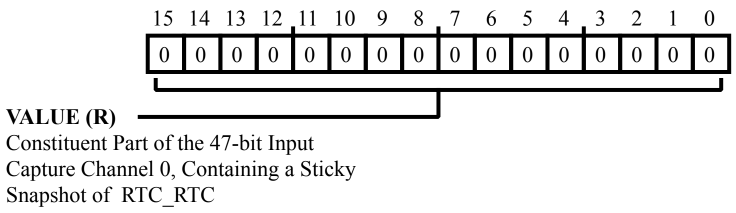

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Constituent Part of the 47-bit Input Capture Channel 0, Containing a Sticky Snapshot of RTC_RTC

Figure 21-27: RTC_SNAP0 Register Diagram

Table 21-23: RTC_SNAP0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Constituent Part of the 47-bit Input Capture Channel 0, Containing a Sticky Snapshot of RTC_CNT0. RTC_SNAP0.VALUE is part of the 47-bit Input Capture channel 0.

RTC Snapshot 1

RTC_SNAP1 is a sticky snapshot of the value of RTC_CNT1. It is updated, along with its counterparts RTC_SNAP0 and RTC_SNAP2, thereby overwriting any previous value, whenever either of the following two events occurs:

(i) the CPU writes a snapshot request key of 16'h7627 to the RTC_GWY MMR.

(ii) an input capture event occurs on the Input Capture channel 0 when enabled, provided the setting of RTC_CR2IC.ICOWUSEN allows such overwriting.

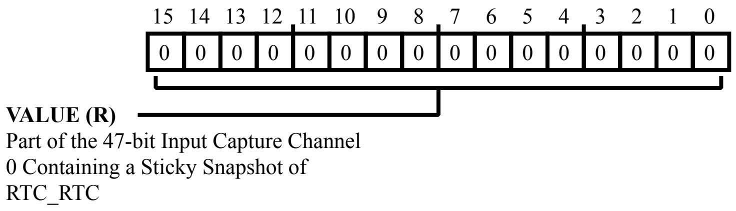

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Part of the 47-bit Input Capture Channel 0 Containing a Sticky Snapshot of RTC_RTC

Figure 21-28: RTC_SNAP1 Register Diagram

Table 21-24: RTC_SNAP1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Part of the 47-bit Input Capture Channel 0 Containing a Sticky Snapshot of RTC_CNT1. RTC_SNAP1.VALUE is part of the 47-bit Input Capture channel 0.

RTC Snapshot 2

RTC_SNAP2 is a sticky snapshot of the value of RTC_CNT2. It is updated, along with its counterparts RTC_SNAP0 and RTC_SNAP1, thereby overwriting any previous value, whenever either of the following two events occurs:

(i) the CPU writes a snapshot request key of 16'h7627 to the RTC_GWY MMR.

(ii) an input capture event occurs on the Input Capture channel 0 when enabled, provided the setting of RTC_CR2IC.ICOWUSEN allows such overwriting.

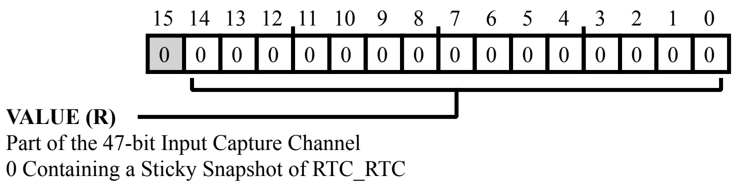

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Part of the 47-bit Input Capture Channel
0 Containing a Sticky Snapshot of RTC_RTC

Figure 21-29: RTC_SNAP2 Register Diagram

Table 21-25: RTC_SNAP2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/NW)	VALUE	Part of the 47-bit Input Capture Channel 0 Containing a Sticky Snapshot of RTC_CNT2. RTC_SNAP2.VALUE is part of the 47-bit Input Capture channel 0.

RTC Status 0

Information on RTC operation is made available to the CPU via three status registers `RTC_SR0`, `RTC_SR1`, and `RTC_SR2`. These registers include all flags related to CPU interrupt sources and error conditions within the RTC.

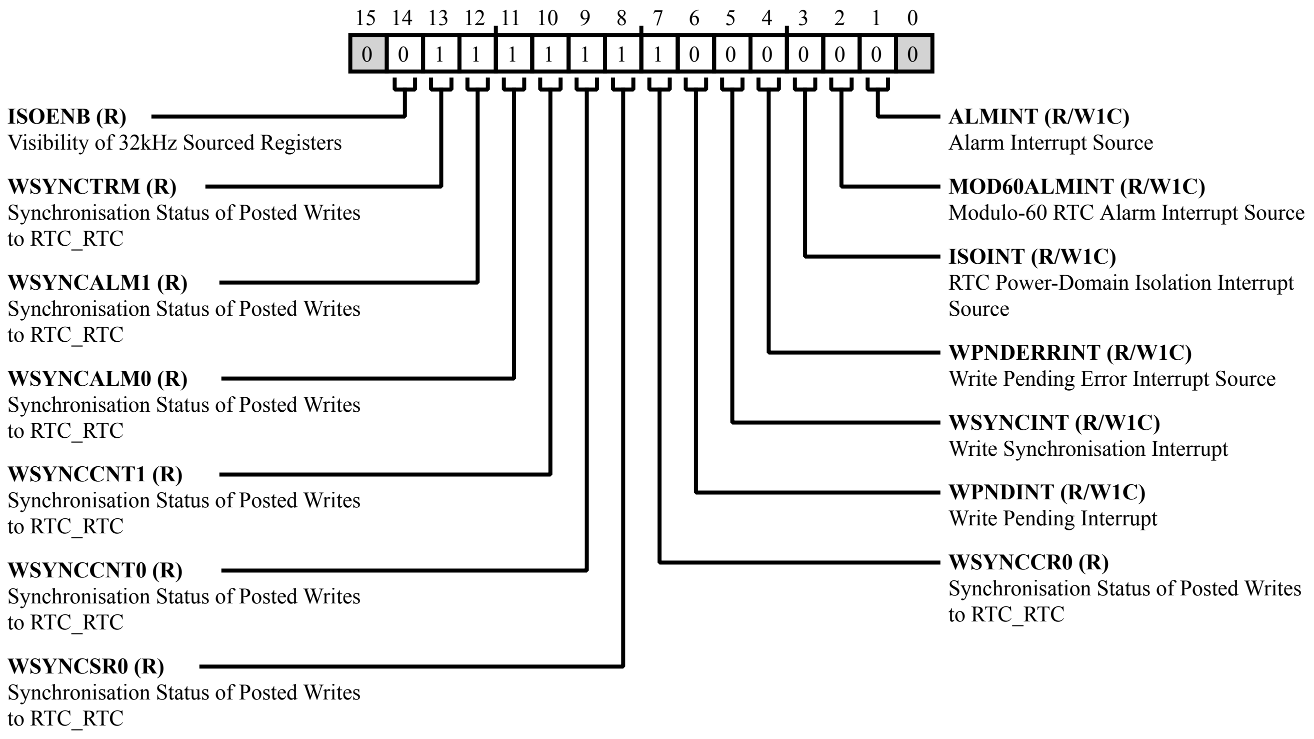

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0
ISOENB (R)
Visibility of 32kHz Sourced Registers
WSYNCTRM (R)
Synchronisation Status of Posted Writes to RTC_RTC
WSYNCALM1 (R)
Synchronisation Status of Posted Writes to RTC_RTC
WSYNCALM0 (R)
Synchronisation Status of Posted Writes to RTC_RTC
WSYNCCNT1 (R)
Synchronisation Status of Posted Writes to RTC_RTC
WSYNCCNT0 (R)
Synchronisation Status of Posted Writes to RTC_RTC
WSYNCSR0 (R)
Synchronisation Status of Posted Writes to RTC_RTC
ALMINT (R/W1C)
Alarm Interrupt Source
MOD60ALMINT (R/W1C)
Modulo-60 RTC Alarm Interrupt Source
ISOINT (R/W1C)
RTC Power-Domain Isolation Interrupt Source
WPNDERRINT (R/W1C)
Write Pending Error Interrupt Source
WSYNCINT (R/W1C)
Write Synchronisation Interrupt
WPNDINT (R/W1C)
Write Pending Interrupt
WSYNCCR0 (R)
Synchronisation Status of Posted Writes to RTC_RTC

Figure 21-30: RTC_SR0 Register Diagram

Table 21-26: RTC_SR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/NW)	ISOENB	Visibility of 32kHz Sourced Registers. Visibility status of 32 kHz sourced registers, taking account of power-domain isolation.	
		0	32 kHz-sourced MMRs in the always-on half of the RTC are not visible to the CPU due to isolation.
		1	32 kHz-sourced MMRs in the always-on half of the RTC are visible to the CPU.
13 (R/NW)	WSYNCTRM	Synchronisation Status of Posted Writes to `RTC_TRM`. This field indicates if the effects of a posted write to `RTC_TRM` are visible to the CPU.	
		0	Results of a posted write are not yet visible to the CPU.
		1	Results of a posted write are visible to the CPU.

Table 21-26: RTC_SR0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/NW)	WSYNCALM1	Synchronisation Status of Posted Writes to RTC_ALM1. This field indicates if the effects of a posted write to RTC_ALM1 are visible to the CPU.	
		0	Results of a posted write are not yet visible to the CPU.
		1	Results of a posted write are visible to the CPU.
11 (R/NW)	WSYNCALM0	Synchronisation Status of Posted Writes to RTC_ALM0. This field indicates if the effects of a posted write to RTC_ALM0 are visible to the CPU.	
		0	Results of a posted write are not yet visible to the CPU.
		1	Results of a posted write are visible to the CPU.
10 (R/NW)	WSYNCCNT1	Synchronisation Status of Posted Writes to RTC_CNT1. This field indicates if the effects of a posted write to RTC_CNT1 are visible to the CPU.	
		0	Results of a posted write are not yet visible to the CPU.
		1	Results of a posted write are visible to the CPU.
9 (R/NW)	WSYNCCNT0	Synchronisation Status of Posted Writes to RTC_CNT0. This field indicates if the effects of a posted write to RTC_CNT0 are visible to the CPU.	
		0	Results of a posted write are not yet visible to the CPU.
		1	Results of a posted write are visible to the CPU.
8 (R/NW)	WSYNCSR0	Synchronisation Status of Posted Writes to RTC_SR0. This field indicates if the effects of a posted write to RTC_SR0 are visible to the CPU.	
		0	Results of a posted write are not yet visible to the CPU.
		1	Results of a posted write are visible to the CPU.
7 (R/NW)	WSYNCCR0	Synchronisation Status of Posted Writes to RTC_CR0. This field indicates if the effects of a posted write to RTC_CR0 are visible to the CPU.	
		0	Results of a posted write are not yet visible by the CPU.
		1	Results of a posted write are visible by the CPU.

Table 21-26: RTC_SR0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1C)	WPNDINT	Write Pending Interrupt. Sticky interrupt source which is activated whenever room frees up for the CPU to post a new write transaction to a 32 kHz sourced MMR or MMR bit field in the RTC. To enable a `RTC_SR0.WPNDINT` interrupt, set `RTC_CR0.WPNDINTEN` to 1. `RTC_SR0.WPNDINT` is cleared by writing 1 to it.	
		0	There has been no change in the pending status of any posted write transaction in the RTC since WPENDINT was last cleared.
		1	A posted write transaction has been dispatched since WPENDINT was last cleared, thus freeing up a slot for a new posted write by the CPU to the same MMR.
5 (R/W1C)	WSYNCINT	Write Synchronisation Interrupt. Sticky interrupt source which is activated whenever a posted write transaction to a 32 kHz sourced MMR or MMR bit field completes and whose effects are then visible to the CPU. To enable a `RTC_SR0.WSYNCINT` interrupt, set `RTC_CR0.WSYNCINTEN` to one. `RTC_SR0.WSYNCINT` is cleared by writing one to it.	
		0	Since the CPU last cleared `RTC_SR0.WSYNCINT`, there has been no occurrence of the effects of a posted write transaction to a 32 kHz-sourced MMR or MMR bit field have becoming newly visible to the CPU's clock domain.
		1	Since the CPU last cleared `RTC_SR0.WSYNCINT`, the effects of a posted write transaction to a 32 kHz-sourced MMR or MMR bit field have becoming newly visible to the CPU's clock domain.

Table 21-26: RTC_SR0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W1C)	WPNDERRINT	Write Pending Error Interrupt Source. Sticky interrupt source which indicates that an error has occurred because the CPU attempted to write to an RTC register while a previous write to the same register was pending execution. A maximum of one pending write transaction per MMR is supported by the RTC when such writes are to MMRs which are sourced in the 32 kHz domain. Note that if multiple write pending errors (i.e. rejected posted writes) occur, `RTC_SR0.WPNDERRINT` sticks active at the first occurrence. `RTC_SR0.WPNDERRINT` is cleared by writing one to it.	
		0	No posted write has been rejected by the RTC since `RTC_SR0.WPNDERRINT` is last cleared by the CPU.
		1	Write Rejected A posted write has been rejected by the RTC due to a previously-posted write to the same MMR which is still awaiting execution. Such a rejection has occurred since the CPU last cleared `RTC_SR0.WPNDERRINT`.
3 (R/W1C)	ISOINT	RTC Power-Domain Isolation Interrupt Source. Sticky interrupt source which indicates whether the RTC has had to activate its power-domain isolation barrier due to a power loss in the core. When the core regains power, the CPU can read ISOINT to inform itself of such a power event. `RTC_SR0.ISOINT` is cleared by the CPU by writing one to it. Note that this bit field only exists in RTC1. In RTC0, the field is reserved and reads back as zero.	
		0	The always-on RTC power domain has not activated its isolation from the core since the `RTC_SR0.ISOINT` interrupt source was last cleared by the CPU.
		1	The always-on RTC power domain has activated and subsequently de-activated its isolation from the core due to a power event. This event occurred since `RTC_SR0.ISOINT` was last cleared by the CPU.

Table 21-26: RTC_SR0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	MOD60ALMINT	Modulo-60 RTC Alarm Interrupt Source. Sticky flag which is the source of an optionally-enabled interrupt to the CPU. This interrupt is activated once every 60 increments of the integer count in RTC_CNT1 and RTC_CNT0, at a displacement of RTC_CR0.MOD60ALM increments past a modulo-60 boundary. It is enabled and its target time is configured using the RTC_CR0.MOD60ALMINTEN and RTC_CR0.MOD60ALM respectively. It is cleared by writing a value of one to its bit position. Note that this bit field only exists in RTC1. In RTC0, the field is reserved and reads back as zero.	
		0	RTC_SR0.MOD60ALMINT interrupt event has not occurred since this bit was last cleared by the CPU.
		1	RTC_SR0.MOD60ALMINT interrupt event has occurred since this bit was last cleared by the CPU.
1 (R/W1C)	ALMINT	Alarm Interrupt Source. Sticky flag which is the source of an optionally-enabled interrupt to the CPU. It indicates that an alarm event has occurred due to a match between the RTC count and alarm register values. A match is defined as the value in RTC_CNT1, RTC_CNT0 and RTC_CNT2 equating to the alarm time given by RTC_ALM1, RTC_ALM0 and RTC_ALM2. The detection of such an event is enabled by RTC_CR0.ALMEN in RTC_CR0, assuming that RTC_CR0.CNTEN is also enabled. RTC_SR0.ALMINT is cleared by writing a value of one to it.	
		0	RTC_SR0.ALMINT interrupt event has not occurred since this bit was last cleared by the CPU.
		1	RTC_SR0.ALMINT interrupt event has occurred since this bit was last cleared by the CPU.

RTC Status 1

Information on RTC operation is made available to the CPU via three status registers RTC_SR0, RTC_SR1 and RTC_SR2. These registers include all flags related to CPU interrupt sources and error conditions within the RTC.

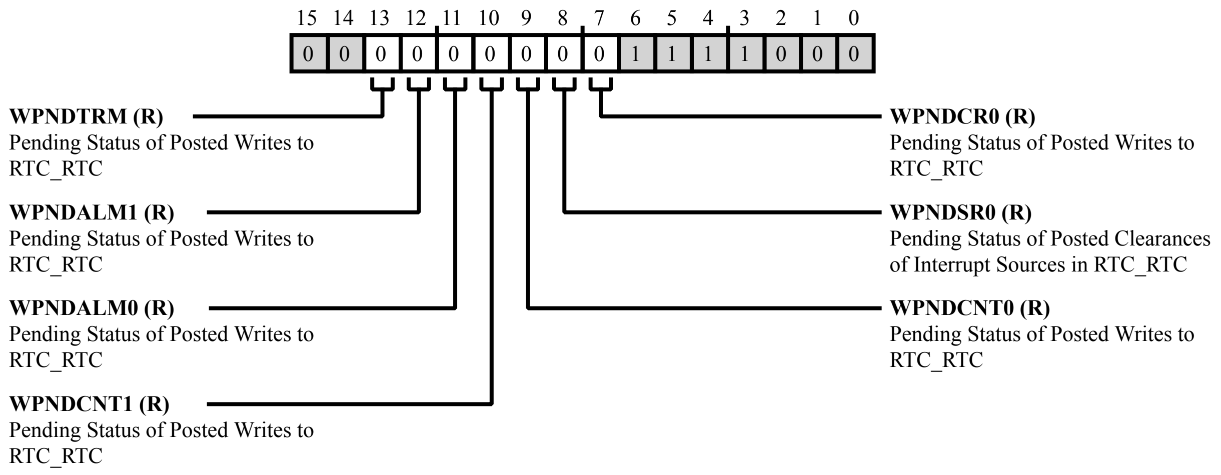

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
WPNDTRM (R)
Pending Status of Posted Writes to RTC_RTC
WPNDALM1 (R)
Pending Status of Posted Writes to RTC_RTC
WPNDALM0 (R)
Pending Status of Posted Writes to RTC_RTC
WPNDCNT1 (R)
Pending Status of Posted Writes to RTC_RTC
WPNDCR0 (R)
Pending Status of Posted Writes to RTC_RTC
WPNDSR0 (R)
Pending Status of Posted Clearances of Interrupt Sources in RTC_RTC
WPNDCNT0 (R)
Pending Status of Posted Writes to RTC_RTC

Figure 21-31: RTC_SR1 Register Diagram

Table 21-27: RTC_SR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/NW)	WPNDTRM	Pending Status of Posted Writes to RTC_TRM. Indicates if a posted register write to RTC_TRM is currently pending (buffered and enqueued) and awaiting execution and therefore no further write to the same MMR can be accepted at this time.	
		0	The RTC can accept a new posted write to the RTC_TRM MMR.
		1	A previously-posted write to RTC_TRM is still awaiting execution, so no new posting to this MMR can be accepted.
12 (R/NW)	WPNDALM1	Pending Status of Posted Writes to RTC_ALM1. Indicates if a posted register write to RTC_ALM1 is currently pending (buffered and enqueued) and awaiting execution and therefore no further write to the same MMR can be accepted at this time.	
		0	The RTC can accept a new posted write to the RTC_ALM1 MMR.
		1	A previously-posted write to RTC_ALM1 is still awaiting execution, so no new posting to this MMR can be accepted.

Table 21-27: RTC_SR1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/NW)	WPNDALM0	Pending Status of Posted Writes to RTC_ALM0. Indicates if a posted register write to RTC_ALM0 is currently pending (buffered and enqueued) and awaiting execution and therefore no further write to the same MMR can be accepted at this time.	
		0	The RTC can accept a new posted write to the RTC_ALM0 MMR.
		1	A previously-posted write to RTC_ALM0 is still awaiting execution, so no new posting to this MMR can be accepted.
10 (R/NW)	WPNDCNT1	Pending Status of Posted Writes to RTC_CNT1. Indicates if a posted register write to RTC_CNT1 is currently pending (buffered and enqueued) and awaiting execution and therefore no further write to the same MMR can be accepted at this time.	
		0	The RTC can accept a new posted write to the RTC_CNT1 MMR.
		1	A previously-posted write to RTC_CNT1 is still awaiting execution, so no new posting to this MMR can be accepted.
9 (R/NW)	WPNDCNT0	Pending Status of Posted Writes to RTC_CNT0. Indicates if a posted register write to RTC_CNT0 is currently pending (buffered and enqueued) and awaiting execution and therefore no further write to the same MMR can be accepted at this time.	
		0	The RTC can accept a new posted write to the RTC_CNT0 MMR.
		1	A previously-posted write to RTC_CNT0 is still awaiting execution, so no new posting to this MMR can be accepted.
8 (R/NW)	WPNDSR0	Pending Status of Posted Clearances of Interrupt Sources in RTC_SR0. Indicates if posted clearances of interrupt sources in RTC_SR0 are currently pending (buffered and enqueued) and awaiting execution.	
		0	The RTC can accept new posted clearances of interrupt sources in RTC_SR0 located in the 32 kHz domain.
		1	A previously-posted clearance of interrupt sources in RTC_SR0 maintained in the 32 kHz domain is still awaiting execution. Additional clearances can still be aggregated into the existing, pending transaction.

Table 21-27: RTC_SR1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	WPNDCR0	Pending Status of Posted Writes to RTC_CR0. Indicates if a posted register write to RTC_CR0 is currently pending (buffered and en-queued) and awaiting execution and therefore no further write to the same MMR can be accepted at this time.	
		0	The RTC can accept a new posted write to RTC_CR0.
		1	A previously-posted write to RTC_CR0 is still awaiting execution, so no new posting to this MMR can be ac-cepted.

RTC Status 2

RTC_SR2 is a status register which further complements the status information provided by RTC_SR0 and RTC_SR1. Note that RTC1 has full RTC_SR2 functionality, whereas RTC0 has reduced features.

All interrupt sources in RTC_SR2 are sticky, active high, level signals. Each source can be cleared by writing one to it.

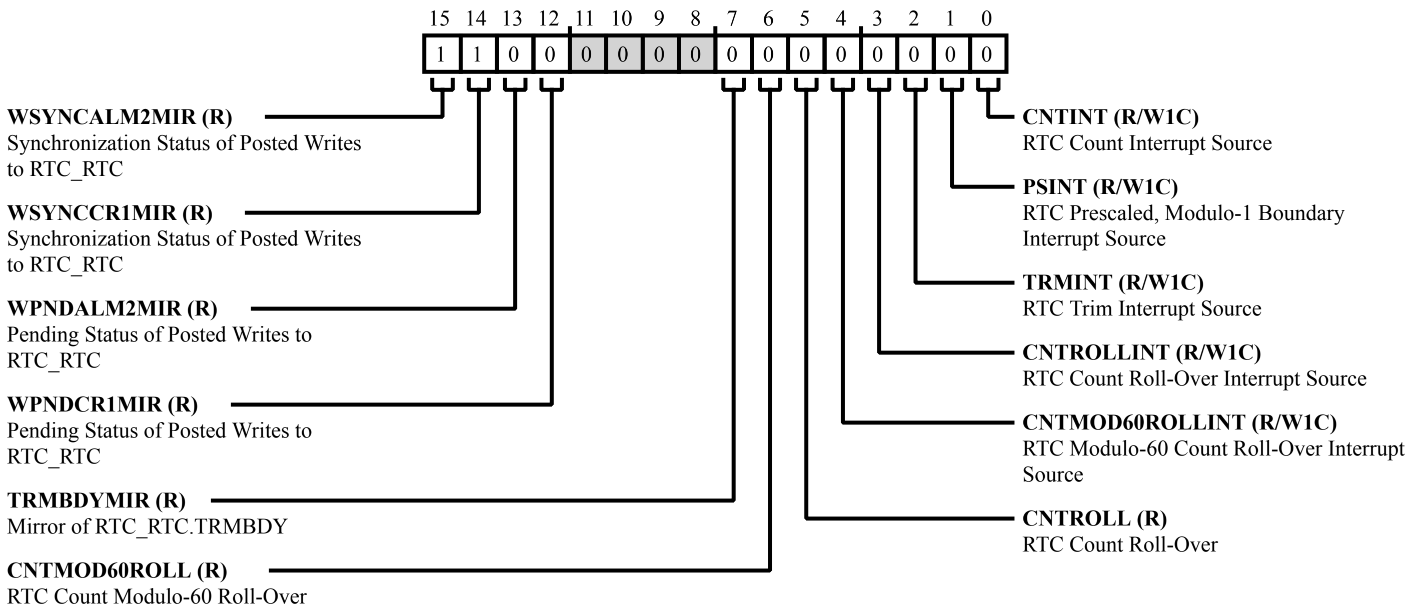

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WSYNCALM2MIR (R)
Synchronization Status of Posted Writes to RTC_RTC
WSYNCCR1MIR (R)
Synchronization Status of Posted Writes to RTC_RTC
WPNDALM2MIR (R)
Pending Status of Posted Writes to RTC_RTC
WPNDCR1MIR (R)
Pending Status of Posted Writes to RTC_RTC
TRMBDYMIR (R)
Mirror of RTC_RTC.TRMBDY
CNTMOD60ROLL (R)
RTC Count Modulo-60 Roll-Over
CNTINT (R/W1C)
RTC Count Interrupt Source
PSINT (R/W1C)
RTC Prescaled, Modulo-1 Boundary Interrupt Source
TRMINT (R/W1C)
RTC Trim Interrupt Source
CNTROLLINT (R/W1C)
RTC Count Roll-Over Interrupt Source
CNTMOD60ROLLINT (R/W1C)
RTC Modulo-60 Count Roll-Over Interrupt Source
CNTROLL (R)
RTC Count Roll-Over

Figure 21-32: RTC_SR2 Register Diagram

Table 21-28: RTC_SR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/NW)	WSYNCALM2MIR	Synchronization Status of Posted Writes to RTC_ALM2. WSYNCALM2 indicates if the effects of a posted write to RTC_ALM2 are visible to the CPU.	
		0	Results of a posted write are not yet visible.
		1	Results of a posted write are visible.
14 (R/NW)	WSYNCCR1MIR	Synchronization Status of Posted Writes to RTC_CR1. WSYNCCR1 indicates if the effects of a posted write to RTC_CR1 are visible to the CPU.	
		0	Results of a posted write are not yet visible.
		1	Results of a posted write are visible.

Table 21-28: RTC_SR2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/NW)	WPNDALM2MIR	Pending Status of Posted Writes to RTC_ALM2. Indicates if a posted register write to RTC_ALM2 is currently pending (i.e. buffered and enqueued) and awaiting execution and therefore no further write to the same MMR can be accepted at this time.	
		0	The RTC can accept a new posted write to RTC_ALM2.
		1	A previously-posted write is still awaiting execution, so no new posting to this MMR can be accepted.
12 (R/NW)	WPNDCR1MIR	Pending Status of Posted Writes to RTC_CR1. WPENDCR1 indicates if a posted register write to RTC_CR1 is currently pending (i.e. buffered and enqueued) and awaiting execution and therefore no further write to the same MMR can be accepted at this time.	
		0	The RTC can accept a new posted write to RTC_CR1.
		1	A previously-posted write is still awaiting execution, so no new posting to this MMR can be accepted.
7 (R/NW)	TRMBDYMIR	Mirror of RTC_MOD.TRMBDY. This bitfield is a read-only mirror of the value of RTC_MOD.TRMBDY MMR. It is included here so that when RTC_SR2 is read, the influence will be indicated of any trimming on a roll-over of the main RTC count or its modulo-60 equivalent.	
6 (R/NW)	CNTMOD60ROLL	RTC Count Modulo-60 Roll-Over. RTC_SR2.CNTMOD60ROLL indicates whether the current modulo-60 value of the RTC count given by RTC_MOD.CNTMOD60 MMR has come about due to a roll-over from/through its maximum possible value to/through its minimum possible value when incremented at the most recent, prescaled time unit. Note that this bit field only exists in RTC1. In RTC0, the field is reserved and reads back as zero.	
		0	The modulo-60 value of the RTC count in RTC_MOD.CNTMOD60 has not arisen due to a roll-over.
		1	The modulo-60 value of the RTC count currently in RTC_MOD.CNTMOD60 has rolled over from a value at or within trimming distance of its maximum to a value at or within trimming distance of its minimum.

Table 21-28: RTC_SR2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/NW)	CNTROLL	RTC Count Roll-Over. RTC_SR2.CNTROLL indicates whether the current value of the RTC real-time count given by RTC_CNT1, RTC_CNT0 and RTC_CNT2 has come about due to a roll-over from/through its maximum possible value to/through its minimum possible value when incremented at the most recent, prescaled time unit.	
		0	The current value of the RTC real-time count has not arisen due to a roll-over.
		1	The current value of the RTC real-time count has rolled over from a value at or within trimming distance of its maximum to a value at or within trimming distance of its minimum.
4 (R/W1C)	CNTMOD60ROLLINT	RTC Modulo-60 Count Roll-Over Interrupt Source. This source sticks active high when the modulo-60 equivalent of the integer count value in RTC_CNT1 and RTC_CNT0 rolls over from 59 to zero or is trimmed such that these values are spanned. Such a roll-over event happens every 60 prescaled increments of the RTC count, or fewer if positive (additive) trimming is active. Note that for RTC_SR2.CNTMOD60ROLLINT to cause an interrupt from the RTC, the corresponding enable bit for this interrupt fan-in term, RTC_CR1.CNTMOD60ROLLINTEN must be active high. This interrupt source is cleared by writing one to it. Note that this bit field only exists in RTC1. In RTC0, the field is reserved and reads back as zero.	
		0	The modulo-60 value of RTC_CNT1 and RTC_CNT0 in RTC_MOD.CNTMOD60 has not rolled over since RTC_SR2.CNTMOD60ROLLINT was last cleared.
		1	The modulo-60 value of of RTC_CNT1 and RTC_CNT0 in RTC_MOD.CNTMOD60 has rolled over since RTC_SR2.CNTMOD60ROLLINT was last cleared.

Table 21-28: RTC_SR2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W1C)	CNTROLLINT	RTC Count Roll-Over Interrupt Source. This source sticks active high when the integer count value in RTC_CNT1 and RTC_CNT0 rolls over from (2^32-1) to zero or is trimmed such that the trim increment causes the RTC to pass through (potentially spanning) these maximum and minimum values. Note that for RTC_SR2.CNTROLLINT to cause an interrupt from the RTC, the corresponding enable bit for this interrupt fan-in term, RTC_CR1.CNTROLLINTEN must be active high. This interrupt source is cleared by writing one to it. Note that in RTC0, the CPU can only obtain information about RTC_SR2.CNTROLLINT by reading it. RTC_SR2.CNTROLLINT cannot be enabled as an interrupt fan-in term for RTC0 because of the absence of the RTC_CR1.CNTROLLINTEN. In contrast, full interrupt capability is available for RTC1.	
		0	The integer count in RTC_CNT1 and RTC_CNT0 has not rolled over since RTC_SR2.CNTROLLINT was last cleared.
		1	The integer count in RTC_CNT1 and RTC_CNT0 has rolled over since RTC_SR2.CNTROLLINT was last cleared.
2 (R/W1C)	TRMINT	RTC Trim Interrupt Source. This source sticks active high when a trim boundary occurs at the end of an enabled trim interval and the integer value of RTC_CNT1 and RTC_CNT0 is adjusted according to the settings of RTC_TRM. Note that for RTC_SR2.TRMINT to cause an interrupt from the RTC, the corresponding enable bit for this interrupt fan-in term, RTC_CR1.TRMINTEN, must be active high. This interrupt source is cleared by writing one to it. Note that in RTC0, the CPU can only obtain information about RTC_SR2.TRMINT by reading it. RTC_SR2.TRMINT cannot be enabled as an interrupt fan-in term for RTC0 because of the absence of the RTC_CR1.TRMINTEN In contrast, full interrupt capability is available for RTC1.	
		0	An RTC trim interval boundary has not occurred since RTC_SR2.TRMINT was last cleared.
		1	An RTC trim interval boundary has occurred since RTC_SR2.TRMINT was last cleared.

Table 21-28: RTC_SR2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	PSINT	RTC Prescaled, Modulo-1 Boundary Interrupt Source. This source sticks active high whenever a prescaled RTC time unit elapses and the RTC integer count is incremented or trimmed. Note that for RTC_SR2.PSINT to cause an interrupt from the RTC, the corresponding enable bit for this interrupt fan-in term, RTC_CR1.PSINTEN, must be active high. This interrupt source is cleared by writing one to its bit position in RTC_SR2. Note that in RTC0, the CPU can only obtain information about RTC_SR2.PSINT by reading it. RTC_SR2.PSINT cannot be enabled as an interrupt fan-in term for RTC0 because of the absence of the RTC_CR1.PSINTEN In contrast, full interrupt capability is available for RTC1.	
		0	Count not Incremented or Trimmed The RTC integer count has not been incremented or trimmed since RTC_SR2.PSINT was last cleared.
		1	The RTC integer count has been incremented or trimmed since RTC_SR2.PSINT was last cleared.
0 (R/W1C)	CNTINT	RTC Count Interrupt Source. This source sticks active high whenever the integer count value in RTC_CNT1 and RTC_CNT0 changes. Note that such an event is not the same as the occurrence of a prescaled RTC time unit (RTC_SR2.PSINT), since the RTC count can either be redefined or trimmed which may or may not lead to value changes. This interrupt source is cleared by writing one to it. Note that in RTC0, the CPU can only obtain information about RTC_SR2.CNTINT by reading it. RTC_SR2.CNTINT cannot be enabled as an interrupt fan-in term for RTC0 because of the absence of the RTC_CR1.CNTINTEN. In contrast, full interrupt capability is available for RTC1.	
		0	The value of the RTC integer count has not changed since RTC_SR2.CNTINT was last cleared.
		1	The value of the RTC integer count has changed since RTC_SR2.CNTINT was last cleared.

RTC Status 3

RTC_SR3 is a status register containing write-one-to-clear, interrupt sources which stick active high whenever events occur for enabled input capture or SensorStrobe channels.

Note that this register only exists in RTC1. In RTC0, the register address is reserved and reads back as the register's reset value.

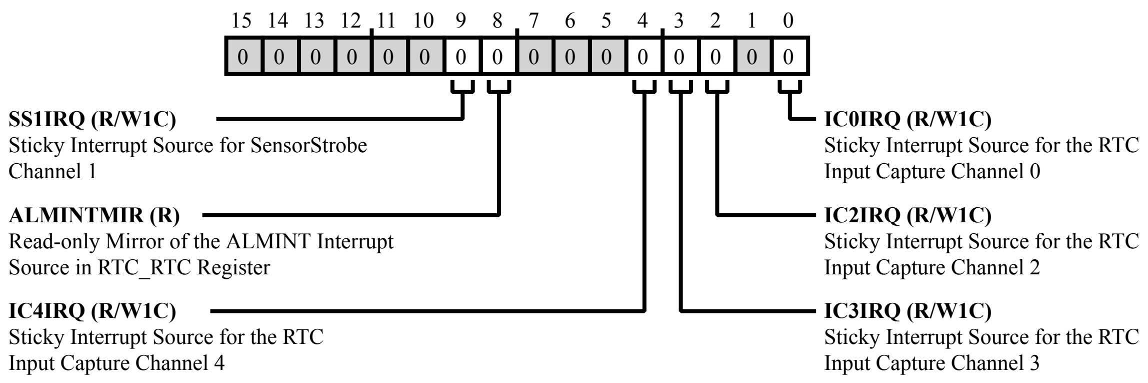

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SS1IRQ (R/W1C)
Sticky Interrupt Source for SensorStrobe Channel 1
ALMINTMIR (R)
Read-only Mirror of the ALMINT Interrupt Source in RTC_RTC Register
IC4IRQ (R/W1C)
Sticky Interrupt Source for the RTC Input Capture Channel 4
IC0IRQ (R/W1C)
Sticky Interrupt Source for the RTC Input Capture Channel 0
IC2IRQ (R/W1C)
Sticky Interrupt Source for the RTC Input Capture Channel 2
IC3IRQ (R/W1C)
Sticky Interrupt Source for the RTC Input Capture Channel 3

Figure 21-33: RTC_SR3 Register Diagram

Table 21-29: RTC_SR3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W1C)	SS1IRQ	Sticky Interrupt Source for SensorStrobe Channel 1. This interrupt source sticks high whenever a scheduled alarm event causing rising edge for the enabled (via `RTC_CR3SS.SS1EN`) SensorStrobe channel 1. `RTC_SR3.SS1IRQ` is cleared by writing one to it. If enabled via `RTC_CR3SS.SS1IRQEN`, the `RTC_SR3.SS1IRQ` source is included as a contributory term to the RTC interrupt lines sent to the CPU and the wake-up controller.	
		0	No enabled SensorStrobe event (rising edge in SensorStrobe output) has occurred on the RTC_SS1 channel since the CPU last cleared this bit.
		1	An enabled SensorStrobe event (rising edge in SensorStrobe output) has occurred on the RTC_SS1 channel since the CPU last cleared this bit.

Table 21-29: RTC_SR3 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	ALMINTMIR	Read-only Mirror of the ALMINT Interrupt Source in RTC_SR0 Register. Read-only mirror of RTC_SR0.ALMINT which is the equivalent of an SensorStrobe channel 0 interrupt source. Note that the 47-bit absolute-time alarm function in the RTC which causes RTC_SR0.ALMINT to activate doubles up as the SensorStrobe channel 0.	
		0	An ALMINT interrupt event has not occurred since the RTC_SR0.ALMINT interrupt source bit was last cleared by the CPU.
		1	An ALMINT interrupt event has occurred since the RTC_SR0.ALMINT interrupt source bit was last cleared by the CPU.
4 (R/W1C)	IC4IRQ	Sticky Interrupt Source for the RTC Input Capture Channel 4. This interrupt source in RTC_SR3 sticks high whenever an enabled (via RTC_CR2IC.IC4EN) input requester to the RTC asks for a snapshot of the RTC count to be taken for the RTC_IC4 input capture channel. It is cleared by writing a value of 1'b1 to its bit position.	
		0	No enabled input capture event has occurred on the RTC_IC4 channel since the CPU last cleared this bit.
		1	An enabled input capture event has occurred on the RTC_IC4 channel since the CPU last cleared this bit.
3 (R/W1C)	IC3IRQ	Sticky Interrupt Source for the RTC Input Capture Channel 3. This interrupt source in RTC_IC3 sticks high whenever an enabled (via RTC_CR2IC.IC3EN) input requester to the RTC asks for a snapshot of the RTC count to be taken for the RTC_IC3 input capture channel. It is cleared by writing a value of 1'b1 to its bit position.	
		0	No enabled input capture event has occurred on the RTC_IC3 channel since the CPU last cleared this bit.
		1	An enabled input capture event has occurred on the RTC_IC3 channel since the CPU last cleared this bit.
2 (R/W1C)	IC2IRQ	Sticky Interrupt Source for the RTC Input Capture Channel 2. This interrupt source in RTC_SR3 sticks high whenever an enabled (via RTC_CR2IC.IC2EN) input requester to the RTC asks for a snapshot of the RTC count to be taken for the RTC_IC2 input capture channel. It is cleared by writing a value of 1'b1 to its bit position.	
		0	No enabled input capture event has occurred on the RTC_IC2 channel since the CPU last cleared this bit.
		1	An enabled input capture event has occurred on the RTC_IC2 channel since the CPU last cleared this bit.

Table 21-29: RTC_SR3 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W1C)	IC0IRQ	Sticky Interrupt Source for the RTC Input Capture Channel 0. This interrupt source in `RTC_SR3` sticks high whenever an enabled (via `RTC_CR2IC.IC0EN`) input requester (non-CPU) to the RTC asks for a snapshot of the RTC count to be taken for the IC0 input capture channel. It is cleared by writing a value of 1'b1 to its bit position. Note that this field is unaffected by CPU-prompted snapshots, requested by writing a specific key value of 0x7627 to the GWY register.	
		0	No enabled, non-CPU input capture event has occurred on the IC0 channel since the CPU last cleared this bit.
		1	An enabled, non-CPU input capture event has occurred on the IC0 channel since the CPU last cleared this bit.

RTC Status 4

RTC_SR4 is a status register which provides the synchronization status of posted writes and posted reads to those registers related to input capture and output control which are sourced in the 32 kHz always-on half of the RTC.

Note that RTC_SR4 only exists in RTC1. In RTC0, the register address is reserved and reads back as the register's reset value.

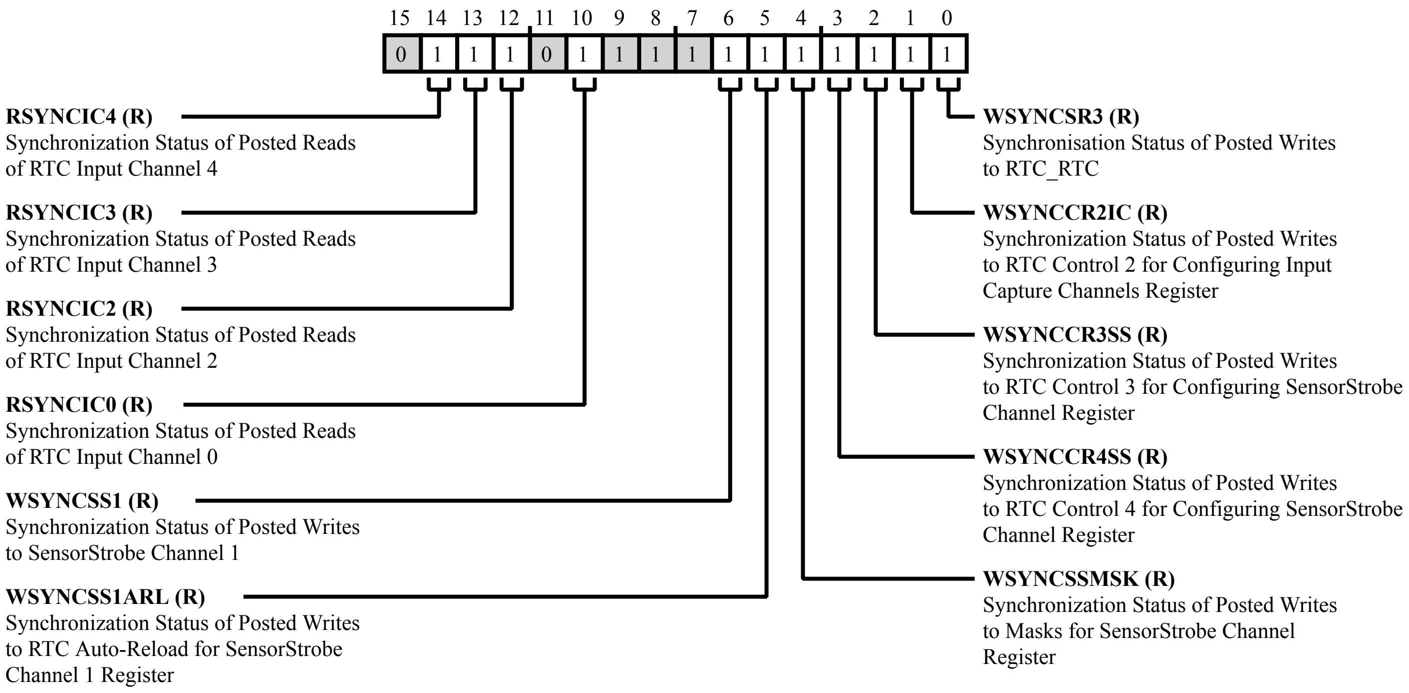

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
RSYNCIC4 (R)
Synchronization Status of Posted Reads of RTC Input Channel 4
RSYNCIC3 (R)
Synchronization Status of Posted Reads of RTC Input Channel 3
RSYNCIC2 (R)
Synchronization Status of Posted Reads of RTC Input Channel 2
RSYNCIC0 (R)
Synchronization Status of Posted Reads of RTC Input Channel 0
WSYNCSS1 (R)
Synchronization Status of Posted Writes to SensorStrobe Channel 1
WSYNCSS1ARL (R)
Synchronization Status of Posted Writes to RTC Auto-Reload for SensorStrobe Channel 1 Register
WSYNCSR3 (R)
Synchronisation Status of Posted Writes to RTC_RTC
WSYNCCR2IC (R)
Synchronization Status of Posted Writes to RTC Control 2 for Configuring Input Capture Channels Register
WSYNCCR3SS (R)
Synchronization Status of Posted Writes to RTC Control 3 for Configuring SensorStrobe Channel Register
WSYNCCR4SS (R)
Synchronization Status of Posted Writes to RTC Control 4 for Configuring SensorStrobe Channel Register
WSYNCSSMSK (R)
Synchronization Status of Posted Writes to Masks for SensorStrobe Channel Register

Figure 21-34: RTC_SR4 Register Diagram

Table 21-30: RTC_SR4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/NW)	RSYNCIC4	Synchronization Status of Posted Reads of RTC Input Channel 4. This field indicates if the effects of a read of RTC_IC4 are visible to the CPU, namely if RTC_SR6.IC4UNR has been updated to reflect the unread status of that input-capture channel.	
		0	Results of a read of IC4 are not yet visible to the CPU.
		1	Results of a read of IC4 are now visible to the CPU.

Table 21-30: RTC_SR4 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/NW)	RSYNCIC3	Synchronization Status of Posted Reads of RTC Input Channel 3. This field indicates if the effects of a read of `RTC_IC3` are visible to the CPU, namely if `RTC_SR6.IC3UNR` has been updated to reflect the unread status of that input-capture channel.	
		0	Results of a read of IC3 are not yet visible to the CPU.
		1	Results of a read of IC3 are now visible to the CPU.
12 (R/NW)	RSYNCIC2	Synchronization Status of Posted Reads of RTC Input Channel 2. This field indicates if the effects of a read of `RTC_IC2` are visible to the CPU, namely if `RTC_SR6.IC2UNR` has been updated to reflect the unread status of that input-capture channel.	
		0	Results of a read of IC2 are not yet visible to the CPU.
		1	Results of a read of IC2 are now visible to the CPU.
10 (R/NW)	RSYNCIC0	Synchronization Status of Posted Reads of RTC Input Channel 0. This field indicates if the effects of a read of all 47 bits of RTC_IC0 are visible to the CPU, namely whether `RTC_SR6.IC0UNR` has been updated to reflect the unread status of that input capture channel.	
		0	Results of a read of IC0 are not yet visible to the CPU.
		1	Results of a read of IC0 are now visible to the CPU.
6 (R/NW)	WSYNCSS1	Synchronization Status of Posted Writes to SensorStrobe Channel 1. This field indicates if the effects of a posted write to `RTC_SS1` are visible to the CPU.	
		0	Results of a posted write to `RTC_SS1` are not yet visible to the CPU.
		1	Results of a posted write to `RTC_SS1` are now visible to the CPU.
5 (R/NW)	WSYNCSS1ARL	Synchronization Status of Posted Writes to RTC Auto-Reload for SensorStrobe Channel 1 Register. This field indicates if the effects of a posted write to `RTC_SS1ARL` are visible to the CPU.	
		0	Results of a posted write to `RTC_SS1ARL` are not yet visible to the CPU.
		1	Results of a posted write to `RTC_SS1ARL` are now visible to the CPU.

Table 21-30: RTC_SR4 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/NW)	WSYNCSSMSK	Synchronization Status of Posted Writes to Masks for SensorStrobe Channel Register. This field indicates if the effects of a posted write to RTC_SSMSK are visible to the CPU.	
		0	Results of a posted write to RTC_SSMSK are not yet visible to the CPU.
		1	Results of a posted write to RTC_SSMSK are now visible to the CPU.
3 (R/NW)	WSYNCCR4SS	Synchronization Status of Posted Writes to RTC Control 4 for Configuring SensorStrobe Channel Register. This field indicates if the effects of a posted write to RTC_CR4SS are visible to the CPU.	
		0	Results of a posted write to RTC_CR4SS are not yet visible to the CPU.
		1	Results of a posted write to RTC_CR4SS are now visible to the CPU.
2 (R/NW)	WSYNCCR3SS	Synchronization Status of Posted Writes to RTC Control 3 for Configuring SensorStrobe Channel Register. This field indicates if the effects of a posted write to RTC_CR3SS are visible to the CPU.	
		0	Results of a posted write to RTC_CR3SS are not yet visible to the CPU.
		1	Results of a posted write to RTC_CR3SS are now visible to the CPU.
1 (R/NW)	WSYNCCR2IC	Synchronization Status of Posted Writes to RTC Control 2 for Configuring Input Capture Channels Register. This field indicates if the effects of a posted write to RTC_CR2IC are visible to the CPU.	
		0	Results of a posted write to RTC_CR2IC are not yet visible to the CPU.
		1	Results of a posted write to RTC_CR2IC are now visible to the CPU.

Table 21-30: RTC_SR4 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/NW)	WSYNCSR3	Synchronisation Status of Posted Writes to RTC_SR3. This field indicates if the effects of a posted write to RTC_SR3 are visible to the CPU.	
		0	Results of a posted interrupt clearance to RTC_SR3 are not yet visible to the CPU.
		1	Results of a posted interrupt clearance to RTC_SR3 are visible to the CPU.

RTC Status 5

RTC_SR5 is a status register which provides the pending (buffered and enqueued) status of posted writes to those registers related to input capture and output control which are sourced in the 32 kHz always-on half of the RTC.

Note that RTC_SR5 only exists in RTC1. In RTC0, the register address is reserved and reads back as the register's reset value.

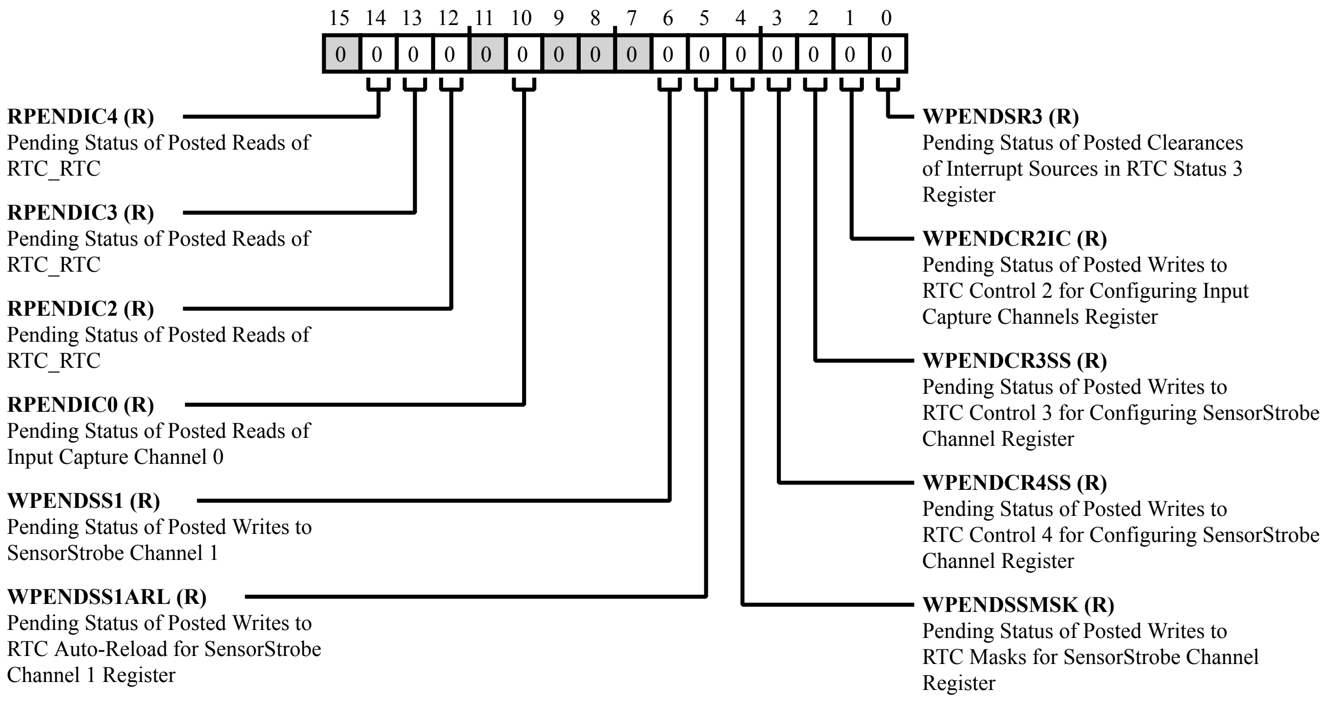

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RPENDIC4 (R)
Pending Status of Posted Reads of RTC_RTC
RPENDIC3 (R)
Pending Status of Posted Reads of RTC_RTC
RPENDIC2 (R)
Pending Status of Posted Reads of RTC_RTC
RPENDIC0 (R)
Pending Status of Posted Reads of Input Capture Channel 0
WPENDSS1 (R)
Pending Status of Posted Writes to SensorStrobe Channel 1
WPENDSS1ARL (R)
Pending Status of Posted Writes to RTC Auto-Reload for SensorStrobe Channel 1 Register
WPENDSR3 (R)
Pending Status of Posted Clearances of Interrupt Sources in RTC Status 3 Register
WPENDCR2IC (R)
Pending Status of Posted Writes to RTC Control 2 for Configuring Input Capture Channels Register
WPENDCR3SS (R)
Pending Status of Posted Writes to RTC Control 3 for Configuring SensorStrobe Channel Register
WPENDCR4SS (R)
Pending Status of Posted Writes to RTC Control 4 for Configuring SensorStrobe Channel Register
WPENDSSMSK (R)
Pending Status of Posted Writes to RTC Masks for SensorStrobe Channel Register

Figure 21-35: RTC_SR5 Register Diagram

Table 21-31: RTC_SR5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/NW)	RPENDIC4	Pending Status of Posted Reads of RTC_IC4. This field indicates if posted reads of RTC_IC4 are currently pending (buffered and enqueued) and still awaiting execution, in order to update the value of RTC_SR6.IC4UNR, sourced in the 32 kHz domain.	
		0	The RTC can accept new reads of RTC_IC4 and post this change in the unread status (to read) to the 32 kHz-sourced RTC_SR6.IC4UNR bit field.
		1	A previously-posted change (to read) in the unread status of RTC_IC4, maintained at RTC_SR6.IC4UNR in the 32kHz domain, is still awaiting execution.

Table 21-31: RTC_SR5 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/NW)	RPENDIC3	Pending Status of Posted Reads of `RTC_IC3`. This field indicates if posted reads of `RTC_IC3` are currently pending (buffered and enqueued) and still awaiting execution, in order to update the value of `RTC_SR6.IC3UNR`, sourced in the 32 kHz domain.	
		0	The RTC can accept new reads of `RTC_IC3` and post this change in the unread status (to read) to the 32kHz-sourced `RTC_SR6.IC3UNR` bit field.
		1	A previously-posted change (to read) in the unread status of `RTC_IC3`, maintained at `RTC_SR6.IC3UNR` in the 32kHz domain, is still awaiting execution.
12 (R/NW)	RPENDIC2	Pending Status of Posted Reads of `RTC_IC2`. This field indicates if posted reads of `RTC_IC2` are currently pending (buffered and enqueued) and still awaiting execution, in order to update the value of `RTC_SR6.IC2UNR`, sourced in the 32kHz domain.	
		0	The RTC can accept new reads of `RTC_IC2` and post this change in the unread status (to read) to the 32 kHz-sourced `RTC_SR6.IC2UNR` bit field.
		1	A previously-posted change (to read) in the unread status of `RTC_IC2`, maintained at `RTC_SR6.IC2UNR` in the 32 kHz domain, is still awaiting execution.
10 (R/NW)	RPENDIC0	Pending Status of Posted Reads of Input Capture Channel 0. This field indicates if posted reads of Input Capture channel 0 are currently pending (buffered and enqueued) and still awaiting execution, in order to update the value of `RTC_SR6.IC0UNR`, sourced in the 32 kHz domain.	
		0	The RTC can accept new reads of IC0 and post this change in the unread status (to read) to the 32 kHz-sourced `RTC_SR6.IC0UNR` bit field.
		1	A previously-posted change (to read) in the unread status of IC0, maintained at `RTC_SR6.IC0UNR` in the 32 kHz domain, is still awaiting execution.
6 (R/NW)	WPENDSS1	Pending Status of Posted Writes to SensorStrobe Channel 1. This field indicates if a posted register write to `RTC_SS1` is currently pending (buffered and enqueued) and awaiting execution and therefore no further write to the same MMR can be accepted at this time.	
		0	The RTC can accept a new posted write to `RTC_SS1`.
		1	A previously-posted write to `RTC_SS1` is still awaiting execution, so no new posting to this MMR can be accepted.

Table 21-31: RTC_SR5 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/NW)	WPENDSS1ARL	Pending Status of Posted Writes to RTC Auto-Reload for SensorStrobe Channel 1 Register. This field indicates if a posted register write to RTC_SS1ARL is currently pending (buffered and enqueued) and awaiting execution and therefore no further write to the same MMR can be accepted at this time.	
		0	The RTC can accept a new posted write to RTC_SS1ARL.
		1	A previously-posted write to RTC_SS1ARL is still awaiting execution, so no new posting to this MMR can be accepted.
4 (R/NW)	WPENDSSMSK	Pending Status of Posted Writes to RTC Masks for SensorStrobe Channel Register. This field indicates if a posted register write to RTC_SSMSK is currently pending (buffered and enqueued) and awaiting execution and therefore no further write to the same MMR can be accepted at this time.	
		0	The RTC can accept a new posted write to RTC_SSMSK.
		1	A previously-posted write to RTC_SSMSK is still awaiting execution, so no new posting to this MMR can be accepted.
3 (R/NW)	WPENDCR4SS	Pending Status of Posted Writes to RTC Control 4 for Configuring SensorStrobe Channel Register. This field indicates if a posted register write to RTC_CR4SS is currently pending (buffered and enqueued) and awaiting execution and therefore no further write to the same MMR can be accepted at this time.	
		0	The RTC can accept a new posted write to RTC_CR4SS.
		1	Write to RTC_CR4SS Pending A previously-posted write to RTC_CR4SS is still awaiting execution, so no new posting to this MMR can be accepted.

Table 21-31: RTC_SR5 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/NW)	WPENDCR3SS	Pending Status of Posted Writes to RTC Control 3 for Configuring SensorStrobe Channel Register. This field indicates if a posted register write to RTC_CR3SS is currently pending (buffered and enqueued) and awaiting execution and therefore no further write to the same MMR can be accepted at this time.	
		0	The RTC can accept a new posted write to RTC_CR3SS.
		1	A previously-posted write to RTC_CR3SS is still awaiting execution, so no new posting to this MMR can be accepted.
1 (R/NW)	WPENDCR2IC	Pending Status of Posted Writes to RTC Control 2 for Configuring Input Capture Channels Register. This field indicates if a posted register write to RTC_CR2IC is currently pending (buffered and enqueued) and awaiting execution and therefore no further write to the same MMR can be accepted at this time.	
		0	The RTC can accept a new posted write to RTC_CR2IC.
		1	A previously-posted write to RTC_CR2IC is still awaiting execution, so no new posting to this MMR can be accepted.
0 (R/NW)	WPENDSR3	Pending Status of Posted Clearances of Interrupt Sources in RTC Status 3 Register. This field indicates if posted clearances of interrupt sources in RTC_SR3 are currently pending (buffered and enqueued) and awaiting execution.	
		0	The RTC can accept new posted clearances of interrupt sources in RTC_SR3 located in the 32kHz domain.
		1	A previously-posted clearance of interrupt sources in RTC_SR3 maintained in the 32kHz domain is still awaiting execution. Additional clearances can still be aggregated into the existing, pending transaction.

RTC Status 6

SR6 is a status register which provides the unread status of snapshots of input-capture channels, IC0, IC2, IC3 and IC4. Note that RTC_SR6 only exists in RTC1. In RTC0, the register address is reserved and reads back as the register's reset value.

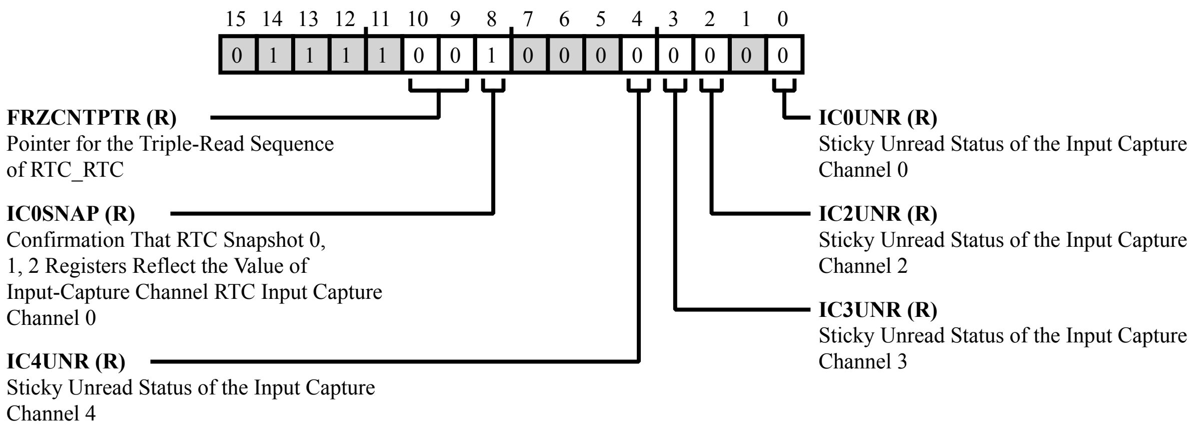

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0
FRZCNTPTR (R)
Pointer for the Triple-Read Sequence of RTC_RTC
IC0SNAP (R)
Confirmation That RTC Snapshot 0, 1, 2 Registers Reflect the Value of Input-Capture Channel RTC Input Capture Channel 0
IC4UNR (R)
Sticky Unread Status of the Input Capture Channel 4
IC0UNR (R)
Sticky Unread Status of the Input Capture Channel 0
IC2UNR (R)
Sticky Unread Status of the Input Capture Channel 2
IC3UNR (R)
Sticky Unread Status of the Input Capture Channel 3

Figure 21-36: RTC_SR6 Register Diagram

Table 21-32: RTC_SR6 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10:9 (R/NW)	FRZCNTPTR	Pointer for the Triple-Read Sequence of RTC_FRZCNT. This field indicates the sequence number for the next read in triple-read sequences of the RTC_FRZCNT register. Note that this field can be zeroed by writing a software key of value 0x9376 to RTC_GWY.	
		0	The next read of RTC_FRZCNT will cause both of the following to happen: (i) The read data of RTC_FRZCNT will be the current value of RTC_CNT0 and (ii) A coherent snapshot of the current value of RTC_CNT2 and RTC_CNT1 will be taken for return during the subsequent two reads of RTC_FRZCNT.
		1	The next read of RTC_FRZCNT will be the second in a triple-read sequence and will return the snapshot of RTC_CNT1 which was taken when the first read of RTC_FRZCNT in the sequence occurred.
		2	The next read of RTC_FRZCNT will be the third in a triple-read sequence and will return the snapshot of RTC_CNT2 which was taken when the first read of RTC_FRZCNT in the sequence occurred.

Table 21-32: RTC_SR6 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	IC0SNAP	Confirmation That RTC Snapshot 0, 1, 2 Registers Reflect the Value of Input-Capture Channel RTC Input Capture Channel 0. This flag indicates if RTC_SNAP0, RTC_SNAP1, and RTC_SNAP2 registers reflect the value of the 47-bit IC0 channel or the value of a software-initiated snapshot of the RTC count.	
		0	Software Initiated Snapshot The value which can be read in RTCSNAP0, RTCSNAP1 and RTCSNAP2 is due to a software-initiated snapshot.
		1	Snapshot Initiated by IC0 Input Capture Event The value which can be read in RTC_SNAP0, RTC_SNAP1, and RTC_SNAP2 is due to an IC0 input-capture event.
4 (R/NW)	IC4UNR	Sticky Unread Status of the Input Capture Channel 4. This field is a sticky, 32 kHz sourced flag which indicates if a new, unread snapshot exists for input capture channel IC4.	
		0	No new, unread snapshot is contained in RTC_IC4 for that input-capture channel.
		1	An unread snapshot is contained in RTC_IC4 for that input-capture channel.
3 (R/NW)	IC3UNR	Sticky Unread Status of the Input Capture Channel 3. This field is a sticky, 32 kHz sourced flag which indicates if a new, unread snapshot exists for input capture channel IC3.	
		0	No new, unread snapshot is contained in RTC_IC3 for that input-capture channel.
		1	An unread snapshot is contained in RTC_IC3 for that input-capture channel.
2 (R/NW)	IC2UNR	Sticky Unread Status of the Input Capture Channel 2. This field is a sticky, 32 kHz sourced flag which indicates if a new, unread snapshot exists for input capture channel IC2.	
		0	No new, unread snapshot is contained in RTC_IC2 for that input-capture channel.
		1	An unread snapshot is contained in RTC_IC2 for that input-capture channel.

Table 21-32: RTC_SR6 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/NW)	IC0UNR	Sticky Unread Status of the Input Capture Channel 0. This field is a sticky, 32 kHz sourced flag which indicates if a new, unread snapshot exists for input capture channel IC0.	
		0	No new, unread snapshot is contained in IC0 for that input-capture channel.
		1	An unread snapshot is contained in IC0 for that input-capture channel.

RTC Trim

RTC_TRM contains the trim value and interval for a periodic adjustment of the RTC count value to track time with the required accuracy. Trimming is enabled and disabled via the RTC_CR0.TRMEN bit. For trimming to occur, the global enable for the RTC, RTC_CR0.CNTEN, must also be active.

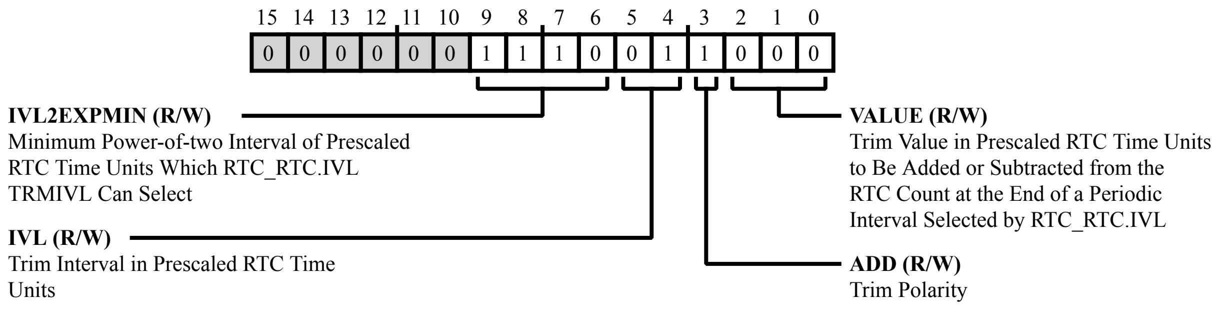

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 1 1 1 0 0 1 1 0 0 0
IVL2EXPMIN (R/W)
Minimum Power-of-two Interval of Prescaled RTC Time Units Which RTC_RTC.IVL TRMIVL Can Select
IVL (R/W)
Trim Interval in Prescaled RTC Time Units
VALUE (R/W)
Trim Value in Prescaled RTC Time Units to Be Added or Subtracted from the RTC Count at the End of a Periodic Interval Selected by RTC_RTC.IVL
ADD (R/W)
Trim Polarity

Figure 21-37: RTC_TRM Register Diagram

Table 21-33: RTC_TRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9:6 (R/W)	IVL2EXPMIN	Minimum Power-of-two Interval of Prescaled RTC Time Units Which RTC_TRM.IVL TRMIVL Can Select. RTC_TRM.IVL2EXPMIN configures the range of trim intervals which are available to the RTC at any one time. The range is 2 to the power of (RTC_TRM.IVL2EXPMIN + 0, 1, 2 or 3), selectable by RTC_TRM.IVL. The minimum supported value of RTC_TRM.IVL2EXPMIN is 2. The maximum supported value (and default) is 14. If a user configures the RTC with a value outside the range [2,14], the default value of 14 will be used.	
5:4 (R/W)	IVL	Trim Interval in Prescaled RTC Time Units. RTC_TRM.IVL specifies the interval at the end of which a periodic adjustment of RTC_TRM.VALUE prescaled RTC time units is made to the RTC count.	
		0	Trim interval is 2^RTC_TRM.IVL2EXPMIN prescaled RTC time units. Equivalent to 4 hours 33 minutes 04 seconds if prescaling to count time at 1Hz.
		1	Trim interval is 2^(RTC_TRM.IVL2EXPMIN + 1) prescaled RTC time units. Equivalent to 9 hours 06 minutes 08 seconds if prescaling to count time at 1 Hz.
		2	Trim interval is 2^(RTC_TRM.IVL2EXPMIN + 2) prescaled RTC time units. Equivalent to 18 hours 12 minutes 16 seconds if prescaling to count time at 1Hz.
		3	Trim interval is 2^(RTC_TRM.IVL2EXPMIN + 3) prescaled RTC time units. Equivalent to 36 hours 24 minutes 32 seconds if prescaling to count time at 1Hz.

Table 21-33: RTC_TRM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	ADD	Trim Polarity. RTC_TRM.ADD specifies whether RTC_TRM.VALUE is a positive (additive) or negative (subtractive) adjustment of the RTC count.	
		0	Subtract (by means of a stall) RTC_TRM.VALUE prescaled RTC time units from the RTC count at a period defined by RTC_TRM.IVL.
		1	Add Trim Value to RTC Count Add RTC_TRM.VALUE prescaled RTC time units to the RTC count at a period defined by RTC_TRM.IVL.
2:0 (R/W)	VALUE	Trim Value in Prescaled RTC Time Units to Be Added or Subtracted from the RTC Count at the End of a Periodic Interval Selected by RTC_TRM.IVL. A trim adjustment of +/- 0,1,2,3,4,5,6,7 prescaled RTC time units in the RTC count (at a period defined by RTC_TRM.IVL) can be specified using RTC_TRM.VALUE. Note that if an RTC interrupt event is configured to occur at a time which is coincidentally trimmed, the interrupt behavior is as follows. If the interrupt time is trimmed out (skipped) because of a positive trim, the interrupt issues at the end of the trim. For negative trims which coincidentally stall the RTC count at the target time for the interrupt event, the interrupt issues at the first occurrence of the target time.	
		0	Make no adjustment to the RTC count at the end of a trim interval.
		1	Add or subtract 1 prescaled RTC time unit to the RTC count.
		2	Add or subtract 2 prescaled RTC time units to the RTC count.
		3	Add or subtract 3 prescaled RTC time units to the RTC count.
		4	Add or subtract 4 prescaled RTC time units to the RTC count.
		5	Add or subtract 5 prescaled RTC time units to the RTC count.
		6	Add or subtract 6 prescaled RTC time units to the RTC count.
		7	Add or subtract 7 prescaled RTC time units to the RTC count.

22 Timer (TMR)

The ADuCM302x MCU has three general purpose timers; each with a 16-bit up/down counter and a clock prescaler of up to 8 bits, specifically with prescale options of 1, 4, 16, 64, or 256. Up to four clocks can be selected for the timer in a separate clocking block.

The timers have a maximum timeout range of ($2^{\text{Counter\_width}}$/Clock Frequency).

For a timer clock frequency of 26 MHz, the maximum timeout range is $2^{(16+8)}$ /26 MHz = 0.6 s.

For a timer clock frequency of 32 kHz, the maximum timeout range is $2^{(16+8)}$ /32 kHz = 512 s.

Some sample use cases of these timers are as follows:

- As a 1 μs clock
- As a SysTick timer to provide a reference time base of 50 to 60 μs for firmware
- As a data rate counter

  The data rate can range between 100 b/s and 400 Kb/s, depending on the wireless protocol.
- PWM generation
- PWM demodulation

TMR Features

The general purpose timers used by the ADuCM302x MCU supports the following features:

- Supports two modes of operation: free running and periodic
- The PWM generation function can be configured to be idle in logic 0 or logic 1
- The PWM output is generated in toggle mode or match mode
- Selectable IRQ source assertions can be used to reset the timers
- Allows PWM demodulation to be performed on all three timers using the reset and capture features

The timers have two modes of operation, free running and periodic. In free running mode, the counter decrements/increments from the maximum/minimum value until zero/full scale and starts again at the maximum/minimum value. In periodic mode, the counter decrements/increments from the value in the load register, `TMR_LOAD` until zero/full scale and starts again at the value stored in the load register.

The value of a counter can be read at any time by accessing its value register (`TMR_ACURCNT.VALUE`). This assuming a synchronized clock and avoids synchronization delay by returning the asynchronous (not synchronized) timer value directly . The alternative is to read `TMR_CURCNT.VALUE` which returns a slightly delayed timer value. Reading `TMR_ACURCNT.VALUE` when the timer and core operate on different clocks is unsupported, and the return value is undefined in this case.

Writing 1 to the `TMR_CTL.EN` bit initiates the up/down counter. An IRQ is generated each time the value of the counter reaches zero when counting down, or each time the counter value reaches full scale when counting up. An IRQ can be cleared by writing 1 to the `TMR_CLRINT.TIMEOUT` bit. An IRQ is also set when a selected event occurs on the event bus.

The PWM generation function may be configured to be idle in logic 0 or logic 1 by writing to the `TMR_PWMCTL.IDLESTATE` bit. An optional match value may be written to the `TMR_PWMMATCH.VALUE` bit and enabled by writing to the `TMR_PWMCTL.MATCH` bit.

The PWM output is generated in one of two modes: toggle or match. In toggle mode, the PWM output toggles on timer zero/full scale which provides a 50% duty cycle with a configurable period. Match mode provides a configurable duty cycle and a configurable period PWM output. In match mode, the PWM output starts in the idle state as configured in the `TMR_CTL` register, is then asserted when the timer and match values are equal, and is deasserted to the idle state again when the timer reaches zero/full scale.

Selectable IRQ source assertions can be used to reset the timers if the `TMR_CTL.RSTEN` bit is set. It can select one from the 16 different events as trigger source as input to the block. This interrupt source can also be used as part of a capture feature. This capture feature is also triggered by a selected IRQ source assertion. When triggered, the current timer value is copied to the `TMR_CAPTURE.VALUE` bit, and the timer continues to run. This feature can be used to determine the assertion of an event with increased accuracy.

TMR Functional Description

This section provides information on the function of the general purpose timers used by the ADuCM302x MCU.

TMR Block Diagram

The figure shows the general purpose timers used by the ADuCM302x MCU.

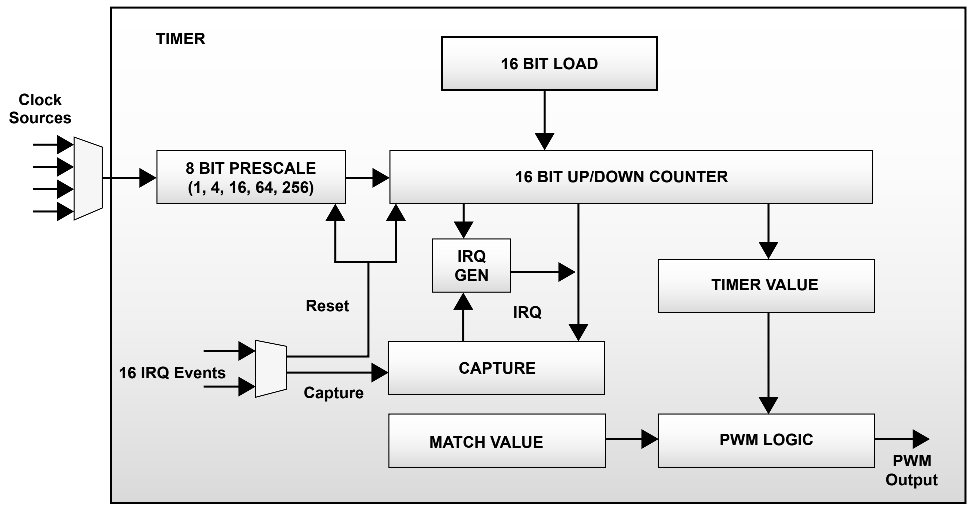


Figure 22-1: Timer Block Diagram

TMR Operating Modes

The general purpose timers used by the ADuCM302x MCU supports the following modes of operation.

Free Running Mode

In free running mode, the timer is started by writing to the `TMR_CTL.EN` bit. The timer increments from zero / full scale to full scale/zero if counting up/down (configured by `TMR_CTL.UP` bit). Full scale is $2^{16} - 1$ or 0xFFFF in hexadecimal format. On reaching full scale (or zero), a time out interrupt occurs, and `TMR_STAT.TIMEOUT` bit is set. To clear the `TMR_STAT.TIMEOUT` bit, user code must write 1 to the `TMR_CLRINT.TIMEOUT` bit. The timer is reloaded with the maximum/minimum value when the time out interrupt occurs. If the `TMR_CTL.RLD` bit is set, the timer also reloads when the `TMR_CLRINT.TIMEOUT` bit is written 1.

Periodic Mode

In periodic mode, the initial `TMR_LOAD.VALUE` value must be loaded before enabling the timer. The timer is started by writing `TMR_CTL.EN` bit. The counter value increments from the value stored in the `TMR_LOAD.VALUE` register to full scale or decrements from the value stored in the `TMR_LOAD.VALUE` register to zero, depending on the `TMR_CTL.UP` settings (count up/down). When the counter reaches full scale or zero, the timer generates an interrupt. The `TMR_LOAD.VALUE` is reloaded into the counter, and it continues counting up/ down. The timer should be disabled prior to changing the or `TMR_LOAD.VALUE` register. By default, the counter is reloaded automatically when generating the IRQ. If `TMR_CTL.RLD` bit is set to 1, the counter is also reloaded when user code writes `TMR_CLRINT.TIMEOUT` bit. To enable timing critical modifications to the load value, a nonsynchronized write address is provided at `TMR_ALOAD`. Writing `TMR_ALOAD` bypasses synchronization logic, providing finer grained control over the load value. If the `TMR_ALOAD` register is changed while the timer is enabled and running on a different clock than the core, undefined results may occur.

TMR_STAT should be read prior to writing to any timer registers following the set or clear of enable. Once TMR_STAT.PDOKbit returns zero, registers can be modified. This assures the timer is done synchronizing timer control between the core and timer clock domains. The typical synchronization time is two timer clock periods. Any modification to TMR_CTL.UP or TMR_CTL.MODE bits must be performed while the timer is disabled.

At any time, TMR_CURCNT.VALUE contains a valid value to be read, synchronized to the core clock. The TMR_CTL register enables the counter, selects the mode, selects the prescale value, and controls the event capture function.

Toggle Mode

In toggle mode, the PWM provides a 50% duty cycle output with configurable period. The PWM output is inverted when a timeout interrupt is generated by the timer. The period is therefore defined by the selected clock and prescaler. If the timer is running in the periodic mode, the PWM output also depends on the TMR_LOAD.VALUE value. Timeout events are evaluated at the end of a timer period (larger than a core clock period).

The following figures show the PWM output in toggle mode when the timer is set to count up/down.

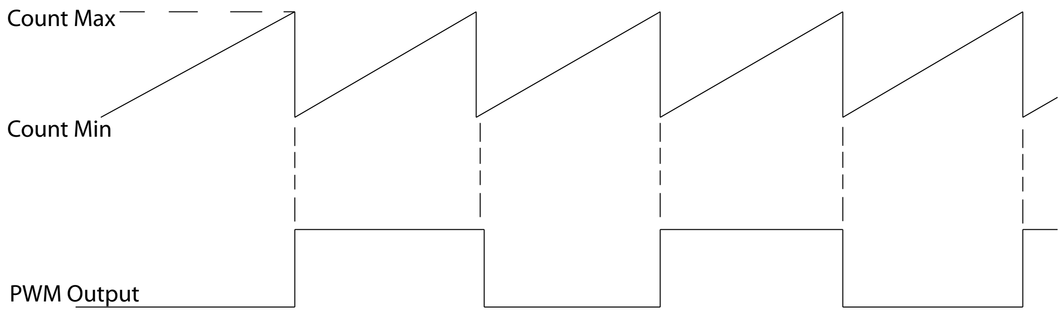


Figure 22-2: PWM Output in Toggle Mode When Timer is Set to Count Up

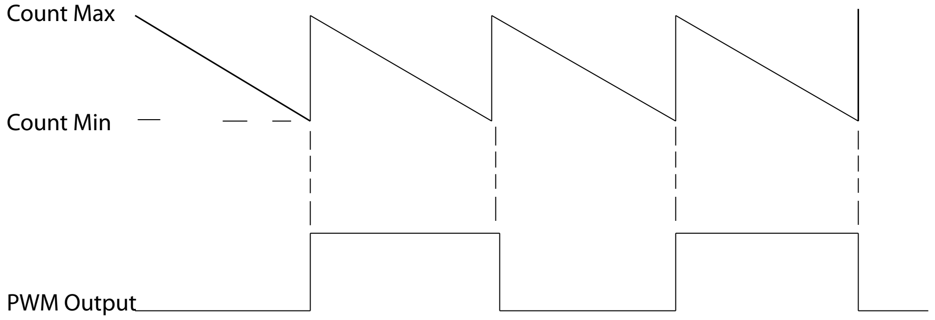


Figure 22-3: PWM Output in Toggle Mode When Timer is Set to Count Down

Match Mode

Match mode provides a configurable duty cycle and configurable period PWM output. Like toggle mode, the period is defined by the selected clock and the prescaler, as well as TMR_LOAD.VALUE (when the timer is run in periodic mode). The duty cycle is defined by the `TMR_PWMMATCH.VALUE` value. When the timers counter value is equal to the value stored in `TMR_PWMMATCH.VALUE`, the PWM output is asserted. It remains asserted until the timer reaches zero/full scale and is then deasserted. Match and timeout events are evaluated at the end of a timer period (potentially much longer than a core clock period). Match events take priority over timeout events when triggered on the same cycle, therefore a 100% duty cycle PWM may be configured by setting `TMR_PWMMATCH.VALUE` equal to `TMR_LOAD.VALUE` when running in periodic mode, or zero/full scale when set to free running mode. A 0% duty cycle is only possible when running in periodic mode and may be configured by setting `TMR_PWMMATCH.VALUE` outside of the range of the timers counter (`TMR_LOAD.VALUE` + 1 when counting down, `TMR_LOAD.VALUE` – 1 when counting up).

The following figures show the PWM output in Match mode when the timer is set to count up/down and PWM idle state set to 0.

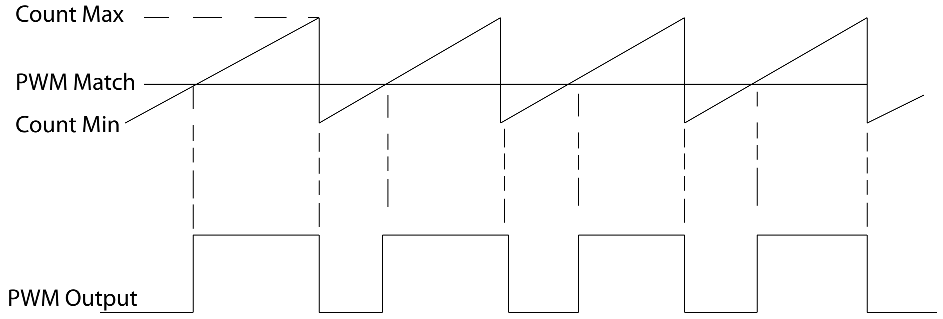


Figure 22-4: PWM Output in Match Mode When Timer is Set to Count Up and PWM Idle State is 0

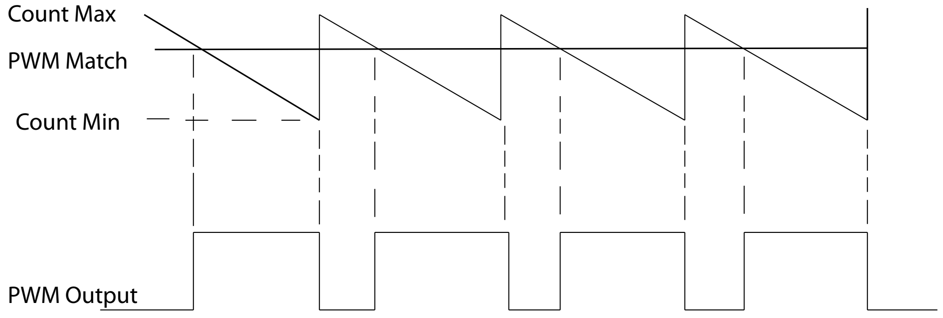


Figure 22-5: PWM Output in Match Mode When Timer is Set to Count Down and PWM Idle State is 0

For a given clock source, prescaler, and `TMR_LOAD.VALUE` value, the PWM period in toggle mode is twice that of the period in match mode, due to toggle mode inverting just once each timer counter period, while match mode both asserts and de-asserts the PWM output. Writes to the `TMR_CTL` and `TMR_PWMMATCH.VALUE` registers while the timer is running are supported. Note however that the PWM output will not reflect changes to `TMR_CTL` until the next match or timeout event. This enables modifying the PWM behavior while maintaining a deterministic PWM duty cycle. It is possible to force the PWM output to match newly written settings by writing to `TMR_CLRINT` or by disabling/enabling the timer.

NOTE: The PWM may be configured to operate in either toggle mode or match mode. In either mode, using the PWM output requires selecting the appropriate mux settings in the GPIO control module.

PWM Modulation

The PWM generation function may be configured to idle in logic 0 or logic 1 by writing the corresponding bit in respective PWM control register. An optional match value may be written to the respective PWM match register and enabled by writing another bit in PWM control register. The PWM output is generated in one of two modes: toggle or match. In toggle mode, the PWM output toggles on timer zero/full scale which provides a 50% duty cycle with a configurable period. Match mode provides a configurable duty cycle and a configurable period PWM output. In match mode, the PWM output starts in the idle state as configured in the PWM control register, is then asserted when the timer and match values are equal, and is deasserted to the idle state again when the timer reaches zero/full scale.

The sequence is as follows:

1. Set the PWM registers.
2. Enable the Timer.

PWM Demodulation

PWM demodulation can be implemented by enabling both the timer reset and capture features with the events logic, which allows separate count values to be read for the high and low levels of the PWM input. In practice, the MCU selects the appropriate PWM (GPIO) interrupt source and triggers an interrupt on the rising edge of the PWM pulse. This resets the counters and interrupts the MCU when a capture is detected. The MCU modifies the interrupt to trigger on the falling edge of the PWM pulse (before the edge occurs). This interrupt captures the high level count in `TMR_CAPTURE.VALUE` and resets the counter and interrupts the MCU again with the updated capture. The MCU reads this captured count and modifies the interrupt to operate off the positive edge again (before the edge occurs). This interrupt captures the low level count in `TMR_CAPTURE.VALUE` and resets the counter and interrupts the MCU again. The MCU can read this captured count. Comparing both read values indicates the PWM values. The PWM pulse widths must be wide enough to allow for the overhead of synchronization delays, IRQ delays, and software setup that is required between the various input edges.

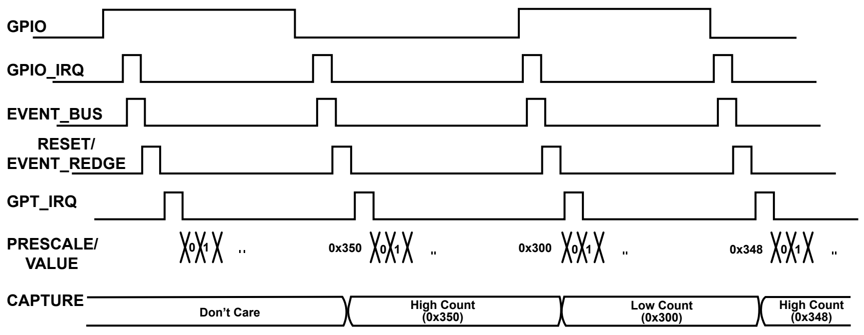


Figure 22-6: PWM Demodulation Waveforms

Clock Select

Four clocks are available for each of the four timers. The selected clock is used, along with a prescaler, to control the frequency of the timers counter logic. The available clocks are selected using the `TMR_CTL.CLK` bits. Only synchronous clocks are used. When 32 kHz clock is selected, it is synchronized to the system clock first. Clock configuration settings are part of the clock control module. For more information, refer to System Clocks.

Table 22-1: Clock Source

Clock Select	GPT Clock Source
00	PCLK
01	HFOSC
10	LFOSC
11	LFXTAL

Capture Event Function

16 interrupt events can be captured by the general purpose timers. Any one of the 16 events associated with a general purpose timer can cause a capture of the 16-bit `TMR_CURCNT` register into the 16-bit `TMR_CAPTURE` register. `TMR_CTL.EVTRANGE` has a 4-bit field selecting which of the 16 events to capture.

When the selected IRQ occurs, the `TMR_CURCNT` register is copied into the `TMR_CAPTURE.VALUE` register. The `TMR_STAT.CAPTURE` bit is set. The IRQ is cleared by writing 1 to the `TMR_CLRINT.EVTCAPT` bit. The `TMR_CAPTURE.VALUE` bit also holds its value and cannot be overwritten until the `TMR_CLRINT.EVTCAPT` bit is written to 1.

Table 22-2: Capture and Reset Event Function

Event Select Bits (**`TMR_CTL.EVTRANGE`**)	GPT0 Capture Source	GPT1 Capture Source	GPT2 Capture Source
0	Wake-up timer/RTC	UART0	Ext Int 0
1	Ext Int 0	SPI0	Ext Int 1
2	Ext Int 1	SPI1	Ext Int 2
3	Ext Int 2	SPIH	Ext Int 3
4	Ext Int 3	I2C0 slave	DMA error
5	WDT	I2C0 Master	RTC
6	VREG Over	Crypto	Timer0
7	Batt. Voltage Range	RESERVED	Timer1
8	P0_8_DIN	Xtal Osc	RESERVED
9	GPIO-IntA	PLL	RESERVED

Table 22-2: Capture and Reset Event Function (Continued)

Event Select Bits (`TMR_CTL.EVTRANGE`)	GPT0 Capture Source	GPT1 Capture Source	GPT2 Capture Source
10	GPIO-IntB	RNG	RESERVED
11	Timer1	Beeper	RESERVED
12	Timer2	Ext Int 0	RESERVED
13	Flash controller	Ext Int 1	RESERVED
14	SPORT0A	Timer0	RESERVED
15	SPORT0B	Timer2	RESERVED

TMR Interrupts and Exceptions

Refer to Events (Interrupts and Exceptions), for more information.

TMR Power Modes Behavior

All the registers in the three general purpose timers are retained in hibernate mode.

ADuCM302x TMR Register Descriptions

General Purpose Timer (TMR) contains the following registers.

Table 22-3: ADuCM302x TMR Register List

Name	Description
`TMR_ALOAD`	16-bit Load Value, Asynchronous
`TMR_ACURCNT`	16-bit Timer Value, Asynchronous
`TMR_CAPTURE`	Capture
`TMR_CLRINT`	Clear Interrupt
`TMR_CTL`	Control
`TMR_LOAD`	16-bit Load Value
`TMR_PWMCTL`	PWM Control Register
`TMR_PWMMATCH`	PWM Match Value
`TMR_STAT`	Status
`TMR_CURCNT`	16-bit Timer Value

16-bit Load Value, Asynchronous

Only use when a synchronous clock source is selected (`TMR_CTL.CLK`=00).

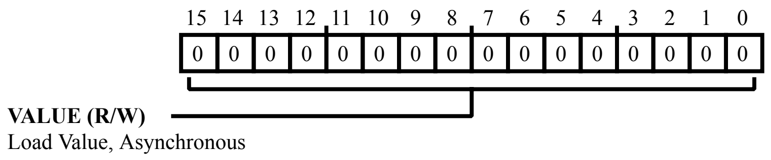


Figure 22-7: TMR_ALOAD Register Diagram

Table 22-4: TMR_ALOAD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Load Value, Asynchronous. The up/down counter is periodically loaded with this value if periodic mode is selected (`TMR_CTL.MODE`=1). Writing this bitfield takes advantage of having the timer run on PCLK by bypassing clock synchronization logic otherwise required.

16-bit Timer Value, Asynchronous

Only use when a synchronous clock source is selected (`TMR_CTL.CLK`=00).

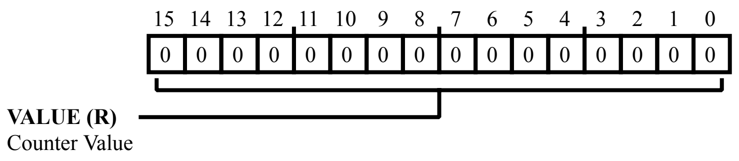


Figure 22-8: TMR_ACURCNT Register Diagram

Table 22-5: TMR_ACURCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Counter Value. Reflects the current up/down counter value. Reading this bitfield takes advantage of having the timer run on PCLK by bypassing clock synchronization logic otherwise required.

Capture

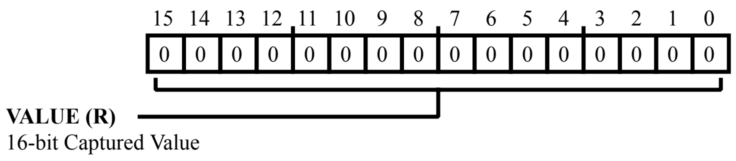


Figure 22-9: TMR_CAPTURE Register Diagram

Table 22-6: TMR_CAPTURE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	16-bit Captured Value. This bitfield will hold its value until `TMR_CLRINT.EVTCAPT` is set by user code. This bitfield will not be over written even if another event occurs without writing to the `TMR_CLRINT.EVTCAPT`.

Clear Interrupt

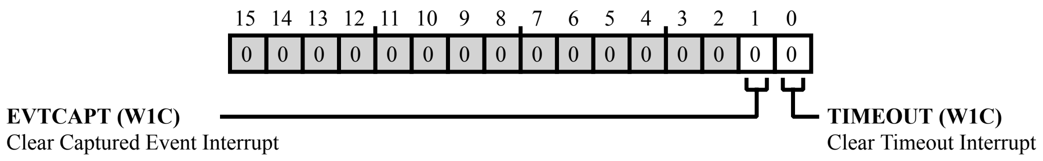


Figure 22-10: TMR_CLRINT Register Diagram

Table 22-7: TMR_CLRINT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (RX/W1C)	EVTCAPT	Clear Captured Event Interrupt. This bit is used to clear a capture event interrupt	
		0	No effect
		1	Clear the capture event interrupt
0 (RX/W1C)	TIMEOUT	Clear Timeout Interrupt. This bit is used to clear a timeout interrupt.	
		0	No effect
		1	Clears the timeout interrupt

Control

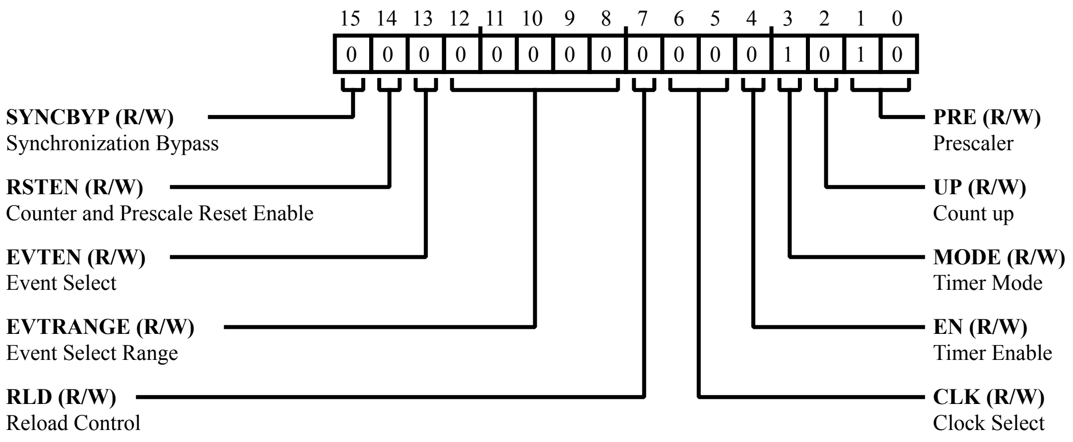


Figure 22-11: TMR_CTL Register Diagram

Table 22-8: TMR_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	SYNCBYP	Synchronization Bypass. Used to bypass the synchronization logic within the block. Use only with synchronous clocks, i.e, when the core clock (PCLK) is selected as the source for Timer clock. This bit also changes `TMR_CTL.PRE` prescaler count from 3 to 0 when this bit is set 1'b1 and `TMR_CTL.PRE` is selected to Source_Clock/1. The value of this bit will affect the prescaler count value when `TMR_CTL.PRE` is selected to Source_Clock/1. When this bit is set, Source_Clock/1 is selected and prescale count of zero will be used by the prescaler. When this bit is cleared, Source_Clock/4 is selected and prescale count of three will be used by the prescaler.	
		0	Synchronization bypass is disabled
		1	Synchronization bypass is enabled
14 (R/W)	RSTEN	Counter and Prescale Reset Enable. Used to enable and disable the reset feature. Used in conjunction with `TMR_CTL.EVTEN` and `TMR_CTL.EVTRANGE`. When a selected event occurs the 16 bit counter and 8 bit prescale are reset. This is required in PWM demodulation mode.	

Table 22-8: TMR_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	EVTEN	Event Select. Used to enable and disable the capture of events. Used in conjunction with the `TMR_CTL.EVTRANGE`: when a selected event occurs the current value of the Up/Down counter is captured in `TMR_CAPTURE`.	
		0	Events will not be captured
		1	Events will be captured
12:8 (R/W)	EVTRANGE	Event Select Range. Timer event select range (0 - 31).	
7 (R/W)	RLD	Reload Control. This bit is only used for periodic mode. It allows the user to select whether the up/down counter should be reset only on a timeout event or also when `TMR_CLRINT.TIMEOUT` is set.	
		0	Up/Down counter is only reset on a timeout event
		1	Resets the up/down counter when the Clear Timeout Interrupt bit is set
6:5 (R/W)	CLK	Clock Select. Used to select a timer clock from the four available clock sources.	
		0	Select CLK Source 0 (default)
		1	Select CLK Source 1
		2	Select CLK Source 2
		3	Select CLK Source 3
4 (R/W)	EN	Timer Enable. Used to enable and disable the timer. Clearing this bit resets the timer, including the `TMR_CURCNT` register.	
		0	Timer is disabled (default)
		1	Timer is enabled
3 (R/W)	MODE	Timer Mode. This bit is used to control whether the timer runs in periodic or free running mode. In periodic mode the up/down counter starts at the defined `TMR_LOAD.VALUE`; in free running mode the up/down counter starts at 0x0000 or 0xFFFF depending on whether the timer is counting up or down.	
		0	Timer runs in free running mode
		1	Timer runs in periodic mode (default)

Table 22-8: TMR_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	UP	Count up. Used to control whether the timer increments (counts up) or decrements (counts down) the Up/Down counter.	
		0	Timer is set to count down (default)
		1	Timer is set to count up
1:0 (R/W)	PRE	Prescaler. Controls the prescaler division factor applied to the timer's selected clock.	
		0	source_clock / 1 or source_clock/4 When `TMR_CTL.SYNCBYP` is set Source_Clock/1, and when cleared Source_Clock/4
		1	Source_clock / 16
		2	Source_clock / 64
		3	Source_clock / 256

16-bit Load Value

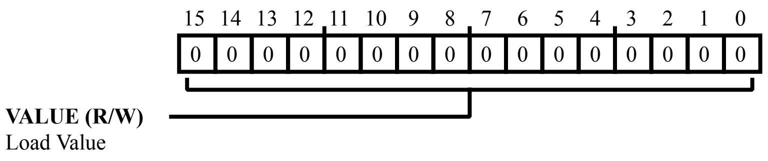


Figure 22-12: TMR_LOAD Register Diagram

Table 22-9: TMR_LOAD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Load Value. The up/down counter is periodically loaded with this value if periodic mode is selected (`TMR_CTL.MODE=1`). `TMR_LOAD.VALUE` writes during up/down counter timeout events are delayed until the event has passed.

PWM Control Register

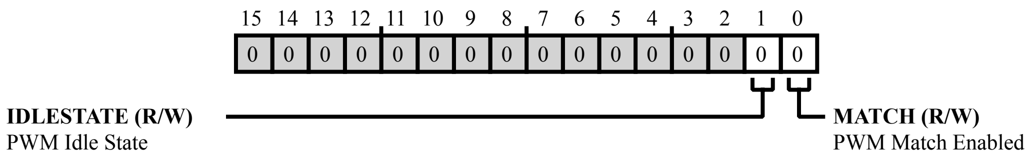


Figure 22-13: TMR_PWMCTL Register Diagram

Table 22-10: TMR_PWMCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	IDLESTATE	PWM Idle State. This bit is used to set the PWM idle state	
		0	PWM idles low
		1	PWM idles high
0 (R/W)	MATCH	PWM Match Enabled. This bit is used to control PWM operational mode	
		0	PWM in toggle mode
		1	PWM in match mode

PWM Match Value

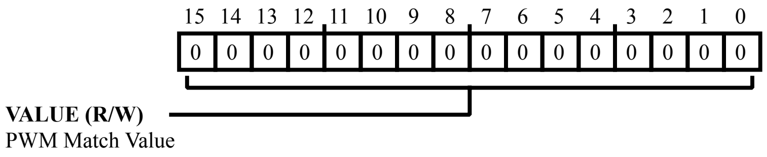


Figure 22-14: TMR_PWMMATCH Register Diagram

Table 22-11: TMR_PWMMATCH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	PWM Match Value. The value is used when the PWM is operating in match mode. The PWM output is asserted when the up/down counter is equal to this match value. PWM output is deasserted again when a timeout event occurs. If the match value is never reached, or occurs simultaneous to a timeout event, the PWM output remains idle.

Status

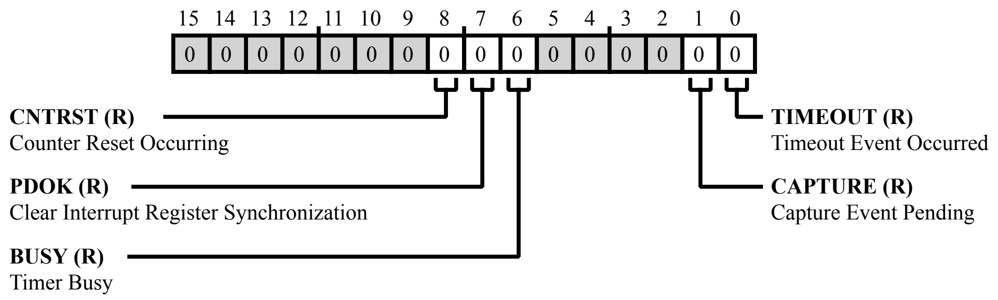


Figure 22-15: TMR_STAT Register Diagram

Table 22-12: TMR_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	CNTRST	Counter Reset Occurring. Indicates that the counter is currently being reset due to an event detection. For this to work, `TMR_CTL.RSTEN` needs to be set	
7 (R/NW)	PDOK	Clear Interrupt Register Synchronization. This bit is set automatically when the user sets `TMR_CLRINT.TIMEOUT=1`. It is cleared automatically when the clear interrupt request has crossed clock domains and taken effect in the timer clock domain	
		0	The interrupt is cleared in the timer clock domain
		1	Clear Timeout Interrupt bit is being updated in the timer clock domain
6 (R/NW)	BUSY	Timer Busy. This bit informs the user that a write to `TMR_CTL` is still crossing into the timer clock domain. This bit should be checked after writing `TMR_CTL` and further writes should be suppressed until this bit is cleared.	
		0	Timer ready to receive commands to Control Register
		1	Timer not ready to receive commands to Control Register
1 (R/NW)	CAPTURE	Capture Event Pending. A capture of the current timer value has occurred.	
		0	No capture event is pending
		1	A capture event is pending

Table 22-12: TMR_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/NW)	TIMEOUT	Timeout Event Occurred. This bit set automatically when the value of the counter reaches zero while counting down or reaches full scale when counting up. This bit is cleared when `TMR_CLRINT.TIMEOUT` is set by the user.	
		0	No timeout event has occurred
		1	A timeout event has occurred

16-bit Timer Value

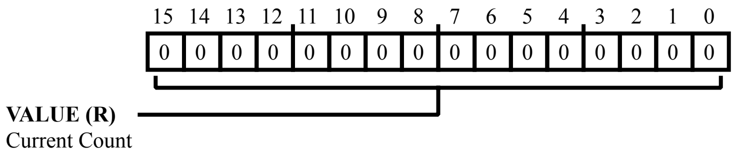


Figure 22-16: TMR_CURCNT Register Diagram

Table 22-13: TMR_CURCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Current Count. Reflects the current up/down counter value. Value delayed two PCLK cycles due to clock synchronizers.

23 Watchdog Timer (WDT)

The watchdog timer is used to recover from an illegal software state. Once enabled by the user code, it requires periodic servicing to prevent it from forcing a reset or an interrupt to the MCU.

WDT Features

The WDT is clocked by 32 kHz on-chip oscillator (LFOSC). The watchdog is clocked at all times except during reset, hibernate, shutdown and debug mode. The timer is a 16-bit down counter with a programmable prescaler. The prescaler source is selectable and can be scaled by factors of 1, 16, or 256. A WDT timeout can generate a reset or an interrupt. The `WDT_CTL.IRQ` bit selects an interrupt instead of a reset; this is intended to be a debug feature. The interrupt can be cleared by writing 0xCCCC to the `WDT_RESTART` write-only register.

WDT Functional Description

This section provides information on the function of the WDT used by the ADuCM302x MCU.

WDT Block Diagram

The figure shows the block diagram of the WDT used by the ADuCM302x MCU.

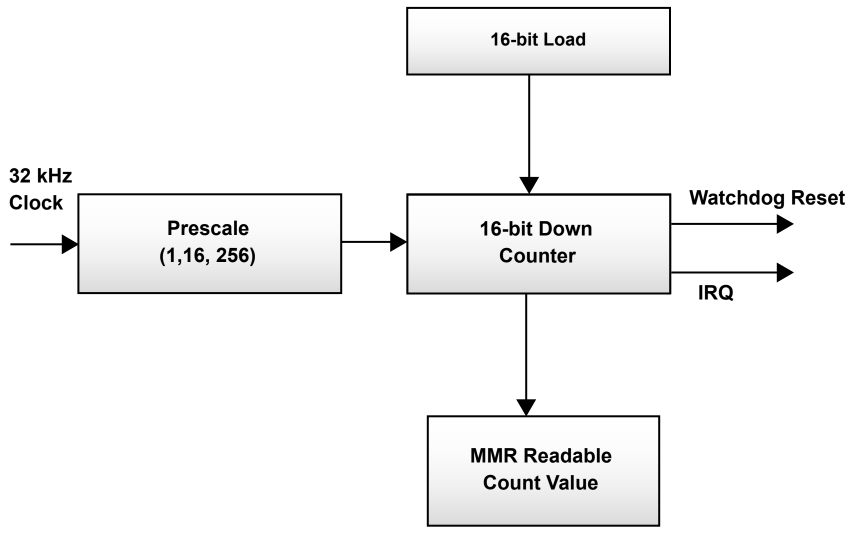


Figure 23-1: Watchdog Timer Block Diagram

WDT Operating Modes

After a valid reset, the watchdog timer is reset in the hardware as follows:

1. Set `WDT_CTL` = 0x00E9
2. Set `WDT_LOAD` = 0x1000
3. Set `WDT_CCNT` = 0x1000

This enables the watchdog timer with a timeout value of 32 seconds. This initial configuration can be modified by user code. However, setting the `WDT_CTL.EN` bit will also set the `WDT_STAT.LOCKED` bit and write-protect `WDT_CTL` and `WDT_LOAD.VALUE` (the load value). Only a reset will clear the write protection and `WDT_STAT.LOCKED` bit and allow reconfiguration of the timer.

If the `WDT_CTL` register is not modified, the user code can change `WDT_LOAD.VALUE` at any time. If the `WDT_CTL.EN` bit is cleared (prior to any write of the `WDT_CTL` register), the timer is disabled. Doing so allows the timer settings to be modified, and the timer can be re-enabled. As soon as the timer is re-enabled by a write to the `WDT_CTL` register, the watchdog configuration is locked to prevent any inadvertent modification to the register values.

If the watchdog timer is set to free-running mode (`WDT_CTL.MODE` = 0), the watchdog timer value will decrement from 0x1000 to zero, wrap around to 0x1000 and continue to decrement.

To achieve a timeout value greater or less than 0x1000 (~32 sec with default prescale = 2), periodic mode should be used (`WDT_CTL.MODE` = 1), and `WDT_LOAD.VALUE` and `WDT_CTL.PRE` written with the values corresponding to the desired timeout period. The maximum timeout is ~8 min (`WDT_LOAD.VALUE` = 0xFFFF, `WDT_CTL.PRE` = 2). When periodic mode is chosen, the timer value will decrement from the `WDT_LOAD.VALUE` value to zero, wrap around to `WDT_LOAD.VALUE` again and continue to decrement.

At any time, `WDT_CCNT.VALUE` will contain a valid value to be read, synchronized to the APB clock.

When the watchdog timer decrements to 0, a reset or an interrupt is generated. This reset can be prevented by writing the value 0xCCCC to the `WDT_RESTART` register before the expiration period. A write to `WDT_RESTART` causes the watchdog timer to reload with the `WDT_LOAD.VALUE` value (or 0x1000 if in free-running mode) immediately to begin a new timeout period and start to count again. If any value other than 0xCCCC is written, a reset or interrupt is generated.

When the timer is disabled by clearing the `WDT_CTL.EN` bit, both the internal prescaler and counter will be cleared. The counter will be reloaded with the value corresponding to the `WDT_CTL.MODE` bit setting once re-enabled.

The WDT setup is re-initialized/reset only after a POR or system reset.

The watchdog reset assertion duration will be one PCLK period when generated by a wrong `WDT_RESTART` access; the reset due to timeout will be a full 32 kHz clock period.

Watchdog Synchronization

The watchdog timer has three status bits, `WDT_STAT.COUNTING`, `WDT_STAT.LOADING`, and `WDT_STAT.CLRIRQ`, which indicate that synchronization between fast clock and slow clock domains is in progress for the `WDT_CTL`, `WDT_LOAD.VALUE` and `WDT_RESTART` registers respectively.

Writes to the `WDT_CTL` and `WDT_LOAD.VALUE` must not occur while the corresponding synchronization bit is set. However, consecutive access to the `WDT_RESTART` register with 0xCCCC value is permitted. The `WDT_RESTART` access with value 0xCCCC that occurs during the synchronization of previous access is ignored.

Watchdog Power Modes Behavior

The watchdog timer is automatically disabled in hibernate and shutdown modes. None of the watchdog timer registers are retained in hibernate mode. The user needs to reprogram the WDT counter after exiting from hibernate and shutdown modes. Otherwise, the WDT timeout value used will be the default value of around 32 seconds.

ADuCM302x WDT Register Descriptions

Watchdog Timer (WDT) contains the following registers.

Table 23-1: ADuCM302x WDT Register List

Name	Description
`WDT_CCNT`	Current Count Value
`WDT_CTL`	Control
`WDT_LOAD`	Load Value
`WDT_RESTART`	Clear Interrupt
`WDT_STAT`	Status

Current Count Value

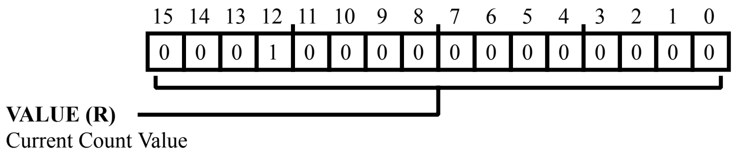


Figure 23-2: WDT_CCNT Register Diagram

Table 23-2: WDT_CCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Current Count Value. This register is synchronized to the APB clock.

Control

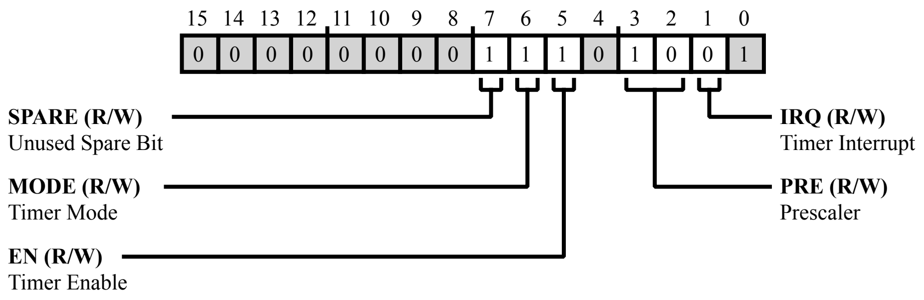


Figure 23-3: WDT_CTL Register Diagram

Table 23-3: WDT_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	SPARE	Unused Spare Bit.	
6 (R/W)	MODE	Timer Mode. Set by user to operate in periodic mode (default). Cleared by user to operate in free running mode. In free running mode, it wraps around at 0x1000.	
		0	Free running mode
		1	Periodic mode
5 (R/W)	EN	Timer Enable. Set by user to enable the timer (default). Cleared by user to disable the timer. Once set, the watchdog cannot be disabled without a system reset. This prevents software inadvertently disabling the watchdog.	
		0	WDT not enabled
		1	WDT enabled
3:2 (R/W)	PRE	Prescaler.	
		0	Source clock/1
		1	Source clock/16
		2	Source clock/256 (default)
		3	Reserved

Table 23-3: WDT_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	IRQ	Timer Interrupt. Set by user to generate an interrupt when the timer times out. This feature is provided for debug purposes and is only available when PCLK is active. Cleared by user to generate a reset on a time out (default).	
		0	WDT asserts reset when timed out
		1	WDT generates interrupt when timed out

Load Value

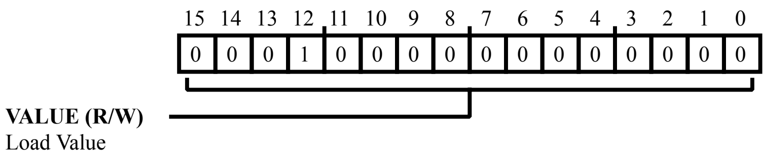


Figure 23-4: WDT_LOAD Register Diagram

Table 23-4: WDT_LOAD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Load Value.

Clear Interrupt

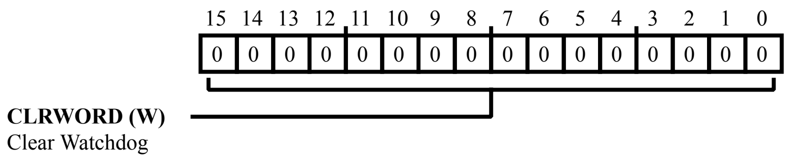
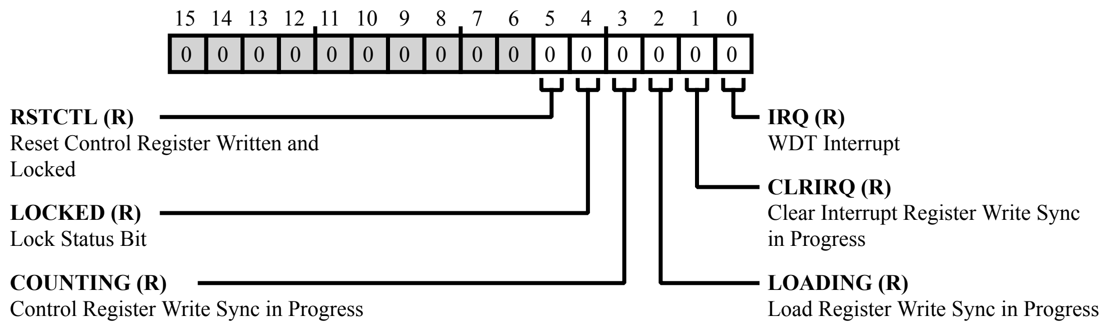


Figure 23-5: WDT_RESTART Register Diagram

Table 23-5: WDT_RESTART Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (RX/W)	CLRWORD	Clear Watchdog. User writes 0xCCCC to reset, reload, restart WDT or clear IRQ. A write of any other value causes a watchdog reset/IRQ. Write only, reads 0.

Status



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
RSTCTL (R)
Reset Control Register Written and Locked
LOCKED (R)
Lock Status Bit
COUNTING (R)
Control Register Write Sync in Progress
IRQ (R)
WDT Interrupt
CLRIRQ (R)
Clear Interrupt Register Write Sync in Progress
LOADING (R)
Load Register Write Sync in Progress

Figure 23-6: WDT_STAT Register Diagram

Table 23-6: WDT_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/NW)	RSTCTL	Reset Control Register Written and Locked.	
		0	Reset Control Register never written yet
		1	Reset Control Register written and locked
4 (R/NW)	LOCKED	Lock Status Bit. Set automatically in hardware if `WDT_CTL.EN` has been set by user code. Cleared by default and until user code sets `WDT_CTL.EN`.	
		0	Enable bit is not set by software
		1	Enable bit is set by software. WDT is locked
3 (R/NW)	COUNTING	Control Register Write Sync in Progress. Software should not write to the `WDT_CTL` register when this bit is set to avoid synchronization issues.	
		0	APB and WDT clock domains WDT_CTL configuration values match.
		1	APB T3CON register values are being synchronized to T3 clock domain.
2 (R/NW)	LOADING	Load Register Write Sync in Progress. Software should not write to the `WDT_LOAD` register when this bit is set to avoid synchronization issues.	
		0	APB and WDT clock domains `WDT_LOAD` values match.
		1	APB `WDT_LOAD` value is being synchronized to WDT clock domain.

Table 23-6: WDT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	CLRIRQ	Clear Interrupt Register Write Sync in Progress. Software should not write to the RESTART register when this bit is set to avoid synchronisation issues.	
		0	APB WDT CLRI write sync not done
		1	APB WDT CLRI write is being synced to WDT clock domain. WDT will be restarted (if 0xCCCC was written) once sync is complete.
0 (R/NW)	IRQ	WDT Interrupt. Write `WDT_RESTART` register with value 0xCCCC to clear this bit.	
		0	WDT interrupt not pending.
		1	WDT interrupt pending.

24 ADuCM302x Register List

This appendix lists Memory Mapped Register address and register names. The modules are presented in alphabetical order.

Table 24-1: ADuCM302x ADC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40007000	ADC0_CFG	ADC0 ADC Configuration	0x00000000
0x40007004	ADC0_PWRUP	ADC0 ADC Power-up Time	0x0000020E
0x40007008	ADC0_CAL_WORD	ADC0 Calibration Word	0x00000040
0x4000700C	ADC0_CNV_CFG	ADC0 ADC Conversion Configuration	0x00000000
0x40007010	ADC0_CNV_TIME	ADC0 ADC Conversion Time	0x00000000
0x40007014	ADC0_AVG_CFG	ADC0 Averaging Configuration	0x00004008
0x40007020	ADC0_IRQ_EN	ADC0 Interrupt Enable	0x00000000
0x40007024	ADC0_STAT	ADC0 ADC Status	0x00000000
0x40007028	ADC0_OVF	ADC0 Overflow of Output Registers	0x00000000
0x4000702C	ADC0_ALERT	ADC0 Alert Indication	0x00000000
0x40007030	ADC0_CH0_OUT	ADC0 Conversion Result Channel 0	0x00000000
0x40007034	ADC0_CH1_OUT	ADC0 Conversion Result Channel 1	0x00000000
0x40007038	ADC0_CH2_OUT	ADC0 Conversion Result Channel 2	0x00000000
0x4000703C	ADC0_CH3_OUT	ADC0 Conversion Result Channel 3	0x00000000
0x40007040	ADC0_CH4_OUT	ADC0 Conversion Result Channel 4	0x00000000
0x40007044	ADC0_CH5_OUT	ADC0 Conversion Result Channel 5	0x00000000
0x40007048	ADC0_CH6_OUT	ADC0 Conversion Result Channel 6	0x00000000
0x4000704C	ADC0_CH7_OUT	ADC0 Conversion Result Channel 7	0x00000000
0x40007050	ADC0_BAT_OUT	ADC0 Battery Monitoring Result	0x00000000
0x40007054	ADC0_TMP_OUT	ADC0 Temperature Result	0x00000000

Table 24-1: ADuCM302x ADC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40007058	ADC0_TMP2_OUT	ADC0 Temperature Result 2	0x00000000
0x4000705C	ADC0_DMA_OUT	ADC0 DMA Output Register	0x00000000
0x40007060	ADC0_LIM0_LO	ADC0 Channel 0 Low Limit	0x00000000
0x40007064	ADC0_LIM0_HI	ADC0 Channel 0 High Limit	0x00000FFF
0x40007068	ADC0_HYS0	ADC0 Channel 0 Hysteresis	0x00000000
0x40007070	ADC0_LIM1_LO	ADC0 Channel 1 Low Limit	0x00000000
0x40007074	ADC0_LIM1_HI	ADC0 Channel 1 High Limit	0x00000FFF
0x40007078	ADC0_HYS1	ADC0 Channel 1 Hysteresis	0x00000000
0x40007080	ADC0_LIM2_LO	ADC0 Channel 2 Low Limit	0x00000000
0x40007084	ADC0_LIM2_HI	ADC0 Channel 2 High Limit	0x00000FFF
0x40007088	ADC0_HYS2	ADC0 Channel 2 Hysteresis	0x00000000
0x40007090	ADC0_LIM3_LO	ADC0 Channel 3 Low Limit	0x00000000
0x40007094	ADC0_LIM3_HI	ADC0 Channel 3 High Limit	0x00000FFF
0x40007098	ADC0_HYS3	ADC0 Channel 3 Hysteresis	0x00000000
0x400070C0	ADC0_CFG1	ADC0 Reference Buffer Low Power Mode	0x00000400

Table 24-2: ADuCM302x BEEP0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40005C00	BEEP0_CFG	BEEP0 Beeper Configuration	0x00000000
0x40005C04	BEEP0_STAT	BEEP0 Beeper Status	0x00000000
0x40005C08	BEEP0_TONEA	BEEP0 Tone A Data	0x00000001
0x40005C0C	BEEP0_TONEB	BEEP0 Tone B Data	0x00000001

Table 24-3: ADuCM302x BUSM0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4004C800	BUSM0_ARBIT0	BUSM0 Arbitration Priority Configuration for FLASH and SRAM0	0x00240024
0x4004C804	BUSM0_ARBIT1	BUSM0 Arbitration Priority Configuration for SRAM1 and SIP	0x00240024
0x4004C808	BUSM0_ARBIT2	BUSM0 Arbitration Priority Configuration for APB32 and APB16	0x00240024

Table 24-3: ADuCM302x BUSM0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4004C80C	BUSM0_ARBIT3	BUSM0 Arbitration Priority Configuration for APB16 priority for core and for DMA1	0x00010002

Table 24-4: ADuCM302x CLKG0_OSC MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4004C10C	CLKG0_OSC_KEY	CLKG0_OSC Key Protection for CLKG_OSC_CTL	0x00000000
0x4004C110	CLKG0_OSC_CTL	CLKG0_OSC Oscillator Control	0x00000002

Table 24-5: ADuCM302x CLKG0_CLK MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4004C300	CLKG0_CLK_CTL0	CLKG0_CLK Miscellaneous Clock Settings	0x00000078
0x4004C304	CLKG0_CLK_CTL1	CLKG0_CLK Clock Dividers	0x00100404
0x4004C30C	CLKG0_CLK_CTL3	CLKG0_CLK System PLL	0x0000691A
0x4004C314	CLKG0_CLK_CTL5	CLKG0_CLK User Clock Gating Control	0x0000001F
0x4004C318	CLKG0_CLK_STAT0	CLKG0_CLK Clocking Status	0x00000000

Table 24-6: ADuCM302x CRC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40040000	CRC0_CTL	CRC0 CRC Control	0x10000000
0x40040004	CRC0_IPDATA	CRC0 Input Data Word	0x00000000
0x40040008	CRC0_RESULT	CRC0 CRC Result	0x00000000
0x4004000C	CRC0_POLY	CRC0 Programmable CRC Polynomial	0x04C11DB7
0x40040010	CRC0_IPBITS[n]	CRC0 Input Data Bits	0x00000000
0x40040010	CRC0_IPBYTE	CRC0 Input Data Byte	0x00000000
0x40040011	CRC0_IPBITS[n]	CRC0 Input Data Bits	0x00000000
0x40040012	CRC0_IPBITS[n]	CRC0 Input Data Bits	0x00000000
0x40040013	CRC0_IPBITS[n]	CRC0 Input Data Bits	0x00000000
0x40040014	CRC0_IPBITS[n]	CRC0 Input Data Bits	0x00000000
0x40040015	CRC0_IPBITS[n]	CRC0 Input Data Bits	0x00000000
0x40040016	CRC0_IPBITS[n]	CRC0 Input Data Bits	0x00000000

Table 24-6: ADuCM302x CRC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40040017	CRC0_IPBITS[n]	CRC0 Input Data Bits	0x00000000

Table 24-7: ADuCM302x CRYPT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40044000	CRYPT0_CFG	CRYPT0 Configuration Register	0x10000000
0x40044004	CRYPT0_DATALEN	CRYPT0 Payload Data Length	0x00000000
0x40044008	CRYPT0_PREFIXLEN	CRYPT0 Authentication Data Length	0x00000000
0x4004400C	CRYPT0_INTEN	CRYPT0 Interrupt Enable Register	0x00000000
0x40044010	CRYPT0_STAT	CRYPT0 Status Register	0x00000001
0x40044014	CRYPT0_INBUF	CRYPT0 Input Buffer	0x00000000
0x40044018	CRYPT0_OUTBUF	CRYPT0 Output Buffer	0x00000000
0x4004401C	CRYPT0_NONCE0	CRYPT0 Nonce Bits [31:0]	0x00000000
0x40044020	CRYPT0_NONCE1	CRYPT0 Nonce Bits [63:32]	0x00000000
0x40044024	CRYPT0_NONCE2	CRYPT0 Nonce Bits [95:64]	0x00000000
0x40044028	CRYPT0_NONCE3	CRYPT0 Nonce Bits [127:96]	0x00000000
0x4004402C	CRYPT0_AESKEY0	CRYPT0 AES Key Bits [31:0]	0x00000000
0x40044030	CRYPT0_AESKEY1	CRYPT0 AES Key Bits [63:32]	0x00000000
0x40044034	CRYPT0_AESKEY2	CRYPT0 AES Key Bits [95:64]	0x00000000
0x40044038	CRYPT0_AESKEY3	CRYPT0 AES Key Bits [127:96]	0x00000000
0x4004403C	CRYPT0_AESKEY4	CRYPT0 AES Key Bits [159:128]	0x00000000
0x40044040	CRYPT0_AESKEY5	CRYPT0 AES Key Bits [191:160]	0x00000000
0x40044044	CRYPT0_AESKEY6	CRYPT0 AES Key Bits [223:192]	0x00000000
0x40044048	CRYPT0_AESKEY7	CRYPT0 AES Key Bits [255:224]	0x00000000
0x4004404C	CRYPT0_CNTRINIT	CRYPT0 Counter Initialization Vector	0x00000000
0x40044050	CRYPT0_SHAH0	CRYPT0 SHA Bits [31:0]	0x6A09E667
0x40044054	CRYPT0_SHAH1	CRYPT0 SHA Bits [63:32]	0xBB67AE85
0x40044058	CRYPT0_SHAH2	CRYPT0 SHA Bits [95:64]	0x3C6EF372
0x4004405C	CRYPT0_SHAH3	CRYPT0 SHA Bits [127:96]	0xA54FF53A
0x40044060	CRYPT0_SHAH4	CRYPT0 SHA Bits [159:128]	0x510E527F
0x40044064	CRYPT0_SHAH5	CRYPT0 SHA Bits [191:160]	0x9B05688C

Table 24-7: ADuCM302x CRYPT0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40044068	CRYPT0_SHAH6	CRYPT0 SHA Bits [223:192]	0x1F83D9AB
0x4004406C	CRYPT0_SHAH7	CRYPT0 SHA Bits [255:224]	0x5BE0CD19
0x40044070	CRYPT0_SHA_LAST_WORD	CRYPT0 SHA Last Word and Valid Bits Information	0x00000000
0x40044074	CRYPT0_CCM_NUM_VALID_BYTES	CRYPT0 NUM_VALID_BYTES	0x00000000

Table 24-8: ADuCM302x DMA0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40010000	DMA0_STAT	DMA0 DMA Status	0x00180000
0x40010004	DMA0_CFG	DMA0 DMA Configuration	0x00000000
0x40010008	DMA0_PDBPTR	DMA0 DMA Channel Primary Control Database Pointer	0x00000000
0x4001000C	DMA0_ADBPTR	DMA0 DMA Channel Alternate Control Database Pointer	0x00000200
0x40010014	DMA0_SWREQ	DMA0 DMA Channel Software Request	0x00000000
0x40010020	DMA0_RMSK_SET	DMA0 DMA Channel Request Mask Set	0x00000000
0x40010024	DMA0_RMSK_CLR	DMA0 DMA Channel Request Mask Clear	0x00000000
0x40010028	DMA0_EN_SET	DMA0 DMA Channel Enable Set	0x00000000
0x4001002C	DMA0_EN_CLR	DMA0 DMA Channel Enable Clear	0x00000000
0x40010030	DMA0_ALT_SET	DMA0 DMA Channel Primary Alternate Set	0x00000000
0x40010034	DMA0_ALT_CLR	DMA0 DMA Channel Primary Alternate Clear	0x00000000
0x40010038	DMA0_PRI_SET	DMA0 DMA Channel Priority Set	0x00000000
0x4001003C	DMA0_PRI_CLR	DMA0 DMA Channel Priority Clear	0x00000000
0x40010048	DMA0_ERRCHNL_CLR	DMA0 DMA per Channel Error Clear	0x00000000
0x4001004C	DMA0_ERR_CLR	DMA0 DMA Bus Error Clear	0x00000000
0x40010050	DMA0_INVALIDDESC_CLR	DMA0 DMA per Channel Invalid Descriptor Clear	0x00000000
0x40010800	DMA0_BS_SET	DMA0 DMA Channel Bytes Swap Enable Set	0x00000000
0x40010804	DMA0_BS_CLR	DMA0 DMA Channel Bytes Swap Enable Clear	0x00000000
0x40010810	DMA0_SRCADDR_SET	DMA0 DMA Channel Source Address Decrement Enable Set	0x00000000

Table 24-8: ADuCM302x DMA0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40010814	DMA0_SRCADDR_CLR	DMA0 DMA Channel Source Address Decrement Enable Clear	0x00000000
0x40010818	DMA0_DSTADDR_SET	DMA0 DMA Channel Destination Address Decrement Enable Set	0x00000000
0x4001081C	DMA0_DSTADDR_CLR	DMA0 DMA Channel Destination Address Decrement Enable Clear	0x00000000
0x40010FE0	DMA0_REVID	DMA0 DMA Controller Revision ID	0x00000002

Table 24-9: ADuCM302x XINT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4004C080	XINT0_CFG0	XINT0 External Interrupt Configuration	0x00200000
0x4004C084	XINT0_EXT_STAT	XINT0 External Wakeup Interrupt Status	0x00000000
0x4004C090	XINT0_CLR	XINT0 External Interrupt Clear	0x00000000
0x4004C094	XINT0_NMICLR	XINT0 Non-Maskable Interrupt Clear	0x00000000

Table 24-10: ADuCM302x FLCC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40018000	FLCC0_STAT	FLCC0 Status	0x00000000
0x40018004	FLCC0_IEN	FLCC0 Interrupt Enable	0x00000060
0x40018008	FLCC0_CMD	FLCC0 Command	0x00000000
0x4001800C	FLCC0_KH_ADDR	FLCC0 Write Address	0x00000000
0x40018010	FLCC0_KH_DATA0	FLCC0 Write Lower Data	0xFFFFFFFF
0x40018014	FLCC0_KH_DATA1	FLCC0 Write Upper Data	0xFFFFFFFF
0x40018018	FLCC0_PAGE_ADDR0	FLCC0 Lower Page Address	0x00000000
0x4001801C	FLCC0_PAGE_ADDR1	FLCC0 Upper Page Address	0x00000000
0x40018020	FLCC0_KEY	FLCC0 Key	0x00000000
0x40018024	FLCC0_WR_ABORT_ADDR	FLCC0 Write Abort Address	0x00000000
0x40018028	FLCC0_WRPROT	FLCC0 Write Protection	0xFFFFFFFF
0x4001802C	FLCC0_SIGNATURE	FLCC0 Signature	0x00000000
0x40018030	FLCC0_UCFG	FLCC0 User Configuration	0x00000000

Table 24-10: ADuCM302x FLCC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40018034	FLCC0_TIME_PARAM0	FLCC0 Time Parameter 0	0xB8955950
0x40018038	FLCC0_TIME_PARAM1	FLCC0 Time Parameter 1	0x00000004
0x4001803C	FLCC0_ABORT_EN_LO	FLCC0 IRQ Abort Enable (Lower Bits)	0x00000000
0x40018040	FLCC0_ABORT_EN_HI	FLCC0 IRQ Abort Enable (Upper Bits)	0x00000000
0x40018044	FLCC0_ECC_CFG	FLCC0 ECC Configuration	0x00000002
0x40018048	FLCC0_ECC_ADDR	FLCC0 ECC Status (Address)	0x00000000
0x40018050	FLCC0_POR_SEC	FLCC0 Flash Security	0x00000000
0x40018054	FLCC0_VOL_CFG	FLCC0 Volatile Flash Configuration	0x00000001

Table 24-11: ADuCM302x FLCC0_CACHE MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40018058	FLCC0_CACHE_STAT	FLCC0_CACHE Cache Status	0x00000000
0x4001805C	FLCC0_CACHE_SETUP	FLCC0_CACHE Cache Setup	0x00000000
0x40018060	FLCC0_CACHE_KEY	FLCC0_CACHE Cache Key	0x00000000

Table 24-12: ADuCM302x GPIO0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40020000	GPIO0_CFG	GPIO0 Port Configuration	0x00000000
0x40020004	GPIO0_OEN	GPIO0 Port Output Enable	0x00000000
0x40020008	GPIO0_PE	GPIO0 Port Output Pull-up/Pull-down Enable	0x000000C0
0x4002000C	GPIO0_IEN	GPIO0 Port Input Path Enable	0x00000000
0x40020010	GPIO0_IN	GPIO0 Port Registered Data Input	0x00000000
0x40020014	GPIO0_OUT	GPIO0 Port Data Output	0x00000000
0x40020018	GPIO0_SET	GPIO0 Port Data Out Set	0x00000000
0x4002001C	GPIO0_CLR	GPIO0 Port Data Out Clear	0x00000000
0x40020020	GPIO0_TGL	GPIO0 Port Pin Toggle	0x00000000
0x40020024	GPIO0_POL	GPIO0 Port Interrupt Polarity	0x00000000
0x40020028	GPIO0_IENA	GPIO0 Port Interrupt A Enable	0x00000000
0x4002002C	GPIO0_IENB	GPIO0 Port Interrupt B Enable	0x00000000
0x40020030	GPIO0_INT	GPIO0 Port Interrupt Status	0x00000000

Table 24-12: ADuCM302x GPIO0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40020034	GPIO0_DS	GPIO0 Port Drive Strength Select	0x00000000

Table 24-13: ADuCM302x GPIO1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40020040	GPIO1_CFG	GPIO1 Port Configuration	0x00000000
0x40020044	GPIO1_OEN	GPIO1 Port Output Enable	0x00000000
0x40020048	GPIO1_PE	GPIO1 Port Output Pull-up/Pull-down Enable	0x000000C0
0x4002004C	GPIO1_IEN	GPIO1 Port Input Path Enable	0x00000000
0x40020050	GPIO1_IN	GPIO1 Port Registered Data Input	0x00000000
0x40020054	GPIO1_OUT	GPIO1 Port Data Output	0x00000000
0x40020058	GPIO1_SET	GPIO1 Port Data Out Set	0x00000000
0x4002005C	GPIO1_CLR	GPIO1 Port Data Out Clear	0x00000000
0x40020060	GPIO1_TGL	GPIO1 Port Pin Toggle	0x00000000
0x40020064	GPIO1_POL	GPIO1 Port Interrupt Polarity	0x00000000
0x40020068	GPIO1_IENA	GPIO1 Port Interrupt A Enable	0x00000000
0x4002006C	GPIO1_IENB	GPIO1 Port Interrupt B Enable	0x00000000
0x40020070	GPIO1_INT	GPIO1 Port Interrupt Status	0x00000000
0x40020074	GPIO1_DS	GPIO1 Port Drive Strength Select	0x00000000

Table 24-14: ADuCM302x GPIO2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40020080	GPIO2_CFG	GPIO2 Port Configuration	0x00000000
0x40020084	GPIO2_OEN	GPIO2 Port Output Enable	0x00000000
0x40020088	GPIO2_PE	GPIO2 Port Output Pull-up/Pull-down Enable	0x000000C0
0x4002008C	GPIO2_IEN	GPIO2 Port Input Path Enable	0x00000000
0x40020090	GPIO2_IN	GPIO2 Port Registered Data Input	0x00000000
0x40020094	GPIO2_OUT	GPIO2 Port Data Output	0x00000000
0x40020098	GPIO2_SET	GPIO2 Port Data Out Set	0x00000000
0x4002009C	GPIO2_CLR	GPIO2 Port Data Out Clear	0x00000000
0x400200A0	GPIO2_TGL	GPIO2 Port Pin Toggle	0x00000000

Table 24-14: ADuCM302x GPIO2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400200A4	GPIO2_POL	GPIO2 Port Interrupt Polarity	0x00000000
0x400200A8	GPIO2_IENA	GPIO2 Port Interrupt A Enable	0x00000000
0x400200AC	GPIO2_IENB	GPIO2 Port Interrupt B Enable	0x00000000
0x400200B0	GPIO2_INT	GPIO2 Port Interrupt Status	0x00000000
0x400200B4	GPIO2_DS	GPIO2 Port Drive Strength Select	0x00000000

Table 24-15: ADuCM302x TMR0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40000000	TMR0_LOAD	TMR0 16-bit Load Value	0x00000000
0x40000004	TMR0_CURCNT	TMR0 16-bit Timer Value	0x00000000
0x40000008	TMR0_CTL	TMR0 Control	0x0000000A
0x4000000C	TMR0_CLRINT	TMR0 Clear Interrupt	0x00000000
0x40000010	TMR0_CAPTURE	TMR0 Capture	0x00000000
0x40000014	TMR0_ALOAD	TMR0 16-bit Load Value, Asynchronous	0x00000000
0x40000018	TMR0_ACURCNT	TMR0 16-bit Timer Value, Asynchronous	0x00000000
0x4000001C	TMR0_STAT	TMR0 Status	0x00000000
0x40000020	TMR0_PWMCTL	TMR0 PWM Control Register	0x00000000
0x40000024	TMR0_PWMMATCH	TMR0 PWM Match Value	0x00000000

Table 24-16: ADuCM302x TMR1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40000400	TMR1_LOAD	TMR1 16-bit Load Value	0x00000000
0x40000404	TMR1_CURCNT	TMR1 16-bit Timer Value	0x00000000
0x40000408	TMR1_CTL	TMR1 Control	0x0000000A
0x4000040C	TMR1_CLRINT	TMR1 Clear Interrupt	0x00000000
0x40000410	TMR1_CAPTURE	TMR1 Capture	0x00000000
0x40000414	TMR1_ALOAD	TMR1 16-bit Load Value, Asynchronous	0x00000000
0x40000418	TMR1_ACURCNT	TMR1 16-bit Timer Value, Asynchronous	0x00000000
0x4000041C	TMR1_STAT	TMR1 Status	0x00000000
0x40000420	TMR1_PWMCTL	TMR1 PWM Control Register	0x00000000

Table 24-16: ADuCM302x TMR1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40000424	TMR1_PWMMATCH	TMR1 PWM Match Value	0x00000000

Table 24-17: ADuCM302x TMR2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40000800	TMR2_LOAD	TMR2 16-bit Load Value	0x00000000
0x40000804	TMR2_CURCNT	TMR2 16-bit Timer Value	0x00000000
0x40000808	TMR2_CTL	TMR2 Control	0x0000000A
0x4000080C	TMR2_CLRINT	TMR2 Clear Interrupt	0x00000000
0x40000810	TMR2_CAPTURE	TMR2 Capture	0x00000000
0x40000814	TMR2_ALOAD	TMR2 16-bit Load Value, Asynchronous	0x00000000
0x40000818	TMR2_ACURCNT	TMR2 16-bit Timer Value, Asynchronous	0x00000000
0x4000081C	TMR2_STAT	TMR2 Status	0x00000000
0x40000820	TMR2_PWMCTL	TMR2 PWM Control Register	0x00000000
0x40000824	TMR2_PWMMATCH	TMR2 PWM Match Value	0x00000000

Table 24-18: ADuCM302x I2C0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40003000	I2C0_MCTL	I2C0 Master Control	0x00000000
0x40003004	I2C0_MSTAT	I2C0 Master Status	0x00006000
0x40003008	I2C0_MRX	I2C0 Master Receive Data	0x00000000
0x4000300C	I2C0_MTX	I2C0 Master Transmit Data	0x00000000
0x40003010	I2C0_MRXCNT	I2C0 Master Receive Data Count	0x00000000
0x40003014	I2C0_MCRXCNT	I2C0 Master Current Receive Data Count	0x00000000
0x40003018	I2C0_ADDR1	I2C0 Master Address Byte 1	0x00000000
0x4000301C	I2C0_ADDR2	I2C0 Master Address Byte 2	0x00000000
0x40003020	I2C0_BYT	I2C0 Start Byte	0x00000000
0x40003024	I2C0_DIV	I2C0 Serial Clock Period Divisor	0x00001F1F
0x40003028	I2C0_SCTL	I2C0 Slave Control	0x00000000
0x4000302C	I2C0_SSTAT	I2C0 Slave I2C Status/Error/IRQ	0x00000001
0x40003030	I2C0_SRX	I2C0 Slave Receive	0x00000000

Table 24-18: ADuCM302x I2C0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40003034	I2C0_STX	I2C0 Slave Transmit	0x00000000
0x40003038	I2C0_ALT	I2C0 Hardware General Call ID	0x00000000
0x4000303C	I2C0_ID0	I2C0 First Slave Address Device ID	0x00000000
0x40003040	I2C0_ID1	I2C0 Second Slave Address Device ID	0x00000000
0x40003044	I2C0_ID2	I2C0 Third Slave Address Device ID	0x00000000
0x40003048	I2C0_ID3	I2C0 Fourth Slave Address Device ID	0x00000000
0x4000304C	I2C0_STAT	I2C0 Master and Slave FIFO Status	0x00000000
0x40003050	I2C0_SHCTL	I2C0 Shared Control	0x00000000
0x40003054	I2C0_TCTL	I2C0 Timing Control Register	0x00000005
0x40003058	I2C0_ASTRETCH_SCL	I2C0 Automatic Stretch SCL	0x00000000

Table 24-19: ADuCM302x NVIC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0xE000E004	NVIC0_INTNUM	NVIC0 Interrupt Control Type	0x00000000
0xE000E010	NVIC0_STKSTA	NVIC0 Systick Control and Status	0x00000000
0xE000E014	NVIC0_STKLD	NVIC0 Systick Reload Value	0x00000000
0xE000E018	NVIC0_STKVAL	NVIC0 Systick Current Value	0x00000000
0xE000E01C	NVIC0_STKCAL	NVIC0 Systick Calibration Value	0x00000000
0xE000E100	NVIC0_INTSETE0	NVIC0 IRQ0..31 Set_Enable	0x00000000
0xE000E104	NVIC0_INTSETE1	NVIC0 IRQ32..63 Set_Enable	0x00000000
0xE000E180	NVIC0_INTCLRE0	NVIC0 IRQ0..31 Clear_Enable	0x00000000
0xE000E184	NVIC0_INTCLRE1	NVIC0 IRQ32..63 Clear_Enable	0x00000000
0xE000E200	NVIC0_INTSETP0	NVIC0 IRQ0..31 Set_Pending	0x00000000
0xE000E204	NVIC0_INTSETP1	NVIC0 IRQ32..63 Set_Pending	0x00000000
0xE000E280	NVIC0_INTCLRP0	NVIC0 IRQ0..31 Clear_Pending	0x00000000
0xE000E284	NVIC0_INTCLRP1	NVIC0 IRQ32..63 Clear_Pending	0x00000000
0xE000E300	NVIC0_INTACT0	NVIC0 IRQ0..31 Active Bit	0x00000000
0xE000E304	NVIC0_INTACT1	NVIC0 IRQ32..63 Active Bit	0x00000000
0xE000E400	NVIC0_INTPRI0	NVIC0 IRQ0..3 Priority	0x00000000
0xE000E404	NVIC0_INTPRI1	NVIC0 IRQ4..7 Priority	0x00000000

Table 24-19: ADuCM302x NVIC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0xE000E408	NVIC0_INTPRI2	NVIC0 IRQ8..11 Priority	0x00000000
0xE000E40C	NVIC0_INTPRI3	NVIC0 IRQ12..15 Priority	0x00000000
0xE000E410	NVIC0_INTPRI4	NVIC0 IRQ16..19 Priority	0x00000000
0xE000E414	NVIC0_INTPRI5	NVIC0 IRQ20..23 Priority	0x00000000
0xE000E418	NVIC0_INTPRI6	NVIC0 IRQ24..27 Priority	0x00000000
0xE000E41C	NVIC0_INTPRI7	NVIC0 IRQ28..31 Priority	0x00000000
0xE000E420	NVIC0_INTPRI8	NVIC0 IRQ32..35 Priority	0x00000000
0xE000E424	NVIC0_INTPRI9	NVIC0 IRQ36..39 Priority	0x00000000
0xE000E428	NVIC0_INTPRI10	NVIC0 IRQ40..43 Priority	0x00000000
0xE000ED00	NVIC0_INTCPID	NVIC0 CPUID Base	0x00000000
0xE000ED04	NVIC0_INTSTA	NVIC0 Interrupt Control State	0x00000000
0xE000ED08	NVIC0_INTVEC	NVIC0 Vector Table Offset	0x00000000
0xE000ED0C	NVIC0_INTAIRC	NVIC0 Application Interrupt/Reset Control	0x00000000
0xE000ED10	NVIC0_INTCON0	NVIC0 System Control	0x00000000
0xE000ED14	NVIC0_INTCON1	NVIC0 Configuration Control	0x00000000
0xE000ED18	NVIC0_INTSHPRIO0	NVIC0 System Handlers 4-7 Priority	0x00000000
0xE000ED1C	NVIC0_INTSHPRIO1	NVIC0 System Handlers 8-11 Priority	0x00000000
0xE000ED20	NVIC0_INTSHPRIO3	NVIC0 System Handlers 12-15 Priority	0x00000000
0xE000ED24	NVIC0_INTSHCSR	NVIC0 System Handler Control and State	0x00000000
0xE000ED28	NVIC0_INTCFSR	NVIC0 Configurable Fault Status	0x00000000
0xE000ED2C	NVIC0_INTHFSR	NVIC0 Hard Fault Status	0x00000000
0xE000ED30	NVIC0_INTDFSR	NVIC0 Debug Fault Status	0x00000000
0xE000ED34	NVIC0_INTMMAR	NVIC0 Mem Manage Address	0x00000000
0xE000ED38	NVIC0_INTBFAR	NVIC0 Bus Fault Address	0x00000000
0xE000ED3C	NVIC0_INTAFSR	NVIC0 Auxiliary Fault Status	0x00000000
0xE000ED40	NVIC0_INTPFR0	NVIC0 Processor Feature Register 0	0x00000000
0xE000ED44	NVIC0_INTPFR1	NVIC0 Processor Feature Register 1	0x00000000
0xE000ED48	NVIC0_INTDFR0	NVIC0 Debug Feature Register 0	0x00000000
0xE000ED4C	NVIC0_INTAFR0	NVIC0 Auxiliary Feature Register 0	0x00000000
0xE000ED50	NVIC0_INTMMFR0	NVIC0 Memory Model Feature Register 0	0x00000000
0xE000ED54	NVIC0_INTMMFR1	NVIC0 Memory Model Feature Register 1	0x00000000

Table 24-19: ADuCM302x NVIC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0xE000ED58	NVIC0_INTMMFR2	NVIC0 Memory Model Feature Register 2	0x00000000
0xE000ED5C	NVIC0_INTMMFR3	NVIC0 Memory Model Feature Register 3	0x00000000
0xE000ED60	NVIC0_INTISAR0	NVIC0 ISA Feature Register 0	0x00000000
0xE000ED64	NVIC0_INTISAR1	NVIC0 ISA Feature Register 1	0x00000000
0xE000ED68	NVIC0_INTISAR2	NVIC0 ISA Feature Register 2	0x00000000
0xE000ED6C	NVIC0_INTISAR3	NVIC0 ISA Feature Register 3	0x00000000
0xE000ED70	NVIC0_INTISAR4	NVIC0 ISA Feature Register 4	0x00000000
0xE000EF00	NVIC0_INTTRGI	NVIC0 Software Trigger Interrupt Register	0x00000000
0xE000EFD0	NVIC0_INTPID4	NVIC0 Peripheral Identification Register 4	0x00000000
0xE000EFD4	NVIC0_INTPID5	NVIC0 Peripheral Identification Register 5	0x00000000
0xE000EFD8	NVIC0_INTPID6	NVIC0 Peripheral Identification Register 6	0x00000000
0xE000EFDC	NVIC0_INTPID7	NVIC0 Peripheral Identification Register 7	0x00000000
0xE000EFE0	NVIC0_INTPID0	NVIC0 Peripheral Identification Bits7:0	0x00000000
0xE000EFE4	NVIC0_INTPID1	NVIC0 Peripheral Identification Bits15:8	0x00000000
0xE000EFE8	NVIC0_INTPID2	NVIC0 Peripheral Identification Bits16:23	0x00000000
0xE000EFEC	NVIC0_INTPID3	NVIC0 Peripheral Identification Bits24:31	0x00000000
0xE000EFF0	NVIC0_INTCID0	NVIC0 Component Identification Bits7:0	0x00000000
0xE000EFF4	NVIC0_INTCID1	NVIC0 Component Identification Bits15:8	0x00000000
0xE000EFF8	NVIC0_INTCID2	NVIC0 Component Identification Bits16:23	0x00000000
0xE000EFFC	NVIC0_INTCID3	NVIC0 Component Identification Bits24:31	0x00000000

Table 24-20: ADuCM302x PMG0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4004C000	PMG0_IEN	PMG0 Power Supply Monitor Interrupt Enable	0x00000000
0x4004C004	PMG0_PSM_STAT	PMG0 Power Supply Monitor Status	0x00000000
0x4004C008	PMG0_PWRMOD	PMG0 Power Mode Register	0x00000000
0x4004C00C	PMG0_PWRKEY	PMG0 Key Protection for PMG_PWRMOD and PMG_SRAMRET	0x00000000
0x4004C010	PMG0_SHDN_STAT	PMG0 Shutdown Status Register	0x00000000
0x4004C014	PMG0_SRAMRET	PMG0 Control for Retention SRAM in Hibernate Mode	0x00000000
0x4004C040	PMG0_RST_STAT	PMG0 Reset Status	0x00000000

Table 24-20: ADuCM302x PMG0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4004C044	PMG0_CTL1	PMG0 HP Buck Control	0x00A00000

Table 24-21: ADuCM302x PMG0_TST MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4004C260	PMG0_TST_SRAM_CTL	PMG0_TST Control for SRAM Parity and Instruction SRAM	0x80000000
0x4004C264	PMG0_TST_SRAM_INITSTAT	PMG0_TST Initialization Status Register	0x00000000
0x4004C268	PMG0_TST_CLR_LATCH_GPIOS	PMG0_TST Clear GPIO After Shutdown Mode	0x00000000
0x4004C26C	PMG0_TST_SCRPAD_IMG	PMG0_TST Scratch Pad Image	0x00000000
0x4004C270	PMG0_TST_SCRPAD_3V_RD	PMG0_TST Scratch Pad Saved in Battery Domain	0x00000000

Table 24-22: ADuCM302x PTI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4004CD00	PTI0_RST_ISR_STARTADDR	PTI0 Reset ISR Start Address	0x00000000
0x4004CD04	PTI0_RST_STACK_PTR	PTI0 Reset Stack Pointer	0x00000000
0x4004CD08	PTI0_CTL	PTI0 Parallel Test Interface Control Register	0x00000000

Table 24-23: ADuCM302x RNG0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40040400	RNG0_CTL	RNG0 RNG Control Register	0x00000000
0x40040404	RNG0_LEN	RNG0 RNG Sample Length Register	0x00003400
0x40040408	RNG0_STAT	RNG0 RNG Status Register	0x00000000
0x4004040C	RNG0_DATA	RNG0 RNG Data Register	0x00000000
0x40040410	RNG0_OSCCNT	RNG0 Oscillator Count	0x00000000
0x40040414	RNG0_OSCDIFF[n]	RNG0 Oscillator Difference	0x00000000
0x40040415	RNG0_OSCDIFF[n]	RNG0 Oscillator Difference	0x00000000
0x40040416	RNG0_OSCDIFF[n]	RNG0 Oscillator Difference	0x00000000

Table 24-23: ADuCM302x RNG0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40040417	RNG0_OSCDIFF[n]	RNG0 Oscillator Difference	0x00000000

Table 24-24: ADuCM302x RTC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40001000	RTC0_CR0	RTC0 RTC Control 0	0x000003C4
0x40001004	RTC0_SR0	RTC0 RTC Status 0	0x00003F80
0x40001008	RTC0_SR1	RTC0 RTC Status 1	0x00000078
0x4000100C	RTC0_CNT0	RTC0 RTC Count 0	0x00000000
0x40001010	RTC0_CNT1	RTC0 RTC Count 1	0x00000000
0x40001014	RTC0_ALM0	RTC0 RTC Alarm 0	0x0000FFFF
0x40001018	RTC0_ALM1	RTC0 RTC Alarm 1	0x0000FFFF
0x4000101C	RTC0_TRM	RTC0 RTC Trim	0x00000398
0x40001020	RTC0_GWY	RTC0 RTC Gateway	0x00000000
0x40001028	RTC0_CR1	RTC0 RTC Control 1	0x000001E0
0x4000102C	RTC0_SR2	RTC0 RTC Status 2	0x0000C000
0x40001030	RTC0_SNAP0	RTC0 RTC Snapshot 0	0x00000000
0x40001034	RTC0_SNAP1	RTC0 RTC Snapshot 1	0x00000000
0x40001038	RTC0_SNAP2	RTC0 RTC Snapshot 2	0x00000000
0x4000103C	RTC0_MOD	RTC0 RTC Modulo	0x00000040
0x40001040	RTC0_CNT2	RTC0 RTC Count 2	0x00000000
0x40001044	RTC0_ALM2	RTC0 RTC Alarm 2	0x00000000
0x40001048	RTC0_SR3	RTC0 RTC Status 3	0x00000000
0x4000104C	RTC0_CR2IC	RTC0 RTC Control 2 for Configuring Input Capture Channels	0x000083A0
0x40001050	RTC0_CR3SS	RTC0 RTC Control 3 for Configuring SensorStrobe Channel	0x00000000
0x40001054	RTC0_CR4SS	RTC0 RTC Control 4 for Configuring SensorStrobe Channel	0x00000000
0x40001058	RTC0_SSMSK	RTC0 RTC Mask for SensorStrobe Channel	0x00000000
0x4000105C	RTC0_SS1ARL	RTC0 RTC Auto-Reload for SensorStrobe Channel 1	0x00000000
0x40001064	RTC0_IC2	RTC0 RTC Input Capture Channel 2	0x00000000

Table 24-24: ADuCM302x RTC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40001068	RTC0_IC3	RTC0 RTC Input Capture Channel 3	0x00000000
0x4000106C	RTC0_IC4	RTC0 RTC Input Capture Channel 4	0x00000000
0x40001070	RTC0_SS1	RTC0 RTC SensorStrobe Channel 1	0x00008000
0x40001080	RTC0_SR4	RTC0 RTC Status 4	0x000077FF
0x40001084	RTC0_SR5	RTC0 RTC Status 5	0x00000000
0x40001088	RTC0_SR6	RTC0 RTC Status 6	0x00007900
0x4000108C	RTC0_SS1TGT	RTC0 RTC SensorStrobe Channel 1 Target	0x00008000
0x40001090	RTC0_FRZCNT	RTC0 RTC Freeze Count	0x00000000

Table 24-25: ADuCM302x RTC1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40001400	RTC1_CR0	RTC1 RTC Control 0	0x000003C4
0x40001404	RTC1_SR0	RTC1 RTC Status 0	0x00003F80
0x40001408	RTC1_SR1	RTC1 RTC Status 1	0x00000078
0x4000140C	RTC1_CNT0	RTC1 RTC Count 0	0x00000000
0x40001410	RTC1_CNT1	RTC1 RTC Count 1	0x00000000
0x40001414	RTC1_ALM0	RTC1 RTC Alarm 0	0x0000FFFF
0x40001418	RTC1_ALM1	RTC1 RTC Alarm 1	0x0000FFFF
0x4000141C	RTC1_TRM	RTC1 RTC Trim	0x00000398
0x40001420	RTC1_GWY	RTC1 RTC Gateway	0x00000000
0x40001428	RTC1_CR1	RTC1 RTC Control 1	0x000001E0
0x4000142C	RTC1_SR2	RTC1 RTC Status 2	0x0000C000
0x40001430	RTC1_SNAP0	RTC1 RTC Snapshot 0	0x00000000
0x40001434	RTC1_SNAP1	RTC1 RTC Snapshot 1	0x00000000
0x40001438	RTC1_SNAP2	RTC1 RTC Snapshot 2	0x00000000
0x4000143C	RTC1_MOD	RTC1 RTC Modulo	0x00000040
0x40001440	RTC1_CNT2	RTC1 RTC Count 2	0x00000000
0x40001444	RTC1_ALM2	RTC1 RTC Alarm 2	0x00000000
0x40001448	RTC1_SR3	RTC1 RTC Status 3	0x00000000
0x4000144C	RTC1_CR2IC	RTC1 RTC Control 2 for Configuring Input Capture Channels	0x000083A0

Table 24-25: ADuCM302x RTC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40001450	RTC1_CR3SS	RTC1 RTC Control 3 for Configuring SensorStrobe Channel	0x00000000
0x40001454	RTC1_CR4SS	RTC1 RTC Control 4 for Configuring SensorStrobe Channel	0x00000000
0x40001458	RTC1_SSMSK	RTC1 RTC Mask for SensorStrobe Channel	0x00000000
0x4000145C	RTC1_SS1ARL	RTC1 RTC Auto-Reload for SensorStrobe Channel 1	0x00000000
0x40001464	RTC1_IC2	RTC1 RTC Input Capture Channel 2	0x00000000
0x40001468	RTC1_IC3	RTC1 RTC Input Capture Channel 3	0x00000000
0x4000146C	RTC1_IC4	RTC1 RTC Input Capture Channel 4	0x00000000
0x40001470	RTC1_SS1	RTC1 RTC SensorStrobe Channel 1	0x00008000
0x40001480	RTC1_SR4	RTC1 RTC Status 4	0x000077FF
0x40001484	RTC1_SR5	RTC1 RTC Status 5	0x00000000
0x40001488	RTC1_SR6	RTC1 RTC Status 6	0x00007900
0x4000148C	RTC1_SS1TGT	RTC1 RTC SensorStrobe Channel 1 Target	0x00008000
0x40001490	RTC1_FRZCNT	RTC1 RTC Freeze Count	0x00000000

Table 24-26: ADuCM302x SPI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40004000	SPI0_STAT	SPI0 Status	0x00000800
0x40004004	SPI0_RX	SPI0 Receive	0x00000000
0x40004008	SPI0_TX	SPI0 Transmit	0x00000000
0x4000400C	SPI0_DIV	SPI0 SPI Baud Rate Selection	0x00000000
0x40004010	SPI0_CTL	SPI0 SPI Configuration	0x00000000
0x40004014	SPI0_IEN	SPI0 SPI Interrupts Enable	0x00000000
0x40004018	SPI0_CNT	SPI0 Transfer Byte Count	0x00000000
0x4000401C	SPI0_DMA	SPI0 SPI DMA Enable	0x00000000
0x40004020	SPI0_FIFO_STAT	SPI0 FIFO Status	0x00000000
0x40004024	SPI0_RD_CTL	SPI0 Read Control	0x00000000
0x40004028	SPI0_FLOW_CTL	SPI0 Flow Control	0x00000000
0x4000402C	SPI0_WAIT_TMR	SPI0 Wait Timer for Flow Control	0x00000000
0x40004030	SPI0_CS_CTL	SPI0 Chip Select Control for Multi-slave Connections	0x00000001

Table 24-26: ADuCM302x SPI0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40004034	SPI0_CS_OVERRIDE	SPI0 Chip Select Override	0x00000000

Table 24-27: ADuCM302x SPI1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40004400	SPI1_STAT	SPI1 Status	0x00000800
0x40004404	SPI1_RX	SPI1 Receive	0x00000000
0x40004408	SPI1_TX	SPI1 Transmit	0x00000000
0x4000440C	SPI1_DIV	SPI1 SPI Baud Rate Selection	0x00000000
0x40004410	SPI1_CTL	SPI1 SPI Configuration	0x00000000
0x40004414	SPI1_IEN	SPI1 SPI Interrupts Enable	0x00000000
0x40004418	SPI1_CNT	SPI1 Transfer Byte Count	0x00000000
0x4000441C	SPI1_DMA	SPI1 SPI DMA Enable	0x00000000
0x40004420	SPI1_FIFO_STAT	SPI1 FIFO Status	0x00000000
0x40004424	SPI1_RD_CTL	SPI1 Read Control	0x00000000
0x40004428	SPI1_FLOW_CTL	SPI1 Flow Control	0x00000000
0x4000442C	SPI1_WAIT_TMR	SPI1 Wait Timer for Flow Control	0x00000000
0x40004430	SPI1_CS_CTL	SPI1 Chip Select Control for Multi-slave Connections	0x00000001
0x40004434	SPI1_CS_OVERRIDE	SPI1 Chip Select Override	0x00000000

Table 24-28: ADuCM302x SPI2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40024000	SPI2_STAT	SPI2 Status	0x00000800
0x40024004	SPI2_RX	SPI2 Receive	0x00000000
0x40024008	SPI2_TX	SPI2 Transmit	0x00000000
0x4002400C	SPI2_DIV	SPI2 SPI Baud Rate Selection	0x00000000
0x40024010	SPI2_CTL	SPI2 SPI Configuration	0x00000000
0x40024014	SPI2_IEN	SPI2 SPI Interrupts Enable	0x00000000
0x40024018	SPI2_CNT	SPI2 Transfer Byte Count	0x00000000
0x4002401C	SPI2_DMA	SPI2 SPI DMA Enable	0x00000000
0x40024020	SPI2_FIFO_STAT	SPI2 FIFO Status	0x00000000

Table 24-28: ADuCM302x SPI2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40024024	SPI2_RD_CTL	SPI2 Read Control	0x00000000
0x40024028	SPI2_FLOW_CTL	SPI2 Flow Control	0x00000000
0x4002402C	SPI2_WAIT_TMR	SPI2 Wait Timer for Flow Control	0x00000000
0x40024030	SPI2_CS_CTL	SPI2 Chip Select Control for Multi-slave Connections	0x00000001
0x40024034	SPI2_CS_OVERRIDE	SPI2 Chip Select Override	0x00000000

Table 24-29: ADuCM302x SPORT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40038000	SPORT0_CTL_A	SPORT0 Half SPORT 'A' Control	0x00000000
0x40038004	SPORT0_DIV_A	SPORT0 Half SPORT 'A' Divisor	0x00000000
0x40038008	SPORT0_IEN_A	SPORT0 Half SPORT A's Interrupt Enable	0x00000000
0x4003800C	SPORT0_STAT_A	SPORT0 Half SPORT A's Status	0x00000000
0x40038010	SPORT0_NUMTRAN_A	SPORT0 Half SPORT A Number of Transfers	0x00000000
0x40038014	SPORT0_CNVT_A	SPORT0 Half SPORT 'A' CNV Width	0x00000000
0x40038020	SPORT0_TX_A	SPORT0 Half SPORT 'A' Tx Buffer	0x00000000
0x40038028	SPORT0_RX_A	SPORT0 Half SPORT 'A' Rx Buffer	0x00000000
0x40038040	SPORT0_CTL_B	SPORT0 Half SPORT 'B' Control	0x00000000
0x40038044	SPORT0_DIV_B	SPORT0 Half SPORT 'B' Divisor	0x00000000
0x40038048	SPORT0_IEN_B	SPORT0 Half SPORT B's Interrupt Enable	0x00000000
0x4003804C	SPORT0_STAT_B	SPORT0 Half SPORT B's Status	0x00000000
0x40038050	SPORT0_NUMTRAN_B	SPORT0 Half SPORT B Number of Transfers	0x00000000
0x40038054	SPORT0_CNVT_B	SPORT0 Half SPORT 'B' CNV Width	0x00000000
0x40038060	SPORT0_TX_B	SPORT0 Half SPORT 'B' Tx Buffer	0x00000000
0x40038068	SPORT0_RX_B	SPORT0 Half SPORT 'B' Rx Buffer	0x00000000

Table 24-30: ADuCM302x SYS MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40002020	SYS_ADIID	SYS ADI Identification	0x00004144
0x40002024	SYS_CHIPID	SYS Chip Identifier	0x00000282
0x40002040	SYS_SWDEN	SYS Serial Wire Debug Enable	0x00007072

Table 24-31: ADuCM302x UART0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40005000	UART0_TX	UART0 Transmit Holding Register	0x00000000
0x40005000	UART0_RX	UART0 Receive Buffer Register	0x00000000
0x40005004	UART0_IEN	UART0 Interrupt Enable	0x00000000
0x40005008	UART0_IIR	UART0 Interrupt ID	0x00000001
0x4000500C	UART0_LCR	UART0 Line Control	0x00000000
0x40005010	UART0_MCR	UART0 Modem Control	0x00000000
0x40005014	UART0_LSR	UART0 Line Status	0x00000060
0x40005018	UART0_MSR	UART0 Modem Status	0x00000000
0x4000501C	UART0_SCR	UART0 Scratch Buffer	0x00000000
0x40005020	UART0_FCR	UART0 FIFO Control	0x00000000
0x40005024	UART0_FBR	UART0 Fractional Baud Rate	0x00000000
0x40005028	UART0_DIV	UART0 Baud Rate Divider	0x00000000
0x4000502C	UART0_LCR2	UART0 Second Line Control	0x00000002
0x40005030	UART0_CTL	UART0 UART Control Register	0x00000100
0x40005034	UART0_RFC	UART0 RX FIFO Byte Count	0x00000000
0x40005038	UART0_TFC	UART0 TX FIFO Byte Count	0x00000000
0x4000503C	UART0_RSC	UART0 RS485 Half-duplex Control	0x00000000
0x40005040	UART0_ACR	UART0 Auto Baud Control	0x00000000
0x40005044	UART0_ASRL	UART0 Auto Baud Status (Low)	0x00000000
0x40005048	UART0_ASRH	UART0 Auto Baud Status (High)	0x00000000

Table 24-32: ADuCM302x WDT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40002C00	WDT0_LOAD	WDT0 Load Value	0x00001000
0x40002C04	WDT0_CCNT	WDT0 Current Count Value	0x00001000
0x40002C08	WDT0_CTL	WDT0 Control	0x000000E9
0x40002C0C	WDT0_RESTART	WDT0 Clear Interrupt	0x00000000
0x40002C18	WDT0_STAT	WDT0 Status	0x00000000

Index