TMS320C5535/34/33/32 Ultra-Low Power DSP

Technical Reference Manual

Texas Instruments

Literature Number: SPRUH87C
August 2011–Revised March 2012

Contents

List of Figures

List of Tables

Read This First

About This Manual

This technical reference manual (TRM) details the integration, environment, functional description, and programming models for each peripheral and subsystem in the device.

Notational Conventions

This document uses the following conventions.

- Hexadecimal numbers are shown with the suffix h. For example, the following number is 40 hexadecimal (decimal 64): 40h.
- Registers in this document are shown in figures and described in tables.
  - Each register figure shows a rectangle divided into fields that represent the fields of the register. Each field is labeled with its bit name, its beginning and ending bit numbers above, and its read/write properties below. A legend explains the notation used for the properties.
  - Reserved bits in a register figure designate a bit that is used for future device expansion.

Related Documentation From Texas Instruments

For a complete listing of related documentation and development-support tools for the ultra-low power DSPs, visit the Texas Instruments website: www.ti.com.

TMS320C5000 is a trademark of Texas Instruments.
SD is a trademark of SanDisk.

Chapter 1

SPRUH87C – August 2011 – Revised March 2012

System Control

This chapter provides an overview of the system control for the device.

1.1 Introduction

The TMS320C5535/34/33/32 digital-signal processor (DSP) contains a high-performance, low-power DSP to efficiently handle tasks required by portable audio, wireless audio devices, industrial controls, software defined radio, fingerprint biometrics, and medical applications. The DSP consists of the following primary components:

- A C55x CPU and associated memory
- FFT hardware accelerator (TMS320C5535 only)
- Four DMA controllers
- Power management module
- A set of I/O peripherals, including I2S, I2C, SPI, UART, Timers for all devices, USB 2.0 for TMS320C5535/34/33, and 10-bit SAR ADC and LCD controller for TMS320C5535

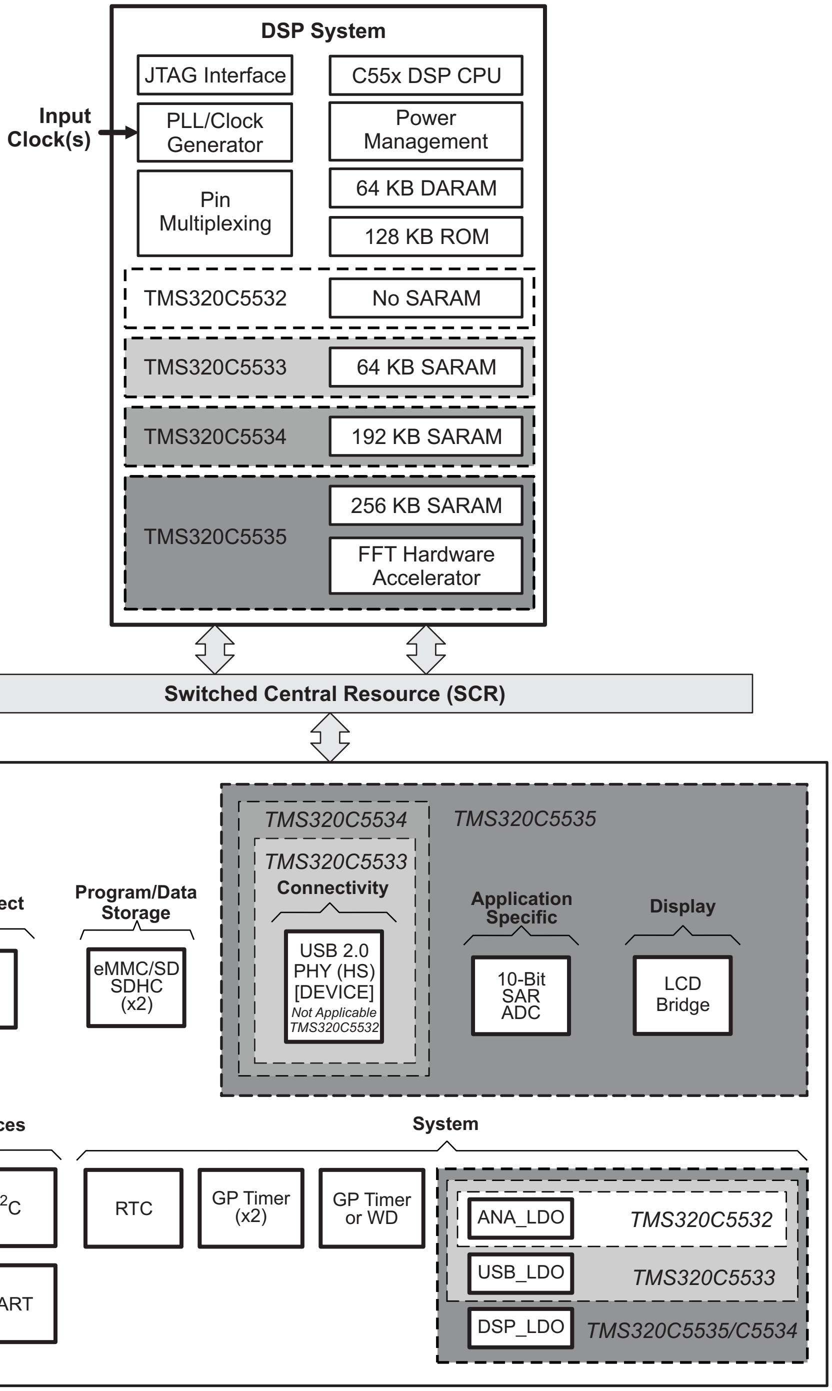
For more information on these components, see the following documents:

- *TMS320C55x 3.0 CPU Reference Guide* (SWPU073).
- *TMS320C55x v3.x CPU Algebraic Instruction Set Reference Guide* (SWPU068E)
- *TMS320C55x v3.x CPU Mnemonic Instruction Set Reference Guide* (SWPU067E)
- *TMS320C55x DSP Peripherals Overview Reference Guide* (SPRU317)

1.1.1 Block Diagram

The DSP block diagram is shown in Figure 1-1.

Figure 1-1. Functional Block Diagram



1.1.2 Device Differences

The differences between the devices are listed in the following table.

Table 1-1. Device Differences

Device	Digital Core Supply Voltage (CV_{DD})		On-chip DARAM	On-chip SARAM	USB	LCD Interface	Tightly-Coupled FFT	SAR ADC	LDO
	1.05 V	1.3 V							
	Maximum CPU Speed								
TMS320C5535A05	50 MHz	-	64 KB	256 KB	x[1]	x	x	x	ANA, DSP, and USB
TMS320C5535A10	50 MHz	100 MHz							
TMS320C5534A05	50 MHz	-	64 KB	192 KB	x	-[2]	-	-	ANA, DSP, and USB
TMS320C5534A10	50 MHz	100 MHz							
TMS320C5533A05	50 MHz	-	64 KB	64 KB	x	-	-	-	ANA and USB
TMS320C5533A10	50 MHz	100 MHz							
TMS320C5532A05	50 MHz	-	64 KB	0 KB	-	-	-	-	ANA only
TMS320C5532A10	50 MHz	100 MHz							

[1] x — Supported

[2] - — Not supported

1.1.3 CPU Core

The C55x CPU is responsible for performing the digital signal processing tasks required by the application. In addition, the CPU acts as the overall system controller, responsible for handling many system functions such as system-level initialization, configuration, user interface, user command execution, connectivity functions, and overall system control.

Tightly coupled to the CPU are the following components:

- DSP internal memories
  - Dual-access RAM (DARAM)
  - Single-access RAM (SARAM) (Not available for TMS320C5532)
  - Read-only memory (ROM)
- FFT hardware accelerator (TMS320C5535 only)
- Ports and buses

The CPU also manages/controls all peripherals on the device. Refer to the device-specific data manual for the full list of peripherals.

shows the functional block diagram of the DSP and how it connects to the rest of the device. The DSP architecture uses the switched central resource (SCR) to transfer data within the system.

1.1.4 FFT Hardware Accelerator (TMS320C5535 Only)

The C55x CPU includes a tightly-coupled FFT hardware accelerator that communicates with the C55x CPU through the use coprocessor instructions. For ease of use, the ROM has a set of C-callable routines that use these coprocessor instructions to perform 8, 16, 32, 64, 128, or 256-point FFTs. The main features of the FFT hardware accelerator are:

- Support for 8 to 1024-point (in powers of 2) real and complex-valued FFTs and IFFTs.
- An internal twiddle factor generator for optimal use of memory bandwidth and more efficient programming.
- Basic and software-driven auto-scaling feature provides good precision vs cycle count trade-off.
- Single-stage and double-stage modes enabling computation of one or two stages in one pass, thus handling odd power of two FFT widths.

1.1.4.1 Using FFT Accelerator ROM Routines

The DSP includes C-callable routines in ROM to execute FFT and IFFT using the tightly coupled FFT accelerator. The routines reside in the following address:

Table 1-2. FFT Accelerator ROM Routines

Address	Name	Description	Calling Convention
00fefe9c	hwafft br	Vector bit-reversal	void hwafft_br(Int32 *data, Int32 *data_br, Uint16 data_len);
00fefeb0	hwafft 8pts	8-pt FFT/IFFT	Uint16 hwafft_8pts(Int32 *data,Int32 *scratch, Uint16 fft_flag, Uint16 scale_flag);
00feff9f	hwafft 16pts	16-pt FFT/IFFT	Uint16 hwafft_16pts(Int32 *data,Int32 *scratch, Uint16 fft_flag, Uint16 scale_flag);
00ff00f5	hwafft 32pts	32-pt FFT/IFFT	Uint16 hwafft_32pts(Int32 *data,Int32 *scratch, Uint16 fft_flag, Uint16 scale_flag);
00ff03fe	hwafft 64pts	64-pt FFT/iFFT	Uint16 hwafft_64pts(Int32 *data,Int32 *scratch, Uint16 fft_flag, Uint16 scale_flag);
00ff0593	hwafft 128pts	128-pt FFT/IFFT	Uint16 hwafft_128pts(Int32 *data,Int32 *scratch, Uint16 fft_flag, Uint16 scale_flag);
00ff07a4	hwafft 256pts	256-pt FFT/IFFT	Uint16 hwafft_256pts(Int32 *data,Int32 *scratch, Uint16 fft_flag, Uint16 scale_flag);
00ff09a2	hwafft 512pts	512-pt FFT/iFFT	Uint16 hwafft_512pts(Int32 *data,Int32 *scratch, Uint16 fft_flag, Uint16 scale_flag);
00ff0c1c	hwafft 1024pts	1024-pt FFT/IFFT	Uint16 hwafft_1024pts(Int32 *data,Int32 *scratch, Uint16 fft_flag, Uint16 scale_flag);

For the FFT routines, output data is dependent on the return value (T0). If return = 0 output data is in-place, meaning the result will overwrite the input buffer. If return =1, output data is placed in the scratch buffer. The 32-bit input and output data consist of 16-bit real and 16-bit imaginary data. If only real data is used, the imaginary part can be zeroed. The Scale flag determines if the butterfly output is divided by 2 to prevent overflow at the expense of resolution. For more information on using these routines, see Chapter 2.

1.1.5 Power Management

Integrated into the DSP are the following power management features:

- One low dropout LDO for analog portions of the device, DSP PLL (V_{DDA_PLL}), and power management circuits (V_{DDA_ANA}): ANA_LDO (all devices)
- One LDO for DSP core (CV_{DD}): DSP_LDO (TMS320C5535/34 only)
- One LDO for USB core and PHY (USB_V_{DDA1P3}): USB_LDO (not supported on TMS320C5532)
- Idle controller with several clock domains:
  - CPU domain
  - Clock generator domain
  - Peripheral domain
  - USB domain
  - Real-time clock (RTC) domain
- Independent voltage and power domains
- LDOI (LDOs and Bandgap Power Supply)
- Analog POR, and PLL (V_{DDA_ANA} and V_{DDA_PLL})
- Real-time clock core (CV_{DDRTC})
- Digital core (CV_{DD})
- USB core (USB_ V_{DD1P3} and USB_V_{DDA1P3})
- USB PHY and USB PLL (USB_V_{DDOSC}, USB_V_{DDA3P3}, and USB_V_{DDPLL})
- RTC I/O (DV_{DDRTC})

- Rest of the I/O (DV_{DDIO})

1.1.6 Peripherals

The DSP includes the following peripherals:

- Four direct memory access (DMA) controllers, each with four independent channels.
- Two serial busses each configurable to support one Embedded Multimedia Card (eMMC) / Secure Digital (SD/SDHC/SDIO) controller, one inter-IC sound bus (I2S) interface with GPIO, or a full GPIO interface.
- One inter-integrated circuit (I2C) multi-master and slave interface with 7-bit and 10-bit addressing modes.
- Three 32-bit timers with 16-bit prescaler; one timer supports watchdog functionality.
- A USB 2.0 slave.
- One real-time clock (RTC) with associated low power mode.

1.2 System Memory

The DSP supports a unified memory map (program code sections and data sections can be mixed and interleaved within the entire memory space) composed of on-chip memory. The on-chip memory consists of 128KB of ROM. It also consists of 320KB of RAM for TMS320C5535, 256KB for C5534, 128KB for C5533, and 64KB for C5532.

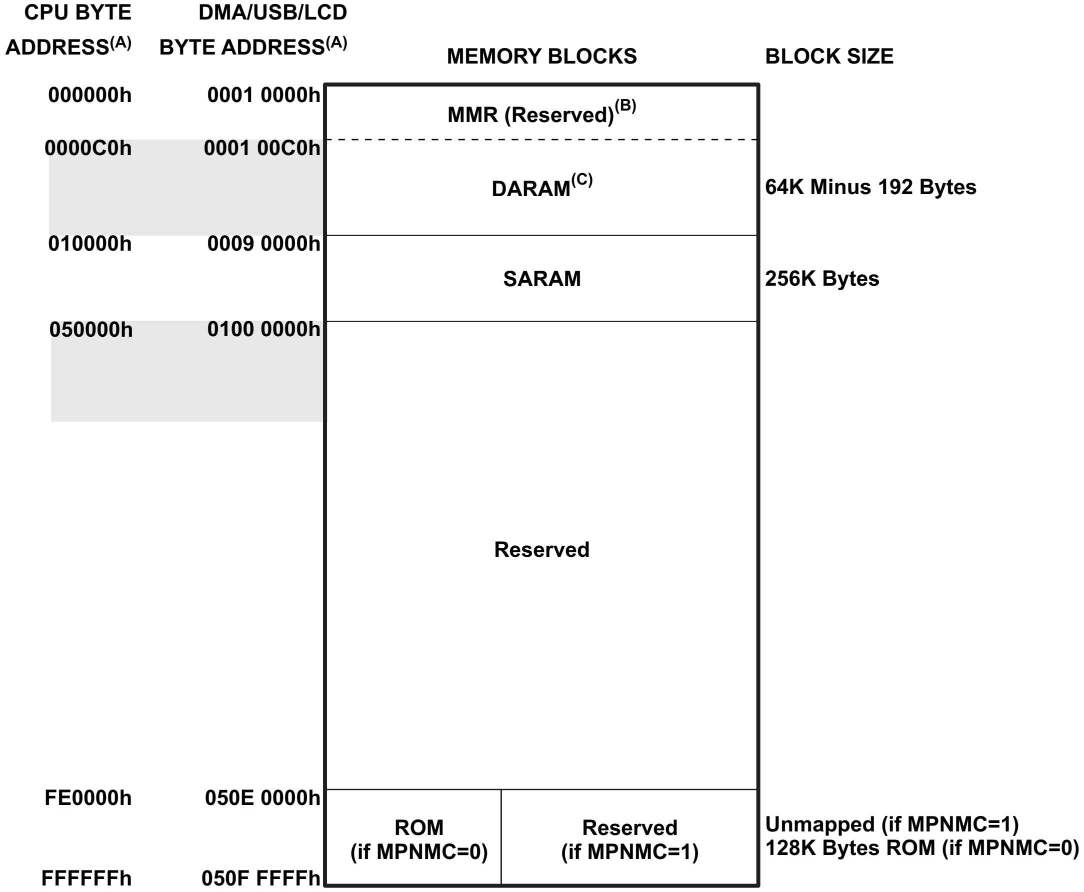
Separate from the program and data space, the DSP also includes a 64K-byte I/O space for peripheral registers.

1.2.1 Program/Data Memory Map

The on-chip, dual-access RAM allows two accesses to a given block during the same cycle. The device has 8 blocks of 8K-bytes of dual-access RAM. The on-chip, single-access RAM allows one access to a given block per cycle. TMS320C5535/34/33 features 32 blocks, 24 blocks, and 8 blocks of 8K-bytes of single-access RAM, respectively. Attempts to perform two accesses in a cycle to single-access memory will cause one access to stall until the next cycle. An access is defined as either a read or write operation. For the most efficient use of DSP processing power (MIPS), it is important to pay attention to the memory blocks that are being simultaneously accessed by the code and data operations.

The DSP memory is accessible by different master modules within the DSP, including the device CPU, the four DMA controllers, and the USB. The DSP memory map as seen by these modules is illustrated in Figure 1-2 through Figure 1-5.

Figure 1-2. TMS320C5535 Memory Map



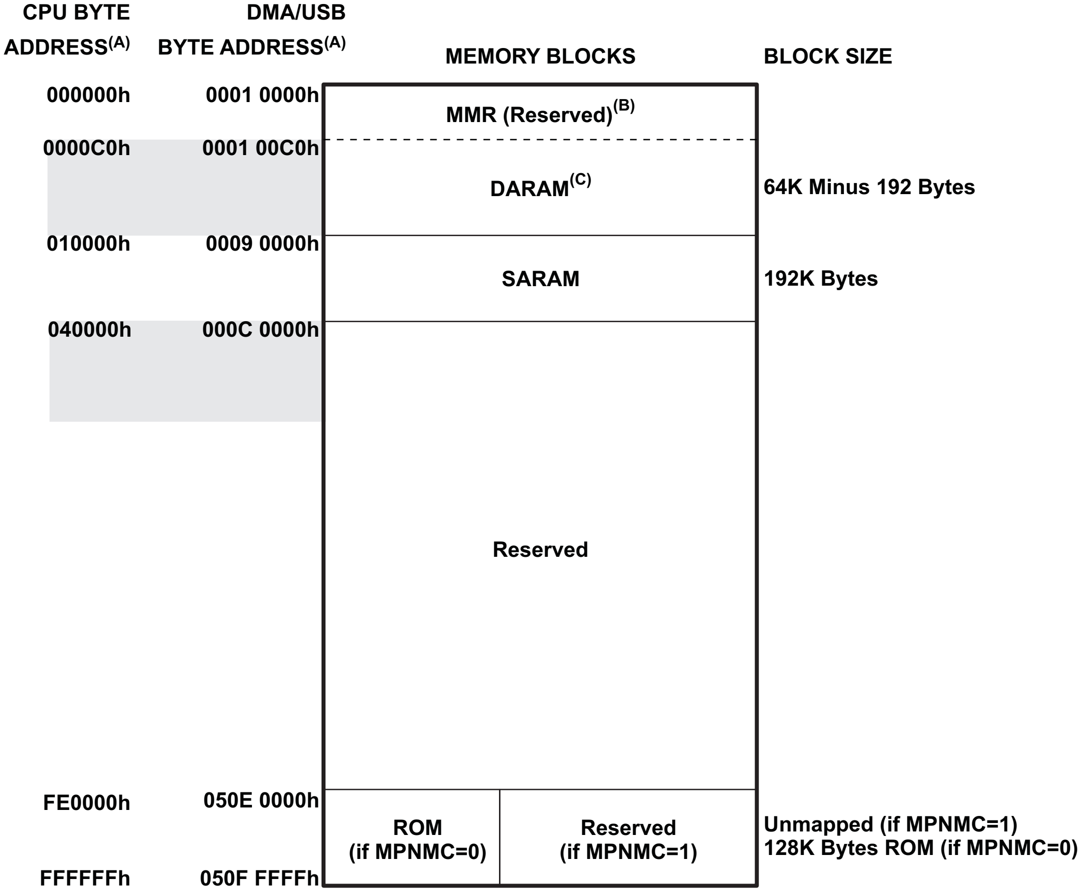
CPU BYTE ADDRESS[A]	DMA/USB/LCD BYTE ADDRESS[A]	MEMORY BLOCKS		BLOCK SIZE
000000h	0001 0000h	MMR (Reserved)[B]		
0000C0h	0001 00C0h	DARAM[C]		64K Minus 192 Bytes
010000h	0009 0000h	SARAM		256K Bytes
050000h	0100 0000h	Reserved		
FE0000h	050E 0000h	ROM (if MPNMC=0)	Reserved (if MPNMC=1)	Unmapped (if MPNMC=1) 128K Bytes ROM (if MPNMC=0)
FFFFFFh	050F FFFFh			

A Address shown represents the first byte address in each block.

B The first 192 bytes are reserved for memory-mapped registers (MMRs).

C The USB and LCD controllers do not have access to DARAM.

Figure 1-3. TMS320C5534 Memory Map

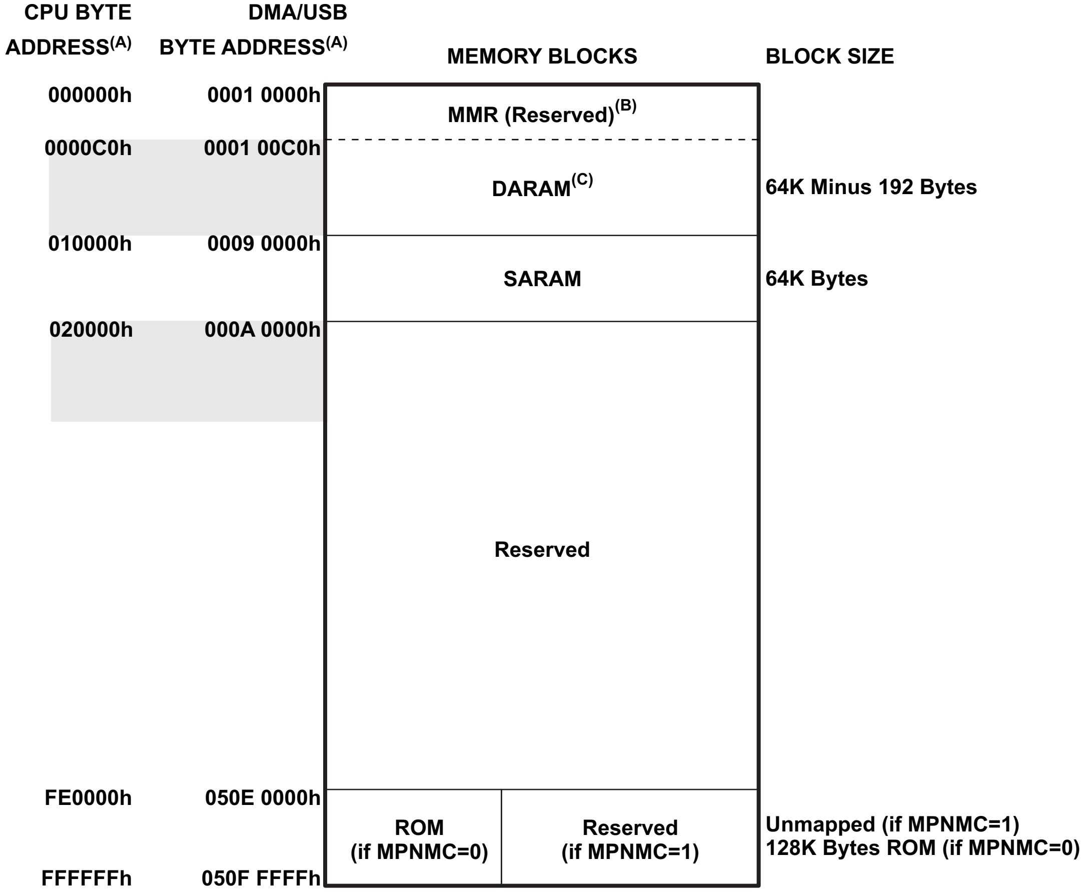


A Address shown represents the first byte address in each block.

B The first 192 bytes are reserved for memory-mapped registers (MMRs).

C The USB and LCD controllers do not have access to DARAM.

Figure 1-4. TMS320C5533 Memory Map

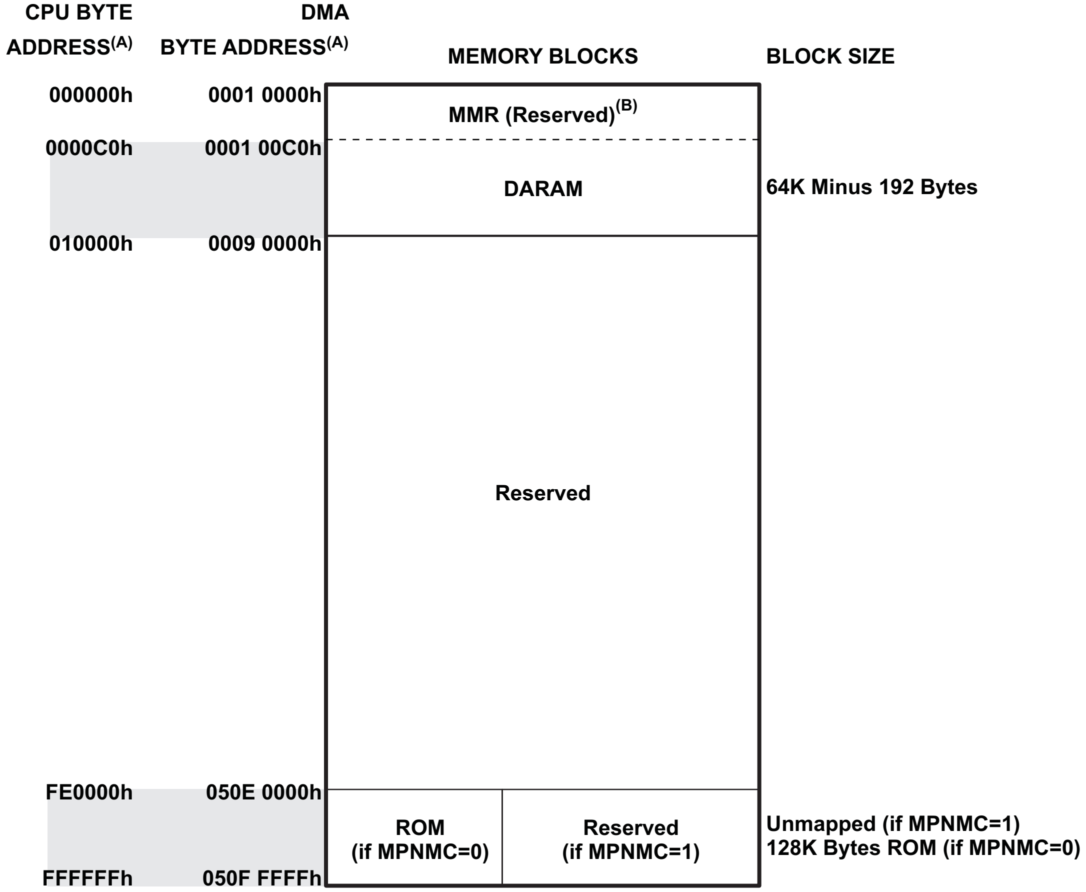


A Address shown represents the first byte address in each block.

B The first 192 bytes are reserved for memory-mapped registers (MMRs).

C The USB controller does not have access to DARAM.

Figure 1-5. TMS320C5532 Memory Map



A Address shown represents the first byte address in each block.

B The first 192 bytes are reserved for memory-mapped registers (MMRs).

1.2.1.1 On-Chip Dual-Access RAM (DARAM)

The DARAM is located in the CPU byte address range 00 00C0h - 00 FFFFh and is composed of eight blocks of 4K words each (see Table 1-3). Each DARAM block can perform two accesses per cycle (two reads, two writes, or a read and a write). DARAM can be accessed by the internal program, data, and DMA buses.

As shown in Table 1-3, the DMA controllers access DARAM at an address offset 0x0001_0000 from the CPU memory byte address space.

Table 1-3. DARAM Blocks

Memory Block	CPU Byte Address Range	DMA/USB Controller Byte Address Range
DARAM 0[1]	00 00C0h - 00 1FFFh	0001 00C0h - 0001 1FFFh
DARAM 1	00 2000h - 00 3FFFh	0001 2000h - 0001 3FFFh
DARAM 2	00 4000h - 00 5FFFh	0001 4000h - 0001 5FFFh
DARAM 3	00 6000h - 00 7FFFh	0001 6000h - 0001 7FFFh
DARAM 4	00 8000h - 00 9FFFh	0001 8000h - 0001 9FFFh
DARAM 5	00 A000h - 00 BFFFh	0001 A000h - 0001 BFFFh
DARAM 6	00 C000h - 00 DFFFh	0001 C000h - 0001 DFFFh
DARAM 7	00 E000h - 00 FFFFh	0001 E000h - 0001 FFFFh

[1] First 192 bytes are reserved for memory-mapped registers (MMRs).

1.2.1.2 On-Chip Single-Access RAM (SARAM) (Not Available for TMS320C5532)

For TMS320C5535, the SARAM is located at the CPU byte address range 01 0000h–04 FFFFh (SARAM 0–31) and is composed of 32 blocks of 4K words each.

For TMS320C5534, the SARAM is located at the CPU byte address range 01 0000h–03 FFFFh (SARAM 0–23) and is composed of 24 blocks of 4K words each.

For TMS320C5533, the SARAM is located at the CPU byte address range 01 0000h–01 FFFFh (SARAM 0–7) and is composed of 8 blocks of 4K words each. See Table 1-4.

Each SARAM block can perform one access per cycle (one read or one write). SARAM can be accessed by the internal program, data, and DMA buses.

As shown in Table 1-4, the DMA controllers access SARAM at an address offset 0x0008_0000 from the CPU memory byte address space.

Table 1-4. SARAM Blocks

Memory Block	CPU Byte Address Range	DMA/USB Controller Byte Address Range
SARAM 0	01 0000h - 01 1FFFh	0009 0000h - 0009 1FFFh
SARAM 1	01 2000h - 01 3FFFh	0009 2000h - 0009 3FFFh
SARAM 2	01 4000h - 01 5FFFh	0009 4000h - 0009 5FFFh
SARAM 3	01 6000h - 01 7FFFh	0009 6000h - 0009 7FFFh
SARAM 4	01 8000h - 01 9FFFh	0009 8000h - 0009 9FFFh
SARAM 5	01 A000h - 01 BFFFh	0009 A000h - 0009 BFFFh
SARAM 6	01 C000h - 01 DFFFh	0009 C000h - 0009 DFFFh
SARAM 7	01 E000h - 01 FFFFh	0009 E000h - 0009 FFFFh
SARAM 8	02 0000h - 02 1FFFh	000A 0000h - 000A 1FFFh
SARAM 9	02 2000h - 02 3FFFh	000A 2000h - 000A 3FFFh
SARAM 10	02 4000h - 02 5FFFh	000A 4000h - 000A 5FFFh
SARAM 11	02 6000h - 02 7FFFh	000A 6000h - 000A 7FFFh
SARAM 12	02 8000h - 02 9FFFh	000A 8000h - 000A 9FFFh
SARAM 13	02 A000h - 02 BFFFh	000A A000h - 000A BFFFh
SARAM 14	02 C000h - 02 DFFFh	000A C000h - 000A DFFFh
SARAM 15	02 E000h - 02 FFFFh	000A E000h - 000A FFFFh
SARAM 16	03 0000h - 03 1FFFh	000B 0000h - 000B 1FFFh
SARAM 17	03 2000h - 03 3FFFh	000B 2000h - 000B 3FFFh
SARAM 18	03 4000h - 03 5FFFh	000B 4000h - 000B 5FFFh
SARAM 19	03 6000h - 03 7FFFh	000B 6000h - 000B 7FFFh
SARAM 20	03 8000h - 03 9FFFh	000B 8000h - 000B 9FFFh
SARAM 21	03 A000h - 03 BFFFh	000B A000h - 000B BFFFh
SARAM 22	03 C000h - 03 DFFFh	000B C000h - 000B DFFFh
SARAM 23	03 E000h - 03 FFFFh	000B E000h - 000B FFFFh
SARAM 24	04 0000h - 04 1FFFh	000C 0000h - 000C 1FFFh
SARAM 25	04 2000h - 04 3FFFh	000C 2000h - 000C 3FFFh
SARAM 26	04 4000h - 04 5FFFh	000C 4000h - 000C 5FFFh
SARAM 27	04 6000h - 04 7FFFh	000C 6000h - 000C 7FFFh
SARAM 28	04 8000h - 04 9FFFh	000C 8000h - 000C 9FFFh
SARAM 29	04 A000h - 04 BFFFh	000C A000h - 000C BFFFh
SARAM 30	04 C000h - 04 DFFFh	000C C000h - 000C DFFFh
SARAM 31	04 E000h - 04 FFFFh	000C E000h - 000C FFFFh

1.2.1.3 On-Chip Single-Access Read-Only Memory (SAROM)

The zero-wait-state ROM is located at the CPU byte address range FE 0000h - FF FFFFh. The ROM is composed of four 16K-word blocks, for a total of 128K-bytes of ROM. Each ROM block can perform one access per cycle (one read or one write). ROM can be accessed by the internal program or data buses, but not the DMA buses. The ROM address space can be mapped by software to the external memory or to the internal ROM via the MPNMC bit in the ST3 status register.

The standard device includes a bootloader program resident in the ROM and the bootloader code is executed immediately after hardware reset. When the MPNMC bit field of the ST3 status register is set through software, the on-chip ROM is disabled and not present in the memory map, and byte address range FE 0000h - FF FFFFh is unmapped. A hardware reset always clears the MPNMC bit, so it is not possible to disable the ROM at hardware reset. However, the software reset instruction does not affect the MPNMC bit. The ROM can be accessed by the program and data buses. Each SAROM block can perform one word read access per cycle.

Table 1-5. SAROM Blocks

Memory Block	CPU Byte Address Range	CPU Word Address Range
SAROM0	FE 0000h - FE 7FFFh	7F 0000h - 7F 3FFFh
SAROM1	FE 8000h - FE FFFFh	7F 4000h - 7F 7FFFh
SAROM2	FF 0000h - FF 7FFFh	7F 8000h - 7F BFFFh
SAROM3	FF 8000h - FF FFFFh	7F C000h - 7F FFFFh

1.2.2 I/O Memory Map

The C5x DSP has a separate memory map for peripheral and system registers, called I/O space. This space is 64K-words in length and is accessed via word read and write instructions dedicated for I/O space.

Separate documentation for I/O space registers related to each peripheral exists and is listed in the preface of this guide. System registers, which provide system-level control and status, are described in detail in other sections throughout this guide. Unused addresses in I/O space should be treated as reserved and should not be accessed. Accessing unused I/O space addresses may stall or hang the DSP.

Each of the four DMA controllers has access to a different set of peripherals and their I/O space registers. This is shown in Section 1.7.4.

NOTE: Writting to I/O space registers incurs in at least 2 CPU cycle latency. Thus, when configuring peripheral devices, wait at least two cycles before accessing data from the peripheral. When more than one peripheral register is updated in a sequence, the CPU only needs to wait following the final register write. The users should consult the respective peripheral user's guide to determine if a peripheral requires additional initialization time.

Before accessing any peripheral register, make sure the peripheral is not held in reset and its internal clock is enabled. The peripheral reset control register (Section 1.7.5.2) and the peripheral clock gating control registers (Section 1.5.3.2.1) control these functions. Accessing a peripheral whose clocks are gated will either return the value of the last address read from the peripheral (when the clocks were last ON) or it may possibly hang the DSP -- depending on the peripheral.

1.3 Device Clocking

1.3.1 Overview

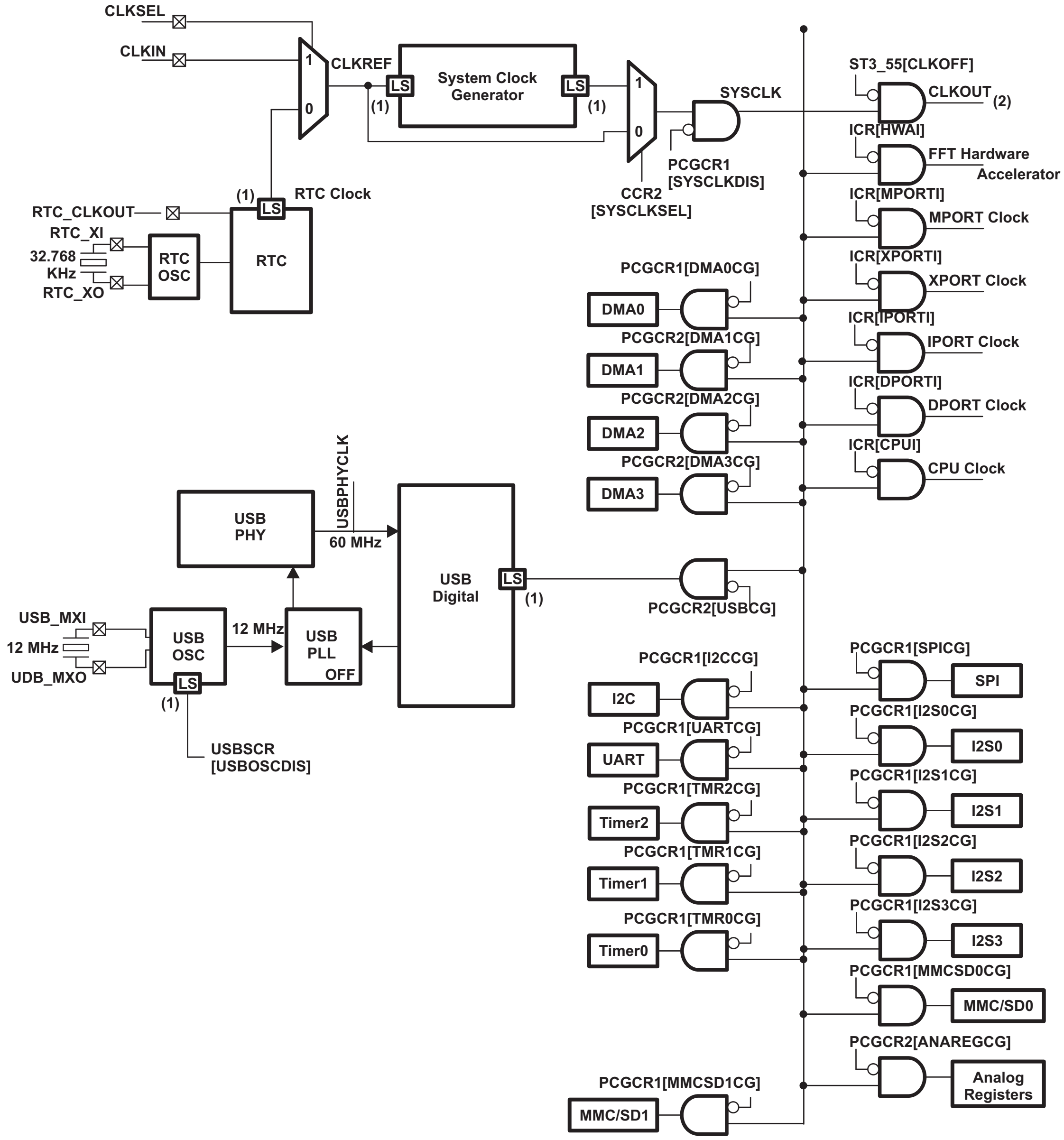
The DSP requires two primary reference clocks: a system reference clock and a USB reference clock. The system clock, which is used by the CPU and most of the DSP peripherals, is controlled by the system clock generator. The system clock generator features a software-programmable PLL multiplier and several dividers. The system clock generator accepts an input reference clock from the CLKIN pin or the output clock of the 32.768-KHz real-time clock (RTC) oscillator. The selection of the input reference clock is based on the state of the CLK_SEL pin. The CLK_SEL pin is required to be statically tied high or low and cannot change dynamically after reset. The system clock generator can be used to modify the system reference clock signal according to software-programmable multiplier and dividers. The resulting clock output, the DSP system clock, is passed to the CPU, peripherals, and other modules inside the DSP. Alternatively, the system clock generator can be fully bypassed and the input reference clock can be passed directly to the DSP system clock. The USB reference clock is generated using a dedicated on-chip oscillator with a 12 MHz external crystal connected to the USB_MXI and USB_MXO pins. This crystal is not required if the USB peripheral is not being used. The USB oscillator cannot be used to provide the system reference clock.

The RTC oscillator generates a clock when a 32.768-KHz crystal is connected to the RTC_XI and RTC_XO pins. RTC core (CV_{DDRTC}) must be powered all the time but the 32.768-KHz crystal can be disabled if CLKIN is used as the clock source for the DSP. However, when the RTC oscillator is disabled, the RTC peripheral will not operate and the RTC registers (I/O address range 1900h - 197Fh) will not be accessible. This includes the RTC power management register (RTCPMGT) which controls the RTCLKOUT and WAKEUP pins. To disable the RTC oscillator, connect the RTC_XI pin to CV_{DDRTC} and the RTC_XO pin to ground.

The USB oscillator is powered down at hardware reset. It must be enabled (by the NNN register) and must be allowed to settle for an amount of time specified by USB Oscillator Startup Time parameter in the device specific manual before using the USB peripheral.

Figure 1-6 shows the overall DSP clock structure. For detailed specifications on clock frequency, voltage requirements, and oscillator/crystal requirements, see the device-specific data manual.

Figure 1-6. DSP Clocking Diagram



(1) LS = Level Shifter

(2) The CLKOUT pin's output driver is enabled/disabled through the CLKOFF bit of the CPU ST3_55 register. At the beginning of the boot sequence, the on-chip Bootloader sets CLKOFF = 1 and CLKOUT pin is disabled (high-impedance). For more information on the ST3_55 register, see the TMS320C55x 3.0 CPU (SWPU073), Algebraic Instruction Set (SWPU068E), and Mnemonic Instruction Set (SWPU067E) reference guides.

1.3.2 Clock Domains

The device has many clock domains defined by individually disabled portions of the clock tree structure. Understanding the clock domains and their clock enable/disable control registers is very important for managing power and for ensuring clocks are enabled for domains that are needed. By disabling the clocks and thus the switching current in portions of the chip that are not used, lower dynamic power consumption can be achieved and prolonging battery life.

Figure 1-6 shows the clock tree structure with the clock gating represented by the AND gates. Each AND gate shows the controlling register that allows the downstream clock signal to be enabled/disabled. Once disabled most clock domains can be re-enabled, when the associated clock domain logic is needed, via software running on the CPU. But some domains actually stop the clocks to the CPU and therefore software running on the CPU cannot be responsible for re-enabling those clock domains. Other mechanism must exist for restarting those clocks, and the specific cases are listed below:

- The System Clock Generator (PLL) can be powered-down by writing a 1 to PLL_PWRDN bit in the clock generator control register CGCR1. This stops the PLL from oscillating and shuts down its analog circuits. It is important to bypass the System Clock Generator by writing 0 to SYSCLKSEL bit in CCR2 (clock confguration register 2) prior to powering it down, else the CPU will loose its clock and not be able to recover without hardware reset.

NOTE: Failsafe logic exists to prevent selecting the PLL clock if it has been powered down but this logic does not protect against powering down the PLL while it is selected as the system clock source. Therefore, software should always maintain responsibility for bypassing the PLL prior to and whenever it is powered down.

- The SYSCLKDIS bit in PCGCR1 [clock gating control register 1) is the master clock gater. Asserting this bit causes the main system clock, SYSCLK, to stop and, therefore, the CPU and all peripherals no longer receive clocks. The WAKEUP pin, INT0 & INT1 pin, or RTC interrupt can be used to re-enable the clock from this condition.
- The ICR bit in CPUI(clock gating control register) gates clocks to the CPU and uses the CPU’s *idle* instruction to initiate the clock off mode. Any non-masked interrupt can be used to re-enable the CPU clocks.

1.4 System Clock Generator

1.4.1 Overview

The system clock generator (Figure 1-7) features a software-programmable PLL multiplier and several dividers. The clock generator accepts an input clock from the CLKIN pin or the output clock of the real-time clock (RTC) oscillator. The clock generator offers flexibility and convenience by way of software-configurable multiplier and divider to modify the clock rate internally. The resulting clock output, SYSCLK, is passed to the CPU, peripherals, and other modules inside the DSP.

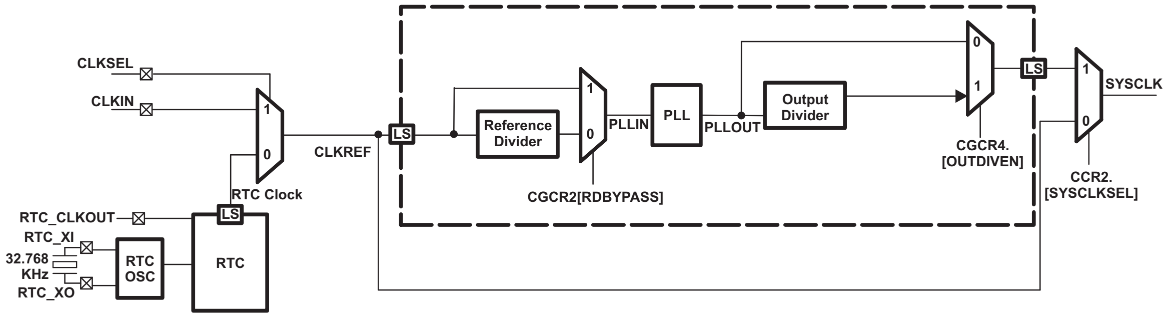
A set of registers are provided for controlling and monitoring the activity of the clock generator. You can write to the SYSCLKSEL bit in CCR2 register to toggle between the two main modes of operation:

- In the BYPASS MODE (see Section 1.4.3.1), the entire clock generator is bypassed, and the frequency of SYSCLK is determined by CLKIN or the RTC oscillator output. Once the PLL is bypassed, the PLL can be powered down to save power.
- In the PLL MODE (see Section 1.4.3.2), the input frequency can be both multiplied and divided to produce the desired SYSCLK frequency, and the SYSCLK signal is phase-locked to the input clock signal (CLKREF).

The clock generator bypass mux (controlled by SYSCLKSEL bit in CCR2 register) is a glitchfree mux, which means that clocks will be switched cleanly and not short cycle pulses when switching among the BYPASS MODE and PLL MODE.

For debug purposes, the CLKOUT pin can be used to see different clocks within the clock generator. For details, see Section 1.4.2.3.

Figure 1-7. Clock Generator



1.4.2 Functional Description

The following sections describe the multiplier and dividers of the clock generator.

1.4.2.1 Multiplier and Dividers

The clock generator has a one multiplier and a two programmable dividers: one before the PLL input and one on the PLL output. The PLL can be programmed to multiply the PLL input clock, PLLIN, using a x4 to x4099 multiplier value. The reference clock divider can be programmed to divide the clock generator input clock from a /4 to /4099 divider ratio and may be bypassed. The Reference Divider and RDBYPASS mux must be programmed such that the PLLIN frequency range is 32.786 KHz to 170 KHz. At the output of the PLL, the output divider can be used to divide the PLL output clock, PLLOUT, from a /1 to a /128 divider ratio and may also be bypassed. The PLL output, PLLOUT, frequency must be programmed within the range of at least 60 MHz and no more than the maximum operating frequency defined by the datasheet, Fsysclk_max parameter. See Table 1-11 for allowed values of PLLIN, PLLOUT, and SYSCLK. Keep in mind that programming the output divider with an odd divisor value other than 1 will result in a non-50% duty cycle SYSCLK. This is not a problem for any of the on-chip logic, but the non-50% duty cycle will be visible on chip pins such as EM_SDCLK (in full-rate mode) and CLKOUT. See Table 1-11 for allowed values of PLLIN, PLLOUT, and SYSCLK.

The multiplier and divider ratios are controlled through the PLL control registers. The M bits define the multiplier rate. The RDRATIO and ODRATIO bits define the divide ratio of the reference divider and programmable output divider, respectively. The RDBYPASS and OUTDIVEN bits are used to enable or bypass the dividers. Table 1-6 lists the formulas for the output frequency based on the setting of these bits.

The clock generator must be placed in BYPASS MODE when any PLL dividers or multipliers are changed. Then, it must remain in BYPASS MODE for at least 4 mS before switching to PLL MODE.

Table 1-6. PLL Output Frequency Configuration

RDBYPASS	OUTDIVEN	SYSCLK Frequency
0	0	$\text{CLKREF} \times \frac{(\text{M} + 4)}{\text{RDRATIO} + 4}$
0	1	$\text{CLKREF} \times \frac{(\text{M} + 4)}{\text{RDRATIO} + 4} \times \frac{1}{\text{ODRATIO} + 1}$
1	0	$\text{CLKREF} \times [\text{M} + 4]$
1	1	$\text{CLKREF} \times [\text{M} + 4] \times \frac{1}{\text{ODRATIO} + 1}$

1.4.2.2 Powering Down and Powering Up the System PLL

To save power, you can put the PLL in its power down mode. You can power down the PLL by setting the PLL_PWRDN = 1 in the clock generator control register CGCR1. However, before powering down the PLL, you must first place the clock generator in bypass mode.

When the PLL is powered up (PLL_PWRDN = 0), the PLL will start its phase-locking sequence. You must keep the clock generator in BYPASS MODE for at least 4 mS while the phase-locking sequence is ongoing. See Section 1.4.3.2 for more details on the PLL_MODE of the clock generator.

1.4.2.3 CLKOUT Pin

For debug purposes, the DSP includes a CLKOUT pin which can be used to tap different clocks within the clock generator. The SRC bits of the CLKOUT control source register (CCSSR) can be used to specify the source for the CLKOUT pin (see Figure 1-8 and Table 1-7).

> **NOTE:** There is no internal logic to prevent glitches while changing the CLKOUT source. Also there is no provision for internally dividing down the CLKOUT frequency other than the options inherently available for selecting the CLKOUT source.

The CLKOUT pin's output driver is enabled/disabled through the CLKOFF bit of the CPU ST3_55 register. At hardware reset, CLKOFF is cleared to 0 so that the clock is visible for debug purposes. But within the bootloader romcode, CLKOFF is set to 1 to conserve power. After the bootloader finishes, the customer application code is free to re-enable CLKOUT. For more information on the ST3_55 register, see the following reference guides:

- *TMS320C55x 3.0 CPU Reference Guide* (SWPU073
- *TMS320C55x v3.x CPU Algebraic Instruction Set Reference Guide* (SWPU068E)
- *TMS320C55x v3.x CPU Mnemonic Instruction Set Reference Guide* (SWPU067E)

The slew rate (i.e., dV/dt) of the CLKOUT pin can be controlled by the CLKOUTSR bits in the output slew rate control register (OSRCR). This feature allows for additional power savings when the CLKOUT pin does not need to drive large loads.

Figure 1-8. CLKOUT Control Source Select Register (CCSSR) [1C24h]

15 4	3 0
Reserved	SRC
R-0	R/W-Bh

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-7. CLKOUT Control Source Select Register (CCSSR) Field Descriptions

Bit	Field	Value	Description
15-4	Reserved	0	Reserved.
3-0	SRC		CLKOUT source bits. These bits specify the source clock for the CLKOUT pin.
		0	CLKOUT pin outputs System PLL output clock, PLLOUT.
		1h	CLKOUT pin is set high.
		2h	CLKOUT pin outputs System PLL output clock, PLLOUT.
		3h	CLKOUT pin is set low.
		4h	CLKOUT pin outputs System PLL output clock, PLLOUT.
		5h	CLKOUT pin is set low.
		6h	CLKOUT pin outputs System PLL output clock, PLLOUT.
		7h	CLKOUT pin outputs USB PLL output clock.
		8h	CLKOUT pin outputs System PLL output clock, PLLOUT.
		9h	CLKOUT pin outputs SAR clock.[1]
		Ah	CLKOUT pin outputs System PLL output clock, PLLOUT.

[1] SAR available on only TMS320C5535.

Table 1-7. CLKOUT Control Source Select Register (CCSSR) Field Descriptions (continued)

Bit	Field	Value	Description
		Bh	CLKOUT pin outputs system clock, SYSCLK (default mode).
		Ch	CLKOUT pin outputs System PLL output clock, PLLOUT.
		Dh	Reserved, do not use.
		Eh	CLKOUT pin outputs System PLL output clock, PLLOUT.
		Fh	CLKOUT pin outputs USB PLL output clock.

1.4.2.4 DSP Reset Conditions of the System Clock Generator

The following sections describe the operation of the system clock generator when the DSP is held in reset state and the DSP is removed from its reset state.

1.4.2.4.1 Clock Generator During Reset

During reset, the PLL_PWRDN bit of the clock generator control register 1 (CGCR1) is set to 1, and the PLL does not generate an output clock. Furthermore, the SYSCLKSEL bit of the clock configuration register 2 (CCR2) defaults to 0 (BYPASS MODE), and the system clock (SYSCLK) is driven by either the CLKIN pin or the real-time clock (RTC). See Section 1.4.3.1 for more information on the bypass mode of the clock generator.

1.4.2.4.2 Clock Generator After Reset

After reset, the on-chip bootloader programs the system clock generator based on the input clock selected via the CLK_SEL pin. If CLK_SEL = 0, the bootloader programs the system clock generator and sets the system clock to 12.288 MHz. If CLK_SEL = 1, the bootloader bypasses the system clock generator altogether and the system clock is driven by the CLKIN pin. In this case, the CLKIN frequency is expected to be 11.2896 MHz, 12.0 MHz, or 12.288 MHz. While the bootloader tries to boot from the USB , the clock generator is programmed to output approximately 36 MHz.

1.4.3 Configuration

1.4.3.1 BYPASS MODE

When the system clock generator is in the BYPASS MODE, the clock generator is not used and the system clock (SYSCLK) is driven by either the CLKIN pin or the real-time clock (RTC).

NOTE: In bypass mode, the PLL is not automatically powered down and will still consume power. For maximum power savings, the PLL should be placed in its power-down mode. See Section 1.4.2.2 for more details.

1.4.3.1.1 Entering and Exiting the BYPASS MODE

To enter the bypass mode, write a 0 to the SYSCLKSEL bit in the clock configuration register 2 (CCR2). In bypass mode, the frequency of the system clock (SYSCLK) is determined by the CLK_SEL pin. If CLK_SEL = 0, SYSCLK is driven by the output of the RTC. Otherwise, SYSCLK will be driven by the CLKIN pin.

To exit the BYPASS MODE, ensure the PLL has completed its phase-locking sequence by waiting at least 4 ms and then write a 1 to the SYSCLKSEL bit. The frequency of SYSCLK will then be determined by the multiplier and divider ratios of the PLL System Clock Generator.

If the clock generator is in the PLL MODE and you want to reprogram the PLL or any of the dividers, you must set the clock generator to BYPASS MODE before changing the PLL and divider settings.

Logic within the clock generator ensures that there are no clock glitches during the transition from PLL MODE to BYPASS MODE and vice versa.

1.4.3.1.2 Register Bits Used in the BYPASS MODE

Table 1-8 describes the bits of the clock generator control registers that are used in the BYPASS MODE. For detailed descriptions of these bits, see Section 1.4.4.

Table 1-8. Clock Generator Control Register Bits Used In BYPASS MODE

Register Bit	Role in BYPASS MODE
SYSCLKSEL	Allows you to switch to the PLL or BYPASS MODES.
PLL_PWRDN	Allows you to power down the PLL.

1.4.3.1.3 Setting the System Clock Frequency In the BYPASS MODE

In the BYPASS MODE, the frequency of SYSCLK is determined by the CLK_SEL pin. If CLK_SEL = 0, SYSCLK is driven by the output of the RTC. Otherwise, SYSCLK will be driven by the CLKIN pin.

NOTE: The CLK_SEL pin must be statically tied high or low; it cannot be changed after the device has been powered up.

Table 1-9. Output Frequency in Bypass Mode

CLK_SEL	SYSCLK Source / Frequency
1	CLKIN, expected to be one of the following values by the bootloader: 11.2896 MHz, 12.0MHz, or 12.288 MHz
0	RTC clock = 32.768 kHz

The state of the CLK_SEL pin is read via the CLKSELSTAT bit in the CCR2 register.

1.4.3.2 PLL MODE

In PLL MODE, the frequency of the input clock signal (CLKREF) can be both multiplied and divided to produce the desired output frequency, and the output clock signal is phase-locked to the input clock signal.

1.4.3.2.1 Entering and Exiting the PLL MODE

To enter the PLL_MODE from BYPASS_MODE, first program the PLL to the desired frequency. You must always ensure the PLL has completed its phase-locking sequence before switching to PLL MODE. This PLL has no lock indicator as such indicators are notoriously unreliable. Instead, a fixed amount of time must be allowed to expire while in BYPASS_MODE to allow the PLL to lock. After 4 msec, write a 1 to the SYSCLKSEL bit in the clock configuration register 2 (CCR2) to set the system clock to the output of the PLL.

Whenever PLL needs to be reprogrammed, first the clock generator must be in bypass mode, and then changed to PLL configuration. After waiting 4 msec, write a 1 to the SYSCLKSEL bit to get into the PLL MODE.

Logic within the clock generator ensures that there are no clock glitches during the transition from BYPASS MODE to PLL MODE and vice versa.

1.4.3.2.2 Register Bits Used in the PLL Mode

Table 1-10 describes the bits of the clock generator control registers that are used in the PLL MODE. For detailed descriptions of these bits, see Section 1.4.4.

Table 1-10. Clock Generator Control Register Bits Used In PLL Mode

Register Bit	Role in Bypass Mode
SYSCLKSEL	Allows you to switch to the PLL or bypass modes.
RDBYPASS	Determines whether reference divider should be bypassed or used.
RDRATIO	Specifies the divider ratio of the reference divider.
M	Specify the multiplier value for the PLL.
OUTDIVEN	Determines whether the output divider is bypassed.
ODRATIO	Specifies the divider ratio of the output divider.

1.4.3.2.3 Frequency Ranges for Internal Clocks

There are specific minimum and maximum frequencies for all the internal clocks. Table 1-11 lists the minimum and maximum frequencies for the internal clocks for the DSP.

NOTE: For actual maximum operating frequencies, see the device-specific data sheet.

Table 1-11. PLL Clock Frequency Ranges

Clock Signal Name	CV_{DD} = 1.05 V			CV_{DD} = 1.3 V			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
CLKIN[(1)]		11.2896, 12, or 12.288			11.2896, 12, or 12.288		MHz
RTC Clock		32.768			32.768		KHz
PLLIN	32.0		170	32.0		170	KHz
PLLOUT	60		120	60		120	MHz
SYSCLK	0		50	0		100	MHz
PLL_LOCKTIME	4			4			ms

[(1)] These CLKIN values are used when the CLK_SEL pin = 1. Bootloader assumes one of these CLKIN frequencies.

1.4.3.2.4 Setting the Output Frequency for the PLL MODE

The clock generator output frequency configured based on the settings programmed in the clock generator control registers. The output frequency depends on primarily on three factors: the reference divider value, the PLL multiplier value, and the output divider value (see Figure 1-7). Based on the register settings controlling these divider and multiplier values, you can calculate the frequency of the output clock using the formulas listed in Table 1-6.

Follow these steps to determine the values for the different dividers and multipliers of the system clock generator:

1. With the desired clock frequency in mind, choose a PLLOUT frequency that falls within the range listed in Table 1-11. Keep in mind that you can use the programmable output divider to divide the output frequency of the PLL.
2. Determine the divider ratio for the reference divider that will generate the PLLIN frequency that meets the requirements listed in Table 1-10. When possible, choose a high value for PLLIN to optimize PLL performance. If the DSP is being clocked by the RTC oscillator output, the reference divider must bypassed (set RDBYPASS = 1); PLLIN will be 32.768 kHz.
3. Determine a multiplier value that generates the desired PLLOUT frequency given the equation: multiplier = round(PLLOUT/PLLIN).
4. Using the multiplier, figure out the values for M (PLL multiplier = M + 4).

Table 1-12 shows programming examples for different PLL MODE frequencies.

Table 1-12. Examples of Selecting a PLL MODE Frequency, When CLK_SEL=L

RDBYPASS	OUTDIVEN	M	RDRATIO	ODRATIO	PLL Output Frequency
1	1	BE8h	X	1	32.768KHz x (BE8h + 4)/2 = 50.00 MHz
1	0	BE8h	X	X	32.768KHz x (BE8h + 4) = 100.01 MHz

1.4.3.2.5 Lock Time

As previously discussed, you must place the clock generator in bypass mode before changing the PLL settings. The time it takes the PLL to complete its phase-locking sequence is referred to as the lock time. The PLL has a lock time of 4 ms. Software is responsible for ensuring the PLL remains in BYPASS_MODE for at least 4 ms before switching to PLL_MODE.

1.4.3.2.6 Software Steps To Modify Multiplier and Divider Ratios

You can follow the steps below to program the PLL of the DSP clock generator. The recommendation is to stop all peripheral operation before changing the PLL frequency, with the exception of the device CPU and USB. The device CPU must be operational to program the PLL controller. Software is responsible for ensuring the PLL remains in BYPASS_MODE for at least 4 ms before switching to PLL_MODE.

1. Make sure the clock generator is in BYPASS MODE by setting SYSCLKSEL = 0.
2. Program RDRATIO, M, and RDBYPASS in CGCR1 and CGCR2 according to your required settings.
3. Program ODRATIO and OUTDIVEN in CGCR4 according to your required settings.
4. Write 0806h to the INIT field of CGCR3.
5. Set PLL_PWRDN = 0.
6. Wait 4 ms for the PLL to complete its phase-locking sequence.
7. Place the clock generator in its PLL MODE by setting SYSCLKSEL = 1.

Note: This is a suggested sequence. It is most important to have all programming done before the last step to place the clock generator in PLL MODE.

1.4.4 Clock Generator Registers

Table 1-13 lists the registers associated with the clock generator of the DSP. The clock generator registers can be accessed by the CPU at the 16-bit addresses specified in Table 1-13. Note that the CPU accesses all peripheral registers through its I/O space. All other register addresses not listed in Table 1-13 should be considered as reserved locations and the register contents should not be modified.

Table 1-13. Clock Generator Registers

CPU Word Address	Acronym	Register Description	Section
1C20h	CGCR1	Clock Generator Control Register 1	Section 1.4.4.1
1C21h	CGCR2	Clock Generator Control Register 2	Section 1.4.4.2
1C22h	CGCR3	Clock Generator Control Register 3	Section 1.4.4.3
1C23h	CGCR4	Clock Generator Control Register 4	Section 1.4.4.4
1C1Fh	CCR2	Clock Configuration Register 2	Section 1.4.4.5

1.4.4.1 Clock Generator Control Register 1 (CGCR1) [1C20h]

The clock generator control register 1 (CGCR1) is shown in Figure 1-9 and described in Table 1-14.

Figure 1-9. Clock Generator Control Register 1 (CGCR1) [1C20h]

15	14 13	12	11 8
Reserved	Reserved	PLL_PWRDN	M
R/W-0	R/W-0	R/W-1	R/W-0

7 6 5 4 3 2 1 0
M
R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-14. Clock Generator Control Register 1 (CGCR1) Field Descriptions

Bit	Field	Value	Description
15	Reserved	0	Reserved. This bit must be set to 1 for normal operation.
14-13	Reserved	0	Reserved. This bit must be always written to be zero.
12	PLL_PWRDN		PLL power down bit. This bit is used to power down the PLL when it is not being used.
		0	PLL is powered up.
		1	PLL is powered down.
11-0	M	0-FFFh	PLL multiplier value bits. These bits define the PLL multiplier value. Multiplier value = M + 4.

1.4.4.2 Clock Generator Control Register 2 (CGCR2) [1C21h]

The clock generator control register 2 (CGCR2) is shown in Figure 1-10 and described in Table 1-15.

Figure 1-10. Clock Generator Control Register 2 (CGCR2) [1C21h]

15	14 12	11 0
RDBYPASS	Reserved	RDRATIO
R/W-0	R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-15. Clock Generator Control Register 2 (CGCR2) Field Descriptions

Bit	Field	Value	Description
15	RDBYPASS		Reference divider bypass control. When this bit is set to 1 the PLL reference divider is bypassed (i.e., $F_{PLLIN} = F_{CLKREF}$). When this bit is set to 0, the reference clock to the PLL is divided by the reference divider (i.e., $F_{PLLIN} = F_{CLKIN}$ / (RDRATIO+4)). The RDRATIO bits specify the divider value.
		0	Use the reference divider.
		1	Bypass the reference divider.
14-12	Reserved	0	Reserved.
11-0	RDRATIO	0-FFFh	Divider ratio bits for the reference divider. Divider value = RDRATIO + 4. For example, setting RDRATIO = 0 means divide the input clock rate by 4.

1.4.4.3 Clock Generator Control Register 3 (CGCR3) [1C22h]

The clock generator control register 3 (CGCR3) is shown in Figure 1-11 and described in Table 1-16.

Figure 1-11. Clock Generator Control Register 3 (CGCR3) [1C22h]

15	0
INIT	
R/W-0806h	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 1-16. Clock Generator Control Register 3 (CGCR3) Field Descriptions

Bit	Field	Value	Description
15-0	INIT	0x0806h	Initialization bits for the DSP clock generator. These bits are used for testing purposes and must be initialized with 0x806 during PLL configuration for proper operation of the PLL.

1.4.4.4 Clock Generator Control Register 4 (CGCR4) [1C23h]

The clock generator control register 4 (CGCR4) is shown in Figure 1-12 and described in Table 1-17.

Figure 1-12. Clock Generator Control Register 4 (CGCR4) [1C23h]

15-10	9	8	7-0
Reserved	OUTDIVEN	Reserved	ODRATIO
R-0	R/W-0	R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-17. Clock Generator Control Register 4 (CGCR4) Field Descriptions

Bit	Field	Value	Description
15-10	Reserved	0	Reserved.
9	OUTDIVEN		Output divider enable bit. This bit determines whether the output divider of the PLL is are enabled or bypassed.
		0	The output divider is bypassed.
		1	The output divider is enabled.
8	Reserved	0	Reserved.
7-0	ODRATIO	0-FFh	Divider ratio bits for the output divider of the PLL. Divider value = ODRATIO + 1.

1.4.4.5 Clock Configuration Register 2 (CCR2) [1C1Fh]

The clock configuration register 2 (CCR2) is shown in Figure 1-13 and described in Table 1-18.

Figure 1-13. Clock Configuration Register 2 (CCR2) [1C1Fh]

15-6	5-4	3	2	1	0
Reserved	SYSCLKSRC	Reserved	CLKSELSTAT	Reserved	SYSCLKSEL
R-0	R-0	R/W-0	R-0	R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-18. Clock Configuration Register 2 (CCR2) Field Descriptions

Bit	Field	Value	Description
15-6	Reserved	0	Reserved.
5-4	SYSCLKSRC		System clock source status bits. These read-only bits reflect the source for the system clock. This status register exists to indicate that switching from the PLL BYPASS_MODE to the PLL_MODE was successful or not. Logic exists on the chip to prevent switching to PLL_MODE if the PLL has its PWRDN bit already asserted. However, this circuit does not protect against asserting the PWRDN bit after already in PLL_MODE. Therefore, software must ultimately make sure not to do something that would cause the system clock to be lost.
		0	The system clock generator is in bypass mode; SYSCLK is driven by the RTC oscillator output.
		1h	The system clock generator is in PLL mode; the RTC oscillator output provides the input clock.
		2h	The system clock generator is in bypass mode; SYSCLK is driven by CLKIN.
		3h	The system clock generator is in PLL mode; the CLKIN pin provides the input clock.
3	Reserved	0	Reserved. This bit must be written to be 0.
2	CLKSELSTAT		CLK_SEL pin status bit. This reflects the state of the CLK_SEL pin.
		0	CLK_SEL pin is low (RTC input clock selected).
		1	CLK_SEL pin is high (CLKIN input clock selected).
1	Reserved	0	Reserved. This bit must be written to be 0.
0	SYSCLKSEL		System clock source select bit. This bit is used to select between the two main clocking modes for the DSP: bypass and PLL mode. In bypass mode, the DSP clock generator is bypassed and the system clock is set to either CLKIN or the RTC output (as determined by the CLKSEL pin). In PLL mode, the system clock is set to the output of the DSP clock generator. Logic in the system clock generator prevents switching from bypass mode to PLL mode if the PLL is powered down.
		0	Bypass mode is selected.
		1	PLL mode is selected.

1.5 Power Management

1.5.1 Overview

In many applications there may be specific requirements to minimize power consumption for both power supply (and battery) and thermal considerations. There are two components to power consumption: active power and leakage power. Active power is the power consumed to perform work and, for digital CMOS circuits, scales roughly with clock frequency and the amount of computations being performed. Active power can be reduced by controlling the clocks in such a way as to either operate at a clock frequency just high enough to complete the required operation in the required time-line or to run at a high enough clock frequency until the work is complete and then drastically cut the clocks (that is, to bypass mode or clock gate) until additional work must be performed.

Leakage power is due to static current leakage and occurs regardless of the clock rate. Leakage, or standby power, is unavoidable while power is applied and scales roughly with the operating junction temperatures. Leakage power can only be avoided by removing power completely.

The DSP has several means of managing the power consumption, as detailed in the following sections. There is extensive use of automatic clock gating in the design as well as software-controlled module clock gating to not only reduce the clock tree power, but to also reduce module power by freezing its state while not operating. Clock management enables you to slow the clocks down on the chip in order to reduce switching power. Independent power domains allow you to shut down parts of the DSP to reduce static power consumption. When not being used, the internal memory of the DSP can also be placed in a low leakage power mode while preserving the memory contents. The operating voltage and drive strength of the I/O pins can also be reduced to decrease I/O power consumption.

Table 1-19 summarizes all of the power management features included in the DSP.

Table 1-19. Power Management Features

Power Management Features	Description
Clock Management	
PLL power-down	The system PLL can be powered-down when not in use to reduce switching and bias power.
Peripheral clock idle	Peripheral clocks can be idled to reduce switching power.
Dynamic Power Management	
Core Voltage Scaling	The DSP logic support two voltage ranges to allow voltage adjustments on-the-fly, increasing voltage during peak processing power demand and decreasing during low demand.
Static Power Management	
DARAM/SARAM low power modes	The internal memory of the DSP can be placed in a low leakage power mode while preserving memory contents.
Independent power domains	DSP Core (CV_{DD}) and USB Core (USB_V_{DD1P3}, USB_V_{DDA1P3}) can be shut off while other supplies remain powered.
I/O Management	
I/O voltage selection	The operating voltage and/or slew rate of the I/O pins can be reduced (at the expense of performance) to decrease I/O power consumption.
USB power-down	The USB peripheral can be powered-down when not being used.

1.5.2 Power Domains

The DSP has separate power domains which provide power to different portions of the device. The separate power domains allow the user to select the optimal voltage to achieve the lowest power consumption at the best possible performance. Note that several power domains have similar voltage requirements and, therefore, could be grouped under a single voltage domain.

Table 1-20. DSP Power Domains

Power Domains	Description
Real-Time Clock Power Domain (CV_{DDRTC})	This domain powers the real-time clock digital circuits and oscillator pins (RTC_XI, RTC_XO). Nominal supply voltage can be 1.05 V through 1.3 V. **Note:** This domain must be always powered for proper operation.
Core Power Domain (CV_{DD})	This domain powers the digital circuits that include the C55x CPU, on-chip memory, and peripherals. Nominal supply voltage is either 1.05 V or 1.3 V.
	Nominal supply voltage can be 1.8, 2.5, 2.75, or 3.3 V.
Digital I/O Power Domain 2 (DV_{DDIO})	This domain powers all I/Os, except the EMIF I/O, USB I/O, USB oscillator I/O, some of the analog related digital pins, and the real-time clock power domain I/O. Nominal supply voltage can be 1.8, 2.5, 2.75, or 3.3 V.
RTC I/O Power Domain (DV_{DDRTC})	This domain powers the WAKEUP and RTC_CLKOUT pins. Nominal supply voltage can be 1.8, 2.5, 2.75, or 3.3 V.
PLL Power Domain (V_{DDA_PLL})	This domain powers the system clock generator PLL. Nominal supply voltage is 1.3 V. This domain can be powered from the on-chip analog LDO output pin (ANA_LDOO).
Analog Power Domain (V_{DDA_ANA})	This domain powers the power management analog circuits and the 10-bit SAR. Nominal supply voltage is 1.3 V. This domain can be powered from the on-chip analog LDO output pin (ANA_LDOO). **Note:** When externally powered, this domain must be always powered for proper operation.
USB Analog Power Domain (USB_V_{DDA1P3})[1]	This domain powers the USB analog PHY. Nominal supply voltage is 1.3 V.
USB Digital Power Domain (USB_V_{DD1P3})[1]	This domain powers the USB digital module. Nominal supply voltage is 1.3 V.
USB Oscillator Power Domain (USB_V_{DDOSC})[1]	This domain powers the USB oscillator. Nominal supply voltage is 3.3 V.
USB Transceiver & Analog Power Domain (USB_V_{DDA3P3})[1]	This domain powers the USB transceiver. Nominal supply voltage is 3.3 V.
USB PLL Power Domain (USB_V_{DDPLL})[1]	This domain powers the USB PLL. Nominal supply voltage is 3.3 V.
LDOI Power Domain (LDOI)	This domain powers LDO, POR comparator, and I/O supply for some pins. Nominal supply voltage is 1.8 V through 3.6 V. **Note:** This domain must be always powered for proper operation.

[1] Not applicable to TMS320C5532.

1.5.3 Clock Management

As mentioned in Section 1.3.2, there are several clock domains within the DSP. The device supports clock gating features that allows software to disable clocks to entire clock domains or modules within a domain in order to reduce the domain's active power consumption to very-near zero (a very small amount of logic will still see a clock).

There are two distinct methods of clock gating. The first uses the ICR CPU register and the CPU's IDLE instruction. This method is used for the following domains: CPU, IPORT, DPORT, MPORT, XPORT & HWA. See Figure 1-6 for a diagram of these domains. In this method, the ICR is written with a value indicating the desired clock gating configuration and then (possibly much later) the IDLE instruction is executed. The contents of the ICR do not become effective until the IDLE instruction is executed. The second method uses system registers, PCGCR1 & PCGCR2. These registers control most of the peripheral clock domains and writes to this register take effect immediately.

The SYSCLKDIS bit in PCGCR register has global effect and, therefore, is a superset of the two methods. When this bit as asserted the whole device is clock gated with the exceptions of the PLL, the USB PLL, the RTC, and the oscillators.

NOTE: Stopping clocks to a domain or a module within that domain only affects active power consumption; it does not affect leakage power consumption.

NOTE: The on-chip Bootloader idles all peripherals and CPU ports at startup, but it enables some peripherals as it uses them. Application code should not assume all peripherals and CPU ports are disabled. To get the minimum power consumption, make sure to disable all peripherals and CPU ports first and then enable only necessary peripherals and CPU ports before using them.

1.5.3.1 CPU Domain Clock Gating

Two registers are provided to individually configure and monitor the clock gating modes of the CPU domain: the idle configuration register (ICR) and the idle status register (ISTR).

ICR lets you configure how the CPU domain will respond the next time the idle instruction is executed. When you execute the idle instruction, the content of ICR is copied to ISTR. Then the ISTR values are propagated to the different portions of the CPU domain.

In the CPU domain, there are five CPU ports.

1.5.3.1.1 Idle Configuration Register (ICR) [0001h] and IDLE Status Register (ISTR) [0002h]

Table 1-21 describes the read/write bits of ICR, and Table 1-22 describes the read-only bits of ISTR.

NOTE: To prevent an emulation lock up, idle requests to these domains may be overridden or ignored when an emulator is connected to the JTAG port of the DSP.

Figure 1-14. Idle Configuration Register (ICR) [0001h]

15	10	9	8
Reserved		HWAI	IPORTI
R/W-0		R/W-0	R/W-0

7	6	5	4	1	0
MPORTI	XPORTI	DPORTI	IDLECFG		CPUI
R/W-0	R/W-0	R/W-0	R/W-0		R/W-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 1-21. Idle Configuration Register (ICR) Field Descriptions

Bit	Field	Value	Description
15-10	Reserved	0	Reserved.
9	HWAI		FFT hardware accelerator idle control bit. (TMS320C5535 only)
		0	Hardware accelerator remains active after execution of an IDLE instruction.
		1	Hardware accelerator is disabled after execution of an IDLE instruction.
8	IPORTI		Instruction port idle control bit. The IPORT is used for all external memory instruction accesses.
		0	IPORT remains active after execution of an IDLE instruction.
		1	IPORT is disabled after execution of an IDLE instruction.
7	MPORTI		Memory port idle control bit. The memory port is used for all DMA, and USB CDMA transactions into on-chip memory.
		0	MPORT remains active after execution of an IDLE instruction.
		1	MPORT is disabled after execution of an IDLE instruction.
6	XPORTI		I/O port idle control bit. The XPORT is used for all CPU I/O memory transactions.
		0	XPORT remains active after execution of an IDLE instruction.
		1	XPORT is disabled after execution of an IDLE instruction.
5	DPORTI		Data port idle control bit. The data port is used for all CPU external memory data accesses.
		0	DPORT remains active after execution of an IDLE instruction.
		1	DPORT is disabled after execution of an IDLE instruction.
4-1	IDLECFG	0111b	Idle configuration bits. You must always set bit 1, 2 and 3 to 1 and bit 4 to 0 before executing the idle instruction.
0	CPUI		CPU idle control bit.
		0	CPU remains active after execution of an IDLE instruction.
		1	CPU is disabled after execution of an IDLE instruction.

Figure 1-15. Idle Status Register (ISTR) [0002h]

15–10	9	8
Reserved	HWAIS	IPORTIS
R-0	R-0	R-0

7	6	5	4–1	0
MPORTIS	XPORTIS	DPORTIS	Reserved	CPUIS
R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 1-22. Idle Status Register (ISTR) Field Descriptions

Bit	Field	Value	Description
15-10	Reserved	0	Reserved.
9	HWAIS		FFT hardware accelerator idle status bit. (TMS320C5535 only)
		0	Hardware accelerator is active.
		1	Hardware accelerator is disabled.
8	IPORTIS		Instruction port idle status bit. The IPORT is used for all external memory instruction accesses.
		0	IPORT is active.
		1	IPORT is disabled.
7	MPORTIS		Memory port idle status bit. The memory port is used for all DMA, and USB CDMA transactions into on-chip memory.
		0	MPORT is active.
		1	MPORT is disabled.
6	XPORTIS		I/O port idle status bit. The XPORT is used for all CPU I/O memory transactions.
		0	XPORT is active.
		1	XPORT is disabled.
5	DPORTIS		Data port idle status bit. The data port is used for all CPU external memory data accesses.
		0	DPORT is active.
		1	DPORT is disabled.
4-1	Reserved	0	Reserved.
0	CPUIS		CPU idle status bit.
		0	CPU is active.
		1	CPU is disabled.

1.5.3.1.2 Valid Idle Configurations

Not all of the values that you can write to the idle configuration register (ICR) provide valid idle configurations. The valid configurations are limited by dependencies within the system. For example, the IDLECFG bits 1, 2 and 3 of ICR must always be set to 1, and bit 4 must always be cleared to 0. As another example, the XPORT cannot be idled unless the CPU is also idled. Before any part of the CPU domain is idled, you must observe the requirements outlined in Section 1.5.3.2.

A bus error will be generated (BERR = 1 in IFR1) if you execute the idle instruction under any of the following conditions and the idle command will not take effect:

1. If you fail to set IDLECFG = 0111 while setting any of these bits: DPORTI, XPORTI, IPORTI or MPORTI.
2. If you set DPORTI, XPORTI, or IPORTI without also setting CPUI.

Table 1-23. CPU Clock Domain Idle Requirements

To Idle the Following Module/Port	Requirements Before Going to Idle
CPU	No requirements.
FFT Hardware Accelerator	No requirements.
MPORT	DMA controllers, and USB CDMA must not be accessing DARAM or SARAM.

Table 1-23. CPU Clock Domain Idle Requirements (continued)

To Idle the Following Module/Port	Requirements Before Going to Idle
XPORT DPORT	CPU CPUI must also be set.

1.5.3.1.3 *Clock Configuration Process*

The clock configuration indicates which portions of the CPU clock domain will be idle, and which will be active. The basic steps to the clock configuration process are:

1. To idle MPORT, DMA controller, and USB CDMA must not be accessing SARAM or DARAM. If any DMA is in active, wait for completion of the DMA transfer.
2. Write the desired configuration to the idle configuration register (ICR). Make sure that you use a valid idle configuration (see Section 1.5.3.1.2).
3. Apply the new idle configuration by executing the IDLE instruction. The content of ICR is copied to the idle status register (ISTR). The bits of ISTR are then propagated through the CPU domain system to enable or disable the specified clocks. If the CPU domain was idled, then program execution will stop immediately after the idle instruction. If the CPU domain was not idled, then program execution will continue past the idle instruction but the appropriate domains will be idle.

The IDLE instruction cannot be executed in parallel with another instruction.

The CPU, DPORT, XPORT, and IPORT domains are enabled automatically by any unmasked interrupts. There is a logic in the DSP core that enables CPU, DPORT, XPORT, and IPORT (clears the bits 0, 5, 6, and 8 of the ISTR register) asynchronously upon detecting an interrupt signal. Therefore, when an unmasked interrupt signal reaches the DSP core, these domains are un-idled automatically. Once the CPU is enabled, it takes 3 CPU cycles to detect the interrupt in the IFR. Note that HWA and MPORT have to be manually enabled after being disabled.

1.5.3.2 Peripheral Domain Clock Gating

The peripheral clock gating allows software to disable clocks to the DSP peripherals, in order to reduce the peripheral's active power consumption to zero. Aside from the analog logic, the DSP is designed in static CMOS; thus, when a peripheral clock stops, the peripheral's state is preserved, and no active current is consumed. When the clock is restarted the peripheral resumes operating from the stopping point.

NOTE: Stopping clocks to a peripheral only affects active power consumption; it does not affect leakage power consumption.

If a peripheral's clock is stopped while being accessed, the access may not occur completely, and could potentially lock-up the device. To avoid this issue, some peripherals have a clock stop request and acknowledge protocol that allows software to ask the peripheral when it is safe to stop the clocks. This is described further in Section 1.5.3.2.2. For the peripherals that do not have the request/acknowledge protocol, the user must ensure that all of the transactions to the peripheral are finished prior to stopping the clocks.

The procedure to turn peripheral clocks on/off is described in Section 1.5.3.2.3.

Some peripherals provide additional power saving features by clock gating components within its peripheral boundary. See the peripheral-specific user's guide for more details on these additional power saving features.

1.5.3.2.1 Peripheral Clock Gating Configuration Registers (PCGCR1 and PCGCR2) [1C02 - 1C03h]

The peripheral clock gating configuration registers (PCGRC1 and PCGCR2) are used to disable the clocks of the DSP peripherals. In contrast to the idle control register (ICR), these bits take effect within 6 SYSCLK cycles and do not require an idle instruction.

The peripheral clock gating configuration register 1 (PCGCR1) is shown in Figure 1-16 and described in Table 1-24.

Figure 1-16. Peripheral Clock Gating Configuration Register 1 (PCGCR1) [1C02h]

15	14	13	12	11	10	9	8
SYSCLKDIS	I2S2CG	TMR2CG	TMR1CG	Reserved	TMR0CG	I2S1CG	I2S0CG
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
MMCSD1CG	I2CCG	Reserved	MMCSD0CG	DMA0CG	UARTCG	SPICG	I2S3CG
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-24. Peripheral Clock Gating Configuration Register 1 (PCGCR1) Field Descriptions

Bit	Field	Value	Description
15	SYSCLKDIS		System clock disable bit. This bit can be used to turn off the system clock. Setting the WAKEUP pin high enables the system clock. Since the WAKEUP pin is used to re-enable the system clock, the WAKEUP pin must be low to disable the system clock. **NOTE** Disabling the system clock disables the clock to most parts of the DSP, including the CPU.
		0	System clock is active.
		1	System clock is disabled.
14	I2S2CG		I2S2 clock gate control bit. This bit is used to enable and disable the I2S2 peripheral clock.
		0	Peripheral clock is active.
		1	Peripheral clock is disabled.
13	TMR2CG		Timer 2 clock gate control bit. This bit is used to enable and disable the Timer 2 peripheral clock.
		0	Peripheral clock is active.
		1	Peripheral clock is disabled.
12	TMR1CG		Timer 1 clock gate control bit. This bit is used to enable and disable the Timer 1 peripheral clock.
		0	Peripheral clock is active.
		1	Peripheral clock is disabled.
11	Reserved	0	Reserved. You must always write 1 to this bit.
10	TMR0CG		Timer 0 clock gate control bit. This bit is used to enable and disable the Timer 0 peripheral clock.
		0	Peripheral clock is active.
		1	Peripheral clock is disabled.
9	I2S1CG		I2S1 clock gate control bit. This bit is used to enable and disable the I2S1 peripheral clock.
		0	Peripheral clock is active.
		1	Peripheral clock is disabled.
8	I2S0CG		I2S0 clock gate control bit. This bit is used to enable and disable the I2S0 peripheral clock.
		0	Peripheral clock is active.
		1	Peripheral clock is disabled.
7	MMCSD1CG		MMC/SD1 clock gate control bit. This bit is used to enable and disable the MMC/SD1 peripheral clock.
		0	Peripheral clock is active.
		1	Peripheral clock is disabled.
6	I2CCG		I2C clock gate control bit. This bit is used to enable and disable the I2C peripheral clock.
		0	Peripheral clock is active.
		1	Peripheral clock is disabled.
5	Reserved	0	Reserved, you must always write 1 to this bit.

Table 1-24. Peripheral Clock Gating Configuration Register 1 (PCGCR1) Field Descriptions (continued)

Bit	Field	Value	Description
4	MMCSD0CG		MMC/SD0 clock gate control bit. This bit is used to enable and disable the MMC/SD0 peripheral clock.
		0	Peripheral clock is active.
		1	Peripheral clock is disabled.
3	DMA0CG		DMA controller 0 clock gate control bit. This bit is used to enable and disable the peripheral clock the DMA controller 0.
		0	Peripheral clock is active.
		1	Peripheral clock is disabled.
2	UARTCG		UART clock gate control bit. This bit is used to enable and disable the UART peripheral clock. **NOTE** You must request permission before stopping the UART clock through the peripheral clock stop request/acknowledge register (CLKSTOP).
		0	Peripheral clock is active.
		1	Peripheral clock is disabled.
1	SPICG		SPI clock gate control bit. This bit is used to enable and disable the SPI controller peripheral clock.
		0	Peripheral clock is active.
		1	Peripheral clock is disabled.
0	I2S3CG		I2S3 clock gate control bit. This bit is used to enable and disable the I2S3 peripheral clock.
		0	Peripheral clock is active.
		1	Peripheral clock is disabled.

The peripheral clock gating configuration register 2 (PCGCR2) is shown in Figure 1-17 and described in Table 1-25.

Figure 1-17. Peripheral Clock Gating Configuration Register 2 (PCGCR2) [1C03h]

15							8
Reserved							
R-0							

7	6	5	4	3	2	1	0
Reserved	ANAREGCG	DMA3CG	DMA2CG	DMA1CG	USBCG	SARCG	LCDCG
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-25. Peripheral Clock Gating Configuration Register 2 (PCGCR2) Field Descriptions

Bit	Field	Value	Description
15-7	Reserved	0	Reserved.
6	ANAREGCG		Analog registers clock gate control bit. This bit is used to enable and disable the clock to the registers that control the analog domain of the device, i.e. registers in the 7000h-70FFh I/O space address range. **NOTE** When SARCG = 0, the clocks to the analog domain registers are enabled regardless of the ANAREGCG setting.
		0	Clock is active.
		1	Clock is disabled.
5	DMA3CG		DMA controller 3 clock gate control bit. This bit is used to enable and disable the DMA controller 3 peripheral clock.
		0	Peripheral clock is active.
		1	Peripheral clock is disabled.
4	DMA2CG		DMA controller 2 clock gate control bit. This bit is used to enable and disable the DMA controller 2 peripheral clock.
		0	Peripheral clock is active.
		1	Peripheral clock is disabled.
3	DMA1CG		DMA controller 1 clock gate control bit. This bit is used to enable and disable the DMA controller 1 peripheral clock.
		0	Peripheral clock is active.
		1	Peripheral clock is disabled.
2	USBCG		USB clock gate control bit. This bit is used to enable and disable the USB controller peripheral clock. **NOTE** You must request permission before stopping the USB clock through the peripheral clock stop request/acknowledge register (CLKSTOP). This register does not stop the USB PLL.
		0	Peripheral clock is active.
		1	Peripheral clock is disabled.
1	SARCG		SAR clock gate control bit. This bit is used to enable and disable the SAR peripheral clock. **Note:** When SARCG = 0, the clock to the analog domain registers is enabled regardless of the ANAREGCG setting.
		0	Peripheral clock is active.
		1	Peripheral clock is disabled.
0	LCDCG		LCD controller clock gate control bit. This bit is used to enable and disable the LCD controller peripheral clock.
		0	Peripheral clock is active.
		1	Peripheral clock is disabled.

1.5.3.2.2 Peripheral Clock Stop Request/Acknowledge Register (CLKSTOP) [1C3Ah]

You must execute a handshaking procedure before stopping the clock to the USB and UART. This handshake procedure ensures that current bus transactions are completed before the clock is stopped. The peripheral clock stop request/acknowledge register (CLKSTOP) enables this handshaking mechanism.

To stop the clock to the USB or UART, set the corresponding clock stop request bit in the CLKSTOP register, then wait for the peripheral to set the corresponding clock stop acknowledge bit. Once this bit is set, you can idle the corresponding clock in the PCGCR1 and PCGCR2.

To enable the clock to the USB or UART, first enable the clock the peripheral through PCGCR1 or PCGCR2, then clear the corresponding clock stop request bit in the CLKSTOP register.

The peripheral clock stop request/acknowledge register (CLKSTOP) is shown in Figure 1-18 and described in Table 1-26.

Figure 1-18. Peripheral Clock Stop Request/Acknowledge Register (CLKSTOP) [1C3Ah]

15	8
Reserved	
R-0	

7	6	5	4	3	2	1	0
Reserved		URTCLKSTPACK	URTCLKSTPREQ	USBCLKSTPACK	USBCLKSTPREQ	Reserved	
R-0		R-1	R/W-1	R-1	R/W-1	R-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-26. Peripheral Clock Stop Request/Acknowledge Register (CLKSTOP) Field Descriptions

Bit	Field	Value	Description
15-6	Reserved	0	Reserved.
5	URTCLKSTPACK		UART clock stop acknowledge bit. This bit is set to 1 when the UART has acknowledged a request for its clock to be stopped. The UART clock should not be stopped until this bit is set to 1.
		0	The request to stop the peripheral clock has not been acknowledged.
		1	The request to stop the peripheral clock has been acknowledged, the clock can be stopped.
4	URTCLKSTPREQ		UART peripheral clock stop request bit. When disabling the UART internal peripheral clock, you must set this bit to 1 to request permission to stop the clock. After the UART acknowledges the request (URTCLKSTPACK = 1) you can stop the clock through the peripheral clock gating control register 1 (PCGCR1). When enabling the UART internal clock, enable the clock through PCGCR1, then set URTCKLSTPREQ to 0.
		0	Normal operating mode.
		1	Request permission to stop the peripheral clock.
3	USBCLKSTPACK		USB clock stop acknowledge bit. This bit is set to 1 when the USB has acknowledged a request for its clock to be stopped. The USB clock should not be stopped until this bit is set to 1.
		0	The request to stop the peripheral clock has not been acknowledged.
		1	The request to stop the peripheral clock has been acknowledged, the clock can be stopped.
2	USBCLKSTPREQ		USB peripheral clock stop request bit. When disabling the USB internal peripheral clock, you must set this bit to 1 to request permission to stop the clock. After the USB acknowledges the request (USBCLKSTPACK = 1) you can stop the clock through the peripheral clock gating control register 2 (PCGCR2). When enabling the USB internal clock, enable the clock through PCGCR2, then set USBCKLSTPREQ to 0.
		0	Normal operating mode.
		1	Request permission to stop the peripheral clock.
1	Reserved	0	Reserved.
0	Reserved	0	Reserved.

1.5.3.2.3 Clock Configuration Process

The clock configuration indicates which portions of the peripheral clock domain will be idle, and which will be active. The basic steps to the clock configuration process are:

1. Wait for completion of all DMA transfers. You can poll the DMA transfer status and disable DMA transfers through the DMA registers.
2. If idling the USB and UART clock, set the corresponding clock stop request bit in CLKSTOP.
3. Wait for confirmation from the module that its clock can be stopped by polling the clock stop acknowledge bits of CLKSTOP.
4. Set the clock configuration for the peripheral domain through PCGCR1 and PCGCR2. The clock configuration takes place as soon as you write to these registers; the idle instruction is not required

1.5.3.3 Clock Generator Domain Clock Gating

To save power, the system clock generator can be placed in its BYPASS MODE and its PLL can be placed in power down mode. When the system clock generator is in the BYPASS MODE, the clock generator is not used and the system clock (SYSCLK) is driven by either the CLKIN pin or the real-time clock (RTC). For more information entering and exiting the bypass mode of the clock generator, see Section 1.4.3.1.1.

When the clock generator is placed in its bypass mode, the PLL continues to generate a clock output. You can save additional power by powering down the PLL. Section 1.4.2.2 provides more information on powering down the PLL.

1.5.3.4 USB Domain Clock Gating

The USB peripheral has two clock domains. The first is a high speed domain that has its clock supplied by a dedicated USB PLL. The reference clock for the USB PLL is the 12.0 MHz USB oscillator. The clock output from the PLL must support the serial data stream that, in high-speed mode, is at a rate of 480 Mb/s. The second clock into the USB peripheral handles the data once it has been packetized and transported in parallel fashion. This clock supports all of the USB registers, CDMA, FIFO, etc., and is clocked by SYSCLK. In order to keep up with the serial data stream, the USB requires SYSCLK to be at least 30 MHz for low-speed/full-speed modes and at least 60 MHz for high-speed mode.

By stopping both of these clocks, it is possible to reduce the USB's active power consumption (in the digital logic) to zero.

NOTE: Stopping clocks to a peripheral only affects active power consumption; it does not affect leakage power consumption. USB leakage power consumption can be reduced to zero by not powering the USB.

1.5.3.4.1 Clock Configuration Process

The clock configuration process for the USB clock domain consists of disabling the USB peripheral clock followed by disabling the USB on-chip oscillator. This procedure will completely shut off USB module, which does not comply with USB suspend/resume protocol.

To set the clock configuration of the USB clock domain to idle follow these steps:

1. Set the SUSPENDM bit in FADDR register. For more information about the SUSPENDM bit, see Chapter 13.
2. Set the USB clock stop request bit (USBCLKSTREQ) in the CLKSTOP register to request permission to shut off the USB peripheral clock.
3. Wait until the USB acknowledges the clock stop request by polling the USB clock stop acknowledge bit (USBCLKSTPACK) in the CLKSTOP register.
4. Disable the USB peripheral clock by setting USBCG = 1 in the peripheral clock gating control register 2 (PCGCR2).
5. Disable the USB oscillator by setting USBOSCDIS = 1 in the USB system control register (USBSCR).

To enable the USB clock domain, follow these steps:

1. Enable the USB oscillator by setting USBOSCDIS = 0 in USBSCR.
2. Wait for the oscillator to stabilize. Refer to the device-specific data manual for oscillator stabilization time.
3. Enable the USB peripheral clock by setting USBCG = 0 in the peripheral clock gating control register 2 (PCGCR2).
4. Clear the USB clock stop request bit (USBCLKSTREQ) in the CLKSTOP register.
5. Clear the SUSPENDM bit in FADDR register.

1.5.3.4.2 USB System Control Register (USBSCR) [1C32h]

The USB system control register is used to disable the USB on-chip oscillator and to power-down the USB.

The USB system control register (USBSCR) is shown in Figure 1-19 and described in Table 1-27.

Figure 1-19. USB System Control Register (USBSCR) [1C32h]

15	14	13	12	11			8
USBPWDN	USBSESSEND	USBVBUSDET	USBPLLEN	Reserved			
R/W-1	R/W-0	R/W-1	R/W-0	R-0			

7	6	5	4	3	2	1	0
Reserved	USBDATPOL	Reserved		USBOSCBIASDIS	USBOSCDIS	BYTEMODE	
R-0	R/W-1	R-0		R/W-1	R/W-1	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-27. USB System Control Register (USBSCR) Field Descriptions

Bit	Field	Value	Description
15	USBPWDN		USB module power. Asserting USBPWDN puts the USB PHY and PLL in their lowest power state. The USB peripheral is not operational in this state.
		0	USB module is powered.
		1	USB module is powered-down.
14	USBSESSEND		USB VBUS session end comparator enable. The USB VBUS pin has two comparators that monitor the voltage level on the pin. These comparators can be disabled for power savings when not needed.
		0	USB VBUS session end comparator is disabled.
		1	USB VBUS session end comparator is enabled.

Table 1-27. USB System Control Register (USBSCR) Field Descriptions (continued)

Bit	Field	Value	Description
13	USBVBUSDET		USB VBUS detect enable. The USB VBUS pin has two comparators that monitor the voltage level on the pin. These comparators can be disabled for power savings when not needed.
		0	USB VBUS detect comparator is disabled.
		1	USB VBUS detect comparator is enabled.
12	USBPLLEN		USB PLL enable. This is normally only used for test purposes.
		0	Normal USB operation.
		1	Override USB suspend end behavior and force release of PLL from suspend state.
11-7	Reserved	0	Reserved. Always write 0 to these bits.
6	USBDATPOL		USB data polarity bit. Changing this bit can be useful since the data polarity is opposite on type-A and type-B connectors.
		0	Reverse polarity on DP and DM signals.
		1	Normal polarity (normal polarity matching pin names).
5-4	Reserved	0	Reserved.
3	USBOSCBIASDIS		USB internal oscillator bias resistor disable.
		0	Internal oscillator bias resistor enabled (normal operating mode).
		1	Internal oscillator bias resistor disabled. Disabling the internal resistor is primarily for production test purposes. But it can also be used when an external oscillator bias resistor is connected between the USB_MXI and USB_MXO pins (but this is not a recommended configuration).
2	USBOSCDIS		USB oscillator disable bit.
		0	USB internal oscillator enabled.
		1	USB internal oscillator disabled. Causes the USB_MXO pin to be tristated and the oscillator's clock into the core is forced low.
1-0	BYTEMODE		USB byte mode select bits.
		0	Word accesses by the CPU are allowed.
		1h	Byte accesses by the CPU are allowed (high byte is selected).
		2h	Byte accesses by the CPU are allowed (low byte is selected).
		3h	Reserved.

1.5.3.5 RTC Domain Clock Gating

Dynamic RTC domain clock gating is not supported. Note that the RTC oscillator, and by extension the RTC domain, can be permanently disabled by not connecting a crystal and tying off the RTC oscillator pins. However, in this configuration, the RTC must still be powered and the RTC registers starting at I/O address 1900h will not be accessible. This includes the RTC Power Management Register (RTCPMGT) that provides powerdown control to the on-chip LDO and control of the WAKEUP and RTC_CLKOUT pins. See the device-specific data manual for more details on permanently disabling the RTC oscillator.

1.5.4 Static Power Management

1.5.4.1 RTC Power Management Register (RTCPMGT) [1930h]

This register enables static power management with power down and wake up register bits as described in the device-specific data sheet and, more generally, below. The RTC power management register (RTCPMGT) is shown in Figure 1-20 and described in Table 1-28.

Figure 1-20. RTC Power Management Register (RTCPMGT) [1930h]

15 5	4	3	2	1	0
Reserved	WU_DOUT	WU_DIR	BG_PD	LDO_PD	RTCCLKOUTEN
R-0	RW-0	RW-0	RW-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-28. RTC Power Management Register (RTCPMGT) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved
4	WU_DOUT		Wakeup output, active low/Open-drain.
		0	WAKEUP pin driven low.
		1	WAKEUP pin driver is in high impedance.
3	WU_DIR		Wakeup pin direction control.
		0	WAKEUP pin is configured as input.
		1	WAKEUP pin is configured as output.
			NOTE: The WAKEUP pin, when configured as an input, is active high. When it is configured as an output, it is open-drain and thus it should have an external pull-up and it is active low.
2	BG_PD		Powerdown control bit for the bandgap, on-chip LDOs, and the analog POR (power on reset) comparator. This bit shuts down the on-chip LDOs (ANA_LDO, DSP_LDO, and USB_LDO), the Analog POR, and Bandgap reference. BG_PD and LDO_PD are only intended to be used when the internal LDOs supply power to the chip. If the internal LDOs are bypassed and not used then the BG_PD and LDO_PD power down mechanisms should not be used since the POR gets powered down and the POWERGOOD signal would not get generated properly.
			After this bit is asserted, the on-chip LDOs, Analog POR, and the Bandgap reference can only be re-enabled by the WAKEUP pin (being driven HIGH externally) or an enabled RTC alarm or an enabled RTC periodic event interrupt. Once reenabled, the Bandgap circuit takes about 100 msec to charge the external 0.1 μF capacitor on the BG_CAP pin via the the internal resistance of aproxmiately. 320 kΩ.
		0	On-chip LDO, Analog POR, and Bandgap reference are enabled.
		1	On-chip LDO, Analog POR, and Bandgap reference are disabled (shutdown).
1	LDO_PD		On-chip LDOs and Analog POR power down bit. This bit shuts down the on-chip LDOs (ANA_LDO, DSP_LDO, and USB_LDO) and the Analog POR. BG_PD and LDO_PD are only intended to be used when the internal LDOs supply power to the chip. If the internal LDOs are bypassed and not used then the BG_PD and LDO_PD power down mechanisms should not be used since POR gets powered down and the POWERGOOD signal is not generated properly.
			After this bit is asserted, the on-chip LDOs and Analog POR can only be re-enabled by the WAKEUP pin (being driven HIGH externally) or an enabled RTC alarm or an enabled RTC periodic event interrupt. This bit keeps the Bandgap reference turned on to allow a faster wake-up time with the expense power consumption of the Bandgap reference.
		0	On-chip LDO and Analog POR are enabled.
		1	On-chip LDO and Analog POR are disabled (shutdown).
0	RTCCLKOUTEN		Clock-out output enable.
		0	Clock output disabled.
		1	Clock output enabled.

1.5.4.2 RTC Interrupt Flag Register (RTCINTFL) [1920h]

The RTC interrupt flag register (RTCINTFL) is shown in Figure 1-21 and described in Table 1-29.

Figure 1-21. RTC Interrupt Flag Register (RTCINTFL) [1920h]

15	14 - 8
ALARMFL	Reserved
R-0	R-0

7 - 6	5	4	3	2	1	0
Reserved	EXTFL	DAYFL	HOURFL	MINFL	SECFL	MSFL
R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-29. RTC Interrupt Flag Register (RTCINTFL) Field Descriptions

Bit	Field	Value	Description
15	ALARMFL		Indicates that an alarm interrupt has been generated.
		0	Alarm interrupt did not occur.
		1	Alarm interrupt occurred (write 1 to clear).
14-6	Reserved	0	Reserved.
5	EXTFL		External event (WAKEUP pin assertion) has occurred.
		0	External event interrupt has not occurred.
		1	External event interrupt occurred (write 1 to clear).
4	DAYFL		Day event has occurred.
		0	Periodic Day event has not occurred.
		1	Periodic Day event occurred (write 1 to clear).
3	HOURFL		Hour event has occurred.
		0	Periodic Hour event has not occurred.
		1	Periodic Hour event occurred (write 1 to clear).
2	MINFL		Minute Event has occurred.
		0	Periodic Minute event has not occurred.
		1	Periodic Minute event occurred (write 1 to clear).
1	SECFL		Second Event occurred.
		0	Periodic Second event has not occurred.
		1	Periodic Second event occurred (write 1 to clear).
0	MSFL		Millisecond event occurred.
		0	Periodic Millisecond event has not occurred.
		1	Periodic Millisecond event occurred (write 1 to clear).

1.5.4.3 Internal Memory Low Power Modes

To save power, software can place on-chip memory (DARAM or SARAM) in one of two power modes: memory retention mode and active mode. These power modes are activated through the SLPZVDD and SLPZVSS bits of the RAM Sleep Mode Control Register 1-5 (RAMSLPMDCNTLR[1:5]). To activate memory retention mode, set SLPZVDD bit and clear SLPZVSS bit of each memory bank to be put in retention mode. The retention/active mode of each 4kW DARAM and SARAM bank is independently controllable.

When either type of memory is placed in memory retention, read and write accesses are not allowed. In memory retention mode, the memory is placed in a low power mode while maintaining its contents. The contents are retained as long as there are no access attempts to that memory. In active mode, the memory is readily accessible by the CPU, but consumes more leakage power.

For the entire duration that the memory is in retention mode, there can be no attempts to read or write to the memories address range. This includes accesses by the CPU or any DMA. If an access is attempted while in retention mode then the memory contents will be lost.

NOTE: You must wait at least 10 CPU clock cycles after taking memory out of a low power mode before initiating any read or write access.

Table 1-30 summarizes the power modes for both DARAM and SARAM.

Table 1-30. On-Chip Memory Standby Modes

SLPZVDD	SLPZVSS	Mode	CV_{DD} Voltage
1	1	Active - Normal operational mode - Read and write accesses are allowed	1.05 V or 1.3 V
1	0	Retention - Low power mode - Contents are retained - No read or write access is allowed	1.05 V or 1.3 V
0	0	Memory Disabled Mode - Lowest leakage mode - Contents are lost - No read or write access is allowed	1.05 V or 1.3 V

1.5.4.3.1 RAM Sleep Mode Control Register 1 (RAMSLPMDCNTLR1) [1C28h]

The RAM sleep mode control register 1 (RAMSLPMDCNTLR1) is shown in Figure 1-22 through Figure 1-26.

Figure 1-22. RAM Sleep Mode Control Register1 [0x1C28]

15	14	13	12	11	10	9	8
DARAM7 SLPZVDD	DARAM7 SLPZVSS	DARAM6 SLPZVDD	DARAM6 SLPZVSS	DARAM5 SLPZVDD	DARAM5 SLPZVSS	DARAM4 SLPZVDD	DARAM4 SLPZVSS
R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1

7	6	5	4	3	2	1	0
DARAM3 SLPZVDD	DARAM3 SLPZVSS	DARAM2 SLPZVDD	DARAM2 SLPZVSS	DARAM1 SLPZVDD	DARAM1 SLPZVSS	DARAM0 SLPZVDD	DARAM0 SLPZVSS
R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 1-23. RAM Sleep Mode Control Register2 [0x1C2A]

15	14	13	12	11	10	9	8
SARAM7 SLPZVDD	SARAM7 SLPZVSS	SARAM6 SLPZVDD	SARAM6 SLPZVSS	SARAM5 SLPZVDD	SARAM5 SLPZVSS	SARAM4 SLPZVDD	SARAM4 SLPZVSS
R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1
7	6	5	4	3	2	1	0
SARAM3 SLPZVDD	SARAM3 SLPZVSS	SARAM2 SLPZVDD	SARAM2 SLPZVSS	SARAM1 SLPZVDD	SARAM1 SLPZVSS	SARAM0 SLPZVDD	SARAM0 SLPZVSS
R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 1-24. RAM Sleep Mode Control Register3 [0x1C2B]

15	14	13	12	11	10	9	8
SARAM15 SLPZVDD	SARAM15 SLPZVSS	SARAM14 SLPZVDD	SARAM14 SLPZVSS	SARAM13 SLPZVDD	SARAM13 SLPZVSS	SARAM12 SLPZVDD	SARAM12 SLPZVSS
R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1
7	6	5	4	3	2	1	0
SARAM11 SLPZVDD	SARAM11 SLPZVSS	SARAM10 SLPZVDD	SARAM10 SLPZVSS	SARAM9 SLPZVDD	SARAM9 SLPZVSS	SARAM8 SLPZVDD	SARAM8 SLPZVSS
R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 1-25. RAM Sleep Mode Control Register4 [0x1C2C]

15	14	13	12	11	10	9	8
SARAM23 SLPZVDD	SARAM23 SLPZVSS	SARAM22 SLPZVDD	SARAM22 SLPZVSS	SARAM21 SLPZVDD	SARAM21 SLPZVSS	SARAM20 SLPZVDD	SARAM20 SLPZVSS
R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1
7	6	5	4	3	2	1	0
SARAM19 SLPZVDD	SARAM19 SLPZVSS	SARAM18 SLPZVDD	SARAM18 SLPZVSS	SARAM17 SLPZVDD	SARAM17 SLPZVSS	SARAM16 SLPZVDD	SARAM16 SLPZVSS
R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 1-26. RAM Sleep Mode Control Register5 [0x1C2D]

15	14	13	12	11	10	9	8
SARAM31 SLPZVDD	SARAM31 SLPZVSS	SARAM30 SLPZVDD	SARAM30 SLPZVSS	SARAM29 SLPZVDD	SARAM29 SLPZVSS	SARAM28 SLPZVDD	SARAM28 SLPZVSS
R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1
7	6	5	4	3	2	1	0
SARAM27 SLPZVDD	SARAM27 SLPZVSS	SARAM26 SLPZVDD	SARAM26 SLPZVSS	SARAM25 SLPZVDD	SARAM25 SLPZVSS	SARAM24 SLPZVDD	SARAM24 SLPZVSS
R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1	R/W+1

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

1.5.5 Power Considerations

1.5.5.1 Power Considerations for TMS320C5535/34

The device provides several means of managing power consumption.

To minimize power consumption, the device divides its circuits into nine main isolated supply domains:

- LDOI (LDOs and Bandgap Power Supply)
- Analog POR, SAR, and PLL (V_{DDA_ANA} and V_{DDA_PLL}) [1]
- RTC Core (CV_{DDRTC})
- Digital Core (CV_{DD})
- USB Core (USB_ V_{DD1P3} and USB_V_{DDA1P3})
- USB PHY and USB PLL (USB_V_{DDOSC}, USB_V_{DDA3P3}, and USB_V_{DDPLL})
- RTC I/O (DV_{DDRTC})
- Rest of the I/O (DV_{DDIO})

1.5.5.1.1 LDO Configuration

The device includes three Low-Dropout Regulators (LDOs) which can be used to regulate the power supplies of the analog PLL and SAR ADC/Power Management (ANA_LDO), Digital Core (DSP_LDO), and USB Core (USB_LDO).

These LDOs are controlled by a combination of pin configuration and register settings. For more detailed information see the following sections.

1.5.5.1.1.1 LDO Inputs

The LDOI pins (B10, B14, C14) provide power to the internal Analog LDO, DSP LDO, USB LDO, the bandgap reference generator, and some I/O input pins, and can range from 1.8 V to 3.6 V. The bandgap provides accurate voltage and current references to the POR, LDOs, PLL, and SAR; therefore, for proper device operation, power **must** always be applied to the LDOI pins even if the LDO outputs are ***not*** used.

1.5.5.1.1.2 LDO Outputs

The ANA_LDOO pin (B9) is the output of the internal ANA_LDO and can provide regulated 1.3 V power of up to 4 mA. The ANA_LDOO pin is intended to be connected, on the board, to the V_{DDA_ANA} and V_{DDA_PLL} pins to provide a regulated 1.3 V to the 10-bit SAR ADC, Power Management Circuits, and System PLL. V_{DDA_ANA} and V_{DDA_PLL} may be powered by this LDO output, which is recommended, to take advantage of the device's power management techniques, or by an external power supply. The ANA_LDO cannot be disabled individually (see Section 1.5.5.1.1.3, *LDO Control*).

The DSP_LDOO pin (A13) is the output of the internal DSP_LDO and provides software-selectable regulated 1.3 V or regulated 1.05 V power of up to 250 mA. The DSP_LDOO pin is intended to be connected, on the board, to the CV_{DD} pins. In this configuration, the $\overline{\text{DSP\_LDO\_EN}}$ pin should be tied to the board V_{SS}, thus enabling the DSP_LDO. Optionally, the CV_{DD} pins may be powered by an external power supply; in this configuration the $\overline{\text{DSP\_LDO\_EN}}$ pin should be tied (high) to LDOI, disabling DSP_LDO. The $\overline{\text{DSP\_LDO\_EN}}$ also affects how reset is generated to the chip (for more details, see the $\overline{\text{DSP\_LDO\_EN}}$ pin description in , *Regulators and Power Management Terminal Functions*). When the DSP_LDO is disabled, its output pin is in a high-impedance state. Note: $\overline{\text{DSP\_LDO\_EN}}$ is ***not*** intended to be changed dynamically.

When DSP_LDO comes out of reset, it is enabled to 1.3 V for the bootloader to operate. For the 50-MHz devices, DSP_LDO must be programmed to 1.05 V to match the core voltage, CV_{DD}, for proper operation after reset.

[1] SAR applies to only TMS320C5535.

The USB_LDOO pin (D13) is the output of the internal USB_LDO and provides regulated 1.3 V, software-switchable (on/off) power of up to 25 mA. The USB_LDOO pin is intended to be connected, on the board, to the USB_V$_{DD1P3}$ and USB_V$_{DDA1P3}$ pins to provide power to portions of the USB. Optionally, the USB_V$_{DD1P3}$ and USB_V$_{DDA1P3}$ may be powered by an external power supply and the USB_LDO can be left disabled. When the USB_LDO is disabled, its output pin is in a high-impedance state. See Section 1.7.3.2 for LDO control programming.

1.5.5.1.1.3 LDO Control

All three LDOs can be simultaneously disabled via software by writing to either the BG_PD bit or the LDO_PD bit in the RTCPMGT register (see Figure 1-27). When the LDOs are disabled via this mechanism, the only way to re-enable them is by asserting the WAKEUP signal pin (which must also have been previously enabled to allow wakeup), *or* by a previously enabled and configured RTC alarm, *or* by cycling power to the CV$_{DDRTC}$ pin.

ANA_LDO: The ANA_LDO is only disabled by the BG_PD and the LDO_PD mechanism described above. Otherwise, it is always enabled.

DSP_LDO: The DSP_LDO can be statically disabled by the $\overline{\text{DSP\_LDO\_EN}}$ pin as described in Section 1.5.5.1.1.2, *LDO Outputs*. It can be also dynamically disabled via the BG_PD and the LDO_PD mechanism described above. The DSP_LDO can change its output voltage dynamically by software via the DSP_LDO_V bit in the LDOCNTL register (see Figure 1-28). The DSP_LDO output voltage is set to 1.3 V at reset.

For the 50-MHz devices, DSP_LDO must be programmed to 1.05 V to match the core voltage, CV$_{DD}$, for proper operation after reset.

USB_LDO: The USB_LDO can be independently and dynamically enabled or disabled by software via the USB_LDO_EN bit in the LDOCNTL register (see Figure 1-28). The USB _LDO is disabled at reset.

Table 1-33 shows the ON/OFF control of each LDO and its register control bit configurations.

15	8
Reserved	
R-0	

7	5	4	3	2	1	0
Reserved		WU_DOUT	WU_DIR	BG_PD	LDO_PD	RTCCLKOUTEN
R-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 1-27. RTC Power Management Register (RTCPMGT) [1930h]

Table 1-31. RTCPMGT Register Bit Descriptions

BIT	NAME	DESCRIPTION
15:5	RESERVED	Reserved. Read-only, writes have no effect.
4	WU_DOUT	Wakeup output, active low/open-drain. 0 = WAKEUP pin driven low. 1 = WAKEUP pin is in high-impedance (Hi-Z).
3	WU_DIR	Wakeup pin direction control. 0 = WAKEUP pin configured as a input. 1 = WAKEUP pin configured as a output. Note: When the WAKEUP pin is configured as an input, it is active high. When the WAKEUP pin is configured as an output, is an open-drain that is active low and should be externally pulled-up via a 10-kΩ resistor to DV_{DDRTC}. WU_DIR must be configured as an input to allow the WAKEUP pin to wake the device up from idle modes.
2	BG_PD	Bandgap, on-chip LDOs, and the analog POR power down bit. This bit shuts down the on-chip LDOs (ANA_LDO, DSP_LDO, and USB_LDO), the Analog POR, and Bandgap reference. BG_PD and LDO_PD are only intended to be used when the internal LDOs supply power to the chip. If the internal LDOs are bypassed and not used then the BG_PD and LDO_PD power down mechanisms should not be used since POR gets powered down and the POWERGOOD signal is not generated properly. After this bit is asserted, the on-chip LDOs, Analog POR, and the Bandgap reference can be re-enabled by the WAKEUP pin (high) or the RTC alarm interrupt. The Bandgap circuit will take about 100 msec to charge the external 0.1 uF capacitor via the internal 326-kΩ resistor. 0 = On-chip LDOs, Analog POR, and Bandgap reference are enabled. 1 = On-chip LDOs, Analog POR, and Bandgap reference are disabled (shutdown).
1	LDO_PD	On-chip LDOs and Analog POR power down bit. This bit shuts down the on-chip LDOs (ANA_LDO, DSP_LDO, and USB_LDO) and the Analog POR. BG_PD and LDO_PD are only intended to be used when the internal LDOs supply power to the chip. If the internal LDOs are bypassed and not used then the BG_PD and LDO_PD power down mechanisms should not be used since POR gets powered down and the POWERGOOD signal is not generated properly. After this bit is asserted, the on-chip LDOs and Analog POR can be re-enabled by the WAKEUP pin (high) or the RTC alarm interrupt. This bit keeps the Bandgap reference turned on to allow a faster wake-up time with the expense power consumption of the Bandgap reference. 0 = On-chip LDOs and Analog POR are enabled. 1 = On-chip LDOs and Analog POR are disabled (shutdown).
0	RTCCLKOUTEN	Clockout output enable bit. 0 = Clock output disabled. 1 = Clock output enabled.

15	8
Reserved	
R-0	

7	2	1	0
Reserved		DSP_LDO_V	USB_LDO_EN
R-0		R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 1-28. LDO Control Register (LDOCNTL) [7004h]

Table 1-32. LDOCNTL Register Bit Descriptions

BIT	NAME	DESCRIPTION
15:2	RESERVED	Reserved. Read-only, writes have no effect.
1	DSP_LDO_V	DSP_LDO voltage select bit. 0 = DSP_LDOO is regulated to 1.3 V. 1 = DSP_LDOO is regulated to 1.05 V Note: For the 50-MHz devices, DSP_LDO must be programmed to 1.05 V to match the core voltage, CV_{DD}, for proper operation after reset.
0	USB_LDO_EN	USB_LDO enable bit. 0 = USB_LDO output is disabled. USB_LDOO pin is placed in high-impedance (Hi-Z) state. 1 = USB_LDO output is enabled. USB_LDOO is regulated to 1.3 V.

Table 1-33. LDO Controls Matrix

RTCPMGT Register (0x1930)		LDOCNTL Register (0x7004)	$\overline{\text{DSP\_LDO\_EN}}$ (Pin C13)	ANA_LDO	DSP_LDO	USB_LDO
BG_PD Bit	LDO_PD Bit	USB_LDO_EN Bit				
1	Don't Care	Don't Care	Don't Care	OFF	OFF	OFF
Don't Care	1	Don't Care	Don't Care	OFF	OFF	OFF
0	0	0	Low	ON	ON	OFF
0	0	0	High	ON	OFF	OFF
0	0	1	Low	ON	ON	ON

1.5.5.2 Power Considerations for TMS320C5533

The device provides several means of managing power consumption.

To minimize power consumption, the device divides its circuits into nine main isolated supply domains:

- LDOI (LDOs and Bandgap Power Supply)
- Analog POR and PLL (V_{DDA_ANA} and V_{DDA_PLL})
- RTC Core (CV_{DDRTC})
- Digital Core (CV_{DD})
- USB Core (USB_ V_{DD1P3} and USB_V_{DDA1P3})
- USB PHY and USB PLL (USB_V_{DDOSC}, USB_V_{DDA3P3}, and USB_V_{DDPLL})
- RTC I/O (DV_{DDRTC})
- Rest of the I/O (DV_{DDIO})

1.5.5.2.1 LDO Configuration

The device includes two Low-Dropout Regulators (LDOs) which can be used to regulate the power supplies of the analog PLL and Power Management (ANA_LDO) and USB Core (USB_LDO).

These LDOs are controlled by a combination of pin configuration and register settings. For more detailed information see the following sections.

1.5.5.2.1.1 LDO Inputs

The LDOI pins (B10, B14, C14) provide power to the internal Analog and USB LDOs, the bandgap reference generator, and some I/O input pins, and can range from 1.8 V to 3.6 V. The bandgap provides accurate voltage and current references to the POR, LDOs, and PLL; therefore, for proper device operation, power **must** always be applied to the LDOI pins even if the LDO outputs are ***not*** used.

1.5.5.2.1.2 LDO Outputs

The ANA_LDOO pin (B9) is the output of the internal ANA_LDO and can provide regulated 1.3 V power of up to 4 mA. The ANA_LDOO pin is intended to be connected, on the board, to the V_{DDA_ANA} and V_{DDA_PLL} pins to provide a regulated 1.3 V to the Power Management Circuits and System PLL. V_{DDA_ANA} and V_{DDA_PLL} may be powered by this LDO output, which is recommended, to take advantage of the device's power management techniques, or by an external power supply. The ANA_LDO cannot be disabled individually (see Section 1.5.5.1.1.3, *LDO Control*).

The USB_LDOO pin (D13) is the output of the internal USB_LDO and provides regulated 1.3 V, software-switchable (on/off) power of up to 25 mA. The USB_LDOO pin is intended to be connected, on the board, to the USB_V_{DD1P3} and USB_V_{DDA1P3} pins to provide power to portions of the USB. Optionally, the USB_V_{DD1P3} and USB_V_{DDA1P3} may be powered by an external power supply and the USB_LDO can be left disabled. When the USB_LDO is disabled, its output pin is in a high-impedance state. See Section 1.7.3.3 for LDO control programming.

1.5.5.3 Power Considerations for TMS320C5532

The device provides several means of managing power consumption.

To minimize power consumption, the device divides its circuits into nine main isolated supply domains:

- LDOI (ANA_LDO and Bandgap Power Supply)
- Analog POR and PLL (V_{DDA_ANA} and V_{DDA_PLL})
- RTC Core (CV_{DDRTC})
- Digital Core (CV_{DD})
- USB Core (USB_V_{DD1P3} and USB_V_{DDA1P3}) — C5533 Only
- USB PHY and USB PLL (USB_V_{DDOSC}, USB_V_{DDA3P3}, and USB_V_{DDPLL}) — C5533 Only
- RTC I/O (DV_{DDRTC})
- Rest of the I/O (DV_{DDIO})

1.5.5.3.1 LDO Configuration

The device includes one Low-Dropout Regulators (LDO) which can be used to regulate the power supplies of the analog PLL.

1.5.5.3.2 LDO Inputs

The LDOI pins (B10, B14, C14) provide power to the internal Analog LDO, the bandgap reference generator, and some I/O input pins, and can range from 1.8 V to 3.6 V. The bandgap provides accurate voltage and current references to the LDO PLL; therefore, for proper device operation, power must always be applied to the LDOI pins even if the LDO output is not used.

1.5.5.3.3 LDO Outputs

The ANA_LDOO pin (B9) is the output of the internal ANA_LDO and can provide regulated 1.3 V power of up to 4 mA. The ANA_LDOO pin is intended to be connected, on the board, to the VDDA_ANA and VDDA_PLL pins to provide a regulated 1.3 V to the System PLL. VDDA_ANA and VDDA_PLL may be powered by this LDO output. However, when VDDA_PLL requires 1.4 V, VDDA_PLL must be powered externally and ANA_LDO output can provide a regulated 1.3 V, but only to VDDA_ANA, not both.

NOTE: The DSP_LDOO is not supported on this device. However, DSP_LDO can be enabled to support the RTC-only mode (for details, see the *RTC Only Mode* section in the device-specific data manual). Otherwise, DSP_LDO should be disabled on this device and the DSP_LDO output pin must be always left unconnected. The USB_LDOO is **not** supported on this device, so the USB_LDO must be left disabled. USB_LDO is disabled at reset, so it does not require any action to disable the USB_LDO. When the USB_LDO is disabled, the USB_LDOO pin is in a high-impedance (Hi-Z) state and should be left unconnected.

1.5.6 Power Configurations

The power-saving features described in the previous sections, such as peripheral clock gating, and on-chip memory power down to name a few, can be combined to form a power configuration. Many different power configurations can be created by enabling and disabling different power domains and clock domains, however, this section defines some basic power configurations that may be useful. These are shown and described in Table 1-34. Please note that there is no single instruction or register that can place the device in these power configurations. Instead, these power configurations are achieved by modifying multiple registers.

NOTE: Before you change the power configuration, make sure that there is a method for the device to exit the power configuration. After exiting a power configuration, your software may have to take additional steps to change the clock and power configuration for other domains.

NOTE: The on-chip Bootloader idles all peripherals and CPU ports at startup. It enables some peripherals as it uses them. Your application code should check the idle configuration of peripherals and CPU ports before using them to be sure these are not idle.

Table 1-34. Power Configurations

Power Configuration	Power Domain State	Clock Domain State	Steps to Enter Clock and Power Configuration	Available Methods for Changing/Exiting Clock and Power Configuration
RTC only mode	DV_{DDRTC}, LDOI, and CV_{DDRTC} powered all others powered-down	Only RTC clock is running	Set LDO_PD and BG_PD bits in RTCPMGT register	A. RTC interrupt B. WAKEUP pin
IDLE3	All power domains on	RTC clock domain enabled	Idle peripheral domain	A. WAKEUP pin
		Other clock domains disabled. Clock generator domain disabled (BYPASS MODE and PLL powerdown).	Idle CPU domain	B. RTC interrupt
			PLL in BYPASS MODE PLL powerdown	C. External hardware interrupt (INT0 or INT1).
			Master clock disable	D. Hardware Reset
			Execute idle instruction	
IDLE2	All power domains on	RTC clock domain enabled	Idle peripheral domains	A. WAKEUP pin
		Clock generator domain enabled (PLL_MODE)	Idle CPU domain	B. RTC interrupt
		Other clock domains disabled	Execute idle instruction	C. External hardware interrupt (INT0, INT1).
				D. Any unmasked peripheral interrupt.
				E. Hardware Reset
Active	All power domains on	All clock domains enabled	Turn on all power domains	
			Enable all clock domains	

1.5.6.1 IDLE2 Procedure

In this power configuration all the power domains are turned on, the RTC and clock generator domains are enabled, the CPU domain is disabled, and the DSP peripherals are disabled. When you enter this power configuration all CPU and peripheral activity in the DSP is stopped. Leaving the clock generator domain enabled allows the DSP to quickly exit this power configuration since there is no need to wait for power domains to turn on or for the PLL to re-lock.

Follow these steps to enter the IDLE2 power configuration:

1. Wait for completion of all DMA transfers. You can poll the DMA transfer status and disable DMA transfers through the DMA registers.
2. Disable the USB clock domain as described in Section 1.5.3.4.
3. Idle all the desired peripherals in the peripheral clock domain by modifying the peripheral clock gating configuration registers (PCGCR1 and PCGCR2). See Section 1.5.3.2 for more details on setting the DSP peripherals to idle mode.
4. Clear all interrupts by writing ones to the CPU interrupt flag registers (IFR0 and IFR1).
5. Enable the appropriate wake-up interrupt in the CPU interrupt enable registers (IER0 and IER1). If using the WAKEUP pin to exit this mode, configure the WAKEUP pin as input by setting WU_DIR = 1 in the RTC power management register (RTCPMGT). If using the RTC alarm or periodic interrupt as a wake-up event, the RTCINTEN bit must be set in the RTC interrupt enable register (RTCINTEN).
6. Disable the CPU domain by setting to 1 the CPUI, MPORTI, XPORTI, DPORTI, IPORTI, and CPI bits of the idle configuration register (ICR). Note that the MPORT will not go into idle mode if the USB CDMA or DMA controllers is not idled.
7. Apply the new idle configuration by executing the “IDLE” instruction. The content of ICR is copied to the idle status register (ISTR). The bits of ISTR are then propagated through the CPU domain system to enable or disable the specified clocks.

The IDLE instruction cannot be executed in parallel with another instruction.

To exit the IDLE2 power configuration, follow these steps:

1. Generate the wake-up interrupt you specified during the IDLE2 power down procedure.
2. After the interrupt is generated, the DSP will execute the interrupt service routine.
3. After exiting the interrupt service routine, code execution will resume from the point where the “IDLE” instruction was originally executed.

You can also exit the IDLE2 power configuration by generating a hardware reset. However, in this case, the DSP is completely reset and the state of the DSP before going into IDLE2 is lost.

1.5.6.2 IDLE3 Procedure

In this power configuration all the power domains are turned on, the CPU and clock generator domains are disabled, and the RTC clock domain is enabled. The DSP peripherals and the USB are also disabled in this mode. When you enter this power configuration, all CPU and peripheral activity in the DSP is stopped.

Since the clock generator domain is disabled, you must allow enough time for the PLL to re-lock before exiting this power configuration.

Follow these steps to enter the IDLE3 power configuration:

1. Wait for completion of all DMA transfers. You can poll the DMA transfer status and disable DMA transfers through the DMA registers.
2. Disable the USB clock domain as described in Section 1.5.3.4.
3. Idle all the desired peripherals in the peripheral clock domain by modifying the peripheral clock gating configuration registers (PCGCR1 and PCGCR2). See Section 1.5.3.2 for more details on setting the DSP peripherals to idle mode.
4. Disable the clock generator domain as described in Section 1.5.3.3.
5. Clear all interrupts by writing ones to the CPU interrupt flag registers (IFR0 and IFR1).
6. Enable the appropriate wake-up interrupt in the CPU interrupt enable registers (IER0 and IER1). If using the WAKEUP pin to exit this mode, configure the WAKEUP pin as input by setting WU_DIR = 1 in the RTC power management register (RTCPMGT). If using the RTC alarm or periodic interrupt as a wake-up event, the RTCINTEN bit must be set in the RTC interrupt enable register (RTCINTEN).
7. Disable the CPU domain by setting to 1 the CPUI, MPORTI, XPORTI, DPORTI, IPORTI, and CPI bits of the idle configuration register (ICR).
8. Apply the new idle configuration by executing the IDLE instruction. The content of ICR is copied to the idle status register (ISTR). The bits of ISTR are then propagated through the CPU domain system to enable or disable the specified clocks.

The IDLE instruction cannot be executed in parallel with another instruction.

To exit the IDLE3 power configuration, follow these steps:

1. Generate the wake-up interrupt you specified during the IDLE3 power down procedure.
2. After the interrupt is generated, the DSP will execute the interrupt service routine.
3. After exiting the interrupt service routine, code execution will resume from the point where the “IDLE” instruction was originally executed.
4. Enable the clock generator domain as described in Section 1.5.3.3. You can also enable the clock generator domain inside the interrupt service routine.

You can also exit the IDLE3 power configuration by generating a hardware reset, however, in this case the DSP is completely reset and the state of the DSP before going into IDLE3 is lost.

1.5.6.3 Core Voltage Scaling

When the core voltage domain (CV_{DD}) is ON, it can be set to two voltages: 1.3 V or 1.05 V (nominal). The core voltage can be reduced during periods of low processing demand and increased during high demand. Core voltage scaling can be accomplished with an external power management IC (LDO, DC-DC, etc). When the core voltage is decreased (1.3 V to 1.05 V), care must be taken to ensure device stability. The following rules must be followed to maintain stability:

- When using an external PMIC (power management IC), the board designer must ensure that the 1.3 V to 1.05 V transition does not have ringing that would violate our V_{DDC} minimum rating (1.05 V - 5% = 0.998 V).
- Software must ensure that the clock speed of the device does not exceed the maximum speed of the device at the lower voltage before making the voltage transition. For example, if the device is running at 100 MHz @ 1.3 V, then the PLL must be changed to 50 MHz before changing the core voltage to 1.05 V.

When the core voltage is increased (1.05 V to 1.3 V) clock speed is not an issue since the device can operate faster at the higher voltage. However, when switching from 1.05 V to 1.3 V software must allow time for the voltage transition to reach the 1.3 V range. Additionally, external regulators might produce an overshoot that must not pass the maximum operational voltage of the core supply (see the *Recommended Operating Conditions* section in device-specific data manual). Otherwise, the device will be operating out of specification. This could happen if large current draw occurs while the regulator transitions to the higher voltage.

For external PMICs, the step response varies greatly and it is up to the system designer to ensure that the ringing is maintained within the DSP's core supply high voltage operational tolerance (see the *Recommended Operating Conditions* section in device-specific data manual).

1.6 Interrupts

Vector-relative locations and priorities for all internal and external interrupts are shown in Table 1-35.

Table 1-35. Interrupt Table

NAME	SOFTWARE (TRAP) EQUIVALENT	RELATIVE LOCATION (HEX BYTES) [1]	PRIORITY	FUNCTION
RESET	SINT0	0x0	0	Reset (hardware and software)
NMI[2]	SINT1	0x8	1	Non-maskable interrupt
INT0	SINT2	0x10	3	External user interrupt #0
INT1	SINT3	0x18	5	External user interrupt #1
TINT	SINT4	0x20	6	Timer aggregated interrupt
PROG0	SINT5	0x28	7	Programmable transmit interrupt 0 (I2S0 transmit or MMC/SD0 interrupt)
UART	SINT6	0x30	9	UART interrupt
PROG1	SINT7	0x38	10	Programmable receive interrupt 1 (I2S0 receive or MMC/SD0 SDIO interrupt)
DMA	SINT8	0x40	11	DMA aggregated interrupt
PROG2	SINT9	0x48	13	Programmable transmit interrupt 1 (I2S1 transmit or MMC/SD1 interrupt)
-	SINT10	0x50	14	Software interrupt
PROG3	SINT11	0x58	15	Programmable receive interrupt 3 (I2S1 Receive or MMC/SD1 SDIO interrupt)
LCD	SINT12	0x60	17	LCD interrupt.
SAR	SINT13	0x68	18	10-bit SAR A/D conversion or pin interrupt
XMT2	SINT14	0x70	21	I2S2 transmit interrupt
RCV2	SINT15	0x78	22	I2S2 receive interrupt
XMT3	SINT16	0x80	4	I2S3 transmit interrupt
RCV3	SINT17	0x88	8	I2S3 receive interrupt
RTC	SINT18	0x90	12	Wakeup or real-time clock interrupt
SPI	SINT19	0x98	16	SPI interrupt
USB	SINT20	0xA0	19	USB Interrupt
GPIO	SINT21	0xA8	20	GPIO aggregated interrupt
I2C	SINT23	0xB8	24	I2C interrupt
BERR	SINT24	0xC0	2	Bus error interrupt
DLOG	SINT25	0xC8	25	Data log interrupt
RTOS	SINT26	0xD0	26	Real-time operating system interrupt
-	SINT27	0xD8	14	Software interrupt #27
-	SINT28	0xE0	15	Software interrupt #28
-	SINT29	0xE8	16	Software interrupt #29
-	SINT30	0xF0	17	Software interrupt #30
-	SINT31	0xF8	18	Software interrupt #31

[1] Absolute addresses of the interrupt vector locations are determined by the contents of the IVPD and IVPH registers. Interrupt vectors for interrupts 0-15 and 24-31 are relative to IVPD. Interrupt vectors for interrupts 16-23 are relative to IVPH.

[2] The NMI signal is internally tied high (not asserted). However, NMI interrupt vector can be used for SINT1.

1.6.1 IFR and IER Registers

The interrupt flag register 0 (IFR0) and interrupt enable register 0 (IER0) bit layouts are shown in Figure 1-29 and described in Table 1-36.

Figure 1-29. IFR0 and IER0 Bit Locations

15	14	13	12	11	10	9	8
RCV2	XMT2	SAR	LCD	PROG3	Reserved	PROG2	DMA
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
PROG1	UART	PROG0	TINT	INT1	INT0	Reserved	
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-36. IFR0 and IER0 Bit Descriptions

Bit	Field	Value	Description
15	RCV2	1-0	I2S2 receive interrupt flag/mask bit.
14	XMT2	1-0	I2S2 transmit interrupt flag/mask bit.
13	SAR	1-0	10-BIT SAR A/D conversion or pin interrupt flag/mask bit.
12	LCD	1-0	LCD interrupt bit.
11	PROG3	1-0	Programmable receive interrupt 3 flag/mask bit. This bit is used as either the I2S1 receive interrupt flag/mask bit or the MMC/SD1 SDIO interrupt flag/mask bit. The function of this bit is selected depending on the setting of the SP1MODE bit is in external bus selection register. If SP1MODE = 00b, this bit supports MMC/SD1 SDIO interrupts. If SP1MODE = 01, this bit supports I2S1 interrupts.
10	Reserved	0	Reserved. This bit should always be written with 0.
9	PROG2	1-0	Programmable transmit interrupt 2 flag/mask bit. This bit is used as either the I2S1 transmit interrupt flag/mask bit or the MMC/SD1 interrupt flag/mask bit. The function of this bit is selected depending on the setting of the SP1MODE bit in the external bus selection register. If SP1MODE = 00b, this bit supports MMC/SD1 interrupts. If SP1MODE = 01, this bit supports I2S1 interrupts.
8	DMA	1-0	DMA aggregated interrupt flag/mask bit
7	PROG1	1-0	Programmable receive interrupt 1 flag/mask bit. This bit is used as either the I2S0 receive interrupt flag/mask bit or the MMC/SD0 SDIO interrupt flag/mask bit. The function of this bit is selected depending on the setting of the SP0MODE bit in the external bus selection register. If SP0MODE = 00b, this bit supports MMC/SD0 SDIO interrupts. If SP0MODE = 01, this bit supports I2S0 interrupts.
6	UART	1-0	UART interrupt flag/mask bit
5	PROG0	1-0	Programmable transmit interrupt 0 flag/mask bit. This bit is used as either the I2S0 transmit interrupt flag/mask bit or the MMC/SD0 interrupt flag/mask bit. The function of this bit is selected depending on the setting of the SP0MODE bit in the external bus selection register. If SP0MODE = 00b, this bit supports MMC/SD0 interrupts. If SP0MODE = 01, this bit supports I2S0 interrupts.
4	TINT	1-0	Timer aggregated interrupt flag/mask bit.
3	INT1	1-0	External user interrupt #1 flag/mask bit.
2	INT0	1-0	External user interrupt #0 flag/mask bit.
1-0	Reserved	0	Reserved. This bit should always be written with 0.

The interrupt flag register (IFR1) and interrupt enable register 1 (IER1) bit layouts are shown in Figure 1-30 and described in Table 1-37.

Figure 1-30. IFR1 and IER1 Bit Locations

15	11	10	9	8
Reserved		RTOS	DLOG	BERR
R-0		R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
I2C	Reserved	GPIO	USB	SPI	RTC	RCV3	XMT3
R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-37. IFR1 and IER1 Bit Descriptions

Bit	Field	Value	Description
15-11	Reserved	0	Reserved. This bit should always be written with 0.
10	RTOS	1-0	Real-Time operating system interrupt flag/mask bit.
9	DLOG	1-0	Data log interrupt flag/mask bit.
8	BERR	1-0	Bus error interrupt flag/mask bit.
7	I2C	1-0	I2C interrupt flag/mask bit.
6	Reserved	0	Reserved.
5	GPIO	1-0	GPIO aggregated interrupt flag/mask bit.
4	USB	1-0	USB interrupt flag/mask bit.
3	SPI	1-0	SPI interrupt flag/mask bit.
2	RTC	1-0	Wakeup or real-time clock interrupt flag/mask bit.
1	RCV3	1-0	I2S3 receive interrupt flag/mask bit.
0	XMT3	1-0	I2S3 transmit interrupt flag/mask bit.

1.6.2 Interrupt Timing

The interrupt signals on the external interrupts pins ($\overline{\text{INT0}}$ and $\overline{\text{INT1}}$) are detected with a synchronous negative edge detector circuit. To reliably detect the external interrupts, the interrupt signal must have at least 2 SYSCLK high followed by at least 2 SYSCLK low.

To define the minimum low pulse width in nanoseconds scale, you should take into account that the on-chip PLL of the device is software programmable and that your application may be dynamically changing the frequency of PLL. You should use the slowest frequency that will be used by your application to calculate the minimum interrupt pulse duration in nanoseconds.

When the system master clock is disabled (SYSCLKDIS =1), the external interrupt pins ($\overline{\text{INT0}}$ and $\overline{\text{INT1}}$) will be asynchronously latched and held low while the clocks are re-enabled. Once the clocks are re-enabled, the DSP will latch the interrupt in the IFR.

1.6.3 Timer Interrupt Aggregation Flag Register (TIAFR) [1C14h]

The CPU has only one interrupt flag that is shared among the three timers. The CPU's interrupt flag is bit 4 (TINT) of the IFR0 & IER0 registers (see Figure 1-25). Since the interrupt flag is shared, software must have a means of determining which timer instance caused the interrupt. Therefore, the timer interrupt aggregation flag register (TIAFR) is a secondary flag register that serves this purpose.

The timer interrupt aggregation flag register (TIAFR) latches each timer (Timer 0, Timer 1, and Timer 2) interrupt signal when the timer counter expires. Using this register, the programmer can determine which timer generated the timer aggregated CPU interrupt signal (TINT). Each Timer flag in TIAFR needs to be cleared by the CPU with a write of 1. Note that the IFR0[TINT] bit is automatically cleared when entering the interrupt service routine (ISR). Therefore there is no need to manually clear it in the ISR. If two (or more) timers happen to interrupt simultaneously, the TIAFR register will indicate the two (or more) interrupt flags. In this case, the ISR can choose to service both timer interrupts or only one-at-a-time. If the ISR services only one of them, then it should clear only one of the TIAFR flags and upon exiting the ISR, the CPU will immediately be interrupted again to service the second timer flag. If the ISR services all of them, then it should clear all of them in the TIAFR flags and upon exiting the ISR, the CPU won't be interrupted again until a new timer interrupt comes in. For more information, see Section 5.1.

1.6.4 GPIO Interrupt Enable and Aggregation Flag Registers

The CPU has only one interrupt flag that is shared among all GPIO pin interrupt signals. The CPU's interrupt flag is bit 5 (GPIO) of the IFR1 & IER1 registers (see Figure 1-26). Since the interrupt flag is shared, software must have a means of determining which GPIO pin caused the interrupt. Therefore, the GPIO interrupt aggregation flag registers (IOINTFLG1 and IOINTFLG2) are secondary flag registers that serve this purpose.

If any of the GPIO pins are configured as inputs, they can be enabled to accept external signals as interrupts using the GPIO Interrupt Enable Registers (IOINTEN1 and IOINTEN2). The GPIO Interrupt Flag Registers (IOINTFLG1 and IOINTFLG2) can be used to determine which of the 32 GPIO pins triggered the interrupt. Note that the IFR0[GPIO] bit is automatically cleared when entering the interrupt service routine (ISR). Therefore, there is no need to manually clear it in the ISR. If two (or more) GPIO pins happen to interrupt simultaneously, the IOINTFLG1/IOINTFLG2 register indicates the two (or more) interrupt flags. In this case, the ISR can choose to service both/all GPIO interrupts or only one-at-a-time. If the ISR services only one of them, then it should clear only one of the IOINTFLG1/IOINTFLG2 flags and upon exiting the ISR, the CPU is immediately interrupted again to service the others. For more information, see Chapter 12.

1.6.5 DMA Interrupt Enable and Aggregation Flag Registers

The CPU has only one interrupt flag that is shared among the 16 DMA interrupt sources. The CPU's interrupt flag is bit 8 (DMA) of the IFR0 & IER0 registers (see Figure 1-25). Since the interrupt flag is shared, software must have a means of determining which DMA instance caused the interrupt. Therefore, the DMA interrupt aggregation flag registers (DMAIFR) are secondary flag registers that serve this purpose.

Each of the four channels of a DMA controller has its own interrupt, which you can enable or disable a channel interrupt though the DMA*n*CH*m* bits of the DMA Interrupt Enable Register (DMAIER) (see Section 1.7.4.2.1). The interrupts from the four DMA controllers are combined into a single CPU interrupt. You can determine which DMA channel generated the interrupt by reading the bits of the DMA interrupt flag register (DMAIFR). For more information, see Chapter 12.

1.7 System Configuration and Control

1.7.1 Overview

The DSP includes system-level registers for controlling, configuring, and reading status of the device. These registers are accessible by the CPU and support the following features:

- Device Identification
- Device Configuration
  - Pin multiplexing control
  - Output drive strength configuration
  - Internal pull-up and pull-down enable/disable
  - On-chip LDO control
- DMA Controller Configuration
- Peripheral Reset
- USB Byte Access

1.7.2 Device Identification

The DSP includes a set of device ID registers that are intended for use in TI chip manufacturing, but can be used by users as a 128-bit unique ID for each device. These registers are summarized in the following table.

Table 1-38. Die ID Registers

CPU Word Address	Acronym	Register Description	Section
1C40h	DIEIDR0	Die ID Register 0	Section 1.7.2.1
1C41h	DIEIDR1	Die ID Register 1	Section 1.7.2.2
1C42h	DIEIDR2	Die ID Register 2	Section 1.7.2.3
1C43h	DIEIDR3	Die ID Register 3	Section 1.7.2.4
1C44h	DIEIDR4	Die ID Register 4	Section 1.7.2.5
1C45h	DIEIDR5	Die ID Register 5	Section 1.7.2.6
1C46h	DIEIDR6	Die ID Register 6	Section 1.7.2.7
1C47h	DIEIDR7	Die ID Register 7	Section 1.7.2.8

1.7.2.1 Die ID Register 0 (DIEIDR0) [1C40h]

The die ID register 0 (DIEIDR0) is shown in Figure 1-31 and described in Table 1-39.

Figure 1-31. Die ID Register 0 (DIEIDR0) [1C40h]

15	0
DIEID0	
R	

LEGEND: R = Read only; -*n* = value after reset

Table 1-39. Die ID Register 0 (DIEIDR0) Field Descriptions

Bit	Field	Value	Description
15-0	DIEID0	0-FFFFh	Die ID bits.

1.7.2.2 Die ID Register 1 (DIEIDR1) [1C41h]

The die ID register 1 (DIEIDR1) is shown in Figure 1-32 and described in Table 1-40.

Figure 1-32. Die ID Register 1 (DIEIDR1) [1C41h]

15	14	13	0
Reserved		DIEID1	
R		R	

LEGEND: R = Read only; -*n* = value after reset

Table 1-40. Die ID Register 1 (DIEIDR1) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-0	DIEID1	0-3FFFh	Die ID bits.

1.7.2.3 Die ID Register 2 (DIEIDR2) [1C42h]

The die ID register 2 (DIEIDR2) is shown in Figure 1-33 and described in Table 1-41.

Figure 1-33. Die ID Register 2 (DIEIDR2) [1C42h]

15	0
DIEID2	
R	

LEGEND: R = Read only; -*n* = value after reset

Table 1-41. Die ID Register 2 (DIEIDR2) Field Descriptions

Bit	Field	Value	Description
15-0	DIEID2	0-FFFFh	Die ID bits.

1.7.2.4 Die ID Register 3 (DIEIDR3[15:0]) [1C43h]

The die ID register 3 (DIEIDR3) is shown in Figure 1-34 and described in Table 1-42.

Figure 1-34. Die ID Register 3 (DIEIDR3[15:0]) [1C43h]

15 12	11 0
DesignRev	DIEID3
R	R

LEGEND: R = Read only; -*n* = value after reset

Table 1-42. Die ID Register 3 (DIEIDR3[15:0]) Field Descriptions

Bit	Field	Value	Description
15-12	DesignRev	0-Fh	Silicon Revision
		0	Silicon 2.2
11-0	DIEID3	0-FFFFh	Die ID bits.

1.7.2.5 Die ID Register 4 (DIEIDR4) [1C44h]

The die ID register 4 (DIEIDR4) is shown in Figure 1-35 and described in Table 1-43.

Figure 1-35. Die ID Register 4 (DIEIDR4) [1C44h]

15 6	5 0
Reserved	DIEID4
R	R

LEGEND: R = Read only; -*n* = value after reset

Table 1-43. Die ID Register 4 (DIEIDR4) Field Descriptions

Bit	Field	Value	Description
15-6	Reserved	0	Reserved.
5-0	DIEID4	0-3Fh	Die ID bits.

1.7.2.6 Die ID Register 5 (DIEIDR5) [1C45h]

The die ID register 5 (DIEIDR5) is shown in Figure 1-36 and described in Table 1-44.

Figure 1-36. Die ID Register 5 (DIEIDR5) [1C45h]

15 0
Reserved
R

LEGEND: R = Read only; -*n* = value after reset

Table 1-44. Die ID Register 5 (DIEIDR5) Field Descriptions

Bit	Field	Value	Description
15-0	Reserved	0	Reserved.

1.7.2.7 Die ID Register 6 (DIEIDR6) [1C46h]

The die ID register 6 (DIEIDR6) is shown in Figure 1-37 and described in Table 1-45.

Figure 1-37. Die ID Register 6 (DIEIDR6) [1C46h]

15   0
Reserved
R

LEGEND: R = Read only; -*n* = value after reset

Table 1-45. Die ID Register 6 (DIEIDR6) Field Descriptions

Bit	Field	Value	Description
15-0	Reserved	0	Reserved.

1.7.2.8 Die ID Register 7 (DIEIDR7) [1C47h]

The die ID register 7 (DIEIDR7) is shown in Figure 1-38 and described in Table 1-46.

Figure 1-38. Die ID Register 7 (DIEIDR7) [1C47h]

15	14 1	0
Reserved	CHECKSUM	Reserved
R	R	R

LEGEND: R = Read only; -*n* = value after reset

Table 1-46. Die ID Register 7 (DIEIDR7) Field Descriptions

Bit	Field	Value	Description
15	Reserved	0	Reserved.
14-1	CHECKSUM	0-3FFFh	Checksum bits.
0	Reserved	0	Reserved.

1.7.3 Device Configuration

The DSP includes registers for configuring pin multiplexing, the pin output slew rate, the internal pull-ups and pull-downs.

1.7.3.1 External Bus Selection Register (EBSR)

The external bus selection register (EBSR) determines the mapping of the I2S2, I2S3, UART, SPI, and GPIO signals to 21 signals of the external parallel port pins. It also determines the mapping of the I2S or eMMC/SD/SDHC ports to serial port 1 pins and serial port 2 pins. The EBSR register is located at port address 0x1C00. Once the bit fields of this register are changed, the routing of the signals takes place on the next CPU clock cycle.

Before modifying the values of the external bus selection register, you must clock gate all affected peripherals through the Peripheral Clock Gating Control Register (for more information on clock gating peripherals, see Section 1.5.3.2). After the external bus selection register has been modified, you must reset the peripherals before using them through the Peripheral Software Reset Counter Register.

After the boot process is complete, the external bus selection register must be modified only once, during device configuration. Continuously switching the EBSR configuration is not supported.

The external bus selection register (EBSR) is shown in Figure 1-39 and described in Table 1-47.

Figure 1-39. External Bus Selection Register (EBSR) [1C00h]

15	14		12	11	10	9	8
Reserved	PPMODE			SP1MODE		SP0MODE	
R-0	R/W-000			R/W-00		R/W-00	

7	6	5	4	3	2	1	0
Reserved							
R-0							

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-47. EBSR Register Bit Descriptions Field Descriptions

Bit	Field	Value	Description
15	Reserved	0	Reserved. Read-only, writes have no effect.
14-12	PPMODE		Parallel Port Mode Control Bits. These bits control the pin multiplexing of the LCD Controller, SPI, UART, I2S2, I2S3, and GP[31:27, 20:18] pins on the parallel port.
		000	Mode 0 (16-bit LCD Controller). All 21 signals of the LCD Bridge module are routed to the 21 external signals of the parallel port.
		001	Mode 1 (SPI, GPIO, UART, and I2S2). 7 signals of the SPI module, 6 GPIO signals, 4 signals of the UART module and 4 signals of the I2S2 module are routed to the 21 external signals of the parallel port.
		010	Mode 2 (8-bit LCD Controller and GPIO). 8-bits of pixel data of the LCD Controller module and 8 GPIO are routed to the 21 external signals of the parallel port.
		011	Mode 3 (8-bit LCD Controller, SPI, and I2S3). 8-bits of pixel data of the LCD Controller module, 4 signals of the SPI module and 4 signals of the I2S3 module are routed to the 21 external signals of the parallel port.
		100	Mode 4 (8-bit LCD Controller, I2S2, and UART). 8-bits of pixel data of the LCD Controller module, 4 signals of the I2S2 module and 4 signals of the UART module are routed to the 21 external signals of the parallel port.
		101	Mode 5 (8-bit LCD Controller, SPI, and UART). 8-bits of pixel data of the LCD Controller module, 4 signals of the SPI module and 4 signals of the UART module are routed to the 21 external signals of the parallel port.
		110	Mode 6 (SPI, I2S2, I2S3, and GPIO). 7 signals of the SPI module, 4 signals of the I2S2 module, 4 signals of the I2S3 module, and 6 GPIO are routed to the 21 external signals of the parallel port.
		111	Reserved

Table 1-47. EBSR Register Bit Descriptions Field Descriptions (continued)

Bit	Field	Value	Description
11-10	SP1MODE		Serial Port 1 Mode Control Bits. The bits control the pin multiplexing of the MMC1, I2S1, and GPIO pins on serial port 1.
		00	Mode 0 (MMC/SD1). All 6 signals of the MMC/SD1 module are routed to the 6 external signals of the serial port 1.
		01	Mode 1 (I2S1 and GP[11:10]). 4 signals of the I2S1 module and 2 GP[11:10] signals are routed to the 6 external signals of the serial port 1.
		10	Mode 2 (GP[11:6]). 6 GPIO signals (GP[11:6]) are routed to the 6 external signals of the serial port 1.
		11	Reserved
9-8	SP0MODE		Serial Port 0 Mode Control Bits. The bits control the pin multiplexing of the MMC0, I2S0, and GPIO pins on serial port 0.
		00	Mode 0 (MMC/SD0). All 6 signals of the MMC/SD0 module are routed to the 6 external signals of the serial port 0.
		01	Mode 1 (I2S0 and GP[5:0]). 4 signals of the I2S0 module and 2 GP[5:4] signals are routed to the 6 external signals of the serial port 0.
		10	Mode 2 (GP[5:0]). 6 GPIO signals (GP[5:0]) are routed to the 6 external signals of the serial port 0.
		11	Reserved
7-0	Reserved	0	Reserved. Read-only. Writes have no effect.

1.7.3.2 LDO Control for TMS320C5535/34

All three LDOs can be simultaneously disabled via software by writing to either the BG_PD bit or the LDO_PD bit in the RTCPMGT register (see Figure 1-40). When the LDOs are disabled via this mechanism, the only way to re-enable them is by asserting the WAKEUP signal pin (which must also have been previously enabled to allow wakeup), *or* by a previously enabled and configured RTC alarm, *or* by cycling power to the CV_{DDRTC} pin.

ANA_LDO: The ANA_LDO is only disabled by the BG_PD and the LDO_PD mechanism described above. Otherwise, it is always enabled.

DSP_LDO: The DSP_LDO can be statically disabled by the $\overline{\text{DSP\_LDO\_EN}}$ pin as described in Section 1.5.5.1.1.2, *LDO Outputs*. It can be also dynamically disabled via the BG_PD and the LDO_PD mechanism described above. The DSP_LDO can change its output voltage dynamically by software via the DSP_LDO_V bit in the LDOCNTL register (see Figure 1-41). The DSP_LDO output voltage is set to 1.3 V at reset.

For the 50-MHz devices, DSP_LDO must be programmed to 1.05 V to match the core voltage, CV_{DD}, for proper operation after reset.

USB_LDO: The USB_LDO can be independently and dynamically enabled or disabled by software via the USB_LDO_EN bit in the LDOCNTL register (see Figure 1-41). The USB _LDO is disabled at reset.

Table 1-50 shows the ON/OFF control of each LDO and its register control bit configurations.

15–8
Reserved
R-0

7–5	4	3	2	1	0
Reserved	WU_DOUT	WU_DIR	BG_PD	LDO_PD	RTCCLKOUTEN
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 1-40. RTC Power Management Register (RTCPMGT) [1930h]

Table 1-48. RTCPMGT Register Bit Descriptions

BIT	NAME	DESCRIPTION
15:5	RESERVED	Reserved. Read-only, writes have no effect.
4	WU_DOUT	Wakeup output, active low/open-drain. 0 = WAKEUP pin driven low. 1 = WAKEUP pin is in high-impedance (Hi-Z).
3	WU_DIR	Wakeup pin direction control. 0 = WAKEUP pin configured as a input. 1 = WAKEUP pin configured as a output. Note: When the WAKEUP pin is configured as an input, it is active high. When the WAKEUP pin is configured as an output, is an open-drain that is active low and should be externally pulled-up via a 10-kΩ resistor to DV_{DDRTC}. WU_DIR must be configured as an input to allow the WAKEUP pin to wake the device up from idle modes.
2	BG_PD	Bandgap, on-chip LDOs, and the analog POR power down bit. This bit shuts down the on-chip LDOs (ANA_LDO, DSP_LDO, and USB_LDO), the Analog POR, and Bandgap reference. BG_PD and LDO_PD are only intended to be used when the internal LDOs supply power to the chip. If the internal LDOs are bypassed and not used then the BG_PD and LDO_PD power down mechanisms should not be used since POR gets powered down and the POWERGOOD signal is not generated properly. After this bit is asserted, the on-chip LDOs, Analog POR, and the Bandgap reference can be re-enabled by the WAKEUP pin (high) or the RTC alarm interrupt. The Bandgap circuit will take about 100 msec to charge the external 0.1 uF capacitor via the internal 326-kΩ resistor. 0 = On-chip LDOs, Analog POR, and Bandgap reference are enabled. 1 = On-chip LDOs, Analog POR, and Bandgap reference are disabled (shutdown).
1	LDO_PD	On-chip LDOs and Analog POR power down bit. This bit shuts down the on-chip LDOs (ANA_LDO, DSP_LDO, and USB_LDO) and the Analog POR. BG_PD and LDO_PD are only intended to be used when the internal LDOs supply power to the chip. If the internal LDOs are bypassed and not used then the BG_PD and LDO_PD power down mechanisms should not be used since POR gets powered down and the POWERGOOD signal is not generated properly. After this bit is asserted, the on-chip LDOs and Analog POR can be re-enabled by the WAKEUP pin (high) or the RTC alarm interrupt. This bit keeps the Bandgap reference turned on to allow a faster wake-up time with the expense power consumption of the Bandgap reference. 0 = On-chip LDOs and Analog POR are enabled. 1 = On-chip LDOs and Analog POR are disabled (shutdown).
0	RTCCLKOUTEN	Clockout output enable bit. 0 = Clock output disabled. 1 = Clock output enabled.

15	8
Reserved	
R-0	

7	2	1	0
Reserved		DSP_LDO_V	USB_LDO_EN
R-0		R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 1-41. LDO Control Register (LDOCNTL) [7004h]

Table 1-49. LDOCNTL Register Bit Descriptions

BIT	NAME	DESCRIPTION
15:2	RESERVED	Reserved. Read-only, writes have no effect.
1	DSP_LDO_V	DSP_LDO voltage select bit. 0 = DSP_LDOO is regulated to 1.3 V. 1 = DSP_LDOO is regulated to 1.05 V. **Note:** For the 50-MHz devices, DSP_LDO must be programmed to 1.05 V to match the core voltage, CVDD, for proper operation after reset.
0	USB_LDO_EN	USB_LDO enable bit. 0 = USB_LDO output is disabled. USB_LDOO pin is placed in high-impedance (Hi-Z) state. 1 = USB_LDO output is enabled. USB_LDOO is regulated to 1.3 V.

Table 1-50. LDO Controls Matrix

RTCPMGT Register (0x1930)		LDOCNTL Register (0x7004)	$\overline{\text{DSP\_LDO\_EN}}$ (Pin C13)	ANA_LDO	DSP_LDO	USB_LDO
BG_PD Bit	LDO_PD Bit	USB_LDO_EN Bit				
1	Don't Care	Don't Care	Don't Care	OFF	OFF	OFF
Don't Care	1	Don't Care	Don't Care	OFF	OFF	OFF
0	0	0	Low	ON	ON	OFF
0	0	0	High	ON	OFF	OFF
0	0	1	Low	ON	ON	ON

1.7.3.3 LDO Control for TMS320C5533

Both LDOs can be simultaneously disabled via software by writing to either the BG_PD bit or the LDO_PD bit in the RTCPMGT register (see Figure 1-42). When the LDOs are disabled via this mechanism, the only way to re-enable them is by asserting the WAKEUP signal pin (which must also have been previously enabled to allow wakeup), *or* by a previously enabled and configured RTC alarm, *or* by cycling power to the CV_{DDRTC} pin.

ANA_LDO: The ANA_LDO is only disabled by the BG_PD and the LDO_PD mechanism described above. Otherwise, it is always enabled.

USB_LDO: The USB_LDO can be independently and dynamically enabled or disabled by software via the USB_LDO_EN bit in the LDOCNTL register (see Figure 1-43). The USB _LDO is disabled at reset.

Table 1-53 shows the ON/OFF control of each LDO and its register control bit configurations.

15–8
Reserved
R-0

7–5	4	3	2	1	0
Reserved	WU_DOUT	WU_DIR	BG_PD	LDO_PD	RTCCLKOUTEN
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 1-42. RTC Power Management Register (RTCPMGT) [1930h]

Table 1-51. RTCPMGT Register Bit Descriptions

BIT	NAME	DESCRIPTION
15:5	RESERVED	Reserved. Read-only, writes have no effect.
4	WU_DOUT	Wakeup output, active low/open-drain. 0 = WAKEUP pin driven low. 1 = WAKEUP pin is in high-impedance (Hi-Z).
3	WU_DIR	Wakeup pin direction control. 0 = WAKEUP pin configured as a input. 1 = WAKEUP pin configured as a output. Note: When the WAKEUP pin is configured as an input, it is active high. When the WAKEUP pin is configured as an output, is an open-drain that is active low and should be externally pulled-up via a 10-kΩ resistor to DV_{DDRTC}. WU_DIR must be configured as an input to allow the WAKEUP pin to wake the device up from idle modes.
2	BG_PD	Bandgap, on-chip LDOs, and the analog POR power down bit. This bit shuts down the on-chip LDOs (ANA_LDO and USB_LDO), the Analog POR, and Bandgap reference. BG_PD and LDO_PD are only intended to be used when the internal LDOs supply power to the chip. If the internal LDOs are bypassed and not used then the BG_PD and LDO_PD power down mechanisms should not be used since POR gets powered down and the POWERGOOD signal is not generated properly. After this bit is asserted, the on-chip LDOs, Analog POR, and the Bandgap reference can be re-enabled by the WAKEUP pin (high) or the RTC alarm interrupt. The Bandgap circuit will take about 100 msec to charge the external 0.1 uF capacitor via the internal 326-kΩ resistor. 0 = On-chip LDOs, Analog POR, and Bandgap reference are enabled. 1 = On-chip LDOs, Analog POR, and Bandgap reference are disabled (shutdown).
1	LDO_PD	On-chip LDOs and Analog POR power down bit. This bit shuts down the on-chip LDOs (ANA_LDO and USB_LDO) and the Analog POR. BG_PD and LDO_PD are only intended to be used when the internal LDOs supply power to the chip. If the internal LDOs are bypassed and not used then the BG_PD and LDO_PD power down mechanisms should not be used since POR gets powered down and the POWERGOOD signal is not generated properly. After this bit is asserted, the on-chip LDOs and Analog POR can be re-enabled by the WAKEUP pin (high) or the RTC alarm interrupt. This bit keeps the Bandgap reference turned on to allow a faster wake-up time with the expense power consumption of the Bandgap reference. 0 = On-chip LDOs and Analog POR are enabled. 1 = On-chip LDOs and Analog POR are disabled (shutdown).
0	RTCCLKOUTEN	Clockout output enable bit. 0 = Clock output disabled. 1 = Clock output enabled.

15		8
Reserved		
R-0		

7	1	0
Reserved		USB_LDO_EN
R-0		R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 1-43. LDO Control Register (LDOCNTL) [7004h]

Table 1-52. LDOCNTL Register Bit Descriptions

BIT	NAME	DESCRIPTION
15:1	RESERVED	Reserved. Read-only. Writes have no effect.
0	USB_LDO_EN	USB_LDO enable bit. 0 = USB_LDO output is disabled. USB_LDOO pin is placed in high-impedance (Hi-Z) state. 1 = USB_LDO output is enabled. USB_LDOO is regulated to 1.3 V.

Table 1-53. LDO Controls Matrix

RTCPMGT Register (0x1930)		LDOCNTL Register (0x7004)	$\overline{\text{DSP\_LDO\_EN}}$ (Pin C13)	ANA_LDO	USB_LDO
BG_PD Bit	LDO_PD Bit	USB_LDO_EN Bit			
1	Don't Care	Don't Care	High	OFF	OFF
Don't Care	1	Don't Care	High	OFF	OFF
0	0	0	High	ON	OFF
0	0	0	High	ON	OFF
0	0	1	High	ON	ON

1.7.3.4 Output Slew Rate Control Register (OSRCR) [1C16h]

To provide the lowest power consumption setting, the DSP has configurable slew rate control on the CLKOUT output pin. The output slew rate control register (OSRCR) is used to set a subset of the device I/O pins, namely the CLKOUT pin, to either fast or slow slew rate. The slew rate feature is implemented by staging/delaying turn-on times of the parallel p-channel drive transistors and parallel n-channel drive transistors of the output buffer. In the slow slew rate configuration, the delay is longer, but ultimately the same number of parallel transistors are used to drive the output high or low; therefore, the drive strength is ultimately the same. The slower slew rate control can be used for power savings and has the greatest effect at a lower DVDDIO voltage.

The output slew rate control register (OSRCR) is shown in Figure 1-44 and described in Table 1-54.

Figure 1-44. Output Slew Rate Control Register (OSRCR) [1C16h]

15 3	2	1 0
Reserved	CLKOUTSR	Reserved
R-0	RW-1	R-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-54. Output Slew Rate Control Register (OSRCR) Field Descriptions

Bit	Field	Value	Description
15-3	Reserved	0	Reserved.
2	CLKOUTSR		CLKOUT pin output slew rate bits. These bits set the slew rate for the CLKOUT pin.
		0	Slow slew rate
		1	Fast slew rate
1-0	Reserved	0	Reserved.

1.7.3.5 Pull-Up/Pull-Down Inhibit Register (PDINHIBR1, PDINHIBR2, and PDINHIBR3 [1C17h, 1C18h, and 1C19h]

The device allows you to individually enable or disable the internal pull-up and pull-down resistors. You can individually inhibit the pull-up and pull-down resistors of the I/O pins through the pull-down/up inhibit registers (PDINHIBRn). There is one pin, TRSTN, that has a pulldown that is permanently enabled and cannot be disabled.

The pull-down inhibit register 1 (PDINHIBR1) is shown in Figure 1-45 and described in Table 1-55.

Figure 1-45. Pull-Down Inhibit Register 1 (PDINHIBR1) [1C17h]

15	14	13	12	11	10	9	8
Reserved		S15PD	S14PD	S13PD	S12PD	S11PD	S10PD
R-0		R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1

7	6	5	4	3	2	1	0
Reserved		S05PD	S04PD	S03PD	S02PD	S01PD	S00PD
R-0		R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-55. Pull-Down Inhibit Register 1 (PDINHIBR1) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13	S15PD		Serial port 1 pin 5 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
12	S14PD		Serial port 1 pin 4 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
11	S13PD		Serial port 1 pin 3 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
10	S12PD		Serial port 1 pin 2 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
9	S11PD		Serial port 1 pin 1 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
8	S10PD		Serial port 1 pin 0 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
7-6	Reserved	0	Reserved.
5	S05PD		Serial port 0 pin 5 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
4	S04PD		Serial port 0 pin 4 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
3	S03PD		Serial port 0 pin 3 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.

Table 1-55. Pull-Down Inhibit Register 1 (PDINHIBR1) Field Descriptions (continued)

Bit	Field	Value	Description
2	S02PD		Serial port 0 pin 2 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
1	S01PD		Serial port 0 pin 1 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
0	S00PD		Serial port 0 pin 0 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.

The pull-down inhibit register 2 (PDINHIBR2) is shown in and described in .

Figure 1-46. Pull-Down Inhibit Register 2 (PDINHIBR2) [1C18h]

15	14	13	12	11	10	9	8
Reserved	INT1PU	INT0PU	RESETPU	EMU01PU	TDIPU	TMSPU	TCKPU
R-0	R/W-1	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
Reserved							
R-0							

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-56. Pull-Down Inhibit Register 2 (PDINHIBR2) Field Descriptions

Bit	Field	Value	Description
15	Reserved	0	Reserved.
14	INT1PU		Interrupt 1 pin pull-up inhibit bit. Setting this bit to 1 disables the pin's internal pull-up.
		0	Pin pull-up is enabled.
		1	Pin pull-up is disabled.
13	INT0PU		Interrupt 0 pin pull-up inhibit bit. Setting this bit to 1 disables the pin's internal pull-up.
		0	Pin pull-up is enabled.
		1	Pin pull-up is disabled.
12	RESETPU		Reset pin pull-up inhibit bit. Setting this bit to 1 disables the pin's internal pull-up.
		0	Pin pull-up is enabled.
		1	Pin pull-up is disabled.
11	EMU01PU		EMU1 and EMU0 pin pull-up inhibit bit. Setting this bit to 1 disables the pin's internal pull-up.
		0	Pin pull-up is enabled.
		1	Pin pull-up is disabled.
10	TDIPU		TDI pin pull-up inhibit bit. Setting this bit to 1 disables the pin's internal pull-up.
		0	Pin pull-up is enabled.
		1	Pin pull-up is disabled.
9	TMSPU		TMS pin pull-up inhibit bit. Setting this bit to 1 disables the pin's internal pull-up.
		0	Pin pull-up is enabled.
		1	Pin pull-up is disabled.
8	TCKPU		TCK pin pull-up inhibit bit. Setting this bit to 1 disables the pin's internal pull-up.
		0	Pin pull-up is enabled.
		1	Pin pull-up is disabled.
7-0	Reserved	0	Reserved.

The pull-down inhibit register 3 (PDINHIBR3) is shown in Figure 1-47 and described in Table 1-57.

Figure 1-47. Pull-Down Inhibit Register 3 (PDINHIBR3) [1C19h]

15	14	13	12	11	10	9	8
PD15PD	PD14PD	PD13PD	PD12PD	PD11PD	PD10PD	PD9PD	PD8PD
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1

7	6	5	4	3	2	1	0
PD7PD	PD6PD	PD5PD	PD4PD	PD3PD	PD2PD	Reserved	
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-57. Pull-Down Inhibit Register 3 (PDINHIBR3) Field Descriptions

Bit	Field	Value	Description
15	PD15PD		Parallel port pin 15 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
14	PD14PD		Parallel port pin 14 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
13	PD13PD		Parallel port pin 13 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
12	PD12PD		Parallel port pin 12 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
11	PD11PD		Parallel port pin 11 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
10	PD10PD		Parallel port pin 10 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
9	PD9PD		Parallel port pin 9 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
8	PD8PD		Parallel port pin 8 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
7	PD7PD		Parallel port pin 7 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
6	PD6PD		Parallel port pin 6 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
5	PD5PD		Parallel port pin 5 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
4	PD4PD		Parallel port pin 4 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.

Table 1-57. Pull-Down Inhibit Register 3 (PDINHIBR3) Field Descriptions (continued)

Bit	Field	Value	Description
3	PD3PD		Parallel port pin 3 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
2	PD2PD		Parallel port pin 2 pull-down inhibit bit. Setting this bit to 1 disables the pin's internal pull-down.
		0	Pin pull-down is enabled.
		1	Pin pull-down is disabled.
1-0	Reserved	0	Reserved.

1.7.4 DMA Controller Configuration

The DSP includes four DMA controllers that allow movement of blocks of data among internal memory, external memory, and peripherals to occur without intervention from the CPU and in the background of CPU operation. Each DMA has an EVENT input signal (per channel) that can be used to tell it when to start the block transfer. And each DMA has an interrupt output (per channel) that can signal the CPU when the block transfer is completed. While most DMA configuration registers described in Chapter 12, the EVENT source and interrupt aggregation is more of a system-level concern and, therefore, they are best described in this guide.

The following sections provide more details on these features. In this section and subsections, the following notations will be used:

- Lowercase, italicized, *n* is an integer, 0-3, representing each of the 4 DMAs.
- Lowercase, italicized, *m* is an integer, 0-3, representing each of the 4 channels within each DMA.

1.7.4.1 DMA Synchronization Events

The DMA controllers allow activity in their channels to be synchronized to selected events. The DSP supports separate synchronization events and each channel can be tied to separate sync events independent of the other channels. Synchronization events are selected by programming the CH*n*EVT field in the DMA*n* channel event source registers (DMA*n*CESR1 and DMA*n*CESR2) (where *n* is an integer, 0-3, representing each of the 4 DMAs). The synchronization events available to each DMA controller are shown in Table 1-58.

Table 1-58. Channel Synchronization Events for DMA Controllers

CH*m*EVT Options	DMA0 Synchronization Event	DMA1 Synchronization Event	DMA2 Synchronization Event	DMA3 Synchronization Event
0000b	Reserved	Reserved	Reserved	Reserved
0001b	I2S0 transmit event	I2S2 transmit event	I2C transmit event	I2S1 transit event
0010b	I2S0 receive event	I2S2 receive event	I2C receive event	I2S1 receive event
0011b	Reserved	Reserved	SAR A/D event	Reserved
0100b	Reserved	Reserved	I2S3 transmit event	Reserved
0101b	MMC/SD0 transmit event	UART transmit event	I2S3 receive event	Reserved
0110b	MMC/SD0 receive event	UART receive event	Reserved	Reserved
0111b	MMC/SD1 transmit event	Reserved	Reserved	Reserved
1000b	MMC/SD1 receive event	Reserved	Reserved	Reserved
1001b	Reserved	Reserved	Reserved	Reserved
1010b	Reserved	Reserved	Reserved	Reserved
1011b	Reserved	Reserved	Reserved	Reserved
1100b	Timer 0 event	Timer 0 event	Timer 0 event	Timer 0 event
1101b	Timer 1 event	Timer 1 event	Timer 1 event	Timer 1 event
1110b	Timer 2 event	Timer 2 event	Timer 2 event	Timer 2 event
1111b	Reserved	Reserved	Reserved	Reserved

1.7.4.2 DMA Configuration Registers

The system-level DMA registers are listed in Table 3-4. The DMA interrupt flag and enable registers (DMAIFR and DMAIER) are used to control the aggregation and CPU interrupt generation for the four DMA controllers and their associated channels. In addition, there are two registers per DMA controller which control event synchronization in each channel; the DMA*n* channel event source registers (DMA*n*CESR1 and DMA*n*CESR2).

Table 1-59. System Registers Related to the DMA Controllers

CPU Word Address	Acronym	Register Description
1C30h	DMAIFR	DMA Interrupt Flag Register
1C31h	DMAIER	DMA Interrupt Enable Register
1C1Ah	DMA0CESR1	DMA0 Channel Event Source Register 1
1C1Bh	DMA0CESR2	DMA0 Channel Event Source Register 2
1C1Ch	DMA1CESR1	DMA1 Channel Event Source Register 1
1C1Dh	DMA1CESR2	DMA1 Channel Event Source Register 2
1C36h	DMA2CESR1	DMA2 Channel Event Source Register 1
1C37h	DMA2CESR2	DMA2 Channel Event Source Register 2
1C38h	DMA3CESR1	DMA3 Channel Event Source Register 1
1C39h	DMA3CESR2	DMA3 Channel Event Source Register 2

1.7.4.2.1 DMA Interrupt Flag Register (DMAIFR) [1C30h] and DMA Interrupt Enable Register (DMAIER) [1C31h]

The DSP includes two registers for aggregating the four channel interrupts of the four DMA controllers. Use the DMA interrupt enable register (DMAIER) to enable channel interrupts. At the end of a block transfer, if the DMA controller channel interrupt enable (DMA*n*CH*m*IE) bit is 1, an interrupt request is sent to the DSP CPU, where it can be serviced or ignored. Each channel can generate an interrupt, although all channel interrupts are aggregated into a single DMA interrupt signal to the CPU.

To see which channel generated an interrupt, your program can read the DMA interrupt flag register (DMAIFR). The DMA controller channel interrupt flag (DMA*n*CH*m*IF) bits are set to 1 when a DMA channel generates an interrupt. Your program must manually clear the bits of DMAIFR by writing a 1 to the bit positions to be cleared.

Figure 1-48. DMA Interrupt Flag Register (DMAIFR) [1C30h]

15	14	13	12	11	10	9	8
DMA3CH3IF	DMA3CH2IF	DMA3CH1IF	DMA3CH0IF	DMA2CH3IF	DMA2CH2IF	DMA2CH1IF	DMA2CH0IF
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
DMA1CH3IF	DMA1CH2IF	DMA1CH1IF	DMA1CH0IF	DMA0CH3IF	DMA0CH2IF	DMA0CH1IF	DMA0CH0IF
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 1-49. DMA Interrupt Enable Register (DMAIER) [1C31h]

15	14	13	12	11	10	9	8
DMA3CH3IE	DMA3CH2IE	DMA3CH1IE	DMA3CH0IE	DMA2CH3IE	DMA2CH2IE	DMA2CH1IE	DMA2CH0IE
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
DMA1CH3IE	DMA1CH2IE	DMA1CH1IE	DMA1CH0IE	DMA0CH3IE	DMA0CH2IE	DMA0CH1IE	DMA0CH0IE
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-60. DMA Interrupt Flag Register (DMAIFR) Field Descriptions

Bit	Field	Value	Description
15-0	DMA*n*CH*m*IF		Channel interrupt status bits.
		0	DMA controller *n,* channel *m* has not completed its block transfer.
		1	DMA controller *n,* channel *m* block transfer complete.

Table 1-61. DMA Interrupt Enable Register (DMAIER) Field Descriptions

Bit	Field	Value	Description
15-0	DMA*n*CH*m*IE		Channel interrupt enable bits.
		0	DMA controller *n,* channel *m* interrupt is disabled.
		1	DMA controller *n,* channel *m* interrupt is enabled.

1.7.4.2.2 DMAn Channel Event Source Registers (DMAnCESR1 and DMAnCESR2) [1C1Ah, 1C1Bh, 1C1Ch, 1C1Dh, 1C36h, 1C37h, 1C38h, and 1C39h]

Each DMA controller contains two channel event source registers (DMA*n*CESR1 and DMA*n*CESR2). DMA*n*CESR1 controls the synchronization event for DMA*n* channel 0 and 1 while DMA*n*CESR2 controls the synchronization event for DMA*n* channel 2 and 3.

The synchronization events available to each DMA controller are shown in Table 1-58. Multiple DMAs and multiple channels within a DMA are allowed to have the same synchronization event.

Figure 1-50. DMA*n* Channel Event Source Register 1 (DMA*n*CESR1) [1C1Ah, 1C1Ch, 1C36h, and 1C38h]

15 12	11 8	7 4	3 0
Reserved	CH1EVT	Reserved	CH0EVT
R-0	RW-0	R-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 1-51. DMA*n* Channel Event Source Register 2 (DMA*n*CESR2) [1C1Bh, 1C1Dh, 1C37h, and 1C39h]

15 12	11 8	7 4	3 0
Reserved	CH3EVT	Reserved	CH2EVT
R-0	RW-0	R-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-62. DMA*n* Channel Event Source Register 1 (DMA*n*CESR1) Field Descriptions

Bit	Field	Value	Description
15-12	Reserved	0	Reserved.
11-8	CH1EVT	0-Fh	Channel 1 synchronization events. When SYNCMODE = 1 in a channel's DMACH*m*TCR2, the CH1EVT bits in the DMA*n*CESR registers specify the synchronization event for activity in the DMA controller. See Table 1-58 for a list of available synchronization event options.
7-4	Reserved	0	Reserved.
3-0	CH0EVT	0-Fh	Channel 0 synchronization events. when SYNCMODE = 1 in a channel's DMACH*m*TCR2, the CH0EVT bits in the DMA*n*CESR registers specify the synchronization event for activity in the DMA controller. See Table 1-58 for a list of available synchronization event options.

Table 1-63. DMA*n* Channel Event Source Register 2 (DMA*n*CESR2) Field Descriptions

Bit	Field	Value	Description
15-12	Reserved	0	Reserved.
11-8	CH3EVT	0-Fh	Channel 3 synchronization events. When SYNCMODE = 1 in a channel's DMACH*m*TCR2, the CH3EVT bits in the DMA*n*CESR registers specify the synchronization event for activity in the DMA controller. See Table 1-58 for a list of available synchronization event options.
7-4	Reserved	0	Reserved.
3-0	CH2EVT	0-Fh	Channel 2 synchronization events. When SYNCMODE = 1 in a channel's DMACH*m*TCR2, the CH2EVT bits in the DMA*n*CESR registers specify the synchronization event for activity in the DMA controller. See Table 1-58 for a list of available synchronization event options.

1.7.5 Peripheral Reset

All peripherals can be reset through software using the peripheral reset control register (PRCR). The peripheral software reset counter register (PSRCR) controls the duration, in SYSCLK cycles, that the reset signal is asserted low once activated by the bits in PRCR.

To reset a peripheral or group of peripherals, follow these steps:

1. Set COUNT = 08h in PSRCR.
2. Initiate the desired peripheral reset by setting to 1 the bits of PRCR.
3. Do not attempt to access the peripheral for at least the number of clock cycles set in the PSRCR register. A repeated NOP may be necessary.

In some cases, a single reset is used for multiple peripherals. For example, PG4_RST controls the reset to I2S2, I2S3, UART, and SPI.

1.7.5.1 Peripheral Software Reset Counter Register (PSRCR) [1C04h]

The Peripheral Software Reset Counter Register (PSRCR) is shown in Table 1-64 and described in Table 1-64.

Figure 1-52. Peripheral Software Reset Counter Register (PSRCR) [1C04h]

15	0
COUNT	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 1-64. Peripheral Software Reset Counter Register (PSRCR) Field Descriptions

Bit	Field	Value	Description
15-0	COUNT	0-FFFFh	Count bits. These bits specify the number of system clock (SYSCLK) cycles the software reset signals are asserted. When the software counter reaches 0, the software reset bits will be cleared to 0. Always initialize this field with a value of at least 08h.

1.7.5.2 Peripheral Reset Control Register (PRCR) [1C05h]

Writing a 1 to any bits in this register initiates the reset sequence for the associated peripherals. The associated peripherals will be held in reset for the duration of clock cycles set in the PSRCR register and they should not be accessed during that time. Reads of this register return the state of the reset signal for the associated peripherals. In other words, polling may be used to wait for the reset to become de-asserted.

The Peripheral Reset Control Register (PRCR) is shown in Figure 1-53 and described in Table 1-65.

Figure 1-53. Peripheral Reset Control Register (PRCR) [1C05h]

15	14	13	12	11	10	9	8
Reserved							
R-0							

7	6	5	4	3	2	1	0
PG4_RST	Reserved	PG3_RST	DMA_RST	USB_RST	SAR_RST	PG1_RST	I2C_RST
R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1-65. Peripheral Reset Control Register (PRCR) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved. Always write 0 to these bits.
7	PG4_RST		Peripheral group 4 software reset bit. Drives the I2S2, I2S3, UART, and SPI reset signal.
		Write 0	Writing zero has no effect
		Write 1	Writing one starts resetting the peripheral group
		Read 0	Reading zero means that peripheral group is out of reset
		Read 1	Reading one means the peripheral group is being held in reset and should not be accessed
6	Reserved	0	Reserved, always write 0 to this bit.
5	PG3_RST		Peripheral group 3 software reset bit. Drives the MMC/SD0, MMC/SD1, I2S0, and I2S1 reset signal.
		Write 0	Writing zero has no effect
		Write 1	Writing one starts resetting the peripheral group
		Read 0	Reading zero means that peripheral group is out of reset
		Read 1	Reading one means the peripheral group is being held in reset and should not be accessed
4	DMA_RST		DMA software reset bit. Drives the reset signal to all four controllers.
		Write 0	Writing zero has no effect
		Write 1	Writing one starts resetting the peripheral group
		Read 0	Reading zero means that peripheral group is out of reset
		Read 1	Reading one means the peripheral group is being held in reset and should not be accessed
3	USB_RST		USB software reset bit. Drives the USB reset signal.
		Write 0	Writing zero has no effect
		Write 1	Writing one starts resetting the peripheral group
		Read 0	Reading zero means that peripheral group is out of reset
		Read 1	Reading one means the peripheral group is being held in reset and should not be accessed
2	SAR_RST		SAR software reset bit and reset for most analog-related register in the IO-space address range of 0x7000-0x70FF
		Write 0	Writing zero has no effect
		Write 1	Writing one starts resetting the peripheral group
		Read 0	Reading zero means that peripheral group is out of reset
		Read 1	Reading one means the peripheral group is being held in reset and should not be accessed
1	PG1_RST		Peripheral group 1 software reset bit. Drives all three timer reset signal.
		Write 0	Writing zero has no effect
		Write 1	Writing one starts resetting the peripheral group
		Read 0	Reading zero means that peripheral group is out of reset
		Read 1	Reading one means the peripheral group is being held in reset and should not be accessed
0	I2C_RST		I2C software reset bit. Drives the I2C reset signal.
		Write 0	Writing zero has no effect
		Write 1	Writing one starts resetting the peripheral group
		Read 0	Reading zero means that peripheral group is out of reset
		Read 1	Reading one means the peripheral group is being held in reset and should not be accessed

1.7.6 USB Byte Access (Not Available for TMS320C5532)

The C55x CPU architecture cannot generate 8-bit accesses to its data or I/O space. But in some cases specific to the USB peripheral, it is necessary to access a single byte of data.

For these situations, the upper or lower byte of a CPU word access can be masked using the BYTEMODE bits of the USB system control register (USBSCR). The BYTEMODE bits of USBSCR only affect CPU accesses to the USB registers. Table 1-66 summarizes the effect of the BYTEMODE bits for different CPU operations.

The USB system control register (USBSCR) is described in Section 1.5.3.4.2.

Table 1-66. Effect of USBSCR BYTEMODE Bits on USB Access

BYTEMODE Setting	CPU Access to USB Register
BYTEMODE = 00b (16-bit word access)	Entire register contents are accessed
BYTEMODE = 01b (8-bit access with high byte selected)	Only the upper byte of the register is accessed
BYTEMODE = 10b (8-bit access with low byte selected)	Only the lower byte of the register is accessed

FFT Implementation on the TMS320C5535 DSP

This chapter describes how to implement the fast fourier transform for TMS320C5535.

Topic **Page**

2.1 Introduction

The Fast Fourier Transform (FFT) is an efficient means for computing the Discrete Fourier Transform (DFT). It is one of the most widely used computational elements in Digital Signal Processing (DSP) applications. This DSP is ideally suited for such applications. They include an FFT hardware accelerator (HWAFFT) that is tightly coupled with the CPU, allowing high FFT processing performance at very low power. This application report describes FFT computation on the TMS320C5535 DSP and covers the following topics: [1]

- Basics of DFT and FFT
- DSP Overview Including the FFT Accelerator
- HWAFFT Description
- HWAFFT Software Interface
- Simple Example to Illustrate the Use of the FFT Accelerator
- FFT Benchmarks
- Description of Open Source FFT Example Software
- Computation of Large (Greater Than 1024-point) FFTs

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www-s.ti.com/sc/techlit/sprabb6.zip.

2.2 Basics of DFT and FFT

The DFT takes an N-point vector of complex data sampled in time and transforms it to an N-point vector of complex data that represents the input signal in the frequency domain. A discussion on the DFT and FFT is provided as background material before discussing the HWAFFT implementation.

The DFT of the input sequence x(n), n = 0, 1, …, N-1 is defined as

$$X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)W_N^{nk},\ k = 0 \text{ to } N-1 \tag{1}$$

Where W_N, the twiddle factor, is defined as

$$W_N = e^{-j2\pi/N},\ k = 0 \text{ to } N-1 \tag{2}$$

The FFT is a class of efficient DFT implementations that produce results identical to the DFT in far fewer cycles. The Cooley-Tukey algorithm is a widely used FFT algorithm that exploits a divide-and-conquer approach to recursively decompose the DFT computation into smaller and smaller DFT computations until the simplest computation remains. One subset of this algorithm called Radix-2 Decimation-in-Time (DIT) breaks each DFT computation into the combination of two DFTs, one for even-indexed inputs and another for odd-indexed inputs. The decomposition continues until a DFT of just two inputs remains. The 2-point DFT is called a butterfly, and it is the simplest computational kernel of Radix-2 FFT algorithms.

2.2.1 Radix-2 Decimation in Time Equations

The Radix-2 DIT decomposition can be seen by manipulating the DFT equation (Equation 1):

Split x(n) into even and odd indices:

$$X(k) = \frac{1}{N}\sum_{n=0}^{N/2-1} x(2n)W_N^{2nk} + \frac{1}{N}\sum_{n=0}^{N/2-1} x(2n+1)W_N^{(2n+1)k},\ k = 0 \text{ to } N-1 \tag{3}$$

Factor W_N^k from the odd indexed summation:

$$X(k) = \frac{1}{N}\sum_{n=0}^{N/2-1} x(2n)W_N^{2nk} + \frac{W_N^k}{N}\sum_{n=0}^{N/2-1} x(2n+1)W_N^{2nk},\ k = 0 \text{ to } N-1 \tag{4}$$

[1] Fast Fourier Transform (FFT) is available only on TMS320C5535.

Only twiddle factors from 0 to N/2 are needed:

$$W_N^{2nk} = (e^{-j2\Pi/N})^{2nk} = (e^{-j2\Pi/(N/2)})^{nk} = W_{N/2}^{nk} \text{ and } W_N^{k+N/2} = -W_N^k, k = 0 \text{ to } N/2 - 1 \quad (5)$$

This results in:

$$X(k) = \frac{1}{N}\sum_{n=0}^{N/2-1} x(2n)W_{N/2}^{nk} + \frac{W_N^k}{N}\sum_{n=0}^{N/2-1} x(2n+1)W_{N/2}^{nk}, \ k = 0 \text{ to } N - 1 \quad (6)$$

Define $X_{even}(k)$ and $X_{odd}(k)$ such that:

$$X_{even}(k) = \frac{1}{(N/2)}\sum_{n=0}^{N/2-1} x(2n)W_{N/2}^{nk}, \ k = 0 \text{ to } N - 1 \quad (7)$$

and

$$X_{odd}(k) = \frac{1}{(N/2)}\sum_{n=0}^{N/2-1} x(2n+1)W_{N/2}^{nk}, \ k = 0 \text{ to } N - 1 \quad (8)$$

Finally, Equation 6 is rewritten as:

$$X(k) = \frac{1}{2}(X_{even}(k) + W_N^k X_{odd}(k)), \ k = 0 \text{ to } N/2 - 1 \quad (9)$$

and

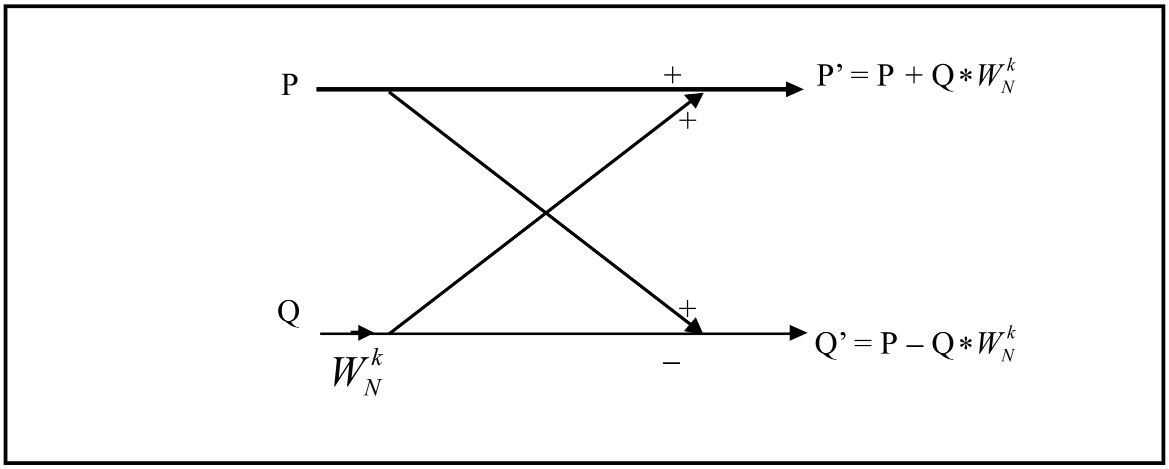
$$X(k + N/2) = \frac{1}{2}(X_{even}(k) - W_N^k X_{odd}(k)), \ k = 0 \text{ to } N/2 - 1 \quad (10)$$

Equation 9 and Equation 10 show that the N-point DFT can be divided into two smaller N/2-point DFTs. Each smaller DFT is then further divided into smaller DFTs until N = 2. The pair of equations that makeup the 2-point DFT is called the Radix2 DIT Butterfly (see Section 2.2.2). The DIT Butterfly is the core calculation of the FFT and consists of just one complex multiplication and two complex additions.

2.2.2 Radix-2 DIT Butterfly

The Radix-2 Butterfly is illustrated in Figure 2-1. In each butterfly structure, two complex inputs P and Q are operated upon and become complex outputs P' and Q'. Complex multiplication is performed on Q and the twiddle factor, then the product is added to and subtracted from input P to form outputs P' and Q'. The exponent of the twiddle factor W_N^k is dependent on the stage and group of its butterfly. The butterfly is usually represented by its flow graph (Figure 2-1), which looks like a butterfly.

Figure 2-1. DIT Radix 2 Butterfly

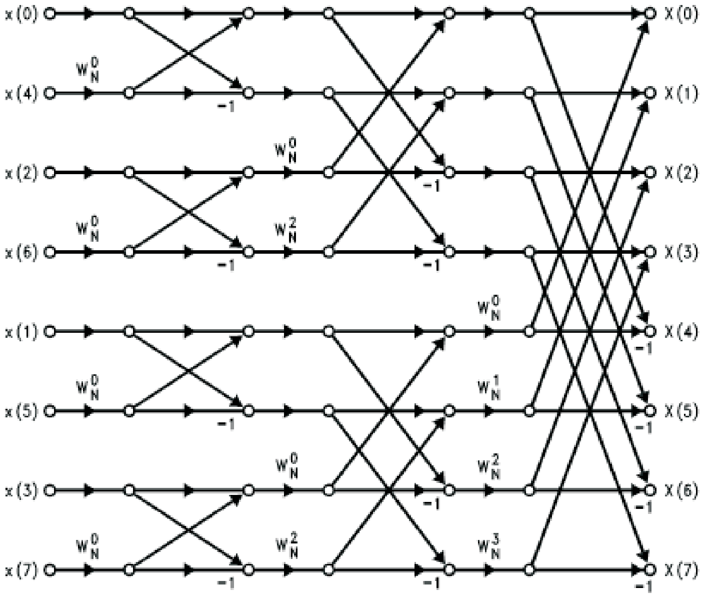


The mathematical meaning of this butterfly is shown below with separate equations for real and imaginary parts:

Complex	Real Part	Imaginary Part
P' = P + Q * W	Pr' = Pr + (Qr * Wr - Qi * Wi)	Pi' = Pi + (Qr * Wi + Qi * Wr)
Q' = P - Q * W	Qr' = Pr - (Qr * Wr - Qi * Wi)	Qi' = Pi - (Qr * Wi + Qi * Wr)

The flow graph in Figure 2-2 shows the interconnected butterflies of an 8-point Radix-2 DIT FFT. Notice that the inputs to the FFT are indexed in bit-reversed order (0, 4, 2, 6, 1, 5, 3, 7) and the outputs are indexed in sequential order (0, 1, 2, 3, 4, 5, 6, 7). Computation of a Radix-2 DIT FFT requires the input vector to be in bit-reversed order, and generates an output vector in sequential order. This bit-reversal is further explained in Section 2.5.3, *Bit-Reverse Function*.

Figure 2-2. DIT Radix 2 8-point FFT



2.2.3 Computational Complexity

The Radix-2 DIT FFT requires log$_2$(N) stages, N/2 * log$_2$(N) complex multiplications, and N * log$_2$(N) complex additions. In contrast, the direct computation of X(k) from the DFT equation (Equation 1) requires N^2 complex multiplications and (N^2 – N) complex additions. Table 2-1 compares the computational complexity for direct DFT versus Radix-2 FFT computations for typical FFT lengths.

Table 2-1. Computational Complexity of Direct DFT Computation versus Radix-2 FFT

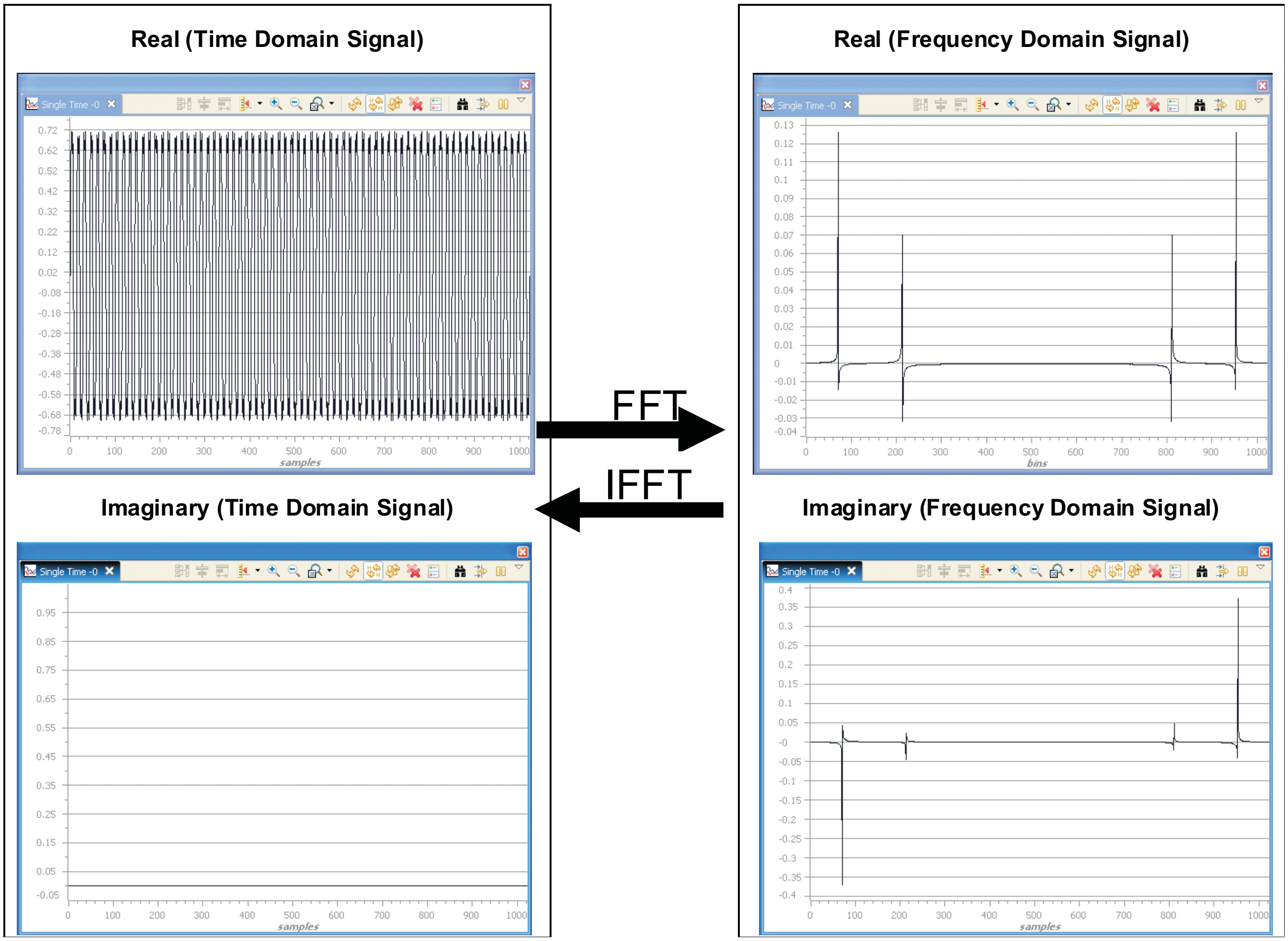
FFT Length	Direct DFT Computation		Radix-2 FFT	
	Complex Multiplications	Complex Additions	Complex Multiplications	Complex Additions
128	16,384	16,256	448	896
256	65,536	65,280	1,024	2,048
512	262,144	264,632	2,304	4,608
1024	1,048,576	1,047,552	5,120	10,240

Table 2-1 clearly shows significant reduction in computational complexity with the Radix-2 FFT, especially for large N. This substantial decrease in computational complexity of the FFT has allowed DSPs to efficiently compute the DFT in reasonable time. For its substantial efficiency improvement over direct computation, the HWAFFT coprocessor in the DSP implements the Radix-2 FFT algorithm.

2.2.4 FFT Graphs

Figure 2-3 is a graphical example of the FFT computation. These results were obtained by using the HWAFFT coprocessor on the DSP. On the left half is the complex time domain signal (real part on top, imaginary part on bottom). On the right half is the complex frequency domain signal produced by the FFT computation (real part on top, imaginary part on bottom). In this example, two sinusoidal tones are present in the time domain. The time domain signal is 1024 points and contains only real data (the imaginary part is all zeros). The two sinusoids are represented in the frequency domain as impulses located at the frequency bins that correspond to the two sinusoidal frequencies. The frequency domain signal is also 1024 points and contains both real parts (top right) and imaginary parts (bottom right).

Figure 2-3. Graphical FFT Computation



Real (Time Domain Signal)

Real (Frequency Domain Signal)

Imaginary (Time Domain Signal)

Imaginary (Frequency Domain Signal)

2.3 DSP Overview Including the FFT Accelerator

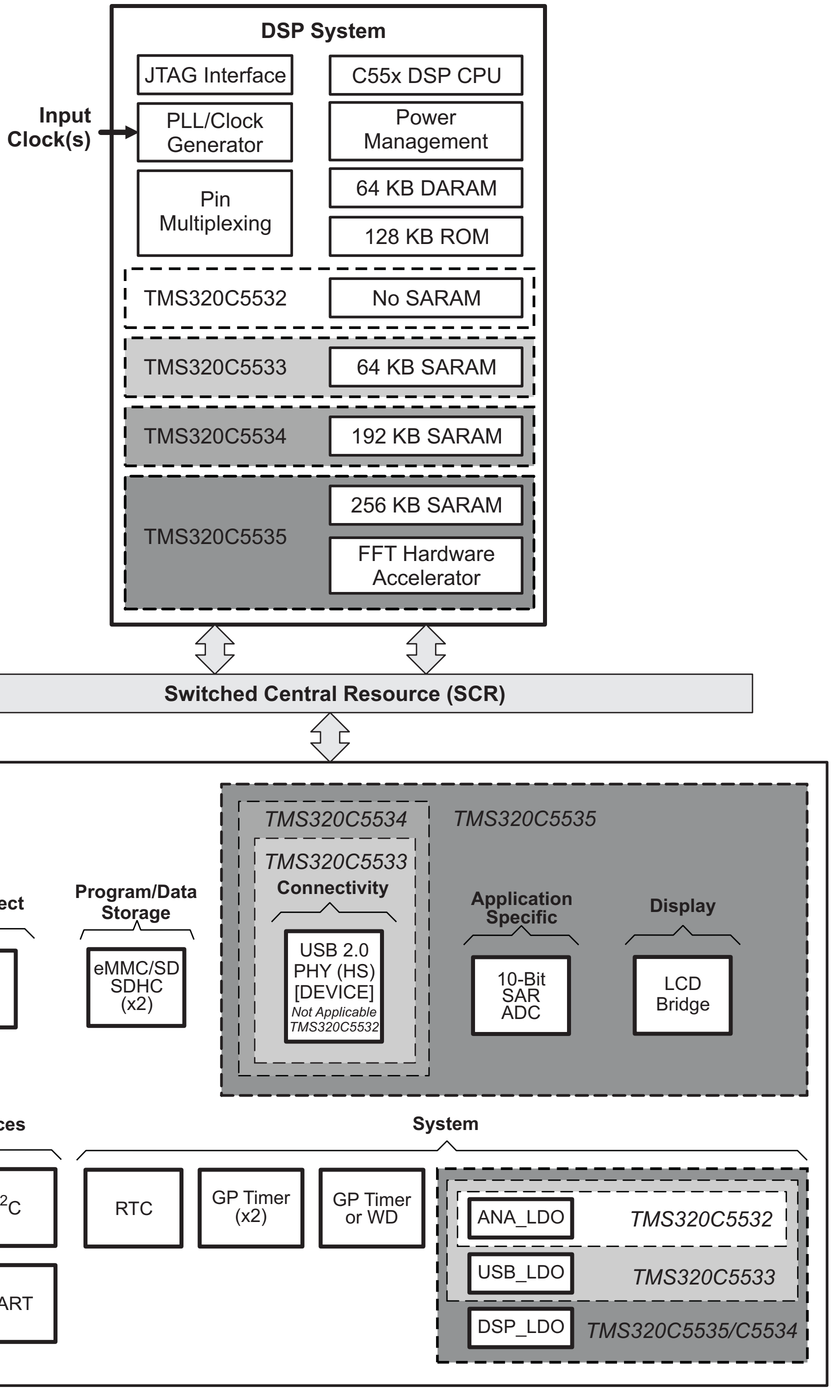
This DSP is a member of TI's TMS320C5000™ fixed-point DSP product family and is designed for low-power applications. With an active mode power consumption of less than 0.15 mW/MHz and a standby mode power consumption of less than 0.15 mW, these DSPs are optimized for applications characterized by sophisticated processing and portable form factors that require low power and longer battery life. Examples of such applications include portable voice/audio devices, noise cancellation headphones, software-defined radio, musical instruments, medical monitoring devices, wireless microphones, biometrics, industrial instruments, telephony, and audio cards.

Figure 2-4 shows an overview of the DSP consisting of the following primary components:

- Dual MAC, C55x CPU

  1.05 V @ 50 MHz, 1.3 V @ 100 MHz)
- On-Chip memory: 320 KB RAM (64 KB DARAM, 256 KB SARAM), 128 KB ROM
- HWAFFT that supports 8- to 1024-point (powers of 2) real and complex-valued FFTs
- Four DMA controllers and external memory interface
- Power management module
- A set of I/O peripherals that includes I^{2}C, I^{2}S, SPI, UART, Timers, MMC/SD, GPIO, 10-bit SAR, LCD Controller, USB 2.0
- Three on-chip LDO regulators (C5535 and C5534), Two on-chip LDO regulators (C5533), one on-chip LDO regulator (C5532)

Figure 2-4. Block Diagram



The C55x CPU includes a tightly coupled FFT accelerator that communicates with the C55x CPU through the use of the coprocessor instructions. The main features of this hardware accelerator are:

- Supports 8- to 1024-point (powers of 2) complex-valued FFTs.
- Internal twiddle factor generation for optimal use of memory bandwidth and more efficient programming.
- Basic and software-driven auto-scaling feature provides good precision versus cycle count trade-off.
- Single-stage and double-stage modes enable computation of one or two stages in one pass, and thus better handle the odd power of two FFT widths.
- Is 4 to 6 times more energy efficient and 2.2 to 3.8 times faster than the FFT computations on the CPU.

2.4 FFT Hardware Accelerator Description

The HWAFFT in the DSP is a tightly-coupled, software-controlled coprocessor designed to perform FFT and inverse FFT (IFFT) computations on complex data vectors ranging in length from 8 to 1024 points (powers of 2). It implements a Radix-2 DIT structure that returns the FFT or IFFT result in bit-reversed order.

2.4.1 Tightly-Coupled Hardware Accelerator

The HWAFFT is tightly-coupled with the DSP core which means that it is physically located outside of the DSP core but has access to the core's full memory read bandwidth (busses B, C, and D), access to the core's internal registers and accumulators, and access to its address generation units. The HWAFFT cannot access the data write busses or memory mapped registers (MMRs). Because the HWAFFT is seen as part of the execution unit of the CPU, it must also comply to the core's pipeline exceptions, and in particular those caused by stalls and conditional execution.

2.4.2 Hardware Butterfly, Double-Stage and Single-Stage Mode

The core of the HWAFFT consists of a single Radix-2 DIT Butterfly implemented in hardware. This hardware supports a double-stage mode where two FFT stages are computed a single pass. In this mode the HWAFFT feeds the results from the first stage back into the hardware butterfly to compute the second stage results in a single pass. This double-stage mode offers significant speed-up especially for large FFT lengths. However, when the number of required stages is odd (FFT lengths = 8, 32, 128, or 512 points) the final stage needs to be computed alone and, consequently, at a lower acceleration rate. For this reason a single-stage mode is also provided.

The HWAFFT supports two stage modes:

- Double-Stage Mode – two FFT stages performed in each pass
- Single-Stage Mode – one FFT stage performed in each pass

2.4.3 Pipeline and Latency

The logic of the HWAFFT is pipelined to deliver maximum throughput. Complex multiplication with the twiddle factors is performed in the first pipeline stage, and complex addition and subtraction is performed in the second pipeline stage. Valid results appear some cycles of latency after the first data is read from memory:

- 5 cycles of latency in single-stage mode
- 9 cycles of latency in double-stage mode

There are three states to consider during computation of a single or double stage:

- Prologue: The hardware accelerator is fed with one complex input at a time but does not output any valid data.
- Kernel: Valid outputs appear while new inputs are received and computed upon.
- Epilogue: A few more cycles are needed to flush the pipeline and output the last butterfly results.

Consecutive stages can be overlapped such that the first data points for the next pass are read while the final output values of the current pass are returned. For odd-power-of-two FFT lengths, the last double-stage pass needs to be completed before starting a final single-stage pass. Thus, the double-stage latency is only experienced once for even-powers-of-2 FFT computations and twice for odd-powers-of-2 FFT computations. Latency has little impact on the total computation performance, and less and less so as the FFT size increases.

2.4.4 Software Control

Software is required to communicate between the CPU and the HWAFFT. The CPU instruction set architecture (ISA) includes a class of coprocessor (copr) instructions that allows the CPU to initialize, pass data to, and execute butterfly computations on the HWAFFT. Computation of an FFT/IFFT is performed by executing a sequence of these copr instructions.

C-callable HWAFFT functions are provided with optimized sequences of copr instructions for each available FFT length. To conserve program memory, these functions are located in the DSP's ROM. A detailed explanation of the HWAFFT software interface and its application is provided in Section 2.5, *HWAFFT Software Interface*.

2.4.5 Twiddle Factors

To conserve memory bus bandwidth, twiddle factors are stored in a look-up-table within the HWAFFT coprocessor. The 512 complex twiddle factors (16-bit real, 16-bit imaginary) are available for computing up to 1024-point FFTs. Smaller FFT lengths use a decimated subset of these twiddle factors. Indexing the twiddle table is pipelined and optimized based on the current FFT stage and group being computed. When the IFFT computation is performed, the complex conjugates of the twiddle factors are used.

2.4.6 Scaling

FFT computation with fixed-point numbers is vulnerable to overflow, underflow, and saturation effects. Depending on the dynamic range of the input data, some scaling may be required to avoid these effects during the FFT computation. This scaling can be done before the FFT computation, by computing the dynamic range of the input points and scaling them down accordingly. If the magnitude of each complex input element is less than 1/N, where N = FFT Length, then the N-point FFT computation will not overflow.

Uniformly dividing the input vector elements by N (Pre-scaling) is equivalent to shifting each binary number right by $\log_2(N)$ bits, which introduces significant error (especially for large FFT lengths). When this error propagates through the FFT flow graph, the output noise-to-signal ratio increases by 1 bit per stage or $\log_2(N)$ bits in total. Overflow will not occur if each input's magnitude is less than 1/N.

Alternatively, a simple divide-by-2 and round scaling after each butterfly offers a good trade-off between precision and overflow protection, while minimizing computation cycles. Because the error introduced by early FFT stages is also scaled after each butterfly, the output noise-to-signal ratio increases by just ½ bit per stage or $½ * \log_2(N)$ bits in total. Overflow is avoided if each input's magnitude is less than 1.

The HWAFFT supports two scale modes:

- NOSCALE
  - Scaling logic disabled
  - Vulnerable to overflow
  - Output dynamic range grows with each stage
  - No overflow if input magnitudes < 1/N
- SCALE
  - Scales each butterfly output by 1/2
  - No overflow if input magnitudes < 1

2.5 HWAFFT Software Interface

The software interface to the HWAFFT is handled through a set of coprocessor instructions that, when decoded by the coprocessor, perform initialization, load/store, and execution operations on the HWAFFT coprocessor. C-callable functions are provided that contain the necessary sequences of coprocessor instructions for performing FFT/ IFFT computations in the range of 8 to 1024 points (by powers of 2). Additionally, an optimized out-of-place bit-reversal function is provided to perform the complex vector bit-reversal required by Radix-2 FFT computations. These functions are defined in the hwafft.asm source code file. Additionally, to conserve on-chip RAM these functions have been placed in the on-chip ROM of the DSP. See Section 2.5.5, *Project Configuration for Calling Function s from ROM*, for steps to configure your project to call the HWAFFT functions from ROM.

2.5.1 Data Types

The input and output vectors of the HWAFFT contain complex numbers. Each real and imaginary part is represented by a two's complement, 16-bit fixed-point number. The most significant bit holds the number's sign value, and the remaining 15 are fraction bits (S16Q15 format). The range of each number is [-1, 1 – $(1/2)^{15}$]. Real and imaginary parts appear in an interleaved order within each vector:

Int16 CMPLX_Vec16[2*N] = …(N = FFT Length)

Real[0]	Imag[0]	Real[1]	Imag[1]	Real[2]	Imag[2]
Bit15,..................,Bit0	Bit15,..................,Bit0	Bit15,..................,Bit0	Bit15,..................,Bit0	Bit15,..................,Bit0	Bit15,..................,Bit0

The HWAFFT functions use an Int32 pointer to reference these complex vectors. Therefore, each 32-bit element contains the 16-bit real part in the most significant 16 bits, and the 16-bit imaginary part in the least significant 16 bits.

Int32 CMPLX_Vec32[N] = …(N = FFT Length)

Real[0] Imag[0]	Real[1] Imag[1]	Real[2] Imag[2]
Bit31,.................., Bit16, Bit15,.................., Bit0	Bit31,.................., Bit16, Bit15,.................., Bit0	Bit31,.................., Bit16, Bit15,.................., Bit0

To extract the real and imaginary parts from the complex vector, it is necessary to mask and shift each 32-bit element into its 16-bit real and imaginary parts:

```
Uint16 Real_Part = CMPLX_Vec32[i] >> 16;
Uint16 Imaginary_Part = CMPLX_Vec32[i] & 0x0000FFFF;
```

2.5.2 HWAFFT Functions

C-Callable HWAFFT Functions are provided for computing FFT/IFFT transforms on the HWAFFT coprocessor. These functions contain optimized sequences of coprocessor instructions for computing scaled or unscaled 8-, 16-, 32-, 64-, 128-, 256-, 512-, and 1024-point FFT/IFFTs. Additionally, an optimized out-of-place bit-reversal function is provided to bit-reverse the input vector before supplying it to the HWAFFT. Computation of a Radix-2 DIT FFT requires the input vector to be in bit-reversed order, and generates an output vector in sequential order.

2.5.2.1 HWAFFT Naming and Format

The HWAFFT functions are named hwafft_Npts, where N is the FFT length. For example, hwafft_32pts is the name of the function for performing 32-point FFT and IFFT operations. The structure of the HWAFFT functions is:

`Uint16 hwafft_Npts(`	Performs N-point complex FFT/IFFT, where N = {8, 16, 32, 64, 128, 256, 512, 1024}
`Int32 *data,`	Input/output – complex vector
`Int32 *scratch,`	Intermediate/output – complex vector
`Uint16 fft_flag,`	Flag determines whether FFT or IFFT performed, (0 = FFT, 1 = IFFT)
`Uint16 scale_flag`	Flag determines whether butterfly output divided by 2 (0 = Scale, 1 = No Scale)
`); Return value`	Flag determines whether output in data or scratch vector (0 = data, 1 = scratch)

2.5.2.2 HWAFFT Parameters

The following describe the parameters for the HWAFFT functions.

Int32 *data

This is the input vector to the HWAFFT. It contains complex data elements (real part in most significant 16 bits, imaginary part in least significant 16 bits). After the HWAFFT function completes, the result will either be stored in this data vector or in the scratch vector, depending on the status of the return value. The return value is Boolean where 0 indicates that the result is stored in this data vector, and 1 indicates the scratch vector. The data and scratch vectors must reside in separate blocks of RAM (DARAM or SARAM) to maximize memory bandwidth.

```
#pragma DATA_SECTION(data_buf, "data_buf");
     //Static Allocation to Section: "data_buf  : > DARAM" in Linker CMD File
Int32 data_buf[N = FFT Length];
Int32 *data = data_buf;
Int32 *data:
```

The *data parameter is a complex input vector to HWAFFT. It contains the output vector if Return Value = 0 = OUT_SEL_DATA. There is a strict address alignment requirement if *data is shared with a bit-reverse destination vector (**recommended**). See Section 2.5.3.1, *Bit Reverse Destination Vector Alignment Requirement*.

Int32 *scratch

This is the scratch vector used by the HWAFFT to store intermediate results between FFT stages. It contains complex data elements (real part in most significant 16 bits, imaginary part in least significant 16 bits). After the HWAFFT function completes the result will either be stored in the data vector or in this scratch vector, depending on the status of the return value. The return value is Boolean, where 0 indicates that the result is stored in the data vector, and 1 indicates this scratch vector. The data and scratch vectors must reside in separate blocks of RAM (DARAM or SARAM) to maximize memory bandwidth.

```
#pragma DATA_SECTION(scratch_buf, "scratch_buf");
     //Static Allocation to Section: "scratch_buf  : > DARAM" in Linker CMD File
Int32 scratch_buf[N = FFT Length];
Int32 *scratch = scratch_buf;
Int32 *scratch:
```

The *scratch parameter is a complex scratchpad vector to HWAFFT. It contains the output vector if Return Value = 1 = OUT_SEL_SCRATCH.

Uint16 fft_flag

The FFT/IFFT selection is controlled by setting the fft_flag to 0 for FFT and 1 for Inverse FFT.

```
#define FFT_FLAG    ( 0 )   /* HWAFFT to perform FFT  */
#define IFFT_FLAG   ( 1 )   /* HWAFFT to perform IFFT */
Uint16 fft_flag:
```

fft_flag = FFT_FLAG:	FFT Performed
fft_flag = IFFT_FLAG:	Inverse FFT Performed

Uint16 scale_flag

The automatic scaling (divide each butterfly output by 2) feature is controlled by setting the scale_flag to 0 to enable scaling and 1 to disable scaling.

```
#define SCALE_FLAG      ( 0 )   /* HWAFFT to scale butterfly output     */
#define NOSCALE_FLAG    ( 1 )   /* HWAFFT not to scale butterfly output */
Uint16 scale_flag:
```

scale_flag = SCALE_FLAG:	Divide by 2 scaling is performed at the output of each FFT Butterfly.
scale_flag = NOSCALE_FLAG:	No scaling is performed, overflow may occur if the input dynamic is too high.

Uint16 <Return Value>:

This is the Uint16 return value of the HWAFFT functions. After the HWAFFT function completes, the result will either be stored in the data vector or in the scratch vector, depending on the status of this return value. The return value is Boolean where 0 indicates that the result is stored in the data vector, and 1 indicates this scratch vector. The program must check the status of the Return Value to determine the location of the FFT/IFFT result.

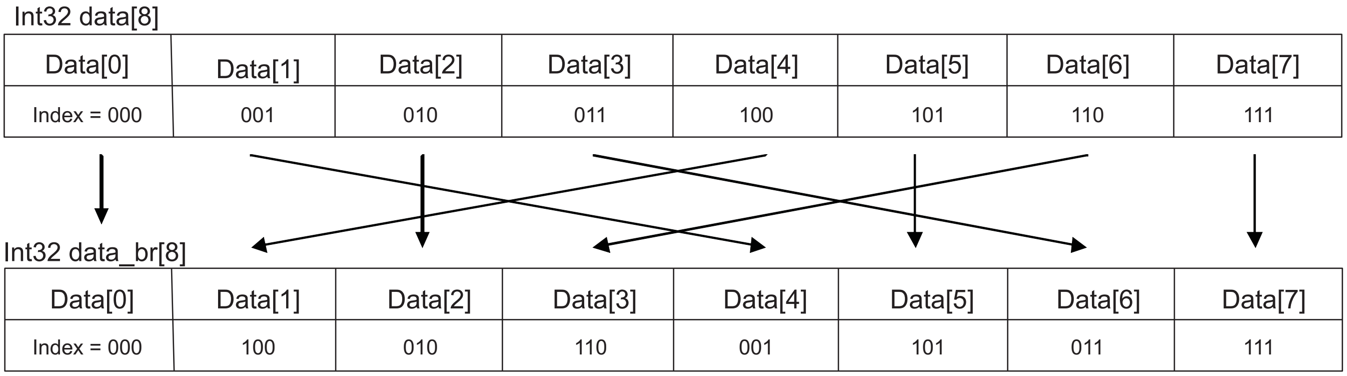
```
#define OUT_SEL_DATA    ( 0 )   /* indicates HWAFFT output located in input data vector   */
#define OUT_SEL_SCRATCH ( 1 )   /* indicates HWAFFT output located in scratch vector */
Uint16 <Return Value>:
```

Return Value = OUT_SEL_DATA:	FFT/IFFT result located in the data vector
Return Value = OUT_SEL_SCRATCH:	FFT/IFFT result located in the scratch vector

2.5.3 Bit Reverse Function

Before computing the FFT/IFFT on the HWAFFT, the input buffer must be bit-reversed to facilitate a Radix-2 DIT computation. This function contains optimized assembly that executes on the CPU to rearrange the Int32 elements of the input vector by placing each element in the destination vector at the index that corresponds to the bit-reversal of its current index. For example, in an 8-element vector, the index of the third element is 011 in binary, then the bit-reversed index is 110 in binary or 6 in decimal, so the third element of the input vector is copied to the sixth element of the bit-reversal destination vector.

Figure 2-5. Bit Reversed Input Buffer



Int32 data[8]

Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]
Index = 000	001	010	011	100	101	110	111

Int32 data_br[8]

Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]
Index = 000	100	010	110	001	101	011	111

2.5.3.1 Bit Reverse Destination Vector Alignment Requirement

Strict requirements are placed on the address of the bit-reversal destination buffer. This buffer must be aligned in RAM such that $\log_2(4 * N)$ zeros appear in the least significant bits of the byte address (8 bits), where N is the FFT Length. For example, a 1024-point FFT needs to bit-reverse 1024 complex array elements (32-bit elements). The address for the bit-reversed buffer needs to have 12 zeros in the least significant bits of its byte address ($\log_2(4 * 1024) = 12$). Since the word address (16 bits) is the byte address shifted right one bit, the word address requires 11 zeros in the least significant bits. This bit-reverse is considered out-of-place because the inputs and outputs are stored in different vectors. In-place bit-reversal is not supported by this function. There are no alignment requirements for the bit-reverse source vector.

2.5.3.2 Bit Reverse Format and Parameters

The structure of the HWAFFT functions is:

`void hwafft_br(`	Performs out-of-place bit-reversal on 32-bit data vector
`    Int32 *data,`	Input – 32-bit data vector
`    Int32 *data_br,`	Output – bit-reversed data vector
`    Uint16 data_len,`	Length of complex data vector
`);`	

The parameters for the hwafft_br function are:

Int32 *data

This is the input vector to the bit reverse function. It contains complex data elements (real part in most significant 16 bits, imaginary part in least significant 16 bits). There are no specific alignment requirements for this vector.

```
#pragma DATA_SECTION(data_buf, "data_buf");
     //Static Allocation to Section: "data_buf  : > DARAM" in Linker CMD File
Int32 data_buf[N = FFT Length];
Int32 *data = data_buf;
```

Int32 *data_br

This is the destination vector of the bit-reverse function. It contains complex data elements (real part in most significant 16 bits, imaginary part in least significant 16 bits). A strict alignment requirement is placed on this destination vector of the bit-reverse function: This buffer must be aligned in RAM such that $\log_2(4 * N)$ zeros appear in the least significant bits of the byte address (8 bits), where N is the FFT Length. See Section 2.10, *Appendix A Methods for Aligning the Bit-Reverse Destination Vector*, for ways to force the linker to enforce this alignment requirement.

```
#define ALIGNMENT 2*N  // ALIGNS data_br_buf to an address with log2(4*N) zeros in the
                          // least significant bits of the byte address
#pragma DATA_SECTION(data_br _buf, "data_br_buf");
                          // Allocation to Section: "data_br _buf : > DARAM" in Linker CMD File
#pragma DATA_ALIGN (data_br_buf, ALIGNMENT);
Int32 data_br_buf[N = FFT Length];
Int32 * data_br = data_br_buf;
```

Int32 *data_br:

Strict address alignment requirement: This buffer must be aligned in RAM such that ($\log_2(4 * N)$ zeros appear in the least significant bits of the byte address (8 bits), where N is the FFT Length. See Section 2.10 for ways to force the linker to enforce this alignment requirement.

Uint16 *data_len

This Uint16 parameter indicates the length of the data and data_br vectors.

Uint16 data_len:

The data_len parameter indicates the length of the Int32 vector (FFT Length). Valid lengths include powers of two: {8, 16, 32, 64, 128, 256, 512, 1024}.

2.5.4 Function Descriptions and ROM Locations

Table 2-2 shows the available HWAFFT routines with descriptions and respective addresses in ROM.

Table 2-2. Available HWAFFT Routines

Function Name	Description	ROM Address
hwafft_br	Int32 (32-bit) vector bit-reversal, Strict alignment requirement	0x00fefe9c
hwafft_8pts	8-point FFT/IFFT, 1 double-stage, 1 single-stage	0x00fefeb0
hwafft_16pts	16-point FFT/IFFT, 2 double-stages, 0 single-stages	0x00feff9f
hwafft_32pts	32-point FFT/IFFT, 2 double-stages, 1 single-stage	0x00ff00f5
hwafft_64pts	64-point FFT/IFFT, 3 double-stages, 0 single-stages	0x00ff03fe
hwafft_128pts	128-point FFT/IFFT, 3 double-stages, 1 single-stage	0x00ff0593
hwafft_256pts	256-point FFT/IFFT, 4 double-stages, 0 single-stages	0x00ff07a4
hwafft_512pts	512-point FFT/IFFT, 4 double-stages, 1 single-stage	0x00ff09a2
hwafft_1024pts	1024-point FFT/IFFT, 5 double-stages, 0 single-stages	0x00ff0c1c

2.5.5 Project Configuration for Calling Functions from ROM

The HWAFFT functions occupy approximately 4 KBytes of memory, so to conserve RAM they have been placed in the DSP’s 128 KBytes of on-chip ROM. These functions are identical to and have the same names as the functions stored in hwafft.asm, but they do not consume any RAM. In order to utilize these HWAFFT routines in ROM, add the following lines to the bottom of the project’s linker CMD file and remove the hwafft.asm file from the project (or exclude it from the build). When the project is rebuilt, the HWAFFT functions will reference the ROM locations. The HWAFFT ROM locations for the DSP is shown in Table 2-2.

```
/*** Add the following code to the linker command file to call HWAFFT Routines from ROM ***/

/* HWAFFT Routines ROM Addresses */
_hwafft_br = 0x00fefe9c;
_hwafft_8pts = 0x00fefeb0;
_hwafft_16pts = 0x00feff9f;
_hwafft_32pts = 0x00ff00f5;
_hwafft_64pts = 0x00ff03fe;
_hwafft_128pts = 0x00ff0593;
_hwafft_256pts = 0x00ff07a4;
_hwafft_512pts = 0x00ff09a2;
_hwafft_1024pts = 0x00ff0c1c;
```

2.6 Simple Example to Illustrate the Use of the FFT Accelerator

The source code below demonstrates typical use of the HWAFFT for the 1024-point FFT and IFFT cases.

The HWAFFT Functions make use of Boolean flag variables to select between FFT and IFFT, Scale and No Scale mode, and Data and Scratch output locations.

```
#define FFT_FLAG        ( 0 )   /* HWAFFT to perform FFT */
#define IFFT_FLAG       ( 1 )   /* HWAFFT to perform IFFT */
#define SCALE_FLAG      ( 0 )   /* HWAFFT to scale butterfly output */
#define NOSCALE_FLAG    ( 1 )   /* HWAFFT not to scale butterfly output */
#define OUT_SEL_DATA    ( 0 )   /* Indicates HWAFFT output located in input data vector */
#define OUT_SEL_SCRATCH ( 1 )   /* Indicates HWAFFT output located in scratch vector */
Int32 *data;
Int32 *data_br;
Uint16 fft_flag;
Uint16 scale_flag;
Int32 *scratch;
Uint16 out_sel;
Int32 *result;
```

2.6.1 *1024-Point FFT, Scaling Disabled*

Compute 1024-point FFT with Scaling enabled: a ½ scale factor after every stage:

```
fft_flag = FFT_FLAG;
scale_flag = SCALE_FLAG;

data = <1024-point Complex input>;

/* Bit-Reverse 1024-point data, Store into data_br, data_br aligned to
   12-least significant binary zeros*/
hwafft_br(data, data_br, DATA_LEN_1024);  /* bit-reverse input data,
                                             Destination buffer aligned */
data = data_br;

/* Compute 1024-point FFT, scaling enabled. */
out_sel = hwafft_1024pts(data, scratch, fft_flag, scale_flag);

if (out_sel == OUT_SEL_DATA) {
   result = data;
}else {
    result = scratch;
}
```

2.6.2 *1024-Point IFFT, Scaling Disabled*

Compute 1024-point IFFT with Scaling disabled:

```
fft_flag = IFFT_FLAG;
scale_flag = NOSCALE_FLAG;

data = <1024-point Complex input>;

/* Bit-Reverse 1024-point data, Store into data_br, data_br aligned to
   12-least significant binary zeros */
hwafft_br(data, data_br, DATA_LEN_1024);
data = data_br;

/* Compute 1024-point IFFT, scaling disabled */
out_sel = hwafft_1024pts(data, scratch, fft_flag, scale_flag);

if (out_sel == OUT_SEL_DATA) {
   result = data;
} else {
    result = scratch;
}
```

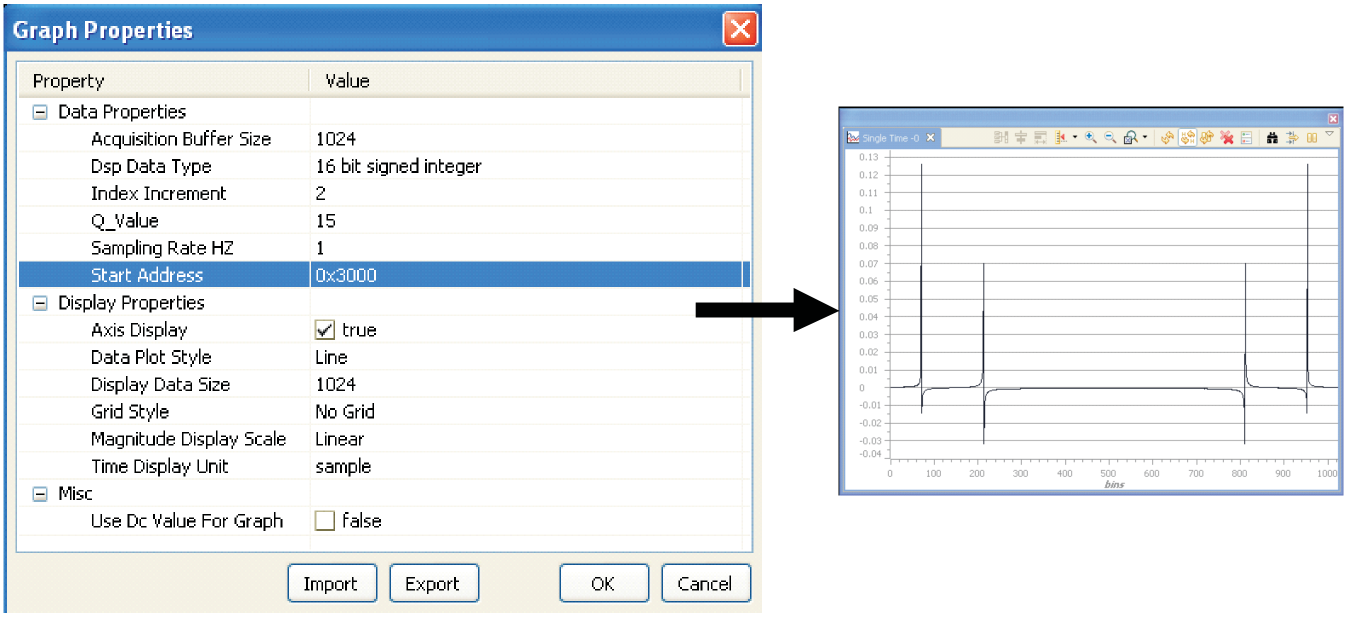
2.6.3 Graphing FFT Results in CCS4

Code Composer includes a graphing utility that makes visualization of the FFT operation quick and easy. The Graph Utility is located in the CCSv4 window, under Tools → Graph → Single Time.

If the FFT Result is stored in scratch (OutSel = 1) and scratch is located at address 0x3000…

Plot the real part:

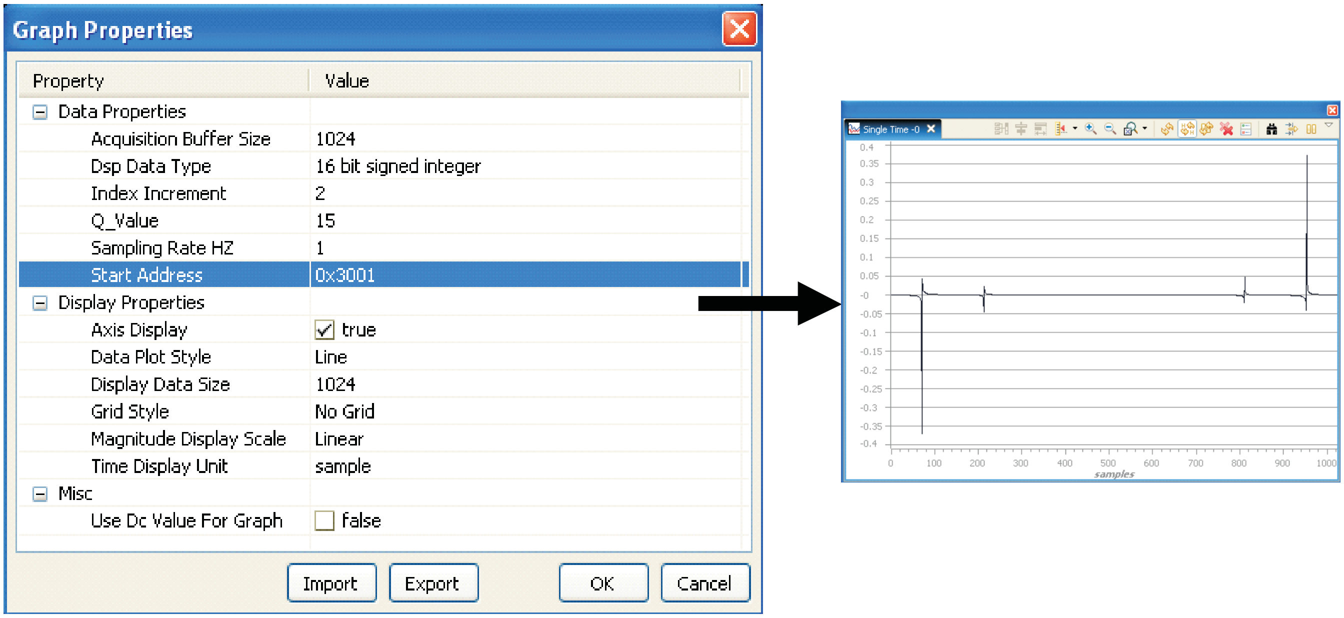
Figure 2-6. Graphing the Real Part of the FFT Result in CCS4



Property	Value
⊟ Data Properties	
Acquisition Buffer Size	1024
Dsp Data Type	16 bit signed integer
Index Increment	2
Q_Value	15
Sampling Rate HZ	1
Start Address	0x3000
⊟ Display Properties	
Axis Display	☑ true
Data Plot Style	Line
Display Data Size	1024
Grid Style	No Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
⊟ Misc	
Use Dc Value For Graph	☐ false

Plot the imaginary part:

Figure 2-7. Graphing the Imaginary Part of the FFT Result in CCS4



Property	Value
⊟ Data Properties	
Acquisition Buffer Size	1024
Dsp Data Type	16 bit signed integer
Index Increment	2
Q_Value	15
Sampling Rate HZ	1
Start Address	0x3001
⊟ Display Properties	
Axis Display	☑ true
Data Plot Style	Line
Display Data Size	1024
Grid Style	No Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
⊟ Misc	
Use Dc Value For Graph	☐ false

2.7 FFT Benchmarks

Table 2-3 compares the FFT performance of the HWAFFT versus FFT computation using the CPU under the following conditions:

- Core voltage = 1.05 V
- PLL = 50 MHz
- Power measurement condition:
  - At room temperature only
  - All peripherals are clock gated
  - Measured at Vddc

Table 2-3. FFT Performance on HWAFFT vs CPU (Vcore = 1.05 V, PLL = 50 MHz)

	FFT with HWA		CPU (Scale)		HWA versus CPU	
Complex FFT	**FFT + BR [1] Cycles**	**Energy/FFT (nJ/FFT)**	**FFT + BR [1] Cycles**	**Energy/FFT (nJ/FFT)**	**x Times Faster (Scale)**	**x Times Energy Efficient (Scale)**
8 pt	92 + 38 = 130	23.6	196 + 95 = 291	95.1	2.2	4
16 pt	115 + 55 = 170	32.1	344 + 117 = 461	157.1	2.7	4.9
32 pt	234 + 87 = 321	69.5	609 + 139 = 748	269.9	2.3	3.9
64 pt	285 + 151 = 436	98.5	1194 + 211 = 1405	531.7	3.2	5.4
128 pt	633 + 279 = 912	219.2	2499 + 299 = 2798	1090.4	3.1	5
256 pt	1133 + 535 = 1668	407.2	5404 + 543 = 5947	2354.2	3.6	5.8
512 pt	2693 + 1047 = 3740	939.7	11829 + 907 = 12736	5097.5	3.4	5.4
1024 pt	5244 + 2071 = 7315	1836.2	25934 + 1783 = 27717	11097.9	3.8	6

[1] BR = Bit Reverse

In summary, Table 2-3 shows that for the test conditions used, HWAFFT is 4 to 6 times more energy efficient and 2.2 to 3.8 times faster than the CPU.

Table 2-4 compares FFT performance of the accelerator versus FFT computation using the CPU under the following conditions:

- Core voltage = 1.3 V
- PLL = 100 MHz
- Power measurement condition:
  - At room temperature only
  - All peripherals are clock gated
  - Measured at Vddc

Table 2-4. FFT Performance on HWAFFT vs CPU (Vcore = 1.3 V, PLL = 100 MHz)

	FFT with HWA		CPU (Scale)		HWA versus. CPU	
Complex FFT	**FFT + BR [1] Cycles**	**Energy/FFT (nJ/FFT)**	**FFT + BR [1] Cycles**	**Energy/FFT (nJ/FFT)**	**x Times Faster (Scale)**	**x Times Energy Efficient (Scale)**
8 pt	92 + 38 = 130	36.3	196 + 95 = 291	145.9	2.2	4
16 pt	115 + 55 = 170	49.3	344 + 117 = 461	241	2.7	4.9
32 pt	234 + 87 = 321	106.9	609 + 139 = 748	414	2.3	3.9
64 pt	285 + 151 = 436	151.3	1194 + 211 = 1405	815.7	3.2	5.4
128 pt	633 + 279 = 912	336.8	2499 + 299 = 2798	1672.9	3.1	5
256 pt	1133 + 535 = 1668	625.6	5404 + 543 = 5947	3612.9	3.6	5.8
512 pt	2693 + 1047 = 3740	1442.8	11829 + 907 = 12736	7823.8	3.4	5.4
1024 pt	5244 + 2071 = 7315	2820.6	25934 + 1783 = 27717	17032.4	3.8	6

[1] BR = Bit Reverse

In summary, Table 2-4 shows that for the test conditions used, HWAFFT is 4 to 6 times more energy efficient and 2.2 to 3.8 times faster than the CPU.

2.8 Description of Open Source FFT Example Software

An example application of the HWAFFT used in a real-time audio filter is available on the Open Source C5505 eZdsp website (http://www.code.google.com/p/c5505-ezdsp). The zip file named “VC5505 FFT Filter Demo” contains a Code Composer 4 Project and source code that implements a real-time low-pass filter on the VC5505 eZdsp USB Stick. Low-pass filtering is achieved through multiplication with a filter in the Frequency Domain. Recall that multiplication in the frequency domain is equivalent to convolution in the time domain.

In this demo, 16-bit stereo samples are captured by the AIC3204 codec at a sampling frequency of 48 kHz and copied to the DSP memory with the DMA over the Inter-IC Sound (I^2S) bus. Samples from the left and right channel are collected in separate ping-pong buffers. When the buffer becomes full a DMA interrupt updates the ping-pong buffer and triggers the FFT filter to convolve the new block of samples with the low-pass filter.

Filtering is performed in three steps:

1. Use the HWAFFT to calculate the FFT of the input block of samples from the ping-pong buffers.
2. Multiply this complex FFT result with the pre-computed FFT result of the filter coefficients.

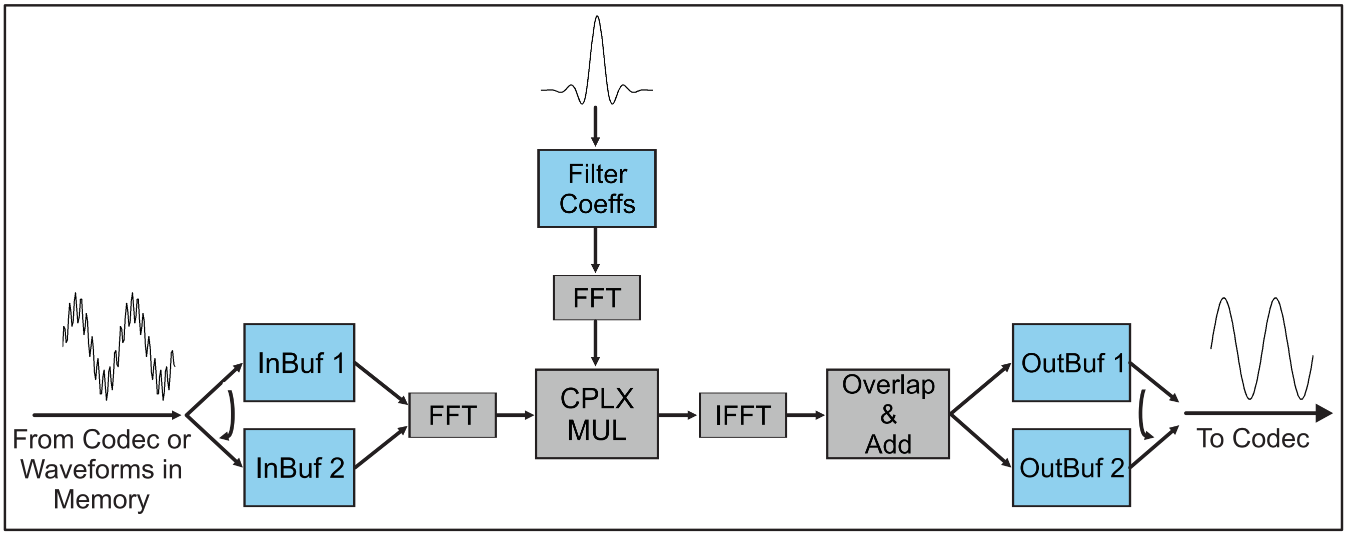
   Note: The FFT result of the filter coefficients is computed once during program initialization and stored for reuse.
3. Calculate the IFFT of that product on the HWAFFT.

Because a stream of samples is constantly arriving at the codec and block processing is utilized to filter the signal, the Constant-Overlap-and-Add (COLA) method is implemented to output a continuous, glitch-free signal to the codec. Finally, the resulting block of filtered and overlapped samples is transferred back to the codec for output with the DMA over the I^2S bus.

This demo provides you control over FFT Lengths (from 8 to 1024 points) and Filter Lengths (from 7 taps to 511 taps) for a thorough comparison. Additionally, simulation modes are available for using ideal sinusoidal signals (stored in memory) as inputs to the FFT Filter Demo.

The block diagram in Figure 2-8 shows the data flow for one channel of the FFT Filter Demo. When processing stereo input (separate left and right channels), this data flow is duplicated for each channel.

Figure 2-8. FFT Filter Demo Block Diagram



2.9 Computation of Large (Greater Than 1024-Point) FFTs

The HWAFFT can perform up to 1024-point complex FFTs/IFFTs at a maximum, but if larger FFT sizes (i.e. 2048-point) are required, the DSP core can be used to compute extra Radix-2 stages that are too large for the HWAFFT to handle.

Recall the Radix-2 DIT equations:

$$X(k) = \frac{1}{2}(X_{even}(k) + W_N^k X_{odd}(k)), k = 0 \text{ to } N/2 - 1 \tag{11}$$

and

$$X(k + N/2) = \frac{1}{2}(X_{even}(k) - W_N^k X_{odd}(k)), k = 0 \text{ to } N/2 - 1 \tag{12}$$

2.9.1 Procedure for Computing Large FFTs

The procedure for computing an additional Radix-2 DIT stage on the CPU is outlined:

- Split the input signal into even and odd indexed signals, X_{even} and X_{odd}.
- Call N/2 point FFTs for the even and odd indexed inputs.
- Complex Multiply the X_{odd} FFT results with the decimated twiddle factors for that stage.
- Add the X_{odd} * Twiddle product to Xeven to find the first half of the FFT result.
- Subtract X_{odd} * Twiddle to find the second half of the FFT result.

2.9.2 Twiddle Factor Computation

The HWAFFT stores 512 complex twiddle factors enabling FFT/IFFT computations up to 1024 points. Recall Equation 5 states that only twiddle factors from 0 to N/2 are needed. To compute FFT/IFFTs larger than 1024 points, you must supply N/2 complex twiddle factors, where N is the FFT length (powers of 2).

The following MATLAB code creates real and imaginary parts of the twiddle factors for any N:

```
N = 2048;
n = 0:(N/2-1);
twid_r = cos(2*pi*n/N);
twid_i = -sin(2*pi*n/N);
```

2.9.3 Bit-Reverse Separates Even and Odd Indexes

A nice property of the bit-reversal process is the automatic separation of odd-indexed data from even-indexed data. Before the bit-reverse, even indexes have a 0 in the least significant bit and odd indexes have a 1 in the least significant bit. After the bit-reverse, even indexes have a 0 in the most significant bit, and odd indexes have a 1 in the most significant bit. Therefore, all even indexed data resides in the first half of the bit-reversed vector, and all odd indexed data resides in the second half of the bit-reversed vector. This process meets two needs: separation of even and odd indexed-data vectors and bit-reversing both vectors.

2.9.4 2048-point FFT Source Code

The following C source code demonstrates a 2048-point FFT using this approach. Two 1024-point FFTs are computed on the HWAFFT, and a final Radix-2 stage is performed on the CPU to generate a 2048-point FFT result:

```
#define FFT_FLAG          ( 0 )       /* HWAFFT to perform FFT */
#define IFFT_FLAG         ( 1 )       /* HWAFFT to perform IFFT */
#define SCALE_FLAG        ( 0 )       /* HWAFFT to scale butterfly output */
#define NOSCALE_FLAG      ( 1 )       /* HWAFFT not to scale butterfly output */
#define OUT_SEL_DATA      ( 0 )       /* Indicates HWAFFT output located in input data vector */
#define OUT_SEL_SCRATCH   ( 1 )       /* Indicates HWAFFT output located in scratch vector   */
#define DATA_LEN_2048     ( 2048 )
#define TEST_DATA_LEN     (DATA_LEN_2048)

// Static Memory Allocations and Alignment:
#pragma DATA_SECTION(data_br_buf, "data_br_buf");
```

```
#pragma DATA_ALIGN (data_br_buf, 4096);
   // Align 2048-pt bit-reverse dest vector to byte addr w/ 13 least sig zeros
Int32 data_br_buf[TEST_DATA_LEN];

#pragma DATA_SECTION(data_even_buf, "data_even_buf");
Int32 data_even_buf[TEST_DATA_LEN/2];

#pragma DATA_SECTION(data_odd_buf, "data_odd_buf");
Int32 data_odd_buf[TEST_DATA_LEN/2];

#pragma DATA_SECTION(scratch_even_buf, "scratch_even_buf");
Int32 scratch_even_buf[TEST_DATA_LEN/2];

#pragma DATA_SECTION(scratch_odd_buf, "scratch_odd_buf");
Int32 scratch_odd_buf[TEST_DATA_LEN/2];

// Function Prototypes:
Int32 CPLX_Mul(Int32 op1, Int32 op2);
   // Yr = op1_r*op2*r - op1_i*op2_i, Yi = op1_r*op2_i + op1_i*op2_r
Int32 CPLX_Add(Int32 op1, Int32 op2, Uint16 scale_flag);
   // Yr = 1/2 * (op1_r + op2_r), Yi = 1/2 *(op1_i + op2_i)
Int32 CPLX_Subtract(Int32 op1, Int32 op2, Uint16 scale_flag);
   // Yr = 1/2 * (op1_r - op2_r), Yi = 1/2 *(op1_i - op2_i)

// Declare Variables
Int32 *data_br;
Int32 *data;
Int32 *data_even, *data_odd;
Int32 *scratch_even, *scratch_odd;
Int32 *twiddle;
Int32 twiddle_times_data_odd;
Uint16 fft_flag;
Uint16 scale_flag;
Uint16 out_sel;
Uint16 k;

// Assign pointers to static memory allocations
data_br = data_br_buf;
data_even = data_even_buf;
data_odd = data_odd_buf;
scratch_even = scratch_even_buf;
scratch_odd = scratch_odd_buf;
twiddle = twiddle_buf;     // 1024-pt Complex Twiddle Table
data = invec_fft_2048pts;  // 2048-pt Complex Input Vector

// HWAFFT flags:
fft_flag = FFT_FLAG;       // HWAFFT to perform FFT (not IFFT)
scale_flag = SCALE_FLAG;   // HWAFFT to scale by 2 after each butterfly stage

// Bit-reverse input data for DIT FFT calculation
hwafft_br(data, data_br, DATA_LEN_2048);
   // data_br aligned to log2(4*2048) = 13 zeros in least sig bits
data = data_br;

// Split data into even-indexed data & odd-indexed data
// data is already bit-reversed, so even-indexed data = first half & odd-
indexed data = second half
for(k=0; k<DATA_LEN_2048/2; k++)
{
    data_even[k] = data[k];
    data_odd[k]  = data[k+DATA_LEN_2048/2];
}

// 1024-pt FFT the even data on the FFT Hardware Accelerator
out_sel = hwafft_1024pts(data_even, scratch_even, fft_flag, scale_flag);
if(out_sel == OUT_SEL_SCRATCH) data_even = scratch_even;
```

```
// 1024-pt FFT the odd data on the FFT Hardware Accelerator
out_sel = hwafft_1024pts(data_odd, scratch_odd, fft_flag, scale_flag);
if(out_sel == OUT_SEL_SCRATCH) data_odd = scratch_odd;

// Combine the even and odd FFT results with a final Radix-2 Butterfly stage on the CPU
for(k=0; k<DATA_LEN_2048/2; k++)  // Computes 2048-point FFT
{
    // X(k)     = 1/2*(X_even[k] + Twiddle[k]*X_odd(k))
    // X(k+N/2) = 1/2*(X_even[k] - Twiddle[k]*X_odd(k))
    // Twiddle[k]*X_odd(k):

    twiddle_times_data_odd = CPLX_Mul(twiddle[k], data_odd[k]);

    // X(k):
    data[k] = CPLX_Add(data_even[k], twiddle_times_data_odd, SCALE_FLAG);  // Add then scale by 2

    // X(k+N/2):
    data[k+DATA_LEN_2048/2] = CPLX_Subtract(data_even[k], twiddle_times_data_odd, SCALE_FLAG);
         //Sub then scale
}

result = data;   //2048-pt FFT result

/* END OF 2048-POINT FFT SOURCE CODE */
```

2.10 Appendix A Methods for Aligning the Bit-Reverse Destination Vector

The optimized bit-reverse function hwafft_br requires the destination vector to be data aligned such that the starting address of the destination vector, data_br, contains $\log_2(4 * N)$ zeros in the least significant bits of the binary address. There are a few different ways to force the linker map the bit-reverse destination vector to an address with $\log_2(4 * N)$ zeros in the least significant bits. Three different methods are shown here. For further details, refer to the *TMS320C55x C/C++ Compiler User's Guide* (SPRU280).

2.10.1 Statically Allocate Buffer at Beginning of Suitable RAM Block

Place the buffer at the beginning of a DARAM or SARAM block with $\log_2(4 * N)$ zeros in the least significant bits of its byte address. For example, memory section DARAM2_3 below starts at address 0x0004000, which contains 14 zeros in the least significant bits of its binary address (0x0004000 = 0b0100 0000 0000 0000). Therefore, this address is a suitable bit-reverse destination vector for FFT Lengths up to 4096-points because $\log_2(4 * 4096) = 14$.

In the Linker CMD File...

```
MEMORY
{
  MMR    (RWIX): origin = 0000000h, length = 0000c0h    /* MMRs */
  DARAM0 (RWIX): origin = 00000c0h, length = 001f40h    /* on-chip DARAM 0,   4000 words */
  DARAM1 (RWIX): origin = 0002000h, length = 002000h    /* on-chip DARAM 1,   4096 words */
  DARAM2_3 (RWIX): origin = 0004000h, length = 004000h  /* on-chip DARAM 2_3, 8192 words */
  DARAM4 (RWIX): origin = 0008000h, length = 002000h    /* on-chip DARAM 4,   4096 words */
  ... (leaving out rest of memory sections)
}

SECTIONS
{
   data_br_buf : > DARAM2_3  /* ADDR = 0x004000, Aligned to addr with 14 least-sig zeros */
}
```

2.10.2 Use the ALIGN Descriptor to Force $\log_2(4 * N)$ Zeros in the Least Significant Bits

The ALIGN descriptor forces the alignment of a specific memory section, while providing the linker with added flexibility to allocate sections across the entire DARAM or SARAM because no blocks are statically allocated. It aligns the memory section to an address with $\log_2$(ALIGN Value) zeros in the least significant bits of the binary address.

For example, the following code aligns data_br_buf to an address with 12 zeros in the least significant bits, suitable for a 1024-point bit-reverse destination vector.

In the Linker CMD File...

```
MEMORY
{
  MMR    (RWIX): origin = 0000000h, length = 0000c0h  /* MMRs */
  DARAM (RWIX): origin = 00000c0h, length = 00ff40h   /* on-chip DARAM  32 Kwords */
  SARAM  (RWIX): origin = 0010000h, length = 040000h  /* on-chip SARAM 128 Kwords */
}

SECTIONS
{
   data_br_buf   : > DARAM   ALIGN = 4096
                         /* 2^12 = 4096 , Aligned to addr with 12 least-sig zeros */
}
```

2.10.3 Use the DATA_ALIGN Pragma

The DATA_ALIGN pragma is placed in the source code where the vector is defined. The syntax is shown below.

#pragma DATA_ALIGN (*symbol***,** *constant***);**

The DATA_ALIGN pragma aligns the symbol to an alignment boundary. The boundary is the value of the constant in words. For example, a constant of 4 specifies a 64-bit alignment. The constant must be a power of 2.

In this example, a constant of 2048 aligns the data_br_buf symbol to an address with 12 zeros in the least significant bits, suitable for a 1024-point bit-reverse destination vector.

In the source file where data_br is declared (e.g., main.c):

```
#pragma DATA_SECTION(data_br_buf, "data_br_buf");
#pragma DATA_ALIGN (data_br_buf, 2048);
Int32 data_br_buf[TEST_DATA_LEN];
```

Chapter 3

SPRUH87C – August 2011 – Revised March 2012

Direct Memory Access (DMA) Controller

This chapter describes the features and operations of the direct memory access (DMA) controller.

Topic **Page**

3.1 Introduction

The following sections describe the features and operation of the direct memory access (DMA) controller in the digital signal processor (DSP). The DMA controller allows movement of data between internal/external memory and other peripherals without CPU intervention.

3.1.1 Purpose of the DMA Controller

The DMA controller is used to move data among internal memory, external memory, and peripherals without intervention from the CPU and in the background of CPU operation.

The DSP includes four DMA controllers with four DMA channels each for a total of 16 DMA channels. Aside from the DSP resources they can access, all four DMA controllers are identical. Throughout this document the general operation of each DMA controller will be described. Differences between each DMA controller will be noted when necessary.

3.1.2 Key Features of the DMA Controller

The DMA controller has the following features:

- Operation that is independent of the CPU.
- Four channels per DMA controller, which allow the DMA controller to keep track of the context of four independent block transfers.
- Event synchronization. DMA transfers in each channel can be made dependent on the occurrence of selected events. For details, see Section 3.2.7.
- An interrupt for each channel. Each channel can send an interrupt to the CPU on completion of the programmed transfer. See Interrupt Support in Section 3.2.14.
- A dedicated clock idle domain. The user can put the four device DMA controllers into a low-power state by turning off their input clock. See Power Management in Section 3.2.15.
- Ping-Pong mode for DMA transfer. This mode provides double buffering capability fully implemented in hardware. For details, see Section 3.2.9.

To read about the registers used to program the DMA controller, see Section 3.4.

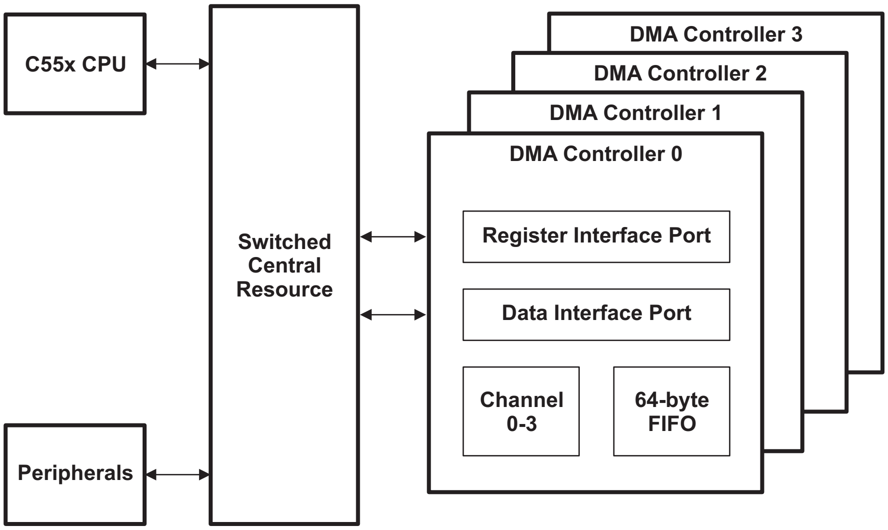
3.1.3 Block Diagram of the DMA Controller

Figure 3-1 is a conceptual diagram of connections between the DMA controller and other parts of the DSP. The DMA controller is made up of the following blocks:

- Register interface port. The CPU uses this port to access the DMA controller registers.
- Data interface port. The DMA controller accesses internal dual-access RAM (DARAM), internal single-access RAM (SARAM), external memory, and on-chip peripherals through its data interface port.
- Data transfers are carried out by the four DMA channels. (The DMA channels are described in Section 3.2.3)
- 64-byte FIFO. As data is read from the source address, it is placed in the DMA controller FIFO. The four DMA channels must share the DMA controller FIFO; the FIFO can only be accessed by a single channel at a time.

It is possible for multiple channels to request access to the DMA controller FIFO at the same time. In this case the DMA controller arbitrates amongst the DMA channels using a round-robin arbitration scheme.

Figure 3-1. Conceptual Block Diagram of the DMA Controller



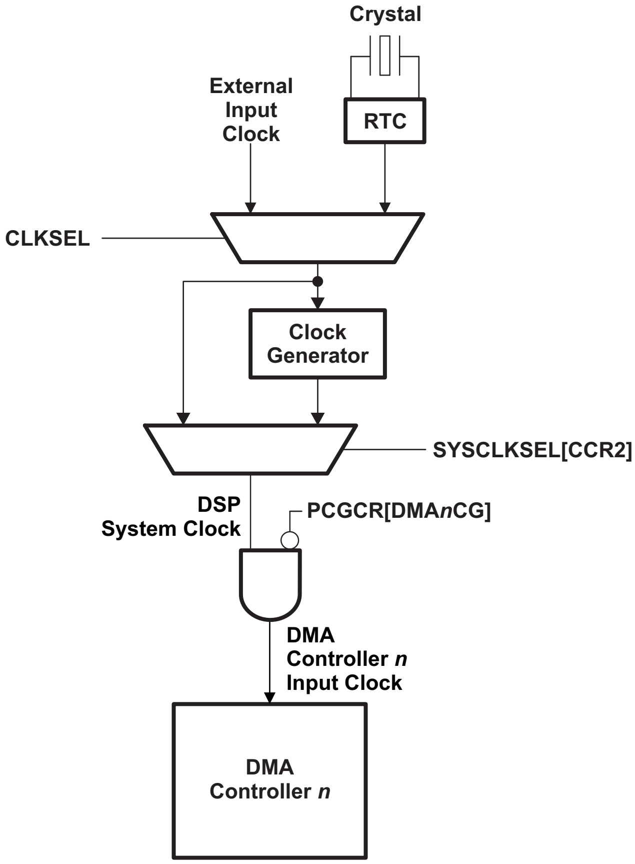
3.2 DMA Controller Architecture

3.2.1 Clock Control

As shown in Figure 3-2, the clock generator receives either the real-time clock (RTC) or a signal from an external clock source and produces the DSP system clock. This clock is used by the DSP CPU and peripherals.

The DSP includes logic which can be used to gate the clock to its on-chip peripherals, including each of the four DMA controllers. The input clock to the DMA controllers can be enabled and disabled through the peripheral clock gating configuration registers (PCGCR1 and PCGCR2).

Figure 3-2. Clocking Diagram for the DMA Controller



3.2.2 Memory Map

On the DSP, although all DMA controllers can access internal dual-access RAM (DARAM) and single-access RAM (SARAM), each DMA controller can only access a subset of on-chip peripherals. Also, only DMA controller 3 has access to external memory. The addresses from the point of view of the CPU as compared to the DMA controller are different for DARAM, SARAM and external memory. The DMA controller reads on-chip and off-chip memory by adding the offsets introduced in Table 3-1. The memory map, as seen by the DMA controllers and the CPU, is shown in Table 3-1. Peripherals not shown in Table 3-1 are not accessible by the DMA controllers.

Table 3-1. DMA Controller Memory Map

DMA Start *Byte* **Address**	**CPU Start** *Word* **Address (I/O Space)**	**CPU Start** *Word* **Address (Data Space)**	**DSP Memory Map**	**DMA Controller 0 Memory Map**	**DMA Controller 1 Memory Map**	**DMA Controller 2 Memory Map**	**DMA Controller 3 Memory Map**
0000 2800h	00 2800h	-	I2S0	I2S0	Reserved	Reserved	Reserved
0000 3A00h	00 3A00h	-	MMC/SD0	MMC/SD0			
0000 3B00h	00 3B00h	-	MMC/SD1	MMCSD1			
0001 0000h*	-	00 0000h [1]	DARAM	DARAM	DARAM	DARAM	DARAM
0009 0000h	-	00 8000h	SARAM	SARAM	SARAM	SARAM	SARAM
0000 1B00h	00 1B00h	-	UART	Reserved	UART	Reserved	Reserved
0000 2A00h	00 2A00h	-	I2S2		I2S2		
0000 1A00h	00 1A00h	-	I2C		Reserved	I2C	
0000 2B00h	00 2B00h	-	I2S3			I2S3	
0000 2900h	00 2900h	-	I2S1			Reserved	I2S1
0000 7000h	00 7000h	-	10-bit SAR			10-bit SAR	Reserved

[1] Word addresses 00 0000h-00 005Fh (which correspond to byte addresses 00 0000h-00 00BFh) are reserved for the memory-mapped registers (MMRs) of the DSP CPU.

3.2.3 DMA Channels

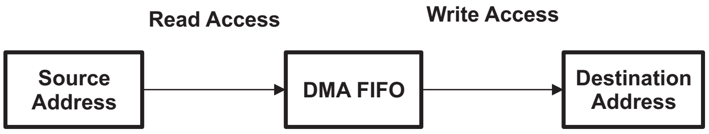
Each DMA controller has four channels to transfer data among the DSP resources (DARAM, SARAM, external memory, and peripherals). Each channel reads data from the source address and writes data to the destination address.

The DMA first in, first out (FIFO) buffer is used by the channels to store transfer data; this allows the data transfer to occur in two stages (see Figure 3-2).

Data read access Transfer of data from the source address to the DMA FIFO buffer.

Data write access Transfer of data from the DMA FIFO buffer to the destination address.

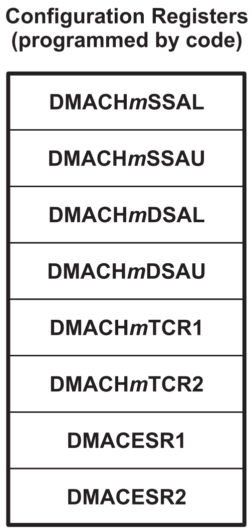
Figure 3-3. Two-Part DMA Transfer



The set of conditions under which transfers occur in a channel is called the channel context. Each of the four channels contains a register structure for programming and updating the channel context (see Figure 3-4). The user code modifies the configuration registers. The DMA channel becomes active when the channel is enabled (EN = 1 in DMACH*m*TCR2).

The channel configuration registers cannot be programmed while the channel is active (EN = 1 in DMACH*m*TCR2). Modifying channel registers while the DMA channel is running may cause unpredictable operation of the channel. To change a DMA channel configuration, the channel must first be disabled (EN = 0 in DMACH*m*TCR2). The DMA controller will always complete any on-going burst transfer before stopping channel activity. Note that a block transfer may consist of a number of burst transfers. The channel is considered to be active until it completes the burst transfer during which the channel is disabled. After a channel has been disabled, the channel context must be fully reloaded.

Figure 3-4. Registers for Controlling the Context of a Channel



3.2.4 Channel Source and Destination Start Addresses

During a data transfer in a DMA channel, the first address at which data is read is called the source start address. The first address to which the data is written is called the destination start address. These are byte addresses. Each channel contains the following registers for specifying the start addresses.

In Ping-Pong mode, the source or destination start address is the start address of the Ping buffer. The length of the **Ping** and the **Pong** buffers should be half of the DMA transfer size as programmed in TCR1. The first half is assumed to be the **Ping** buffer and the second half is assumed to be the **Pong** buffer. These two buffers are required to be contiguous in the memory space and are of equal size. The programmer is responsible for allocating these buffers contiguously. It is recommended to consider the **Ping** and the **Pong** buffers to be a single data buffer in the memory space so that the compiler always allocates them next to each other.

Table 3-2. Registers Used to Define the Start Addresses for a DMA Transfer

Register	Load with...
DMACH*m*SSAL	Source start address (least-significant part)
DMACH*m*SSAU	Source start address (most-significant part)
DMACH*m*DSAL	Destination start address (least-significant part)
DMACH*m*DSAU	Destination start address (most-significant part)

Section 3.2.2 shows a high-level memory map of the DSP as seen by the DMA controllers and the CPU. The table shows both the word addresses (23-bit addresses) used by the CPU and byte addresses (32-bit addresses) used by the DMA controller.

The following sections explain how to determine the start address for memory accesses and I/O accesses.

CAUTION

All data buffers in on-chip or off-chip memory should be 32-bit aligned. For more information on managing memory, see the *TMS320C55x Assembly Language Tools User Guide* (SPRU280).

Additionally, the amount of data (in bytes) to be transferred as programmed in the LENGTH field in DMACH*m*TCR1 should be a multiple of 4 bytes x $2^{BURSTMODE}$ field in DMACH*m*TCR2, i.e. LENGTH = (4 x $2^{BURSTMODE}$) bytes.

3.2.4.1 Start Address for On-Chip Memory

The CPU uses word addresses and the DMA uses byte addresses. Furthermore, an offset must be added to CPU addresses to generate DMA addresses.

Follow these steps to program the DMA controller with a byte address corresponding to a CPU word address:

1. Identify the correct start address. If you have a word address, shift it left by 1 bit to form a byte address of 32 bits. For example, the CPU word address for SARAM block 0 (00 8000h) should be converted to byte address 0001 0000h.
2. Add correct offset to the CPU byte address. For DARAM, add a value of 01 0000h to the desired byte address. For SARAM, add a value of 08 0000h. For example, since byte address 0001 0000h corresponds to SARAM block 0, a value of 0008 0000h should be added to the byte address to generate 0009 0000h.
3. Load the 16 least significant bits (LSBs) of the byte address into DMACH*m*SSAL (for source) or DMACH*m*DSAL (for destination).
4. Load the 16 most significant bits (MSBs) of the byte address into DMACH*m*SSAU (for source) or DMACH*m*DSAU (for destination).

3.2.4.2 Start Address for I/O Space

The CPU uses word addresses and the DMA uses byte addresses. The following steps describe how to program the DMA controller with a byte address for a peripheral memory-mapped register:

1. Identify the correct DMA byte address for the peripheral memory-mapped register. The starting DMA byte addresses for the memory-mapped register space of the DSP peripherals are given in Table 3-1.
2. Load the 16 least significant bits (LSBs) of the byte address into DMACH*m*SSAL (for source) or DMACH*m*DSAL (for destination).
3. Load the 16 most significant bits (MSB) of the byte address into DMACH*m*SSAU (for source) or DMACH*m*DSAU (for destination).

3.2.5 Updating Addresses in a Channel

During data transfers in a DMA channel, the DMA controller begins its read and write accesses at the start addresses you specify (as described in Section 3.2.4). Each time the DMA controller services a channel, it transfers the number of double words specified in the BURSTMODE of the channel's transfer control register (DMACH*m*TCR2). If constant addressing mode is selected (DST/SRCAMODE = 10b), the channel does not update the addressing registers (DMACSSA and DMACDSA). Otherwise, if the channel is set to automatic-post increment addressing mode (DST/SRCAMODE = 00b), the channel increments the value in the addressing registers by the total number of bytes transferred.

To change the source or destination address of a channel, the channel must first be disabled by setting EN = 0. The CPU can then update the Source/Destination Address registers and the Transfer Control registers before restarting the channel.

3.2.6 Data Burst Capability

During a read-write transaction (from source address to destination address) through the Switched Central Resource (see), the DMA controller moves data one double word at a time by default. Since every transaction request involves cycle overheads (see Section 3.2.11), the DMA throughput can be increased by programming the DMA channel to move multiple double words during a transaction, provided both the source and destination targets associated with the transfer support burst capability. The DMA controller can burst to and from SARAM, DARAM, and UART. For a list of DSP resources that support data bursting, see Table 3-3.

The BURSTMODE bits of the Transfer Control Register 2 (DMACH*m*TCR2) specify the number of double words the DMA controller moves each time it services a channel, i.e., the DMA controller executes a burst of n double words (n = 2, 4, 8, or 16) each time a channel is serviced instead of moving only 1 double word.

Note that the DMA controller services one channel at a time. Each time the DMA services a channel it must transfer the number of double words specified by the burst mode bits. Therefore, care must be taken when programming a channel to use a high burst count since this may impact the minimum amount of time it takes the DMA controller to service other channels. DMA channels are serviced in a round-robin fashion.

Table 3-3. Destinations/Sources That Support DMA Bursting

Destination/Source Address	Burst Mode Supported
DARAM	1, 2, 4, 8, 16 double words
SARAM	1, 2, 4, 8, 16 double words
UART [(1)]	1, 2, 4, 8 double words

[(1)] The UART treats each double word transfer as a single byte. Therefore, an 8 double word transfer from the DMA to the UART will yield 8 new bytes in the UART FIFO. For more information, see the *Universal Asynchronous Receiver/Transmitter (UART) Reference Guide* (SPRUFO5).

3.2.7 Synchronizing Channel Activity to DSP Peripheral Events

Activity in a channel can be synchronized to an event in a DSP peripheral. Synchronization is enabled by setting SYNCMODE = 1 in DMACH*m*TCR2. Using the CH*n*EVT bits of DMACESR1 and DMACESR2, the user can specify which synchronization event triggers channel activity. Note that synchronization to an event signaled by the driving of an external interrupt pin is not supported.

If event synchronization is enabled, the channel will wait for the event from the peripheral as programmed in the DMACESR1 and/or DMACESR2 registers before reading from the source address into the DMA FIFO. The synchronization event will trigger the channel to transfer the number of bytes specified by the BURSTMODE bits into the FIFO. Once the FIFO has been filled, the DMA channel will begin writing to the destination address to empty the FIFO.

In non-synchronized transfers (SYNCMODE = 0), the channel sends an access request to the source address as soon as the channel is enabled (EN = 1 in DMACH*m*TCR2). The channel will transfer data from the source address to the FIFO, and then to the destination address until the entire block transfer has been completed or until the user program disables the channel.

The synchronization events are not buffered. Hence, if a synchronization event occurs when the DMA channel is still servicing the previous event, the overlapping (second) event is dropped. In this scenario, the DMA controller does not disable the affected channel nor does it signal an error to the CPU. Therefore, care must be taken when programming the DSP to ensure that the DMA has enough clock cycles to transfer the required number of data bytes on each synchronization event.

Note that some peripherals can generate interrupts whenever a data underrun or overrun condition occurs. The user program can use these interrupts to detect dropped synchronization events.

CAUTION

When using synchronization events, you must set EN = 1 and SYNCMODE = 1 during the same write cycle to DMACH*m*TCR2. The DMA channel will transfer the first data value when it receives the synchronization event specified by CH*n*EVT in DMACESR1 and DMACESR2. Also, when disabling the channel, you must set EN = 0 and SYNCMODE = 0 during the same write cycle to DMACH*m*TCR2.

3.2.8 Channel Auto-Initialization Capability

After a block transfer is completed (all of the bytes specified by LENGTH in DMACH*m*TCR1 have been moved), the DMA controller automatically disables the channel (EN = 0). If it is necessary for the channel to be used again, the CPU can reprogram the new channel context and re-enable the DMA channel, or the DMA controller can automatically initialize the new context and re-enable the channel.

When auto-initialization is used, after each block transfer is completed, the DMA controller automatically reloads the transfer control register and the source and destination start address registers and re-enables the channel allowing the channel to run again. Auto-initialization is enabled by setting the AUTORLD = 1 in the transfer control register (DMACH*m*TCR2).

CAUTION

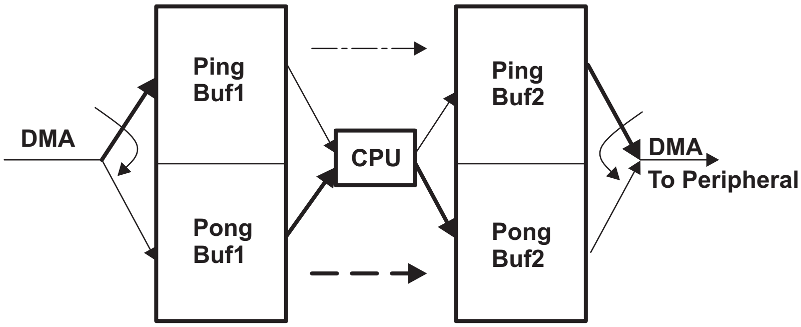
The auto-initialization feature can only be used when event synchronization is used (SYNCMODE = 1 in DMACH*m*TCR2).

Using auto-initialization feature without event synchronization can lead to unintended behavior of the DMA controller.

3.2.9 Ping-Pong DMA Mode

The Ping-Pong mode for DMA transfer can be used to do continuous processing of incoming data without losing any samples. Every channel in the DMA controller has the capability of being configured in the Ping-Pong mode. This mode can be initiated by setting PING_PONG_EN bit of the Transfer Control Register 2 (TCR2) to 1. For more information, see Section 3.4.3. You should also define the transfer buffer length in bytes in the Transfer Control Register 1 (TCR1). The length of the **Ping** and the **Pong** buffers should be half of the DMA transfer size as programmed in TCR1. The first half is assumed to be the **Ping** buffer and the second half is assumed to be the **Pong** buffer. These two buffers are required to be contiguous in the memory space and of equal size. The programmer is responsible for allocating these buffers contiguously. It is recommended to consider the **Ping** and the **Pong** buffers to be a single data buffer in the memory space so that the compiler always allocates them next to each other.

Figure 3-5. Ping-Pong Mode for DMA Data Transfer



As shown in Figure 3-5, in Ping-Pong mode, DMA starts filling up the **Ping** buffer first. Once the **Ping** buffer is full, a DMA interrupt is generated to the CPU and the LAST_XFER bit (bit 1 in TCR2) is set to 0, which indicates that the data in the **Ping** buffer can be processed. The CPU can start processing the samples in the **Ping** buffer while the DMA is transferring the data to the **Pong** buffer. When the **Pong** buffer is full, another DMA interrupt is sent to the CPU to indicate the availability of data in the **Pong** buffer and the LAST_XFER bit in TCR2 is set to 1. If the AUTORLD bit = 1 in TCR2 (Section 4.3), then the DMA automatically reinitiates the DMA transfer until either EN or AUTORLD bit is set to 0. In the case that the AUTORLD bit is set to 0, DMA stops data transfer after the **Pong** buffer is full and the interrupt is generated. It also resets the EN bit to 0 in TCR2.

At any time during the DMA transfer, LAST_XFER bit of TCR2 (Section 3.4.3) can be polled to find whether the last completed transfer was the **Ping** or **Pong** buffer.

3.2.10 Monitoring Channel Activity

The DMA controller can send an interrupt to the CPU whenever a channel has completed a block transfer. Each channel has an interrupt enable (DMA*n*CH*m*IE) bit in the interrupt enable register (DMAIER) and corresponding status bits in the interrupt flag register (DMAIFR). When a channel completes a block transfer, the DMA controller checks the corresponding DMA*n*CH*m*IE bit and acts accordingly:

- If the DMA*n*CH*m*IE bit is 1 (the interrupt is enabled), the DMA controller sets the corresponding status bit and sends the associated interrupt request to the CPU. Your program must manually clear bits in DMAIFR by writing a 1 to them.
- If the DMA*n*CH*m*IE bit is 0, no interrupt is sent and the status bit is not affected.

Each channel also includes a STATUS bit in DMACH*m*TCR2 to indicate the state of the channel transfer. The DMA controller sets the channel STATUS bit to 1 if:

- A nonzero value is written on LENGTH in DMACH*m*TCR1.
- A write access is performed to DMACH*m*TCR2 and LENGTH has a nonzero value.

The DMA controller clears the STATUS bit to 0 if:

- All the bytes specified by LENGTH in DMACH*m*TCR1 have been transferred.
- A value of 0 is written to LENGTH in DMACH*m*TCR1.

The LAST_XFER bit which is bit 1 in TCR2 (Section 3.4.3) indicates whether the last completed transfer was the **Ping** or the **Pong** buffer. This bit is valid only when Ping-Pong DMA mode is enabled (PING_PONG_EN bit in TCR2 is 1).

LAST_XFER bit in TCR2 is set to 0 if:

- The last completed transfer was the **Ping** buffer.

LAST_XFER bit in TCR2 is set to 1 if:

- The last completed transfer was the **Pong** buffer.

3.2.11 Latency in DMA Transfers

Each element transfer in a channel is composed of a read access (a transfer from the source location to the DMA controller FIFO) and a write access (a transfer from the DMA controller FIFO to the destination location). The time to complete this activity depends on factors such as:

- The selected frequency of the CPU clock signal. This signal, as propagated to the DMA controller, determines the timing for all DMA transfers.
- Wait states or other extra cycles added by or resulting from an interface.
- Activity on other channels. Since channels are serviced in a sequential order, the number of pending DMA service requests in the other channels affects how often a given channel can be serviced.
- Competition from the CPU or other DMA controllers. If a DMA controller and the CPU request access to the same internal memory block or peripheral in the same cycle and the memory block or peripheral cannot service both requests at the same time, the CPU request has higher priority. The DMA request is serviced as soon as there are no pending CPU requests.
- The timing of synchronization events (if the channel is synchronized). The DMA controller cannot service a synchronized channel until the synchronization event has occurred. For more details on synchronization, see Section 3.2.7.

The minimum (best-case) latency for a burst DMA transfer can be summarized as follows:

- For transfers initiating from internal memory: the first access for word read and write takes 8 cycles, while consecutive accesses take 2 more cycles. Thus the DMA takes 2N + 6 system clock cycles to complete a burst transfer, where N corresponds to the burst size in words.
- For transfers initiating from a peripheral source: the first access for word read and write takes 6 cycles, while consecutive accesses take 2 more cycles. Thus the DMA takes 2N + 4 system clock cycles to complete a burst transfer, where N corresponds to the burst size in words.

The burst size of the DMA is specified through the BURSTMODE bits. Note that a block transfer may consist of a number of burst transfers.

3.2.12 Reset Considerations

The DMA controller has one reset source: a hardware reset. This reset is always initiated during a full chip reset. Alternatively, software can force a hardware reset on all DMA controllers through the DMA_RST bit of the peripheral reset control register (PRCR). See the device data manual for more details on PRCR. Please note that the DMA controller input clock must be enabled when using DMA_RST (see Section 3.2.2).

When a hardware reset occurs, all the registers of the DMA controllers are set to their default values. The DMA controllers remain inactive until programmed by software.

3.2.13 Initialization

To initialize the DMA controller follow these steps:

1. Ensure the DMA controller is out of reset by setting the DMA_RST bit to 0 in the peripheral reset control register (PRCR). PRCR is a chip configuration register, it is not part of the DMA controller, see the device data manual for more details.
2. Enable the DMA controller input clock by setting the corresponding DMA*n*CG bit to 0 in the peripheral clock gating configuration registers (PCGCR1 and PCGCR2). PCGCR1 and PCGCR2 are chip configuration registers, they are not part of the DMA controller, see the device data manual for more details.
3. Ensure that all DMA channel interrupt flags are cleared by writing a 1 to the bits of the DMA interrupt flag register (DMAIFR). Also, ensure all DMA interrupt flags in the CPU interrupt flag registers (IFR0 and IFR1) are cleared.
4. If using interrupts, enable the desired channel interrupt by setting the DMA*n*CH*m*IE bits of the interrupt enable register (DMAIER). The CPU interrupt enable bit (INTEN) in the transfer control register 2 (DMACH*m*TCR2) must also be set.
5. If using synchronization events, select the event to be used through the CH*m*EVT bits of the channel event source registers (DMA*n*CESR1 and DMA*n*CESR2). The synchronization mode bit (SYNCMODE) of DMACH*m*TCR2 must also be set, although this should be done only when the channel is ready to be enabled.
6. Load the source address to the source start address registers (DMACH*m*SSAL and DMACH*m*SSAU). See Section 3.2.4, Start Address in a Channel, for more information on calculating the correct source start address.
7. Load the destination address to the destination start address registers (DMACH*m*DSAL and DMACH*m*DSAU). See Section 3.2.4, Start Address in a Channel, for more information on calculating the correct destination start address.
8. Load the DMA transfer control register 1 (DMACH*m*TCR1) with the number of double words to transfer. Note that the number of double words must be specified in bytes. For example, for a 256 double word transfer, program this field with 1024 (256 x 4 = 1024). When Ping-Pong DMA mode is enabled, this is the size of the Ping and the Pong buffer combined. For more details, see Section 3.2.4.
9. Configure DMACH*m*TCR2 accordingly. Through this register you can specify the source and destination addressing modes and burst mode. You can also enable automatic reload, event synchronization, CPU interrupts and Ping-Pong mode. Note that you must keep EN = 0 and SYNCMODE = 0 during this step.
10. If the DMA channel is servicing a peripheral, ensure that the corresponding peripheral is not active and hence not generating synchronization events.
11. Enable the DMA channel by setting EN = 1 (and SYNCMODE = 1 if using synchronization events).
12. If necessary, enable peripheral being serviced the DMA channel.

If using synchronization events, the DMA channel will start a data transfer when an event is received. Otherwise, the DMA channel will start the transfer immediately. At the end of the block transfer, if interrupts are enabled, the DMA controller will generate a CPU interrupt. If interrupts are not enabled, your program can poll DMACH*m*TCR1 until either EN or STATUS are cleared to 0 by the DMA controller to determine when the DMA has finished a block transfer. If AUTORLD is set the DMA controller will restart the specified transfer (for more details, see Section 3.2.8,).

For more specific examples of programming the DMA controller, see Section 3.3, DMA Transfer Examples.

NOTE: If a DMA controller is programmed to access on-chip memory, ensure that the MPORT is not idled in the Idle Configuration Register (ICR). Note that the the value programmed in the ICR takes effect only on running the 'idle' instruction on the CPU. For more information on these registers, see the *DSP System* Chapter.

3.2.14 Interrupt Support

3.2.14.1 Interrupt Events and Requests

Each of the four channels of a DMA controller has its own interrupt which the user can enable or disable a channel interrupt though the DMA*n*CH*m* bits of the DMA interrupt mask register (DMAIMR). The interrupts from the four DMA controllers are combined into a single CPU interrupt. You can determine which DMA channel generated the interrupt by reading the bits of the DMA interrupt flag register (DMAIFR). Your program must manually clear bits in DMAIFR by writing a 1 to them.

3.2.14.2 Interrupt Multiplexing

As described in the previous section, on the DSP, the interrupts from each of the four DMA controllers are combined into a single CPU interrupt. However, the resulting DMA interrupt is not multiplexed with any other interrupt source.

3.2.15 Power Management

Each DMA controller can be idled independently to conserve power if it is not being actively used. This is achieved by turning off the peripheral clock of each DMA controller in the peripheral clock gating configuration register (PCGCR). For more information on the PCGCR register, see Chapter 1, *System Control*.

3.2.16 Emulation Considerations

The DMA controller is not interrupted by emulation events such as an emulation breakpoint. However, an emulation suspend may halt activity in a peripheral being serviced by the DMA controller. In this case, the DMA controller activity will be indirectly suspended.

3.3 DMA Transfer Examples

The DMA controller can be used to perform two basic types of transfers: block transfers and peripheral servicing transfers. The following sections provide examples for these two typical use case scenarios.

3.3.1 Block Move Example

The most basic transfer performed by the DMA is a block move. During device operation it is often necessary to transfer a block of data from one location to another, usually between on-chip and off-chip memory.

In this example, data is copied from one section of the internal single-access memory (SARAM) to another section. A data block of 256 double words (1024 bytes) residing at internal CPU word address 02 6000h (SARAM, block 11) needs to be transferred to internal CPU word address 02 7000h (SARAM, block 31), as shown in Figure 3-6.

The source address for the transfer is set to the equivalent DMA byte address of the data block in external memory, and the destination address is set to the equivalent DMA byte address of the data block in SARAM. More specifically the equivalent DMA byte addresses for source and destination buffers described in this example are 000A 6000h and 000C E000h, respectively. For more information on DMA byte addresses, see Section 3.2.2.

Figure 3-7 shows the DMA channel register contents during the transfer after the channel is enabled.

Figure 3-6. Block Move Example

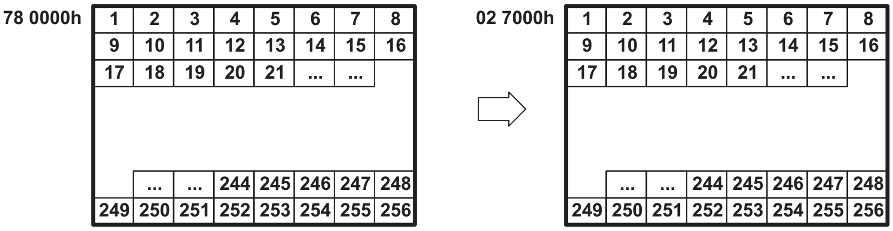


Figure 3-7. Block Move Example DMA Configuration

(a) DMA Register Contents

Register Contents		→	Parameter	
000Ah	6000h		DMACH*m*SSAU	DMACH*m*SSAL
000Ch	E000h		DMACH*m*DSAU	DMACH*m*DSAL
E020h	0400h		DMACH*m*TCR2	DMACH*m*TCR1

(b) Channel Transfer Control Options

15	14	13	12	11 10	9 8
1	1	1	1	00	00
EN	STATUS	INTEN	AUTORLD	RSV	DSTAMODE

7 6	5 3	2	1	0
10	000	1	0	0
SRCAMODE	BURSTMODE	SYNCMODE	LAST_XFER	PING_PONG_EN

3.3.2 Peripheral Servicing Example

The DMA controllers can service peripherals in the background of CPU operation, without requiring any CPU intervention. Through proper initialization of the DMA channels, they can be configured to continuously service on-chip and off-chip peripherals throughout device operation. Each DMA controller has a set of synchronization events which can trigger activity in DMA channels specified by the user. When programming a DMA channel to service a peripheral, it is necessary to know how data is to be presented to the DSP. Data is always provided with some kind of synchronization event as either one data sample per event (non-bursting) or multiple data samples per event (bursting).

3.3.2.1 Non-Bursting Peripherals

Non-bursting peripherals include the on-chip inter-integrated circuit (I2C) module and many external devices, such as codecs. Regardless of the peripheral, the DMA channel configuration is the same.

The I2C transmit and receive data streams are treated independently by the DMA. The transmit and receive data streams can have completely different counts, data sizes, and formats. Figure 3-8 shows DMA servicing incoming I2C data.

To transfer the incoming data stream to its proper location in internal memory, the DMA channel must be set up for a non-burst transfer with synchronization enabled. Since a receive event (ICREVT) is generated for every data sample as it arrives, it is necessary to have the DMA transfer each data sample individually. Figure 3-8 shows the DMA channel register contents for this transfer after the channel is enabled.

The source address of the DMA channel is set to the data receive register (ICDRR) address for the I2C, and the destination address is set to the start of the data block in internal memory. Since the address of ICDRR is fixed, the source address mode is set to 10b (constant address) and the destination address mode is set to 00b (automatic post-increment). Note that in this example the destination address is set to the DMA byte address 000C E000h, which corresponds to SARAM block 31. For more information on DMA byte addresses, see Section 3.2.2.

Note that the DMA will transfer a full double word from ICDRR to the destination address every time a receive synchronization event is generated by the I2C. When allocating memory for the receive buffer, two 16-bit words must be allocated for every I2C data sample.

Figure 3-8. Servicing Incoming I2C Data Example

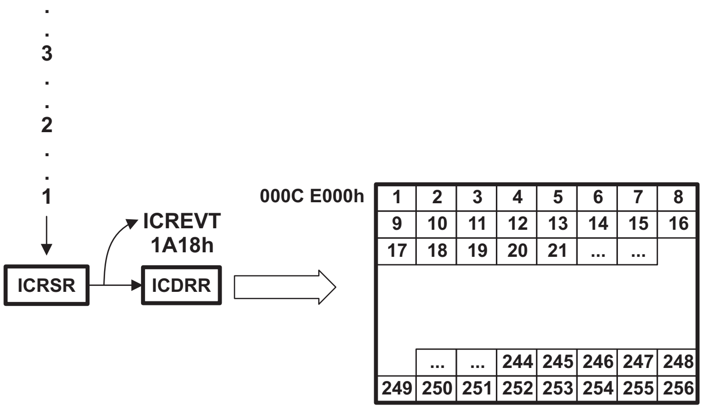


Figure 3-9. Servicing Incoming I2C Data Example DMA Configuration

(a) DMA Register Contents

Register Contents		➔	Parameter	
0000h	1A18h		DMACH*m*SSAU	DMACH*m*SSAL
000Ch	E000h		DMACH*m*DSAU	DMACH*m*DSAL
F084h	0400h		DMACH*m*TCR2	DMACH*m*TCR1

(b) Channel Transfer Control Options

15	14	13	12	11–10	9–8
1	1	1	1	00	00
EN	STATUS	INTEN	AUTORLD	RSV	DSTAMODE

7–6	5–3	2	1	0
10	000	1	0	0
SRCAMODE	BURSTMODE	SYNCMODE	LAST_XFER	PING_PONG_EN

3.3.2.2 Bursting Peripherals

Bursting peripherals include only the universal asynchronous receiver/transmitter (UART). For it, the DMA can be configured to transfer multiple data samples every time the channel is serviced.

The UART transmit and receive data streams are treated independently by the DMA. The transmit and receive data streams can have completely different counts, data sizes, and formats. Furthermore, the DMA burst size feature can be used to empty or fill the UART FIFO every time the UART generates a synchronization event. Figure 3-10 shows DMA servicing incoming UART data.

To transfer the incoming data in the UART FIFO to its proper location in internal memory, the DMA channel must be set up for a burst transfer with synchronization enabled. Since a receive event (URXEVT) is generated every time the FIFO trigger level is reached, it is necessary to have the DMA channel burst transfer size match the UART FIFO trigger level. For example, if the UART FIFO trigger level is set to 8 bytes, the DMA channel burst size must be set to 8 double words. Note that although the DMA always transfers double words, the UART treats each double word request as a single byte request. Also, when allocating memory for the receive buffer, four bytes must be allocated for every UART data sample.

Figure 3-11 shows the DMA channel register contents for this transfer after the channel is enabled. The source address of the DMA channel is set to the receive buffer register (RBR) address for the UART, and the destination address is set to the start of the data block in internal memory. Since the address of RBR is fixed, the source address mode is set to 10b (constant address) and the destination address mode is set to 00b (automatic post-increment).

Note that in this example the destination address is set to the DMA byte address 000C E000h, which corresponds to SARAM block 31 .

For more information on DMA byte addresses, see Section 3.2.2.

Figure 3-10. Servicing Incoming UART Data Example

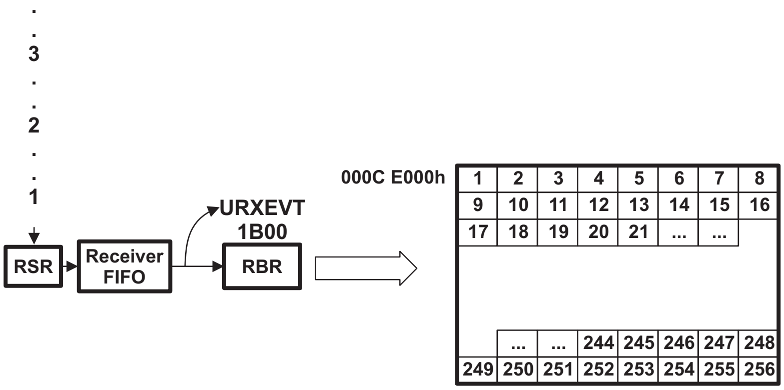


Figure 3-11. Servicing Incoming UART Data Example DMA Configuration

(a) DMA Register Contents

Register Contents			Parameter	
0000h	1B00h		DMACH*m*SSAU	DMACH*m*SSAL
000Ch	E000h	➔	DMACH*m*DSAU	DMACH*m*DSAL
F09Ch	0400h		DMACH*m*TCR2	DMACH*m*TCR1

(b) Channel Transfer Control Options

15	14	13	12	11 10	9 8
1	1	1	1	00	00
EN	STATUS	INTEN	AUTORLD	RSV	DSTAMODE

7 6	5 3	2	1	0
10	000	1	0	0
SRCAMODE	BURSTMODE	SYNCMODE	LAST_XFER	PING_PONG_EN

3.3.3 Ping-Pong DMA Example

The example here describes Ping-Pong transfer from an external codec (non-bursting peripheral) to the internal memory. Figure 3-12 shows DMA transfer of incoming data from the codec through the I2S0 in a Ping-Pong fashion. To transfer the incoming data stream to its destination in the internal memory, the DMA channel must be set up for a non-burst transfer with synchronization enabled. A receive synchronization event is generated for every data sample as it arrives; hence it is necessary to have the DMA transfer each data sample individually.

Figure 3-12 shows the DMA channel register contents for this transfer after the channel is enabled. The source address of the DMA channel is set to the left data0 receive register (I2S0RXLT0) address for I2S0, and the destination address is set to the start of the data block in internal memory. Since the address of I2S0RXLT0 is fixed, the source address mode is set to 10b (constant address) and the destination address mode is set to 00b (automatic post-increment). Note that in this example the destination address is set to the DMA byte address 000C E000h, which corresponds to SARAM block 31 at 0004 E000h and DMA transfer length is 1K bytes. For more information on DMA byte addresses, see Section 3.2.2. When allocating memory for the receive buffer, two 16-bit words must be allocated for every I2S data sample. It is also assumed that 1K bytes data buffer has been allocated at 000C E000h.

On every receive event generated by the I2S, the DMA will transfer a full double word from I2S0RXLT0 to the destination address. After the DMA has transferred the 128th sample, it sends an interrupt to the CPU, sets the LAST_XFER bit to 0 and continues the transfer. Once the 256th sample is transferred, the DMA controller again generates an interrupt to the CPU and sets the LAST_XFER bit to 1. Since the AUTORLD bit has been set to 1, the destination address is reloaded and the transfer resumes.

Figure 3-12. Servicing Incoming I2S Data Example in Ping-Pong DMA Mode

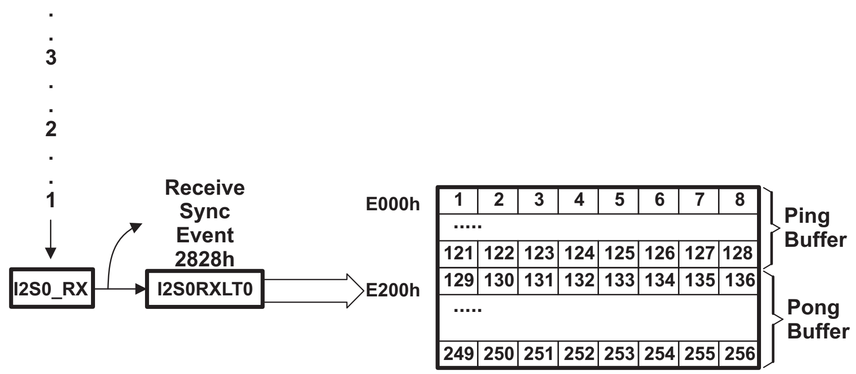


Figure 3-13. Servicing Incoming I2S Data Example DMA Configuration

(a) DMA Register Contents

Register Contents			Parameter	
0000h	2828h		DMACH*m*SSAU	DMACH*m*SSAL
000Ch	E000h	➔	DMACH*m*DSAU	DMACH*m*DSAL
F085h	0400h		DMACH*m*TCR2	DMACH*m*TCR1

(b) Channel Transfer Control Options

15	14	13	12	11 - 10	9 - 8
1	1	1	1	00	00
EN	STATUS	INTEN	AUTORLD	RSV	DSTAMODE

7 - 6	5 - 3	2	1	0
10	000	1	0	1
SRCAMODE	BURSTMODE	SYNCMODE	LAST_XFER	PING_PONG_EN

3.4 Registers

Table 3-4 through Table 3-8 list the memory-mapped registers associated with the four direct memory access (DMA) controllers. The DMA controller registers can be accessed by the CPU at the word addresses specified in each table. Note that the CPU accesses all peripheral registers through its I/O space. All other register addresses not listed in the tables below should be considered as reserved locations and the register contents should not be modified.

There are several other registers that affect the operation of the DMA controllers. The DMA interrupt flag and enable registers (DMAIFR and DMAIER) are used to control the interrupt generation of the four DMA controllers. In addition, there are two registers per DMA controller which control event synchronization in each channel—the DMA*n* channel event source registers (DMA*n*CESR1 and DMA*n*CESR2). These registers are not part of the DMA controllers; they are part of the DSP system. For more information on these registers, see Chapter 1

Table 3-4. System Registers Related to the DMA Controllers[1]

CPU Word Address	Acronym	Register Description
1C30h	DMAIFR	DMA Interrupt Flag Register
1C31h	DMAIER	DMA Interrupt Enable Register
1C1Ah	DMA0CESR1	DMA0 Channel Event Source Register 1
1C1Bh	DMA0CESR2	DMA0 Channel Event Source Register 2
1C1Ch	DMA1CESR1	DMA1 Channel Event Source Register 1
1C1Dh	DMA1CESR2	DMA1 Channel Event Source Register 2
1C36h	DMA2CESR1	DMA2 Channel Event Source Register 1
1C37h	DMA2CESR2	DMA2 Channel Event Source Register 2
1C38h	DMA3CESR1	DMA3 Channel Event Source Register 1
1C39h	DMA3CESR2	DMA3 Channel Event Source Register 2

[1] See Chapter 1 for more information on these registers.

Table 3-5. DMA Controller 0 (DMA0) Registers

CPU Word Address	Acronym	Register Description	Section
0C00h	DMACH0SSAL	Channel 0 Source Start Address (Lower Part) Register	Section 3.4.1
0C01h	DMACH0SSAU	Channel 0 Source Start Address (Upper Part) Register	Section 3.4.1
0C02h	DMACH0DSAL	Channel 0 Destination Start Address (Lower Part) Register	Section 3.4.2
0C03h	DMACH0DSAU	Channel 0 Destination Start Address (Upper Part) Register	Section 3.4.2
0C04h	DMACH0TCR1	Channel 0 Transfer Control Register 1	Section 3.4.3
0C05h	DMACH0TCR2	Channel 0 Transfer Control Register 2	Section 3.4.3
0C20h	DMACH1SSAL	Channel 1 Source Start Address (Lower Part) Register	Section 3.4.1
0C21h	DMACH1SSAU	Channel 1 Source Start Address (Upper Part) Register	Section 3.4.1
0C22h	DMACH1DSAL	Channel 1 Source Start Address (Lower Part) Register	Section 3.4.2
0C23h	DMACH1DSAU	Channel 1 Destination Start Address (Upper Part) Register	Section 3.4.2
0C24h	DMACH1TCR1	Channel 1 Transfer Control Register 1	Section 3.4.3
0C25h	DMACH1TCR2	Channel 1 Transfer Control Register 2	Section 3.4.3
0C40h	DMACH2SSAL	Channel 2 Source Start Address (Lower Part) Register	Section 3.4.1
0C41h	DMACH2SSAU	Channel 2 Source Start Address (Upper Part) Register	Section 3.4.1
0C42h	DMACH2DSAL	Channel 2 Destination Start Address (Lower Part) Register	Section 3.4.2
0C43h	DMACH2DSAU	Channel 2 Destination Start Address (Upper Part) Register	Section 3.4.2
0C44h	DMACH2TCR1	Channel 2 Transfer Control Register 1	Section 3.4.3
0C45h	DMACH2TCR2	Channel 2 Transfer Control Register 2	Section 3.4.3
0C60h	DMACH3SSAL	Channel 3 Source Start Address (Lower Part) Register	Section 3.4.1
0C61h	DMACH3SSAU	Channel 3 Source Start Address (Upper Part) Register	Section 3.4.1

Table 3-5. DMA Controller 0 (DMA0) Registers (continued)

CPU Word Address	Acronym	Register Description	Section
0C62h	DMACH3DSAL	Channel 3 Destination Start Address (Lower Part) Register	Section 3.4.2
0C63h	DMACH3DSAU	Channel 3 Destination Start Address (Upper Part) Register	Section 3.4.2
0C64h	DMACH3TCR1	Channel 3 Transfer Control Register 1	Section 3.4.3
0C65h	DMACH3TCR2	Channel 3 Transfer Control Register 2	Section 3.4.3

Table 3-6. DMA Controller 1 (DMA1) Registers

CPU Word Address	Acronym	Register Description	Section
0D00h	DMACH0SSAL	Channel 0 Source Start Address (Lower Part) Register	Section 3.4.1
0D01h	DMACH0SSAU	Channel 0 Source Start Address (Upper Part) Register	Section 3.4.1
0D02h	DMACH0DSAL	Channel 0 Destination Start Address (Lower Part) Register	Section 3.4.2
0D03h	DMACH0DSAU	Channel 0 Destination Start Address (Upper Part) Register	Section 3.4.2
0D04h	DMACH0TCR1	Channel 0 Transfer Control Register 1	Section 3.4.3
0D05h	DMACH0TCR2	Channel 0 Transfer Control Register 2	Section 3.4.3
0D20h	DMACH1SSAL	Channel 1 Source Start Address (Lower Part) Register	Section 3.4.1
0D21h	DMACH1SSAU	Channel 1 Source Start Address (Upper Part) Register	Section 3.4.1
0D22h	DMACH1DSAL	Channel 1 Destination Start Address (Lower Part) Register	Section 3.4.2
0D23h	DMACH1DSAU	Channel 1 Destination Start Address (Upper Part) Register	Section 3.4.2
0D24h	DMACH1TCR1	Channel 1 Transfer Control Register 1	Section 3.4.3
0D25h	DMACH1TCR2	Channel 1 Transfer Control Register 2	Section 3.4.3
0D40h	DMACH2SSAL	Channel 2 Source Start Address (Lower Part) Register	Section 3.4.1
0D41h	DMACH2SSAU	Channel 2 Source Start Address (Upper Part) Register	Section 3.4.1
0D42h	DMACH2DSAL	Channel 2 Destination Start Address (Lower Part) Register	Section 3.4.2
0D43h	DMACH2DSAU	Channel 2 Destination Start Address (Upper Part) Register	Section 3.4.2
0D44h	DMACH2TCR1	Channel 2 Transfer Control Register 1	Section 3.4.3
0D45h	DMACH2TCR2	Channel 2 Transfer Control Register 2	Section 3.4.3
0D60h	DMACH3SSAL	Channel 3 Source Start Address (Lower Part) Register	Section 3.4.1
0D61h	DMACH3SSAU	Channel 3 Source Start Address (Upper Part) Register	Section 3.4.1
0D62h	DMACH3DSAL	Channel 3 Destination Start Address (Lower Part) Register	Section 3.4.2
0D63h	DMACH3DSAU	Channel 3 Destination Start Address (Upper Part) Register	Section 3.4.2
0D64h	DMACH3TCR1	Channel 3 Transfer Control Register 1	Section 3.4.3
0D65h	DMACH3TCR2	Channel 3 Transfer Control Register 2	Section 3.4.3

Table 3-7. DMA Controller 2 (DMA2) Registers

CPU Word Address	Acronym	Register Description	Section
0E00h	DMACH0SSAL	Channel 0 Source Start Address (Lower Part) Register	Section 3.4.1
0E01h	DMACH0SSAU	Channel 0 Source Start Address (Upper Part) Register	Section 3.4.1
0E02h	DMACH0DSAL	Channel 0 Destination Start Address (Lower Part) Register	Section 3.4.2
0E03h	DMACH0DSAU	Channel 0 Destination Start Address (Upper Part) Register	Section 3.4.2
0E04h	DMACH0TCR1	Channel 0 Transfer Control Register 1	Section 3.4.3
0E05h	DMACH0TCR2	Channel 0 Transfer Control Register 2	Section 3.4.3
0E20h	DMACH1SSAL	Channel 1 Source Start Address (Lower Part) Register	Section 3.4.1
0E21h	DMACH1SSAU	Channel 1 Source Start Address (Upper Part) Register	Section 3.4.1
0E22h	DMACH1DSAL	Channel 1 Destination Start Address (Lower Part) Register	Section 3.4.2
0E23h	DMACH1DSAU	Channel 1 Destination Start Address (Upper Part) Register	Section 3.4.2

Table 3-7. DMA Controller 2 (DMA2) Registers (continued)

CPU Word Address	Acronym	Register Description	Section
0E24h	DMACH1TCR1	Channel 1 Transfer Control Register 1	Section 3.4.3
0E25h	DMACH1TCR2	Channel 1 Transfer Control Register 2	Section 3.4.3
0E40h	DMACH2SSAL	Channel 2 Source Start Address (Lower Part) Register	Section 3.4.1
0E41h	DMACH2SSAU	Channel 2 Source Start Address (Upper Part) Register	Section 3.4.1
0E42h	DMACH2DSAL	Channel 2 Destination Start Address (Lower Part) Register	Section 3.4.2
0E43h	DMACH2DSAU	Channel 2 Destination Start Address (Upper Part) Register	Section 3.4.2
0E44h	DMACH2TCR1	Channel 2 Transfer Control Register 1	Section 3.4.3
0E45h	DMACH2TCR2	Channel 2 Transfer Control Register 2	Section 3.4.3
0E60h	DMACH3SSAL	Channel 3 Source Start Address (Lower Part) Register	Section 3.4.1
0E61h	DMACH3SSAU	Channel 3 Source Start Address (Upper Part) Register	Section 3.4.1
0E62h	DMACH3DSAL	Channel 3 Destination Start Address (Lower Part) Register	Section 3.4.2
0E63h	DMACH3DSAU	Channel 3 Destination Start Address (Upper Part) Register	Section 3.4.2
0E64h	DMACH3TCR1	Channel 3 Transfer Control Register 1	Section 3.4.3
0E65h	DMACH3TCR2	Channel 3 Transfer Control Register 2	Section 3.4.3

Table 3-8. DMA Controller 3 (DMA3) Registers

CPU Word Address	Acronym	Register Description	Section
0F00h	DMACH0SSAL	Channel 0 Source Start Address (Lower Part) Register	Section 3.4.1
0F01h	DMACH0SSAU	Channel 0 Source Start Address (Upper Part) Register	Section 3.4.1
0F02h	DMACH0DSAL	Channel 0 Destination Start Address (Lower Part) Register	Section 3.4.2
0F03h	DMACH0DSAU	Channel 0 Destination Start Address (Upper Part) Register	Section 3.4.1
0F04h	DMACH0TCR1	Channel 0 Transfer Control Register 1	Section 3.4.3
0F05h	DMACH0TCR2	Channel 0 Transfer Control Register 2	Section 3.4.3
0F20h	DMACH1SSAL	Channel 1 Source Start Address (Lower Part) Register	Section 3.4.1
0F21h	DMACH1SSAU	Channel 1 Source Start Address (Upper Part) Register	Section 3.4.1
0F22h	DMACH1DSAL	Channel 1 Destination Start Address (Lower Part) Register	Section 3.4.2
0F23h	DMACH1DSAU	Channel 1 Destination Start Address (Upper Part) Register	Section 3.4.2
0F24h	DMACH1TCR1	Channel 1 Transfer Control Register 1	Section 3.4.3
0F25h	DMACH1TCR2	Channel 1 Transfer Control Register 2	Section 3.4.3
0F40h	DMACH2SSAL	Channel 2 Source Start Address (Lower Part) Register	Section 3.4.1
0F41h	DMACH2SSAU	Channel 2 Source Start Address (Upper Part) Register	Section 3.4.1
0F42h	DMACH2DSAL	Channel 2 Destination Start Address (Lower Part) Register	Section 3.4.2
0F43h	DMACH2DSAU	Channel 2 Destination Start Address (Upper Part) Register	Section 3.4.2
0F44h	DMACH2TCR1	Channel 2 Transfer Control Register 1	Section 3.4.3
0F45h	DMACH2TCR2	Channel 2 Transfer Control Register 2	Section 3.4.3
0F60h	DMACH3SSAL	Channel 3 Source Start Address (Lower Part) Register	Section 3.4.1
0F61h	DMACH3SSAU	Channel 3 Source Start Address (Upper Part) Register	Section 3.4.1
0F62h	DMACH3DSAL	Channel 3 Destination Start Address (Lower Part) Register	Section 3.4.2
0F63h	DMACH3DSAU	Channel 3 Destination Start Address (Upper Part) Register	Section 3.4.2
0F64h	DMACH3TCR1	Channel 3 Transfer Control Register 1	Section 3.4.3
0F65h	DMACH3TCR2	Channel 3 Transfer Control Register 2	Section 3.4.3

3.4.1 Source Start Address Registers (DMACHmSSAL and DMACHmSSAU)

Each channel has two source start address registers, which are shown in Figure 3-14 and Figure 3-15 and described in Table 3-9 and Table 3-10. For the first access to the source port of the channel, the DMA controller generates a byte address by concatenating the contents of the two I/O-mapped registers. DMACH*m*SSAU supplies the upper bits, and DMACH*m*SSAL supplies the lower bits:

Source start address = DMACH*m*SSAU:DMACH*m*SSAL

The channel updates the source start address registers every time it is serviced the DMA controller. The amount of data transferred each time the channel is serviced is specified by the BURSTMODE bits of DMACH*m*TCR2.

The destination start address is supplied by DMACH*m*DSAL and DMACH*m*DSAU, which are described in Section 3.4.2.

NOTE:

1. You must load the source start address registers with a byte address. For more details, see Section 3.2.4.
2. There are four DMA controllers in the DSP, although all DMA controllers can access DARAM and SARAM, each DMA controller can only access a subset of on-chip peripherals. Also, only DMA controller 3 has access to external memory. For more details, see Section 3.2.2.
3. All data buffers in on-chip or off-chip memory should be aligned on an even boundary. For more information on managing memory, see the *TMS320C55x Assembly Language Tools User Guide* (SPRU280).

Figure 3-14. Source Start Address Register - Lower Part (DMACH*m*SSAL)

<table>
<tr><td>15</td><td align="right">0</td></tr>
<tr><td colspan="2" align="center">SSAL</td></tr>
<tr><td colspan="2" align="center">RW-0</td></tr>
</table>

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 3-15. Source Start Address Register - Upper Part (DMACH*m*SSAU)

<table>
<tr><td>15</td><td align="right">0</td></tr>
<tr><td colspan="2" align="center">SSAU</td></tr>
<tr><td colspan="2" align="center">RW-0</td></tr>
</table>

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 3-9. Source Start Address Register - Lower Part (DMACH*m*SSAL) Field Description

Bit	Field	Value	Description
15-0	SSAL	0-FFFFh	Lower part of source start address (byte address).

Table 3-10. Source Start Address Register - Upper Part (DMACH*m*SSAU) Field Description

Bit	Field	Value	Description
15-0	SSAU	0-FFFFh	Upper part of source start address (byte address).

3.4.2 Destination Start Address Registers (DMACHmDSAL and DMACHmDSAU)

Each channel has two destination start address registers, which are shown in Figure 3-16 as well as Figure 3-17 and described in Table 3-11 and Table 3-12. For the first access to the destination port of the channel, the DMA controller generates a byte address by concatenating the contents of the two I/O-mapped registers. DMACH*m*DSAU supplies the upper bits, and DMACH*m*DSAL supplies the lower bits:

Destination start address = DMACH*m*DSAU:DMACH*m*DSAL

The channel updates the source start address registers every time it is serviced the DMA controller. The amount of data transferred each time the channel is serviced is specified by the BURSTMODE bits of DMACH*m*TCR2.

The source start address is supplied by DMACH*m*SSAL and DMACH*m*SSAU, which are described in Section 3.4.1.

NOTE:

1. You must load the source start address registers with a byte address. For more details, see Section 3.2.4.
2. There are four DMA controllers in the DSP, although all DMA controllers can access DARAM and SRAM, each DMA controller can only access a subset of on-chip peripherals. Also, only DMA controller 3 has access to external memory. For more details, see Section 3.2.2.
3. All data buffers in on-chip or off-chip memory should be aligned on an even boundary. For more information on managing memory, see the *TMS320C55x Assembly Language Tools User Guide* (SPRU280).

Figure 3-16. Destination Start Address Register - Lower Part (DMACH*m*DSAL)

15	0
DSAL	
RW-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 3-17. Destination Start Address Register - Upper Part (DMACH*m*DSAU)

15	0
DSAU	
RW-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 3-11. DMA Destination Start Address Register - Lower Part (DMACH*m*DSAL) Field Description

Bit	Field	Value	Description
15-0	DSAL	0-FFFFh	Lower part of destination start address (byte address).

Table 3-12. DMA Destination Start Address Register - Upper Part (DMACH*m*DSAU) Field Description

Bit	Field	Value	Description
15-0	DSAU	0-FFFFh	Upper part of destination start address (byte address).

3.4.3 Transfer Control Registers (DMACHmTCR1 and DMACHmTCR2)

Each channel has two transfer control registers shown in Figure 3-18 and Figure 3-19. These I/O-mapped register enables you to specify the size of the transfer, enable event synchronization and auto-initialization, select the source and destination addressing mode, and specify the burst mode of the channel. You can also monitor the status of the DMA channel through these registers. Table 3-13 and Table 3-14 describes the fields of these registers.

> **CAUTION**
>
> When using synchronization events, you must set EN = 1 and SYNCMODE = 1 during the same write cycle to DMACH*m*TCR2. The DMA channel will transfer the first data value when it receives the synchronization event specified by CH*n*EVT in DMACESR1 and DMACESR2. Also, when disabling the channel, you must set EN = 0 and SYNCMODE = 0 during the same write cycle to DMACH*m*TCR2.
>
> The amount of data (in bytes) to be transferred as programmed in the LENGTH field in DMACH*m*TCR1 should be a multiple of 4 bytes x $2^{BURSTMODE}$ field in DMACH*m*TCR2, i.e., LENGTH = (4 x $2^{BURSTMODE}$) bytes.

Figure 3-18. Transfer Control Register 1 (DMACH*m*TCR1)

15 – 0
LENGTH
RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 3-19. Transfer Control Register 2 (DMACH*m*TCR2)

15	14	13	12	11 – 10	9 – 8
EN	STATUS	INTEN	AUTORLD	RSV	DSTAMODE
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7 – 6	5 – 3	2	1	0
SRCAMODE	BURSTMODE	SYNCMODE	LAST_XFER	PING_PONG_EN
R/W-0	R/W-0	R/W-0	R/W-0	R-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 3-13. Transfer Control Register 1 (DMACH*m*TCR1) Field Description

Bit	Field	Value	Description
15-0	LENGTH		Size of transfer. When Ping-Pong mode is enabled, this is the size of the **Ping** and the **Pong** transfer combined.
		0000h - 0003h	Reserved, do not use.
		0004h - FFFCh	These bits specify the number of double words (specified in bytes) to be transferred in multiples of (4 x $2^{BURSTMODE}$) bytes. **Note:** These bits are not updated by the channel each time it is serviced by the DMA controller.

Table 3-14. Transfer Control Register 2 (DMACH*m*TCR2) Field Descriptions

Bit	Field	Value	Description
15	EN		Channel enable bit. Use EN to enable or disable transfers in the channel. The DMA controller clears EN once a block transfer in the channel is complete.
		0	The channel is disabled. the channel cannot be serviced by the DMA controller. IF DMA burst transfer is ongoing in the channel, the DMA controller allows the burst transfer to complete.
			Note: When you write a 0 to this bit, you must also write a 0 to the SYNCMODE bit.
		1	The channel is enabled. the channel can be serviced by the DMA in the next available time slot.
14	STATUS		Channel status bit. This bit indicates the status o the DMA channel transfer.
			The DMA controller sets the channel STATUS bit to 1 if:
			- A nonzero value is written on LENGTH in DMACH*m*TCR1
			- A write access is performed to DMACH*m*TCR2 and LENGTH has a nonzero value.
			The DMA controller clears the STATUS bit to 0 if:
			- All the bytes specified by LENGTH in DMACH*m*TCR1 have been transferred.
			- A value of 0 is written to LENGTH in DMACH*m*TCR1
		0	Corresponding DMA channel has transferred all the bytes specified by LENGTH in DMACH*m*TCR1.
		1	Corresponding DMA channel has not finished transferring all the bytes specified by LENGTH in DMACH*m*TCR1.
13	INTEN		CPU interrupt enable bit. The DMA channel is capable of generating a CPU interrupt when a block transfer is finished. In order for the CPU to receive the interrupt, the corresponding channel interrupt mask bit in the interrupt mask register (DMAIMR) must be set to 1.
		0	Disable channel interrupt.
		1	Enabled channel interrupt.
12	AUTORLD		Automatic reload bit. Once a transfer is finished, the DMA automatically reloads the transfer control register and the source and destination start address registers and restarts the transfer.
			Note: Automatic reload can only be used when SYNCMODE = 1.
		0	DMA transfer does not automatically reload.
		1	Upon completion of a full transfer, the registers are reloaded and the transfer is restarted.
11-10	Reserved	0	Reserved, always write zeroes to these bits.
9-8	DSTAMODE		Destination addressing mode bits. DSTAMODE determines the addressing mode used by the DMA controller when it writes to the destination address.
		0	Automatic post increment. The destination byte address is incremented by four each transfer.
		1h	Reserved, do not use.
		2h	Constant address.
		3h	Reserved, do not use.
7-6	SRAMODE		Source addressing mode bits. SRCAMODE determines the addressing mode used by the DMA controller when it reads from the source address.
		0	Automatic post increment. The source byte address is incremented by four after each transfer.
		1h	Reserved, do not use.
		2h	Constant address.
		3h	Reserved, do not use.
5-3	BURSTMODE		Burst mode bits. These bits specify the number of double word transfers that each channel performs at once before the DMA controller moves on to the active channel.
			Note: The burst mode selected must always be less than or equal to the number of bytes specified in DMACH*m*TCR1.
		0	1 double word (4 bytes).
		1h	2 double words (8 bytes).
		2h	4 double words (16 bytes).
		3h	8 double words (32 bytes).
		4h	16 double words (64 bytes).
		5h-7h	Reserved, do not use.

Table 3-14. Transfer Control Register 2 (DMACH*m*TCR2) Field Descriptions (continued)

Bit	Field	Value	Description
2	SYNCMODE		Synchronization mode bit. CH*n*EVT bits in DMACESR*n* determine which event in the DSP (for example, a timer countdown) initiates a DMA transfer in the channel. Multiple channels can have the same SYNCEVT value; in other words, one synchronization event can initiate activity in multiple channels. On each sync event, the DMA transfers the number of double words specified by the BURSTMODE bits. For example, BURSTMODE = 010b, the DMA will transfer a total of 4 double words per sync event. A DSP reset selects SYNCMODE = 0 (no synchronization). When SYNCMODE = 0, the DMA controller does not wait for a synchronization event before beginning a DMA transfer in the channel; channel activity begins as soon as the channel is enabled (EN = 1).
		0	When synchronization is disabled, the DMA controller does not wait for a synchronization event before beginning a DMA transfer. **Note:** When you write a 0 to the EN bit, you must also write a 0 to this bit.
		1	Activity in the DMA controller is synchronized to the event specified in the CH*n*EVT bits of DMACESR*n*. **Note:** When you set this bit to 1, you must also write a 1 to the EN bit.
1	LAST_XFER		Indicates whether the most recent completed transfer was the **Ping** buffer or the **Pong** buffer. This status bit is only valid when the PING_PONG_EN bit is set to 1.
		0	The last completed transfer was the **Ping** buffer
		1	The last completed transfer was the **Pong** buffer
0	PING_PONG_EN		Enable Ping-Pong DMA transfer mode.
		0	Ping-Pong mode is disabled
		1	Ping-Pong mode is enabled

Real-Time Clock (RTC)

This chapter describes the features and operations of the real-time clock (RTC).

4.1 Introduction

The following sections describe the features and operation of the real-time clock (RTC) on the digital signal processor (DSP).

4.1.1 Purpose of the Peripheral

The device includes a real time clock (RTC) that provides a time reference to an application executing on the DSP.

The RTC has its own crystal input, clock domain, and core and I/O power supplies. The separate clock domain allows the RTC to run while the rest of the device is clock gated. All RTC registers are preserved and the counter continues to operate when the host CPU is clock gated. The RTC has the capability to wake-up the rest of the device through an alarm interrupt, periodic interrupt, or external WAKEUP signal.

4.1.2 Features

The real-time clock (RTC) provides the following features:

- RTC-only mode
- 100-year calendar up to year 2099
- Counts milliseconds, seconds, minutes, hours, and date (including day, month, and year with leap year compensation)
- Millisecond time correction
- Binary-coded-decimal (BCD) representation of time, calendar, and alarm
- 24-hour clock mode
- Alarm interrupt for specific millisecond, second, minute, hour, day, month, and year
- Periodic interrupt: every millisecond, second, minute, hour, or day
- Single interrupt to the DSP CPU
- 32.768kHz oscillator with frequency calibration

The current date and time is tracked in a set of counter registers that update once per millisecond. The time is represented in 24-hour mode. For information on how to set the time and date, see Section 4.2.4.

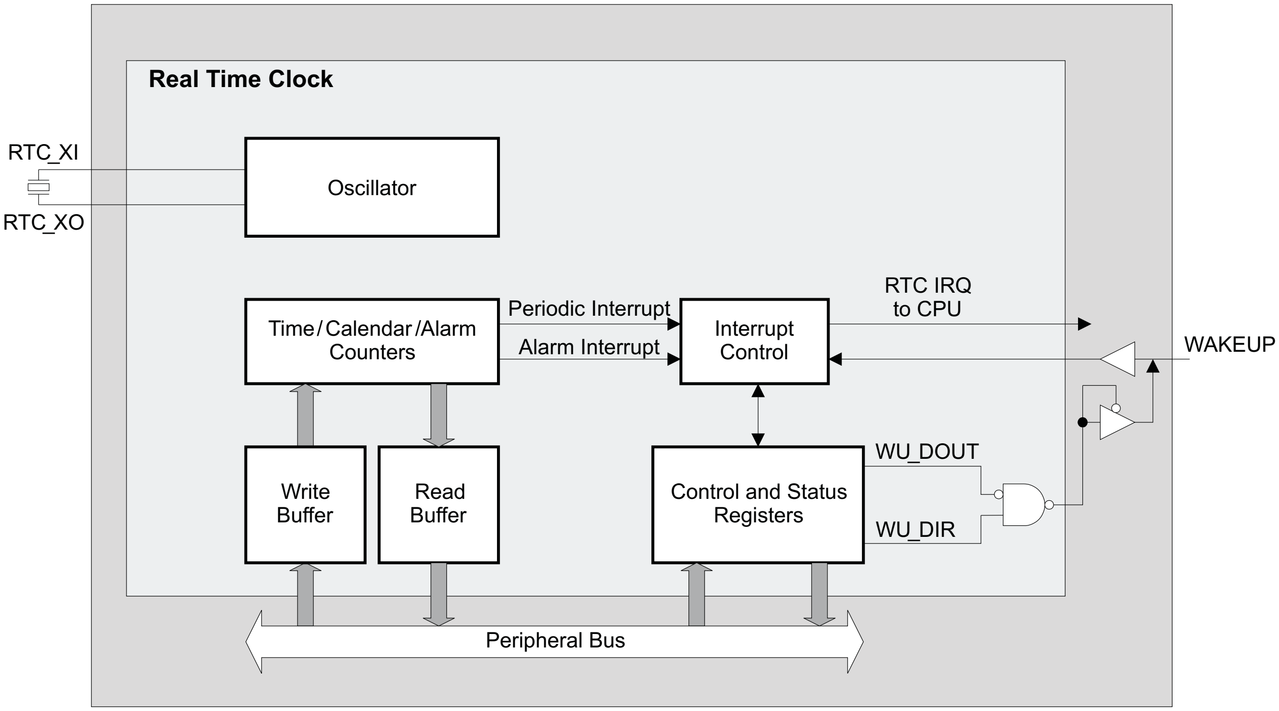
Alarms can be set to interrupt the DSP CPU at a particular time, or at periodic time intervals, such as once per minute or once per day. For information on how to set and use alarms, see Section 4.2.5.

The clock reference for the RTC is an external 32.768kHz crystal (connected between signals RTC_XI and RTC_XO). The RTC also has separate core and I/O power supplies that are isolated from the rest of the DSP.

4.1.3 Functional Block Diagram

Figure 4-1 shows a block diagram of the RTC.

Figure 4-1. Block Diagram



4.2 Peripheral Architecture

This section describes the Real-Time Clock (RTC) Peripheral.

NOTE: If the RTC is not used, CV_{DDRTC} and DV_{DDRTC} must still be supplied because internal signals from the RTC are required to drive the DSP's Power On Reset circuitry. In addition, the RTC_XI pin must be tied to CV_{DDRTC} and the RTC_XO pin must be tied to VSS. If the RTC_XI and RTC_XO pins are tied off, then the RTC registers are inaccessible.

4.2.1 Clock Control

The RTC oscillator is driven by an external 32.768 KHz crystal connected between RTC_XI and RTC_XO.

4.2.2 Signal Descriptions

As shown in Figure 4-1, the WAKEUP pin is a bidirectional pin that can be used as an input to wake-up the DSP clock domains or it can be used as an open-drain output to wake-up an external device. At power up the WAKEUP pin is configured as an input. This signal can be used to trigger the RTC interrupt to the CPU and to wake-up gated clocks regardless of whether the clocks were gated by the master clock gate or whether they were gated by the DSP's idle instruction.

When the WAKEUP pin is configured as an input (RTCPMGT:WU_DIR = 0), the RTC interrupt is enabled (RTCINTEN: 0 = 1) and the External Event Interrupt in RTCINTREG is enabled (RTCINTREG: EXTINTEN = 1), a 0 to 1 transition on the WAKEUP pin will trigger the RTC interrupt. Additionally, when WAKEUP is high, then the Master Clock Gater for the whole digital core is forced into the un-gated state (clocks on). Note that the interrupt generation is edge sensitive while the clocks on condition is level sensitive. Please see Section 4.2.6 for Interrupt support.

4.2.3 RTC-Only Mode

RTC-only mode allows all supplies except DV_{DDRTC} and CV_{DDRTC} to be powered-down. In this mode, the RTC counter continues to operate. The RTC has the capability to wakeup the device from idle states via alarms, periodic interrupts, or an external WAKEUP input. In addition, the RTC is able to output an alarm or periodic interrupt on the WAKEUP pin to cause external power management to re-enable power to the DSP core and I/O.

4.2.4 Using the Real-Time Clock Time and Calendar Registers

The current time and date are maintained in the RTC time and calendar registers. Information about how to use these registers is in the sections that follow.

4.2.4.1 Time/Calendar Data Format

The time and calendar data in the RTC is stored as binary-coded decimal (BCD) format. In BCD format, the decimal numbers 0 through 9 are encoded with their binary equivalent. Although most of the time/calendar registers have 4 bits assigned to each BCD digit, some of the register fields are shorter since the range of valid numbers may be limited. For example, only 3 bits are required to represent the first digit (most significant digit) of the “seconds” because only 0 through 5 are required.

The summary of the time/calendar registers is shown in Table 4-1. The alarm registers are interleaved with the time/calendar registers and are not shown in this table. The alarm registers are shown in Table 4-2. A complete description all RTC registers is available in Section 4.3.

Table 4-1. Time/Calendar Registers

Address (Hex)	Name	Function	Decimal Range	BCD Format
1904h	RTCMIL	Milliseconds	0-1023	0000-1023
1908h	RTCSEC	Seconds	0-59	00-59
190Ch	RTCMIN	Minutes	0-59	00-59
1910h	RTCHOUR	Hours (24)	0-23	00-23
1914h	RTCDAY	Days	1-31	01-31
1918h	RTCMONTH	Months (January = 01)	1-12	01-12
191Ch	RTCYEAR	Years	0-99	00-99

- The RTC Milliseconds Register (RTCMIL) stores the milliseconds value of the current time. After the milliseconds count reaches 1023 then the seconds register is updated by one. The reason for the rollover occurring at 1024, rather than 1000, is due to the crystal's oscillation frequency being a power of two, 32.768kHz. 32768 / 1024 = 32 clocks per 'millisecond'. Thus, calling this register a 'milliseconds' register is a bit of a misnomer. The milliseconds digit 3 is 1 bit and the milliseconds digits 2:0 are 4 bits; digits 3:0 are encoded BCD values 0000 (0b 0000b 0000b 0000b) through 1023 (1b 0000b 0010b 0011b).
- The RTC Seconds Register (RTCSEC) stores the seconds value of the current time. The seconds digit 1 is 3 bits and the seconds digit 0 is 4 bits; digits 1 and 0 are encoded BCD values 00 (000b 0000b) through 59 (101b 1001b).
- The RTC Minutes Register (RTCMIN) stores the minutes value of the current time. The minutes digit 1 is 3 bits and the minutes digit 0 is 4 bits; digits 1 and 0 are encoded BCD values 00 (000b 0000b) through 59 (101b 1001b).
- The RTC Hours Register (RTCHOUR) stores the hours value of the current time. The hours digit 1 is 2 bits and the hours digit 0 is 4 bits; digits 1 and 0 are encoded BCD values 01 (00b 0001b) through 23 (10b 0011b).
- The RTC Days Register (RTCDAY) stores the day of the month for the current date. The days digit 1 is 2 bits and the days digit 0 is 4 bits; digits 1 and 0 are encoded BCD 01 (00b 0001b) through 31 (11b 0001b).
- The RTC Months Register (RTCMONTH) stores the month for the current date. The months digit 1 is 1 bit and the months digit 0 is 4 bits; digits 1 and 0 are encoded BCD values 01 (0b 0000b) through 12 (1b 0010b).
- The RTC Years Register (RTCYEAR) stores the year for the current date. The years digit 1 is 4 bits and the years digit 0 is 4 bits; digits 1 and 0 are encoded BCD values 00 (0000b 0000b) through 99 (1001b 1001b).

4.2.4.2 Setting the Time/Calendar Register

The time/calendar registers are set or initialized by writing to the appropriate register bytes. To set date and time, unlock the RTC registers and write all the time and date registers. When written to, the data is stored to a buffer. Next, set the TIMEUPDT bit in the RTC Update Register (RTCUPDATE). Setting this bit causes the time/calendar values in the buffer to be loaded into the RTC simultaneously. All values should be encoded as BCD values.

4.2.4.3 Reading the Time/Calendar Registers

The time/calendar registers are updated every millisecond as the time changes. To get the most accurate time reading you should start with reading the Millisecond register (RTCMIL) and then the Second register (RTCSEC) followed by the remaining time/calendar register values (RTCMIN, RTCHOUR, RTCDAY, RTCMONTH, and RTCYEAR). Read the RTCMIL again and compare to the previous value. If both values are the same, an RTC update did not occur while the other registers were being read and all the values read represent the current time. If the Milliseconds have changed, this indicates that and RTC update occurred while the registers were being read and the process should be repeated. Results are unpredictable if values are written out of the register's normal range.

4.2.5 Using the Real-Time Clock Time and Calendar Alarms

Alarms can be configured to interrupt the CPU at a specific time, i.e., at specific values for the following:

- Milliseconds
- Seconds
- Minutes
- Hours
- Days of the month
- Months
- On specific Years

The time/calendar alarm registers control the setting of alarms. Information about how to use these registers can be found in the following sections. The alarms can be configured to generate an interrupt to the CPU or to wake-up the clocks. The operation of the alarm interrupt is described in Section 4.2.6.4.

4.2.5.1 Time/Calendar Alarm Data Format

The time and calendar alarm data in the RTC is stored as binary-coded decimal (BCD) format. In BCD format, the decimal numbers 0 through 9 are encoded with their binary equivalent. Although most of the time/calendar alarm registers have 4 bits assigned to each BCD digit, some of the register field lengths may differ to accommodate the desired function.

The summary of the time/calendar alarm registers is shown in Table 4-2. The time/calendar registers are interleaved with the alarm registers and are not shown in this table. The time/calendar registers are shown in Table 4-1. A complete description of all RTC registers is available in Section 4.3.

Table 4-2. Time and Calendar Alarm Data

Address (Hex)	Name	Function	Decimal Range	BCD Format
1905h	RTCMILA	Milliseconds alarm	0-1023	0000-1023
1909h	RTCSECA	Seconds alarm	0-59	00-59
190Dh	RTCMINA	Minutes alarm	0-59	00-59
1911h	RTCHOURA	Hours (24) alarm	0-23	00-23
1915h	RTCDAYA	Days alarm	1-31	01-31
1919h	RTCMONTHA	Months (January = 01) alarm	1-12	01-12
191Dh	RTCYEARA	Years alarm	0-99	00-99

The RTC Milliseconds Alarm Register (RTCMILA) stores the milliseconds value of the desired alarm. The milliseconds alarm digit 3 is 1 bit and the milliseconds alarm digits 2:0 are 4; digits 3:0 are encoded BCD values 0000 (0b 0000b 0000b 0000b) through 1023 (1b 0000b 0010b 0011b). Values outside of the decimal range of 0 – 1023 will cause the alarm to never occur.

The RTC Seconds Alarm Register (RTCSECA) stores the seconds value of the desired alarm. The seconds alarm digit 1 is 3 bits and the seconds alarm digit 0 is 4 bits; digits 1 and 0 are encoded BCD values 00 (000b 0000b) through 59 (101b 1001b). Values outside of the decimal range of 0 - 59 will cause the alarm to never occur.

The RTC Minutes Alarm Register (RTCMINA) stores the minute value of the desired alarm. The minutes alarm digit 1 is 3 bits and the minutes alarm digit 0 is 4 bits; digits 1 and 0 are encoded BCD values 00 (000b 0000b) through 59 (101b 1001b). Values outside of the decimal range of 0 - 59 will cause the alarm to never occur.

The RTC Hours Alarm Register (RTCHOURA) stores the hour value of the desired alarm. The hours alarm digit 1 is 2 bits and the hours alarm digit 0 is 4 bits; digits 1 and 0 are encoded BCD values 01 (00b 0001b) through 23 (10b 0011b). Values outside of the decimal range of 1 - 23 will cause the alarm to never occur.

The RTC Days Alarm Register (RTCDAYA) stores the day of the month value of the desired alarm. The days alarm digit 1 is 2 bits and the days alarm digit 0 is 4 bits; digits 1 and 0 are encoded BCD 01 (00b 0001b) through 31 (11b 0001b). Values outside of the decimal range of 1 - 31 will cause the alarm to never occur.

The RTC Months Alarm Register (RTCMONTHA) stores the month of the year value of the desired alarm. The months alarm digit 1 is 1 bit and the months alarm digit 0 is 4 bits; digits 1 and 0 are encoded BCD values 01 (0b 0000b) through 12 (1b 0010b). Values outsides the range 1 - 12 will cause the alarm to never occur.

The RTC Years Alarm Register (RTCYEARA) stores the year value of the desired alarm. The years alarm digit 1 is 4 bits and the years alarm digit 0 is 4 bits; digits 1 and 0 are encoded BCD values 00 (0000b 0000b) through 99 (1001b 1001b). Values outsides the range 0 - 99 will cause the alarm to never occur.

4.2.5.2 Setting and Reading the Time/Calendar Alarm Registers

The time/calendar alarm registers are set or initialized by writing to the appropriate register bytes. To set date and time, write all the time and date registers. Then set the ALARMUPDT bit in the RTC Update Register (RTCUPDATE). This will simultaneously copy all the alarm register settings in one RTC cycle.

Time/calendar alarm registers can be read at any time and are not updated by the RTC.

4.2.5.3 Examples of Time/Calendar Alarm Settings

Some examples of various alarm settings are shown in Table 4-3. A complete description of the RTC registers and their functions is provided in Section 4.3.

Table 4-3. Time/Calendar Alarm Settings

Alarm Occurs...	RTCYEARA	RTCMONTHA	RTCDAYA	RTCHOURA	RTCMINA	RTCSECA	RTCMILA
May 7, 2010 @ 3:19:46 AM	10h	5h	7h	3h	19h	46h	0h
Dec 22, 2099 @ 5:50:15 and 300ms PM	99h	12h	22h	17h	50h	15h	300h

4.2.6 *Real-Time Clock Interrupt Requests*

The RTC provides the ability to interrupt the CPU based on three events: a periodic interrupt, an alarm interrupt, or an external "Wakeup" interrupt. Although three interrupt sources are available, the RTC makes a single interrupt request to the CPU. Specific information about using each of the interrupt types is in the sections that follow.

4.2.6.1 Interrupt Enable

The RTC has two registers for enabling interrupts. The RTC Interrupt Enable (RTCINTEN) enables the RTC interrupt to the CPU. This bit allows any interrupt that is triggered in the RTC to be sent to the CPU. The second register is the RTC Interrupt Register (RTSINTREG) is used to enable the different interrupt events that can be passed to the CPU. These include the following:

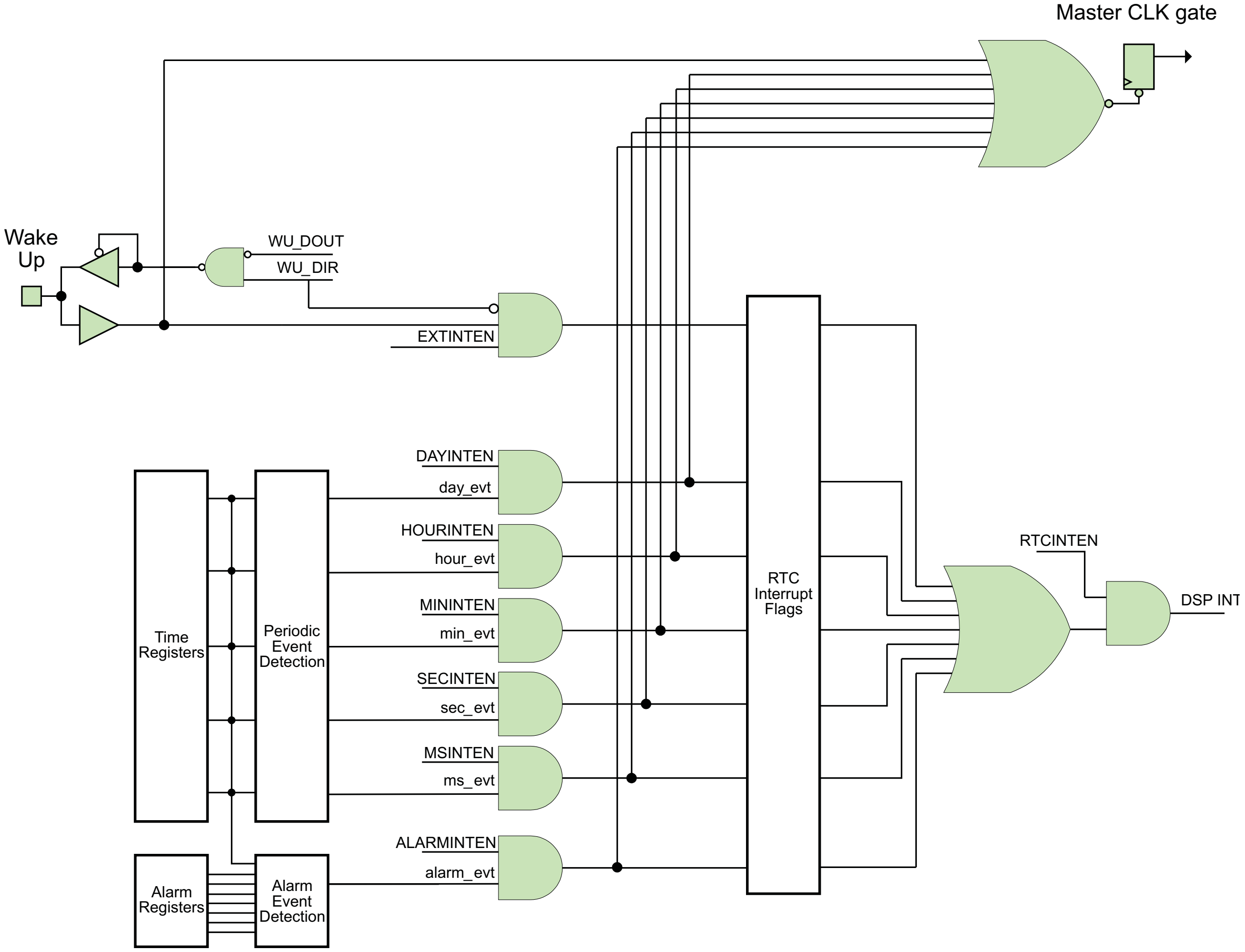
- Alarm Interrupt
- External “wakeup” Interrupt
- Periodic Day Interrupt
- Periodic Hour Interrupt
- Periodic Minute Interrupt
- Periodic Second Interrupt
- Periodic Millisecond Interrupt

NOTE: To use the external wakeup interrupt, you must set the WU_DIR bit in the RTCPMGT register to 0.

When a RTC interrupt is generated, the RTC's interrupt is directed to two places (see Figure 4-2).

1. System Clock Wakeup Logic – the interrupt will cause the Master Clock Gater to enable the Master Clock.
2. The RTC interrupt can be directed to the CPU if RTCINTEN = 1. The CPU's RTC interrupt must be unmasked for the CPU to respond to the interrupt.
   - If the CPU is idled, the interrupt will cause the CPU to exit idle. If interrupts are globally enabled, the CPU will execute the RTC ISR.
   - If the CPU is not idled and interrupts are globally enabled, the CPU will execute the RTC ISR.

Figure 4-2. RTC Interrupt and Wakeup Logic



4.2.6.2 Interrupt Flag Bits

When the interrupts are enabled in Section 4.2.6.1 and the event occurs, the equivalent flag is set in the RTC Interrupt Flag Register (RTCINTFL). See Section 4.3 for complete details of the RTC registers. The flagged event is cleared when the programmer writes a "1" to the flag bit.

There is also an RTC Lost Power Register (RTCNOPWR). If this flag is set the RTC has lost power and requires a software reset. NOTE: at least 3 RTC clock cycles must elapse after power-up in order to read valid data since the synchronization logic between the CPU and RTC consumes 3 RTC clock cycles.

If the RTC Interrupt enable bit is set and any of the active events occur then an RTC interrupt is sent to the CPU. The RTC interrupt is asserted as long as at least one of the interrupt flag bits are set. When an interrupt occurs from the RTC, the source of the interrupt can be determined by reading the flag bits in RTCINTFL.

4.2.6.3 Periodic Interrupt Request

Periodic Interrupts cause the RTC to make an interrupt request to the CPU periodically. The periodicity can be every millisecond, every second, every minute, hourly, or daily. The periodic interrupt rate is selected using the RTC Interrupts Register (RTCINTREG), see Table 4-4. Writing a 1 to these bits enables the periodic interrupt. Writing a 0 disables the interrupt. Once the interrupt occurs it will remain active until the corresponding flag bit in the RTC Status Register is cleared.

NOTE: The interrupt occurs whenever that particular time value is incremented.

Table 4-4. Periodic Interrupts

RTCINTREG bits	Periodic Interrupt Rate
Bit 0	Every Millisecond
Bit 1	Every Second
Bit 2	Every Minute
Bit 3	Every Hour
Bit 4	Every Day

To use the RTC Periodic interrupt:

- Select the desired interrupt period by enabling the proper interrupt in the RTCINTREG
- Enable the RTC interrupt to the CPU by setting bit 0 of RTCINTEN

When the periodic interrupt occurs, the corresponding interrupt flag will be set in the RTC Interrupt Flag (RTCINTFL) register and the interrupt is sent to the CPU.

4.2.6.4 Alarm Interrupt Request

The RTC alarm interrupt can be used to generate an interrupt to the CPU at a specific time. The alarm interrupt occurs when the alarm time programmed in the RTC alarm registers match the current time. For information about programming an alarm time, see Section 4.2.5.

To use the RTC alarm interrupt:

- Select the desired alarm time by configuring the RTC alarm registers.
- Enable the RTC alarm interrupt by setting bit 15 of the RTCINTREG.
- Enable the RTC interrupt to the CPU by setting bit 0 of RTCINTEN

When the alarm interrupt occurs, the Alarm Flag (bit 15) in the RTCINTFL register will be set and the RTC interrupt is sent to the CPU.

4.2.6.5 WAKEUP Interrupt Request

The external WAKEUP signal or RTC alarm trigger sends a WAKEUP event to the System Clock Wakeup Logic. This asynchronously clears the clock gate which gates the Master Clock and enables the Master Clock. When the DSP wakes up due to an RTC alarm, periodic interrupt, or by the external WAKEUP signal, the DSP latches the RTC interrupt. Because there is only one interrupt line for the RTC, the user must look at the RTC status register to determine which RTC event caused the wake-up.

4.2.7 Reset Considerations

The RTC can be reset by the RTCRESET bit located in the RTC oscillator register (RTCOSC). The RTC can also be reset by an internal POR circuit that monitors VDD_RTC. Neither the RESETN pin nor the DSP's POR can reset the RTC.

4.2.7.1 Software Reset Considerations

The DSP can cause a software reset of the RTC when the RTCRESET bit is set to 1. When this occurs, all RTC registers are reset to the default settings. The RTC will not be reset when the RESETN pin goes low. After a RTC software reset, do not access any RTC register for three 32.768kHz clock cycles after setting the software reset bit.

4.2.7.2 Hardware Reset Considerations

The RTC has a hardware reset that is tied to a POR circuit that monitors the VDD_RTC. The RTC is not reset with the RESETN pin or the DSP's POR.

4.3 Registers

4.3.1 Overview

This section describes the memory-mapped registers for the Real Time Clock (RTC).

Control of the RTC is maintained through a set of I/O memory mapped registers shown in Table 4-5. The first two registers in Table 5, RTCINTEN and RTCUPDATE, are located in the DSP core power domain, while the remaining registers in Table 5 are located in the RTC power domain.

Writes to registers in the RTC power domain are synchronized to the RTC 32.768 kHz clock and can therefore take many CPU clock cycles to complete. The CPU clock must run at least three-times faster than the RTC, and writes to registers in the RTC domain will not be evident for up to two 32.768kHz clock cycles. If the RTC oscillator is disabled (RTC_XI and RTC_XO pins tied off), no RTC register in the RTC power domain can be written.

4.3.2 RTC Registers

Table 4-5. Real Time Clock (RTC) Registers

CPU Word Address	Acronym	Register Description	Section
1900h	RTCINTEN	RTC Interrupt Enable Register	Section 4.3.2.1
1901h	RTCUPDATE	RTC Update Register	Section 4.3.2.2
1904h	RTCMIL	Milliseconds Register	Section 4.3.2.3
1905h	RTCMILA	Milliseconds Alarm Register	Section 4.3.2.4
1908h	RTCSEC	Seconds Register	Section 4.3.2.5
1909h	RTCSECA	Seconds Alarm Register	Section 4.3.2.6
190Ch	RTCMIN	Minutes Register	Section 4.3.2.7
190Dh	RTCMINA	Minutes Alarm Register	Section 4.3.2.8
1910h	RTCHOUR	Hours Register	Section 4.3.2.9
1911h	RTCHOURA	Hours Alarm Register	Section 4.3.2.10
1914h	RTCDAY	Days Register	Section 4.3.2.11
1915h	RTCDAYA	Days Alarm Register	Section 4.3.2.12
1918h	RTCMONTH	Months Register	Section 4.3.2.13
1919h	RTCMONTHA	Months Alarm Register	Section 4.3.2.14
191Ch	RTCYEAR	Years Register	Section 4.3.2.15
191Dh	RTCYEARA	Years Alarm Register	Section 4.3.2.16
1920h	RTCINTFL	RTC Interrupt Flag Register	Section 4.3.2.17
1921h	RTCNOPWR	RTC Lost Power Status Register	Section 4.3.2.18
1924h	RTCINTREG	RTC Interrupt Register	Section 4.3.2.19
1928h	RTCDRIFT	RTC Compensation Register	Section 4.3.2.20
192Ch	RTCOSC	RTC Oscillator Register	Section 4.3.2.21
1930h	RTCPMGT	RTC Power Management Register	Section 4.3.2.22
1960h	RTCSCR1	RTC LSW Scratch Register 1	Section 4.3.2.23
1961h	RTCSCR2	RTC MSW Scratch Register 2	Section 4.3.2.24
1964h	RTCSCR3	RTC LSW Scratch Register 3	Section 4.3.2.25
1965h	RTCSCR4	RTC MSW Scratch Register 4	Section 4.3.2.26

4.3.2.1 RTC Interrupt Enable Register (RTCINTEN)

The RTC interrupt enable register (RTCINTEN) is shown in Figure 4-3 and described in Table 4-6.

Figure 4-3. RTC Interrupt Enable Register (RTCINTEN)

15 1	0
Reserved	RTCINTEN
R-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-6. RTC Interrupt Enable Register (RTCINTEN) Field Descriptions

Bit	Field	Value	Description
15-1	Reserved	0	Reserved.
0	RTCINTEN		RTC interrupt enable.
		0	RTC interrupt is disabled.
		1	RTC interrupt is enabled.

4.3.2.2 RTC Update Register (RTCUPDATE)

The RTC update register (RTCUPDATE) is shown in Figure 4-4 and described in Table 4-7.

Figure 4-4. RTC Update Register (RTCUPDATE)

15	14	13 0
TIMEUPDT	ALARMUPDT	Reserved
RW-0	RW-0	R-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-7. RTC Update Register (RTCUPDATE) Field Descriptions

Bit	Field	Value	Description
15	TIMEUPDT		Initiates the Time updates.
		0	RTC time registers updated.
		1	Initiates the transfer of the time registers from the DSP to the RTC.
14	ALARMUPDT		Initiates the alarm updates.
		0	RTC alarm registers updated.
		1	Initiate update of the alarm registers.
13-0	Reserved	0	Reserved.

4.3.2.3 Milliseconds Register (RTCMIL)

The milliseconds register (RTCMIL) is shown in Figure 4-5 and described in Table 4-8.

Figure 4-5. Milliseconds Register (RTCMIL)

15 13	12	11 8	7 4	3 0
Reserved	MS3	MS2	MS1	MS0
R-0	RW-0	RW-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-8. Milliseconds Register (RTCMIL) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12	MS3		Digit 3 of Milliseconds in BCD format.
		0	Digit 3 of MS is 0.
		1	Digit 3 of MS is 1.
11-8	MS2	0-9	Digit 2 of Milliseconds in BCD format.
7-4	MS1	0-9	Digit 1 of Milliseconds in BCD format.
3-0	MS0	0-9	Digit 0 of Milliseconds in BCD format.

4.3.2.4 Milliseconds Alarm Register (RTCMILA)

The milliseconds alarm register (RTCMILA) is shown in Figure 4-6 and described in Table 4-9.

Figure 4-6. Milliseconds Alarm Register (RTCMILA)

15 13	12	11 8	7 4	3 0
Reserved	MSA3	MSA2	MSA1	MSA0
R-0	RW-0	RW-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-9. Milliseconds Alarm Register (RTCMILA) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12	MSA3		Digit 3 of Millisecond alarm in BCD format.
		0	Digit 3 of MS Alarm is 0.
		1	Digit 3 of MS Alarm is 1.
11-8	MSA2	0-9	Digit 2 of Millisecond alarm in BCD format.
7-4	MSA1	0-9	Digit 1 of Millisecond alarm in BCD format.
3-0	MSA0	0-9	Digit 0 of Millisecond alarm in BCD format.

4.3.2.5 Seconds Register (RTCSEC)

The seconds register (RTCSEC) is shown in Figure 4-7 and described in Table 4-10.

Figure 4-7. Seconds Register (RTCSEC)

15	7	6	4	3	0
Reserved		SEC1		SEC0	
R-0		RW-0		RW-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-10. Seconds Register (RTCSEC) Field Descriptions

Bit	Field	Value	Description
15-7	Reserved	0	Reserved.
6-4	SEC1	0-5	Digit 1 of Seconds in BCD format.
3-0	SEC0	0-9	Digit 0 of Seconds in BCD format.

4.3.2.6 Seconds Alarm Register (RTCSECA)

The seconds alarm register (RTCSECA) is shown in Figure 4-8 and described in Table 4-11.

Figure 4-8. Seconds Alarm Register (RTCSECA)

15	7	6	4	3	0
Reserved		SECA1		SECA0	
R-0		RW-0		RW-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-11. Seconds Alarm Register (RTCSECA) Field Descriptions

Bit	Field	Value	Description
15-7	Reserved	0	Reserved.
6-4	SECA1	0-5	Digit 1 of Seconds Alarm in BCD format.
3-0	SECA0	0-9	Digit 0 of Seconds Alarm in BCD format.

4.3.2.7 Minutes Register (RTCMIN)

The minutes register (RTCMIN) is shown in Figure 4-9 and described in Table 4-12.

Figure 4-9. Minutes Register (RTCMIN)

15 7	6 4	3 0
Reserved	MIN1	MIN0
R-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-12. Minutes Register (RTCMIN) Field Descriptions

Bit	Field	Value	Description
15-7	Reserved	0	Reserved.
6-4	MIN1	0-5	Digit 1 of Minutes in BCD format.
3-0	MIN0	0-9	Digit 0 of Minutes in BCD format.

4.3.2.8 Minutes Alarm Register (RTCMINA)

The minutes alarm register (RTCMINA) is shown in Figure 4-10 and described in Table 4-13.

Figure 4-10. Minutes Alarm Register (RTCMINA)

15 7	6 4	3 0
Reserved	MINA1	MINA0
R-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-13. Minutes Alarm Register (RTCMINA) Field Descriptions

Bit	Field	Value	Description
15-7	Reserved	0	Reserved.
6-4	MINA1	0-5	Digit 1 of Minutes Alarm in BCD format.
3-0	MINA0	0-9	Digit 0 of Minutes Alarm in BCD format.

4.3.2.9 Hours Register (RTCHOUR)

The hours register (RTCHOUR) is shown in Figure 4-11 and described in Table 4-14.

Figure 4-11. Hours Register (RTCHOUR)

15 6	5 4	3 0
Reserved	HOUR1	HOUR0
R-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-14. Hours Register (RTCHOUR) Field Descriptions

Bit	Field	Value	Description
15-6	Reserved	0	Reserved.
5-4	HOUR1	0-2	Digit 1 of Hours in BCD format.
3-0	HOUR0	0-9	Digit 0 of Hours in BCD format.

4.3.2.10 Hours Alarm Register (RTCHOURA)

The hours alarm register (RTCHOURA) is shown in Figure 4-12 and described in Table 4-15.

Figure 4-12. Hours Alarm Register (RTCHOURA)

15 6	5 4	3 0
Reserved	HOURA1	HOURA0
R-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-15. Hours Alarm Register (RTCHOURA) Field Descriptions

Bit	Field	Value	Description
15-6	Reserved	0	Reserved.
5-4	HOURA1	0-2	Digit 1 of Hours Alarm in BCD format.
3-0	HOURA0	0-9	Digit 0 of Hours Alarm in BCD format.

4.3.2.11 Days Register (RTCDAY)

The days register (RTCDAY) is shown in Figure 4-13 and described in Table 4-16.

Figure 4-13. Days Register (RTCDAY)

15 6	5 4	3 0
Reserved	DAY1	DAY0
R-0	RW-0	RW-1

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-16. Days Register (RTCDAY) Field Descriptions

Bit	Field	Value	Description
15-6	Reserved	0	Reserved.
5-4	DAY1	0-3	Digit 1 of Days in BCD format.
3-0	DAY0	0-9	Digit 0 of Days in BCD format.

4.3.2.12 Days Alarm Register (RTCDAYA)

The days alarm register (RTCDAYA) is shown in Figure 4-14 and described in Table 4-17.

Figure 4-14. Days Alarm Register (RTCDAYA)

15 6	5 4	3 0
Reserved	DAYA1	DAYA0
R-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-17. Days Alarm Register (RTCDAYA) Field Descriptions

Bit	Field	Value	Description
15-6	Reserved	0	Reserved.
5-4	DAYA1	0-3	Digit 1 of Days Alarm in BCD format.
3-0	DAYA0	0-9	Digit 0 of Days Alarm in BCD format.

4.3.2.13 Months Register (RTCMONTH)

The months register (RTCMONTH) is shown in Figure 4-15 and described in Table 4-18.

Figure 4-15. Months Register (RTCMONTH)

15 5	4	3 0
Reserved	MONTH1	MONTH0
R-0	RW-0	RW-1

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-18. Months Register (RTCMONTH) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved.
4	MONTH1	0-1	Digit 1 of Months in BCD format.
3-0	MONTH0	0-9	Digit 0 of Months in BCD format.

4.3.2.14 Months Alarm Register (RTCMONTHA)

The months alarm register (RTCMONTHA) is shown in Figure 4-16 and described in Table 4-19.

Figure 4-16. Months Alarm Register (RTCMONTHA)

15 5	4	3 0
Reserved	MONTHA1	MONTHA0
R-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-19. Months Alarm Register (RTCMONTHA) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved.
4	MONTHA1	0-1	Digit 1 of Months Alarm in BCD format.
3-0	MONTHA0	0-9	Digit 0 of Months Alarm in BCD format.

4.3.2.15 Years Register (RTCYEAR)

The years register (RTCYEAR) is shown in Figure 4-17 and described in Table 4-20.

Figure 4-17. Years Register (RTCYEAR)

15 8	7 4	3 0
Reserved	YEAR1	YEAR0
R-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-20. Years Register (RTCYEAR) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved.
7-4	YEAR1	0-9	Digit 1 of Year in BCD format (20XX).
3-0	YEAR0	0-9	Digit 0 of Year in BCD format (20XX).

4.3.2.16 Years Alarm Register (RTCYEARA)

The years alarm register (RTCYEARA) is shown in Figure 4-18 and described in Table 4-21.

Figure 4-18. Years Alarm Register (RTCYEARA)

15 8	7 4	3 0
Reserved	YEARA1	YEARA0
R-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-21. Years Alarm Register (RTCYEARA) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved.
7-4	YEARA1	0-9	Digit 1 of Year Alarm in BCD format (20XX).
3-0	YEARA0	0-9	Digit 0 of Year Alarm in BCD format (20XX).

4.3.2.17 RTC Interrupt Flag Register (RTCINTFL)

The RTC interrupt flag register (RTCINTFL) is shown in Figure 4-19 and described in Table 4-22.

Figure 4-19. RTC Interrupt Flag Register (RTCINTFL)

15	14	8
ALARMFL	Reserved	
R/W-0	R-0	

7	6	5	4	3	2	1	0
Reserved		EXTFL	DAYFL	HOURFL	MINFL	SECFL	MSFL
R-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-22. RTC Interrupt Flag Register (RTCINTFL) Field Descriptions

Bit	Field	Value	Description
15	ALARMFL		Flag indicating whether an Alarm interrupt has occurred.
		0	Alarm interrupt did not occur.
		1	Alarm interrupt occurred (write 1 to clear).
14-6	Reserved	0	Reserved.
5	EXTFL		Flag indicating whether an external event interrupt (WAKEUP pin) has occurred.
		0	External event interrupt did not occur.
		1	External event interrupt occurred (write 1 to clear).
4	DAYFL		Flag indicating whether a periodic Day interrupt has occurred.
		0	Periodic Day interrupt did not occur.
		1	Periodic Day interrupt occurred (write 1 to clear).
3	HOURFL		Flag indicating whether a periodic Hour event interrupt has occurred.
		0	Periodic Hour interrupt did not occur.
		1	Periodic Hour interrupt occurred (write 1 to clear).
2	MINFL		Flag indicating whether a periodic Minute event interrupt has occurred.
		0	Periodic Minute interrupt did not occur.
		1	Periodic Minute interrupt occurred (write 1 to clear).
1	SECFL		Flag indicating whether a periodic Second event interrupt has occurred.
		0	Periodic Second interrupt did not occur.
		1	Periodic Second interrupt occurred (write 1 to clear).
0	MSFL		Flag indicating whether a periodic Millisecond event interrupt has occurred.
		0	Periodic Milisecond interrupt did not occur.
		1	Periodic Milisecond interrupt occurred (write 1 to clear).

4.3.2.18 RTC Lost Power Status Register (RTCNOPWR)

The RTC lost power status register (RTCNOPWR) is shown in Figure 4-20 and described in Table 4-23.

Figure 4-20. RTC Lost Power Status Register (RTCNOPWR)

15 - 1	0
Reserved	PWRUP
R-0	R-1

LEGEND: R = Read only; -*n* = value after reset

Table 4-23. RTC Lost Power Status Register (RTCNOPWR) Field Descriptions

Bit	Field	Value	Description
15-1	Reserved	0	Reserved.
0	PWRUP		RTC has lost power Flag.
		0	RTC has not lost power since software reset.
		1	RTC has lost power and requires a software reset and initialization of the time registers to the current time and date.
			PWRUP is cleared by a read of RTCINTFL or RTCNOPWR. Therefore, read RTCNOPWR before reading RTCINTFL to obtain the correct PWRUP value.

4.3.2.19 RTC Interrupt Register (RTCINTREG)

The RTC interrupt register (RTCINTREG) is shown in Figure 4-21 and described in Table 4-24.

Figure 4-21. RTC Interrupt Register (RTCINTREG)

15	14	8
ALARMINTEN	Reserved	
R/W-0	R-0	

7	6	5	4	3	2	1	0
Reserved		EXTINTEN	DAYINTEN	HOURINTEN	MININTEN	SECINTEN	MSINTEN
R-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-24. RTC Interrupt Register (RTCINTREG) Field Descriptions

Bit	Field	Value	Description
15	ALARMINTEN		Alarm interrupt enable.
		0	Alarm interrupt not enabled.
		1	Alarm interrupt enabled.
14-6	Reserved	0	Reserved.
5	EXTINTEN		External event (WAKEUP pin) interrupt enable.
		0	External event interrupt not enabled.
		1	External event interrupt enabled.
4	DAYINTEN		Periodic Day interrupt enable.
		0	Periodic Day interrupt not enabled.
		1	Periodic Day interrupt enabled.
3	HOURINTEN		Periodic Hour interrupt enable.
		0	Periodic Hour interrupt not enabled.
		1	Periodic Hour interrupt enabled.
2	MININTEN		Periodic Minute interrupt enable.
		0	Periodic Minute interrupt not enabled.
		1	Periodic Minute interrupt enabled.
1	SECINTEN		Periodic Second interrupt enable.
		0	Periodic Second interrupt not enabled.
		1	Periodic Second interrupt enabled.
0	MSINTEN		Periodic Millisecond interrupt enable.
		0	Periodic Milisecond interrupt not enabled.
		1	Periodic Milisecond interrupt enabled.

4.3.2.20 RTC Compensation Register (RTCDRIFT)

Every hour on the hour, a positive or negative number of milliseconds is added to the milliseconds register to compensate for inaccuracy in the 32.768kHz crystal based on the value of COMP[3:0]. If this value is 0 then no compensation will be applied.

NOTE: Any positive compensation value must not be a multiple of 10.

The RTC compensation register (RTCDRIFT) is shown in Figure 4-22 and described in Table 4-25.

Figure 4-22. RTC Compensation Register (RTCDRIFT)

15	14 13	12	11 8	7 4	3 0
DRIFT	Reserved	COMP3	COMP2	COMP1	COMP0
RW-0	R-0	RW-0	RW-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-25. RTC Compensation Register (RTCDRIFT) Field Descriptions

Bit	Field	Value	Description
15	DRIFT		Positive or Negative Compensation.
		0	Negative compensation.
		1	Positive compensation.
14-13	Reserved	0	Reserved.
12	COMP3		Digit 3 of Compensation in BCD format.
		0	Digit 3 of Compensation is 0.
		1	Digit 3 of Compensation is 1.
11-8	COMP2	0-9	Digit 2 of Compensation register in BCD format.
7-4	COMP1	0-9	Digit 1 of Compensation register in BCD format.
3-0	COMP0	0-9	Digit 0 of Compensation register in BCD format.

4.3.2.21 RTC Oscillator Register (RTCOSC)

The RTC oscillator register (RTCOSC) is shown in Figure 4-23 and described in Table 4-26.

Figure 4-23. RTC Oscillator Register (RTCOSC)

15	14 4	3 0
RTCRESET	Reserved	OSCRES
W-0	R-0	RW-1011b

LEGEND: R/W = Read/Write; R = Read only; W = Write only; *n* = value after reset

Table 4-26. RTC Oscillator Register (RTCOSC) Field Descriptions

Bit	Field	Value	Description
15	RTCRESET		RTC software reset. The RTC only resets when this bit is set. The RTC is not reset when RESTN goes low. Once set, this bit is cleared by the RTC. Do not access any RTC register for three 32.768kHz clock cycles after setting this bit.
		0	RTC not reset.
		1	RTC reset.
14-4	Reserved	0	Reserved.
3-0	OSCRES	0-Fh	Value of the oscillator cell's internal resistor. The default (reset state) is 1011b and this gives faster startup but higher power. Once the oscillator is running it can be changed to 1000b for lower power consumption.

4.3.2.22 RTC Power Management Register (RTCPMGT)

The RTC power management register (RTCPMGT) is shown in Figure 4-24 and described in Table 4-27.

Figure 4-24. RTC Power Management Register (RTCPMGT)

15	5	4	3	2	1	0
Reserved		WU_DOUT	WU_DIR	BG_PD	LDO_PD	RTCCLKOUTEN
R-0		RW-1	RW-0	RW-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-27. RTC Power Management Register (RTCPMGT) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved.
4	WU_DOUT		Wake-up output, active low/open-drain.
		0	WAKEUP pin driven low.
		1	WAKEUP pin is in high impedance.
3	WU_DIR		Wake-up pin direction control.
		0	WAKEUP pin is configured as input.
		1	WAKEUP pin is configured as output.
			Note: The WAKEUP pin, when configured as an input, is active high. When it is configured as an output, it is open-drain and thus it should have an external pullup and it is active low.
2	BG_PD		[Not supported on this device. Reserved for compatibility with future devices]. Bandgap power down.
		0	Normal.
		1	Reserved.
1	LDO_PD		[Not supported on this device. Reserved for compatibility with future devices]. LDO power down.
		0	Normal.
		1	Reserved.
0	RTCCLKOUTEN		Clockout output enable.
		0	RTC clock output disabled.
		1	RTC clock output enabled.

4.3.2.23 RTC LSW Scratch Register 1 (RTCSCR1)

The RTC Scratch Registers are general purpose memory that can be used to store a value that will be preserved even when the DSP power is off.

The RTC LSW scratch register 1 (RTCSCR1) is shown in Figure 4-25 and described in Table 4-28.

Figure 4-25. RTC LSW Scratch Register 1 (RTCSCR1)

15	0
SCRATCH0	
RW-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-28. RTC LSW Scratch Register 1 (RTCSCR1) Field Descriptions

Bit	Field	Value	Description
15-0	SCRATCH0	0-FFFFh	Scratch registers, available to program.

4.3.2.24 RTC MSW Scratch Register 2 (RTCSCR2)

The RTC MSW scratch register 2 (RTCSCR2) is shown in Figure 4-26 and described in Table 4-29.

Figure 4-26. RTC MSW Scratch Register 2 (RTCSCR2)

15	0
SCRATCH2	
RW-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-29. RTC MSW Scratch Register 2 (RTCSCR2) Field Descriptions

Bit	Field	Value	Description
15-0	SCRATCH2	0-FFFFh	Scratch registers, available to program.

4.3.2.25 RTC LSW Scratch Register 3 (RTCSCR3)

The RTC LSW scratch register 3 (RTCSCR3) is shown in Figure 4-27 and described in Table 4-30.

Figure 4-27. RTC LSW Scratch Register 3 (RTCSCR3)

15	0
SCRATCH3	
RW-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-30. RTC LSW Scratch Register 3 (RTCSCR3) Field Descriptions

Bit	Field	Value	Description
15-0	SCRATCH3	0-FFFFh	Scratch registers, available to program.

4.3.2.26 RTC MSW Scratch Register 4 (RTCSCR4)

The RTC MSW scratch register 4 (RTCSCR4) is shown in Figure 4-28 and described in Table 4-31.

Figure 4-28. RTC MSW Scratch Register 4 (RTCSCR4)

15	0
SCRATCH4	
RW-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-31. RTC MSW Scratch Register 4 (RTCSCR4) Field Descriptions

Bit	Field	Value	Description
15-0	SCRATCH4	0-FFFFh	Scratch registers, available to program.

32-Bit Timer/Watchdog Timer

This chapter describes the features and operations of the timers for the device.

Topic **Page**

5.1 Introduction

The following information describes the operation of the three 32-bit software programmable timers in the digital signal process (DSP). Each timer can be used as a general-purpose (GP) timer. Timer2 also contains a 16-bit Watchdog (WD) timer, which shares the same clock gating bit as the GP but works independently.

5.1.1 Purpose of the Timers

General purpose timers are typically used to provide interrupts to the CPU to schedule periodic tasks or a delayed task. The general-purpose (GP) timers are 32-bit timers with a 13-bit prescaler that divides the source clock and uses this scaled value as a reference clock. These timers can be used to generate periodic interrupts.

Watchdog timers are used to reset the CPU in the event of a deadlocked state, such as a non-exiting code loop. This device includes a timer that functions as a timer or a watchdog timer simultaneously, Timer2. This watchdog timer is a 32-bit timer composed of a 16-bit counter with a 16-bit prescaler that divides the CPU system clock and uses this scaled value as a reference clock. The programmer must continuously service the watchdog timer to prevent it from resetting the device. Once the code fails to service the watchdog timer due to a deadlock or non-exiting code loop, the watchdog expires and resets the device.

5.1.2 Features

32-bit Timers:

- 32-bit programmable count down timer
- 13-bit prescaler divider
- Auto reload option
- Generates a single interrupt to the CPU. The interrupt is individually latched to determine which timer triggered the interrupt
- Interrupt can be used for DMA event

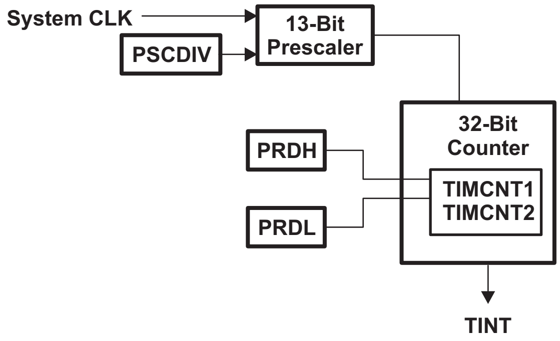
Watchdog Timer:

- 16-bit programmable count down timer
- 16-bit prescaler divider
- Lock registers require a specific sequence of keys to enable and change the watchdog settings. These sequence of keys prevent loose pointers from corrupting the state of the watchdog.
- Generated an active low pulse to the hardware reset when the Watchodg timer expires.

5.1.3 Functional Timer Block Diagram

A block diagram of the timer is shown in Figure 5-1. Detailed information about the architecture and operation of the timers is in Section 5.3.

Figure 5-1. The Architecture and Operation of the GP Timers



5.2 General-Purpose Timer

All three timers (Timer0, Timer1, and Timer2) can be used as GP timers. Timer2 can also be used as a WD timer. To use the WD timer see Section 5.3.

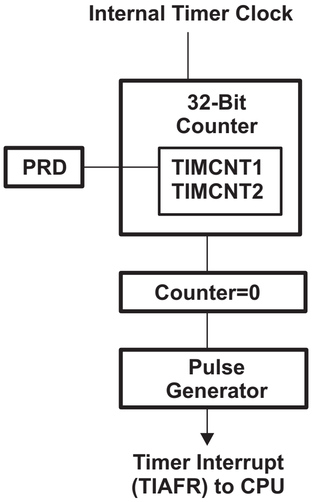
5.2.1 General-Purpose Timer Clock Control

The clock source to the GP timer and to the WD timer is driven by the system clock. This clock source determines the speed of the timer since the timer counts down in units of source clock cycles. The source clock for the GP timer can be divided down by a 13-bit prescaler and uses this scaled value as the reference clock of the timer. Each GP timer has its own 13-bit prescaler. When determining the period and prescaler setting for the timer, choose the desired period in units of source clock cycles.

5.2.2 Using the 32-bit General Purpose Timer

The general-purpose timers consist of a 32-bit timer with a 13-bit prescaler. Figure 5-2 shows a high-level diagram of the timer.

Figure 5-2. 32-Bit GP Timer With a 13-Bit Prescaler



Each GP timer has a count register (TIMCNT*n*) which consists of two 16-bit words (TIMCNT1 and TIMCNT2) and a period register (TIMPRD*n*) which also consists of two 16-bit words (TIMPRD1 and TIMPRD2). When the timer is set to start the contents of the TIMPRD*n* register is loaded into the TIMCNT register and begins to count down. A timer control register (TCR) controls the operation of the timer.

The Prescaler Divider (PSCDIV), located in the TCR, is used to divide the internal CPU clock.

When the START bit is set to 1 in the TCR, the contents of the Timer Period Registers (TIMPRD1 and TIMPRD2) are loaded into the concatenated Timer Counter Registers (TIMCNT1 and TIMCNT2) and the timer starts to count down with every cycle of the prescale divided clock. When TIMCNT1 and TIMCNT2 reach 0, the timer sends an interrupt request (TINT) to the CPU and a DMA event to the four DMAs.

The timer can be configured in auto-reload mode by setting the AUTORELOAD bit in the TCR. In this mode, the timer counter is reloaded with the timer period register when the timer counter reaches 0 and the timer re-starts its count down.

5.3 Watchdog Timer

This section describes the timer in the watchdog (WD) timer mode. Only Timer2 can be used as a Watchdog timer. This timer can also be used as general-purpose timer. To use it as a general-purpose timer, see Section 5.2.

5.3.1 Watchdog Timer Function

The watchdog timer function consists of a 16-bit main counter preceded by 16-bit prescaler. The combination of the 16-bit counter with the 16-bit prescaler allows for a maximum countdown value of 4,294,967,296. As a watchdog, the timer can be used to prevent system lockup when the software becomes deadlocked or trapped in a loop with no controlled exit. When the counter expires, a hardware reset is generated. To prevent the hardware reset from being generated, the watchdog must be serviced periodically before the counter expires. The service reinitializes the counter to its starting value and starts counting down again.

After a hardware reset occurs, the watchdog timer is disabled, allowing a flexible period of time for code to be loaded into the on-chip memory. To configure the watchdog timer, each control register has a corresponding lock register that requires a specific key sequence (with two or three keys written in order) to unlock that control register. Once the appropriate key sequence has been written to the lock register, then the corresponding control register is unlocked and can be written. After the control register is written, it is automatically locked. To write the register, the sequence must be repeated to unlock the register. If the wrong value is written for any of the keys, then it is necessary to restart at the first key. Each control and lock register pair has its own state machine to keep track of the key sequence. Therefore, it is possible to write the 1st key to each of the 4 lock registers, and then proceed to write the 2nd key to each of the 4 lock registers, and so on until all of the keys have been written. In other words, each control register must have it's specific keys written in order; but the CPU can perform other things while writing the keys in order. In the watchdog timer mode, the timer requires a periodic execution of this unlock/write sequence to reinitialize the watchdog counter to its starting value. Without this periodic servicing, the watchdog timer counter reaches zero and causes a watchdog timeout event.

When the timeout event occurs, the watchdog timer resets the entire chip with the exception of the RTC (real time clock).

5.3.2 Watchdog Timer Operation

The Watchdog (WD) timer function is enabled when:

- The Start Value and Prescale registers have been unlocked.
- The Start Value and Prescale registers have been programmed.
- The Watchdog Enable Lock register is unlocked.
- A 1 is written to bit 0 of the Watchdog Enable register.

There are example codes in the Chip Support Library (CSL) on how to program GP and WD timers. After the Watchdog timer is enabled, a write to the Watchdog Kick register when it is unlocked starts the count down. To enable the watchdog timer, a certain sequence of events must be followed. First, to configure the watchdog timer, each control register has a lock register that requires a specific key sequence (with two or three keys written in the proper order) to unlock the corresponding control register. Table 5-1 details the unlock sequence for the watchdog lock registers:

Table 5-1. Unlock Sequence for the Watchdog Lock Registers

Address	Register	First Key Sequence	Second Key Sequence	Third Key Sequence
1880h	Watchdog Kick Lock Register	5555h	AAAAh	n/a
1884h	Watchdog Start Value Lock Register	6666h	BBBBh	n/a
1888h	Watchdog Enable Lock Register	7777h	CCCCh	DDDDh
188Ch	Watchdog Prescale Lock Register	5A5Ah	A5A5h	n/a

Once the Start Value and Prescale register have been unlocked with their corresponding two word sequence and programmed for the desired interval, the Watchdog is enabled by unlocking the Enable Lock Register with a sequence of three words (7777h, CCCCh, and DDDDh) and writing a 1 on the EN bit (bit 0) of the WDEN register. The Watchdog starts counting down by unlocking the Kick Lock register and writing any value into the Kick register. At this time, the counter starts to count down from the programmed start value. The Watchdog counts down by first decrement the Prescale register by one with each CPU clock cycle. After the watchdog Prescale register reaches 0, the internal watchdog counter register is decremented by one and the Prescale register restarts its countdown from its programmed value. The counter will reach zero when the counter value and pre-scalar value are exhausted, consequently, triggering a hardware reset of the device. To prevent the hardware reset from occurring, the Kick Lock register must be unlocked and the Kick register bit 0 set to 1 to restart the countdown before the countdown is exhausted.

Once the Watchdog is enabled, it can be disabled with software by unlocking the Enable Lock Register with the three word sequence and setting the EN bit (bit 0) of the WDEN register to 0.

5.4 Reset Considerations

The timers will be reset upon a hardware reset. The timers may also be reset by software reset (see PER_RESET). However, the timer interrupt aggregation flag register (TIAFR) is not affected by software reset of the timers.

5.4.1 Hardware Reset Considerations

When a hardware reset is asserted, all timer registers, including the TIAFR register, are set to their default values.

5.5 Interrupt Support

The general-purpose timer has a timer interrupt signal. The timer interrupt request is sent to the CPU when the main count register (TIMCNT1 and TIMCNT2) counts down to 0. The same interrupt signal is also routed to the DMAs and can be used as a DMA trigger event.

The TIAFR latches each timer's interrupt signal when the timer counter expires. Using this register, the programmer can determine which of the three timers generated the timer interrupt since the bits in the TIAFR are OR'ed together and sent to the DSP as a single interrupt. Each timer interrupt flag needs to be cleared by the CPU with a write of "1" to the corresponding flag bit.

5.6 Registers

Table 5-2 through Table 5-6 list the memory mapped registers associated with the 3 Timers.

The timer registers can be accessed by the CPU at the 16-bit addresses.

Table 5-2. Watchdog Timer Registers

CPU Word Address	Acronym	Register Description	Section
1880h	WDKCKLK	Watchdog Kick Lock Register	Section 5.6.1
1882h	WDKICK	Watchdog Kick Register	Section 5.6.2
1884h	WDSVLR	Watchdog Start Value Lock Register	Section 5.6.3
1886h	WDSVR	Watchdog Start Value Register	Section 5.6.4
1888h	WDENLOK	Watchdog Enable Lock Register	Section 5.6.5
188Ah	WDEN	Watchdog Enable Register	Section 5.6.6
188Ch	WDPSLR	Watchdog Prescale Lock Register	Section 5.6.7
188Eh	WDPS	Watchdog Prescale Register	Section 5.6.8

Table 5-3. General-Purpose Timer 0 Registers

CPU Word Address	Acronym	Register Description	Section
1810h	TCR	Timer 0 Control Register	Section 5.6.9
1812h	TIMPRD1	Timer 0 Period Register 1	Section 5.6.10
1813h	TIMPRD2	Timer 0 Period Register 2	Section 5.6.11
1814h	TIMCNT1	Timer 0 Counter Register 1	Section 5.6.12
1815h	TIMCNT2	Timer 0 Counter Register 2	Section 5.6.13

Table 5-4. General-Purpose Timer 1 Registers

CPU Word Address	Acronym	Register Description	Section
1850h	TCR	Timer 1 Control Register	Section 5.6.9
1852h	TIMPRD1	Timer 1 Period Register 1	Section 5.6.10
1853h	TIMPRD2	Timer 1 Period Register 2	Section 5.6.11
1854h	TIMCNT1	Timer 1 Counter Register 1	Section 5.6.12
1855h	TIMCNT2	Timer 1 Counter Register 2	Section 5.6.13

Table 5-5. General-Purpose Timer 2 Registers

CPU Word Address	Acronym	Register Description	Section
1890h	TCR	Timer 2 Control Register	Section 5.6.9
1892h	TIMPRD1	Timer 2 Period Register 1	Section 5.6.10
1893h	TIMPRD2	Timer 2 Period Register 2	Section 5.6.11
1894h	TIMCNT1	Timer 2 Counter Register 1	Section 5.6.12
1895h	TIMCNT2	Timer 2 Counter Register 2	Section 5.6.13

Table 5-6. Timer Interrupt Aggregation Register

CPU Word Address	Acronym	Register Description	Section
1C14h	TIAFR	Timer Interrupt Aggregation Flag Register	Section 5.6.14

5.6.1 Watchdog Kick Lock Register (WDKCKLK)

The watchdog kick lock register (WDKCKLK) is shown in Figure 5-3 and described in Table 5-7.

Figure 5-3. Watchdog Kick Lock Register (WDKCKLK)

15 … 0
KICKLOK
RW-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 5-7. Watchdog Kick Lock Register (WDKCKLK) Field Descriptions

Bit	Field	Value	Description
15-0	KICKLOK	0-FFFFh	Used to unlock the Watchdog Kick Register. A 2 word key sequence must be written to this register. The following keys must be written in this order: Key 1 = 5555h and Key 2 = AAAAh. When this is written, the Kick register can now be written. When reading back the WDKCKLK register, the value that is returned in bits [1:0] give the current state of the lock state machine where 00 = Idle/waiting for Key 1; 01 = Waiting for key 2; 11 = unlocked.

5.6.2 Watchdog Kick Register (WDKICK)

The watchdog kick register (WDKICK) is shown in Figure 5-4 and described in Table 5-8.

Figure 5-4. Watchdog Kick Register (WDKICK)

15 … 0
KICK
RW-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 5-8. Watchdog Kick Register (WDKICK) Field Descriptions

Bit	Field	Value	Description
15-0	KICK	0-FFFFh	A write to the kick register when it is unlocked causes the Watchdog counter to be reloaded with the value in the WD Start Value register and start counting down again. It does not matter what value is written. Reading the register returns the current value of the counter.

5.6.3 *Watchdog Start Value Lock Register (WDSVLR)*

The watchdog start value lock register (WDSVLR) is shown in Figure 5-5 and described in Table 5-9.

Figure 5-5. Watchdog Start Value Lock Register (WDSVLR)

15	0
STVALLOK	
RW-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 5-9. Watchdog Start Value Lock Register (WDSVLR) Field Descriptions

Bit	Field	Value	Description
15-0	STVALLOK	0-FFFFh	Used to unlock the Watchdog Start Value Register. A 2 word key sequence must be written to this register. The following keys must be written in this order: Key 1 = 6666h and Key 2 = BBBBh. When this is written, the WD Start Value register can now be input. When reading back the WDSVLR register, the value that is returned in bits [1:0] give the current state of the lock state machine where 00 = Idle/waiting for Key 1; 01 = Waiting for key 2; 11 = unlocked.

5.6.4 *Watchdog Start Value Register (WDSVR)*

The watchdog start value register (WDSVR) is shown in Figure 5-6 and described in Table 5-10.

Figure 5-6. Watchdog Start Value Register (WDSVR)

15	0
STRTVAL	
RW-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 5-10. Watchdog Start Value Register (WDSVR) Field Descriptions

Bit	Field	Value	Description
15-0	STRTVAL	0-FFFFh	The value written to this register is what is loaded into the WD counter when the kick register is written to. A read of this register will return the Start Value for the counter.

5.6.5 Watchdog Enable Lock Register (WDENLOK)

The watchdog enable lock register (WDENLOK) is shown in Figure 5-7 and described in Table 5-11.

Figure 5-7. Watchdog Enable Lock Register (WDENLOK)

15 ... 0
ENLOK
RW-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 5-11. Watchdog Enable Lock Register (WDENLOK) Field Descriptions

Bit	Field	Value	Description
15-0	ENLOK	0-FFFFh	Used to unlock the Watchdog Enable Register. A 3 word key sequence must be written to this register. The following keys must be written in this order: Key 1 = 7777h, Key 2 = CCCCh, and Key 3 = DDDDh. When this is written, the WDENLOK register can now be input. When reading back the WDENLOK register, the value that is returned in bits [1:0] give the current state of the lock state machine where 00 = Idle/waiting for Key 1; 01 = Waiting for key 2; 10 = Waiting for Key 3; 11 = unlocked.

5.6.6 Watchdog Enable Register (WDEN)

The watchdog enable register (WDEN) is shown in Figure 5-8 and described in Table 5-12.

Figure 5-8. Watchdog Enable Register (WDEN)

15 ... 1	0
Reserved	EN
R-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 5-12. Watchdog Enable Register (WDEN) Field Descriptions

Bit	Field	Value	Description
15-1	Reserved	0	Reserved
0	EN		used to enable/disable the WD timer. When enabled, the counter begins counting down when the Watchdog Kick register is written, if the counter is allowed to reach 0 the chip will be reset.
		0	Watchdog Timer is disabled
		1	Watchdog Timer is enabled.

5.6.7 Watchdog Prescaler Lock Register (WDPSLR)

The watchdog prescaler lock register (WDPSLR) is shown in Figure 5-9 and described in Table 5-13.

Figure 5-9. Watchdog Prescaler Lock Register (WDPSLR)

15	0
PSLOK	
RW-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 5-13. Watchdog Prescaler Lock Register (WDPSLR) Field Descriptions

Bit	Field	Value	Description
15-0	PSLOK	0-FFFFh	Used to unlock the Watchdog Prescaler Register. A 2 word key sequence must be written to this register. The following keys must be written in this order: Key 1 = 5A5Ah and Key 2 = A5A5h. When this is written, the WDPSLR register can now be loaded. When reading back the WDPSLR register, the value that is returned in bits [1:0] give the current state of the lock state machine where 00 = Idle/waiting for Key 1; 01 = Waiting for key 2; 11 = unlocked.

5.6.8 Watchdog Prescaler Register (WDPS)

The watchdog prescaler register (WDPS) is shown in Figure 5-10 and described in Table 5-14.

Figure 5-10. Watchdog Prescaler Register (WDPS)

15	0
PS	
RW-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 5-14. Watchdog Prescaler Register (WDPS) Field Descriptions

Bit	Field	Value	Description
15-0	PS	0-FFFFh	The WD Prescaler register stores the start value for the WD Prescaler. Each time the PS register counts down to 0 the WD counter is decremented by 1. A read will return the last value written to this register (the Prescaler start value)

5.6.9 Timer n Control Register (TCR)

The timer *n* control register (TCR) is shown in Figure 5-11 and described in Table 5-15.

Figure 5-11. Timer *n* Control Register (TCR)

15	14 6	5 2	1	0
TIMEN	Reserved	PSCDIV	AUTORELOAD	START
RW-0	R-0	RW-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 5-15. Timer *n* Control Register (TCR) Field Descriptions

Bit	Field	Value	Description
15	TIMEN		This enables/disables both the Prescaler and the main counter.
		0	Timer counters are disabled
		1	Timer counters and prescaler are enabled
14-6	Reserved	0	Reserved.
5-2	PSCDIV	0-Fh	Prescaler divider. The range is 0000 = divide by 2 to 1100 = divide by 8192
1	AUTORELOAD		automatically reloads the counter when it reaches 0
		0	Auto Reload is disabled
		1	Auto Reload is enabled
0	START	0	When written to this bit loads and starts the counter.
		0	Stops the countdown
		1	Loads and starts the counter to counting down

5.6.10 Timer n Period Register 1 (TIMPRD1)

The timer *n* period register 1 (TIMPRD1) is shown in Figure 5-12 and described in Table 5-16.

Figure 5-12. Timer *n* Period Register 1 (TIMPRD1)

15	0
PRD1	
RW-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 5-16. Timer *n* Period Register 1 (TIMPRD1) Field Descriptions

Bit	Field	Value	Description
15-0	PRD1	0-FFFFh	The Timer period register is 32 bits wide. This is the LSW for the Timer period.

5.6.11 Timer n Period Register 2 (TIMPRD2)

The timer *n* period register 2 (TIMPRD2) is shown in Figure 5-13 and described in Table 5-17.

Figure 5-13. Timer *n* Period Register 2 (TIMPRD2)

15	0
PRD2	
RW-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 5-17. Timer *n* Period Register 2 (TIMPRD2) Field Descriptions

Bit	Field	Value	Description
15-0	PRD2	0-FFFFh	The Timer period register is 32 bits wide. This is the MSW for the Timer period.

5.6.12 Timer n Counter Register 1 (TIMCNT1)

The timer *n* counter register 1 (TIMCNT1) is shown in Figure 5-14 and described in Table 5-18.

Figure 5-14. Timer *n* Counter Register 1 (TIMCNT1)

15	0
TIM1	
RW-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 5-18. Timer *n* Counter Register 1 (TIMCNT1) Field Descriptions

Bit	Field	Value	Description
15-0	TIM1	0-FFFFh	The timer is 32bits wide. This is the LSW for the timer counter

5.6.13 Timer n Counter Register 2 (TIMCNT2)

The timer *n* counter register (TIMCNT2) is shown in Figure 5-15 and described in Table 5-19.

Figure 5-15. Timer *n* Counter Register (TIMCNT2)

15	0
TIM2	
RW-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 5-19. Timer *n* Counter Register (TIMCNT2) Field Descriptions

Bit	Field	Value	Description
15-0	TIM2	0-FFFFh	The timer is 32bits wide. This is the MSW for the timer counter

5.6.14 Timer Interrupt Aggregation Flag Register (TIAFR)

The timer interrupt aggregation flag register latches each timer (Timer0, Timer1, and Timer2) interrupt signal when the timer counter expires. Using this register, the programmer can determine which Timer generated the single Timer CPU interrupt signal. Each Timer flag needs to be cleared by the CPU with a write of '1' to the corresponding flag bit. Note that the corresponding Timer Interrupt Register must also be cleared.

The Timer Interrupt Aggregation Flag Register (TIAFR) is shown in Figure 5-16 and described in Table 5-20.

Figure 5-16. Timer Interrupt Aggregation Flag Register (TIAFR)

15	3	2	1	0
Reserved		TIM2FLAG	TIM1FLAG	TIM0FLAG
R-0		R/W1C	R/W1C	R/W1C

LEGEND: R/W1C = Read/Write 1 to Clear; R = Read only; -*n* = value after reset

Table 5-20. Timer Interrupt Aggregation Flag Register (TIAFR) Field Description

Bit	Field	Value	Description
15-3	Reserved	0	Reserved.
2	TIM2FLAG		Timer 2 interrupt flag bit. This bit latches the timer interrupt signal when the timer counter expires. You can clear this flag bit by writing a 1 to it.
		0	Timer has not generated an interrupt.
		1	Timer interrupt has occurred.
1	TIM1FLAG		Timer 1 interrupt flag bit. This bit latches the timer interrupt signal when the timer counter expires. You can clear this flag bit by writing a 1 to it.
		0	Timer has not generated an interrupt.
		1	Timer interrupt has occurred.
0	TIM0FLAG		Timer 0 interrupt flag bit. This bit latches the timer interrupt signal when the timer counter expires. You can clear this flag bit by writing a 1 to it.
		0	Timer has not generated an interrupt.
		1	Timer interrupt has occurred.

5.6.15 Clock Gating

Each timer has an individual clock gating control bit in Peripheral Clock Gating Configuration LSW Register (PCGCR) [0x1C02]. Timer0 is bit 10, Timer1 is bit 12 and Timer3 is bit 13. All timers are defaulted to active with value equals to "0".

Embedded Multimedia Card (eMMC)/Secure Digital (SD) Card Controller

This chapter describes the embedded multimedia card (eMMC) and secure digital (SD) card controller.

Topic **Page**

6.1 Introduction

This document describes the Embedded Multimedia Card (eMMC)/Secure Digital (SD) Card Controller in the digital signal processor (DSP).

6.1.1 Purpose of the Peripheral

The Embedded Multimedia Card (eMMC)/secure digital (SD) card is used in a number of applications to provide removable data storage. The eMMC/SD card controller provides an interface to eMMC, SD and SDHC external cards.

6.1.2 Features

The eMMC/SD card controller has the following features:

- Supports interface to Embedded Multimedia Cards (eMMC).
- Supports interface to secure digital (SD) memory cards.
- Supports the use of both eMMC/SD and SDIO protocols.
- Programmable clock frequency to control the timing of transfers between the eMMC/SD controller and memory card.
- 256-bit read/write FIFO to lower system overhead.
- Signaling to support direct memory access (DMA) transfers (slave).

The device includes two independent eMMC/SD card controllers.

6.1.3 Functional Block Diagram

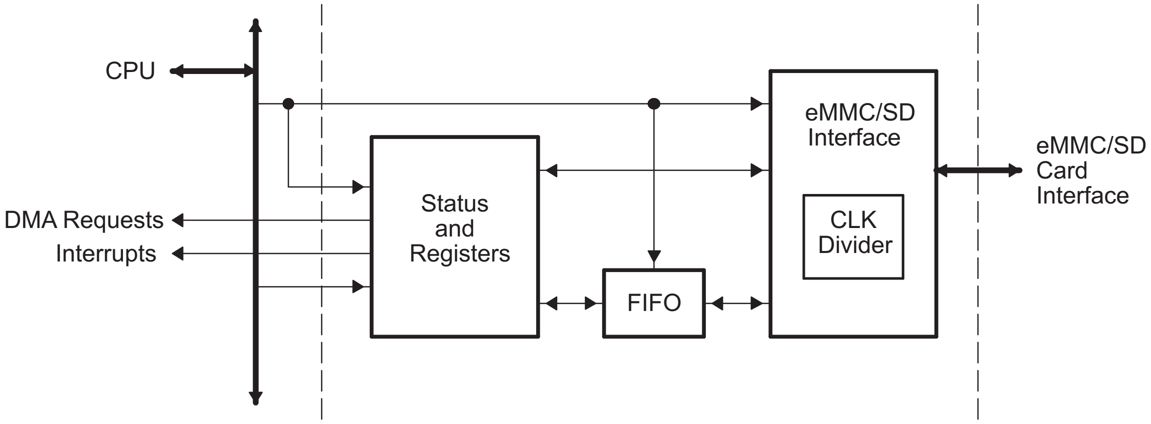
The eMMC/SD card controller transfers data between the CPU, DMA, and eMMC/SD as shown in Figure 6-1. The CPU and DMA controller can read/write the data in the card by accessing the registers in the eMMC/SD controller.

6.1.4 Supported Use Case Statement

The eMMC/SD card controller supports the following user cases:

- eMMC/SD card identification
- eMMC/SD single-block read using CPU
- eMMC/SD single-block read using DMA
- eMMC/SD single-block write using CPU
- eMMC/SD single-block write using DMA
- eMMC/SD multiple-block read using CPU
- eMMC/SD multiple-block read using DMA
- eMMC/SD multiple-block write using CPU
- eMMC/SD multiple-block write using DMA
- SDIO function

Figure 6-1. eMMC/SD Card Controller Block Diagram



6.1.5 *Industry Standard(s) Compliance Statement*

The eMMC/SD card controller will support the following industry standards (with the exception noted below):

- Embedded Multimedia Card (eMMC) Specification V3.31.
- Secure Digital (SD) Physical Layer Specification V2.0.
- Secure Digital Input Output (SDIO) Specification V2.0.

The information in this document assumes the reader is familiar with these standards.

The eMMC/SD controller does not support the SPI mode of operation.

6.2 Peripheral Architecture

The eMMC/SD controller uses the eMMC/SD protocol to communicate with the eMMC/SD cards. The eMMC/SD controller can be configured to work as an eMMC or SD controller, based on the type of card with which the controller is communicating. Figure 6-2 summarizes the eMMC/SD mode interface. Figure 6-3 illustrates how the controller is interfaced to the cards in eMMC/SD mode.

In the eMMC/SD mode, the controller supports one or more eMMC/SD cards. When multiple cards are connected, the eMMC/SD controller selects one by using identification broadcast on the data line. The following eMMC/SD controller pins are used:

- CMD: This pin is used for two-way communication between the connected card and the eMMC/SD controller. The eMMC/SD controller transmits commands to the card and the memory card drives responses to the commands on this pin.
- DAT0 or DAT0-3: eMMC cards use only one data line (DAT0) and SD cards use one or four data lines. The number of DAT pins (the data bus width) is set by the WIDTH bit in the SD Control Register (SDCTL), see Section 6.4.1).
- CLK: This pin provides the clock to the memory card from the eMMC/SD controller.

Figure 6-2. MMC/SD Controller Interface Diagram

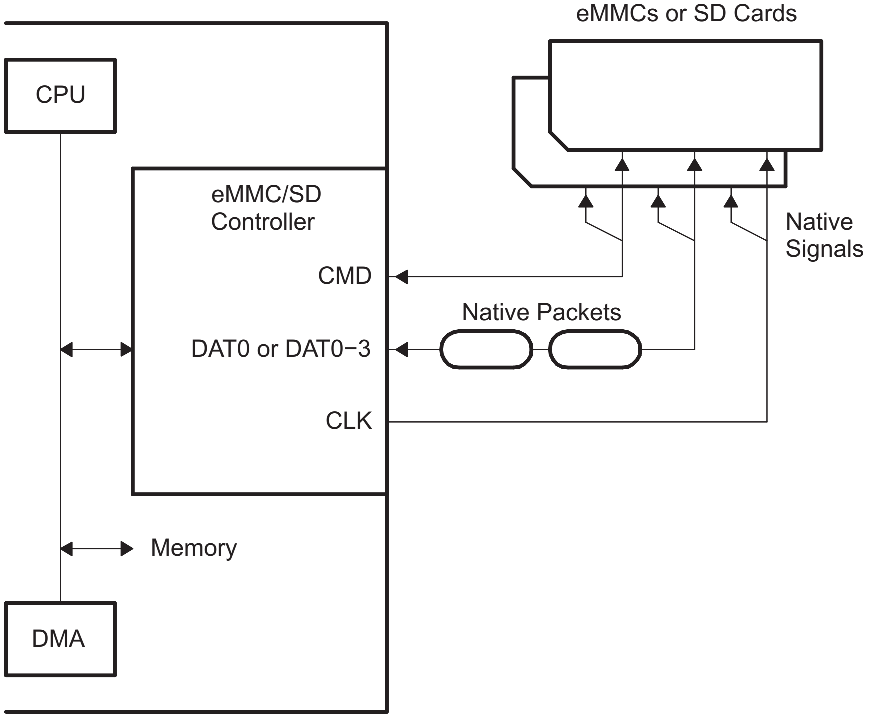
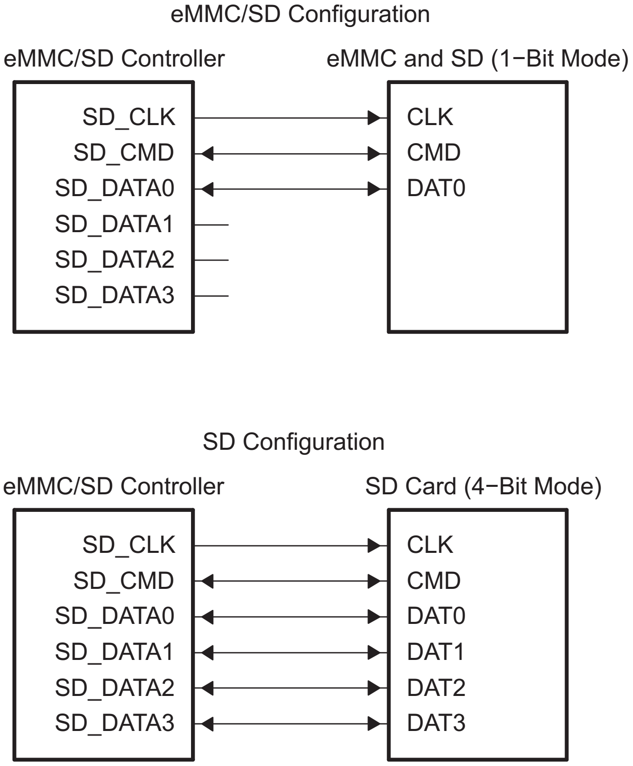


Figure 6-3. eMMC Configuration and SD Configuration Diagram



6.2.1 Clock Control

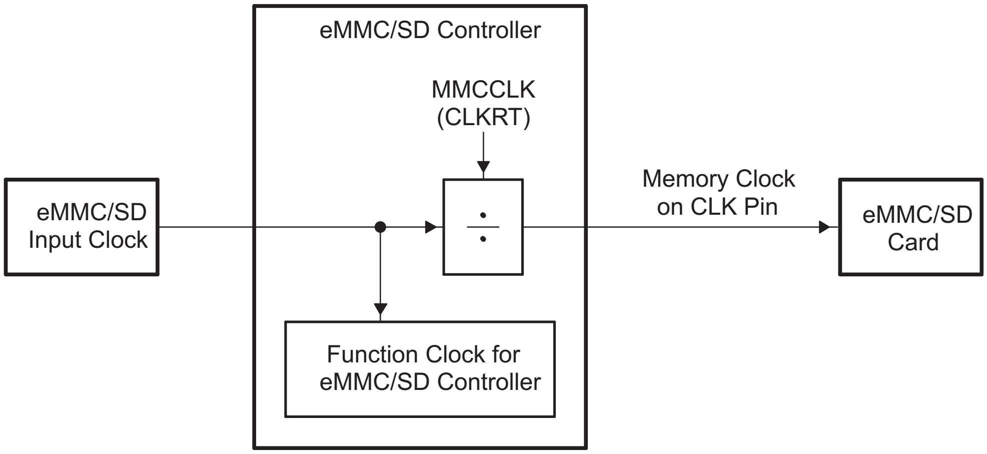
The eMMC/SD controller has two clocks: the function clock and the memory clock (see Figure 6-4).

The function clock determines the operational frequency of the eMMC/SD controller and is the input clock to the eMMC/SD controller on the device. The eMMC/SD controller is capable of operating with a function clock up to 100 MHz.

The memory clock appears on the SD_CLK pin of the eMMC/SD controller interface. The memory clock controls the timing of communication between the eMMC/SD controller and the connected memory card. The memory clock is generated by dividing the function clock in the eMMC/SD controller. The divide-down value is set by CLKRT bits in the SD memory clock control register (SDCLK) and is determined by the equation:

memory clock frequency = function clock frequency/(2 * (CLKRT + 1))

Figure 6-4. eMMC/SD Controller Clocking Diagram



(1) Maximum memory clock frequency for eMMC card can be 20 MHz.

(2) Maximum memory clock frequency for SD card can be 50 MHz.

6.2.2 Signal Descriptions

Table 6-1 shows the eMMC/SD controller pins used in each mode. The eMMC/SD protocol uses the clock, command (two-way communication between the eMMC controller and memory card), and data (DAT0 for eMMC card, DAT0-3 for SD card) pins.

Table 6-1. eMMC/SD Controller Pins Used in Each Mode

		Function	
Pin	**Type**[1]	**eMMC Communications**	**SD Communications**
CLK	O	Clock line	Clock line
CMD	I/O	Command line	Command line
DAT0	I/O	Data line 0	Data line 0
DAT1	I/O	(Not used)	Data line 1
DAT2	I/O	(Not used)	Data line 2
DAT3	I/O	(Not used)	Data line 3

[1] I = input to the eMMC/SD controller; O = output from the eMMC/SD controller.

6.2.3 Pin Multiplexing

The eMMC/SD card controller pins are multiplexed with other peripherals on the DSP. Before using the controller, the DSP should be configured to route the eMMC/SD card controller signals to the multiplexed Serial Port 0 or Serial Port 1 pins by writing to the External Bus Selection Register (EBSR). For more information on pin multiplexing, see Section 1.1, *System Control*.

NOTE: Configuring the EBSR to route the eMMC/SD0 or eMMC/SD1 signals to Serial Port0 or Serial Port1, respectively, also routes those eMMC/SD interrupts to the CPU.

6.2.4 Protocol Descriptions

The eMMC/SD controller follows the eMMC/SD protocol for completing any kind of transaction with the Embedded Multimedia Card and secure digital cards. For detailed information, refer to the supported eMMC and SD specifications in Section 6.1.5.

6.2.4.1 eMMC/SD Mode Write Sequence

Figure 6-5 and Table 6-2 show the signal activity when the eMMC/SD controller is in the eMMC/SD mode and is writing data to a memory card. Before initiating a write transfer, ensure that the block length definition in the eMMC/SD controller and the memory card are identical.

- The eMMC/SD controller sends a write command to the card.
- The card receives the command and sends responses to the command.
- The eMMC/SD controller sends a block of data to the card.
- The card sends the CRC status to the eMMC/SD controller.
- The card sends a low BUSY bit until all the data has been programmed into the flash memory inside the card.

Figure 6-5. eMMC/SD Mode Write Sequence Timing Diagram

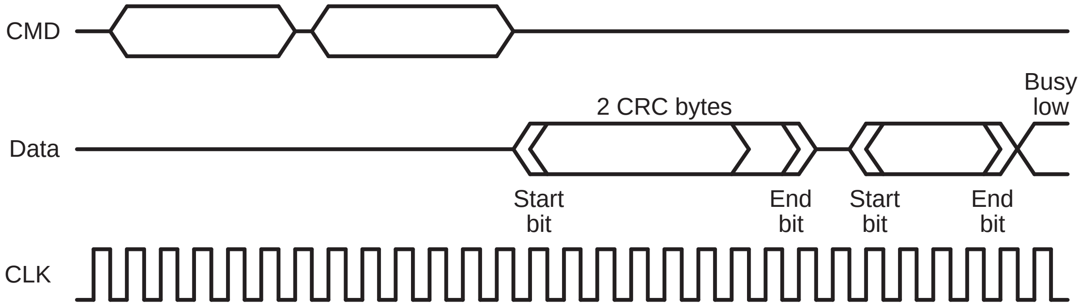


Table 6-2. eMMC/SD Mode Write Sequence

Portion of the Sequence	Description
WR CMD	Write command: A 6-byte WRITE_BLOCK command token is sent from the CPU to the card.
CMD RSP	Command response: The card sends a 6-byte response of type R1 to acknowledge the WRITE_BLOCK to the CPU.
DAT BLK	Data block: The CPU writes a block of data to the card. The data content is preceded by one start bit and is followed by two CRC bytes and one end bit.
CRC STAT	CRC status: The card sends a one byte CRC status information, which indicates to the CPU whether the data has been accepted by the card or rejected due to a CRC error. The CRC status information is preceded by one start bit and is followed by one end bit.
BUSY	BUSY bit: The CRC status information is followed by a continuous stream of low busy bits until all of the data has been programmed into the flash memory on the card.

6.2.4.2 eMMC/SD Mode Read Sequence

Figure 6-6 and Table 6-3 show the signal activity when the eMMC controller is in the eMMC/SD mode and is reading data from a memory card. Before initiating a read transfer, ensure that the block length definition in the eMMC/SD controller and the memory card are identical. In a successful read sequence, the following steps occur:

- The eMMC/SD controller sends a read command to the card.
- The card drives responses to the command.
- The card sends a block of data to the CPU.

Figure 6-6. eMMC/SD Mode Read Sequence Timing Diagram

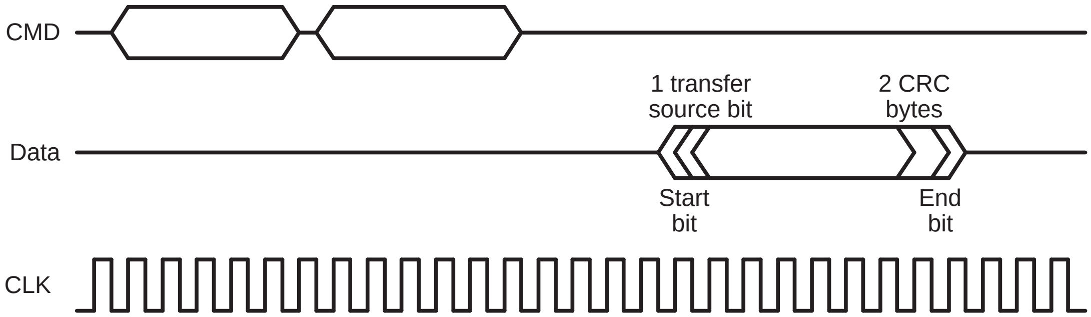


Table 6-3. eMMC/SD Mode Read Sequence

Portion of the Sequence	Description
RD CMD	Read command: A 6-byte READ_SINGLE_BLOCK command token is sent from the CPU to the card.
CMD RSP	Command response: The card sends a response of type R1 to acknowledge the READ_SINGLE_BLOCK command to the CPU.
DAT BLK	Data block: The card sends a block of data to the CPU. The data content is preceded by a start bit and is followed by two CRC byte and an end bit.

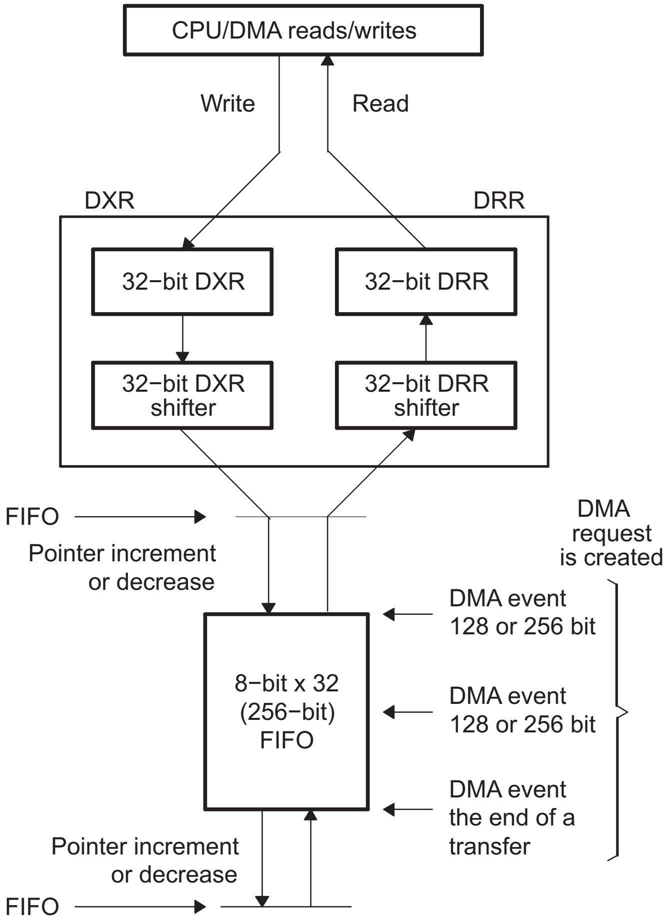
6.2.5 Data Flow in the Input/Output FIFO

The eMMC/SD controller contains a single 256-bit FIFO that is used for both reading data from the memory card and writing data to the memory card (see Figure 6-7). The FIFO is organized as 32 eight-bit entries. The conversion from the 32-bit bus to the byte format of the FIFO follows the little-endian convention (details are provided in later sections). The FIFO includes logic to generate DMA events and interrupts based on the amount of data available in the FIFO. FIFO depth (threshold) is a programmable number that describes how many bytes can be received/transmitted at a time. There are also flags set when the FIFO is full or empty. A high-level operational description is:

- Data is written to the FIFO through the SD Data Transmit Registers (SDDXR1 and 2). Data is read from the FIFO through the SD Data Receive Registers (SDDRR1 and 2).
- The ACCWD bits in the SD FIFO Control Register (SDFIFOCTL) determines the behavior of the FIFO full (FIFOFUL) and FIFO empty (FIFOEMP) status flags in the SD Status Register 1 (SDST1):
  - If ACCWD = 00b:
    - FIFO full is active when the write pointer + 4 > read pointer
    - FIFO empty is active when the write pointer - 4 < read pointer
  - If ACCWD = 01b:
    - FIFO full is active when the write pointer + 3 > read pointer
    - FIFO empty is active when the write pointer - 3 < read pointer
  - If ACCWD = 10b:
    - FIFO full is active when the write pointer + 2 > read pointer
    - FIFO empty is active when the write pointer - 2 < read pointer

– If ACCWD = 11b:
  - FIFO full is active when the write pointer + 1 > read pointer
  - FIFO empty is active when the write pointer - 1 < read pointer

Figure 6-7. FIFO Operation Diagram



Transmission of data

Step 1: Reset FIFO
Step 2: Set FIFO direction
Step 3: DMA driven transaction
Step 4: DMA sends xmit data
Step 5: If DXR ready is active, 32−bit DXR -> FIFO
Step 6: CPU driven transaction: Fill the FIFO by writing to SDDXR (only first time) or every 128 or 256−bits transmitted and DXRDYINT interrupt is generated

Reception of data

Step 1: Reset FIFO
Step 2: Set FIFO direction
Step 3: DMA driven transaction
Step 4: FIFO-> 32-bit DRR
Step 5: DRRDYINT interrupt occur when FIFO every 128 or 256−bits of data received by FIFO
Step 6: DMA reads reception data

6.2.6 Data Flow in the Data Registers (SDDRR and SDDXR)

The CPU can read 16 bits and the DMA controller can read 32 bits at a time from the FIFO by reading the SD data receive registers (SDDRR1 and 2) and, similarly, write to the FIFO by writing to the SD data transmit registers (SDDXR1 and 2). However, since the memory card is an 8-bit device, it transmits or receives one byte at a time. Figure 6-8 and Figure 6-9 shows how the data-size difference is handled by the data registers.

Figure 6-8. Little-Endian Access to SDDXR/SDDRR1 and 2 From the CPU or the DMA

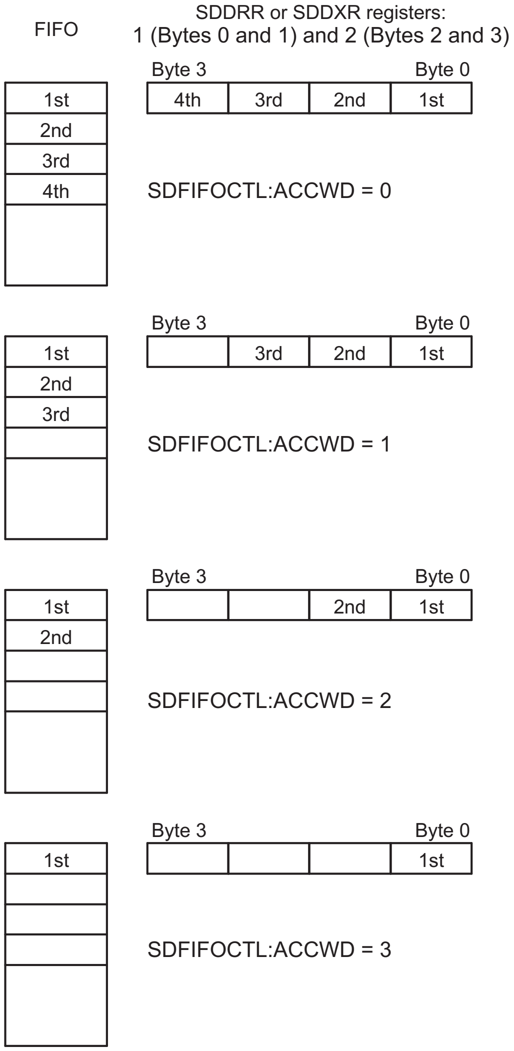
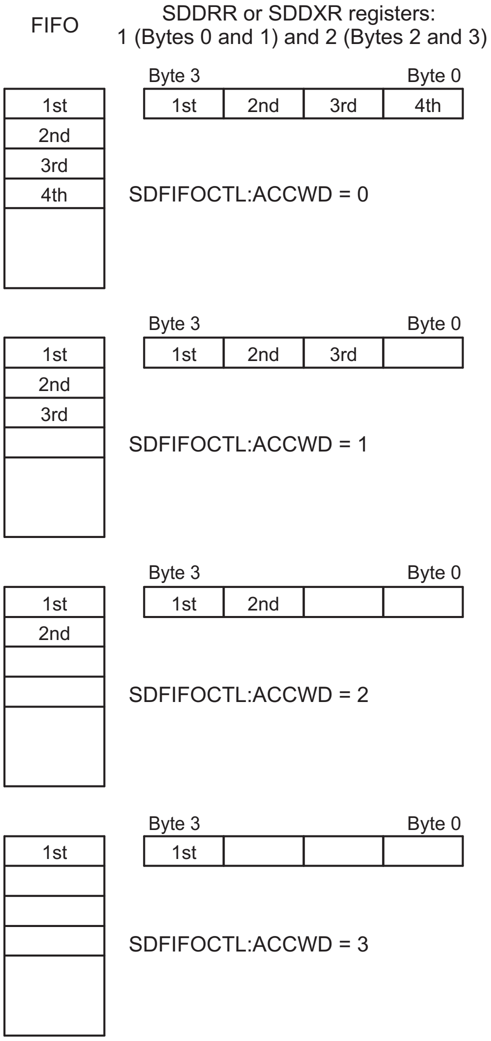


Figure 6-9. Big-Endian Access to SDDXR/SDDRR1 and 2 From the CPU or the DMA



6.2.7 *FIFO Operation During Card Read Operation*

The eMMC/SD controller supports 1-, 2-, 3-, or 4-byte reads as shown in and . The CPU makes 16-bit and the DMA makes 32-bit accesses to the SDDRR registers.

6.2.7.1 DMA Reads

The FIFO controller manages the activities of reading the data in from the card and issuing DMA read events. Each time a DMA read event is issued, a DMA read request interrupt is also generated.

Figure 6-10 provides details of the FIFO controllers operation. As data is received from the card, it is read into the FIFO. When the number of bytes of data received is equal to the level set by the FIFOLEV bits in SDFIFOCTL, a DMA read event is issued and new DMA events are disabled. Data is read from the FIFO by way of SDDRR registers (SDDRR1 should be used as the destination address in the DMA configuration). The FIFO controller continues to read in data from the card while checking for the DMA event to occur or the FIFO to become full. Once the DMA event finishes, new DMA events are enabled. If the FIFO fills up, the FIFO controller stops the eMMC/SD controller from reading anymore data until the FIFO is no longer full.

A DMA read event is also generated when the last data arrives as determined by the SD block length register (SDBLEN) and the SD number of blocks register (SDNBLK) settings. This DMA event enables the FIFO to be flushed of all the data that was read from the card.

Each time a DMA read event is generated, an interrupt (DRRDYINT) is also generated (if enabled in the SD Interrupt Mask Register (SDIM) register) and the DRRDY bit in the SD status register 0 (SDST0) is also set.

6.2.7.2 CPU Reads

The system CPU can also directly read the card data by reading the SD data receive register (SDDRR 1 and/or 2) based on the ACCWD field in the SDFIFOCTL. Data is ready to be read when the DRRDYINT interrupt is posted or when the DRRDY bit in the SDST0 register is set.

6.2.8 FIFO Operation During Card Write Operation

The eMMC/SD controller supports 1-, 2-, 3-, or 4-byte writes as shown in and . The CPU makes 16-bit and the DMA makes 32-bit accesses to the SDDXR registers.

6.2.8.1 DMA Writes

The FIFO controller manages the activities of accepting data from the CPU or DMA and passing the data to the eMMC/SD controller. The FIFO controller issues DMA write events as appropriate. However, the first DMA event has to be manually generated by setting the DMATRIG bit in the eMMC Command Register (eMMCSD2) after the desired write command is written to the eMMCSD1 register.

Figure 6-11 provides details of the FIFO controller's operation. Data is written by the DMA to the FIFO by the way of SDDXR registers (SDDXR1 should be used as the destination address in the DMA configuration). The FIFO then passes the data to the eMMC/SD controller which manages to write the data to the card. When the number of bytes of data in the FIFO is less than the level set by the FIFOLEV bits in SDFIFOCTL, a DMA write event is issued and new DMA events are disabled. The FIFO controller continues to transfer data to the eMMC/SD controller while checking for the DMA event to finish or for the FIFO to become empty. Once the DMA event completes, new DMA events are enabled. If the FIFO becomes empty, the FIFO controller informs the eMMC/SD controller.

Each time a DMA write event is generated, an interrupt (DXRDYINT) is also generated (if enabled in the SD Interrupt Mask Register (SDIM) register) and the DXRDY bit in the SD status register 0 (SDST0) is also set.

6.2.8.2 CPU Writes

The system CPU can also directly write the card data by writing to the SD data transmit register (SDDXR 1 and/or 2) based on the ACCWD field in the SDFIFOCTL. Data is ready to be written when the DXRDYINT interrupt is posted or when the DXRDY bit in the SDST0 register is set.

Figure 6-10. FIFO Operation During Card Read Diagram

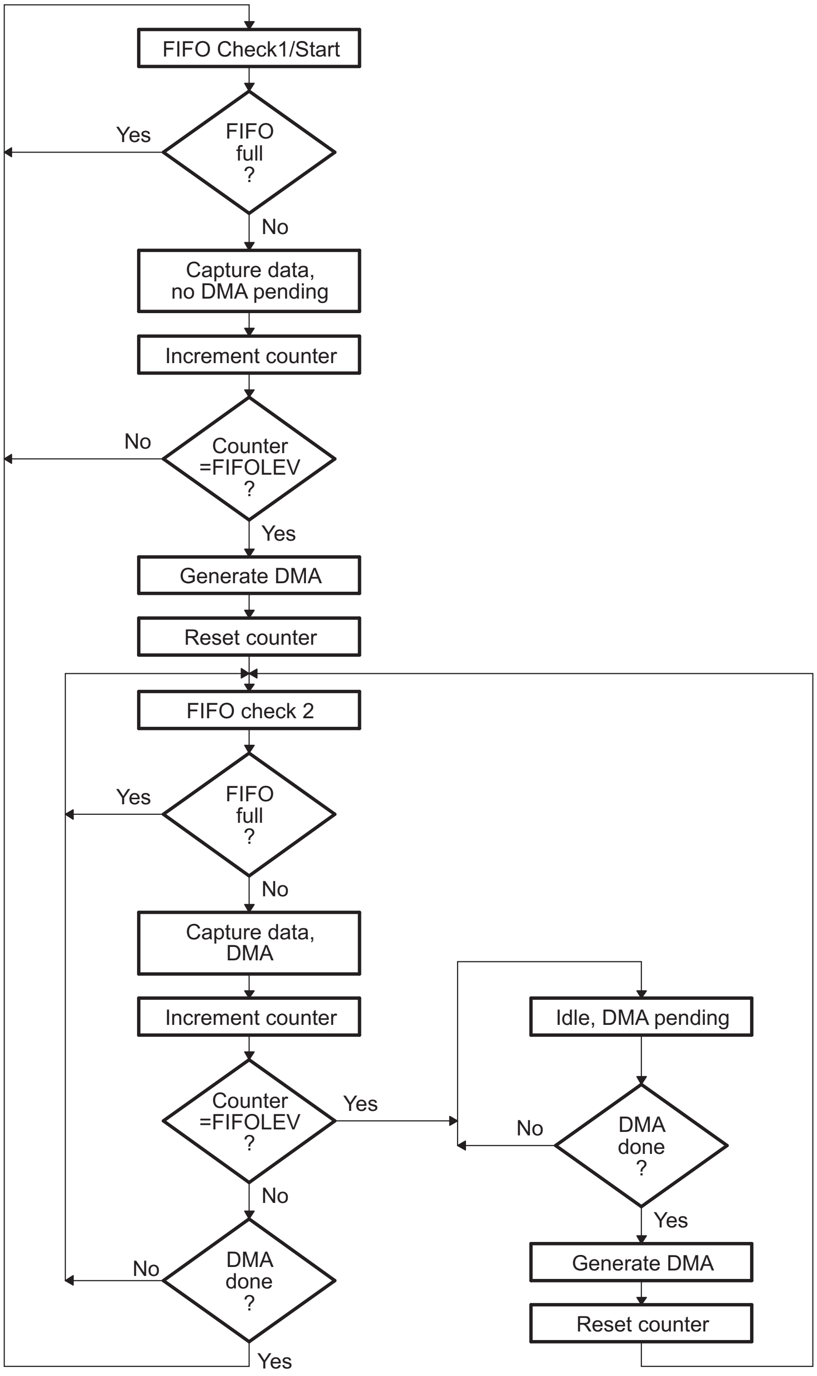
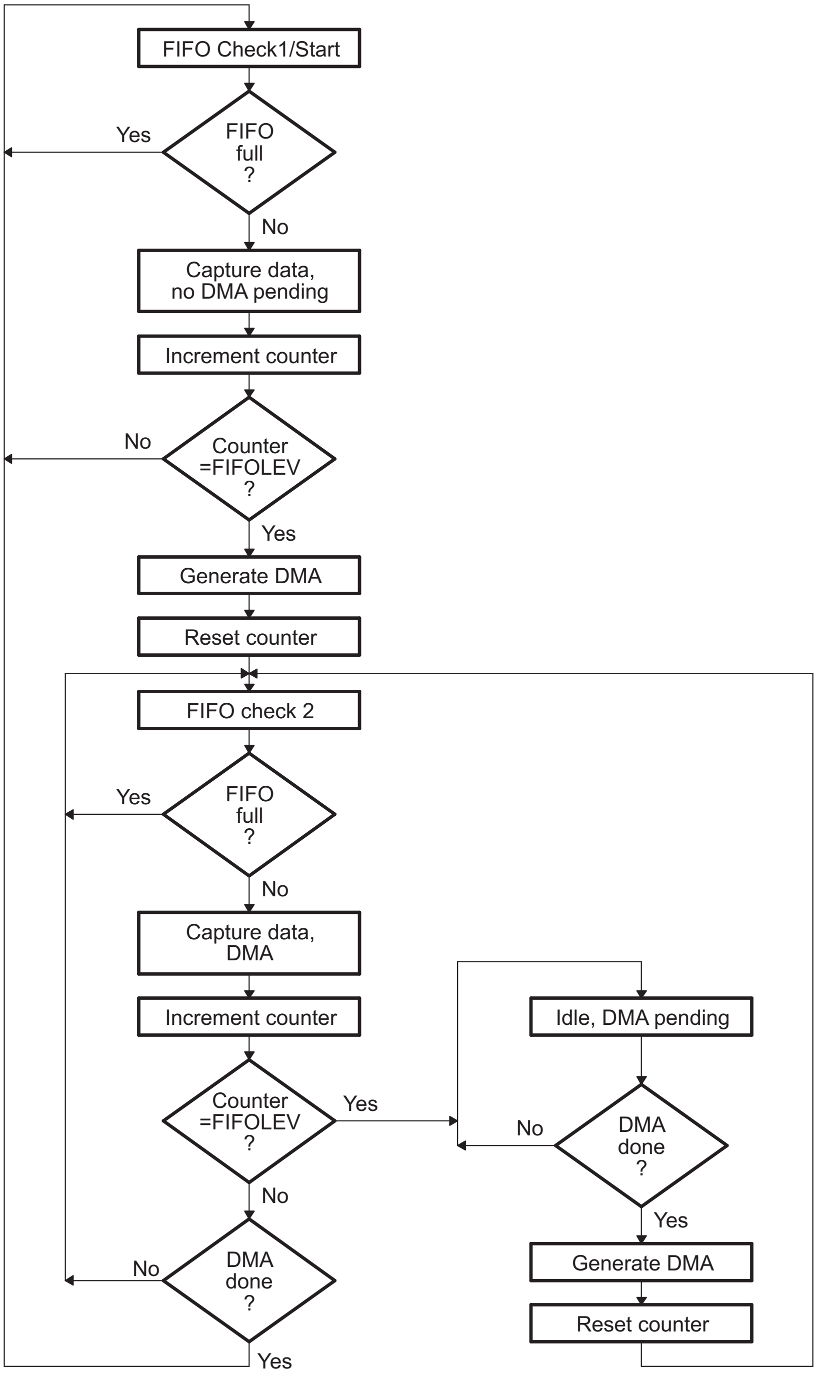


Figure 6-11. FIFO Operation During Card Write Diagram



6.2.9 Reset Considerations

The eMMC/SD controller has two reset sources: hardware reset and software reset.

6.2.9.1 Software Reset Considerations

A software reset (such as a reset generated by the emulator) will not cause the eMMC/SD controller registers to be altered. After a software reset, the eMMC/SD controller continues to operate as it was configured prior to the reset.

6.2.9.2 Hardware Reset Considerations

A hardware reset of the processor will cause the eMMC/SD controller registers to return to their default values after reset.

6.2.10 Programming and Using the SD Controller

6.2.10.1 eMMC/SD Mode Initialization

The general procedure for initializing the eMMC/SD controller is given in the following steps. Details about the registers or register bit fields to be configured in the eMMC/SD mode are in the subsequent subsections.

1. Place the eMMC/SD controller in its reset state by setting the CMDRST bit and DATRST bit in the SDCTL. After the reset, other bits in SDCTL can be set.
2. Write the required values to eMMC/SD controller registers to complete the eMMC/SD controller configuration.
3. Clear the CMDRST bit and DATRST bit in SDCTL to release the eMMC/SD controller from its reset state. It is recommended not to rewrite the values written to the other bits of SDCTL in step 1.
4. Enable the SD_CLK pin so that the memory clock is sent to the memory card by setting the CLKEN bit in the SD memory clock control register (SDCLK).

NOTE: The External Bus Selection Register must be configured to enable eMMC/SD signals at the pins as described in Section 6.2.3 before the controller can communicate with the eMMC/SD card.

6.2.10.2 Initializing the SD Control Register (SDCTL)

When operating the eMMC/SD controller in the eMMC/SD mode, the bits in the SD control register (SDCTL) affect the operation of the eMMC/SD controller. The subsections that follow help you decide how to initialize each of the control register bits.

The DATEG bits in SDCTL enable or disable edge detection on the SD_DATA3 pin. If edge detection is enabled and an edge is detected, the DATEG flag bit in the SD status register 0 (SDST0) is set. In addition, if the EDATED bit in the SD interrupt mask register (SDIM) is set, an interrupt request is generated.

In the eMMC/SD mode, the eMMC/SD controller must know how wide the data bus must be for the memory card that is connected. If an eMMC card is connected, specify a 1-bit data bus (WIDTH = 0 in SDCTL); if an SD card is connected, specify a 4-bit data bus (WIDTH = 1 in SDCTL).

To place the eMMC/SD controller in its reset state and disable it, set the CMDRST bit and DATRST bit in SDCTL. The first step of the eMMC/SD controller initialization process is to disable both sets of logic. When initialization is complete but before you enable the SD_CLK pin, enable the eMMC/SD controller by clearing the CMDRST bit and DATRST bit in SDCTL.

6.2.10.3 Initializing the Clock Controller Registers (SDCLK)

A clock divider in the eMMC/SD controller divides-down the function clock to produce the memory clock. Load the divide-down value into the CLKRT bits in the SD memory clock control register (SDCLK). The divide-down value is determined by the equation:

memory clock frequency = function clock frequency/(2 * (CLKRT + 1))

The CLKEN bit in SDCLK determines whether the memory clock appears on the SD_CLK pin. If CLKEN is cleared to 0, the memory clock is not provided except when required.

6.2.10.4 Initialize the Interrupt Mask Register (SDIM)

The bits in the SD interrupt mask register (SDIM) individually enable or disable the interrupt requests. To enable the associated interrupt request, set the corresponding bit in SDIM. To disable the associated interrupt request, clear the corresponding bit. Load zeros into the bits not used in the eMMC/SD mode.

6.2.10.5 Initialize the Time-Out Registers (SDTOR and SDTOD)

Specify the time-out period for responses using the SD response time-out register (SDTOR) and the time-out period for reading data using the SD data read time-out register (SDTOD).

When the eMMC/SD controller sends a command to a memory card, it often must wait for a response. The eMMC/SD controller can wait indefinitely or up to 255 memory clock cycles. If you load 0 into SDTOR, the eMMC/SD controller waits indefinitely for a response. If you load a nonzero value into SDTOR, the eMMC/SD controller stops waiting after the specified number of memory clock cycles and then sets a response time-out flag (TOUTRS) in the SD status register 0 (SDST0). If the associated interrupt request is enabled, the eMMC/SD controller also sends an interrupt request to the CPU.

When the eMMC/SD controller requests data from a memory card, it can wait indefinitely for that data or it can stop waiting after a programmable number of cycles. If you load 0 into SDTOD, the eMMC/SD controller waits indefinitely. If you load a nonzero value into SDTOD, the eMMC/SD controller waits the specified number of memory clock cycles and then sets a read data time-out flag (TOUTRD) in SDST0. If the associated interrupt request is enabled, the eMMC/SD controller also sends an interrupt request to the CPU.

6.2.10.6 Initialize the Data Block Registers (SDBLEN and SDNBLK)

Specify the number of bytes in a data block in the SD block length register (SDBLEN) and the number of blocks in a multiple-block transfer in the SD number of blocks register (SDNBLK).

In SDBLEN, you must define the size for each block of data transferred between the eMMC/SD controller and a memory card. The valid size depends on the type of read/write operations. A length of 0 bytes is prohibited.

For multiple-block transfers, you must specify how many blocks of data are to be transferred between the eMMC/SD controller and a memory card. You can specify an infinite number of blocks by loading 0 into SDNBLK. When SDNBLK = 0, the eMMC/SD controller continues to transfer data blocks until the transferring is stopped with a STOP_TRANSMISSION command. To transfer a specific number of blocks, load SDNBLK with a value from 1 to 65535.

For high capacity cards (2 GB and larger), by default the read/write block length of the card is 1024 bytes. Note that CMD16 (SET_BLOCK_LEN) can only set the block length up to 512 bytes even for high capacity cards. Therefore, if you want to change the block length of a high capacity card, you are limited to 512 bytes. In this case, you should discard the block length read from the CSD register in the card and set the block length up to 512 bytes.

6.2.10.7 Using the Command Registers (MMCSD1 and MMCSD2)

The MMCSD1 register can be programmed to choose the type command to be sent to the eMMC/SD card and the expected outcome of the transaction. Any writes to this register triggers the controller to send a command to the eMMC/SD card as programmed in the CMD field. This behavior makes it necessary to program the whole register in a single write.

The DMATRIG field is a write-only field in the MMCSD2 register. It is only used for write operations involving the DMA. Setting this bit field triggers the associated DMA channel to transfer data to the controller FIFO while the eMMC/SD card prepares itself for the write operation. Subsequent DMA triggers are automatically generated when the controller writes the data in the FIFO out to the eMMC/SD card and do not need the DMATRIG bit to be set. This field should only be used for write operations involving the DMA. Data read operations do not require this intervention.

NOTE: You must only write to the DMATRIG bit in MMCSD2 after the desired write command has been written to the MMCSD1 register. If this bit is written to at any other time, the controller will resend the last command (configured in the MMCSD1 register) to the card.

6.2.10.8 Monitoring Activity in the eMMC/SD Mode

This section describes registers and specific register bits that you use to obtain the status of the eMMC/SD controller in the eMMC/SD mode. The status of the eMMC/SD controller is determined by reading the bits in the SD status register 0 (SDST0) and SD status register 1 (SDST1).

6.2.10.8.1 Detecting Edges on the DAT3 Pin

The eMMC/SD controller sets the DATED bit in SDST0 if SD_DATA3 edge detection is enabled (DATEG bits are nonzero in SDCTL) and the specified edge is detected. The CPU is also notified of the SD_DATA3 by an interrupt if the interrupt request is enabled (EDATED = 1 in SDIM).

6.2.10.8.2 Detecting Level Changes on the DAT3 Pin

The DAT3ST bit in SDST1 monitors the signal level on the SD_DATA3 pin.

6.2.10.8.3 Determining Whether New Data is Available in SDDRR Registers

The eMMC/SD controller sets the DRRDY bit in SDST0 when data in the FIFO is greater than the threshold set in SDFIFOCTL. The CPU is also notified of the event by an interrupt if the interrupt request is enabled (EDRRDY = 1 in SDIM). The DRRDY flag is cleared by a read of SDDDR register.

6.2.10.8.4 Verifying that SDDXR is Ready to Accept New Data

The eMMC/SD controller sets the DXRDY bit in eMMCST0 when the amount of data in the FIFO is less than the threshold set in SDFIFOCTL. The CPU is also notified of the event by an interrupt if the interrupt request is enabled (EDXRDY = 1 in SDIM).

6.2.10.8.5 Checking for CRC Errors

The eMMC/SD controller sets the CRCRS, CRCRD, and CRCWR bits in SDST0 in response to the corresponding CRC errors of command response, data read, and data write. The CPU is also notified of the CRC error by an interrupt if the interrupt request is enabled (ECRCRS/ECRCRD/ECRCWR = 1 in SDIM).

6.2.10.8.6 Checking for Time-Out Events

The eMMC/SD controller sets the TOUTRS and TOUTRD bits in SDST0 in response to the corresponding command response or data read time-out event. The CPU is also notified of the event by an interrupt if the interrupt request is enabled (ETOUTRS/ETOUTRD = 1 in SDIM).

6.2.10.8.7 Determining When a Response/Command is Done

The eMMC/SD controller sets the RSPDNE bit in SDST0 when the response is done or, in the case of commands that do not require a response, when the command is done. The CPU is also notified of the done condition by an interrupt if the interrupt request is enabled. (ERSPDNE = 1 in SDIM).

6.2.10.8.8 Determining Whether the Memory Card is Busy

The card sends a busy signal either when waiting for an R1b-type response or when programming the last write data into its flash memory. The eMMC/SD controller has two flags to notify you whether the memory card is sending a busy signal. The two flags are complements of each other:

- BSYDNE flag in SDST0 is set if the card did not send or is not sending a busy signal when the eMMC/SD controller is expecting a busy signal (BSYEXP = 1 in eMMCSD). The interrupt by this bit is enabled by a corresponding interrupt enable bit (EBSYDNE = 1 in SDIM).
- BUSY flag in SDST1 is set when a busy signal is received from the card.

6.2.10.8.9 Determining Whether a Data Transfer is Done

The eMMC/SD controller sets the DATDNE bit in SDST0 when all the bytes of a data transfer have been transmitted/received. The DATDNE bit is polled to determine when to stop writing to the data transmit register (for a write operation) or when to stop reading from the data receive register (for a read operation). The CPU is also notified of the time-out event by an interrupt if the interrupt request is enabled (EDATDNE = 1 in SDIM).

6.2.10.8.10 Determining When Last Data has Been Written to Card (SanDisk SD cards)

Some SanDisk brand SD™ cards exhibit a behavior that requires a multiple-block write command to be terminated with a STOP (CMD12) command before the data write sequence is completed. To enable support of this function, the transfer done interrupt (TRNDNE) is provided. The TRNDNE interrupt is enabled by setting the ETRNDNE bit in SDIM. This interrupt is issued when the last byte of data (as defined by SDNBLK and SDBLEN) is transferred from the FIFO to the output shift register. The CPU should respond to this interrupt by sending a STOP command to the card. This interrupt differs from DATDNE by timing. DATDNE does not occur until after the CRC and memory programming are completed.

6.2.10.8.11 Checking For a Data Transmit Empty Condition

During transmission, a data value is passed from the SD data transmit registers (SDDXR1 and 2) to the data transmit shift register. The data is then passed from the shift register to the memory card one bit at a time. The DXEMP bit in SDST1 indicates when the shift register is empty.

Typically, the DXEMP bit is not used to control data transfers; rather, it is checked during recovery from an error condition. There is no interrupt associated with the DXEMP bit.

6.2.10.8.12 Checking for a Data Receive Full Condition

During reception, the data receive shift register accepts a data value one bit at a time. The entire value is then passed from the shift register to the SD data receive registers (SDDRR1 and 2). The DRFUL bit in SDST1 indicates when the shift register is full; no new bits can be shifted in from the memory card.

Typically, the DRFUL bit is not used to control data transfers; rather, it is checked during recovery from an error condition. There is no interrupt associated with the DRFUL bit.

6.2.10.8.13 Checking the Status of the SD_CLK Pin

Read the CLKSTP bit in SDST1 to determine whether the memory clock has been stopped on the SD_CLK pin.

6.2.10.8.14 Checking the Remaining Block Count During a Multiple-Block Transfer

During a transfer of multiple data blocks, the SD number of blocks counter register (SDNBLC) indicates how many blocks are remaining to be transferred. SDNBLC is a read-only register.

6.2.11 Interrupt Support

6.2.11.1 Interrupt Events and Requests

The eMMC/SD controller generates the interrupt requests described in Table 6-4. When an interrupt event occurs, its flag bit is set in the SD status register 0 (SDST0). If the enable bits corresponding to each flag are set in the SD interrupt mask register (SDIM), an interrupt request is generated. All such requests are multiplexed to a single eMMC/SD interrupt request from the eMMC/SD controller to the CPU.

The eMMC/SD interrupts can be masked into the CPU by means of the 4 programmable interrupt sources. This is accomplished through the External Bus Selection Register. Selecting Serial Port 0 or 1 Mode will route the appropriate I2S or eMMC/SD interrupt to the CPU.

The interrupt service routine (ISR) for the SDIO0 interrupt can determine the event that caused the interrupt by checking the bits in SDST0. When SDST0 is read (either by CPU or emulation), all of the register bits are automatically cleared.

Table 6-4. Description of eMMC/SD Interrupt Requests

Interrupt Request	Interrupt Event
TRNDNEINT	**For read operations:** The eMMC/SD controller has received the last byte of data (before CRC check).
	For write operations: The eMMC/SD controller has transferred the last word of data to the output shift register.
DATEDINT	An edge was detected on the DAT3 pin.
DRRDYINT	SDDRR is ready to be read (data in FIFO is above threshold).
DXRDYINT	SDDXR is ready to transmit new data (data in FIFO is less than threshold).
CRCRSINT	A CRC error was detected in a response from the memory card.
CRCRDINT	A CRC error was detected in the data read from the memory card.
CRCWRINT	A CRC error was detected in the data written to the memory card.
TOUTRSINT	A time-out occurred while the eMMC controller was waiting for a response to a command.
TOUTRDINT	A time-out occurred while the eMMC controller was waiting for the data from the memory card.
RSPDNEINT	**For a command that requires a response:** The eMMC controller has received the response without a CRC error.
	For a command that does not require a response: The eMMC controller has finished sending the command.
BSYDNEINT	The memory card stops or is no longer sending a busy signal when the eMMC controller is expecting a busy signal.
DATDNEINT	**For read operations:** The eMMC controller has received data without a CRC error.
	For write operations: The eMMC controller has finished sending data.

6.2.12 DMA Event Support

The eMMC/SD controller is capable of generating DMA events for both read and write operations in order to request service from a DMA controller. Based on the FIFO threshold setting, the DMA event signals would be generated every time 128-bit or 256-bit data is transferred from the FIFO.

6.2.13 Emulation Considerations

The eMMC/SD controller is not affected by emulation halt events (such as breakpoints).

6.3 Procedures for Common Operations

6.3.1 Card Identification Operation

Before the eMMC/SD controller starts data transfers to or from memory cards in the eMMC/SD native mode, it has to first identify how many cards are present on the bus and configure them. For each card that responds to the ALL_SEND_CID broadcast command, the controller reads that card's unique card identification address (CID) and then assigns it a relative address (RCA). This address is much shorter than the CID and is used by the eMMC/SD controller to identify the card in all future commands that involve the card.

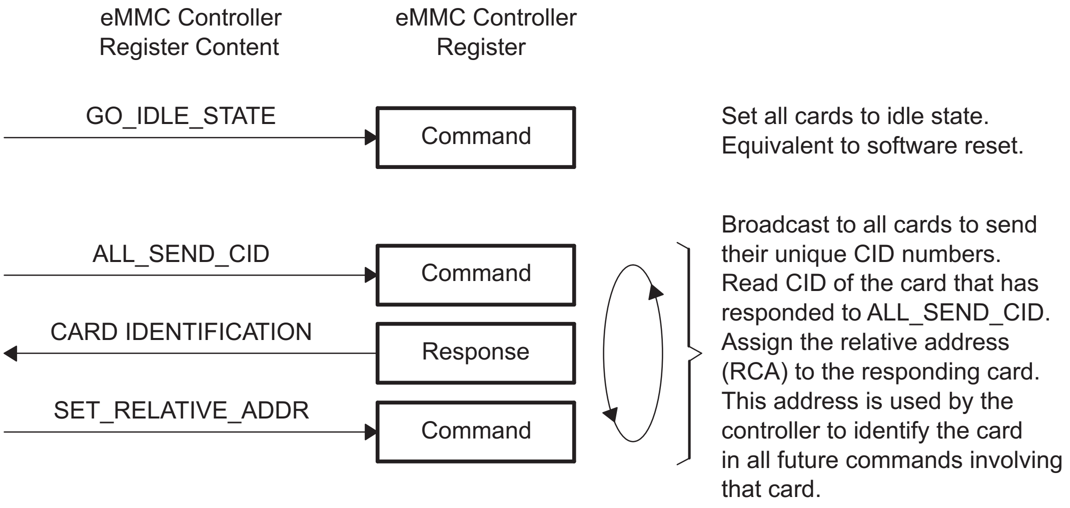
Only one card completes the response to ALL_SEND_CID at any one time. The absence of any response to ALL_SEND_CID indicates that all cards have been identified and configured.

The procedure for a card identification operation is:

1. Use eMMCSD1 to send the GO_IDLE_STATE command to the cards. This puts all cards in the idle state. The SEND_IF_COND command should be used next to check for SD card version, followed by the SD_SEND_OP_COND command for host capacity support operating condition information exchange to see if card is standard or high capacity (if card has been identified as ver2.0).
2. Use eMMCSD1 to send the ALL_SEND_CID command to the cards. This notifies all cards to identify themselves.
3. Wait for a card to respond. If a card responds, go to step 4; otherwise, stop.
4. Read the CID from the eMMC response registers (SDRSP0–7) and assign a relative address to the card by sending the SET_RELATIVE_ADDR command.

The sequence of events in this operation is shown in Figure 6-12.

Figure 6-12. Card Identification (Native eMMC/SD Mode)


eMMC Controller Register Content
eMMC Controller Register
GO_IDLE_STATE
Command
Set all cards to idle state. Equivalent to software reset.
ALL_SEND_CID
Command
CARD IDENTIFICATION
Response
SET_RELATIVE_ADDR
Command
Broadcast to all cards to send their unique CID numbers. Read CID of the card that has responded to ALL_SEND_CID. Assign the relative address (RCA) to the responding card. This address is used by the controller to identify the card in all future commands involving that card.

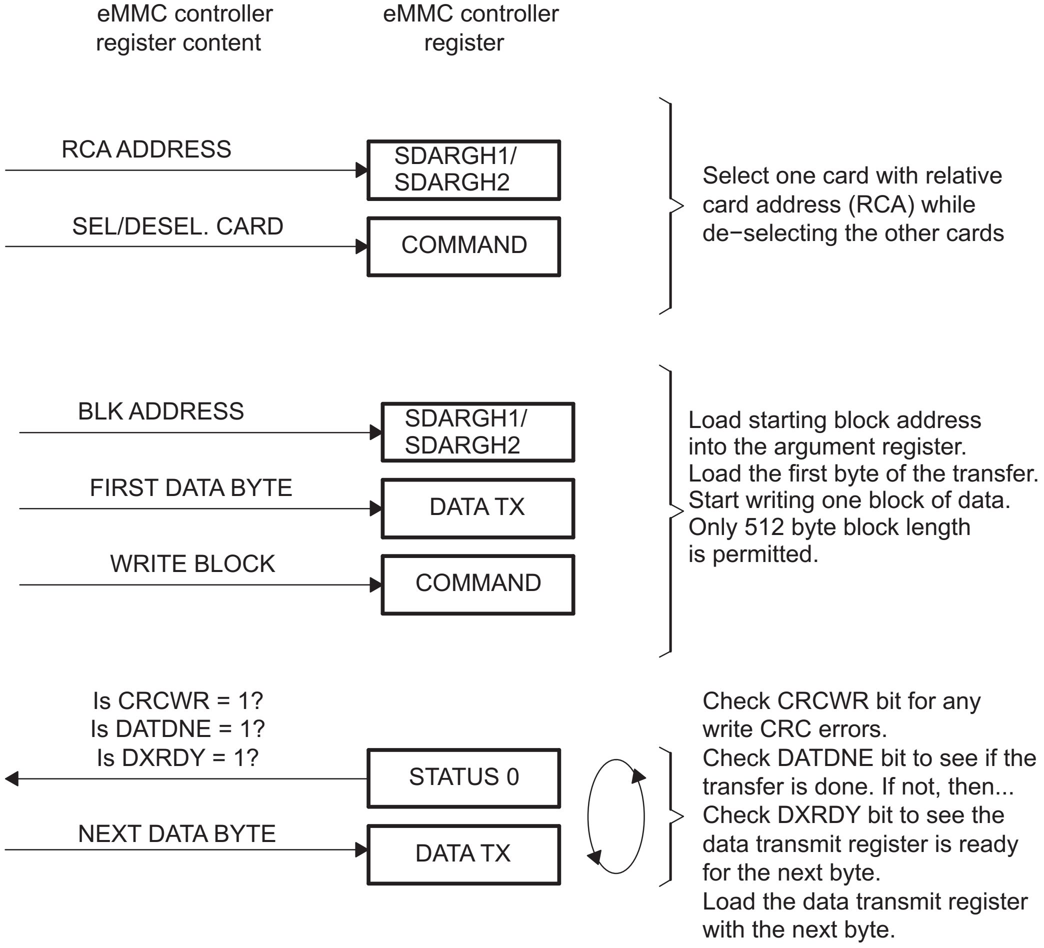
6.3.2 eMMC/SD Mode Single-Block Write Operation Using CPU

To perform a single-block write, the block length must be 512 bytes and the same length needs to be set in both the eMMC/SD controller and the memory card. The procedure for this operation is:

1. Write the card's relative address to the SD argument registers (SDARG1/SDARG2).
2. Use the eMMCSD1 to send the SELECT/DESELECT_CARD broadcast command. This selects the addressed card and deselects the others.
3. Write the destination start address to the SD argument registers.
4. Read card CSD to determine the card's maximum block length.
5. Use eMMCSD1 to send the SET_BLOCKLEN command (if the block length is different than the length used in the previous operation). The block length must be a multiple of 512 bytes and less then the maximum block length specified in the CSD.
6. Reset the FIFO by setting the FIFORST bit in SDFIFOCTL.
7. Bring the FIFO out of reset by clearing the FIFORST bit and set the FIFO direction to transmit (FIFODIR bit in SDFIFOCTL).
8. Set the access width (ACCWD bits in SDFIFOCTL).
9. Enable the eMMC interrupt.
10. Enable DXRDYINT interrupt.
11. Write the first 32 bits of the data block to the data transmit register (SDDXR).
12. Use eMMCSD1 to send the WRITE_BLOCK command to the card.
13. Wait for the eMMC interrupt
14. Use the eMMC status register 0 (SDST0) to check for errors and the status of the FIFO. If all of the data has not been written and if the FIFO is not full, go to step 15. If all of the data has been written, stop.
15. Write the next *n* bytes (depends on setting of FIFOLEV in SDFIFOCTL:0 = 16 bytes , 1 = 32 bytes) of the data block to the SD data transmit register (SDDXR) and go to step 13.

The sequence of events in this operation is shown in Figure 6-13.

Figure 6-13. eMMC/SD Mode Single-Block Write Operation



eMMC controller register content
eMMC controller register
RCA ADDRESS
SDARGH1/ SDARGH2
SEL/DESEL. CARD
COMMAND
Select one card with relative card address (RCA) while de−selecting the other cards
BLK ADDRESS
SDARGH1/ SDARGH2
FIRST DATA BYTE
DATA TX
WRITE BLOCK
COMMAND
Load starting block address into the argument register.
Load the first byte of the transfer.
Start writing one block of data.
Only 512 byte block length is permitted.
Is CRCWR = 1?
Is DATDNE = 1?
Is DXRDY = 1?
STATUS 0
NEXT DATA BYTE
DATA TX
Check CRCWR bit for any write CRC errors.
Check DATDNE bit to see if the transfer is done. If not, then...
Check DXRDY bit to see the data transmit register is ready for the next byte.
Load the data transmit register with the next byte.

6.3.3 eMMC/SD Mode Single-Block Write Operation Using DMA

To perform a single-block write, the block length must be 512 bytes and the same length needs to be set in both the eMMC/SD controller and the card. The procedure for this operation is:

1. Write the card’s relative address to the eMMC argument registers (SDARG1/SDARG2).
2. Read card CSD to determine the card's maximum block length.
3. Use eMMCSD1 to send the SET_BLOCKLEN command (if the block length is different than the length used in the previous operation). The block length must be a multiple of 512 bytes and less then the maximum block length specified in the CSD.
4. Reset the FIFO (FIFORST bit in SDFIFOCTL).
5. Set the FIFO direction to transmit (FIFODIR bit in SDFIFOCTL).
6. Set the access width (ACCWD bits in SDFIFOCTL).
7. Set the FIFO threshold (FIFOLEV bit in SDFIFOCTL).
8. Set up DMA (DMA size needs to be greater than or equal to FIFOLEV setting).
9. Use eMMCSD1 to send the WRITE _BLOCK command to the card.
10. Use eMMCSD2 to trigger first DMA transfer to FIFO by setting the DMATRIG bit.
11. Wait for DMA sequence to complete or the DATADNE flag in the eMMC status register 0 (SDST0) is set.
12. Use SDST0 to check for errors.

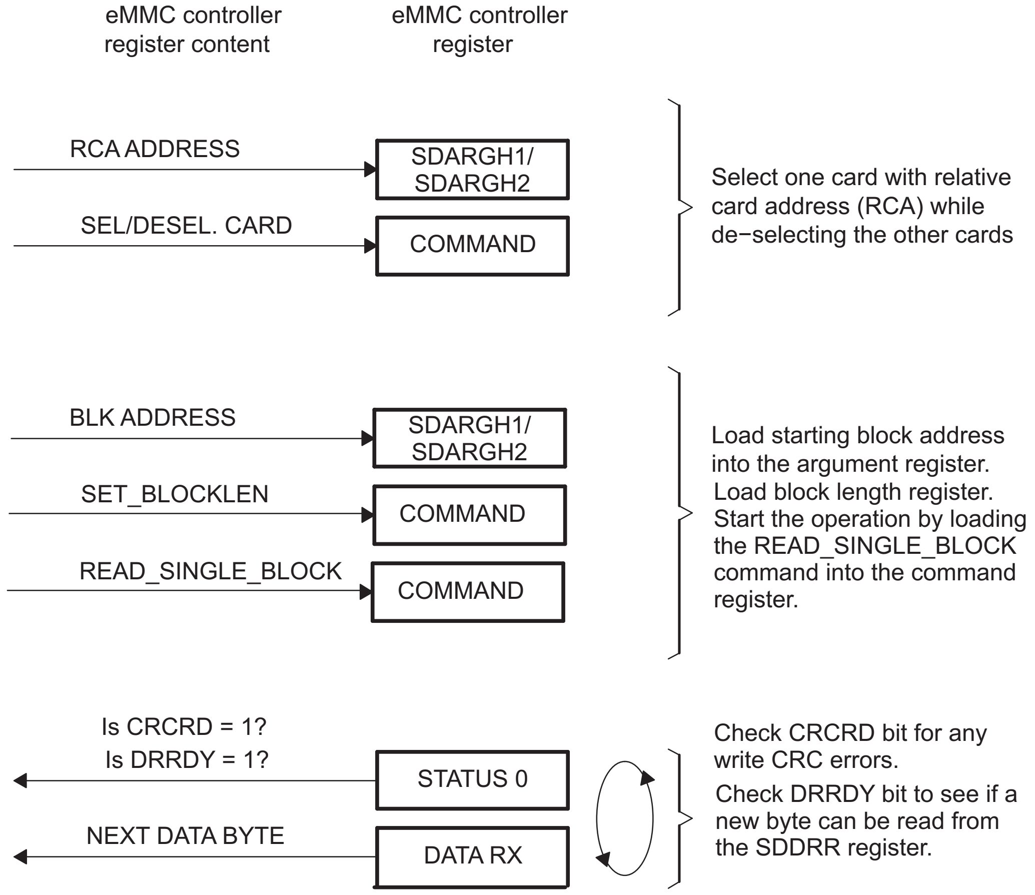
6.3.4 eMMC/SD Mode Single-Block Read Operation Using CPU

To perform a single-block read, the same block length needs to be set in both the eMMC/SD controller and the card. The procedure for this operation is:

1. Write the card’s relative address to the SD argument registers (SDARG1/SDARG2).
2. Use eMMCSD1 to send the SELECT/DESELECT_CARD broadcast command. This selects the addressed card and deselects the others.
3. Write the source start address to the SD argument registers.
4. Read card CSD to determine the card's maximum block length.
5. Use eMMCSD1 to send the SET_BLOCKLEN command (if the block length is different than the length used in the previous operation). The block length must be a multiple of 512 bytes and less then the maximum block length specified in the CSD.
6. Reset the FIFO by setting the FIFORST bit in SDFIFOCTL.
7. Bring the FIFO out of reset by clearing the FIFORST bit and set the FIFO direction to receive (FIFODIR bit in SDFIFOCTL).
8. Set the access width (ACCWD bits in SDFIFOCTL).
9. Set the FIFO threshold (FIFOLEV bit in SDFIFOCTL).
10. Enable the SD interrupt.
11. Enable DRRDYINT interrupt.
12. Use eMMCSD1 to send the READ_SINGLE_BLOCK command.
13. Wait for SD interrupt.
14. Use the SD status register 0 (SDST0) to check for errors and the status of the FIFO. If the FIFO is not empty, go to step 14. If the all of the data has been read, stop.
15. Read the next *n* bytes of data (depends on setting of FIFOLEV in SDFIFOCTL: 0 = 16 bytes, 1 = 32 bytes) from the SD data receive register (SDDRR) and go to step 13.

The sequence of events in this operation is shown in Figure 6-14.

Figure 6-14. eMMC/SD Mode Single-Block Read Operation


eMMC controller register content
eMMC controller register
RCA ADDRESS
SDARGH1/ SDARGH2
SEL/DESEL. CARD
COMMAND
Select one card with relative card address (RCA) while de−selecting the other cards
BLK ADDRESS
SDARGH1/ SDARGH2
SET_BLOCKLEN
COMMAND
READ_SINGLE_BLOCK
COMMAND
Load starting block address into the argument register.
Load block length register.
Start the operation by loading the READ_SINGLE_BLOCK command into the command register.
Is CRCRD = 1?
Is DRRDY = 1?
STATUS 0
NEXT DATA BYTE
DATA RX
Check CRCRD bit for any write CRC errors.
Check DRRDY bit to see if a new byte can be read from the SDDRR register.

6.3.5 *eMMC/SD Mode Single-Block Read Operation Using DMA*

To perform a single-block read, the same block length needs to be set in both the eMMC/SD controller and the card. The procedure for this operation is:

1. Write the card’s relative address to the SD argument registers (SDARG1/SDARG2).
2. Read card CSD to determine the card's maximum block length.
3. Use the eMMCSD1 to send the SET_BLOCKLEN command (if the block length is different than the length used in the previous operation). The block length must be a multiple of 512 bytes and less then the maximum block length specified in the CSD.
4. Reset the FIFO by setting the FIFORST bit in SDFIFOCTL.
5. Bring the FIFO out of reset by clearing the FIFORST bit and set the FIFO direction to receive (FIFODIR bit in SDFIFOCTL).
6. Set the access width (ACCWD bits in SDFIFOCTL).
7. Set the FIFO threshold (FIFOLEV bit in SDFIFOCTL).
8. Set up DMA (DMA size needs to be greater than or equal to FIFOLEV setting).
9. Use eMMCSD1 to send the READ _BLOCK command to the card.
10. Wait for DMA sequence to complete.
11. Use the SD status register 0 (SDST0) to check for errors.

6.3.6 eMMC/SD Mode Multiple-Block Write Operation Using CPU

To perform a multiple-block write, the same block length needs to be set in both the eMMC/SD controller and the card.

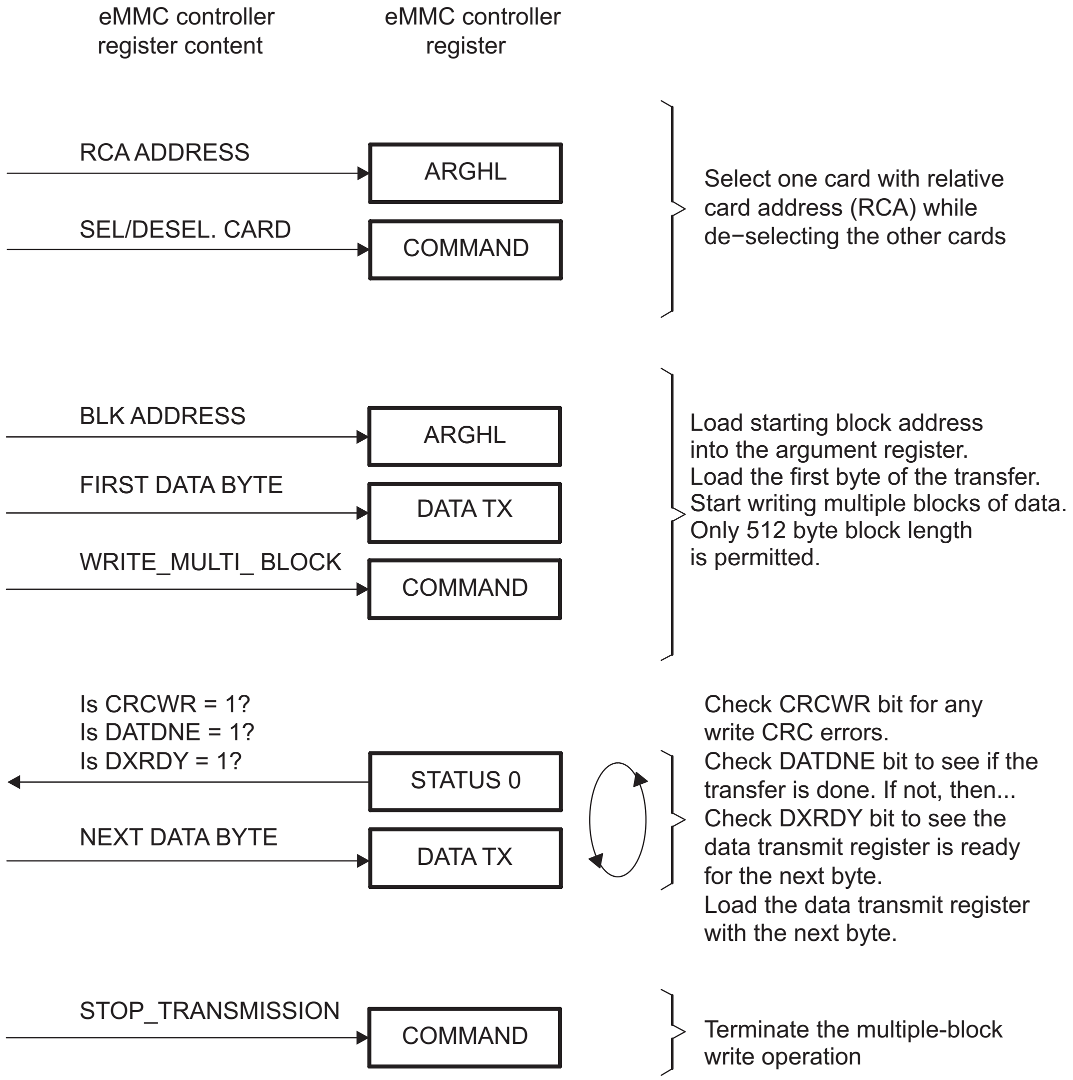
NOTE: The procedure in this section uses a STOP_TRANSMISSION command to end the block transfer. This assumes that the value in the SD number of blocks counter register (SDNBLK) is 0. A multiple-block operation terminates itself if you load SDNBLK with the exact number of blocks you want transferred.

The procedure for this operation is:

1. Write the card’s relative address to the SD argument registers (SDARG1/SDARG2).
2. Read card CSD to determine the card's maximum block length.
3. Use eMMCSD1 to send the SET_BLOCKLEN command (if the block length is different than the length used in the previous operation). The block length must be a multiple of 512 bytes and less then the maximum block length specified in the CSD.
4. Reset the FIFO by setting the FIFORST bit in SDFIFOCTL.
5. Bring the FIFO out of reset by clearing the FIFORST bit and set the FIFO direction to transmit (FIFODIR bit in SDFIFOCTL).
6. Set the access width (ACCWD bits in SDFIFOCTL).
7. Set the FIFO threshold (FIFOLEV bit in SDFIFOCTL).
8. Enable the SD interrupt.
9. Enable DXRDYINT interrupt.
10. Write the first 32 bits of the data block to the eMMC data transmit register (SDDXR).
11. Use eMMCSD1 to send the WRITE_MULTI_BLOCK command to the card.
12. Wait for SD interrupt.
13. Use the eMMC status register 1 (SDST1) to check for errors and to determine the status of the FIFO. If more bytes are to be written and the FIFO is not full, go to step 14. If the all of the data has been written, go to step 15.
14. Write the next *n* bytes (depends on setting of FIFOLEV in SDFIFOCTL:0 = 16 bytes , 1 = 32 bytes) of the data block to SDDXR, and go to step 12.
15. Use eMMCSD1 to send the STOP_TRANSMISSION command.

The sequence of events in this operation is shown in Figure 6-15.

Figure 6-15. eMMC/SD Multiple-Block Write Operation


eMMC controller register content
eMMC controller register
RCA ADDRESS
ARGHL
SEL/DESEL. CARD
COMMAND
Select one card with relative card address (RCA) while de−selecting the other cards
BLK ADDRESS
ARGHL
FIRST DATA BYTE
DATA TX
WRITE_MULTI_ BLOCK
COMMAND
Load starting block address into the argument register.
Load the first byte of the transfer.
Start writing multiple blocks of data.
Only 512 byte block length is permitted.
Is CRCWR = 1?
Is DATDNE = 1?
Is DXRDY = 1?
STATUS 0
NEXT DATA BYTE
DATA TX
Check CRCWR bit for any write CRC errors.
Check DATDNE bit to see if the transfer is done. If not, then...
Check DXRDY bit to see the data transmit register is ready for the next byte.
Load the data transmit register with the next byte.
STOP_TRANSMISSION
COMMAND
Terminate the multiple-block write operation

6.3.7 eMMC/SD Mode Multiple-Block Write Operation Using DMA

To perform a multiple-block write, the same block length needs to be set in both the eMMC/SD controller and the card. The procedure for this operation is:

1. Write the card’s relative address to the SD argument registers (SDARG1/SDARG2).
2. Read card CSD to determine the card's maximum block length.
3. Use eMMCSD1 to send the SET_BLOCKLEN command (if the block length is different than the length used in the previous operation). The block length must be a multiple of 512 bytes and less then the maximum block length specified in the CSD.
4. Reset the FIFO by setting the FIFORST bit in SDFIFOCTL.
5. Bring the FIFO out of reset by clearing the FIFORST bit and set the FIFO direction to transmit (FIFODIR bit in SDFIFOCTL).
6. Set the FIFO threshold (FIFOLEV bit in SDFIFOCTL).
7. Set the access width (ACCWD bits in SDFIFOCTL).
8. Set up DMA (DMA size needs to be greater than or equal to FIFOLEV setting).
9. Use eMMCSD1 to send the WRITE_MULTI_BLOCK command to the card.

10. Use eMMCSD2 to trigger first DMA transfer to FIFO by setting the DMATRIG bit.
11. Wait for DMA sequence to complete or the DATADNE flag in the eMMC status register 0 (SDST0) is set.
12. Use SDST0 to check for errors.
13. Use eMMCSD1 to send the STOP_TRANSMISSION command.

6.3.8 eMMC/SD Mode Multiple-Block Read Operation Using CPU

To perform a multiple-block read, the same block length needs to be set in both the eMMC/SD controller and the card.

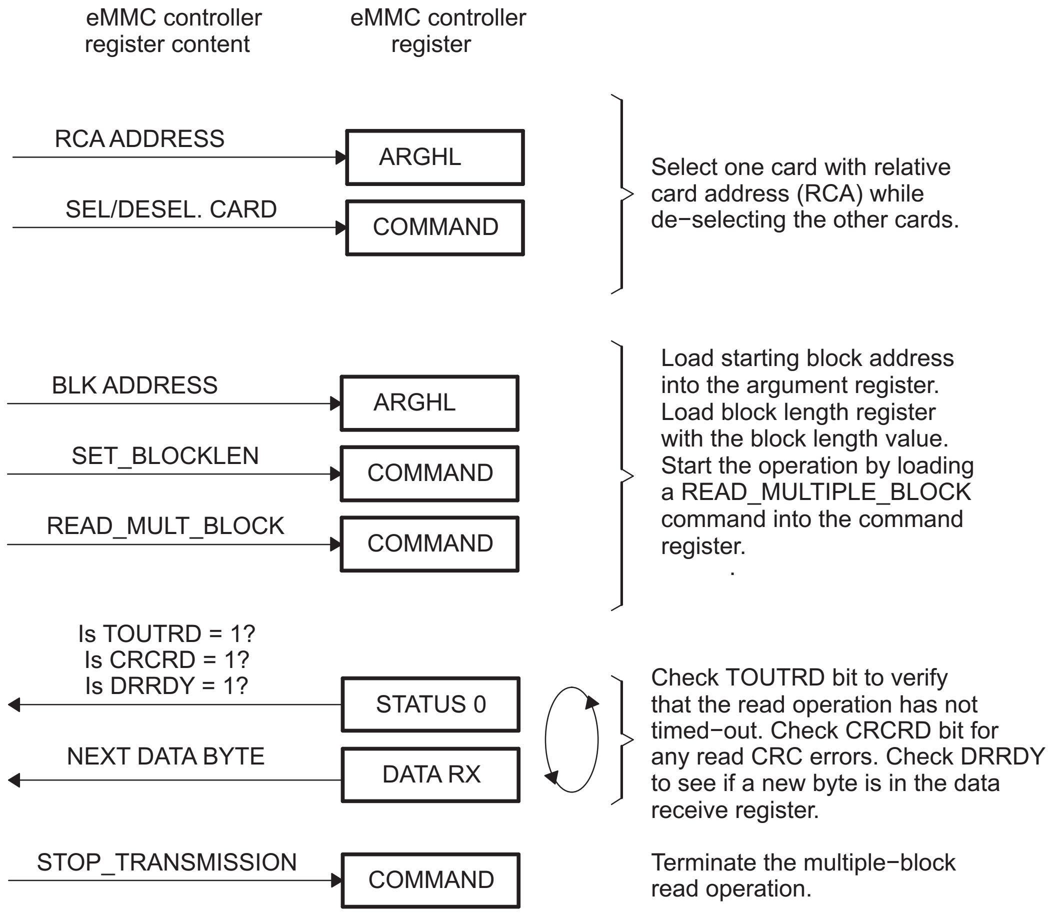
NOTE: The procedure in this section uses a STOP_TRANSMISSION command to end the block transfer. This assumes that the value in the SD number of blocks counter register (SDNBLK) is 0. A multiple-block operation terminates itself if you load SDNBLK with the exact number of blocks you want transferred.

The procedure for this operation is:

1. Write the card’s relative address to the SD argument registers (SDARG1/SDARG2).
2. Read card CSD to determine the card's maximum block length.
3. Use eMMCSD1 to send the SET_BLOCKLEN command (if the block length is different than the length used in the previous operation). The block length must be a multiple of 512 bytes and less then the maximum block length specified in the CSD.
4. Reset the FIFO by setting the FIFORST bit in SDFIFOCTL.
5. Bring the FIFO out of reset by clearing the FIFORST bit and set the FIFO direction to receive (FIFODIR bit in SDFIFOCTL).
6. Set FIFO threshold (FIFOLEV bit in SDFIFOCTL).
7. Set the access width (ACCWD bits in SDFIFOCTL).
8. Enable the SD interrupt.
9. Enable DRRDYINT interrupt.
10. Use eMMCSD1 to send the READ_MULT_BLOCKS command.
11. Wait for SD interrupt.
12. Use the SD status register 1 (SDST1) to check for errors and to determine the status of the FIFO. If FIFO is not empty and more bytes are to be read, go to step 13. If the all of the data has been read, go to step 14.
13. Read *n* bytes (depends on setting of FIFOLEV in SDFIFOCTL:0 = 16 bytes , 1 = 32 bytes) of data from the SD data receive register (SDDRR) and go to step 10.
14. Use eMMCSD1 to send the STOP_TRANSMISSION command.

The sequence of events in this operation is shown in Figure 6-16.

Figure 6-16. eMMC/SD Mode Multiple-Block Read Operation


eMMC controller register content
eMMC controller register
RCA ADDRESS
ARGHL
SEL/DESEL. CARD
COMMAND
Select one card with relative card address (RCA) while de−selecting the other cards.
BLK ADDRESS
ARGHL
SET_BLOCKLEN
COMMAND
READ_MULT_BLOCK
COMMAND
Load starting block address into the argument register. Load block length register with the block length value. Start the operation by loading a READ_MULTIPLE_BLOCK command into the command register.
Is TOUTRD = 1?
Is CRCRD = 1?
Is DRRDY = 1?
STATUS 0
NEXT DATA BYTE
DATA RX
Check TOUTRD bit to verify that the read operation has not timed−out. Check CRCRD bit for any read CRC errors. Check DRRDY to see if a new byte is in the data receive register.
STOP_TRANSMISSION
COMMAND
Terminate the multiple−block read operation.

6.3.9 eMMC/SD Mode Multiple-Block Read Operation Using DMA

To perform a multiple-block read, the same block length needs to be set in both the eMMC/SD controller and the card. The procedure for this operation is:

1. Write the card’s relative address to the SD argument registers (SDARG1/SDARG2).
2. Read card CSD to determine the card's maximum block length.
3. Use eMMCSD1 to send the SET_BLOCKLEN command (if the block length is different than the length used in the previous operation). The block length must be a multiple of 512 bytes and less then the maximum block length specified in the CSD.
4. Reset the FIFO by setting the FIFORST bit in SDFIFOCTL.
5. Bring the FIFO out of reset by clearing the FIFORST bit and set the FIFO direction to receive (FIFODIR bit in SDFIFOCTL).
6. Set the FIFO threshold (FIFOLEV bit in SDFIFOCTL).
7. Set the access width (ACCWD bits in SDFIFOCTL).
8. Set up DMA (DMA size needs to be greater than or equal to FIFOLEV setting).
9. Use eMMCSD1 to send the READ_MULTI_BLOCK command to the card.
10. Wait for DMA sequence to complete.
11. Use the SD status register 0 (SDST0) to check for errors.
12. Use eMMCSD1 to send the STOP_TRANSMISSION command.

6.3.10 SD High Speed Mode

To perform the high-speed mode operation the card need to be placed in high-speed mode. The procedure for this operation is:

1. Follow the normal card identification procedure, since all the high-speed cards are by default initially normal SD cards. Once card is successfully identified, it needs to be switched into high-speed mode.
2. Send CMD16 (SET_BLOCK_LEN) with argument as 8 (8 bytes) to set the block length in the card.
3. Set the block length as 8 bytes and number of blocks as 1 in SD controller registers SDBLEN and MMNBLK, respectively.
4. Read the SCR register by sending ACMD51.
5. Parse the 64-bit response received from the card to check whether the card has support of SD spec Ver1.10. The high-speed support is available only cards those are supporting SD VER 1.10.
6. Send CMD16 (SET_BLOCK_LEN) with argument as 64 (64 bytes) to set the block length in the card.
7. Set the block length as 64 bytes and number of blocks as 1 in SD controller registers SDBLEN and MMNBLK, respectively.
8. Send CMD6 with Mode 0.
9. Parse the 512-bit response received from the card to check whether the card has high-speed function support. If yes, check that the Maximum current consumption for this function is within the limit which is specified in CSD register (CSD register bits 61:50, response for the CMD9, SEND_CSD).
10. Send CMD6 with Mode 1 to enable the high-speed function.
11. Parse the 512-bit response received from the card to check whether the card has successfully been placed in high-speed mode.
12. Increase the SD clock rate up to 50 MHz.
13. Follow normal read/write operation.

6.3.11 SDIO Card Function

To support the SDIO card, the following features are available in the eMMC/SD controller:

- Read wait operation request.

When in 1-bit mode and the transfer clock (memory clock) is off, this peripheral cannot recognize an SDIO interrupt from SD_DATA1 line. Two options are available to deal with this situation:

1. Do not turn off the memory clock in 1-bit mode. The clock is enabled by the CLKEN bit in the SD memory clock control register (SDCLK).
2. If the memory clock needs to be turned off, physically connect a GPIO signal and SD_DATA1, and use the GPIO as an external interrupt input. When the memory clock is enabled, disable the GPIO interrupt and enable the SDIO interrupt. When the memory clock is disabled, enable the GPIO interrupt and disable the SDIO interrupt by software.

6.3.11.1 SDIO Control Register (SDIOCTL)

The SDIO card control register (SDIOCTL) is used to configure the read wait operation using the SD_DATA2 line.

6.3.11.2 SDIO Status Register 0 (SDIOST0)

The SDIO card status register 0 (SDIOST0) is used to check the status of the SD_DATA1 signal, check the status of being in an interrupt period, or check the status of being in a read wait operation.

6.3.11.3 SDIO Interrupt Control Registers (SDIOIEN, SDIOIST)

The SDIO card controller issues an interrupt to the CPU when the read wait operation starts or when an SDIO interrupt is detected on the SD_DATA1 line.

Interrupt flags of each case are checked with the SDIO interrupt status register (SDIOIST). To issue an actual interrupt to CPU, enabling each interrupt in the SDIO interrupt enable register (SDIOIEN) is required.

When both interrupts are enabled, they are both reported to the CPU as a single interrupt (whether one or both occurred). The interrupt(s) that occurred are determined by reading SDIOIST.

6.4 Registers

Table 6-5 list the memory-mapped registers associated with the two Multimedia Card/Secure Digital 0 (MMC/SD0) controllers and Table 6-6 list the memory-mapped registers associated with the two Multimedia Card/Secure Digital 1 (MMC/SD1) controllers. Note that the CPU accesses all peripheral registers through its I/O space. All other register addresses not listed should be considered as reserved locations and the register contents should not be modified.

Table 6-5. Embedded Multimedia Card/Secure Digital 0 (eMMC/SD0) Card Controller Registers

CPU Word Address	Acronym	Register Description	Section
3A00h	SDCTL	SD Control Register	Section 6.4.1
3A04h	SDCLK	SD Memory Clock Control Register	Section 6.4.2
3A08h	SDST0	SD Status Register 0	Section 6.4.3
3A0Ch	SDST1	SD Status Register 1	Section 6.4.4
3A10h	SDIM	SD Interrupt Mask Register	Section 6.4.5
3A14h	SDTOR	SD Response Time-Out Register	Section 6.4.6
3A18h	SDTOD	SD Data Read Time-Out Register	Section 6.4.7
3A1Ch	SDBLEN	SD Block Length Register	Section 6.4.8
3A20h	SDNBLK	SD Number of Blocks Register	Section 6.4.9
3A24h	SDNBLC	SD Number of Blocks Counter Register	Section 6.4.10
3A28h	SDDRR1	SD Data Receive Register 1	Section 6.4.11
3A29h	SDDRR2	SD Data Receive Register 2	Section 6.4.11
3A2Ch	SDDXR1	SD Data Transmit Register 1	Section 6.4.12
3A2Dh	SDDXR2	SD Data Transmit Register 2	Section 6.4.12
3A30h	MMCSD1	MMC Command Register 1	Section 6.4.13
3A31h	MMCSD2	MMC Command Register 2	Section 6.4.13
3A34h	SDARG1	SD Argument Register 1	Section 6.4.14
3A35h	SDARG2	SD Argument Register 2	Section 6.4.14
3A38h	SDRSP0	SD Response Register 0	Section 6.4.15
3A39h	SDRSP1	SD Response Register 1	Section 6.4.15
3A3Ch	SDRSP2	SD Response Register 2	Section 6.4.15
3A3Dh	SDRSP3	SD Response Register 3	Section 6.4.15
3A40h	SDRSP4	SD Response Register 4	Section 6.4.15
3A41h	SDRSP5	SD Response Register 5	Section 6.4.15
3A44h	SDRSP6	SD Response Register 6	Section 6.4.15
3A45h	SDRSP7	SD Response Register 7	Section 6.4.15
3A48h	SDDRSP	SD Data Response Register	Section 6.4.16
3A50h	SDCIDX	SD Command Index Register	Section 6.4.17
3A64h	SDIOCTL	SDIO Control Register	Section 6.4.18
3A68h	SDIOST0	SDIO Status Register 0	Section 6.4.19
3A6Ch	SDIOIEN	SDIO Interrupt Enable Register	Section 6.4.20
3A70h	SDIOIST	SDIO Interrupt Status Register	Section 6.4.21
3A74h	SDFIFOCTL	SD FIFO Control Register	Section 6.4.22

Table 6-6. Embedded Multimedia Card/Secure Digital 1 (eMMC/SD1) Card Controller Registers

CPU Word Address	Acronym	Register Description	Section
3B00h	SDCTL	SD Control Register	Section 6.4.1
3B04h	SDCLK	SD Memory Clock Control Register	Section 6.4.2
3B08h	SDST0	SD Status Register 0	Section 6.4.3
3B0Ch	SDST1	SD Status Register 1	Section 6.4.4
3B10h	SDIM	SD Interrupt Mask Register	Section 6.4.5
3B14h	SDTOR	SD Response Time-Out Register	Section 6.4.6
3B18h	SDTOD	SD Data Read Time-Out Register	Section 6.4.7
3B1Ch	SDBLEN	SD Block Length Register	Section 6.4.8
3B20h	SDNBLK	SD Number of Blocks Register	Section 6.4.9
3B24h	SDNBLC	SD Number of Blocks Counter Register	Section 6.4.10
3B28h	SDDRR1	SD Data Receive Register 1	Section 6.4.11
3B29h	SDDRR2	SD Data Receive Register 2	Section 6.4.11
3B2Ch	SDDXR1	SD Data Transmit Register 1	Section 6.4.12
3B2Dh	SDDXR2	SD Data Transmit Register 2	Section 6.4.12
3B30h	MMCSD1	eMMC Command Register 1	Section 6.4.13
3B31h	MMCSD2	eMMC Command Register 2	Section 6.4.13
3B34h	SDARG1	SD Argument Register 1	Section 6.4.14
3B35h	SDARG2	SD Argument Register 2	Section 6.4.14
3B38h	SDRSP0	SD Response Register 0	Section 6.4.15
3B39h	SDRSP1	SD Response Register 1	Section 6.4.15
3B3Ch	SDRSP2	SD Response Register 2	Section 6.4.15
3B3Dh	SDRSP3	SD Response Register 3	Section 6.4.15
3B40h	SDRSP4	SD Response Register 4	Section 6.4.15
3B41h	SDRSP5	SD Response Register 5	Section 6.4.15
3B44h	SDRSP6	SD Response Register 6	Section 6.4.15
3B45h	SDRSP7	SD Response Register 7	Section 6.4.15
3B48h	SDDRSP	SD Data Response Register	Section 6.4.16
3B50h	SDCIDX	SD Command Index Register	Section 6.4.17
3B64h	SDIOCTL	SDIO Control Register	Section 6.4.18
3B68h	SDIOST0	SDIO Status Register 0	Section 6.4.19
3B6Ch	SDIOIEN	SDIO Interrupt Enable Register	Section 6.4.20
3B70h	SDIOIST	SDIO Interrupt Status Register	Section 6.4.21
3B74h	SDFIFOCTL	SD FIFO Control Register	Section 6.4.22

6.4.1 SD Control Register (SDCTL)

The SD control register (SDCTL) is used to enable or configure various modes of the SD controller. Set or clear the DATRST and CMDRST bits at the same time to reset or enable the SD controller. SDCTL is shown in Figure 6-17 and described in Table 6-7.

Figure 6-17. SD Control Register (SDCTL)

15	11	10	9	8
Reserved		PERMDX	PERMDR	Reserved
R-0		R/W-0	R/W-0	R-0

7	6	5	3	2	1	0
DATEG		Reserved		WIDTH	CMDRST	DATRST
R/W-0		R-0		R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-7. SD Control Register (SDCTL) Field Descriptions

Bit	Field	Value	Description
15-11	Reserved	0	Reserved.
10	PERMDX		Endian select enable when writing.
		0	Little Endian is selected.
		1	Big Endian is selected.
9	PERMDR		Endian select enable when reading.
		0	Little Endian is selected.
		1	Big Endian is selected.
8	Reserved	0	Reserved.
7-6	DATEG	0-3h	DAT3 edge detection select.
		0	DAT3 edge detection is disabled.
		1h	DAT3 rising-edge detection is enabled.
		2h	DAT3 falling-edge detection is enabled.
		3h	DAT3 rising-edge and falling-edge detections are enabled.
5-3	Reserved	0	Reserved.
2	WIDTH		Data bus width (MMC mode only).
		0	Data bus has 1 bit (only DAT0 is used).
		1	Data bus has 4 bits (all DAT0-3 are used).
1	CMDRST		CMD logic reset.
		0	CMD line portion is enabled.
		1	CMD line portion is disabled and in reset state.
0	DATRST		DAT logic reset.
		0	DAT line portion is enabled.
		1	DAT line portion is disabled and in reset state.

6.4.2 SD Memory Clock Control Register (SDCLK)

The SD memory clock control register (SDCLK) is used to:

- Select whether the CLK pin is enabled or disabled (CLKEN bit).
- Select how much the function clock is divided-down to produce the memory clock (CLKRT bits). When the CLK pin is enabled, the SD controller drives the memory clock on this pin to control the timing of communications with attached memory cards. For more details about clock generation, see Section 6.2.1.

SDCLK is shown in Figure 6-18 and described in Table 6-8.

Figure 6-18. SD Memory Clock Control Register (SDCLK)

15 9	8	7 0
Reserved	CLKEN	CLKRT
R-0	R/W-0	R/W-FFh

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-8. SD Memory Clock Control Register (SDCLK) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved.
8	CLKEN		CLK pin enable.
		0	CLK pin is disabled and fixed low.
		1	The CLK pin is enabled; it shows the memory clock signal.
7-0	CLKRT	0-FFh	Clock rate. Use this field to set the divide-down value for the memory clock. The function clock is divided down as follows to produce the memory clock: *memory clock frequency = function clock frequency/(2 * (CLKRT + 1)*

6.4.3 SD Status Register 0 (SDST0)

The SD status register 0 (SDST0) records specific events or errors. The transition from 0 to 1 on each bit in SDST0 can cause an interrupt signal to be sent to the CPU. If an interrupt is desired, set the corresponding interrupt enable bit in the SD interrupt mask register (SDIM).

When a status bit is read (by CPU or emulation) it is cleared. Additionally DRRDY bit and the DXRDY bit are also cleared in response to the functional events described for them in Table 6-9, or in response to a hardware reset.

SDST0 is shown in Figure 6-19 and described in Table 6-9.

Figure 6-19. SD Status Register 0 (SDST0)

15		13	12	11	10	9	8
Reserved			TRNDNE	DATED	DRRDY	DXRDY	Reserved
R-0			RC-0	RC-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
CRCRS	CRCRD	CRCWR	TOUTRS	TOUTRD	RSPDNE	BSYDNE	DATDNE
RC-0	RC-0	RC-0	RC-0	RC-0	RC-0	RC-0	RC-0

LEGEND: R = Read only; RC = Cleared to 0 when read; -*n* = value after reset

Table 6-9. SD Status Register 0 (SDST0) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved. Any writes to these bit(s) must always have a value of 0.
12	TRNDNE		Transfer done.
		0	No data transfer is done.
		1	Data transfer of specified length is done.
11	DATED		DAT3 edge detected. DATED is cleared when read by CPU.
		0	A DAT3 edge has not been detected.
		1	A DAT3 edge has been detected.
10	DRRDY		Data receive ready. DRRDY is cleared to 0 when the DAT logic is reset (DATRST = 1 in SDCTL), when a command is sent with data receive/transmit clear (DCLR = 1 in MMCSD), or when data is read from the MMC data receive registers (SDDRR1 and SDDRR2).
		0	SDDRR is not ready.
		1	SDDRR is ready. New data has arrived and can be read by the CPU or by the DMA controller.
9	DXRDY		Data transmit ready. DXRDY is set to 1 when the DAT logic is reset (DATRST = 1 in SDCTL), when a command is sent with data receive/transmit clear (DCLR = 1 in MMCSD), or when data is written to the MMC data transmit register (MMCDXR).
		0	SDDXR is not ready.
		1	SDDXR is ready. The data in SDDXR has been transmitted; SDDXR can accept new data from the CPU or from the DMA controller.
8	Reserved	0	Reserved.
7	CRCRS		Response CRC error.
		0	A response CRC error has not been detected.
		1	A response CRC error has been detected.
6	CRCRD		Read-data CRC error.
		0	A read-data CRC error has not been detected.
		1	A read-data CRC error has been detected.
5	CRCWR		Write-data CRC error.
		0	A write-data CRC error has not been detected.
		1	A write-data CRC error has been detected.

Table 6-9. SD Status Register 0 (SDST0) Field Descriptions (continued)

Bit	Field	Value	Description
4	TOUTRS		Response time-out event.
		0	A response time-out event has not occurred.
		1	A time-out event has occurred while the MMC controller was waiting for a response to a command.
3	TOUTRD		Read-data time-out event.
		0	A read-data time-out event has not occurred.
		1	A time-out event has occurred while the MMC controller was waiting for data.
2	RSPDNE		Command/response done.
		0	No response is received.
		1	Command has been sent without response or response has been received for the command sent.
1	BSYDNE		Busy done.
		0	No busy releasing is done.
		1	Released from busy state or expected busy is not detected.
0	DATDNE		Data done.
		0	The data has not been fully transmitted.
		1	The data has been fully transmitted.

NOTE: 1) As the command portion and the data portion of the SD controller are independent, any command such as CMD0 (GO_IDLE_STATE) or CMD12 (STOP_TRANSMISSION) can be sent to the card, even if during block transfer. In this situation, the data portion will detect this and wait, releasing the busy state only when the command sent was R1b (to be specific, command with BSYEXP bit), otherwise it will keep transferring data.

2) Bit 12 (TRNDNE) indicates that the last byte of a transfer has been completed. Bit 0 (DATDNE) occurs at end of a transfer but not until the CRC check and programming has been completed.

6.4.4 SD Status Register 1 (SDST1)

The SD status register 1 (SDST1) records specific events or errors. There are no interrupts associated with these events or errors. SDST1 is shown in Figure 6-20 and described in Table 6-10.

Figure 6-20. SD Status Register 1 (SDST1)

15 – 8
Reserved
R-0

7	6	5	4	3	2	1	0
Reserved	FIFOFUL	FIFOEMP	DAT3ST	DRFUL	DXEMP	CLKSTP	BUSY
R-0	R-0	R-0	R-0	R-0	R-0	R-1	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 6-10. SD Status Register 1 (SDST1) Field Descriptions

Bit	Field	Value	Description
15-7	Reserved	0	Reserved.
6	FIFOFUL		FIFO is full.
		0	FIFO is not full.
		1	FIFO is full.
5	FIFOEMP		FIFO is empty.
		0	FIFO is not empty.
		1	FIFO is empty.
4	DAT3ST		DAT3 status.
		0	The signal level on the DAT3 pin is a logic-low level.
		1	The signal level on the DAT3 pin is a logic-high level.
3	DRFUL		Data receive register (SDDRR) is full.
		0	A data receive register full condition is not detected.
		1	A data receive register full condition is detected.
2	DXEMP		Data transmit register (MMCDXR) is empty.
		0	A data transmit register empty condition is not detected. The data transmit shift register is not empty.
		1	A data transmit register empty condition is detected. The data transmit shift register is empty. No bits are available to be shifted out to the memory card.
1	CLKSTP		Clock stop status.
		0	CLK is active. The memory clock signal is being driven on the pin.
		1	CLK is held low because of a manual stop (CLKEN = 0 in MMCCLK), receive shift register is full, or transmit shift register is empty.
0	BUSY		Busy.
		0	No busy signal is detected.
		1	A busy signal is detected (the memory card is busy).

6.4.5 SD Interrupt Mask Register (SDIM)

The SD interrupt mask register (SDIM) is used to enable (bit = 1) or disable (bit = 0) status interrupts. If an interrupt is enabled, the transition from 0 to 1 of the corresponding interrupt bit in the SD status register 0 (SDST0) can cause an interrupt signal to be sent to the CPU. SDIM is shown in Figure 6-21 and described in Table 6-11.

Figure 6-21. SD Interrupt Mask Register (SDIM)

15	13	12	11	10	9	8
Reserved		ETRNDNE	EDATED	EDRRDY	EDXRDY	Reserved
R-0		R/W-0	R/W-0	R/W-0	R/W-0	R-0

7	6	5	4	3	2	1	0
ECRCRS	ECRCRD	ECRCWR	ETOUTRS	ETOUTRD	ERSPDNE	EBSYDNE	EDATDNE
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-11. SD Interrupt Mask Register (SDIM) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12	ETRNDNE		Transfer done (TRNDNE) interrupt enable.
		0	Transfer done interrupt is disabled.
		1	Transfer done interrupt is enabled.
11	EDATED		DAT3 edge detect (DATED) interrupt enable.
		0	DAT3 edge detect interrupt is disabled.
		1	DAT3 edge detect interrupt is enabled.
10	EDRRDY		Data receive register ready (DRRDY) interrupt enable.
		0	Data receive register ready interrupt is disabled.
		1	Data receive register ready interrupt is enabled.
9	EDXRDY		Data transmit register (MMCDXR) ready interrupt enable.
		0	Data transmit register ready interrupt is disabled.
		1	Data transmit register ready interrupt is enabled.
8	Reserved	0	Reserved.
7	ECRCRS		Response CRC error (CRCRS) interrupt enable.
		0	Response CRC error interrupt is disabled.
		1	Response CRC error interrupt is enabled.
6	ECRCRD		Read-data CRC error (CRCRD) interrupt enable.
		0	Read-data CRC error interrupt is disabled.
		1	Read-data CRC error interrupt is enabled.
5	ECRCWR		Write-data CRC error (CRCWR) interrupt enable.
		0	Write-data CRC error interrupt is disabled.
		1	Write-data CRC error interrupt is disabled.
4	ETOUTRS		Response time-out event (TOUTRS) interrupt enable.
		0	Response time-out event interrupt is disabled.
		1	Response time-out event interrupt is enabled.
3	ETOUTRD		Read-data time-out event (TOUTRD) interrupt enable.
		0	Read-data time-out event interrupt is disabled.
		1	Read-data time-out event interrupt is enabled.
2	ERSPDNE		Command/response done (RSPDNE) interrupt enable.
		0	Command/response done interrupt is disabled.
		1	Command/response done interrupt is enabled.

Table 6-11. SD Interrupt Mask Register (SDIM) Field Descriptions (continued)

Bit	Field	Value	Description
1	EBSYDNE		Busy done (BSYDNE) interrupt enable.
		0	Busy done interrupt is disabled.
		1	Busy done interrupt is enabled.
0	EDATDNE		Data done (DATDNE) interrupt enable.
		0	Data done interrupt is disabled.
		1	Data done interrupt is enabled.

6.4.6 SD Response Time-Out Register (SDTOR)

The SD response time-out register (SDTOR) defines how long the SD controller waits for a response from a memory card before recording a time-out condition in the TOUTRS bit of the SD status register 0 (SDST0). If the corresponding ETOUTRS bit in the SD interrupt mask register (SDIM) is set, an interrupt is generated when the TOUTRS bit is set in SDST0. If a memory card should require a longer time-out period than SDTOR can provide, a software time-out mechanism can be implemented.

SDTOR is shown in Figure 6-22 and described in Table 6-12.

Figure 6-22. SD Response Time-Out Register (SDTOR)

15	8	7	0
TOD_23_16		TOR	
R/W-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-12. SD Response Time-Out Register (SDTOR) Field Descriptions

Bit	Field	Value	Description
15-8	TOD_23_16	0-1Fh	Data read time-out count upper 8 bits. Used in conjunction with the TOD_15_0 bits in MMCTOD to form a 24-bit count. See MMCTOD (Section 6.4.7).
7-0	TOR	0-FFh	Time-out count for response.
		0	No time out.
		1-FFh	1 CLK memory clock cycle to 255 CLK memory clock cycles.

6.4.7 SD Data Read Time-Out Register (SDTOD)

The SD data read time-out register (SDTOD) defines how long the SD controller waits for the data from a memory card before recording a time-out condition in the TOUTRD bit of the SD status register 0 (SDST0). If the corresponding ETOUTRD bit in the SD interrupt mask register (SDIM) is set, an interrupt is generated when the TOUTRD bit is set in SDST0. If a memory card should require a longer time-out period than SDTOD can provide, a software time-out mechanism can be implemented.

SDTOD is shown in Figure 6-23 and described in Table 6-13.

Figure 6-23. SD Data Read Time-Out Register (SDTOD)

15–0
TOD_15_0
R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-13. SD Data Read Time-Out Register (SDTOD) Field Descriptions

Bit	Field	Value	Description
15-0	TOD_15_0	0-1F FFFFh	Data read time-out count. Used in conjunction with the TOD_23_16 bits in MMCTOR to form a 24-bit count. See MMCTOR (Section 6.4.6).
		0	No time out.
		1-FFFFh	1 CLK clock cycle to 64,000 CLK clock cycles. When used in conjunction with TOD_23_16, the value range will be 1-1F FFFF clock cycles.

6.4.8 SD Block Length Register (SDBLEN)

The SD block length register (SDBLEN) specifies the data block length in bytes. This value must match the block length setting in the memory card. SDBLEN is shown in Figure 6-24 and described in Table 6-14.

Figure 6-24. SD Block Length Register (SDBLEN)

15–12	11–0
Reserved	BLEN
R-0	R/W-200h

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-14. SD Block Length Register (SDBLEN) Field Descriptions

Bit	Field	Value	Description
15-12	Reserved	0	Reserved.
11-0	BLEN	1h-FFFh	Block length. This field is used to set the block length, which is the byte count of a data block. The value 0 is prohibited.

NOTE: The BLEN bits value must be the same as the CSD register settings in the eMMC/SD card. To be precise, it should match the value of the READ_BL_LEN field for read, or WRITE_BL_LEN field for write.

6.4.9 SD Number of Blocks Register (SDNBLK)

The SD number of blocks register (SDNBLK) specifies the number of blocks for a multiple-block transfer. SDNBLK is shown in Figure 6-25 and described in Table 6-15.

Figure 6-25. SD Number of Blocks Register (SDNBLK)

15	0
NBLK	
R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-15. SD Number of Blocks Register (SDNBLK) Field Descriptions

Bit	Field	Value	Description
15-0	NBLK	0-FFFFh	Number of blocks. This field is used to set the total number of blocks to be transferred.
		0	Infinite number of blocks. The MMC controller reads/writes blocks of data until a STOP_TRANSMISSION command is written to the MMC command registers (MMCSD1 and MMCSD2).
		1h-FFFFh	*n* blocks. The MMC controller reads/writes only *n* blocks of data, even if the STOP_TRANSMISSION command has not been written to the MMC command registers (MMCSD1 and MMCSD2).

6.4.10 SD Number of Blocks Counter Register (SDNBLC)

The SD number of blocks counter register (SDNBLC) is a down-counter for tracking the number of blocks remaining to be transferred during a multiple-block transfer. SDNBLC is shown in Figure 6-26 and described in Table 6-16.

Figure 6-26. SD Number of Blocks Counter Register (SDNBLC)

15	0
NBLC	
R-FFFFh	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-16. SD Number of Blocks Counter Register (SDNBLC) Field Descriptions

Bit	Field	Value	Description
15-0	NBLC	0-FFFFh	Read this field to determine the number of blocks remaining to be transferred.

6.4.11 SD Data Receive Register (SDDRR1) and (SDDRR2)

The SD data receive registers (SDDRR1 and SDDRR2) are used for storing the data received from the SD card. The CPU or the DMA controller can read data from this register. SDDRR1 and SDDRR2 expects the data in little-endian format. SDDRR1 is shown in Figure 6-27 and described in Table 6-17. SDDRR2 is shown in Figure 6-28 and described in Table 6-18.

Figure 6-27. SD Data Receive Register (SDDRR1)

15	0
DRR1	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 6-17. SD Data Receive Register (SDDRR1) Field Descriptions

Bit	Field	Value	Description
15-0	DRR1	0-FFFFh	Data receive 1.

Figure 6-28. SD Data Receive Register (SDDRR2)

15	0
DRR2	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 6-18. SD Data Receive Register (SDDRR2) Field Descriptions

Bit	Field	Value	Description
15-0	DRR2	0-FFFFh	Data receive 2.

6.4.12 SD Data Transmit Registers (SDDXR1) and (SDDXR2)

The SD data transmit registers (SDDXR1 and SDDXR2) are used for storing the data to be transmitted from the SD controller to the memory card. The CPU or the DMA controller can write data to this register to be transmitted. SDDXR1 and SDDXR2 data is based on the endian setting in the SDCTL register. SDDXR1 is shown in Figure 6-29 and described in Table 6-19. SDDXR2 is shown in Figure 6-30 and described in Table 6-20.

Figure 6-29. SD Data Transmit Register (SDDXR)

15	0
DXR1	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 6-19. SD Data Transmit Register (SDDXR) Field Descriptions

Bit	Field	Value	Description
15-0	DXR1	0-FFFFh	Data transmit 1.

Figure 6-30. SD Data Transmit Register (SDDXR2)

15	0
DXR2	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 6-20. SD Data Transmit Register (SDDXR2) Field Descriptions

Bit	Field	Value	Description
15-0	DXR2	0-FFFFh	Data transmit 2.

6.4.13 eMMC Command Registers (MMCSD1) and (MMCSD2)

NOTE: Writing to the MMC command registers (MMCSD1 and MMCSD2) causes the MMC controller to send the programmed command. Therefore, the MMC argument registers (MMCARG1/MMCARG2) must be loaded properly before a write to MMCSD.

The MMC command registers (MMCSD1 and 2) specifies the type of command to be sent and defines the operation (command, response, additional activity) for the MMC controller. The content of MMCSD is kept after the transfer to the transmit shift register. MMCSD is shown in Figure 6-31 and described in Table 6-21.

When the CPU writes to MMCSD1 and 2, the MMC controller sends the programmed commands, including any arguments in the MMCARG1/MMCARG2 registers. For the format of a command (index, arguments, and other bits), see Figure 6-33 and Table 6-23.

NOTE: Writes to MMCSD2 should only occur after a write command has been written to the MMCSD1 register for DMA data transfers.

Figure 6-31. eMMC Command Register 1 (MMCSD1)

15	14	13	12	11	10	9	8
DCLR	INITCK	WDATX	STRMTP	DTRW	RSPFMT		BSYEXP
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0		R/W-0

7	6	5	0
PPLEN	Reserved	CMD	
R/W-0	R-0	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-21. eMMC Command Register 1 (MMCSD1) Field Descriptions

Bit	Field	Value	Description
15	DCLR		Data receive/transmit clear. Use this bit to clear the data receive ready (DRRDY) bit and the data transmit ready (DXRDY) bit in the MMC status register 0 (MMCST0) before a new read or write sequence. This clears any previous status.
		0	Do not clear DRRDY and DXRDY bits in MMCST0.
		1	Clear DRRDY and DXRDY bits in MMCST0.
14	INITCK		Initialization clock cycles.
		0	Do not insert initialization clock cycles.
		1	Insert initialization clock cycles; insert 80 CLK cycles before sending the command specified in the CMD bits. These dummy clock cycles are required for resetting a card after power on.
13	WDATX		Data transfer indicator.
		0	There is no data transfer associated with the command being sent.
		1	There is data transfer associated with the command being sent.
12	STRMTP		Stream transfer enable.
		0	If WDATX = 1, the data transfer is a block transfer. The data transfer stops after the movement of the programmed number of bytes (defined by the programmed block size and the programmed number of blocks).
		1	If WDATX = 1, the data transfer is a stream transfer. Once the data transfer is started, the data transfer does not stop until the MMC controller issues a stop command to the memory card.
11	DTRW		Data transfer write enable.
		0	If WDATX = 1, the data transfer is a read operation.
		1	If WDATX = 1, the data transfer is a write operation.

Table 6-21. eMMC Command Register 1 (MMCSD1) Field Descriptions (continued)

Bit	Field	Value	Description
10-9	RSPFMT	0-3h	Response format (expected type of response to the command).
		0	No response.
		1h	R1, R4, R5, or R6 response. 48 bits with CRC.
		2h	R2 response. 136 bits with CRC.
		3h	R3 response. 48 bits with no CRC.
8	BSYEXP		Busy expected. If an R1b (R1 with busy) response is expected, set RSPFMT = 1h and BSYEXP = 1.
		0	A busy signal is not expected.
		1	A busy signal is expected.
7	PPLEN		Push pull enable.
		0	Push pull driver of CMD line is disabled (open drain).
		1	Push pull driver of CMD line is enabled.
6	Reserved	0	Reserved.
5-0	CMD	0-3Fh	Command index. This field contains the command index for the command to be sent to the memory card.

Figure 6-32. eMMC Command Register 2 (MMCSD2)

15 — 1	0
Reserved	DMATRIG
R-0	W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-22. eMMC Command Register 2 (MMCSD2) Field Descriptions

Bit	Field	Value	Description
15-1	Reserved	0	Reserved.
0	DMATRIG		Generate a DMA event once to trigger the first DMA transfer for data write operations (subsequent DMA events are automatically generated).
		0	DMA transfer event generation is disabled.
		1	Trigger a DMA transfer event for the first data transfer to the FIFO.

Figure 6-33. Command Format

47	46	45 — 40	39 — 24
Start	Transmission	Command index	Argument, high part

23 — 8	7 — 1	0
Argument, low part	CRC7	End

Table 6-23. Command Format

Bit Position of Command	Register	Description
47	-	Start bit
46	-	Transmission bit
45-40	MMCSD(5-0)	Command index (CMD)
39-24	MMCARG1	Argument, high part
23-8	MMCARG2	Argument, low part
7-1	-	CRC7
0	-	End bit

6.4.14 SD Argument Registers (SDARG1) and (SDARG2)

NOTE: Do not modify the SD argument registers (SDARG1 and SDARG2) while they are being used for an operation.

The SD argument registers (SDARG1 and SDARG2) specifies the arguments to be sent with the command specified in the MMC command register (MMCSD). Writing to MMCSD causes the MMC controller to send a command; therefore, SDARG1 and SDARG2 must be configured before writing to MMCSD. The content of SDARG1 and SDARG2 are kept after the transfer to the shift register; however, modification to SDARG1 and SDARG2 are not allowed during a sending operation. SDARG1 is shown in Figure 6-34 and described in Table 6-24. SDARG2 is shown in Figure 6-35 and described in Table 6-25 For the format of a command, see Figure 6-33 and Table 6-23.

Figure 6-34. SD Data Transmit Register (SDARG1)

15   0
ARG1
R/W-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 6-24. SD Argument Register (SDARG1) Field Descriptions

Bit	Field	Value	Description
15-0	ARG1	0-FFFFh	Argument, high and low parts.

Figure 6-35. SD Data Transmit Register (SDARG2)

15   0
ARG2
R/W-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 6-25. SD Argument Register (SDARG2) Field Descriptions

Bit	Field	Value	Description
15-0	ARG2	0-FFFFh	Argument, high and low parts.

6.4.15 SD Response Registers (SDRSP0-SDRSP7)

Each command has a preset response type. When the SD controller receives a response, it is stored in some or all of the 4 SD response registers (SDRSP0-SDRSP7). The response registers are updated as the responses arrive, even if the CPU has not read the previous contents.

As shown in Figure 6-36 through Figure 6-43, each of the SD response registers holds up to 16 bits. Table 6-26 and Table 6-27 show the format for each type of response and which SD response registers are used for the bits of the response. The first byte of the response is a command index byte and is stored in the SD command index register (SDCIDX).

Figure 6-36. SD Response Register 0 (SDRSP0)

15	0
MMCRSP0	
R-0	

LEGEND: R = Read only; -*n* = value after reset

Figure 6-37. SD Response Register 1 (SDRSP1)

15	0
MMCRSP1	
R-0	

LEGEND: R = Read only; -*n* = value after reset

Figure 6-38. SD Response Register 2 (SDRSP2)

15	0
MMCRSP2	
R-0	

LEGEND: R = Read only; -*n* = value after reset

Figure 6-39. SD Response Register 3 (SDRSP3)

15	0
MMCRSP3	
R-0	

LEGEND: R = Read only; -*n* = value after reset

Figure 6-40. SD Response Register 4 (SDRSP4)

15	0
MMCRSP4	
R-0	

LEGEND: R = Read only; -*n* = value after reset

Figure 6-41. SD Response Register 5 (SDRSP5)

15	0
MMCRSP5	
R-0	

LEGEND: R = Read only; -*n* = value after reset

Figure 6-42. SD Response Register 6 (SDRSP6)

15	0
MMCRSP6	
R-0	

LEGEND: R = Read only; -*n* = value after reset

Figure 6-43. SD Response Register 7 (SDRSP7)

15 0
MMCRSP7
R-0

LEGEND: R = Read only; -*n* = value after reset

Table 6-26. R1, R3, R4, R5, or R6 Response (48 Bits)

Bit Position of Response	Register
47-40	MMCCIDX
39-24	MMCRSP7
23-8	MMCRSP6
7-0	MMCRSP5
-	MMCRSP4-0

Table 6-27. R2 Response (136 Bits)

Bit Position of Response	Register
135-128	MMCCIDX
127-112	MMCRSP7
111-96	MMCRSP6
95-80	MMCRSP5
79-64	MMCRSP4
63-48	MMCRSP3
47-37	MMCRSP2
31-16	MMCRSP1
15-0	MMCRSP0

6.4.16 SD Data Response Register (SDDRSP)

After the SD controller sends a data block to a memory card, the CRC status from the memory card is stored in the CRC status register, shown in Figure 6-44 and described in Table 6-28.

Figure 6-44. SD Data Response Register (SDDRSP)

15	8	7	0
Reserved		DRSP	
R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-28. SD Data Response Register (SDDRSP) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved.
7-0	DRSP	0-FFh	During a write operation (see Section 6.2.4.1), the CRC status token is stored in DRSP.

6.4.17 SD Command Index Register (SDCIDX)

The SD command index register (SDCIDX) stores the first byte of a response from a memory card. Table 6-26 and Table 6-27 show the format for each type of response. SDCIDX is shown in Figure 6-45 and described in Table 6-29.

Figure 6-45. SD Command Index Register (SDCIDX)

15	8	7	6	5	0
Reserved		STRT	XMIT	CIDX	
R-0		R/W-0	R/W-0	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-29. SD Command Index Register (SDCIDX) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved.
7	STRT	0-1	Start bit. When the MMC controller receives a response, the start bit is stored in STRT.
6	XMIT	0-1	Transmission bit. When the MMC controller receives a response, the transmission bit is stored in XMIT.
5-0	CIDX	0-3Fh	Command index. When the MMC controller receives a response, the command index is stored in CIDX.

6.4.18 SDIO Control Register (SDIOCTL)

The SDIO control register (SDIOCTL) is shown in Figure 6-46 and described in Table 6-30.

Figure 6-46. SDIO Control Register (SDIOCTL)

15 2	1	0
Reserved	RDWTCR	RDWTRQ
R-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-30. SDIO Control Register (SDIOCTL) Field Descriptions

Bit	Field	Value	Description
15-2	Reserved	0	Reserved.
1	RDWTCR		Read wait enable for CRC error. To end the read wait operation, write 0 to RDWTRQ. (No need to clear RDWTCR).
		0	Read wait is disabled.
		1	Automatically start read wait on CRC error detection during multiple block read access and not the last block to be transferred. RDWTRQ is automatically set to 1.
0	RDWTRQ		Read wait request. To end the read wait operation, write 0 to RDWTRQ.
		0	End read wait operation and release DAT[2].
		1	Set a read wait request. Read wait operation starts 2 clocks after the end of the read data block. MMC interface asserts low level on DAT[2] until RDWTRQ is cleared to 0.

6.4.19 SDIO Status Register 0 (SDIOST0)

The SDIO status register 0 (SDIOST0) is shown in Figure 6-47 and described in Table 6-31.

Figure 6-47. SDIO Status Register 0 (SDIOST0)

15	2	1	0
Reserved	RDWTST	INTPRD	DAT1
R-0	R-0	R-0	R-1

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-31. SDIO Status Register 0 (SDIOST0) Field Descriptions

Bit	Field	Value	Description
15-3	Reserved	0	Reserved.
2	RDWTST		Read wait status.
		0	Read wait operation not in progress.
		1	Read wait operation in progress.
1	INTPRD		Interrupt period.
		0	Interrupt not in progress.
		1	Interrupt in progress.
0	DAT1		This bit reflects the external state of the SD_DATA1 pin.
		0	Logic-low level on the SD_DATA1 pin.
		1	Logic-high level on the SD_DATA1 pin.

6.4.20 SDIO Interrupt Enable Register (SDIOIEN)

The SDIO interrupt enable register (SDIOIEN) is shown in Figure 6-48 and described in Table 6-32.

Figure 6-48. SDIO Interrupt Enable Register (SDIOIEN)

15 … 2	1	0
Reserved	RWSEN	IOINTEN
R-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-32. SDIO Interrupt Enable Register (SDIOIEN) Field Descriptions

Bit	Field	Value	Description
15-2	Reserved	0	Reserved.
1	RWSEN		Read wait interrupt enable.
		0	Read wait interrupt is disabled.
		1	Read wait interrupt is enabled.
0	IOINTEN		SDIO card interrupt enable.
		0	SDIO card interrupt is disabled.
		1	SDIO card interrupt is enabled.

6.4.21 SDIO Interrupt Status Register (SDIOIST)

The SDIO interrupt status register (SDIOIST) is shown in Figure 6-49 and described in Table 6-33.

Figure 6-49. SDIO Interrupt Status Register (SDIOIST)

15 … 2	1	0
Reserved	RWS	IOINT
R-0	R/W1C-0	R/W1C-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-33. SDIO Interrupt Status Register (SDIOIST) Field Descriptions

Bit	Field	Value	Description
15-2	Reserved	0	Reserved.
1	RWS		Read wait interrupt status. Write a 1 to clear this bit.
		0	Read wait interrupt did not occur.
		1	Read wait interrupt occurred. Read wait operation starts and read wait interrupt is enabled (RWSEN = 1 in SDIOIEN).
0	IOINT		SDIO card interrupt status. Write a 1 to clear this bit.
		0	SDIO card interrupt did not occur.
		1	SDIO card interrupt occurred. SDIO card interrupt is detected and SDIO card interrupt is enabled (IOINTEN = 1 in SDIOIEN).

6.4.22 SD FIFO Control Register (SDFIFOCTL)

The MMC FIFO control register (MMCFIFOCTL) is shown in Figure 6-50 and described in Table 6-34.

Figure 6-50. SD FIFO Control Register (SDFIFOCTL)

15 8
Reserved
R-0

7 5	4 3	2	1	0
Reserved	ACCWD	FIFOLEV	FIFODIR	FIFORST
R-0	R/W-0	R/W-0	R/W-0	W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-34. SD FIFO Control Register (SDFIFOCTL) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved
4-3	ACCWD	0-3h	Access width. Used by FIFO control to determine full/empty flag.
		0	CPU/DMA access width of 4 bytes
		1h	CPU/DMA access width of 3 bytes
		2h	CPU/DMA access width of 2 bytes
		3h	CPU/DMA access width of 1 byte
2	FIFOLEV		FIFO level. Sets the threshold level that determines when the DMA request and the FIFO threshold interrupt are triggered.
		0	DMA request every 128 bits sent/received.
		1	DMA request every 256 bits sent/received.
1	FIFODIR		FIFO direction. Determines if the FIFO is being written to or read from.
		0	Read from FIFO.
		1	Write to FIFO.
0	FIFORST		FIFO reset. Resets the internal state of the FIFO.
		0	FIFO reset is disabled.
		1	FIFO reset is enabled.

Universal Asynchronous Receiver/Transmitter (UART)

This chapter describes the features and operations of the universal asynchronous receiver/transmitter (UART) in the device.

Topic **Page**

7.1 Introduction

The following sections describe the features of the universal asynchronous receiver/transmitter (UART) peripheral.

7.1.1 *Purpose of the Peripheral*

The UART performs serial-to-parallel conversions on data received from a peripheral device and parallel-to-serial conversion on data received from the CPU. The CPU can read the UART status at any time. The UART includes control capability and a processor interrupt system that can be tailored to minimize software management of the communications link.

The UART includes a programmable baud generator capable of dividing the UART input clock by divisors from 1 to 65,535 and producing a 16 x reference clock for the internal transmitter and receiver logic. For detailed timing and electrical specifications for the UART, see the device-specific data manual.

The UART peripheral is based on the industry standard TL16C550 asynchronous communications element, which in turn is a functional upgrade of the TL16C450. Functionally similar to the TL16C450 on power up (single character or TL16C450 mode), the UART can be placed in an alternate FIFO (TL16C550) mode. This relieves the CPU of excessive software overhead by buffering received and transmitted characters. The receiver and transmitter FIFOs store up to 16 bytes including three additional bits of error status per byte for the receiver FIFO.

7.1.2 *Features*

The UART peripheral has the following features:

- Programmable baud rates (frequency pre-scale values from 1 to 65535).
- Fully programmable serial interface characteristics:
  - 5, 6, 7, or 8-bit characters.
  - Even, odd, or no PARITY bit generation and detection.
  - 1, 1.5, or 2 STOP bit generation.
- 16-byte depth transmitter and receiver FIFOs:
  - The UART can be operated with or without the FIFOs.
  - 1, 4, 8, or 14 byte selectable receiver FIFO trigger level for autoflow control and DMA.
- DMA signaling capability for both received and transmitted data.
- CPU interrupt capability for both received and transmitted data.
- False START bit detection.
- Line break generation and detection.
- Internal diagnostic capabilities:
  - Loopback controls for communications link fault isolation.
  - Break, parity, overrun, and framing error simulation.
- Programmable autoflow control using CTS and RTS signals.

Table 7-1 summarizes the capabilities supported on the UART.

Table 7-1. UART Supported Features/Characteristics by Instance

Feature	Support
5, 6, 7 or 8-bit characters	Supported
Even, odd, or no PARITY bit	Supported
1, 1.5, or 2 STOP bit generation	Supported
Line break generation and detection	Supported
Internal loop back	Supported
DMA sync events for both received and transmitted data	Supported
1, 4, 8, or 14 byte selectable receiver FIFO trigger level	Supported
Polling/Interrupt	Supported
Modem control functions using CTS and RTS	Supported
Autoflow control using CTS and RTS	Supported
DTR and DSR	Not supported
Ring indication	Not supported
Carrier detection	Not supported
Single-character transfer mode (mode 0) in DMA mode	Not supported

7.1.3 Functional Block Diagram

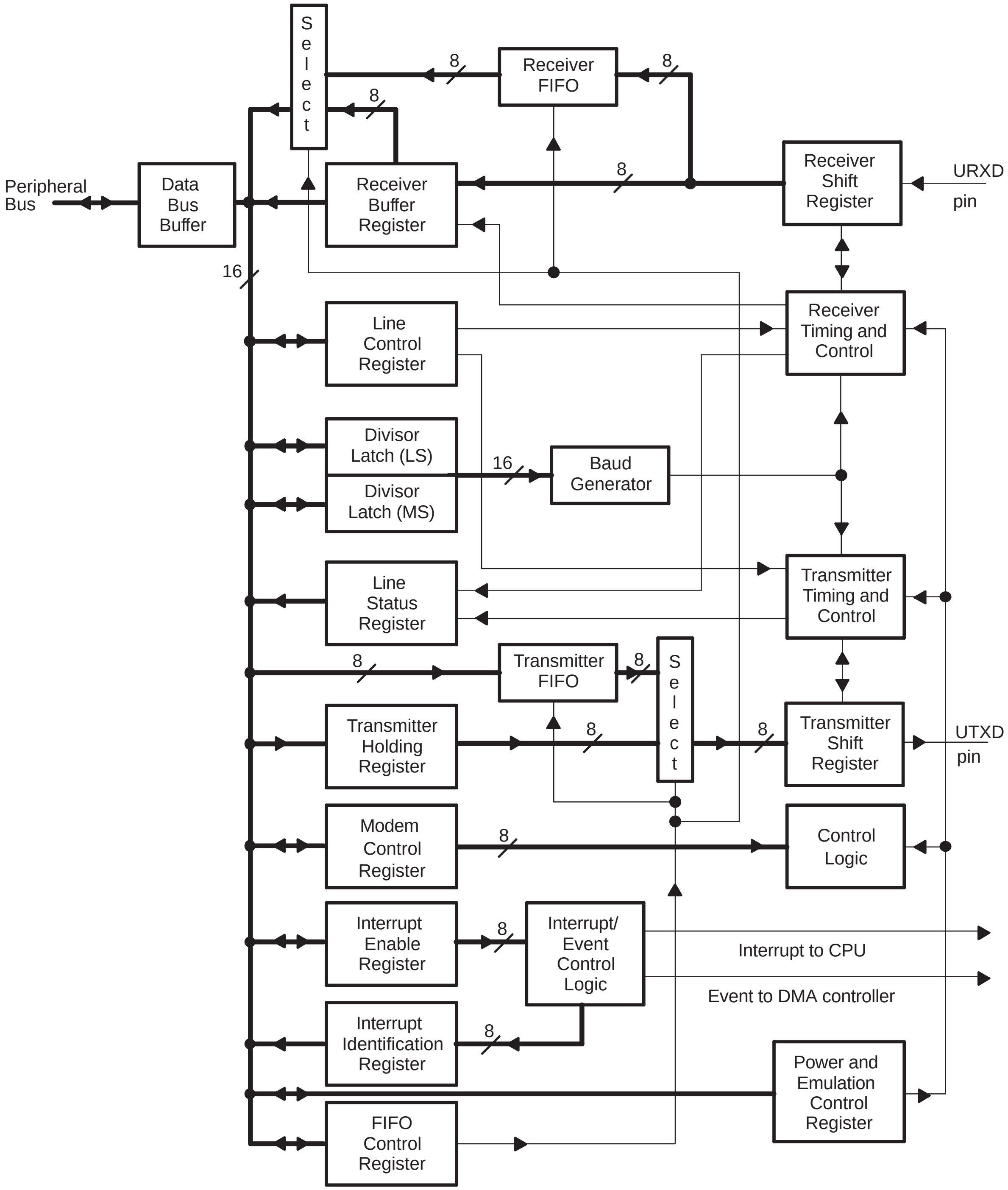
A functional block diagram of the UART is shown in Figure 7-1.

7.1.4 Industry Standard(s) Compliance Statement

The UART peripheral is based on the industry standard TL16C550 asynchronous communications element, which is a functional upgrade of the TL16C450. Any deviations in supported functions are indicated in Table 7-1.

The information in this document assumes the reader is familiar with these standards.

Figure 7-1. UART Block Diagram



7.2 Peripheral Architecture

7.2.1 Clock Generation and Control

The UART bit clock is derived from the internal system clock. Figure 7-2 is a conceptual clock generation diagram for the UART. The clock generator receives either the real-time clock (RTC) or a signal from an external clock source and produces DSP system clock. This clock is used by the DSP CPU and peripherals.

The UART contains a programmable baud generator that takes the UART input clock and divides it by a divisor in the range between 1 and (2^{16} - 1) to produce a baud clock (BCLK). The frequency of BCLK is sixteen times (16 ×) the baud rate; each received or transmitted bit lasts 16 BCLK cycles. When the UART is receiving, the bit is sampled in the 8th BCLK cycle. The formula to calculate the divisor is:

$$\text{Divisor} = \frac{\text{UART input clock frequency}}{\text{Desired baud rate} \times 16} \qquad (13)$$

Two 8-bit register fields (DLH and DLL), called divisor latches, hold this 16-bit divisor. DLH holds the most significant bits of the divisor, and DLL holds the least significant bits of the divisor. For information about these register fields, see Section 7.3. These divisor latches must be loaded during initialization of the UART in order to ensure desired operation of the baud generator. Writing to the divisor latches results in two wait states being inserted during the write access while the baud generator is loaded with the new value.

Figure 7-2 summarizes the relationship between the transferred data bit, BCLK, and the UART input clock.

Example baud rates and divisor values relative to a 50-and 100-MHz UART input clock are shown in Table 7-2 and Table 7-3 respectively. Refer to the device-specific data sheet to determine the maximum baud rate supported on the DSP.

The device DSP includes logic which can be used to gate the clock to its on-chip peripherals. The UART input clock can be enabled and disabled through the peripheral clock gating configuration register 1 (PCGCR1).

Figure 7-2. UART Clock Generation Diagram

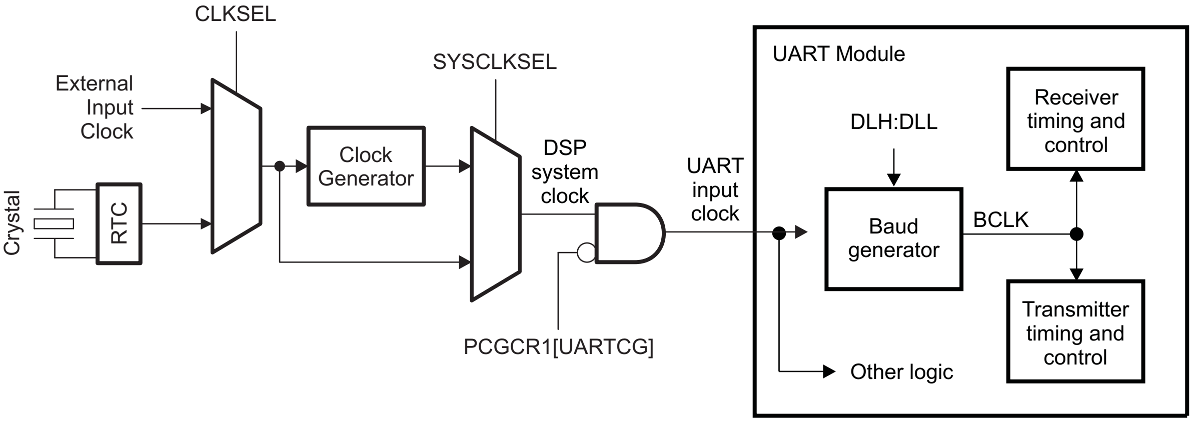


Figure 7-3. Relationship between Data Bit, BCLK, and UART Input Clock

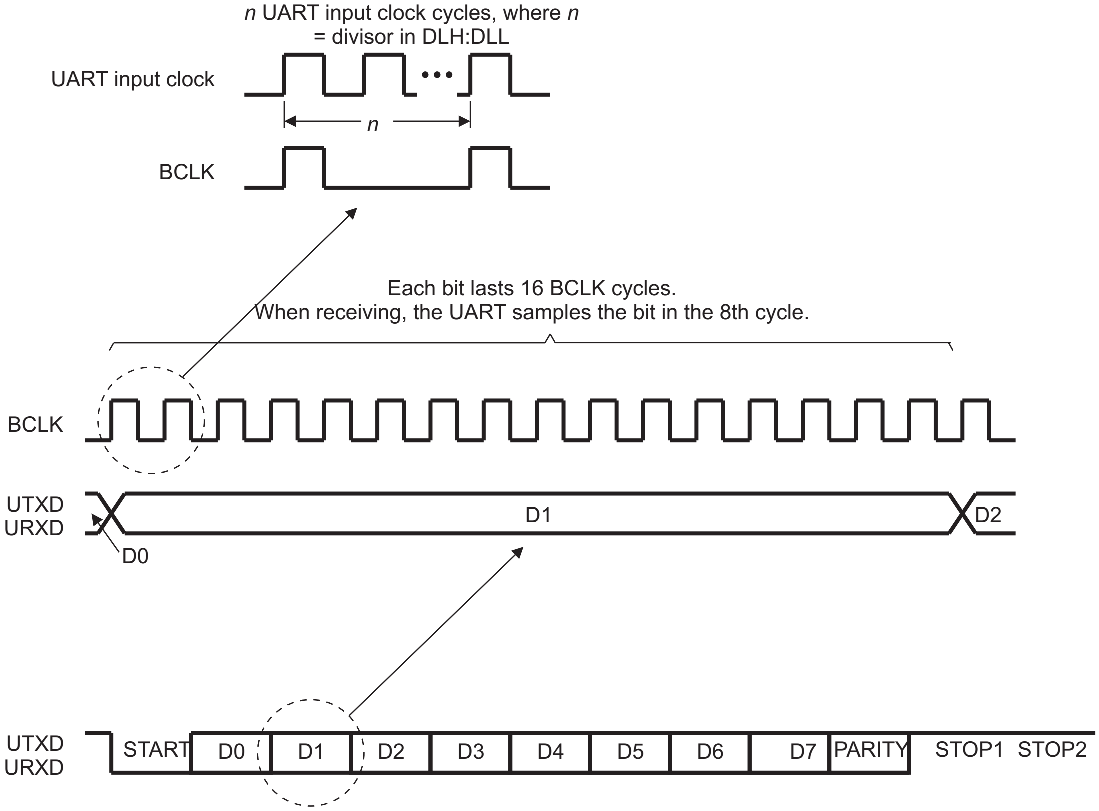

n UART input clock cycles, where *n* = divisor in DLH:DLL
UART input clock
n
BCLK
Each bit lasts 16 BCLK cycles.
When receiving, the UART samples the bit in the 8th cycle.
BCLK
UTXD
URXD
D0
D1
D2
UTXD
URXD
START
D0
D1
D2
D3
D4
D5
D6
D7
PARITY
STOP1
STOP2

Table 7-2. Baud Rate Examples for 50-MHz UART Input Clock

Baud Rate	Divisor Value	Actual Baud Rate	Error (%)
2400	1563	2399.23	-0.032
4800	781	4801.54	0.032
9600	391	9590.79	-0.096
19200	195	19230.77	0.16
38400	98	38265.31	-0.351
56000	67	55970.15	-0.053
128000	29	129310.34	1.024

Table 7-3. Baud Rate Examples for 100-MHz UART Input Clock

Baud Rate	Divisor Value	Actual Baud Rate	Error (%)
2400	2604	2400.15	0.006
4800	1302	4800.31	0.006
9600	651	9600.61	0.006
19200	326	19171.78	-0.147
38400	163	38343.56	-0.147
56000	112	55803.57	-0.351
128000	49	127551.02	-0.351

7.2.2 Signal Descriptions

The UARTs utilize a minimal number of signal connections to interface with external devices. The UART signal descriptions are included in Table 7-4.

Table 7-4. UART Signal Descriptions

Signal Name	Signal Type	Function
UTXD	Output	Serial data transmit
URXD	Input	Serial data receive
UCTS	Input	Clear-to-Send handshaking signal
URTS	Output	Request-to-Send handshaking signal

7.2.3 Pin Multiplexing

The UART pins are multiplexed with other peripherals on the DSP device. To enable UART pin functionality, software must set the parallel port mode bits of the external bus selection register (EBSR) to either 001b, 100b, or 101b. For more information on the pin multiplexing options of the device DSP, please refer to the device-specific data manual.

7.2.4 Protocol Description

7.2.4.1 Transmission

The UART transmitter section includes a transmitter hold register (THR) and a transmitter shift register (TSR). When the UART is in the FIFO mode, THR is a 16-byte FIFO. Transmitter section control is a function of the UART line control register (LCR). Based on the settings chosen in LCR, the UART transmitter sends the following to the receiving device:

- 1 START bit.
- 5, 6, 7, or 8 data bits.
- 1 PARITY bit (optional).
- 1, 1.5, or 2 STOP bits.

7.2.4.2 Reception

The UART receiver section includes a receiver shift register (RSR) and a receiver buffer register (RBR). When the UART is in the FIFO mode, RBR is a 16-byte FIFO. Receiver section control is a function of the UART line control register (LCR). Based on the settings chosen in LCR, the UART receiver accepts the following from the transmitting device:

- 1 START bit.
- 5, 6, 7, or 8 data bits.
- 1 PARITY bit (optional).
- 1 STOP bit (any other STOP bits transferred with the above data are not detected).

7.2.4.3 Data Format

The UART transmits in the following format:

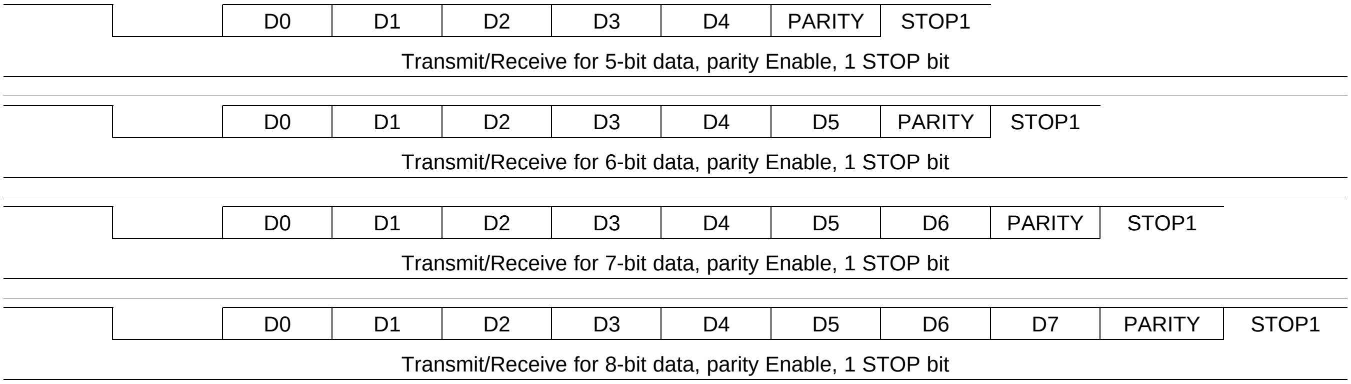
1 START bit 5, 6, 7, or 8 data bits, depending on the data width selection. 1 PARITY bit, if parity is selected; 1, 1.5, or 2 STOP bits, depending on the STOP bit selection.

The UART receives in the following format:

1 START bit 5, 6, 7, or 8 data bits, depending on the data width selection. 1 PARITY bit, if parity is selected; 1 STOP bit.

Examples of different protocol formats are shown in Figure 7-4.

Figure 7-4. UART Example Protocol Formats



Transmit/Receive for 5-bit data, parity Enable, 1 STOP bit

Transmit/Receive for 6-bit data, parity Enable, 1 STOP bit

Transmit/Receive for 7-bit data, parity Enable, 1 STOP bit

Transmit/Receive for 8-bit data, parity Enable, 1 STOP bit

7.2.5 Operation

7.2.5.1 Transmission

The UART transmitter section includes a transmitter hold register (THR) and a transmitter shift register (TSR). When the UART is in the FIFO mode, THR is a 16-byte FIFO. Transmitter section control is a function of the UART line control register (LCR). Based on the settings chosen in LCR, the UART transmitter sends the following to the receiving device:

- 1 START bit.
- 5, 6, 7, or 8 data bits.
- 1 PARITY bit (optional).
- 1, 1.5, or 2 STOP bits.

THR receives data from the internal data bus, and when TSR is ready, the UART moves the data from THR to TSR. The UART serializes the data in TSR and transmits the data on the TX pin. In the non-FIFO mode, if THR is empty and the THR empty interrupt is enabled in the interrupt enable register (IER), an interrupt is generated. This interrupt is cleared when a character is loaded into THR. In the FIFO mode, the interrupt is generated when the transmitter FIFO is empty, and it is cleared when at least one byte is loaded into the FIFO.

7.2.5.2 Reception

The UART receiver section includes a receiver shift register (RSR) and a receiver buffer register (RBR). When the UART is in the FIFO mode, RBR is a 16-byte FIFO. Timing is supplied by the receiver clock. Receiver section control is a function of the UART line control register (LCR). Based on the settings chosen in LCR, the UART receiver accepts the following from the transmitting device:

- 1 START bit.
- 5, 6, 7, or 8 data bits.
- 1 PARITY bit (optional).
- 1 STOP bit (any other STOP bits transferred with the above data are not detected).

RSR receives the data bits from the RX pin. Then RSR concatenates the data bits and moves the resulting value into RBR (or the receiver FIFO). The UART also stores three bits of error status information next to each received character, to record a parity error, framing error, or break.

In the non-FIFO mode, when a character is placed in RBR and the receiver data-ready interrupt is enabled in the interrupt enable register (IER), an interrupt is generated. This interrupt is cleared when the character is read from RBR. In the FIFO mode, the interrupt is generated when the FIFO is filled to the trigger level selected in the FIFO control register (FCR), and it is cleared when the FIFO contents drop below the trigger level.

7.2.5.3 FIFO Modes

The following two modes can be used for servicing the receiver and transmitter FIFOs:

- FIFO interrupt mode. The FIFO is enabled and the associated interrupts are enabled. Interrupts are sent to the CPU to indicate when specific events occur.
- FIFO poll mode. The FIFO is enabled but the associated interrupts are disabled. The CPU polls status bits to detect specific events.

Because the receiver FIFO and the transmitter FIFO are controlled separately, either one or both can be placed into the interrupt mode or the poll mode.

7.2.5.3.1 FIFO Interrupt Mode

When the receiver FIFO is enabled in the FIFO control register (FCR) and the receiver interrupts are enabled in the interrupt enable register (IER), the interrupt mode is selected for the receiver FIFO. The following are important points about the receiver interrupts:

- The receiver data-ready interrupt is issued to the CPU when the FIFO has reached the trigger level that is programmed in FCR. It is cleared when the CPU or the DMA controller reads enough characters from the FIFO such that the FIFO drops below its programmed trigger level.
- The receiver line status interrupt is generated in response to an overrun error, a parity error, a framing error, or a break. This interrupt has higher priority than the receiver data-ready interrupt. For details, see Section 7.2.9.
- The data-ready (DR) bit in the line status register (LSR) indicates the presence or absence of characters in the receiver FIFO. The DR bit is set when a character is transferred from the receiver shift register (RSR) to the empty receiver FIFO. The DR bit remains set until the FIFO is empty again.

- A receiver time-out interrupt occurs if all of the following conditions exist:
  - At least one character is in the FIFO,
  - The most recent character was received more than four continuous character times ago. A character time is the time allotted for 1 START bit, *n* data bits, 1 PARITY bit, and 1 STOP bit, where *n* depends on the word length selected with the WLS bits in the line control register (LCR). See Table 7-5.
  - The most recent read of the FIFO has occurred more than four continuous character times before.
- Character times are calculated by using the baud rate.
- When a receiver time-out interrupt has occurred, it is cleared and the time-out timer is cleared when the CPU or the DMA controller reads one character from the receiver FIFO. The interrupt is also cleared if a new character is received in the FIFO or if the URRST bit is cleared in the power and emulation management register (PWREMU_MGMT).
- If a receiver time-out interrupt has not occurred, the time-out timer is cleared after a new character is received or after the CPU or DMA reads the receiver FIFO.

When the transmitter FIFO is enabled in FCR and the transmitter holding register empty interrupt is enabled in IER, the interrupt mode is selected for the transmitter FIFO. The transmitter holding register empty interrupt occurs when the transmitter FIFO is empty. It is cleared when the transmitter hold register (THR) is loaded (1 to 16 characters may be written to the transmitter FIFO while servicing this interrupt).

Table 7-5. Character Time for Word Lengths

Word Length (*n*)	Character Time	Four Character Times
5	Time for 8 bits	Time for 32 bits
6	Time for 9 bits	Time for 36 bits
7	Time for 10 bits	Time for 40 bits
8	Time for 11 bits	Time for 44 bits

7.2.5.3.2 FIFO Poll Mode

When the receiver FIFO is enabled in the FIFO control register (FCR) and the receiver interrupts are disabled in the interrupt enable register (IER), the poll mode is selected for the receiver FIFO. Similarly, when the transmitter FIFO is enabled and the transmitter interrupts are disabled, the transmitter FIFO is in the poll mode. In the poll mode, the CPU detects events by checking bits in the line status register (LSR):

- The RXFIFOE bit indicates whether there are any errors in the receiver FIFO.
- The TEMT bit indicates that both the transmitter holding register (THR) and the transmitter shift register (TSR) are empty.
- The THRE bit indicates when THR is empty.
- The BI (break), FE (framing error), PE (parity error), and OE (overrun error) bits specify which error or errors have occurred.
- The DR (data-ready) bit is set as long as there is at least one byte in the receiver FIFO.

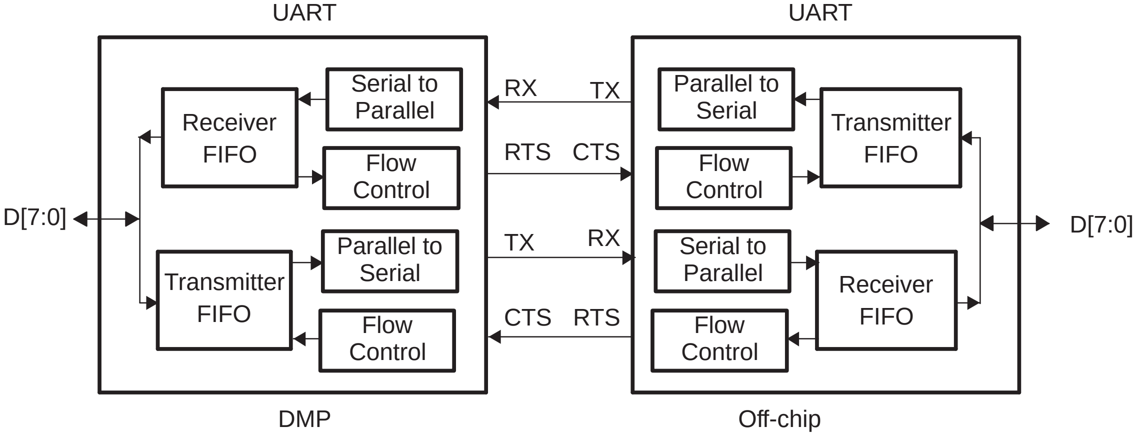
Also, in the FIFO poll mode:

- The interrupt identification register (IIR) is not affected by any events because the interrupts are disabled.
- The UART does not indicate when the receiver FIFO trigger level is reached or when a receiver time-out occurs.

7.2.5.4 Autoflow Control

The UART can employ autoflow control by connecting the CTS and RTS signals. The CTS input must be active before the transmitter FIFO can transmit data. The RTS becomes active when the receiver needs more data and notifies the sending device. When RTS is connected to CTS, data transmission does not occur unless the receiver FIFO has space for the data. Therefore, when two UARTs are connected as shown in Figure 7-5 with autoflow enabled, overrun errors are eliminated.

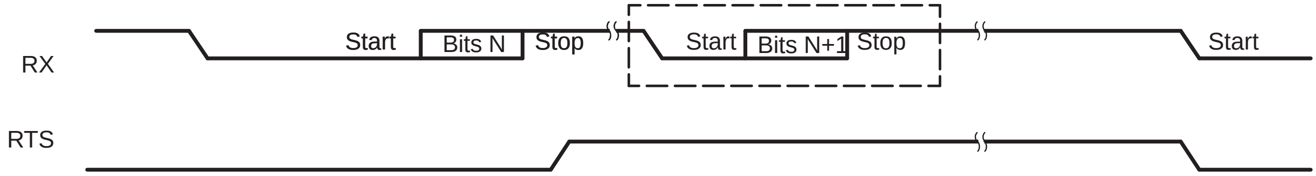
Figure 7-5. UART Interface Using Autoflow Diagram



7.2.5.4.1 RTS Behavior

RTS data flow control originates in the receiver block (see Figure 7-1). When the receiver FIFO level reaches a trigger level of 1, 4, 8, or 14 (see Figure 7-6), RTS is deasserted. The sending UART may send an additional byte after the trigger level is reached (assuming the sending UART has another byte to send), because it may not recognize the deassertion of RTS until after it has begun sending the additional byte. For trigger level 1, 4, and 8, RTS is automatically reasserted once the receiver FIFO is emptied. For trigger level 14, RTS is automatically reasserted once the receiver FIFO drops below the trigger level.

Figure 7-6. Autoflow Functional Timing Waveforms for RTS



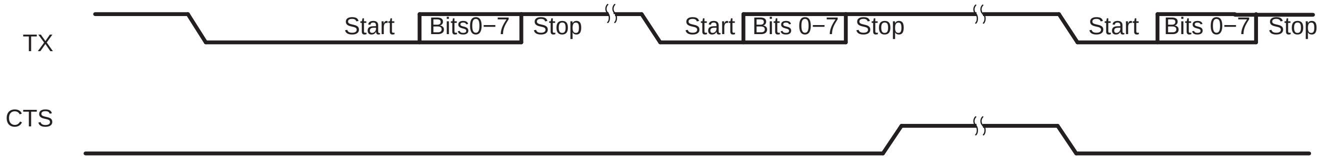
(1) N = Receiver FIFO trigger level.

(2) The two blocks in dashed lines cover the case where an additional byte is sent.

7.2.5.4.2 CTS Behavior

The transmitter checks CTS before sending the next data byte. If CTS is active, the transmitter sends the next byte. To stop the transmitter from sending the following byte, CTS must be released before the middle of the last STOP bit that is currently being sent (see Figure 7-7). When flow control is enabled, CTS level changes do not trigger interrupts because the device automatically controls its own transmitter. Without autoflow control, the transmitter sends any data present in the transmitter FIFO and a receiver overrun error may result.

Figure 7-7. Autoflow Functional Timing Waveforms for CTS



(1) When CTS is active (low), the transmitter keeps sending serial data out.

(2) When CTS goes high before the middle of the last STOP bit of the current byte, the transmitter finishes sending the current byte but it does not send the next byte.

(3) When CTS goes from high to low, the transmitter begins sending data again.

7.2.5.5 Loopback Control

The UART can be placed in the diagnostic mode using the LOOP bit in the modem control register (MCR), which internally connects the UART output back to the UART input. In this mode, the transmit and receive data paths, the transmitter and receiver interrupts, and the modem control interrupts can be verified without connecting to another UART.

7.2.6 Exception Processing

7.2.6.1 Divisor Latch Not Programmed

Since the processor reset signal has no effect on the divisor latch, the divisor latch will have an unknown value after power up. If the divisor latch is not programmed after power up, the baud clock (BCLK) will not operate and will instead be set to a constant logic 1 state.

The divisor latch values should always be reinitialized following a processor reset.

7.2.6.2 Changing Operating Mode During Busy Serial Communication

Since the serial link characteristics are based on how the control registers are programmed, the UART will expect the control registers to be static while it is busy engaging in a serial communication. Therefore, changing the control registers while the module is still busy communicating with another serial device will most likely cause an error condition and should be avoided.

7.2.7 Reset Considerations

The UART peripheral has two reset sources: software reset and hardware reset.

7.2.7.1 Software Reset Considerations

The UART peripheral can be reset by software through the transmitter reset (UTRST) and the receiver reset (URRST) bits of the UART power and emulation management register (PWREMU_MGMT) or through the UART_RST bit in the peripheral reset control register (PRCR).

The UTRST bit controls the transmitter part of the UART only. When UTRST is cleared to 0, the transmitter is disabled and placed in reset. When UTRST is set to 1, the transmitter is enabled. The URRST bit controls the receiver portion of the UART in a similar fashion. In each case, placing the receiver and/or transmitter in reset will reset the state machine of the affected portion but does not affect the UART registers.

When PG4_RST in the peripheral reset control register (PRCR) is set to 1, a hardware reset is forced on the UART. The effects of a hardware reset are described in the next section. Please note that the UART input clock must be enabled when using UART_RST (see Section 7.2.1). Refer to the device-specific data manual for more details on PRCR.

7.2.7.2 Hardware Reset Considerations

A hardware reset is always initiated during a full chip reset. Alternatively, software can force an UART hardware reset through the PG4_RST bit of the peripheral reset control register (PRCR). See the device data manual for more details on PRCR.

NOTE: PRCR resets other peripherals besides the UART. For more details on this bit and register, refer to the device-specific data manual.

When a hardware reset occurs, all the registers of the UART peripheral are set to their default values and the UART peripheral remains disabled until the transmitter reset (UTRST) and the receiver reset (URRST) bits of the UART power and emulation management register (PWREMU_MGMT) are changed to 1.

7.2.8 Initialization

The following steps are required to initialize the UART:

1. Perform the necessary device pin multiplexing setup (see Section 7.2.3 for more details).
2. Ensure the UART is out of reset by setting UART_RST = 0 in the peripheral reset control register (PRCR).
3. Enable the UART input clock by setting UARTCG to 1 in the peripheral clock gating configuration register (PCGCR1). See the device-specific data manual for more information on PCGCR1.
4. Place the UART transmitter and receiver in reset by setting UTRST and URRST to 0 in the UART power and emulation management register (PWREMU_MGMT).
5. Set the desired baud rate by writing the appropriate clock divisor values to the divisor latch registers (DLL and DLH).
6. Select the desired trigger level and enable the FIFOs by writing the appropriate values to the FIFO control register (FCR) if the FIFOs are used. The FIFOEN bit in FCR must be set first, before the other bits in FCR are configured. Be sure to set the DMAMODE1 bit to 1 as required for proper operation between the DMA and UART.
7. Choose the desired protocol settings by writing the appropriate values to the line control register (LCR).
8. Write appropriate values to the modem control register (MCR) if autoflow control is desired. Note that all UARTs do not support autoflow control, see the device-specific data manual for supported features.
9. Choose the desired response to emulation suspend events by configuring the FREE bit and enable the UART by setting the UTRST and URRST bits in the power and emulation management register (PWREMU_MGMT).

7.2.9 Interrupt Support

7.2.9.1 Interrupt Events and Requests

The UART generates the interrupt requests described in Table 7-6. All requests are multiplexed through an arbiter to a single UART interrupt request to the CPU, as shown in Figure 7-8. Each of the interrupt requests has an enable bit in the interrupt enable register (IER) and is recorded in the interrupt identification register (IIR).

If an interrupt occurs and the corresponding enable bit is set to 1, the interrupt request is recorded in IIR and is forwarded to the CPU. If an interrupt occurs and the corresponding enable bit is cleared to 0, the interrupt request is blocked. The interrupt request is neither recorded in IIR nor forwarded to the CPU.

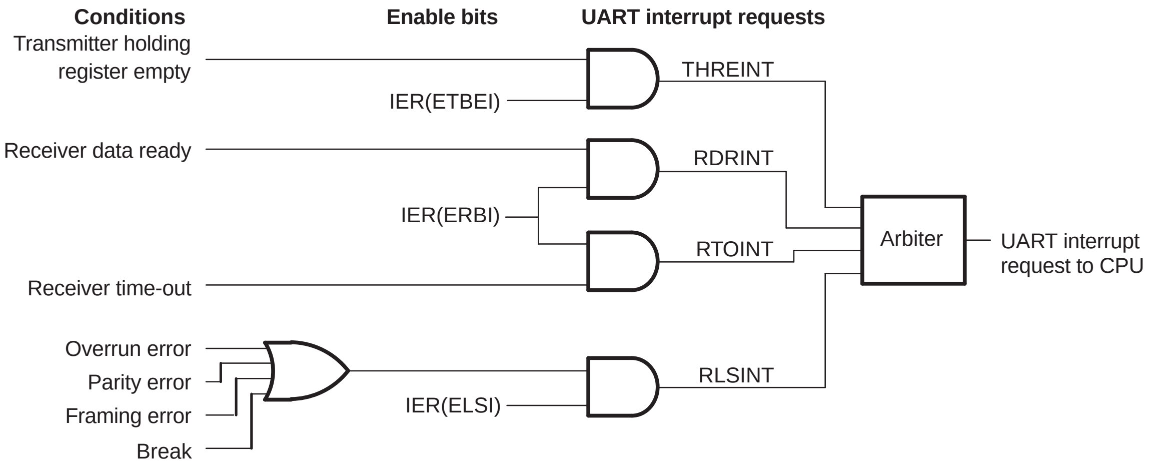
Table 7-6. UART Interrupt Requests Descriptions

UART Interrupt Request	Interrupt Source	Comment
THREINT	THR-empty condition: The transmitter holding register (THR) or the transmitter FIFO is empty. All of the data has been copied from THR to the transmitter shift register (TSR).	If THREINT is enabled in IER, by setting the ETBEI bit, it is recorded in IIR. As an alternative to using THREINT, the CPU can poll the THRE bit in the line status register (LSR).
RDAINT	Receive data available in non-FIFO mode or trigger level reached in the FIFO mode.	If RDAINT is enabled in IER, by setting the ERBI bit, it is recorded in IIR. As an alternative to using RDAINT, the CPU can poll the DR bit in the line status register (LSR). In the FIFO mode, this is not a functionally equivalent alternative because the DR bit does not respond to the FIFO trigger level. The DR bit only indicates the presence or absence of unread characters.
RTOINT	Receiver time-out condition (in the FIFO mode only): No characters have been removed from or input to the receiver FIFO during the last four character times (see Table 7-5), and there is at least one character in the receiver FIFO during this time.	The receiver time-out interrupt prevents the UART from waiting indefinitely, in the case when the receiver FIFO level is below the trigger level and thus does not generate a receiver data-ready interrupt. If RTOINT is enabled in IER, by setting the ERBI bit, it is recorded in IIR. There is no status bit to reflect the occurrence of a time-out condition.

Table 7-6. UART Interrupt Requests Descriptions (continued)

UART Interrupt Request	Interrupt Source	Comment
RLSINT	Receiver line status condition: An overrun error, parity error, framing error, or break has occurred.	If RLSINT is enabled in IER, by setting the ELSI bit, it is recorded in IIR. As an alternative to using RLSINT, the CPU can poll the following bits in the line status register (LSR): overrun error indicator (OE), parity error indicator (PE), framing error indicator (FE), and break indicator (BI).

Figure 7-8. UART Interrupt Request Enable Paths



7.2.9.2 Interrupt Multiplexing

The UART has a dedicated interrupt signal to the CPU that is not multiplexed with any other interrupt source.

7.2.10 DMA Event Support

In the FIFO mode, the UART generates the following two DMA events:

- **Receive event (URXEVT):** The trigger level for the receiver FIFO (1, 4, 8, or 14 characters) is set with the RXFIFTL bit in the FIFO control register (FCR). Every time the trigger level is reached or a receiver time-out occurs, the UART sends a receive event to the DMA controller. In response, the DMA controller reads the data from the receiver FIFO by way of the receiver buffer register (RBR).
- **Transmit event (UTXEVT):** When the transmitter FIFO is empty (when the last byte in the transmitter FIFO has been copied to the transmitter shift register), the UART sends an UTXEVT signal to the DMA controller. In response, the DMA controller refills the transmitter FIFO by way of the transmitter holding register (THR). The UTXEVT signal is also sent to the DMA controller when the UART is taken out of reset using the UTRST bit in the power and emulation management register (PWREMU_MGMT).

Activity in DMA channels can be synchronized to these events. In the non-FIFO mode, the UART generates no DMA events. Any DMA channel synchronized to either of these events must be enabled at the time the UART event is generated. Otherwise, the DMA channel will miss the event and, unless the UART generates a new event, no data transfer will occur.

7.2.11 Power Management

The UART peripheral can be clock-gated to conserve power during periods of no activity. The input clock of the UART can be turned off by using the peripheral clock gating configuration register (PCGCR). For detailed information on PCGCR, see the device-specific data manual.

7.2.12 Emulation Considerations

The FREE bit in the power and emulation management register (PWREMU_MGMT) determines how the UART responds to an emulation suspend event such as an emulator halt or breakpoint. If FREE = 0 and a transmission is in progress, the UART halts after completing the one-word transmission; if FREE = 0 and a transmission is not in progress, the UART halts immediately. If FREE = 1, the UART does not halt and continues operating normally.

Note also that emulator accesses are essentially transparent to UART operation. Emulator read operations do not affect any register contents, status bits, or operating states. Emulator writes, however, may affect register contents and may affect UART operation, depending on what register is accessed and what value is written.

The UART registers can be read from or written to during emulation suspend events, even if the UART activity has stopped.

7.3 Registers

The system programmer has access to and control over any of the UART registers that are listed in Table 7-7. These registers, which control UART operations, receive data, and transmit data, and can be accessed by the CPU at the word address specified in Table 7-7. Note that the CPU accesses all peripheral registers through its I/O space. All other addresses not listed in Table 7-7 should be considered as reserved locations and the register contents should not be modified.

The following registers share one address:

- RBR, THR, and DLL. When the DLAB bit in LCR is 0, reading from the address gives the content of RBR, and writing to the address modifies THR. When DLAB = 1, all accesses at the address read or modify DLL. DLL can also be accessed by the CPU at word address 1B10h.
- IER and DLH. When DLAB = 0, all accesses read or modify IER. When DLAB = 1, all accesses read or modify DLH. DLH can also be accessed by the CPU at word address 1B12h.
- IIR and FCR share one address. Regardless of the value of the DLAB bit, reading from the address gives the content of IIR, and writing modifies FCR.

Table 7-7. UART Registers

CPU Word Address	Acronym	Register Description	Section
1B00h	RBR	Receiver Buffer Register (read only)	Section 7.3.1
1B00h	THR	Transmitter Holding Register (write only)	Section 7.3.2
1B02h	IER	Interrupt Enable Register	Section 7.3.3
1B04h	IIR	Interrupt Identification Register (read only)	Section 7.3.4
1B04h	FCR	FIFO Control Register (write only)	Section 7.3.5
1B06h	LCR	Line Control Register	Section 7.3.6
1B08h	MCR	Modem Control Register	Section 7.3.7
1B0Ah	LSR	Line Status Register	Section 7.3.8
1B0Eh	SCR	Scratch Register	Section 7.3.9
1B10h	DLL	Divisor LSB Latch	Section 7.3.10
1B12h	DLH	Divisor MSB Latch	Section 7.3.10
1B18h	PWREMU_MGMT	Power and Emulation Management Register	Section 7.3.11

7.3.1 Receiver Buffer Register (RBR)

The receiver buffer register (RBR) is shown in Figure 7-9 and described in Table 7-8.

The UART receiver section consists of a receiver shift register (RSR) and a receiver buffer register (RBR). When the UART is in the FIFO mode, RBR is a 16-byte FIFO. Timing is supplied by the 16x receiver clock. Receiver section control is a function of the line control register (LCR).

RSR receives serial data from the RX pin. Then RSR concatenates the data and moves it into RBR (or the receiver FIFO). In the non-FIFO mode, when a character is placed in RBR and the receiver data-ready interrupt is enabled (DR = 1 in IER), an interrupt is generated. This interrupt is cleared when the character is read from RBR. In the FIFO mode, the interrupt is generated when the FIFO is filled to the trigger level selected in the FIFO control register (FCR), and it is cleared when the FIFO contents drop below the trigger level.

Access considerations:

RBR, THR, and DLL share one address. To read RBR, write 0 to the DLAB bit in LCR, and read from the shared address. When DLAB = 0, writing to the shared address modifies THR. When DLAB = 1, all accesses at the shared address read or modify DLL.

DLL also has a dedicated address. If you use the dedicated address, you can keep DLAB = 0, so that RBR and THR are always selected at the shared address.

Figure 7-9. Receiver Buffer Register (RBR)

15 8	7 0
Reserved	DATA
R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 7-8. Receiver Buffer Register (RBR) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved
7-0	DATA	0-FFh	Received data

7.3.2 Transmitter Holding Register (THR)

The transmitter holding register (THR) is shown in Figure 7-10 and described in Table 7-9.

The UART transmitter section consists of a transmitter hold register (THR) and a transmitter shift register (TSR). When the UART is in the FIFO mode, THR is a 16-byte FIFO. Transmitter section control is a function of the line control register (LCR).

THR receives data from the internal data bus and when TSR is idle, the UART moves the data from THR to TSR. The UART serializes the data in TSR and transmits the data on the TX pin. In the non-FIFO mode, if THR is empty and the THR empty (THRE) interrupt is enabled (ETBEI = 1 in IER), an interrupt is generated. This interrupt is cleared when a character is loaded into THR. In the FIFO mode, the interrupt is generated when the transmitter FIFO is empty, and it is cleared when at least one byte is loaded into the FIFO.

Access considerations:

RBR, THR, and DLL share one address. To load THR, write 0 to the DLAB bit of LCR, and write to the shared address. When DLAB = 0, reading from the shared address gives the content of RBR. When DLAB = 1, all accesses at the address read or modify DLL.

DLL also has a dedicated address. If you use the dedicated address, you can keep DLAB = 0, so that RBR and THR are always selected at the shared address.

Figure 7-10. Transmitter Holding Register (THR)

15 8	7 0
Reserved	DATA
R-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Table 7-9. Transmitter Holding Register (THR) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved
7-0	DATA	0-FFh	Data to transmit

7.3.3 Interrupt Enable Register (IER)

The interrupt enable register (IER) is used to individually enable or disable each type of interrupt request that can be generated by the UART. Each interrupt request that is enabled in IER is forwarded to the CPU. IER is shown in Figure 7-11 and described in Table 7-10.

Access considerations:

IER and DLH share one address. To read or modify IER, write 0 to the DLAB bit in LCR. When DLAB = 1, all accesses at the shared address read or modify DLH.

DLH also has a dedicated address. If you use the dedicated address, you can keep DLAB = 0, so that IER is always selected at the shared address.

Figure 7-11. Interrupt Enable Register (IER)

15	4	3	2	1	0
Reserved		Rsvd	ELSI	ETBEI	ERBI
R-0		R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 7-10. Interrupt Enable Register (IER) Field Descriptions

Bit	Field	Value	Description
15-4	Reserved	0	Reserved
3	Reserved	0	Reserved. This bit must always be written with a 0.
2	ELSI		Receiver line status interrupt enable.
		0	Receiver line status interrupt is disabled.
		1	Receiver line status interrupt is enabled.
1	ETBEI		Transmitter holding register empty interrupt enable.
		0	Transmitter holding register empty interrupt is disabled.
		1	Transmitter holding register empty interrupt is enabled.
0	ERBI		Receiver data available interrupt and character timeout indication interrupt enable.
		0	Receiver data available interrupt and character timeout indication interrupt is disabled.
		1	Receiver data available interrupt and character timeout indication interrupt is enabled.

7.3.4 Interrupt Identification Register (IIR)

The interrupt identification register (IIR) is a read-only register at the same address as the FIFO control register (FCR), which is a write-only register. When an interrupt is generated and enabled in the interrupt enable register (IER), IIR indicates that an interrupt is pending in the IPEND bit and encodes the type of interrupt in the INTID bits. IIR is shown in Figure 7-12 and described in Figure 7-12.

The UART has an on-chip interrupt generation and prioritization capability that permits flexible communication with the CPU. The UART provides three priority levels of interrupts:

- Priority 1 - Receiver line status (highest priority)
- Priority 2 - Receiver data ready or receiver timeout
- Priority 3 - Transmitter holding register empty

The FIFOEN bit in IIR can be checked to determine whether the UART is in the FIFO mode or the non-FIFO mode.

Access consideration:

IIR and FCR share one address. Regardless of the value of the DLAB bit in LCR, reading from the address gives the content of IIR, and writing to the address modifies FCR.

Figure 7-12. Interrupt Identification Register (IIR)

15 8	7 6	5 4	3 1	0
Reserved	FIFOEN	Reserved	INTID	IPEND
R-0	R-0	R-0	R-0	R-1

LEGEND: R = Read only; -*n* = value after reset

Table 7-11. Interrupt Identification Register (IIR) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved
7-6	FIFOEN	0-3h	FIFOs enabled.
		0	Non-FIFO mode.
		1h-2h	Reserved.
		3h	FIFOs are enabled. FIFOEN bit in the FIFO control register (FCR) is set to 1.
5-4	Reserved	0	Reserved.
3-1	INTID	0-7h	Interrupt type. See Table 7-12.
		0	Reserved.
		1h	Transmitter holding register empty (priority 3).
		2h	Receiver data available (priority 2).
		3h	Receiver line status (priority 1, highest).
		4h-5h	Reserved.
		6h	Character timeout indication (priority 2).
		7h	Reserved.
0	IPEND		Interrupt pending. When any UART interrupt is generated and is enabled in IER, IPEND is forced to 0. IPEND remains 0 until all pending interrupts are cleared or until a hardware reset occurs. If no interrupts are enabled, IPEND is never forced to 0.
		0	Interrupts pending.
		1	No interrupts pending.

Table 7-12. Interrupt Identification and Interrupt Clearing Information

Priority Level	IIR Bits				Interrupt Type	Interrupt Source	Event That Clears Interrupt
	3	2	1	0			
None	0	0	0	1	None	None	None
1	0	1	1	0	Receiver line status	Overrun error, parity error, framing error, or break is detected.	For an overrun error, reading the line status register (LSR) clears the interrupt. For a parity error, framing error, or break, the interrupt is cleared only after all the erroneous data have been read.
2	0	1	0	0	Receiver data-ready	Non-FIFO mode: Receiver data is ready.	Non-FIFO mode: The receiver buffer register (RBR) is read.
						FIFO mode: Trigger level reached. If four character times (see Table 7-5) pass with no access of the FIFO, the interrupt is asserted again.	FIFO mode: The FIFO drops below the trigger level. [1]
2	1	1	0	0	Receiver time-out	FIFO mode only: No characters have been removed from or input to the receiver FIFO during the last four character times (see Table 7-5), and there is at least one character in the receiver FIFO during this time.	One of the following events: • A character is read from the receiver FIFO. [1] • A new character arrives in the receiver FIFO. • The URRST bit in the power and emulation management register (PWREMU_MGMT) is loaded with 0.
3	0	0	1	0	Transmitter holding register empty	Non-FIFO mode: Transmitter holding register (THR) is empty. FIFO mode: Transmitter FIFO is empty.	A character is written to the transmitter holding register (THR).

[1] In the FIFO mode, the receiver data-ready interrupt or receiver time-out interrupt is cleared by the CPU or by the DMA controller, whichever reads from the receiver FIFO first.

7.3.5 *FIFO Control Register (FCR)*

The FIFO control register (FCR) is a write-only register at the same address as the interrupt identification register (IIR), which is a read-only register. Use FCR to enable and clear the FIFOs and to select the receiver FIFO trigger level FCR is shown in Figure 7-13 and described in Table 7-13. The FIFOEN bit must be set to 1 before other FCR bits are written to or the FCR bits are not programmed.

Access consideration:

IIR and FCR share one address. Regardless of the value of the DLAB bit, reading from the address gives the content of IIR, and writing to the address modifies FCR.

CAUTION

For proper communication between the UART and the DMA controller, the DMAMODE1 bit must be set to 1. Always write a 1 to the DMAMODE1 bit, and after a hardware reset, change the DMAMODE1 bit from 0 to 1.

Figure 7-13. FIFO Control Register (FCR)

15							8
Reserved							
R-0							

7	6	5	4	3	2	1	0
RXFIFTL		Reserved		DMAMODE1	TXCLR	RXCLR	FIFOEN
W-0		R-0		W-0	W1C-0	W1C-0	W-0

LEGEND: R = Read only; W = Write only; W1C = Write 1 to clear (writing 0 has no effect); -*n* = value after reset

Table 7-13. FIFO Control Register (FCR) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved.
7-6	RXFIFTL	0-3h	Receiver FIFO trigger level. RXFIFTL sets the trigger level for the receiver FIFO. When the trigger level is reached, a receiver data-ready interrupt is generated (if the interrupt request is enabled). Once the FIFO drops below the trigger level, the interrupt is cleared.
		0	1 byte
		1h	4 bytes
		2h	8 bytes
		3h	14 bytes
5-4	Reserved	0	Reserved.
3	DMAMODE1		DMA mode enable. Always write 1 to DMAMODE1. After a hardware reset, change DMAMODE1 from 0 to 1. DMAMODE1 = 1 is a requirement for proper communication between the UART and the DMA controller.
		0	DMA mode is disabled.
		1	DMA mode is enabled.
2	TXCLR		Transmitter FIFO clear. Write a 1 to TXCLR to clear the bit.
		0	No effect.
		1	Clears transmitter FIFO and resets the transmitter FIFO counter. The shift register is not cleared.
1	RXCLR		Receiver FIFO clear. Write a 1 to RXCLR to clear the bit.
		0	No effect.
		1	Clears receiver FIFO and resets the receiver FIFO counter. The shift register is not cleared.
0	FIFOEN		Transmitter and receiver FIFOs mode enable. FIFOEN must be set before other FCR bits are written to or the FCR bits are not programmed. Clearing this bit clears the FIFO counters.
		0	Non-FIFO mode. The transmitter and receiver FIFOs are disabled, and the FIFO pointers are cleared.
		1	FIFO mode. The transmitter and receiver FIFOs are enabled.

7.3.6 *Line Control Register (LCR)*

The line control register (LCR) is shown in Figure 7-14 and described in Table 7-14.

The system programmer controls the format of the asynchronous data communication exchange by using LCR. In addition, the programmer can retrieve, inspect, and modify the content of LCR; this eliminates the need for separate storage of the line characteristics in system memory.

Figure 7-14. Line Control Register (LCR)

15	8	7	6	5	4	3	2	1	0
Reserved		DLAB	BC	SP	EPS	PEN	STB	WLS	
R-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 7-14. Line Control Register (LCR) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved
7	DLAB		Divisor latch access bit. The divisor latch registers (DLL and DLH) can be accessed at dedicated addresses or at addresses shared by RBR, THR, and IER. Using the shared addresses requires toggling DLAB to change which registers are selected. If you use the dedicated addresses, you can keep DLAB = 0.
		0	Allows access to the receiver buffer register (RBR), the transmitter holding register (THR), and the interrupt enable register (IER) selected. At the address shared by RBR, THR, and DLL, the CPU can read from RBR and write to THR. At the address shared by IER and DLH, the CPU can read from and write to IER.
		1	Allows access to the divisor latches of the baud generator during a read or write operation (DLL and DLH). At the address shared by RBR, THR, and DLL, the CPU can read from and write to DLL. At the address shared by IER and DLH, the CPU can read from and write to DLH.
6	BC		Break control.
		0	Break condition is disabled.
		1	Break condition is transmitted to the receiving UART. A break condition is a condition where the UART_TX signal is forced to the spacing (cleared) state.
5	SP		Stick parity. The SP bit works in conjunction with the EPS and PEN bits. The relationship between the SP, EPS, and PEN bits is summarized in Table 7-15.
		0	Stick parity is disabled.
		1	Stick parity is enabled. • When odd parity is selected (EPS = 0), the PARITY bit is transmitted and checked as set. • When even parity is selected (EPS = 1), the PARITY bit is transmitted and checked as cleared.
4	EPS		Even parity select. Selects the parity when parity is enabled (PEN = 1). The EPS bit works in conjunction with the SP and PEN bits. The relationship between the SP, EPS, and PEN bits is summarized in Table 7-15.
		0	Odd parity is selected (an odd number of logic 1s is transmitted or checked in the data and PARITY bits).
		1	Even parity is selected (an even number of logic 1s is transmitted or checked in the data and PARITY bits).
3	PEN		Parity enable. The PEN bit works in conjunction with the SP and EPS bits. The relationship between the SP, EPS, and PEN bits is summarized in Table 7-15.
		0	No PARITY bit is transmitted or checked.
		1	Parity bit is generated in transmitted data and is checked in received data between the last data word bit and the first STOP bit.

Table 7-14. Line Control Register (LCR) Field Descriptions (continued)

Bit	Field	Value	Description
2	STB		Number of STOP bits generated. STB specifies 1, 1.5, or 2 STOP bits in each transmitted character. When STB = 1, the WLS bit determines the number of STOP bits. The receiver clocks only the first STOP bit, regardless of the number of STOP bits selected. The number of STOP bits generated is summarized in Table 7-16.
		0	1 STOP bit is generated.
		1	WLS bit determines the number of STOP bits: • When WLS = 0, 1.5 STOP bits are generated. • When WLS = 1h, 2h, or 3h, 2 STOP bits are generated.
1-0	WLS	0-3h	Word length select. Number of bits in each transmitted or received serial character. When STB = 1, the WLS bit determines the number of STOP bits.
		0	5 bits
		1h	6 bits
		2h	7 bits
		3h	8 bits

Table 7-15. Relationship Between ST, EPS, and PEN Bits in LCR

ST Bit	EPS Bit	PEN Bit	Parity Option
x	x	0	Parity disabled: No PARITY bit is transmitted or checked.
0	0	1	Odd parity selected: Odd number of logic 1s.
0	1	1	Even parity selected: Even number of logic 1s.
1	0	1	Stick parity selected with PARITY bit transmitted and checked as set.
1	1	1	Stick parity selected with PARITY bit transmitted and checked as cleared.

Table 7-16. Number of STOP Bits Generated

STB Bit	WLS Bits	Word Length Selected With WLS Bits	Number of STOP Bits Generated	Baud Clock (BCLK) Cycles
0	x	Any word length	1	16
1	0h	5 bits	1.5	24
1	1h	6 bits	2	32
1	2h	7 bits	2	32
1	3h	8 bits	2	32

7.3.7 Modem Control Register (MCR)

The modem control register (MCR) is shown in Figure 7-15 and described in Table 7-17. The modem control register provides the ability to enable/disable the autoflow functions, and enable/disable the loopback function for diagnostic purposes.

Figure 7-15. Modem Control Register (MCR)

15 6	5	4	3 2	1	0
Reserved	AFE	LOOP	Reserved	RTS	Rsvd
R-0	R/W-0	R/W-0	R-0	R/W-0	R-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 7-17. Modem Control Register (MCR) Field Descriptions

Bit	Field	Value	Description
15-6	Reserved	0	Reserved
5	AFE		Autoflow control enable. Autoflow control allows the RTS and CTS signals to provide handshaking between UARTs during data transfer. When AFE = 1, the RTS bit determines the autoflow control enabled.
		0	Autoflow control is disabled.
		1	Autoflow control is enabled: • When RTS = 0, CTS is only enabled. • When RTS = 1, RTS and CTS are enabled.
4	LOOP		Loop back mode enable. LOOP is used for the diagnostic testing using the loop back feature.
		0	Loop back mode is disabled.
		1	Loop back mode is enabled. When LOOP is set, the following occur: • The UART_TX signal is set high. • The UART_RX pin is disconnected. • The output of the transmitter shift register (TSR) is lopped back in to the receiver shift register (RSR) input.
3-2	Reserved	0	Reserved
1	RTS		RTS control. When AFE = 1, the RTS bit determines the autoflow control enabled.
		0	RTS is disabled, CTS is only enabled.
		1	RTS and CTS are enabled.
0	Reserved	0	Reserved.

7.3.8 Line Status Register (LSR)

The line status register (LSR) is shown in Figure 7-16 and described in Table 7-18. LSR provides information to the CPU concerning the status of data transfers. LSR is intended for read operations only; do not write to this register. Bits 1 through 4 record the error conditions that produce a receiver line status interrupt.

Figure 7-16. Line Status Register (LSR)

15	8	7	6	5	4	3	2	1	0
Reserved		RXFIFOE	TEMT	THRE	BI	FE	PE	OE	DR
R-0		R-0	R-1	R-1	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 7-18. Line Status Register (LSR) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved.
7	RXFIFOE		Receiver FIFO error.
			In non-FIFO mode:
		0	There has been no error, or RXFIFOE was cleared because the CPU read the erroneous character from the receiver buffer register (RBR).
		1	There is a parity error, framing error, or break indicator in the receiver buffer register (RBR).
			In FIFO mode:
		0	There has been no error, or RXFIFOE was cleared because the CPU read the erroneous character from the receiver FIFO and there are no more errors in the receiver FIFO.
		1	At least one parity error, framing error, or break indicator in the receiver FIFO.
6	TEMT		Transmitter empty (TEMT) indicator.
			In non-FIFO mode:
		0	Either the transmitter holding register (THR) or the transmitter shift register (TSR) contains a data character.
		1	Both the transmitter holding register (THR) and the transmitter shift register (TSR) are empty.
			In FIFO mode:
		0	Either the transmitter FIFO or the transmitter shift register (TSR) contains a data character.
		1	Both the transmitter FIFO and the transmitter shift register (TSR) are empty.
5	THRE		Transmitter holding register empty (THRE) indicator. If the THRE bit is set and the corresponding interrupt enable bit is set (ETBEI = 1 in IER), an interrupt request is generated.
			In non-FIFO mode:
		0	Transmitter holding register (THR) is not empty. THR has been loaded by the CPU.
		1	Transmitter holding register (THR) is empty (ready to accept a new character). The content of THR has been transferred to the transmitter shift register (TSR).
			In FIFO mode:
		0	Transmitter FIFO is not empty. At least one character has been written to the transmitter FIFO. You can write to the transmitter FIFO if it is not full.
		1	Transmitter FIFO is empty. The last character in the FIFO has been transferred to the transmitter shift register (TSR).

Table 7-18. Line Status Register (LSR) Field Descriptions (continued)

Bit	Field	Value	Description
4	BI		Break indicator. The BI bit is set whenever the receive data input (RX) was held low for longer than a full-word transmission time. A full-word transmission time is defined as the total time to transmit the START, data, PARITY, and STOP bits. If the BI bit is set and the corresponding interrupt enable bit is set (ELSI = 1 in IER), an interrupt request is generated.
			In non-FIFO mode:
		0	No break has been detected, or the BI bit was cleared because the CPU read the erroneous character from the receiver buffer register (RBR).
		1	A break has been detected with the character in the receiver buffer register (RBR).
			In FIFO mode:
		0	No break has been detected, or the BI bit was cleared because the CPU read the erroneous character from the receiver FIFO and the next character to be read from the FIFO has no break indicator.
		1	A break has been detected with the character at the top of the receiver FIFO.
3	FE		Framing error (FE) indicator. A framing error occurs when the received character does not have a valid STOP bit. In response to a framing error, the UART sets the FE bit and waits until the signal on the RX pin goes high. Once the RX signal goes high, the receiver is ready to detect a new START bit and receive new data. If the FE bit is set and the corresponding interrupt enable bit is set (ELSI = 1 in IER), an interrupt request is generated.
			In non-FIFO mode:
		0	No framing error has been detected, or the FE bit was cleared because the CPU read the erroneous data from the receiver buffer register (RBR).
		1	A framing error has been detected with the character in the receiver buffer register (RBR).
			In FIFO mode:
		0	No framing error has been detected, or the FE bit was cleared because the CPU read the erroneous data from the receiver FIFO and the next character to be read from the FIFO has no framing error.
		1	A framing error has been detected with the character at the top of the receiver FIFO.
2	PE		Parity error (PE) indicator. A parity error occurs when the parity of the received character does not match the parity selected with the EPS bit in the line control register (LCR). If the PE bit is set and the corresponding interrupt enable bit is set (ELSI = 1 in IER), an interrupt request is generated.
			In non-FIFO mode:
		0	No parity error has been detected, or the PE bit was cleared because the CPU read the erroneous data from the receiver buffer register (RBR).
		1	A parity error has been detected with the character in the receiver buffer register (RBR).
			In FIFO mode:
		0	No parity error has been detected, or the PE bit was cleared because the CPU read the erroneous data from the receiver FIFO and the next character to be read from the FIFO has no parity error.
		1	A parity error has been detected with the character at the top of the receiver FIFO.
1	OE		Overrun error (OE) indicator. An overrun error in the non-FIFO mode is different from an overrun error in the FIFO mode. If the OE bit is set and the corresponding interrupt enable bit is set (ELSI = 1 in IER), an interrupt request is generated.
			In non-FIFO mode:
		0	No overrun error has been detected, or the OE bit was cleared because the CPU read the content of the line status register (LSR).
		1	Overrun error has been detected. Before the character in the receiver buffer register (RBR) could be read, it was overwritten by the next character arriving in RBR.
			In FIFO mode:
		0	No overrun error has been detected, or the OE bit was cleared because the CPU read the content of the line status register (LSR).
		1	Overrun error has been detected. If data continues to fill the FIFO beyond the trigger level, an overrun error occurs only after the FIFO is full and the next character has been completely received in the shift register. An overrun error is indicated to the CPU as soon as it happens. The new character overwrites the character in the shift register, but it is not transferred to the FIFO.

Table 7-18. Line Status Register (LSR) Field Descriptions (continued)

Bit	Field	Value	Description
0	DR		Data-ready (DR) indicator for the receiver. If the DR bit is set and the corresponding interrupt enable bit is set (ERBI = 1 in IER), an interrupt request is generated.
			In non-FIFO mode:
		0	Data is not ready, or the DR bit was cleared because the character was read from the receiver buffer register (RBR).
		1	Data is ready. A complete incoming character has been received and transferred into the receiver buffer register (RBR).
			In FIFO mode:
		0	Data is not ready, or the DR bit was cleared because all of the characters in the receiver FIFO have been read.
		1	Data is ready. There is at least one unread character in the receiver FIFO. If the FIFO is empty, the DR bit is set as soon as a complete incoming character has been received and transferred into the FIFO. The DR bit remains set until the FIFO is empty again.

7.3.9 Scratch Register

The scratch register (SCR) is intended for programmer’s use as a scratch pad in the sense that it temporarily holds programmer’s data without affecting UART operation. The SCR is described is shown in Figure 7-17 and described in Table 7-19.

Figure 7-17. Scratch Register

15 8	7 0
Reserved	DATA
R-0	R/W-0

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Table 7-19. Scratch Register (SCR) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved.
7-0	DATA	0-FFh	Scratch pad data.

7.3.10 Divisor Latches (DLL and DLH)

Two 8-bit register fields (DLL and DLH), called divisor latches, store the 16-bit divisor for generation of the baud clock in the baud generator. The latches are in DLH and DLL. DLH holds the most-significant bits of the divisor, and DLL holds the least-significant bits of the divisor. These divisor latches must be loaded during initialization of the UART in order to ensure desired operation of the baud generator. Writing to the divisor latches results in two wait states being inserted during the write access while the baud generator is loaded with the new value.

Access considerations:

- RBR, THR, and DLL share one address. When DLAB = 1 in LCR, all accesses at the shared address are accesses to DLL. When DLAB = 0, reading from the shared address gives the content of RBR, and writing to the shared address modifies THR.
- IER and DLH share one address. When DLAB = 1 in LCR, accesses to the shared address read or modify to DLH. When DLAB = 0, all accesses at the shared address read or modify IER.

DLL and DLH also have dedicated addresses. If you use the dedicated addresses, you can keep the DLAB bit cleared, so that RBR, THR, and IER are always selected at the shared addresses.

The divisor LSB latch (DLL) is shown in Figure 7-18 and described in Table 7-20. The divisor MSB latch (DLH) is shown in Figure 7-19 and described in Table 7-21.

Figure 7-18. Divisor LSB Latch (DLL)

15 8	7 0
Reserved	DLL
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 7-20. Divisor LSB Latch (DLL) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved.
7-0	DLL	0-FFh	The 8 least-significant bits (LSBs) of the 16-bit divisor for generation of the baud clock in the baud rate generator.

Figure 7-19. Divisor MSB Latch (DLH)

15 8	7 0
Reserved	DLH
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 7-21. Divisor MSB Latch (DLH) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved.
7-0	DLH	0-FFh	The 8 most-significant bits (MSBs) of the 16-bit divisor for generation of the baud clock in the baud rate generator.

7.3.11 Power and Emulation Management Register (PWREMU_MGMT)

The power and emulation management register (PWREMU_MGMT) is shown in Figure 7-20 and described in Table 7-22.

Figure 7-20. Power and Emulation Management Register (PWREMU_MGMT)

15	14	13	12 – 1	0
Rsvd	UTRST	URRST	Reserved	FREE
R/W-0	R/W-0	R/W-0	R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 7-22. Power and Emulation Management Register (PWREMU_MGMT) Field Descriptions

Bit	Field	Value	Description
15	Reserved	0	Reserved. This bit must always be written with a 0.
14	UTRST		UART transmitter reset. Resets and enables the transmitter.
		0	Transmitter is disabled and in reset state.
		1	Transmitter is enabled.
13	URRST		UART receiver reset. Resets and enables the receiver.
		0	Receiver is disabled and in reset state.
		1	Receiver is enabled.
12-1	Reserved	1	Reserved.
0	FREE		Free-running enable mode bit. This bit determines the emulation mode functionality of the UART. When halted, the UART can handle register read/write requests, but does not generate any transmission/reception, interrupts or events.
		0	If a transmission is not in progress, the UART halts immediately. If a transmission is in progress, the UART halts after completion of the one-word transmission.
		1	Free-running mode is enabled; UART continues to run normally.

Chapter 8

SPRUH87C–August 2011–Revised March 2012

Serial Peripheral Interface (SPI)

This chapter describes the features and operations of the serial peripheral interface (SPI).

Topic **Page**

8.1 Introduction

This document describes the serial peripheral interface (SPI) in the digital signal processor (DSP).

8.1.1 Purpose of the Peripheral

The SPI is a high-speed synchronous serial input/output port that allows a serial bit stream of programmed length (1 to 32 bits) to be shifted into and out of the device at a programmed bit-transfer rate. The SPI supports multi-chip operation of up to four SPI slave devices. The SPI can operate as a master device only.

The SPI is normally used for communication between the DSP and external peripherals. Typical applications include an interface to external I/O or peripheral expansion via devices such as shift registers, display drivers, SPI EEPROMs, and analog-to-digital converters.

8.1.2 Features

The SPI has the following features:

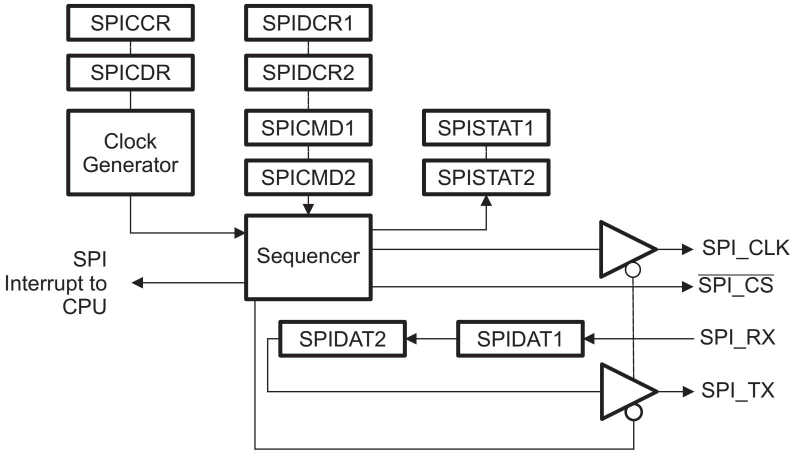
- Programmable divider for serial data clock generation.
- Four pin interface (SPI_CLK, SPI_CS*n*, SPI_TX and SPI_RX).
- Programmable data length (1 to 32 bits).
- 4 external chip select signals.
- Programmable transfer or frame size (1 to 4096 characters).
- Optional interrupt generation on character completion or frame completion.
- Programmable SPI_CS*n* to SPI_TX delay insertion from 0 to 3 SPI_CLK cycles.
- Programmable signal polarities.
- Programmable active clock edge.
- Internal loopback mode for testing.

The SPI can only operate in master mode only, slave mode is not supported.

8.1.3 Functional Block Diagram

Figure 8-1 illustrates the main components of the SPI.

Figure 8-1. Serial Peripheral Interface (SPI) Block Diagram

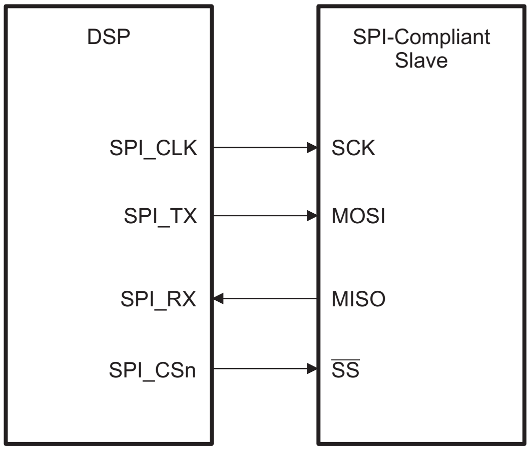


8.1.4 Supported Use Case Statement

The SPI is intended for communication between the DSP and up to four SPI-complaint slave devices. Typical applications include an interface to external I/O or peripheral expansion via devices such as shift registers, display drivers, SPI EEPROMs, and analog-to-digital converters. The programmable configuration capability of the SPI allows it to interface to a variety of SPI format devices without the need for glue logic.

A typical SPI interface with a single slave device is shown in Figure 8-2. The DSP controls the flow of communication by providing shift-clock (SPI_CLK) and slave-select signals (SPI_CS*n*).

Figure 8-2. Typical SPI Interface



8.1.5 Industry Standard(s) Compliance Statement

The SPI does not conform to a specific industry standard.

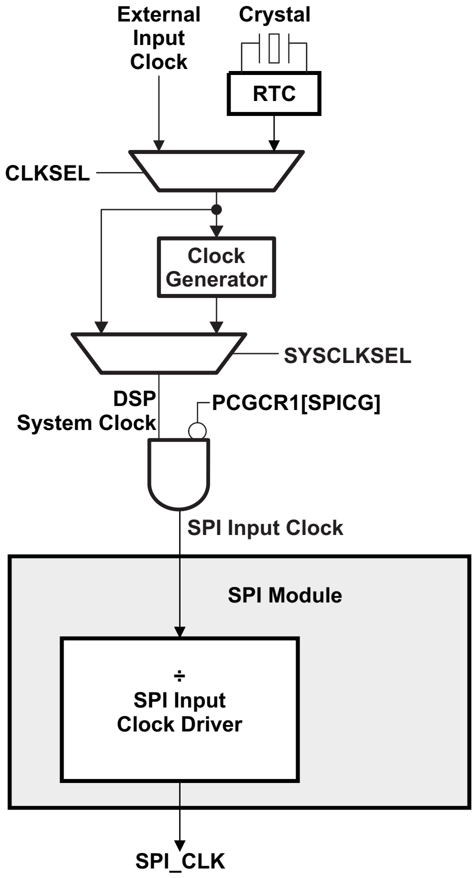
8.2 Serial Peripheral Interface Architecture

8.2.1 Clock Control

As shown in Figure 8-3, the clock generator receives either the real-time clock (RTC) or a signal from an external clock source and produces the DSP system clock. This internal clock is used by the DSP CPU and peripherals. A programmable clock divider in the SPI module divides down the SPI input clock to produce the SPI interface output clock (SPI_CLK). For proper device operation, the frequency of the SPI input clock must be at least four times greater than the frequency of SPI_CLK.

The DSP device includes logic which can be used to gate the clock to its on-chip peripherals, including the SPI. The input clock to the SPI can be enabled and disabled through the peripheral clock gating configuration register 1 (PCGCR1).

Figure 8-3. Clocking Diagram for the SPI



8.2.2 Signal Descriptions

Table 8-1 shows the SPI pins used to interface to external devices. A typical SPI interface with a single slave device is shown in Figure 8-2.

Table 8-1. Serial Peripheral Interface (SPI) Pins

Pin	Type	Function
SPI_CLK	Output	Serial clock output pin (also referred to as SCK)
SPI_TX	Output	Serial data output pin (also referred to as Master Output - Slave Input, or MOSI)
SPI_RX	Input	Serial data input pin (also referred to Master Input - Slave Output, or MISO)
SPI_CS0	Output	Slave 0 chip select pin (also referred to as Slave Select, or $\overline{SS}$)
SPI_CS1	Output	Slave 1 chip select pin (also referred to as Slave Select, or $\overline{SS}$)
SPI_CS2	Output	Slave 2 chip select pin (also referred to as Slave Select, or $\overline{SS}$)
SPI_CS3	Output	Slave 3 chip select pin (also referred to as Slave Select, or $\overline{SS}$)

8.2.3 Units of Data: Characters and Frames

This documentation describes SPI module communication using two terms: characters and frames. Characters are the smallest unit of data that can be transferred by the SPI module. During a transfer, the SPI generates enough clock cycles to send a character of data. The length of the character is specified by the CLEN bits of SPICMD2. The character length can be from 1 to 32 bits and can be of different size each time the SPI initiates a character transfer.

The total number of characters transmitted by the SPI module is referred to as a frame. At the beginning of a frame, the SPI module will assert the chip select pin specified by the chip select bits (CSNUM) of SPICMD2 and transfer a character of data. The SPI module will keep the chip select pin asserted until all the characters in the frame have been transferred. The frame length bits (FLEN) of SPICMD1 define the total number of characters in a frame. A frame can have up to 4096 characters. Please note that you should complete a frame transfer before using a different chip select pin.

The character count bits (CCNT) of SPISTAT2 keep track of the total number of times a character has been transferred by the SPI module. The character count is not affected by the size of the character. For example, a transmission of an 8-bit character followed by a 16-bit character increments CCNT by two.

The frame complete bit (FC) of SPISTAT1 is set after all the requested characters in a frame have been transferred. This bit is reset when SPISTAT1 is read or when a new SPI transfer is initiated via a write of a read or write command to CMD in SPICMD2.

8.2.4 Chip Select Control

The SPI module initiates a slave access by asserting one of its four chip select pins. The chip select pin remains asserted until an entire frame of data has been transferred. You can specify which chip select pin is activated through the chip select bits (CSNUM) of SPICMD2. Please note that writing to SPICMD2 immediately initiates a slave access; therefore you should only write to CSNUM when you are ready to initiate a slave access. Also, after initiating an access to a particular slave, you must complete the entire frame transfer to that slave before activating a different chip select pin.

The polarity of each of the four chip select pins can be configured through the CSP*n* bits of SPIDCR1 and SPIDCR2. Setting CSP*n* = 0 configures the corresponding SPI_CS*n* pin as active low while setting CSP*n* = 1 configures the SPI_CS*n* pin as active high.

8.2.5 Clock Polarity and Phase

Before communication between the SPI module and a slave can begin, you must specify the clock polarity and clock phase to be used. Together, the clock polarity and clock phase specify on which clock edge data is shifted in and out as well as the default state of the clock signal.

The configuration of the clock polarity and clock phase is referred to as the SPI mode. There are a total of four SPI modes (see Table 8-2), all which are supported by the SPI module. The clock polarity and clock phase must be configured correctly for successful communication between the SPI module and slave.

The clock polarity and phase can be specified through the CKP*n* and CKPH*n* bits of the SPI device configuration register (SPIDC). You can program a different clock polarity and phase for each slave.

Table 8-2. Definition of SPI Modes

SPI Mode	Clock Polarity	Clock Phase
0	Active low (base value of clock is low)	Data shifted out on the falling edge, input captured on the rising edge.
1	Active low (base value of clock is low)	Data shifted out on the rising edge, input captured on the falling edge.
2	Active high (base value of clock is high)	Data shifted out on the rising edge, input captured on the falling edge.
3	Active high (base value of clock is high)	Data shifted out on the falling edge, input captured on the rising edge.

The timing diagrams for the four possible SPI modes are shown in Figure 8-4 through Figure 8-7. Please note the following about these figures:

- Although the timing diagrams show an 8-bit character transfer, the character length can be set to 1 through 32 bits. The character length is selected with the CLEN bits SPICMD2.
- The number of characters transferred during one slave access is specified through the FLEN bits of SPICMD1. The figures show the case of FLEN = 0 (1 character).
- The polarity of the chip select pins (SPI_CS*n*) can be configured through the CSP*n* bits of SPIDCR1 and SPIDCR2. The figures show a chip select polarity of active low.
- The SPI module automatically delays the first clock edge with respect to the activation of the SPI_CS*n* pin by half a SPI_CLK cycle plus a system clock cycle. Additional clock delay cycles can be added using the data delay bits (DD*n*) of SPIDCR1 and SPIDCR2. The figures below show the case of DD*n* = 0 (zero data delay) and CLKDV is odd.

Figure 8-4. SPI Mode 0 Transfer (CKP*n* = 0, CKPH*n* = 0)

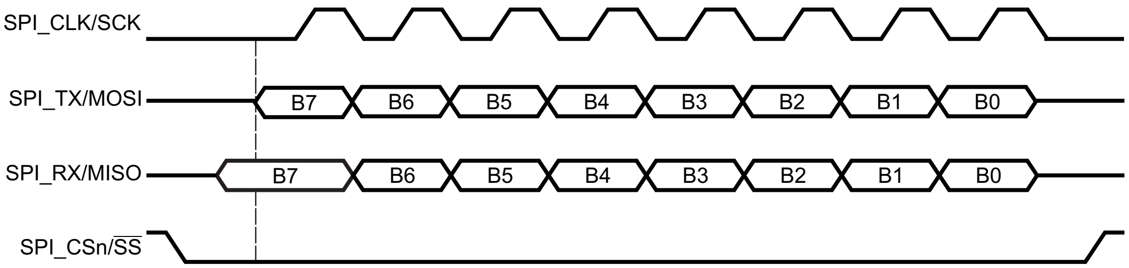


Figure 8-5. SPI Mode 1 Transfer (CKP*n* = 0, CKPH*n* = 1)

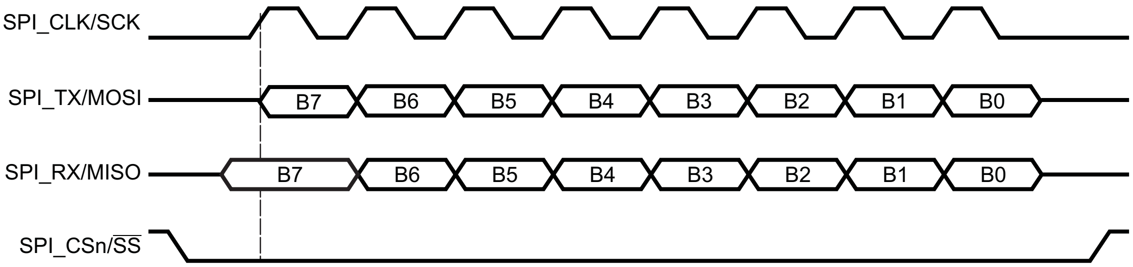


Figure 8-6. SPI Mode 2 Transfer (CKP*n* = 1, CKPH*n* = 0)

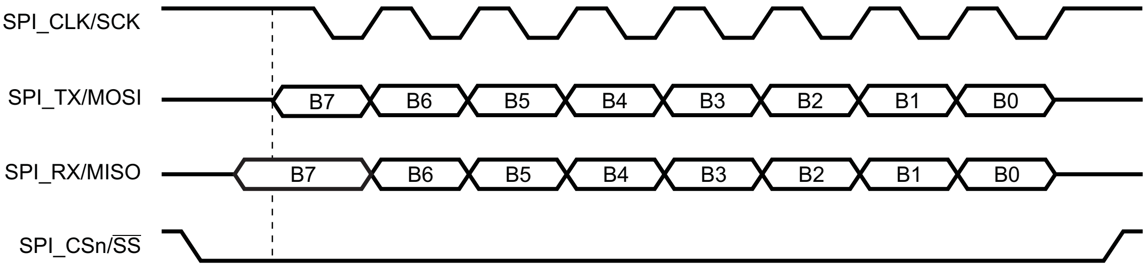
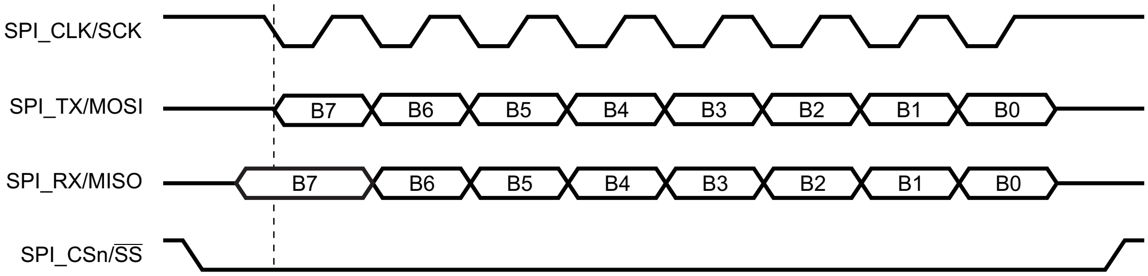


Figure 8-7. SPI Mode 3 Transfer (CKP*n* = 1, CKPH*n* = 1)

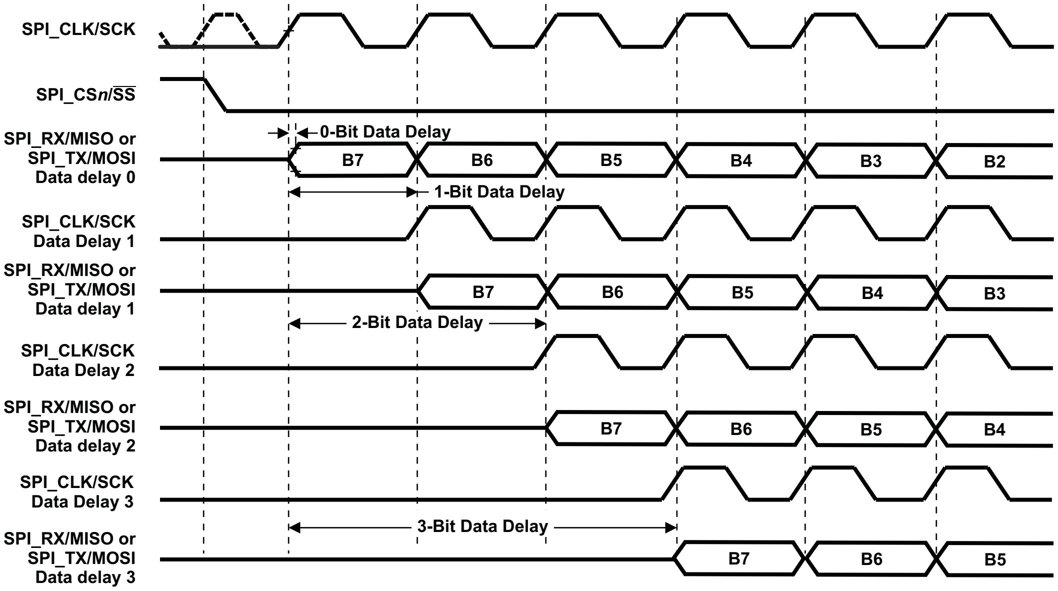


8.2.6 Data Delay

As described in the previous section, the SPI module automatically delays the first clock edge with respect to the activation of the SPI_CS*n* pin by half a SPI_CLK cycle plus a system clock cycle. You can program the SPI module to insert additional clock delay cycles using the data delay bits (DD*n*) of SPIDCR1 and SPIDCR2 to determine the number of clock delay cycles to insert. The data delay can be specified from zero to three clock cycles (DD*n* = 00b - 11b). Figure 8-8 below shows the effect of the data delay bits. Please note the following about Figure 8-8:

- The data transferred is an 8-bit character with bits labeled B7, B6, B5, and so on. The character length can be set to 1 through 32 bits through the CLEN bits of the SPICMD2).
- The polarity of the chip select pins (SPI_CS*n*) can be configured through the CSP*n* bits of SPIDCR1 and SPIDCR2. The figures show a chip select polarity of active low.
- The clock polarity and clock phase can be configured through the CKP*n* and CKPH*n* bits of SPIDCR1 and SPIDCR2. Figure 8-8 shows an example CKP*n* = 0 and CKPH*n* = 1 (SPI Mode 1).
- The first clock edge is automatically delayed by half a SPI_CLK cycle plus a system clock cycle by the SPI module.

Figure 8-8. Range of Programmable Data Delay



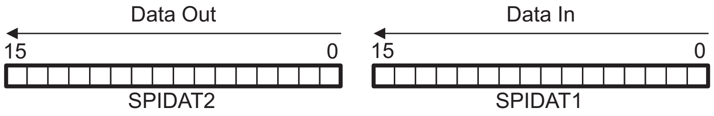
8.2.7 Data Input and Output

The data registers (SPIDAT1 and SPIDAT2) hold the data that is either shifted in or shifted out during a slave access. The SPIDAT1 and SPIDAT2 registers are treated as a single 32-bit shift register. Data received by the SPI is shifted into the least-significant bit of SPIDAT1 and the contents of both registers are shifted to the left. Similarly, data transferred by the SPI is shifted out of the most-significant bit of SPIDAT2 and the contents of both registers are shifted to the left. This process is illustrated in Figure 8-9.

The CLEN bits of SPICMD2 determine the length of the character. The character length can be set to any value between 1 and 32 bits.

The data registers are not cleared between reads or writes, and old may exist in the registers. Therefore, you must : (1) clear the data registers prior to initiating a read; and (2) mask off any unneeded data upon completing the read.

Figure 8-9. Data Shift Process



8.2.8 Loopback Mode

The SPI includes a loopback mode which can be used for testing purposes. In the loopback mode, the SPI_TX and SPI_RX are internally connected to each other. All SPI modes are usable when loopback mode is enabled.

8.2.9 Monitoring SPI Activity

The status registers (SPISTAT1 and SPISTAT2) contain indicators that allow you to monitor the progression of a frame transfer. These bits are described in Table 8-3. Additionally, the SPI module can be configured to generate interrupts after a frame or character has been transferred. These interrupts are described in Section 8.2.13.

Table 8-3. SPI Module Status Bits

Status Bit	Register	Description
FC	SPISTAT1	The frame complete bit. This bit is set after all the requested characters in a frame have been transferred. This bit is reset when SPISTAT1 is read or when a new SPI transfer is initiated via a write of a read or write command to CMD in SPICMD2.
CC	SPISTAT1	Character complete bit. This bit is set after each transfer is completed. This bit is reset when SPISTAT1 is read or when a new SPI transfer is initiated via a write of a read or write command to CMD in SPICMD2.
CCNT	SPISTAT2	Character count bits. These bits keep track of the total number of times a character has been transferred by the SPI module. The CCNT bits are not affected by the size of the character. For example, a transmission of an 8-bit character followed by a 16-bit character increments CCNT by two. These bits are reset upon completion of a frame. These bits should only be read after determining CC = 1.
BUSY	SPISTAT1	Busy bit. This bit is set during an active character transfer. Between characters this bit will be cleared to signal that the data registers can be accessed.

8.2.10 Slave Access

After you have initialized the SPI following the steps outlined in Section 8.2.12, you can follow these steps to initiate a slave access.

1. Specify the total number of characters you intend to transfer by setting the FLEN bits of SPICMD1. You must finish transferring this number of characters before initiating transfers with a different slave.
2. For writes, load the output data value into SPIDAT1 and SPIDAT2. For reads, clear SPIDAT1 and SPIDAT2. The SPI module will shift data out starting with the most-significant bit of SPIDAT2. Similarly, the SPI will shift data in starting at the least-significant bit of SPIDAT1. The amount of bits the SPI module shifts in or out is determined by the CLEN bits of SPICMD2.
3. Write to the command register (SPICMD2) to start the SPI access. The value you write SPICMD2 must set CSNUM, CLEN, and CMD to the chip select, character length, and command to be used.
4. Poll SPISTAT1 until CC = 1 or wait for the character complete interrupt (CIRQ).
5. Poll SPISTAT1 until BUSY = 0. This ensures that a character is not being transferred.
6. For reads, the data is loaded into SPIDAT1 and SPIDAT2. You must mask off any invalid bits when reading SPIDAT1 and SPIDAT2. The number of valid bits is determined by CLEN in SPICMD2.
7. Repeat steps 2 though 5 until all the characters have been transferred.

Please note the following points:

- The chip select pin specified in SPICMD2 will be activated as soon as the first character transfer is initiated. The chip select pin will remain activated until all the characters specified by FLEN have been transferred.
- SPIDAT1 and SPIDAT2 are treated as a 32-bit shift register where SPIDAT2 holds the most-significant word and SPIDAT1 holds the least-significant word. During writes, data is shifted out starting with the most-significant bit of SPIDAT2. For reads, data is shifted in starting at the least-significant bit of SPIDAT1.
- It is important for you to always load SPIDAT1 and SPIDAT2 with the values you intend to shift out before initiating an SPI write. It is equally important to always mask off invalid bits after reading values from SPIDAT1 and SPIDAT2 after every SPI read.
- You must finish transferring the number of characters specified by FLEN in SPICMD1 before initiating transfers with a different slave.
- SPI_CLK will only be activated while data is being shifted out.

Figure 8-10 illustrates the steps described above.

Figure 8-10. Flow Diagram for SPI Read or Write

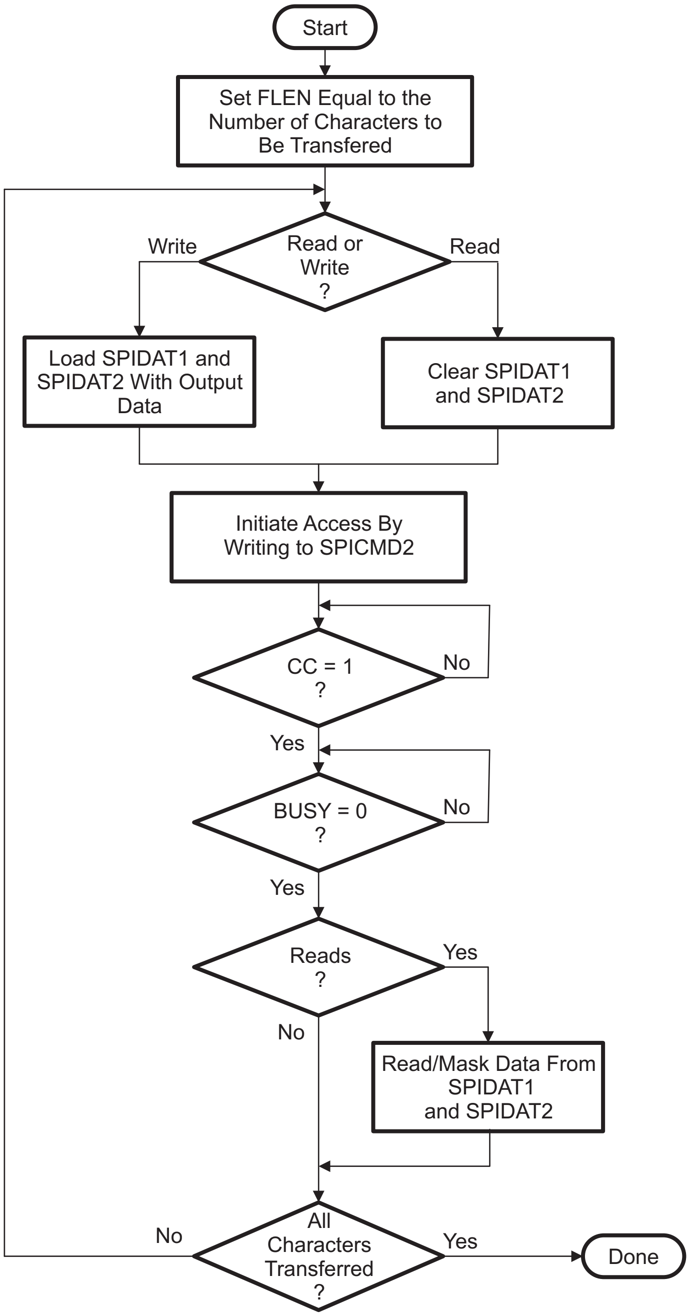
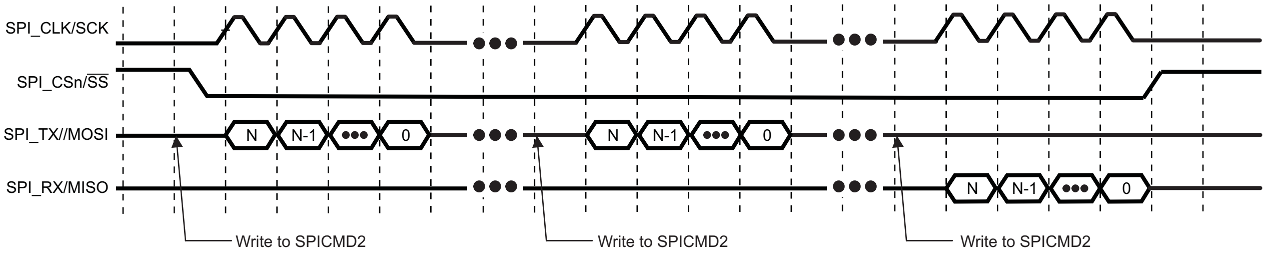

Start
Set FLEN Equal to the Number of Characters to Be Transfered
Read or Write ?
Write
Read
Load SPIDAT1 and SPIDAT2 With Output Data
Clear SPIDAT1 and SPIDAT2
Initiate Access By Writing to SPICMD2
CC = 1 ?
No
Yes
BUSY = 0 ?
No
Yes
Reads ?
Yes
No
Read/Mask Data From SPIDAT1 and SPIDAT2
All Characters Transferred ?
No
Yes
Done

Figure 8-11 illustrates the activity at the pins of the SPI module.

Figure 8-11. SPI Access



8.2.11 Reset Considerations

The SPI module has two reset sources: software reset and hardware reset.

8.2.11.1 Software Reset Considerations

The SPI module can be reset by software through the RST bit in the clock control register (SPICCR) or through the PG4_RST bit in the peripheral reset control register (PRCR).

When RST is set to 1, the SPI module is reset.

The entire SPI module may also be reset through the PG4_RST bit of PRCR. When this reset is asserted, all SPI registers are set to their default values. The SPI remains inactive until programmed by software. Note that from the SPI perspective, this reset appears as a hardware reset.

NOTE: The PG4_RST bit of PRCR resets other peripherals besides the SPI. For more details on this bit and register, refer to the device-specific data manual.

8.2.11.2 Hardware Reset Considerations

When the SPI is reset due to a device reset, all the SPI registers are set to their default values. The SPI module remains inactive until programmed by software.

8.2.12 Initialization

The following initialization procedure describes the basic setup of the SPI module. Please note that the exact configuration will depend on the characteristics of the slave device. For specific examples, please refer to Section 8.3.

1. Ensure the SPI is out reset by setting SPI_RST = 0 in the peripheral reset control register (PRCR). See the device-specific data manual for more information on PRCR.
2. Enable the SPI input clock by setting SPICG to 0 in the peripheral clock gating configuration register (PCGCR1). See the device-specific data manual for more information on PCGCR1.
3. Set CLKEN = 0 to disable SPI_CLK. The CLKEN bit is part of SPICCR.
4. Set CLKDV in SPICCR to provide the appropriate SPI_CLK frequency. Note that for proper device operation, CLKDV must be greater than or equal to 3. Also, note that the clock frequency selected in this step applies to all slave devices. If a different frequency is required for each slave, you must reprogram the clock register each time you access a different slave.
5. Set CLKEN = 1 to enable SPI_CLK.
6. Program the device configuration registers (SPIDCR1 and SPIDCR2) with the required clock phase, clock polarity, chip select pin polarity, and data delay settings for each slave.
7. Program the SPI module to generate an interrupt after a frame of characters has been transferred (FIRQ = 1) or after a single character has been transferred (CIRQ = 1), If you want to use interrupts. The FIRQ and CIRQ bits are located in SPICMD1.

Enable the SPI pins through the external bus selection register (EBSR). EBSR controls the pin multiplexing options of the device. Refer to the device-specific data manual for more information on EBSR.

8.2.13 Interrupt Support

8.2.13.1 Interrupt Events and Requests

The SPI module is capable of interrupting the CPU after every character transfer and after every frame transfer. The SPI interrupts are enabled through the FIRQ and CIRQ bits of the command register 1 (SPICMD1).

8.2.13.2 Interrupt Multiplexing

The SPI module generates a single interrupt to the CPU. The SPI interrupt is not multiplexed with any other interrupt source.

8.2.14 DMA Event Support

The SPI module does not generate any DMA events; it must be fully serviced by the CPU. Furthermore, none of the device DMA controllers have access to the SPI memory-mapped registers.

8.2.15 Power Management

The SPI module can be clock-gated to conserve power during periods of no activity. The SPI input clock can be turned off by using the peripheral clock gating configuration register (PCGCR). For detailed information on PCGCR, see the device-specific data manual.

8.2.16 Emulation Considerations

The SPI module is not interrupted by emulation events such as an emulation breakpoint.

8.3 Interfacing the SPI to an SPI EEPROM

The SPI module can be used to communicate with an SPI EEPROM. This section gives an example of interfacing the SPI a 256K-bit SPI EEPROM from Catalyst Semiconductor (CAT25C256). Software sequences for common tasks such as data block reads and writes are also included.

8.3.1 Operational Description

The SPI module is totally controlled by the CPU. The SPI initiates transfers with SPI devices when the CPU writes to the SPICMD2 register. The SPI can interrupt the CPU when it has completed a character transfer and when it has completed a frame transfer. Alternatively, the CPU can poll the SPI status registers. The SPI cannot generate DMA events; therefore, it is the task of the CPU to move data to and from the SPI data registers.

Communication with SPI EEPROM is carried out completely by the SPI module. The SPI module drives the chip select pin of the memory device and allows the clocks to shift data in and out.

The SPI clock is derived from the chip system clock. The clock divider bits (CLKDV) of SPICCR are used to configure the frequency of the SPI clock (SPI_CLK) as follows:

SPI_CLK frequency = SPI module input clock / (CLKDV + 1)

The number of characters sent or received by the SPI module is specified through the FLEN bits of SPICMD1. During communication with a SPI device, the SPI module will assert the chip select pin until all the characters specified through FLEN have been transferred.

8.3.2 Hardware Interface

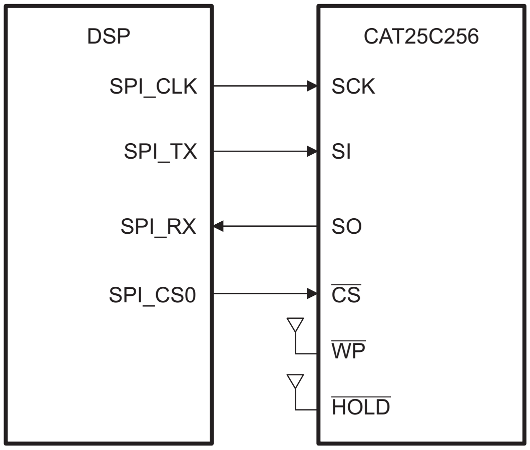
Figure 8-12 shows the required hardware interface. Please note the following:

- The $\overline{\text{WP}}$ pin is shown as tied high (write-protect feature disabled); however, you can choose to tie the

WP pin high or low depending on whether or not you want to use the write-protect feature of the CAT25C256 device.

- The HOLD pin is not used in this example and is pulled high.
- The example described in this section assumes chip-select 0 is used; however, you can use any other chip select.

Figure 8-12. Hardware Interface



8.3.3 SW Configuration

The following sections describe the software sequence for common tasks such as data block writes and data block reads.

8.3.3.1 Basic Initialization

The following procedure describes the basic steps required to setup the SPI module for communication with the CAT25C256 device.

1. Ensure the SPI is out reset by setting SPI_RST=0 in the peripheral reset control register (PRCR). See the device-specific data manual for more information on PRCR.
2. Enable the SPI input clock by setting SPICG to 0 in the peripheral clock gating configuration register (PCGCR1). See the device-specific data manual for more information on PCGCR1.
3. Set CLKEN = 0 to disable SPI_CLK. The CLKEN bit is part of SPICCR .
4. Set CLKDV in SPICCR to provide the appropriate SPI_CLK frequency. Note that for proper device operation, CLKDV must be greater than or equal to 3 .
5. Set CLKEN = 1 to enable SPI_CLK.
6. The CAT25C256 device latches input data on the rising edge of the clock and shifts out data on the falling edge of the clock. Therefore, the SPI must be programmed to shift out data on the falling edge of the clock and shift in data on the rising edge of the clock. According to Table 8-2, this corresponds to SPI mode 0. To enable SPI mode 0, set CKP0 = 0 and CKPH0 = 0 in SPIDCR1. Also, the data delay field should be programmed such that the chip select setup requirement of the CAT25C256 is met. A 1-bit data delay is recommended when running the SPI_CLK at frequencies greater than 2.5 MHz; otherwise a 0-bit data delay can be used.
7. Program the SPI module to generate an interrupt after a frame of characters has been transferred (FIRQ = 1) or after a single character has been transferred (CIRQ = 1), If you want to use interrupts. The FIRQ and CIRQ bits are located in SPICMD1. This example assumes interrupts are not used.

8. Enable the SPI pins through the external bus selection register (EBSR). EBSR controls the pin multiplexing options of the device. The SPI pins are multiplexed with other peripherals on the device and the pin multiplexing configuration you choose will depend on your use-case scenario. Refer to the device-specific data manual for more information on EBSR.

8.3.3.2 Reading the SPI EEPROM Status Register

The status register of the CAT25C256 device provides useful information including the write status and write protect feature status. The SPI module can be easily used to read the contents of this register by following these steps:

1. Ensure the SPI is not busy with another transfer by polling the CC and BUSY bits in SPISTAT1.
2. Reading the EEPROM status register requires 8 clock cycles for the read status register command (RDSR) and eight cycles for the actual data. Therefore, set FLEN = 1 (2 character transfer) in SPICMD1.
3. Load the opcode for the read status register command (05h) into the upper byte of SPIDAT2.
4. Write to the command register (SPICMD2) to start the SPI write. The value you write SPICMD2 must set CSNUM = 0 (chip-select 0 active), CLEN = 7 (8-bit character length), and CMD = 10b (write).
5. Poll SPISTAT1 until CC = 1 or wait for the character complete interrupt (CIRQ).
6. Poll SPISTAT1 until BUSY = 0. This ensures that a character is not being transferred.
7. Prepare SPIDAT1 and SPIDAT2 for data reception by loading both registers with 0000h.
8. Write to SPICMD2 to start an SPI read. The value you write SPICMD2 must set CSNUM = 0 (chip-select 0 active), CLEN = 7 (8-bit character length), and CMD = 01b (read).
9. Poll SPISTAT1 until CC = 1 or wait for the character complete interrupt (CIRQ).
10. Poll SPISTAT1 until BUSY = 0. This ensures that a character is not being transferred.
11. Read the value shifted out by the SPI EEPROM from the lower byte of SPIDAT1 (mask lower byte in data read from SPIDAT1).

8.3.3.3 Enabling and Disabling Writes

Before writing data to the CAT25C256 device, writes must be enabled. Similarly, to prevent inadvertent writes, writes should be disabled. Writes can be enabled by sending the write enable command (WREN) and writes can be disabled by sending the write disable command (WRDI). Both of these commands can be easily sent to the SPI EEPROM using the SPI module.

1. Ensure the SPI is not busy with another transfer by polling the CC and BUSY bits in SPISTAT1.
2. Sending the WREN or WRDI command to the EEPROM requires a total of 8 clock cycles. Therefore, set FLEN = 0 (1 character transfer) in SPICMD1.
3. Load the opcode for either WREN (06h) or WRDI (04h) into the upper byte of SPIDAT2.
4. Write to the command register (SPICMD2) to start the SPI write. The value you write SPICMD2 must set CSNUM = 0 (chip-select 0 active), CLEN = 7 (8-bit character length), and CMD = 10b (write).
5. Poll SPISTAT1 until CC = 1 or wait for the character complete interrupt (CIRQ).
6. Poll SPISTAT1 until BUSY = 0. This ensures that a character is not being transferred.

You can read the EEPROM status register to verify the WREN or WRDI commands were received successfully. See Section 8.3.3.2 for more details on reading the status register of the SPI EEPROM.

8.3.3.4 Writing a Block of Data to a SPI EEPROM

Once the CAT25C256 device has been configured to accept writes, you can use the write data memory command (WRITE) to program the contents of the device. Please note that the CAT25C256 device allows for a page write sequence in which you are allowed to write up to 64 bytes (a page) while the EEPROM automatically increments its address counter. The only restriction during page writes is that page boundaries must not be crossed. If the edge of a page boundary is reached, the address counter rolls over to the beginning of the page. The following steps outline the procedure for conducting a page write to the SPI EEPROM.

1. Configure the SPI EEPROM to allow writes through the use of the WREN command. See Section 8.3.3.3 for more details.
2. Ensure the SPI is not busy with another transfer by polling the CC and BUSY bits in SPISTAT1.
3. Writing a page of data to the EEPROM requires 8 clock cycles for the WRITE command, 16 clock cycles for the address, and eight cycles for each data byte. Therefore, set FLEN = 1 + 2 + N – 1, where N is the number of data bytes to be written. For example, if you want to write 64 data bytes, set FLEN = 66.
4. Load the opcode for the WRITE command (02h) into the upper byte of SPIDAT2.
5. Write to the command register (SPICMD2) to start the SPI write. The value you write SPICMD2 must set CSNUM = 0 (chip-select 0 active), CLEN = 7 (8-bit character length), and CMD = 10b (write).
6. Poll SPISTAT1 until CC = 1 or wait for the character complete interrupt (CIRQ).
7. Poll SPISTAT1 until BUSY = 0. This ensures that a character is not being transferred.
8. Load the most-significant byte of the SPI EEPROM address into the upper byte of SPIDAT2.
9. Write to the command register (SPICMD2) to start the SPI write. The value you write SPICMD2 must set CSNUM = 0 (chip-select 0 active), CLEN = 7 (8-bit character length), and CMD = 10b (write).
10. Poll SPISTAT1 until CC = 1 or wait for the character complete interrupt (CIRQ).
11. Poll SPISTAT1 until BUSY = 0. This ensures that a character is not being transferred.
12. Load the least-significant byte of the SPI EEPROM address into the upper byte of SPIDAT2.
13. Write to the command register (SPICMD2) to start the SPI write. The value you write SPICMD2 must set CSNUM = 0 (chip-select 0 active), CLEN = 7 (8-bit character length), and CMD = 10b (write).
14. Poll SPISTAT1 until CC = 1 or wait for the character complete interrupt (CIRQ).
15. Poll SPISTAT1 until BUSY = 0. This ensures that a character is not being transferred.
16. Load the data byte to be written to the upper byte of SPIDAT2. The SPI module will shift data out starting with the most-significant bit of SPIDAT2.
17. Write to SPICMD2 to start an SPI write. The value you write SPICMD2 must set CSNUM = 0 (chip-select 0 active), CLEN = 7 (8-bit character length), and CMD = 10b (write).
18. Poll SPISTAT1 until CC = 1 or wait for the character complete interrupt (CIRQ).
19. Poll SPISTAT1 until BUSY = 0. This ensures that a character is not being transferred.
20. Repeat steps 16 through 19 for the remainder of the bytes.

At the end of this sequence, you can poll the $\overline{\text{RDY}}$ bit in the SPI EEPROM status register (see Section 8.3.3.2) to determine when the EEPROM has completed the writes. At that time, you can restart this sequence to write more bytes to the EEPROM at a different address.

8.3.3.5 Reading a Block of Data from a SPI EEPROM

Unlike a page write, the CAT25C256 device allows you to read the entire memory contents without regard for page boundaries. The read data from memory command (READ) is used to read data from the SPI EEPROM. The following steps outline the procedure required to read a block of data from the memory device.

1. Ensure the SPI is not busy with another transfer by polling the CC and BUSY bits in SPISTAT1.
2. Reading a block of data from the EEPROM requires 8 clock cycles for the READ command, 16 clock cycles for the address, and eight cycles for each data byte. Therefore, set FLEN = 1 + 2 + N – 1, where N is the number of bytes to be read. For example, if you want to read 64 bytes, set FLEN = 66.
3. Load the opcode for the READ command (03h) into the upper byte of SPIDAT2.
4. Write to the command register (SPICMD2) to start the SPI write. The value you write SPICMD2 must set CSNUM = 0 (chip-select 0 active), CLEN = 7 (8-bit character length), and CMD = 10b (write).
5. Poll SPISTAT1 until CC = 1 or wait for the character complete interrupt (CIRQ).
6. Poll SPISTAT1 until BUSY = 0. This ensures that a character is not being transferred.
7. Load the most-significant byte of the SPI EEPROM address into the upper byte of SPIDAT2.
8. Write to the command register (SPICMD2) to start the SPI write. The value you write SPICMD2 must set CSNUM = 0 (chip-select 0 active), CLEN = 7 (8-bit character length), and CMD = 10b (write).
9. Poll SPISTAT1 until CC = 1 or wait for the character complete interrupt (CIRQ).
10. Poll SPISTAT1 until BUSY = 0. This ensures that a character is not being transferred.
11. Load the least-significant byte of the SPI EEPROM address into the upper byte of SPIDAT2.
12. Write to the command register (SPICMD2) to start the SPI write. The value you write SPICMD2 must set CSNUM = 0 (chip-select 0 active), CLEN = 7 (8-bit character length), and CMD = 10b (write).
13. Poll SPISTAT1 until CC = 1 or wait for the character complete interrupt (CIRQ).
14. Poll SPISTAT1 until BUSY = 0. This ensures that a character is not being transferred.
15. Prepare SPIDAT1 and SPIDAT2 for data reception by loading both registers with 0000h.
16. Write to SPICMD2 to start an SPI read. The value you write SPICMD2 must set CSNUM = 0 (chip-select 0 active), CLEN = 7 (8-bit character length), and CMD = 01b (read).
17. Poll SPISTAT1 until CC = 1 or wait for the character complete interrupt (CIRQ).
18. Poll SPISTAT1 until BUSY = 0. This ensures that a character is not being transferred.
19. Read the value shifted out by the SPI EEPROM from the lower byte of (mask lower byte in data read from SPIDAT1) SPIDAT1.
20. Repeat steps 15 through 19 for the remainder of the bytes.

8.4 Registers

Table 8-4 lists the memory-mapped registers associated with the SPI module of the device DSP. The SPI registers can be accessed by the CPU at the 16-bit addresses specified in Table 8-4. Note that the CPU accesses all peripheral registers through its I/O space. All other register addresses not listed in Table 8-4 should be considered as reserved locations and the register contents should not be modified.

Table 8-4. SPI Module Registers

CPU Word Address	Acronym	Register Description	Section
3000h	SPICDR	Clock Divider Register	Section 8.4.1
3001h	SPICCR	Clock Control Register	Section 8.4.1
3002h	SPIDCR1	Device Configuration Register 1	Section 8.4.2
3003h	SPIDCR2	Device Configuration Register 2	Section 8.4.2
3004h	SPICMD1	Command Register 1	Section 8.4.3
3005h	SPICMD2	Command Register 2	Section 8.4.3
3006h	SPISTAT1	Status Register 1	Section 8.4.4
3007h	SPISTAT2	Status Register 2	Section 8.4.4
3008h	SPIDAT1	Data Register 1	Section 8.4.5
3009h	SPIDAT2	Data Register 2	Section 8.4.5

8.4.1 Clock Divider Register (SPICDR) and Clock Control Register (SPICCR)

The SPI includes two registers for controlling the SPI interface clock (SPI_CLK). The SPI input clock is divided down to generate a SPI_CLK. The clock divider register (SPICDR) specifies the clock divider value for the SPI input clock. The clock control register (SPICCR) is used to enable the clock output and to initiate a soft reset to the SPI module.

Figure 8-13. Clock Divider Register(SPICDR)

15–0
CLKDV
RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 8-5. Clock Divider Register (SPICDR) Field Descriptions

Bit	Field	Value	Description
15-0	CLKDV	0-3FFFh	Clock divider bits. The SPI input clock is divided down to generate the SPI interface clock (SPI_CLK). You can specify the divider value through the CLKDV bits. The frequency SPI_CLK is: SPI_CLK frequency = (SPI input clock frequency) / (CLKDV + 1) The duty cycle of SPI_CLK depends on the value of CLKDV + 1. When CLKDIV+1 is even, the duty cycle is 50%. When CLKDIV+1 is odd, the clock high and low times depend on the clock polarity. Low clock polarity: %high = (CLKDIV/2)/(CLK_DIV+1) %low = ((CLK_DIV/2)+1)/(CLK_DIV+1) High clock polarity %low = (CLKDIV/2)/(CLK_DIV+1) %high = ((CLK_DIV/2)+1)/(CLK_DIV+1) **NOTE:** For proper device operation CLKDV must be greater than or equal to 3. See Section 8.2.5 for more information on the SPI modes.

Figure 8-14. Clock Control Register (SPICCR)

15	14	13–0
CLKEN	RST	Reserved
RW-0	W-0	R-xxxx

LEGEND: R/W = Read/Write; W = Write only; R = Read only; -*n* = value after reset

Table 8-6. Clock Control Register (SPICCR) Field Descriptions

Bit	Field	Value	Description
15	CLKEN		Clock enable bit. This bit is used to enable and disable the SPI interface clock (SPI_CLK).
		0	SPI_CLK is disabled and held at the logic value specified by the clock polarity bit of SPIDC.
		1	SPI_CLK is enabled.
14	RST		Soft reset bit. Writing a 1 to this bit will reset the SPI module. This bit is self-clearing and will deactivate the soft reset on the next SPI input clock cycle after this bit is set to 1.
		0	Soft reset is released.
		1	Soft reset is asserted.
13-0	Reserved	0	Reserved.

8.4.2 Device Configuration Registers (SPIDCR1 and SPIDCR2)

The device configuration registers (SPIDCR1 and SPIDCR2) are used to specify the clock phase, clock polarity, and data delay for each SPI slave connected to the SPI chip select pins. Together, the clock phase and clock polarity determine the SPI mode (refer to Section 8.2.5 for more details). The loopback mode of the SPI can also be enabled through SPIDCR2.

Figure 8-15. Device Configuration Register 1 (SPIDCR1)

15 13	12 11	10	9	8	7 5	4 3	2	1	0
Reserved	DD1	CKPH1	CSP1	CKP1	Reserved	DD0	CKPH0	CSP0	CKP0
R-0	RW-0	RW-0	RW-0	RW-0	R-0	RW-0	RW-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 8-7. Device Configuration Register 1 (SPIDCR1) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12-11	DD1	0-3h	Data delay for chip select 1 pin (SPI_CS1).
		0	First SPI_CLK edge is delayed 0.5 clock cycles from SPI_CS1 assertion.
		1h	First SPI_CLK edge is delayed 1.5 SPI clock cycles from SPI_CS1 assertion.
		2h	First SPI_CLK edge is delayed 2.5 SPI clock cycles from SPI_CS1 assertion.
		3h	First SPI_CLK edge is delayed 3.5 SPI clock cycles from SPI_CS1 assertion.
10	CKPH1		Clock phase for chip select 1 pin (SPI_CS1). The clock phase bit, in conjunction with the clock polarity bit (CKP1), controls the clock-data relationship between master and slave.
		0	When CKP1 = 0, data shifted out on falling edge, input captured on rising edge. When CKP1 = 1, data shifted out on rising edge, input captured on falling edge.
		1	When CKP1 = 0, data shifted out on rising edge, input captured on falling edge. When CKP1 = 1, data shifted out on falling edge, input captured on rising edge.
9	CSP1		Polarity for chip select 1 pin (SPI_CS1).
		0	Active low.
		1	Active high.
8	CKP1		Clock polarity inactive state for the clock pin during accesses to chip select 1.
		0	When data is not being transferred, a steady state low value is produced at the SPI_CLK pin.
		1	When data is not being transferred, a steady state high value is produced at the SPI_CLK pin.
7-5	Reserved	0	Reserved.
4-3	DD0	0-3h	Data delay for chip select 0 pin (SPI_CS0).
		0	First SPI_CLK edge is delayed 0.5 clock cycles from SPI_CS0 assertion.
		1h	First SPI_CLK edge is delayed 1.5 clock cycles from SPI_CS0 assertion.
		2h	First SPI_CLK edge is delayed 2.5 clock cycles from SPI_CS0 assertion.
		3h	First SPI_CLK edge is delayed 3.5 clock cycles from SPI_CS0 assertion.
2	CKPH0		Clock phase for chip select 0 pin (SPI_CS0). The clock phase bit, in conjunction with the clock polarity bit (CKP0), controls the clock-data relationship between master and slave.
		0	When CKP0 = 0, data shifted out on falling edge, input captured on rising edge. When CKP0 = 1, data shifted out on rising edge, input captured on falling edge.
		1	When CKP0 = 0, data shifted out on rising edge, input captured on falling edge. When CKP0 = 1, data shifted out on falling edge, input captured on rising edge.
1	CSP0		Polarity for chip select 0 pin (SPI_CS0).
		0	Active low.
		1	Active high.
0	CKP0		Clock polarity inactive state for the clock pin during accesses to chip select 0.
		0	When data is not being transferred, a steady state low value is produced at the SPI_CLK pin.
		1	When data is not being transferred, a steady state high value is produced at the SPI_CLK pin.

Figure 8-16. Device Configuration Register 2 (SPIDCR2)

15	14 13	12 11	10	9	8	7 5	4 3	2	1	0
LPBK	Reserved	DD3	CKPH3	CSP3	CKP3	Reserved	DD2	CKPH2	CSP2	CKP2
RW-0	R-0	RW-0	RW-0	RW-0	RW-0	R-0	RW-0	RW-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 8-8. Device Configuration Register 2 (SPIDCR2) Field Descriptions

Bit	Field	Value	Description
15	LPBK		Loopback mode.
		0	Loopback mode is disabled.
		1	Loopback mode is enabled.
14-13	Reserved	0	Reserved
12-11	DD3	0-3h	Data delay for chip select 3 pin (SPI_CS3).
		0	First SPI_CLK edge is delayed 0.5 clock cycles from SPI_CS3 assertion.
		1h	First SPI_CLK edge is delayed 1.5 clock cycles from SPI_CS3 assertion.
		2h	First SPI_CLK edge is delayed 2.5 clock cycles from SPI_CS3 assertion.
		3h	First SPI_CLK edge is delayed 3.5 clock cycles from SPI_CS3 assertion.
10	CKPH3		Clock phase for chip select 3 pin (SPI_CS3). The clock phase bit, in conjunction with the clock polarity bit (CKP3), controls the clock-data relationship between master and slave.
		0	When CKP3 = 0, data shifted out on falling edge, input captured on rising edge. When CKP3 = 1, data shifted out on rising edge, input captured on falling edge.
		1	When CKP3 = 0, data shifted out on rising edge, input captured on falling edge. When CKP3 = 1, data shifted out on falling edge, input captured on rising edge.
9	CSP3		Polarity for chip select 3 pin (SPI_CS3).
		0	Active low.
		1	Active high.
8	CKP3		Clock polarity inactive state for the clock pin during accesses to chip select 3.
		0	When data is not being transferred, a steady state low value is produced at the SPI_CLK pin.
		1	When data is not being transferred, a steady state high value is produced at the SPI_CLK pin.
7-5	Reserved	0	Reserved
4-3	DD2	0-3h	Data delay for chip select 2 pin (SPI_CS2).
		0	First SPI_CLK edge is delayed 0.5 clock cycles from SPI_CS2 assertion.
		1h	First SPI_CLK edge is delayed 1.5 clock cycles from SPI_CS2 assertion.
		2h	First SPI_CLK edge is delayed 2.5 clock cycles from SPI_CS2 assertion.
		3h	First SPI_CLK edge is delayed 3.5 clock cycles from SPI_CS2 assertion.
2	CKPH2		Clock phase for chip select 2 pin (SPI_CS2). The clock phase bit, in conjunction with the clock polarity bit (CKP2), controls the clock-data relationship between master and slave.
		0	When CKP2 = 0, data shifted out on falling edge, input captured on rising edge. When CKP2 = 1, data shifted out on rising edge, input captured on falling edge.
		1	When CKP2 = 0, data shifted out on rising edge, input captured on falling edge. When CKP2 = 1, data shifted out on falling edge, input captured on rising edge.
1	CSP2		Polarity for chip select 2 pin (SPI_CS2).
		0	Active low.
		1	Active high.
0	CKP2		Clock polarity inactive state for the clock pin during accesses to chip select 2.
		0	When data is not being transferred, a steady state low value is produced at the SPI_CLK pin.
		1	When data is not being transferred, a steady state high value is produced at the SPI_CLK pin.

8.4.3 Command Registers (SPICMD1 and SPICMD2)

The command registers (SPICMD1 and SPICMD2) are used to control several aspects of an SPI access. You can use SPICMD1 to enable character and frame complete interrupts and to specify the number of characters in a frame. The command to use (read or write), character length, and chip select pin used during SPI transfers are specified through SPICMD2. Writing to SPICMD2 will cause the SPI to execute the command specified by the transfer command bits (CMD).

Figure 8-17. Command Register 1 (SPICMD1)

15	14	13	12	11	0
FIRQ	CIRQ	Reserved		FLEN	
RW-0	RW-0	R-0		RW-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 8-9. Command Register 1 (SPICMD1) Field Descriptions

Bit	Field	Value	Description
15	FIRQ		Frame count interrupt enable.
		0	No interrupt generated at the end of the frame count.
		1	Interrupt generated at the end of the frame count.
14	CIRQ		Character interrupt enable.
		0	No interrupt generated at the end of the character transfer.
		1	Interrupt generated at the end of the character transfer.
13-12	Reserved	0	Reserved.
11-0	FLEN	0-FFFh	Frame length bits. These bits are used to specify the length of entire transfer. The total number of characters transferred equals FLEN + 1. For example, if FLEN = 63, a frame consists of a total of 64 characters.

Figure 8-18. Command Register 2 (SPICMD2)

15 14	13 12	11 8	7 3	2	1 0
Reserved	CSNUM	Reserved	CLEN	RSV	CMD
R-0	RW-0	R-0	RW-0	R-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 8-10. Command Register 2 (SPICMD2) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-12	CSNUM	0-3h	Device select. Sets the active chip select for the transfer.
		0	Chip select 0 is active.
		1h	Chip select 1 is active.
		2h	Chip select 2 is active.
		3h	Chip select 3 is active.
11-8	Reserved	0	Reserved.
7-3	CLEN	0-1Fh	Character length. Sets the transfer size of the individual transfer elements from 1 to 32 bits. The character length is set to CLEN + 1. For example, if CLEN = 7, the character length is set to 8 bits.
2	Reserved	0	Reserved.
1-0	CMD	0-3h	Transfer command bits. These bits specify the type of transaction being used.
		0	Reserved.
		1h	Read.
		2h	Write.
		3h	Reserved.

8.4.4 Status Registers (SPISTAT1 and SPISTAT2)

The status registers (SPISTAT1 and SPISTAT2) contain indicators to allow monitoring of the progression of a frame transfer. The character complete (CC) and frame complete (FC) bits are used as stimulus for generating interrupts. Setting the corresponding interrupt enable bits in the SPICMD1 command register allows these events to generate an interrupt. The CC and FC bits are reset every time (1) SPISTAT1 is read, or (2) a new SPI transfer is initiated via a write of a read or write command to CMD in the SPICMD2 command register.

Figure 8-19. Status Register 1 (SPISTAT1)

15	3	2	1	0
Reserved		FC	CC	BUSY
R-0		R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 8-11. Status Register 1 (SPISTAT1) Field Descriptions

Bit	Field	Value	Description
15-3	Reserved	0	Reserved
2	FC		Frame complete. This bit is set after all the requested characters have been transferred. This bit is reset when SPISTAT1 is read or when a new SPI transfer is initiated via a write of a read or write command to CMD in SPICMD2.
		0	Transfer is not complete.
		1	All characters have been transferred.
1	CC		Character complete. This bit is set after each character transfer is completed. This bit is reset when SPISTAT1 is read or when a new SPI transfer is initiated via a write of a read or write command to CMD in SPICMD2.
		0	Character transfer is not complete.
		1	Character transfer is complete.
0	BUSY		Busy bit. This bit is set during an active character transfer. Between characters this bit will be cleared to signal that the data registers can be accessed.
		0	Idle, no character transfers in progress.
		1	Active, a character transfer is in progress.

Figure 8-20. Status Register 2 (SPISTAT2)

15	12	11	0
Reserved		CCNT	
R–0		R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 8-12. Status Register 2 (SPISTAT2) Field Descriptions

Bit	Field	Value	Description
15-12	Reserved	0	Reserved.
12-0	CCNT	0-FFFh	Character count. These bits reflect the total number of characters transferred. For example, when CCNT = 64, a total of 64 characters have been transferred. These bits are reset upon completion of a frame. and should only be read after determining CC = 1.

8.4.5 Data Registers (SPIDAT1 and SPIDAT2)

The data registers (SPIDAT1 and SPIDAT2) are treated as a 32-bit shift register. Data received by the SPI is shifted into the least-significant bit of SPIDAT1 and the contents of both registers are shifted to the left. Similarly, data transferred by the SPI is shifted out of the most-significant bit of SPIDAT2 and the contents of both registers are shifted to the left. This process is illustrated in Figure 8-9.

The data registers are not cleared between reads or writes. Old data may exist in the register and it is the responsibility of the user to mask off any unneeded data on a read.

Figure 8-21. Data Register 1 (SPIDAT1)

15	0
DATA	
RW-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 8-13. Data Register 1 (SPIDAT1) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Low part of data during read and write operations.

Figure 8-22. Data Register 2 (SPIDAT2)

15	0
DATA	
RW-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 8-14. Data Register 2 (SPIDAT2) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	High part of data during read and write operations.

Chapter 9

SPRUH87C – August 2011 – Revised March 2012

Inter-Integrated Circuit (I2C) Peripheral

This chapter describes the features and operations of the inter-integrated circuit (I2C) peripheral.

Topic **Page**

9.1 Introduction

Described in the following sections is the operation of the inter-integrated circuit (I2C) peripheral. The scope of this document assumes that you are familiar with the Philips Semiconductors Inter-IC bus (I2C-bus) specification version 2.1.

9.1.1 Purpose of the Peripheral

The I2C peripheral provides an interface between the DSP and other devices that are compliant with the I2C-bus specification and connected by way of an I2C-bus. External components that are attached to this two-wire serial bus can transmit and receive data that is up to eight bits wide both to and from the DSP through the I2C peripheral.

9.1.2 Features

The I2C peripheral has the following features:

- Compliance with the Philips Semiconductors I2C-bus specification (version 2.1):
  - Support for byte format transfer.
  - 7-bit and 10-bit addressing modes.
  - General call.
  - START byte mode.
  - Support for multiple master-transmitters and slave-receivers mode.
  - Support for multiple slave-transmitters and master-receivers mode.
  - Combined master transmit/receive and receive/transmit mode.
  - I2C data transfer rate of from 10 kbps up to 400 kbps (Philips I2C rate).
- 2 to 8 bit format transfer.
- Free data format mode.
- One read DMA event and one write DMA event that the DMA can use.
- Seven interrupts that the CPU can use.
- Peripheral enable/disable capability.

9.1.2.1 Features Not Supported

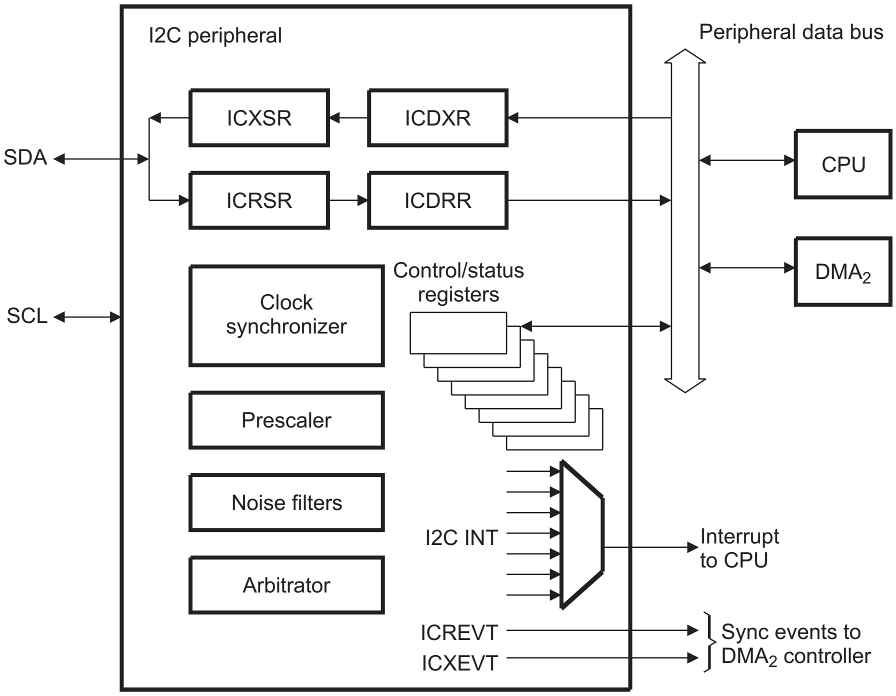
The I2C peripheral does not support the following features:

- High-speed mode.
- CBUS-compatibility mode.
- The combined format in 10-bit addressing mode (the I2C sends the slave address the second byte every time it sends the slave address the first byte).

9.1.3 Functional Block Diagram

A block diagram of the I2C peripheral is shown in Figure 9-1. Refer to Section 9.2 for detailed information about the architecture of the I2C peripheral.

Figure 9-1. I2C Peripheral Block Diagram



A Out of the four DMA controllers included in the DSP, only DMA controller 2 (DMA2) can access the I2C peripheral registers and use its DMA events.

9.1.4 Industry Standard(s) Compliance Statement

The I2C peripheral is compliant with the Philips Semiconductors Inter-IC bus (I2C-bus) specification version 2.1.

9.2 Peripheral Architecture

The I2C peripheral consists of the following primary blocks:

- A serial interface: one data pin (SDA) and one clock pin (SCL).
- Data registers to temporarily hold receive data and transmit data traveling between the SDA pin and the CPU or the DMA2 controller.
- Control and status registers.
- A peripheral data bus interface to enable the CPU and the DMA2 controller to access the I2C peripheral registers.
- A clock synchronizer to synchronize the I2C input clock (from the processor clock generator) and the clock on the SCL pin, and to synchronize data transfers with masters of different clock speeds.
- A prescaler to divide down the input clock that is driven to the I2C peripheral.
- A noise filter on each of the two pins, SDA and SCL.
- An arbitrator to handle arbitration between the I2C peripheral (when it is a master) and another master.
- Interrupt generation logic, so that an interrupt can be sent to the CPU.
- DMA event generation logic, so that activity in the DMA2 controller can be synchronized to data reception and data transmission in the I2C peripheral.

Figure 9-1 shows the four registers used for transmission and reception. The CPU or the DMA controller writes data for transmission to ICDXR and reads received data from ICDRR. When the I2C peripheral is configured as a transmitter, data written to ICDXR is copied to ICXSR and shifted out on the SDA pin one bit at a time. When the I2C peripheral is configured as a receiver, received data is shifted into ICRSR and then copied to ICDRR.

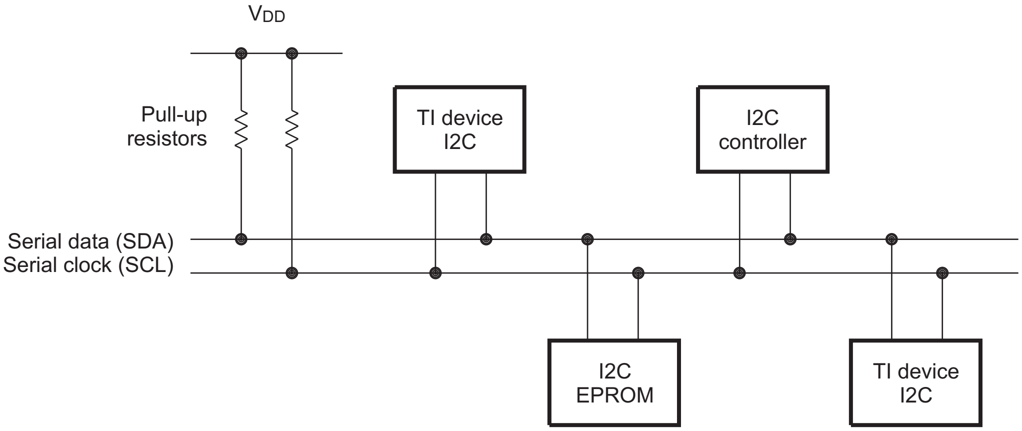
9.2.1 Bus Structure

Figure 9-1 shows how the I2C peripheral is connected to the I2C bus. The I2C bus is a multi-master bus that supports a multi-master mode. This allows more than one device capable of controlling the bus that is connected to it. A unique address recognizes each I2C device. Each I2C device can operate as either transmitter or receiver, depending on the function of the device. Devices that are connected to the I2C bus can be considered a master or slave when performing data transfers, in addition to being a transmitter or receiver.

NOTE: A master device is the device that initiates a data transfer on the bus and generates the clock signals to permit that transfer. Any device that is addressed by this master is considered a slave during this transfer.

An example of multiple I2C modules that are connected for a two-way transfer from one device to other devices is shown in Figure 9-2.

Figure 9-2. Multiple I2C Modules Connected

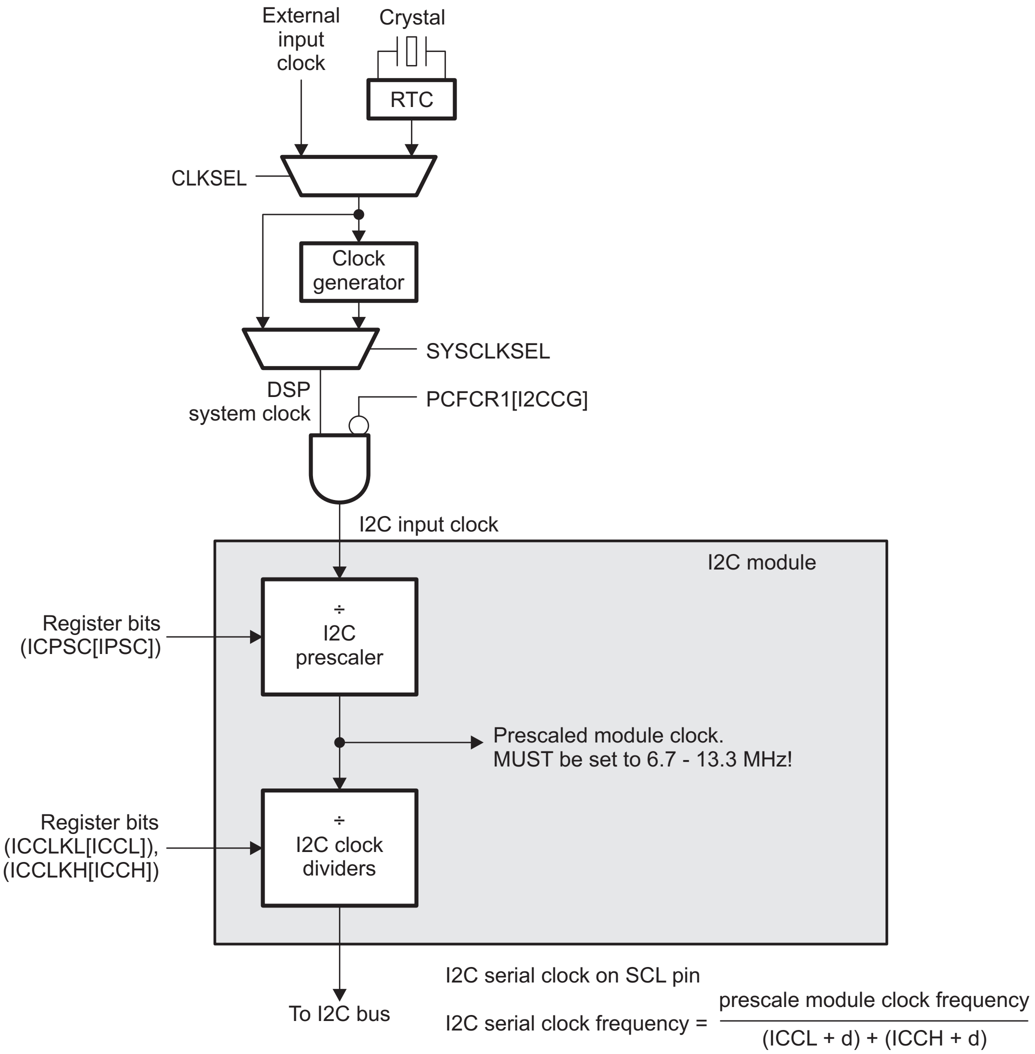


9.2.2 Clock Generation

As shown in Figure 9-3, the clock generator receives either the real-time clock (RTC) or a signal from an external clock source and produces the DSP system clock. This clock is used by the DSP CPU and peripherals. A programmable prescaler (IPSC bit in ICPSC) in the I2C module divides down the I2C input clock to produce a prescaled module clock. The prescaled module clock must be operated within the range of 6.7 to 13.3 MHz. The I2C clock dividers divide-down the high (ICCH bit in ICCLKH) and low portions (ICCL bit in ICCLKL) of the prescaled module clock signal to produce the I2C serial clock, which appears on the SCL pin when the I2C module is configured to be a master on the I2C bus.

The DSP includes logic which can be used to gate the clock to its on-chip peripherals. The I2C input clock can be enabled and disabled through the peripheral clock gating configuration register 1 (PCGCR1).

Figure 9-3. Clocking Diagram for the I2C Peripheral



$$\text{I2C serial clock frequency} = \frac{\text{prescale module clock frequency}}{(\text{ICCL} + d) + (\text{ICCH} + d)}$$

Where d depends on IPSC value in I2CPSC:

IPSC value	d
0	7
1	6
2h-FFh	5

CAUTION

Prescaled Module Clock Frequency Range:

The I2C module must be operated with a prescaled module clock frequency of 6.7 to 13.3 MHz. The I2C prescaler register (ICPSC) must be configured to this frequency range.

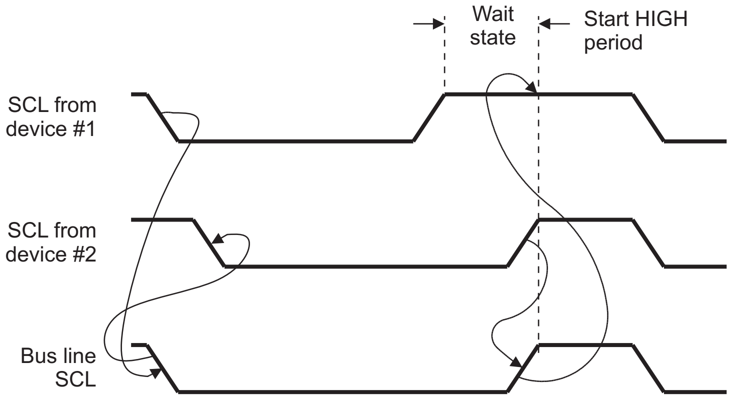
The prescaler (IPSC bit in ICPSC) must only be initialized while the I2C module is in the reset state (IRS = 0 in ICMDR). The prescaled frequency only takes effect when the IRS bit in ICMDR is changed to 1. Changing the IPSC bit in ICPSC while IRS = 1 in ICMDR has no effect. Likewise, you must configure the I2C clock dividers (ICCH bit in ICCLKH and ICCL bit in ICCLKL) while the I2C module is still in reset (IRS = 0 in ICMDR).

9.2.3 Clock Synchronization

Under normal conditions a master device generates its own clock signal on the SCL pin. However, when there are two or more masters connected to the I2C bus the clock must be synchronized such that bit-by-bit arbitration (described in Section 9.2.9) can take place. Figure 9-4 illustrates the clock synchronization. The wired-AND property of SCL means that a device that first generates a low period on SCL (device #1) overrules the other devices. At this high-to-low transition, the clock generators of the other devices are forced to start their own low period. The SCL is held low by the device with the longest low period. The other devices that finish their low periods must wait for SCL to be released before starting their high periods. Using this approach, a synchronized signal on SCL is obtained where the slowest device determines the length of the low period and the fastest device determines the length of the high period.

If a device pulls down the clock line for a longer time, the result is that all clock generators must enter the wait state. This way, a slave slows down a fast master and the slow device creates enough time to store a received data word or to prepare a data word that you are going to transmit.

Figure 9-4. Synchronization of Two I2C Clock Generators



9.2.4 Signal Descriptions

The I2C peripheral has a serial data pin (SDA) and a serial clock pin (SCL) for data communication, as shown in Figure 9-1. These two pins carry information between the device and other devices that are connected to the I2C-bus. The SDA and SCL pins both are bi-directional. They each must be connected to a positive supply voltage using a pull-up resistor. When the bus is free, both pins are high. The driver of these two pins has an open-drain configuration to perform the required wired-AND function.

See the device-specific data manual for additional timing and electrical specifications for these pins.

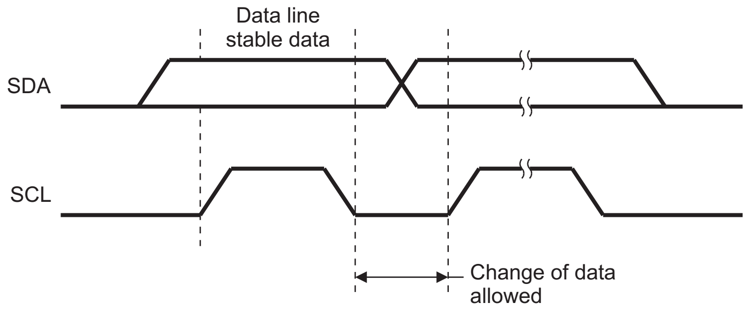
9.2.4.1 Input and Output Voltage Levels

The master device generates one clock pulse for each data bit that is transferred. Due to a variety of different technology devices that can be connected to the I2C-bus, the levels of logic 0 (low) and logic 1 (high) are not fixed and depend on the associated power supply level. See the device-specific data manual for more information.

9.2.4.2 Data Validity

The data on SDA must be stable during the high period of the clock (see Figure 9-5). The high or low state of the data line, SDA, can change only when the clock signal on SCL is low.

Figure 9-5. Bit Transfer on the I2C-Bus



9.2.5 START and STOP Conditions

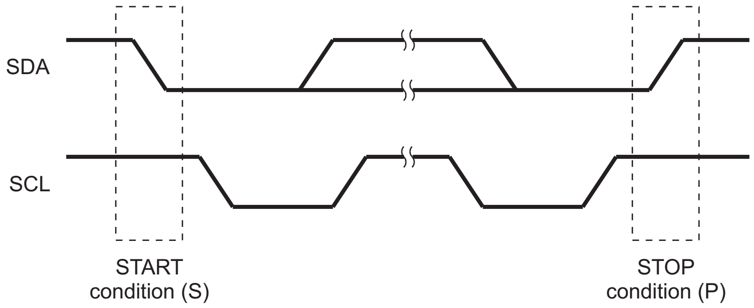
The I2C peripheral can generate START and STOP conditions when the peripheral is configured to be a master on the I2C-bus, as shown in Figure 9-6:

- The START condition is defined as a high-to-low transition on the SDA line while SCL is high. A master drives this condition to indicate the start of a data transfer.
- The STOP condition is defined as a low-to-high transition on the SDA line while SCL is high. A master drives this condition to indicate the end of a data transfer.

The I2C-bus is considered busy after a START condition and before a subsequent STOP condition. The bus busy (BB) bit of ICSTR is 1. The bus is considered free between a STOP condition and the next START condition. The BB is 0.

The master mode (MST) bit and the START condition (STT) bit in ICMDR must both be 1 for the I2C peripheral to start a data transfer with a START condition. The STOP condition (STP) bit must be set to 1 for the I2C peripheral to end a data transfer with a STOP condition. A repeated START condition generates when BB is set to 1 and STT is also set to 1. See Section 9.3.9 for a description of ICMDR (including the MST, STT, and STP bits).

Figure 9-6. I2C Peripheral START and STOP Conditions



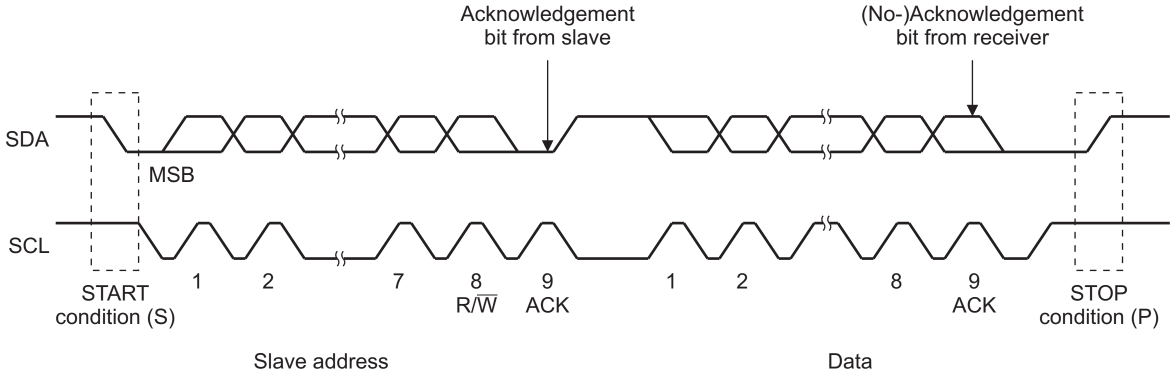
9.2.6 Serial Data Formats

Figure 9-7 shows an example of a data transfer on the I2C-bus. The I2C peripheral supports 2-bit to 8-bit data values. Figure 9-7 shows a typical I2C data transfer using (BC = 000 in ICMDR). Each bit put on the SDA line is equivalent to one pulse on the SCL line. The data is always transferred with the most-significant bit (MSB) first. The number of data values that can be transmitted or received is unrestricted; however, in most systems, the transmitter and receiver have agreed upon the number of data values to transfer before transfer begins.

The I2C peripheral supports the following data formats:

- 7-bit addressing mode.
- 10-bit addressing mode.
- Free data format mode.

Figure 9-7. I2C Peripheral Data Transfer



9.2.6.1 7-Bit Addressing Format

In the 7-bit addressing format (Figure 9-8), the first byte after a START condition (S) consists of a 7-bit slave address followed by a R/$\overline{W}$ bit. The R/$\overline{W}$ bit determines the direction of the data.

- R/$\overline{W}$ = 0: The master writes (transmits) data to the addressed slave.
- R/$\overline{W}$ = 1: The master reads (receives) data from the slave.

An extra clock cycle dedicated for acknowledgment (ACK) is inserted after the R/$\overline{W}$ bit. If the slave inserts the ACK bit, *n* bits of data from the transmitter (master or slave, depending on the R/$\overline{W}$ bit) follow it. *n* is a number from 2 to 8 that the bit count (BC) bits of ICMDR determine. The receiver inserts an ACK bit after the data bits have been transferred.

Write a 0 to the expanded address enable (XA) bit of ICMDR to select the 7-bit addressing format.

Figure 9-8. I2C Peripheral 7-Bit Addressing Format (FDF = 0, XA = 0 in ICMDR)

1	7	1	1	n	1	n	1	1
S	Slave address	R/$\overline{W}$	ACK	Data	ACK	Data	ACK	P

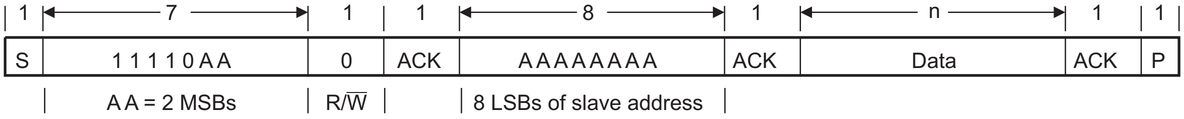
n = The number of data bits (from 2 to 8) specified by the bit count (BC) field of ICM DR.

9.2.6.2 10-Bit Addressing Format

The 10-bit addressing format (Figure 9-9) is like the 7-bit addressing format, but the master sends the slave address in two separate byte transfers. The first byte consists of 11110b, the two MSBs of the 10-bit slave address, and R/$\overline{W}$ = 0 (write). The second byte is the remaining 8 bits of the 10-bit slave address. The slave must send acknowledgment (ACK) after each of the two byte transfers. Once the master has written the second byte to the slave, the master can either write data or use a repeated START condition to change the data direction. (For more information about using 10-bit addressing, see the Philips Semiconductors I2C-bus specification.)

Write 1 to the XA bit of ICMDR to select the 10-bit addressing format.

Figure 9-9. I2C Peripheral 10-Bit Addressing Format With Master-Transmitter Writing to Slave-Receiver (FDF = 0, XA = 1 in ICMDR)



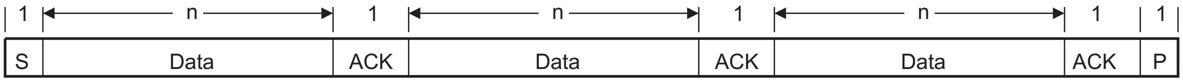
n = The number of data bits (from 2 to 8) specified by the bit count (BC) field of ICMDR.

9.2.6.3 Free Data Format

In the free data format (Figure 9-10), the first bits after a START condition (S) are a data word. An ACK bit is inserted after each data word, which can be from 2 to 8 bits, depending on the bit count (BC) bits of ICMDR. No address or data-direction bit is sent. Therefore, the transmitter and the receiver must both support the free data format, and the direction of the data must be constant throughout the transfer.

To select the free data format, write 1 to the free data format (FDF) bit of ICMDR.

Figure 9-10. I2C Peripheral Free Data Format (FDF = 1 in ICMDR)

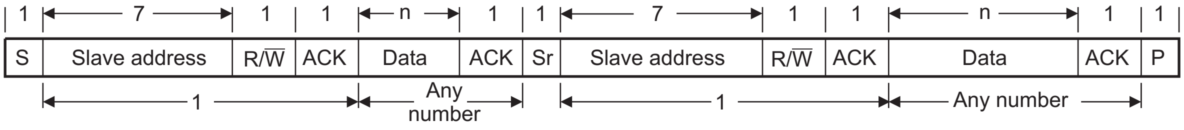


n = The number of data bits (from 2 to 8) specified by the bit count (BC) field of ICMDR.

9.2.6.4 Using a Repeated START Condition

The repeated START condition can be used with the 7-bit addressing, 10-bit addressing, and free data formats. An example of using the repeated START condition (Sr) in the 7-bit addressing format is shown in Figure 9-11. At the end of each data word, the master can drive another START condition. Using this capability, a master can transmit/receive any number of data words before driving a STOP condition. The length of a data word can be from 2 to 8 bits and is selected with the bit count (BC) bits of ICMDR.

Figure 9-11. I2C Peripheral 7-Bit Addressing Format With Repeated START Condition (FDF = 0, XA = 0 in ICMDR)



n = The number of data bits (from 2 to 8) specified by the bit count (BC) field of ICMDR.

9.2.7 Operating Modes

The I2C peripheral has four basic operating modes to support data transfers as a master and as a slave. See Table 9-1 for the names and descriptions of the modes.

If the I2C peripheral is a master, it begins as a master-transmitter and, typically, transmits an address for a particular slave. When giving data to the slave, the I2C peripheral must remain a master-transmitter. In order to receive data from a slave, the I2C peripheral must be changed to the master-receiver mode.

If the I2C peripheral is a slave, it begins as a slave-receiver and, typically, sends acknowledgment when it recognizes its slave address from a master. If the master will be sending data to the I2C peripheral, the peripheral must remain a slave-receiver. If the master has requested data from the I2C peripheral, the peripheral must be changed to the slave-transmitter mode.

Table 9-1. Operating Modes of the I2C Peripheral

Operating Mode	Description
Slave-receiver mode	The I2C peripheral is a slave and receives data from a master. All slave modules begin in this mode. In this mode, serial data bits received on SDA are shifted in with the clock pulses that are generated by the master. As a slave, the I2C peripheral does not generate the clock signal, but it can hold SCL low while the intervention of the processor is required (RSFULL = 1 in ICSTR) after data has been received.
Slave-transmitter mode	The I2C peripheral is a slave and transmits data to a master. This mode can only be entered from the slave-receiver mode; the I2C peripheral must first receive a command from the master. When you are using any of the 7-bit/10-bit addressing formats, the I2C peripheral enters its slave-transmitter mode if the slave address is the same as its own address (in ICOAR) and the master has transmitted R/$\overline{W}$ = 1. As a slave-transmitter, the I2C peripheral then shifts the serial data out on SDA with the clock pulses that are generated by the master. While a slave, the I2C peripheral does not generate the clock signal, but it can hold SCL low while the intervention of the processor is required (XSMT = 0 in ICSTR) after data has been transmitted.
Master-receiver mode	The I2C peripheral is a master and receives data from a slave. This mode can only be entered from the master-transmitter mode; the I2C peripheral must first transmit a command to the slave. When you are using any of the 7-bit/10-bit addressing formats, the I2C peripheral enters its master-receiver mode after transmitting the slave address and R/$\overline{W}$ = 1. Serial data bits on SDA are shifted into the I2C peripheral with the clock pulses generated by the I2C peripheral on SCL. The clock pulses are inhibited and SCL is held low when the intervention of the processor is required (RSFULL = 1 in ICSTR) after data has been received.
Master-transmitter mode	The I2C peripheral is a master and transmits control information and data to a slave. All master modules begin in this mode. In this mode, data assembled in any of the 7-bit/10-bit addressing formats is shifted out on SDA. The bit shifting is synchronized with the clock pulses generated by the I2C peripheral on SCL. The clock pulses are inhibited and SCL is held low when the intervention of the processor is required (XSMT = 0 in ICSTR) after data has been transmitted.

9.2.8 NACK Bit Generation

When the I2C peripheral is a receiver (master or slave), it can acknowledge or ignore bits sent by the transmitter. To ignore any new bits, the I2C peripheral must send a no-acknowledge (NACK) bit during the acknowledge cycle on the bus. Table 9-2 summarizes the various ways the I2C peripheral sends a NACK bit.

Table 9-2. Ways to Generate a NACK Bit

I2C Peripheral Condition	NACK Bit Generation	
	Basic	Optional
Slave-receiver mode	• Disable data transfers (STT = 0 in ICSTR). • Allow an overrun condition (RSFULL = 1 in ICSTR). • Reset the peripheral (IRS = 0 in ICMDR).	Set the NACKMOD bit of ICMDR before the rising edge of the last data bit you intend to receive.
Master-receiver mode AND Repeat mode (RM = 1 in ICMDR)	• Generate a STOP condition (STOP = 1 in ICMDR). • Reset the peripheral (IRS = 0 in ICMDR).	Set the NACKMOD bit of ICMDR before the rising edge of the last data bit you intend to receive.
Master-receiver mode AND Nonrepeat mode (RM = 0 in ICMDR)	• If STP = 1 in ICMDR, allow the internal data counter to count down to 0 and force a STOP condition. • If STP = 0, make STP = 1 to generate a STOP condition. • Reset the peripheral (IRS = 0 in ICMDR).	Set the NACKMOD bit of ICMDR before the rising edge of the last data bit you intend to receive.

9.2.9 Arbitration

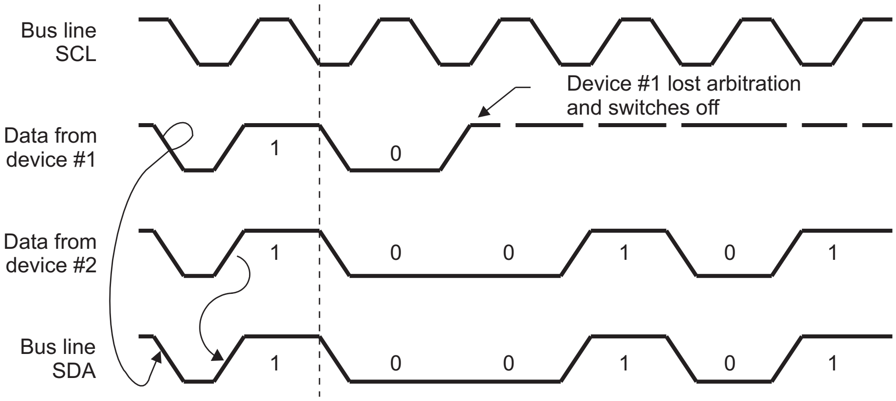
If two or more master-transmitters simultaneously start a transmission on the same bus, an arbitration procedure is invoked. The arbitration procedure uses the data presented on the serial data bus (SDA) by the competing transmitters. Figure 9-12 illustrates the arbitration procedure between two devices. The first master-transmitter, which drives SDA high, is overruled by another master-transmitter that drives SDA low. The arbitration procedure gives priority to the device that transmits the serial data stream with the lowest binary value. Should two or more devices send identical first bytes, arbitration continues on the subsequent bytes.

If the I2C peripheral is the losing master, it switches to the slave-receiver mode, sets the arbitration lost (AL) flag, and generates the arbitration-lost interrupt.

If during a serial transfer the arbitration procedure is still in progress when a repeated START condition or a STOP condition is transmitted to SDA, the master-transmitters involved must send the repeated START condition or the STOP condition at the same position in the format frame. Arbitration is not allowed between:

- A repeated START condition and a data bit.
- A STOP condition and a data bit.
- A repeated START condition and a STOP condition.

Figure 9-12. Arbitration Procedure Between Two Master-Transmitters



9.2.10 Reset Considerations

The I2C peripheral has two reset sources: software reset and hardware reset.

9.2.10.1 Software Reset Considerations

The I2C peripheral can be reset by software through the I2C reset (IRS) bit in the I2C mode register (ICMDR) or through the I2C_RST bit in the peripheral reset control register (PRCR).

When IRS is cleared to 0, all status bits in the I2C interrupt status register (ICSTR) are forced to their default values, and the I2C peripheral remains disabled until IRS is changed to 1. The SDA and SCL pins are in the high-impedance state.

NOTE: If the IRS bit is cleared to 0 during a transfer, this can cause the I2C bus to hang (SDA and SCL are in the high-impedance state).

When I2C_RST is set to 1, a hardware reset is forced on the I2C. The effects of a hardware reset are described in the next section. Please note that the I2C input clock must be enabled when using I2C_RST (see Section 9.2.2).

9.2.10.2 Hardware Reset Considerations

A hardware reset is always initiated during a full chip reset. Alternatively, software can force an I2C hardware reset through the I2C_RST bits of the peripheral reset control register (PRCR). See the device-specific data manual for more details on PRCR.

When a hardware reset occurs, all the registers of the I2C peripheral are set to their default values and the I2C peripheral remains disabled until the I2C reset (IRS) bit in the I2C mode register (ICMDR) is changed to 1.

NOTE: The IRS bit must be cleared to 0 while you configure/reconfigure the I2C peripheral. Forcing IRS to 0 can be used to save power and to clear error conditions.

9.2.11 Initialization

Proper I2C initialization is required prior to starting communication with other I2C device(s). Unless a fully fledged software driver is in place, you need to determine the required I2C configuration needed (for example, Master Receiver, etc.) and configure the I2C controller with the desired settings. Enabling the I2C input clock should be the first task. With the I2C controller still in reset, you are now ready to configure the I2C controller. Once configuration is done, you need to enable the I2C controller by releasing the controller from reset. Prior to starting communication, you need to make sure that all status bits are cleared and no pending interrupts exist. Once the bus is determined to be available (the bus is not busy), the I2C is ready to proceed with the desired communication.

9.2.11.1 Configuring the I2C in Master Receiver Mode and Servicing Receive Data via CPU

The following initialization procedure is for the I2C controller configured in Master Receiver mode. The CPU is used to move data from the I2C receive register to internal DSP memory.

1. Ensure the I2C is out of reset by setting I2C-RST=0 in the peripheral reset control register (PRCR). See the device-specific data manual for more information on PRCR.
2. Enable the I2C input clock by setting I2CCG to 1 in the peripheral clock gating configuration register (PCGCR1). See the device-specific data manual for more information on PCGCR1.
3. Place I2C in reset (clear IRS = 0 in ICMDR).
4. Configure ICMDR:
   - Configure I2C as Master (MST = 1).
   - Indicate the I2C configuration to be used; for example, Data Receiver (TRX = 0).
   - Indicate 7-bit addressing is to be used (XA = 0).
   - Disable repeat mode (RM = 0).
   - Disable loopback mode (DLB = 0).
   - Disable free data format (FDF = 0).
   - Optional: Disable start byte mode if addressing a fully fledged I2C device (STB = 0).
   - Set number of bits to transfer to be 8 bits (BC = 0).
   - Optional: Enable the receive interrupt (ICRINT) by setting ICRRDY = 1 in ICIMR.
5. Specify the slave address of the I2C device that will be accessed using ICSAR. For 7-bit addressing mode, only the first seven bits of ICSAR are used.
6. Configure the peripheral clock operation frequency (ICPSC). This value should be selected in such a way that the frequency is between 6.7 and 13.3 MHz.
7. Configure I2C master clock frequency:
   - Configure the low-time divider value (ICCLKL).
   - Configure the high-time divider value (ICCLKH).
8. Make sure the interrupt status register (ICSTR) and interrupt vector register (ICIVR) are cleared:
   - Read ICSTR and write back the same value (writing a 1 to the bits of ICSTR clears them).
   - Read ICIVR until it is zero.

9. Take I2C controller out of reset by setting IRS = 1 in ICMDR.
10. Wait until bus busy bit is cleared (BB = 0 in ICSTR).
11. Start the transfer by setting STT = 1 in ICMDR. The I2C controller will start the transfer using the 7-bit addressing format as described in Section 9.2.6.1.
12. Wait until data is received.
    - If using polling method, wait until ICRRDY = 1 in ICSTR.
    - If using interrupt method, wait until the I2C controller generates an interrupt, read the ICIVR register to determine if a receive interrupt (ICRINT) has been generated, then clear the interrupt flag by writing a 1 to it.
13. Read the received data from ICDRR.
14. Repeat steps 11 and 12 until last data byte has been received.
15. End transfer/release bus when transfer is done by generating a STOP event (set STP = 1 in ICMDR).

9.2.11.2 Configuring the I2C in Slave Receiver and Transmitter Mode

The following initialization procedure is for the I2C controller configured in Slave Receiver and Transmitter mode.

1. Enable I2C clock from PSC Level. Do this so that you will be able to configure the I2C registers.
2. Place I2C in Reset (Clear IRS bit).
    - ICMDR.IRS=0.
3. Assign the Address (7-bit or 10-bit address) to which the I2C Controller will be responding. This is the Address that the Master is going to broadcast when attempting to start communication with this slave device; I2C Controller.
    - If the I2C is able to respond to 7-bit Addressing: Configure ICMDR.XA=0.
    - If the I2C is able to respond to 10-bit Addressing: Configure ICMDR.XA=1.
    - Program ICOAR=Assigned Address (7-bit or 10-bit address).
4. Enable the desired interrupt you need to receive by setting the desired interrupt bit field within ICIMR to enable the particular Interrupt.
    - ICIMR.AAS=1; Expect an interrupt when Master's Address matches yours (ICOAR programmed value).
    - ICIMR.ICRRDY=1; Expect a receive interrupt when a byte worth data sent from the master is ready to be read.
    - ICIMR.ICXRDY=1; Expect to receive interrupt when the transmit register is ready to be written with a new data that is to be sent to the master.
    - ICIMR.SCD=1; Expect to receive interrupt when Stop Condition is detected.
5. Configure the I2C Controller Operating frequency; this is not the serial clock frequency. This should be between 6.7 and 13.3 MHz. Program IPSC to generate a 6.7 to 13.3 MHz operating frequency.
    - Prescaled Module Clock Frequency = PLL1 Output Frequency / (IPSC + 1).
6. Configure the I2C Serial Clock Frequency. It is advised to configure this frequency to operate at 400 KHz. This will allow the slave device to be able to attend to all Master speeds. Program ICCH and ICCL.
    - 400 KHz = I2C Operating Frequency (6.7-13.3 MHz from Step 5) / [(ICCH+5) + (ICCL+5)].
    - If ICCL==ICCH ≥ 400 KHz = Prescaled Module Clock Frequency / [2×ICCH+10].
7. Configure the Mode Register.
    - ICMDR.MST=0; Configure the I2C Controller to operate as SLAVE.
    - ICMDR.FDF=0; Free Data Format is disabled.
    - ICMDR.BC=0; Set data width to 8 bytes.
    - ICMDR.DLB=0; Disable Loopback Mode.
    - ICMDR.STB=0; I2C Controller can detect Start condition via H/W.
    - ICMDR.RM=1, STP=0, STT=1. See Table 16. (No Activity case).

- Configure remaining bits other than ICMDR.IRS to 0.

8. Release I2C from Reset.
   - ICMDR.IRS=1; Make sure you do not over write your previous configurations.
9. Make sure Interrupt Status Register is cleared.
   - ICSTR=ICSTR; Clear Interrupt fields that require writing '1' requirements.
   - While (ICIVR != 0) Read ICIVR; Read until it is cleared to Zero.
10. Instruct I2C Controller to detect START Condition and Its Own Address.
    - ICMDR.STT=1; Make sure you do not over write your previous configurations.

    MASTER desires to perform a write transfer.
11. If Master requests a Write, i.e, I2C needs to receive data, perform the following:
    - Wait for Receive Interrupt to be received, i.e, ICSTR.ICRRDY=1.
    - Read Data.
12. Perform Step 11 until one of the two happens:
    - Master generates a STOP Condition (ICSTR.STP=1) or
    - I2C Slave desires to end receive transfer.

      If the latter, then the I2C needs to Not Acknowledge the last byte to be received from the Master. After reading the byte prior from the last byte, set NACKMOD bit so that the I2C automatically NACKs the following received data byte, which is the last data byte.
    - ICMDR.NACKMOD=1; set this field on the 2nd data prior from the last.

    Master desires to perform a read transfer.
13. If Master requests a Read, i.e, I2C needs to transmit data, perform the following.
    - Write Data.
    - Wait for Transmit Interrupt to be received, i.e, ICSTR.ICXRDY=1.
14. Perform step 13 until a STOP condition is detected, i.e. (ICSTR.STP=1).

9.2.12 Interrupt Support

The I2C is capable of interrupting the CPU. The CPU can determine which I2C events caused the interrupt by reading the I2C interrupt vector register (ICIVR). ICIVR contains a binary-coded interrupt vector type to indicate which interrupt has occurred. Reading ICIVR clears the interrupt flag; if other interrupts are pending, a new interrupt is generated. If there is more than one pending interrupt flag, reading ICIVR clears the highest-priority interrupt flag.

9.2.12.1 Interrupt Events and Requests

The I2C peripheral can generate the interrupts described in Table 9-3. Each interrupt has a flag bit in the I2C interrupt status register (ICSTR) and a mask bit in the interrupt mask register (ICIMR). When one of the specified events occurs, its flag bit is set. If the corresponding mask bit is 0, the interrupt request is blocked; if the mask bit is 1, the request is forwarded to the CPU as an I2C interrupt.

Table 9-3. Descriptions of the I2C Interrupt Events

I2C Interrupt	Initiating Event
Arbitration-lost interrupt (AL)	Generated when the I2C arbitration procedure is lost or illegal START/STOP conditions occur.
No-acknowledge interrupt (NACK)	Generated when the master I2C does not receive any acknowledge from the receiver.
Registers-ready-for-access interrupt (ARDY)	Generated by the I2C when the previously programmed address, data and command have been performed and the status bits have been updated. This interrupt is used to let the controlling processor know that the I2C registers are ready to be accessed.
Receive interrupt/status (ICRINT and ICRRDY)	Generated when the received data in the receive-shift register (ICRSR) has been copied into the ICDRR. The ICRRDY bit can also be polled by the CPU to read the received data in the ICDRR.

Table 9-3. Descriptions of the I2C Interrupt Events (continued)

I2C Interrupt	Initiating Event
Transmit interrupt/status (ICXINT and ICXRDY)	Generated when the transmitted data has been copied from ICDXR to the transmit-shift register (ICXSR) and shifted out on the SDA pin. This bit can also polled by the CPU to write the next transmitted data into the ICDXR. Note that since ICXINT is generated during the copy of ICDXR to ICXSR, ICXINT will be generated even if no slave acknowledges the value being shifted out of ICXSR.
Stop-Condition-Detection interrupt (SCD)	Generated when a STOP condition has been detected.
Address-as-Slave interrupt (AAS)	Generated when the I2C has recognized its own slave address or an address of all (8) zeros.

9.2.12.2 Interrupt Multiplexing

The I2C interrupt to the CPU is not multiplexed with any other interrupt source.

9.2.13 DMA Events Generated by the I2C Peripheral

The I2C peripheral generates two DMA events. Activity in the DMA controller can be synchronized to these events. Note that out of the four DMA controllers included in the device, only DMA controller 2 (DMA2) can synchronize its activity using the I2C events.

- Receive event (ICREVT): When receive data has been copied from the receive shift register (ICRSR) to the data receive register (ICDRR), the I2C peripheral sends an REVT signal to the DMA controller. In response, the DMA controller can read the data from ICDRR.
- Transmit event (ICXEVT): When transmit data has been copied from the data transmit register (ICDXR) to the transmit shift register (ICXSR), the I2C peripheral sends an XEVT signal to the DMA controller. In response, the DMA controller can write the next transmit data value to ICDXR. Note that since ICXEVT is generated during the copy of ICDXR to ICXSR, ICXEVT will be generated even if no slave acknowledges the value being shifted out of ICXSR.

9.2.14 Power Management

There are several ways to reduce the power consumption of the I2C peripheral. First, the I2C peripheral can be clock-gated to conserve power during periods of no activity. The I2C peripheral clock can be turned off by using the peripheral clock gating configuration register (PCGCR). Second, the I2C peripheral clock and output clock (SCL) can be reduced to the minimal possible value allowed by the system. As described in Section 9.2.2, Clock Generation, these two clocks are controlled through the ICPSC, ICCLKH, ICCLKL registers. Note that the I2C peripheral clock must be maintained between 7 and 12 MHz. For detailed information on PCGCR, see the device-specific data manual.

9.2.15 Emulation Considerations

The response of the I2C events to emulation suspend events (such as halts and breakpoints) is controlled by the FREE bit in the I2C mode register (ICMDR). The I2C peripheral either stops exchanging data (FREE = 0) or continues to run (FREE = 1) when an emulation suspend event occurs. How the I2C peripheral terminates data transactions is affected by whether the I2C peripheral is acting as a master or a slave. For more information, see the description of the FREE bit in ICMDR (see Section 9.3.9).

9.3 Registers

Table 9-4 lists the memory-mapped registers for the inter-integrated circuit (I2C) peripheral. The I2C registers can be accessed by the CPU and DMA controller at the absolute 16-bit addresses specified in Table 9-4. Note that the CPU accesses all peripheral registers through its I/O space. All other register addresses not listed in Table 9-4 should be considered as reserved locations and the register contents should not be modified.

Table 9-4. Inter-Integrated Circuit (I2C) Registers

CPU Word Address	Acronym	Register Description	Section
1A00h	ICOAR	I2C Own Address Register	Section 9.3.1
1A04h	ICIMR	I2C Interrupt Mask Register	Section 9.3.2
1A08h	ICSTR	I2C Interrupt Status Register	Section 9.3.3
1A0Ch	ICCLKL	I2C Clock Low-Time Divider Register	Section 9.3.4
1A10h	ICCLKH	I2C Clock High-Time Divider Register	Section 9.3.4
1A14h	ICCNT	I2C Data Count Register	Section 9.3.5
1A18h	ICDRR	I2C Data Receive Register	Section 9.3.6
1A1Ch	ICSAR	I2C Slave Address Register	Section 9.3.7
1A20h	ICDXR	I2C Data Transmit Register	Section 9.3.8
1A24h	ICMDR	I2C Mode Register	Section 9.3.9
1A28h	ICIVR	I2C Interrupt Vector Register	Section 9.3.10
1A2Ch	ICEMDR	I2C Extended Mode Register	Section 9.3.11
1A30h	ICPSC	I2C Prescaler Register	Section 9.3.12
1A34h	ICPID1	I2C Peripheral Identification Register 1	Section 9.3.13
1A38h	ICPID2	I2C Peripheral Identification Register 2	Section 9.3.13

9.3.1 *I2C Own Address Register (ICOAR)*

The I2C own address register (ICOAR) is used to specify its own slave address, which distinguishes it from other slaves connected to the I2C-bus. If the 7-bit addressing mode is selected (XA = 0 in ICMDR), only bits 6-0 are used; bits 9-7 are ignored.

The I2C own address register (ICOAR) is shown in Figure 9-13 and described in Table 9-5.

Figure 9-13. I2C Own Address Register (ICOAR)

15 — 10	9 — 0
Reserved	OADDR
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 9-5. I2C Own Address Register (ICOAR) Field Descriptions

Bit	Field	Value	Description
15-10	Reserved	0	These reserved bit locations are always read as zeros. A value written to this field has no effect.
9-0	OADDR	0-3FFh	Own slave address. Provides the slave address of the I2C. In 7-bit addressing mode (XA = 0 in ICMDR): bits 6-0 provide the 7-bit slave address of the I2C. Bits 9-7 are ignored. In 10-bit addressing mode (XA = 1 in ICMDR): bits 9-0 provide the 10-bit slave address of the I2C.

9.3.2 I2C Interrupt Mask Register (ICIMR)

The I2C interrupt mask register (ICIMR) is used to individually enable or disable I2C interrupt requests.

The I2C interrupt mask register (ICIMR) is shown in Figure 9-14 and described Table 9-6.

Figure 9-14. I2C Interrupt Mask Register (ICIMR)

15							8
Reserved							
R-0							

7	6	5	4	3	2	1	0
Reserved	AAS	SCD	ICXRDY	ICRRDY	ARDY	NACK	AL
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 9-6. I2C Interrupt Mask Register (ICIMR) Field Descriptions

Bit	Field	Value	Description
15-7	Reserved	0	These reserved bit locations are always read as zeros. A value written to this field has no effect.
6	AAS		Address-as-slave interrupt enable bit.
		0	Interrupt request is disabled.
		1	Interrupt request is enabled.
5	SCD		Stop condition detected interrupt enable bit.
		0	Interrupt request is disabled.
		1	Interrupt request is enabled.
4	ICXRDY		Transmit-data-ready interrupt enable bit.
		0	Interrupt request is disabled.
		1	Interrupt request is enabled.
3	ICRRDY		Receive-data-ready interrupt enable bit.
		0	Interrupt request is disabled.
		1	Interrupt request is enabled.
2	ARDY		Register-access-ready interrupt enable bit.
		0	Interrupt request is disabled.
		1	Interrupt request is enabled.
1	NACK		No-acknowledgment interrupt enable bit.
		0	Interrupt request is disabled.
		1	Interrupt request is enabled.
0	AL		Arbitration-lost interrupt enable bit
		0	Interrupt request is disabled.
		1	Interrupt request is enabled.

9.3.3 I2C Interrupt Status Register (ICSTR)

The I2C interrupt status register (ICSTR) is used to determine which interrupt has occurred and to read status information.

The I2C interrupt status register (ICSTR) is shown in Figure 9-15 and described in Table 9-7.

Figure 9-15. I2C Interrupt Status Register (ICSTR)

15	14	13	12	11	10	9	8
Reserved	SDIR	NACKSNT	BB	RSFULL	XSMT	AAS	AD0
R-0	R/W1C-0	R/W1C-0	R/W1C-0	R-0	R-1	R-0	R-0

7	6	5	4	3	2	1	0
Reserved		SCD	ICXRDY	ICRRDY	ARDY	NACK	AL
R-0		R/W1C-0	R/W1C-1	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0

LEGEND: R/W = Read/Write; R = Read only; W1C = Write 1 to clear (writing 0 has no effect); -*n* = value after reset

Table 9-7. I2C Interrupt Status Register (ICSTR) Field Descriptions

Bit	Field	Value	Description
15	Reserved	0	These reserved bit locations are always read as zeros. A value written to this field has no effect.
14	SDIR		Slave direction bit. In digital-loopback mode (DLB), the SDIR bit is cleared to 0.
		0	I2C is acting as a master-transmitter/receiver or a slave-receiver. SDIR is cleared by one of the following events: • A STOP or a START condition. • SDIR is manually cleared. To clear this bit, write a 1 to it.
		1	I2C is acting as a slave-transmitter.
13	NACKSNT		No-acknowledgment sent bit. NACKSNT bit is used when the I2C is in the receiver mode. One instance in which NACKSNT is affected is when the NACK mode is used (see the description for NACKMOD in Section 9.3.9).
		0	NACK is not sent. NACKSNT is cleared by one of the following events: • It is manually cleared. To clear this bit, write a 1 to it. • The I2C is reset (either when 0 is written to the IRS bit of ICMDR or when the processor is reset).
		1	NACK is sent. A no-acknowledge bit was sent during the acknowledge cycle on the I2C-bus.
12	BB		Bus busy bit. BB bit indicates whether the I2C-bus is busy or is free for another data transfer. In the master mode, BB is controlled by the software.
		0	Bus is free. BB is cleared by one of the following events: • The I2C receives or transmits a STOP bit (bus free). • BB is manually cleared. To clear this bit, write a 1 to it. • The I2C is reset (either when 0 is written to the IRS bit of ICMDR or when the processor is reset).
		1	Bus is busy. When the STT bit in ICMDR is set to 1, a restart condition is generated. BB is set by one of the following events: • The I2C has received or transmitted a START bit on the bus. • SCL is in a low state and the IRS bit in ICMDR is 0.
11	RSFULL		Receive shift register full bit. RSFULL indicates an overrun condition during reception. Overrun occurs when the receive shift register (ICRSR) is full with new data but the previous data has not been read from the data receive register (ICDRR). The new data will not be copied to ICDRR until the previous data is read. As new bits arrive from the SDA pin, they overwrite the bits in ICRSR.
		0	No overrun is detected. RSFULL is cleared by one of the following events: • ICDRR is read. • The I2C is reset (either when 0 is written to the IRS bit of ICMDR or when the processor is reset).
		1	Overrun is detected.

Table 9-7. I2C Interrupt Status Register (ICSTR) Field Descriptions (continued)

Bit	Field	Value	Description
10	XSMT		Transmit shift register empty bit. XSMT indicates that the transmitter has experienced underflow. Underflow occurs when the transmit shift register (ICXSR) is empty but the data transmit register (ICDXR) has not been loaded since the last ICDXR-to-ICXSR transfer. The next ICDXR-to-ICXSR transfer will not occur until new data is in ICDXR. If new data is not transferred in time, the previous data may be re-transmitted on the SDA pin.
		0	Underflow is detected.
		1	No underflow is detected. XSMT is set by one of the following events: • Data is written to ICDXR. • The I2C is reset (either when 0 is written to the IRS bit of ICMDR or when the processor is reset).
9	AAS		Addressed-as-slave bit.
		0	The AAS bit has been cleared by a repeated START condition or by a STOP condition.
		1	AAS is set by one of the following events: • I2C has recognized its own slave address or an address of all zeros (general call). • The first data word has been received in the free data format (FDF = 1 in ICMDR).
8	AD0		Address 0 bit.
		0	AD0 has been cleared by a START or STOP condition.
		1	An address of all zeros (general call) is detected.
7-6	Reserved	0	These reserved bit locations are always read as zeros. A value written to this field has no effect.
5	SCD		Stop condition detected bit. SCD indicates when a STOP condition has been detected on the I2C bus. The STOP condition could be generated by the I2C or by another I2C device connected to the bus.
		0	No STOP condition has been detected. SCD is cleared by one of the following events: • By reading the INTCODE bits in ICIVR as 110b. • SCD is manually cleared. To clear this bit, write a 1 to it.
		1	A STOP condition has been detected.
4	ICXRDY		Transmit-data-ready interrupt flag bit. ICXRDY indicates that the data transmit register (ICDXR) is ready to accept new data because the previous data has been copied from ICDXR to the transmit shift register (ICXSR). The CPU can poll ICXRDY or use the XRDY interrupt request.
		0	ICDXR is not ready. ICXRDY is cleared by one of the following events: • Data is written to ICDXR. • ICXRDY is manually cleared. To clear this bit, write a 1 to it.
		1	ICDXR is ready. Data has been copied from ICDXR to ICXSR. ICXRDY is forced to 1 when the I2C is reset.
3	ICRRDY		Receive-data-ready interrupt flag bit. ICRRDY indicates that the data receive register (ICDRR) is ready to be read because data has been copied from the receive shift register (ICRSR) to ICDRR. The CPU can poll ICRRDY or use the RRDY interrupt request.
		0	ICDRR is not ready. ICRRDY is cleared by one of the following events: • ICDRR is read. • ICRRDY is manually cleared. To clear this bit, write a 1 to it. • The I2C is reset (either when 0 is written to the IRS bit of ICMDR or when the processor is reset).
		1	ICDRR is ready. Data has been copied from ICRSR to ICDRR.

Table 9-7. I2C Interrupt Status Register (ICSTR) Field Descriptions (continued)

Bit	Field	Value	Description
2	ARDY		Register-access-ready interrupt flag bit (only applicable when the I2C is in the master mode). ARDY indicates that the I2C registers are ready to be accessed because the previously programmed address, data, and command values have been used. The CPU can poll ARDY or use the ARDY interrupt request.
		0	The registers are not ready to be accessed. ARDY is cleared by one of the following events: • The I2C starts using the current register contents. • ARDY is manually cleared. To clear this bit, write a 1 to it. • The I2C is reset (either when 0 is written to the IRS bit of ICMDR or when the processor is reset).
		1	The registers are ready to be accessed. This bit is set after the slave address appears on the I2C bus. • In the nonrepeat mode (RM = 0 in ICMDR): If STP = 0 in ICMDR, ARDY is set when the internal data counter counts down to 0. If STP = 1, ARDY is not affected (instead, the I2C generates a STOP condition when the counter reaches 0). • In the repeat mode (RM = 1): ARDY is set at the end of each data word transmitted from ICDXR.
1	NACK		No-acknowledgment interrupt flag bit. NACK applies when the I2C is a transmitter (master or slave). NACK indicates whether the I2C has detected an acknowledge bit (ACK) or a no-acknowledge bit (NACK) from the receiver. The CPU can poll NACK or use the NACK interrupt request.
		0	ACK received/NACK is not received. NACK is cleared by one of the following events: • An acknowledge bit (ACK) has been sent by the receiver. • NACK is manually cleared. To clear this bit, write a 1 to it. • The CPU reads the interrupt source register (ICISR) when the register contains the code for a NACK interrupt. • The I2C is reset (either when 0 is written to the IRS bit of ICMDR or when the processor is reset).
		1	NACK bit is received. The hardware detects that a no-acknowledge (NACK) bit has been received. **Note:** While the I2C performs a general call transfer, NACK is 1, even if one or more slaves send acknowledgment.
0	AL		Arbitration-lost interrupt flag bit (only applicable when the I2C is a master-transmitter). AL primarily indicates when the I2C has lost an arbitration contest with another master-transmitter. The CPU can poll AL or use the AL interrupt request.
		0	Arbitration is not lost. AL is cleared by one of the following events: • AL is manually cleared. To clear this bit, write a 1 to it. • The CPU reads the interrupt source register (ICISR) when the register contains the code for an AL interrupt. • The I2C is reset (either when 0 is written to the IRS bit of ICMDR or when the processor is reset).
		1	Arbitration is lost. AL is set by one of the following events: • The I2C senses that it has lost an arbitration with two or more competing transmitters that started a transmission almost simultaneously. • The I2C attempts to start a transfer while the BB (bus busy) bit is set to 1. When AL is set to 1, the MST and STP bits of ICMDR are cleared, and the I2C becomes a slave-receiver.

9.3.4 I2C Clock Divider Registers (ICCLKL and ICCLKH)

When the I2C is a master, the prescaled module clock is divided down for use as the I2C serial clock on the SCL pin. The shape of the I2C serial clock depends on two divide-down values, ICCL and ICCH. For detailed information on how these values are programmed, see Section 9.2.2.

9.3.4.1 I2C Clock Low-Time Divider Register (ICCLKL)

For each I2C serial clock cycle, ICCL determines the amount of time the signal is low. ICCLKL must be configured while the I2C is still in reset (IRS = 0 in ICMDR).

The I2C clock low-time divider register (ICCLKL) is shown in Figure 9-16 and described in Table 9-8.

Figure 9-16. I2C Clock Low-Time Divider Register (ICCLKL)

15	0
ICCL	
R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 9-8. I2C Clock Low-Time Divider Register (ICCLKL) Field Descriptions

Bit	Field	Value	Description
15-0	ICCL	0-FFFFh	Clock low-time divide-down value of 1-65536. The period of the module clock is multiplied by (ICCL + d) to produce the low-time duration of the I2C serial on the SCL pin.

9.3.4.2 I2C Clock High-Time Divider Register (ICCLKH)

For each I2C serial clock cycle, ICCH determines the amount of time the signal is high. ICCLKH must be configured while the I2C is still in reset (IRS = 0 in ICMDR).

The I2C clock high-time divider register (ICCLKH) is shown in Figure 9-17 and described in Table 9-9.

Figure 9-17. I2C Clock High-Time Divider Register (ICCLKH)

15	0
ICCH	
R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 9-9. I2C Clock High-Time Divider Register (ICCLKH) Field Descriptions

Bit	Field	Value	Description
15-0	ICCH	0-FFFFh	Clock high-time divide-down value of 1-65536. The period of the module clock is multiplied by (ICCH + d) to produce the high-time duration of the I2C serial on the SCL pin.

9.3.5 I2C Data Count Register (ICCNT)

The I2C data count register (ICCNT) is used to indicate how many data words to transfer when the I2C is configured as a master-transmitter (MST = 1 and TRX = 1 in ICMDR) and the repeat mode is off (RM = 0 in ICMDR). In the repeat mode (RM = 1), ICCNT is not used.

The value written to ICCNT is copied to an internal data counter. The internal data counter is decremented by 1 for each data word transferred (ICCNT remains unchanged). If a STOP condition is requested (STP = 1 in ICMDR), the I2C terminates the transfer with a STOP condition when the countdown is complete (that is, when the last data word has been transferred).

The data count register (ICCNT) is shown in Figure 9-18 and described in Table 9-10.

Figure 9-18. I2C Data Count Register (ICCNT)

15-0
ICDC
R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 9-10. I2C Data Count Register (ICCNT) Field Descriptions

Bit	Field	Value	Description
15-0	ICDC	0-FFFFh	Data count value. When RM = 0 in ICMDR, ICDC indicates the number of data words to transfer in the nonrepeat mode. When RM = 1 in ICMDR, the value in ICCNT is a don't care. If STP = 1 in ICMDR, a STOP condition is generated when the internal data counter counts down to 0.
		0	The start value loaded to the internal data counter is 65536.
		1h-FFFFh	The start value loaded to internal data counter is 1-65535.

9.3.6 I2C Data Receive Register (ICDRR)

The I2C data receive register (ICDRR) is used to read the receive data. The ICDRR can receive a data value of up to 8 bits; data values with fewer than 8 bits are right-aligned in the D bits and the remaining D bits are undefined. The number of data bits is selected by the bit count bits (BC) of ICMDR. The I2C receive shift register (ICRSR) shifts in the received data from the SDA pin. Once data is complete, the I2C copies the contents of ICRSR into ICDRR. The CPU and the DMA controller cannot access ICRSR.

The I2C data receive register (ICDRR) is shown in Figure 9-19 and described in Table 9-11.

Figure 9-19. I2C Data Receive Register (ICDRR)

15-8	7-0
Reserved	D
R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 9-11. I2C Data Receive Register (ICDRR) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	These reserved bit locations are always read as zeros. A value written to this field has no effect.
7-0	D	0-FFh	Receive data.

9.3.7 I2C Slave Address Register (ICSAR)

The I2C slave address register (ICSAR) contains a 7-bit or 10-bit slave address. When the I2C is not using the free data format (FDF = 0 in ICMDR), it uses this address to initiate data transfers with a slave or slaves. When the address is nonzero, the address is for a particular slave. When the address is 0, the address is a general call to all slaves. If the 7-bit addressing mode is selected (XA = 0 in ICMDR), only bits 6-0 of ICSAR are used; bits 9-7 are ignored. The I2C slave address register (ICSAR) is shown in Figure 9-20 and described in Table 9-12.

Figure 9-20. I2C Slave Address Register (ICSAR)

15 10	9 0
Reserved	SADDR
R-0	R/W-3FFh

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 9-12. I2C Slave Address Register (ICSAR) Field Descriptions

Bit	Field	Value	Description
15-10	Reserved	0	These reserved bit locations are always read as zeros. A value written to this field has no effect.
9-0	SADDR	0-3FFh	Slave address. Provides the slave address that the I2C transmits when it is in master-transmitter mode. In 7-bit addressing mode (XA = 0 in ICMDR): bits 6-0 provide the 7-bit slave address that the I2C transmits when it is in the master-transmitter mode. Bits 9-7 are ignored. In 10-bit addressing mode (XA = 1 in ICMDR): Bits 9-0 provide the 10-bit slave address that the I2C transmits when it is in the master-transmitter mode.

9.3.8 I2C Data Transmit Register (ICDXR)

The CPU or DMA writes transmit data to the I2C data transmit register (ICDXR). The ICDXR can accept a data value of up to 8 bits. When writing a data value with fewer than 8 bits, the written data must be right-aligned in the D bits. The number of data bits is selected by the bit count bits (BC) of ICMDR. Once data is written to ICDXR, the I2C copies the contents of ICDXR into the I2C transmit shift register (ICXSR). The ICXSR shifts out the transmit data from the SDA pin. The CPU and the DMA controller cannot access ICXSR.

The I2C data transmit register (ICDXR) is shown in Figure 9-21 and described in Table 9-13.

Figure 9-21. I2C Data Transmit Register (ICDXR)

15 8	7 0
Reserved	D
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 9-13. I2C Data Transmit Register (ICDXR) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	These reserved bit locations are always read as zeros. A value written to this field has no effect.
7-0	D	0-FFh	Transmit data.

9.3.9 I2C Mode Register (ICMDR)

The I2C mode register (ICMDR) contains the control bits of the I2C.

The I2C mode register (ICMDR) is shown in shown in Figure 9-22 and described in Table 9-14.

Figure 9-22. I2C Mode Register (ICMDR)

15	14	13	12	11	10	9	8
NACKMOD	FREE	STT	Reserved	STP	MST	TRX	XA
R/W-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2		0
RM	DLB	IRS	STB	FDF	BC		
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0		

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 9-14. I2C Mode Register (ICMDR) Field Descriptions

Bit	Field	Value	Description
15	NACKMOD		No-acknowledge (NACK) mode bit (only applicable when the I2C is a receiver).
		0	In slave-receiver mode: The I2C sends an acknowledge (ACK) bit to the transmitter during the each acknowledge cycle on the bus. The I2C only sends a no-acknowledge (NACK) bit if you set the NACKMOD bit. In master-receiver mode: The I2C sends an ACK bit during each acknowledge cycle until the internal data counter counts down to 0. When the counter reaches 0, the I2C sends a NACK bit to the transmitter. To have a NACK bit sent earlier, you must set the NACKMOD bit.
		1	In either slave-receiver or master-receiver mode: The I2C sends a NACK bit to the transmitter during the next acknowledge cycle on the bus. Once the NACK bit has been sent, NACKMOD is cleared. To send a NACK bit in the next acknowledge cycle, you must set NACKMOD before the rising edge of the last data bit.
14	FREE		This emulation mode bit is used to determine the state of the I2C when a breakpoint is encountered in the high-level language debugger.
		0	When I2C is master: If SCL is low when the breakpoint occurs, the I2C stops immediately and keeps driving SCL low, whether the I2C is the transmitter or the receiver. If SCL is high, the I2C waits until SCL becomes low and then stops. When I2C is slave: A breakpoint forces the I2C to stop when the current transmission/reception is complete.
		1	The I2C runs free; that is, it continues to operate when a breakpoint occurs.
13	STT		START condition bit (only applicable when the I2C is a master). The RM, STT, and STP bits determine when the I2C starts and stops data transmissions (see Table 9-15). Note that the STT and STP bits can be used to terminate the repeat mode.
		0	In master mode, STT is automatically cleared after the START condition has been generated. In slave mode, if STT is 0, the I2C does not monitor the bus for commands from a master. As a result, the I2C performs no data transfers.
		1	In master mode, setting STT to 1 causes the I2C to generate a START condition on the I2C-bus. In slave mode, if STT is 1, the I2C monitors the bus and transmits/receives data in response to commands from a master.
12	Reserved	0	These reserved bit locations are always read as zeros. A value written to this field has no effect.
11	STP		STOP condition bit (only applicable when the I2C is a master). The RM, STT, and STP bits determine when the I2C starts and stops data transmissions (see Table 9-15). Note that the STT and STP bits can be used to terminate the repeat mode.
		0	STP is automatically cleared after the STOP condition has been generated.
		1	STP has been set to generate a STOP condition when the internal data counter of the I2C counts down to 0.
10	MST		Master mode bit. MST determines whether the I2C is in the slave mode or the master mode. MST is automatically changed from 1 to 0 when the I2C master generates a STOP condition. See Table 9-16.
		0	Slave mode. The I2C is a slave and receives the serial clock from the master.
		1	Master mode. The I2C is a master and generates the serial clock on the SCL pin.

Table 9-14. I2C Mode Register (ICMDR) Field Descriptions (continued)

Bit	Field	Value	Description
9	TRX		Transmitter mode bit. When relevant, TRX selects whether the I2C is in the transmitter mode or the receiver mode. Table 9-16 summarizes when TRX is used and when it is a don't care.
		0	Receiver mode. The I2C is a receiver and receives data on the SDA pin.
		1	Transmitter mode. The I2C is a transmitter and transmits data on the SDA pin.
8	XA		Expanded address enable bit.
		0	7-bit addressing mode (normal address mode). The I2C transmits 7-bit slave addresses (from bits 6-0 of ICSAR), and its own slave address has 7 bits (bits 6-0 of ICOAR).
		1	10-bit addressing mode (expanded address mode). The I2C transmits 10-bit slave addresses (from bits 9-0 of ICSAR), and its own slave address has 10 bits (bits 9-0 of ICOAR).
7	RM		Repeat mode bit (only applicable when the I2C is a master-transmitter). The RM, STT, and STP bits determine when the I2C starts and stops data transmissions (see Table 9-15). If the I2C is configured in slave mode, the RM bit is don't care.
		0	Nonrepeat mode. The value in the data count register (ICCNT) determines how many data words are received/transmitted by the I2C.
		1	Repeat mode. Data words are continuously received/transmitted by the I2C regardless of the value in ICCNT until the STP bit is manually set to 1.
6	DLB		Digital loopback mode bit (only applicable when the I2C is a master-transmitter). This bit disables or enables the digital loopback mode of the I2C. The effects of this bit are shown in Figure 9-23. Note that DLB mode in the free data format mode (DLB = 1 and FDF = 1) is not supported.
		0	Digital loopback mode is disabled.
		1	Digital loopback mode is enabled. In this mode, the MST bit must be set to 1 and data transmitted out of ICDXR is received in ICDRR after n clock cycles by an internal path, where: n = ((I2C input clock frequency/prescaled module clock frequency) × 8) The transmit clock is also the receive clock. The address transmitted on the SDA pin is the address in ICOAR.
5	IRS		I2C reset bit. Note that if IRS is reset during a transfer, it can cause the I2C bus to hang (SDA and SCL are in a high-impedance state).
		0	The I2C is in reset/disabled. When this bit is cleared to 0, all status bits (in ICSTR) are set to their default values.
		1	The I2C is enabled.
4	STB		START byte mode bit (only applicable when the I2C is a master). As described in version 2.1 of the Philips I2C-bus specification, the START byte can be used to help a slave that needs extra time to detect a START condition. When the I2C is a slave, the I2C ignores a START byte from a master, regardless of the value of the STB bit.
		0	The I2C is not in the START byte mode.
		1	The I2C is in the START byte mode. When you set the START condition bit (STT), the I2C begins the transfer with more than just a START condition. Specifically, it generates: 1. A START condition 2. A START byte (0000 0001b) 3. A dummy acknowledge clock pulse 4. A repeated START condition The I2C sends the slave address that is in ICSAR.
3	FDF		Free data format mode bit. Note that DLB mode in the free data format mode (DLB = 1 and FDF = 1) is not supported. See Table 9-16.
		0	Free data format mode is disabled. Transfers use the 7-/10-bit addressing format selected by the XA bit.
		1	Free data format mode is enabled.

Table 9-14. I2C Mode Register (ICMDR) Field Descriptions (continued)

Bit	Field	Value	Description
2-0	BC	0-7h	Bit count bits. BC defines the number of bits (2 to 8) in the next data word that is to be received or transmitted by the I2C. The number of bits selected with BC must match the data size of the other device. Note that when BC = 0, a data word has 8 bits. If the bit count is less than 8, receive data is right aligned in the D bits of ICDRR and the remaining D bits are undefined. Also, transmit data written to ICDXR must be right aligned.
		0	8 bits per data word
		1h	Reserved
		2h	2 bits per data word
		3h	3 bits per data word
		4h	4 bits per data word
		5h	5 bits per data word
		6h	6 bits per data word
		7h	7 bits per data word

Table 9-15. Master-Transmitter/Receiver Bus Activity Defined by RM, STT, and STP Bits

ICMDR Bit				
RM	STT	STP	Bus Activity [1]	Description
0	0	0	None	No activity
0	0	1	P	STOP condition
0	1	0	S-A-D..(*n*)..D	START condition, slave address, *n* data words (*n* = value in ICCNT)
0	1	1	S-A-D..(*n*)..D-P	START condition, slave address, *n* data words, STOP condition (*n* = value in ICCNT)
1	0	0	None	No activity
1	0	1	P	STOP condition
1	1	0	S-A-D-D-D..	Repeat mode transfer: START condition, slave address, continuous data transfers until STOP condition or next START condition
1	1	1	None	Reserved bit combination (No activity

[1] A = Address; D = Data word; P = STOP condition; S = START condition

Table 9-16. How the MST and FDF Bits Affect the Role of TRX Bit

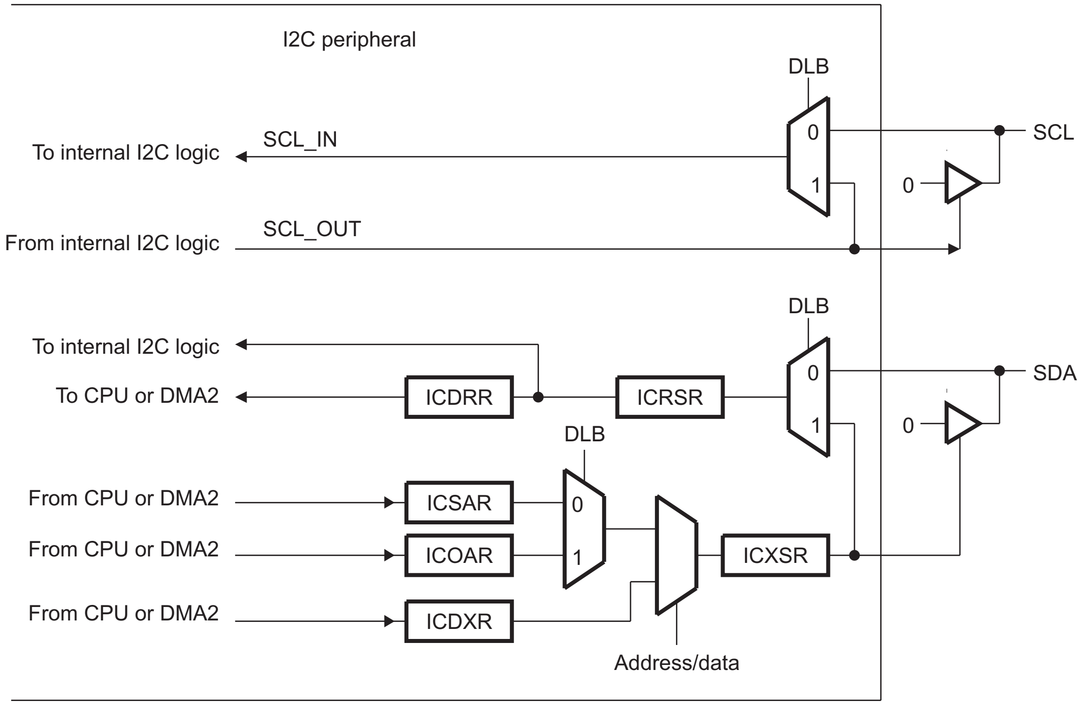
ICMDR Bit			
MST	**FDF**	**I2C State**	**Function of TRX Bit**
0	0	In slave mode but not free data format mode	TRX is a don't care. Depending on the command from the master, the I2C responds as a receiver or a transmitter.

0	1	In slave mode and free data format mode	The free data format mode requires that the transmitter and receiver be fixed. TRX identifies the role of the I2C:
			TRX = 0: The I2C is a receiver.
			TRX = 1: The I2C is a transmitter.

1	0	In master mode but not free data format mode	TRX identifies the role of the I2C:
			TRX = 0: The I2C is a receiver.
			TRX = 1: The I2C is a transmitter.

1	1	In master mode and free data format mode	The free data format mode requires that the transmitter and receiver be fixed. TRX identifies the role of the I2C:
			TRX = 0: The I2C is a receiver.
			TRX = 1: The I2C is a transmitter.

Figure 9-23. Block Diagram Showing the Effects of the Digital Loopback Mode (DLB) Bit



9.3.10 I2C Interrupt Vector Register (ICIVR)

The I2C interrupt vector register (ICIVR) is used by the CPU to determine which event generated the I2C interrupt. Reading ICIVR clears the interrupt flag; if other interrupts are pending, a new interrupt is generated. If there are more than one interrupt flag, reading ICIVR clears the highest priority interrupt flag. Note that you must read (clear) ICIVR before doing another start; otherwise, ICIVR could contain an incorrect (old interrupt flags) value.

The I2C interrupt vector register (ICIVR) is shown in Figure 9-24 and described in Table 9-17.

Figure 9-24. I2C Interrupt Vector Register (ICIVR)

15 3	2 0
Reserved	INTCODE
R-0	R-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 9-17. I2C Interrupt Vector Register (ICIVR) Field Descriptions

Bit	Field	Value	Description
15-3	Reserved	0	These reserved bit locations are always read as zeros. A value written to this field has no effect.
2-0	INTCODE	0-7h	Interrupt code bits. The binary code in INTCODE indicates which event generated an I2C interrupt.
		0	None
		1h	Arbitration-lost interrupt (AL)
		2h	No-acknowledgment interrupt (NACK)
		3h	Register-access-ready interrupt (ARDY)
		4h	Receive-data-ready interrupt (ICRRDY)
		5h	Transmit-data-ready interrupt (ICXRDY)
		6h	Stop condition detected interrupt (SCD)
		7h	Address-as-slave interrupt (AAS)

9.3.11 I2C Extended Mode Register (ICEMDR)

The I2C extended mode register (ICEMDR) is used to indicate which condition generates a transmit data ready interrupt.

The I2C extended mode register (ICEMDR) is shown in Figure 9-25 and described in Table 9-18.

Figure 9-25. I2C Extended Mode Register (ICEMDR)

15	2	1	0
Reserved		IGNACK	BCM
R-0		R/W-0	R/W-1

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 9-18. I2C Extended Mode Register (ICEMDR) Field Descriptions

Bit	Field	Value	Description
15-2	Reserved	0	These reserved bit locations are always read as zeros. A value written to this field has no effect.
1	IGNACK		Ignore NACK mode.
		0	Master transmitter operates normally, that is, it discontinues the data transfer and sets the ARDY and NACK bits in ICSTR when receiving a NACK from the slave.
		1	Master transmitter ignores a NACK from the slave.
0	BCM		Backward compatibility mode bit. Determines which condition generates a transmit data ready interrupt. The BCM bit only has an effect when the I2C is operating as a slave-transmitter.
		0	The transmit data ready interrupt is generated when the master requests more data by sending an acknowledge signal after the transmission of the last data.
		1	The transmit data ready interrupt is generated when the data in ICDXR is copied to ICXSR.

9.3.12 I2C Prescaler Register (ICPSC)

The I2C prescaler register (ICPSC) is used for dividing down the I2C input clock to obtain the desired prescaled module clock for the operation of the I2C.

The IPSC bits must be initialized while the I2C is in reset (IRS = 0 in ICMDR). The prescaled frequency takes effect only when the IRS bit is changed to 1. Changing the IPSC value while IRS = 1 has no effect.

The I2C prescaler register (ICPSC) is shown in Figure 9-26 and described in Table 9-19.

Figure 9-26. I2C Prescaler Register (ICPSC)

15 8	7 0
Reserved	IPSC
R-0	R/W-0

LEGEND: R = Read only; -*n* = value after reset

Table 9-19. I2C Prescaler Register (ICPSC) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	These reserved bit locations are always read as zeros. A value written to this field has no effect.
7-0	IPSC	0-FFh	I2C prescaler divide-down value. IPSC determines how much the I2C input clock is divided to create the I2C prescaled module clock: I2C clock frequency = I2C input clock frequency/(IPSC + 1) **Note:** IPSC must be initialized while the I2C is in reset (IRS = 0 in ICMDR).

9.3.13 I2C Peripheral Identification Register (ICPID1)

The I2C peripheral identification registers (ICPID1) contain identification data (class, revision, and type) for the peripheral.

The I2C peripheral identification register (ICPID1) is shown in Figure 9-27 and described in Table 9-20.

Figure 9-27. I2C Peripheral Identification Register 1 (ICPID1)

<table>
<tr><td>15</td><td>8</td><td>7</td><td>0</td></tr>
<tr><td colspan="2">Class</td><td colspan="2">Revision</td></tr>
<tr><td colspan="2">R-01h</td><td colspan="2">R-06h</td></tr>
</table>

LEGEND: R = Read only; -*n* = value after reset

Table 9-20. I2C Peripheral Identification Register 1 (ICPID1) Field Descriptions

Bit	Field	Value	Description
15-8	Class		Identifies class of peripheral.
		01h	Serial port
7-0	Revision		Identifies revision of peripheral.
		06h	Current revision of peripheral.

9.3.14 I2C Peripheral Identification Register (ICPID2)

The I2C peripheral identification register (ICPID2) is shown in Figure 9-28 and described in Table 9-21.

Figure 9-28. I2C Peripheral Identification Register 2 (ICPID2)

<table>
<tr><td>15</td><td>8</td><td>7</td><td>0</td></tr>
<tr><td colspan="2">Reserved</td><td colspan="2">TYPE</td></tr>
<tr><td colspan="2">R-0</td><td colspan="2">R-05h</td></tr>
</table>

LEGEND: R = Read only; -*n* = value after reset

Table 9-21. I2C Peripheral Identification Register 2 (ICPID2) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	These reserved bit locations are always read as zeros. A value written to this field has no effect.
7-0	TYPE		Identifies type of peripheral.
		05h	I2C

Chapter 10

SPRUH87C – August 2011 – Revised March 2012

Inter-IC Sound (I2S) Bus

This chapter describes the features and operation of Inter-IC Sound (I2S) Bus.

Topic **Page**

10.1 Introduction

This document describes the features and operation of Inter-IC Sound (I2S) Bus. This peripheral allows serial transfer of full duplex streaming data, usually streaming audio, between DSP and an external I2S peripheral such as an audio codec.

10.1.1 Purpose of the Peripheral

The I2S bus is used as an interface for full-duplex serial ports such as those found in audio or voice-band analog to digital converters (ADC) to acquire audio signals or digital-to analog converters (DAC) to drive speakers and headphones.

10.1.2 Features

The I2S bus supports the following features:

- Full-duplex (transmit and receive) communication.
- Double buffered data registers that allow for continuous data stream.
- Most significant bit (MSB) - first data transfers.
- I2S/Left-justified and DSP serial data communication formats with a data delay of 1 or 2 bits.
- Data word-lengths of 8, 10, 12, 14, 16, 18, 20, 24, or 32 bits.
- Ability to sign-extend received data samples for easy use in signal processing algorithms.
- Ability to pack multiple data words into CPU or DMA accessible data registers to reduce interrupts for more efficient operation.
- Programmable polarity for both frame synchronization and bit-serial clocks.
- Digital loopback of data from transmit to receive data register(s) for application code debug.
- Stereo (in I2S/Left-justified or DSP data formats) or mono (in DSP data format) mode.
- Programmable divider for serial data clock (bit-clock) generation when I2S bus is used as a master device.
- Programmable divider for frame sync clock generation when I2S bus is used as the master device.
- Detection of over-run, under-run, and frame-synchronization error conditions.

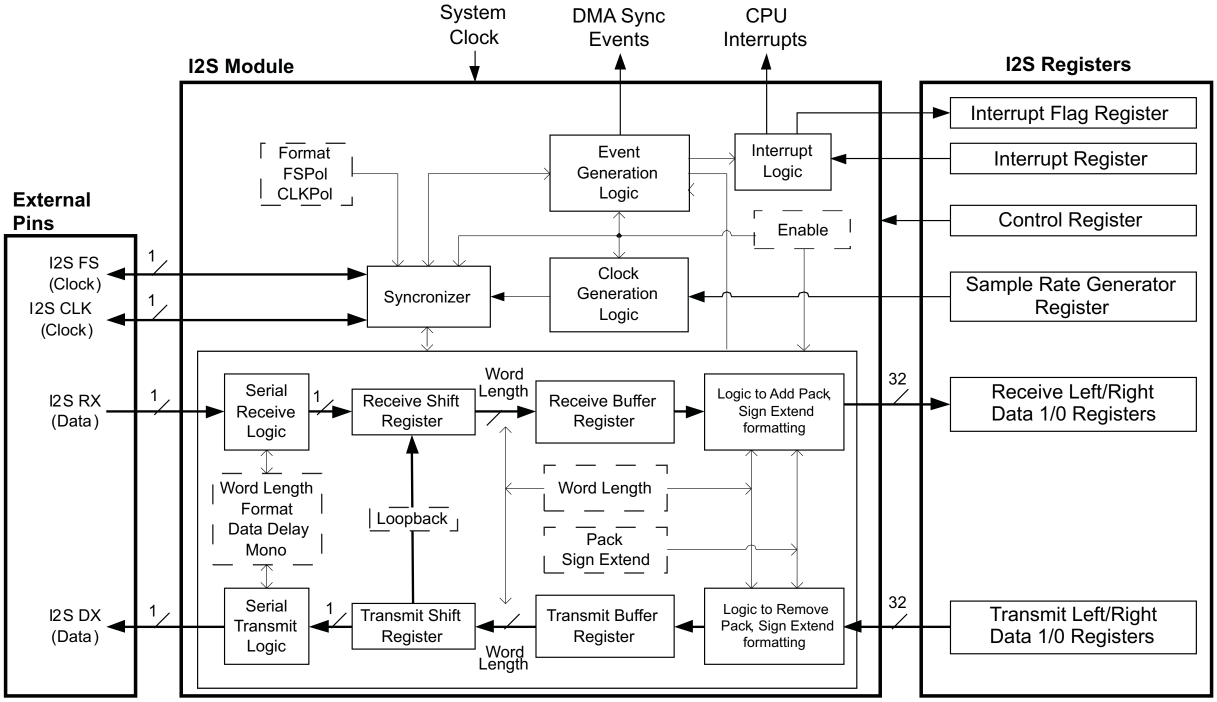
The DSP includes four independent I2S modules.

10.1.3 Functional Block Diagram

Figure 10-1 is a functional block diagram of the I2S bus illustrating the different control, data transfer, clock generation and event management blocks and their interactions. The I2S peripheral has a set of control and data registers which the CPU can access through its I/O space. The DMA can also make 32-bit accesses to receive and transmit data registers for efficient data transfer.

The bus is configured by writing to the I2S*n* Serializer Control Register (I2SSCTRL) bit fields. The bit fields in this register determine the communication protocol over the I2S bus and the arrangement of data in the data registers.

Figure 10-1. Functional Block Diagram



Data on the I2S*n*_RX pin is shifted serially into the Receive Shift Register and then copied into the Receive Buffer Register. The data is then copied to I2S*n* Receive Left/Right Data *n* Registers. For each channel (left and right), these registers can be accessed as two 16-bit registers by the CPU or as a 32-bit register by the DMA. Similarly, the I2S*n* Transmit Left/Right Data *n* Registers store the data to be transmitted out of the I2S peripheral. The CPU or DMA writes the transmit data to the I2S*n* Transmit Left/Right Data *n* Registers which is then copied to the Transmit Shift Register via the Transmit Buffer Register and shifted serially out to I2S*n*_DX pin. This structure allows internal data movement and external data communications simultaneously. Data handling and movement is discussed in further detail in later sections.

The control block consists of internal clock generation, frame synchronization signal generation, and their control. The I2S*n* Sample Rate Generator Register (I2SSRATE) contains fields to configure the frame-synchronization and bit-clock dividers to drive the I2S*n*_FS and I2S*n*_CLK clocks when the I2S peripheral is configured as a master device. When configured as a slave device, the internal clock generation logic is disabled and frame synchronization is performed on the clocks generated by the external master I2S device (see Section 10.2.2). The polarities of the bit-clock and the frame-synchronization can be set by the CLKPOL bit and FSPOL bit respectively in the I2SSCTRL register. The I2S supports a data delay of 1 bit or 2 bits as configured by the DATADLY bit in the I2SSCTRL register.

The I2S peripheral can be configured to interrupt the CPU by writing to the I2Sn Interrupt Mask Register (I2SINTMASK). When interrupts are enabled, the event-generation block posts a transmit interrupt when transmit data registers are empty and a receive interrupt when receive data registers are full. The corresponding flag is set in the I2Sn Interrupt Flag Register (I2SINTFL). In addition to data transaction interrupts, error events are also flagged in this register. Error events are not connected to interrupts on the CPU. The I2S also sends synchronization events corresponding to the transmit and receive events to the DMA controller associated with the I2S module. The I2SINTMASK register has no effect on DMA sync signal generation events. (See Section 10.2.10, Section 10.2.11, Section 10.3.7 and Section 10.3.8)

10.1.4 Industry Standard(s) Compliance

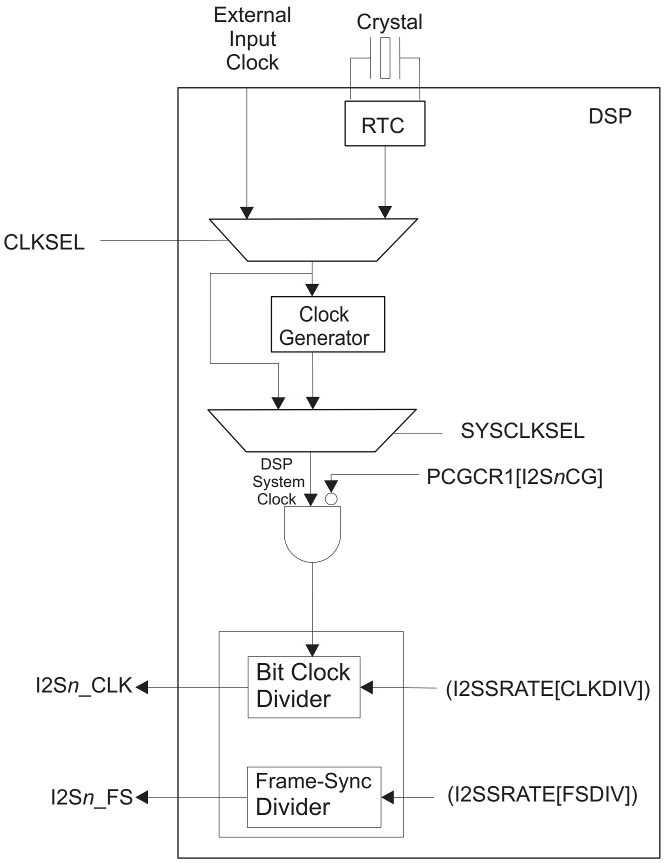
This Inter-IC Sound (I2S) Bus is compliant to industry I2S Bus Standard.

10.2 Architecture

10.2.1 Clock Control

As shown in Figure 10-2, the I2S bus is driven by the system clock. Unused I2S modules can be independently idled (clock-gated) via the peripheral clock gating configuration register 1 (PCGCR 1) for power dissipation savings. Each I2S bus should be brought out of idle before being programmed. For more details, see Section 10.2.12.

Figure 10-2. Inter-IC Sound Clock Control Diagram



If the I2S bus is configured as the master device, the DSP clock generator may need to be programmed to achieve an appropriate system clock so that the I2S clock dividers can generate the required clock rates. For more information on the DSP clock generation options, see the device-specific DSP system guide.

10.2.2 I2S Clock Generator

The I2S*n* Sample Rate Generator Register (I2SSRATE) controls the clock generation logic in the I2S bus. In slave mode (MODE = 0 in I2SSCTRL - see Section 10.3.1), the required clock signals (I2Sn_CLK and I2Sn_FS) are generated by the external master I2S device and the internal I2S clock generator is not used. Configuring the I2SSRATE register has no effect in this mode. However, when configured as a master device (MODE = 1), the I2S module generates these clocks by dividing the system clock by a value calculated from the CLKDIV and FSDIV fields programmed in the I2SSRATE register (see Section 10.3.2). The clocks can be calculated as shown below:

$$I2Sn\_CLK = SystemClock / (2^{CLKDIV+1})$$

$$I2Sn\_FS = I2Sn\_CLK / (2^{FSDIV+3})$$

I2S*n*_CLK is the bit-clock that determines the rate at which data bits are transferred on the serial I2S bus. I2S*n*_FS is referred to as the frame-sync clock or word clock and is the rate at which a data word is transferred to and/or from the I2S module. This can also be seen as the frequency at which data (audio from a microphone for example) is sampled by an analog-to-digital converter (ADC) or an audio codec.

The clock divide-by value, $2^{FSDIV+3}$, that derives the frame-sync clock from the bit-clock (as shown above), gives the number of data bits (bit-clocks) in one cycle of the frame-sync clock. Since one cycle of the frame-sync clock should accommodate two data words (one left and one right channel data word) for stereo operation and one data word (one left channel only) for mono operation, the following restrictions apply while choosing an appropriate setting for FSDIV:

$2^{FSDIV+3} \geq 2 * WDLNGTH$ (for stereo mode)

$2^{FSDIV+3} \geq WDLNGTH$ (for mono mode)

For example, to achieve a particular sampling rate of I2S*n*_FS = 48000 Hz with stereo operation of data length of 16 bits, the value of the FSDIV bit in the I2SSRATE register should be first chosen such that:

$2^{FSDIV+3} \geq 2*16 = 32$.

If we choose

FSDIV = 2 (010 binary)

the resultant I2S*n*_CLK can be calculated as:

I2S*n*_CLK = I2S*n*_FS * ($2^{FSDIV+3}$) = 48000 * 32 = 1.536 MHz.

Based on application requirements, if the DSP needs to be run at a minimum system clock or DSP clock of 45 MHz, the CLKDIV bit in the I2SSRATE register can be chosen such that,

SystemClock ≥ I2S*n*_CLK * $2^{CLKDIV+1}$ ≥ 45 MHz

Hence we should choose:

CLKDIV = 4 (100 binary)

Which will give us,

SystemClock = 1.536 MHz * 2^{4+1} = 49.15 MHz

As a result, the DSP clock generator should be configured to generate the required clock of 49.15 MHz. Due to limitations/restrictions of the DSP clock generator, it may not be possible to generate the exact system clock required, in which case I2S*n*_FS will deviate from the expected value. While it is sufficient to choose a setting for FSDIV such that 2FSDIV+3 is equal to the required number of data bits per frame-sync clock (as shown in example above), it may sometimes be necessary to choose a higher setting so that a faster bit-clock can be achieved. For a given system clock, this expedites transfer of data bits of the programmed word length on the I2S bus. As a result, the interrupts/events occur earlier in the frame-sync cycle, providing more time for the CPU/DMA to service the interrupt/event before the next interrupt/event. This is a particularly useful technique when the I2S uses CPU to handle data transfers to/from its data registers.

NOTE: I2S peripheral clock generator should only be configured if the I2S is configured as a master device. When the I2S is configured as the slave, the external master device supplies the required clocks.

10.2.3 Signal and Pin Descriptions

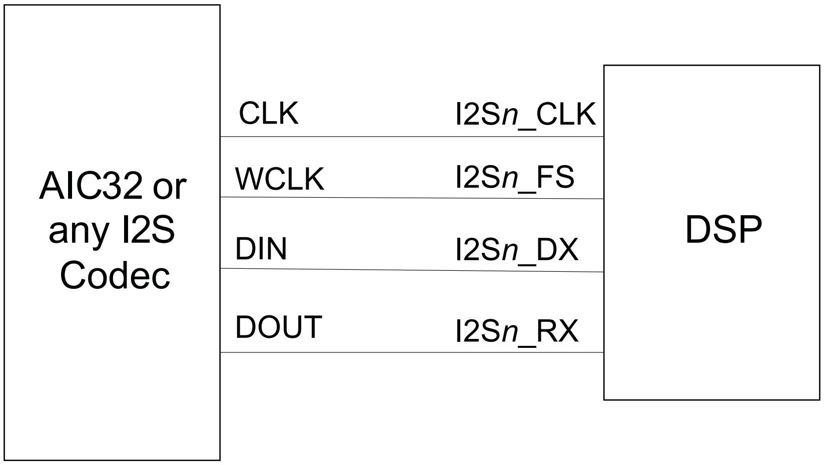
The I2S bus is a four-wire interface with two clock pins, bit-serial clock (I2S*n*_CLK) and frame-synchronization or word clock (I2S*n*_FS), and two data pins, serial data transmit (I2S*n*_DX) and serial data receive (I2S*n*_RX), for data communication as shown in Figure 10-1. The I2S*n*_CLK and I2S*n*_FS pins are bi-directional based on whether the I2S peripheral is configured as a master or slave device.

Table 10-1. I2S Signal Descriptions

Name	Signal	Description
I2S*n*_CLK	INPUT /OUTPUT	I2S Clock
I2S*n*_FS	INPUT /OUTPUT	I2S Frame Sync Clock
I2S*n*_DX	OUTPUT	I2S Data Transmit
I2S*n*_RX	INPUT	I2S Data Receive

The diagram below is a typical connection between I2S interface to an audio or voice-band Codec.

Figure 10-3. Block Diagram of I2S Interface to Audio/Voice Band Codec



10.2.3.1 Pin Multiplexing

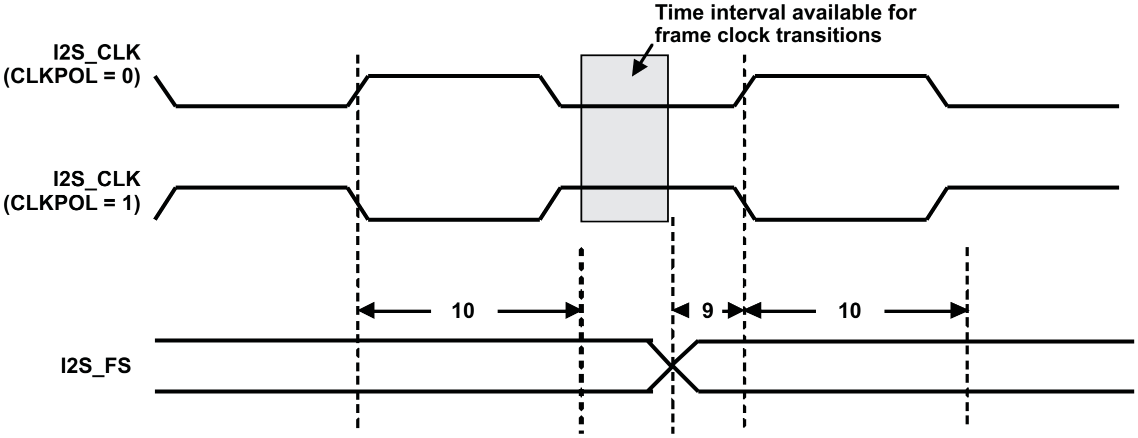
Depending on the I2S bus being used, the DSP should be configured to route those I2S signals to the multiplexed Serial Port 0, Serial Port 1, or Parallel Port pins by writing to the External Bus Selection Register (EBSR). For more information on pin multiplexing, see the device-specific DSP system guide.

NOTE: Configuring the EBSR to route I2S0 or I2S1 signals to Serial Port0 or Serial Port1 respectively also routes those I2S interrupts to the CPU (see Section 10.2.10).

10.2.4 Frame Clock Timing Requirement in Slave Mode

When configured as the slave, frame clock (I2S_FS) is required to be latched on both edges of the bit clock (I2S_CLK), which are generated by the external master device. This imposes an additional constraint on the timing of I2S_FS as illustrated in Figure 10-4. The generated frame clock should meet the specified setup and hold requirements with respect to the sampling edge of the generated bit clock. For actual timing requirements, see the I2S section of the TMS320C55xx data sheet. These constraints imply that the frame clock transitions should occur in the time window as indicated by the shaded region in the figure.

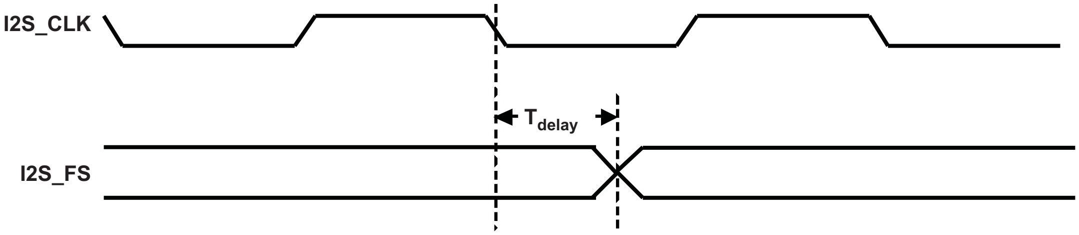
Figure 10-4. I2S Frame Clock Timing Constraint in Slave Mode



No.	Parameter	Description
9	$t_{su(FSV-CLKH)}$	Minimum setup time, I2S_FS valid before I2S_CLK high (CLKPOL = 0)
	$t_{su(FSV-CLKL)}$	Minimum setup time, I2S_FS valid before I2S_CLK low (CLKPOL = 1)
10	$t_{h(CLKH-FSV)}$	Minimum hold time, I2S_CLK high to I2S_FS (CLKPOL = 0)
	$t_{h(CLKL-FSV)}$	Minimum hold time, I2S_CLK low to I2S_FS (CLKPOL = 1)

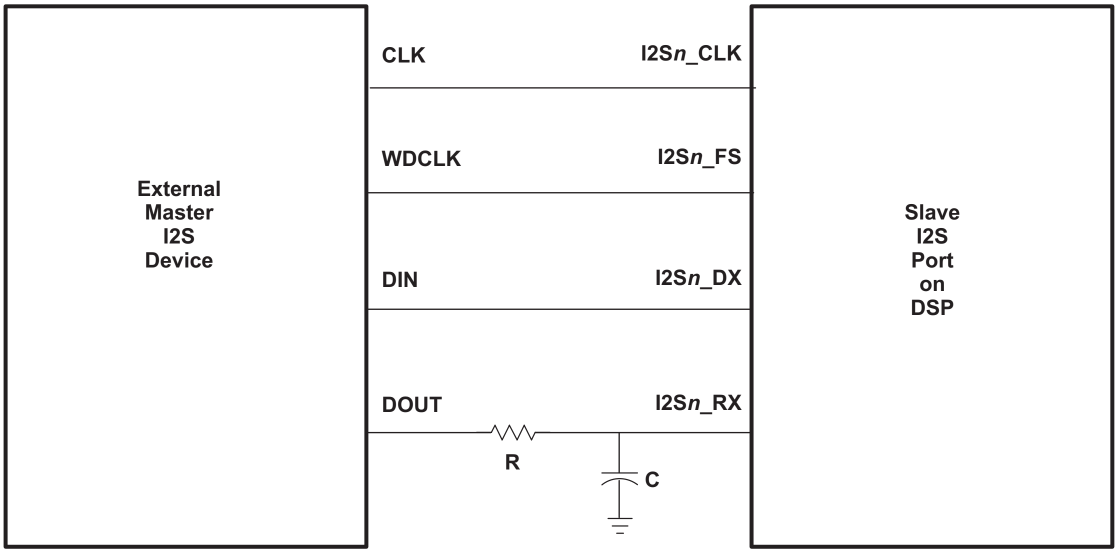
Devices (ADCs, DACS and audio/voice-band codecs) that interface to the I2S module usually only specify a maximum delay for the frame clock transition from the falling edge of the bit clock in master mode as indicated by parameter T_{delay} in Figure 10-5.

Figure 10-5. Typical Frame Clock Timing Specification



Synchronization issues may occur if the frame clock transitions close to the falling edge of the bit clock violating the previously described hold requirement resulting in incorrect data transfer. In these circumstances, the frame clock should be delayed with respect to the bit clock by introducing a time delay in its signal path as shown in Figure 10-6. The RC circuit delays the frame clock by a value given by the relation Trc = RC.

Figure 10-6. Delaying I2S Frame Clock to Overcome Synchronization Problems



NOTE: Signal should be probed as close to the TMS320C55xx device pins as possible for better results.

10.2.5 Protocol Description

The I2S bus communicates with a corresponding external I2S peripheral in a series of 1’s and 0’s. This series has a hierarchical organization that can be described in terms of bits, words, and frames.

A bit is the smallest entity in the serial stream. A "1" is represented by logic high on the data pin for the entire duration of a single bit clock. A "0" is represented by a logic low for the entire duration of a single bit clock.

A word is a group of bits that make up the data being transmitted or received. The length of the word is programmed by the user in the WDLNGTH field in the I2S*n* Serializer Control Register (I2SSCTRL).

A frame is a group of words (usually one – mono or two – stereo) that make up the data being transmitted or received. The number of bit clocks per frame and the frame rate (sampling frequency) is programmed by the user in the I2S*n* Sample Rate Generator Register (I2SSRATE).

I2S supports two serial data communication formats with external I2S devices: I2S/Left-justified format and DSP format. The I2S format is a specialized case of the more general left-justified data format. In DSP mode, the frame is marked between two consecutive pulses of the frame sync signal. On I2S, the frame is marked by a whole clock cycle of the frame sync signal with 50% duty cycle.

10.2.5.1 I2S/Left-Justified Format

In the left-justified format, the frame-synchronization or word clock has a 50% duty cycle indicating dual channel data fields with left channel data transferred during one half of the cycle and right channel data transferred during the other half. The MSB-first data is transferred serially, left justified in its own field with appropriate bit delays.

As shown in Figure 10-7, the typical I2S format utilizes left-justified format with a data delay of one bit and low frame synchronization pulse for left channel data and high pulse for right channel data. Serial data sent by the transmitter may be synchronized with either the trailing or the leading edge of serial clock I2S*n*_CLK. However, the serial data must be latched by the receiver on the leading edge of I2S*n*_CLK. In this format, the MSB of the left channel is valid on the second leading edge of the bit-clock, I2S*n*_CLK after the trailing edge of the frame-synchronization clock, I2S*n*_FS. Similarly the MSB of the right channel is valid on the second leading edge of I2S*n*_CLK after the leading edge of I2S*n*_FS.

Figure 10-7. Timing Diagram for Left-Justified Mode with Inverse Frame-Sync Polarity and One-Bit Delay

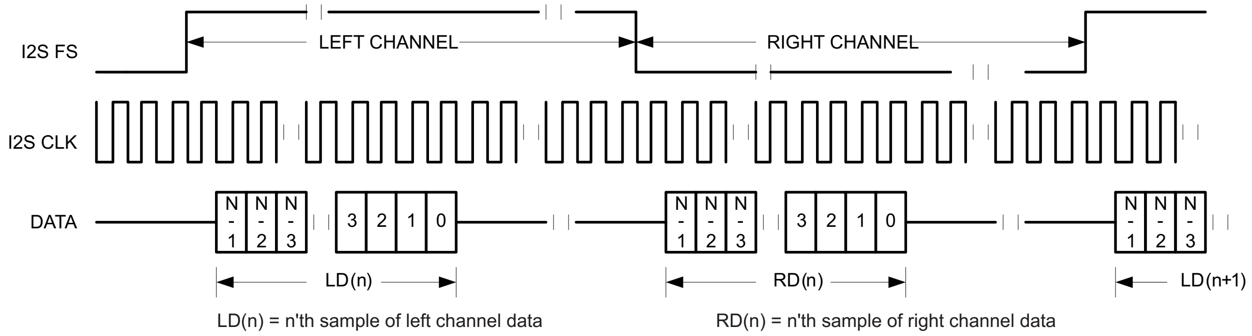

I2S FS
LEFT CHANNEL
RIGHT CHANNEL
I2S CLK
DATA
N-1 N-2 N-3 3 2 1 0
LD(n)
RD(n)
LD(n+1)
LD(n) = n'th sample of left channel data
RD(n) = n'th sample of right channel data

Figure 10-8. Timing Diagram for I2S Mode

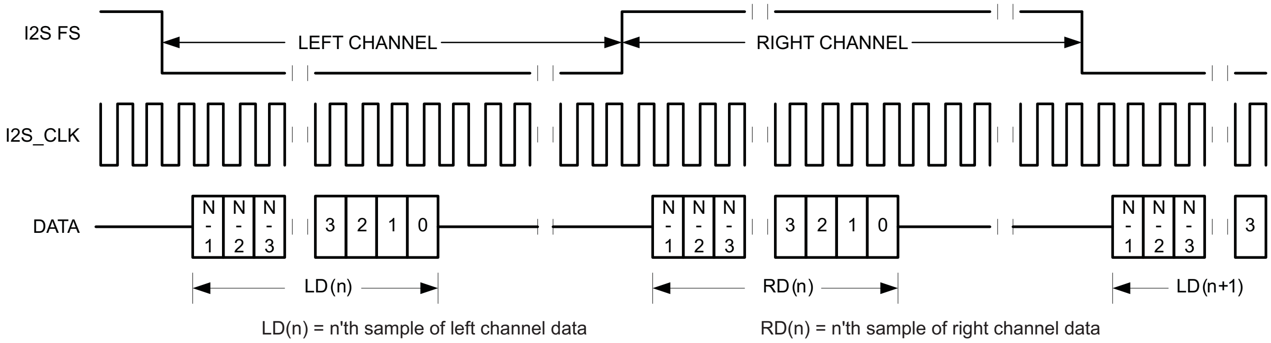
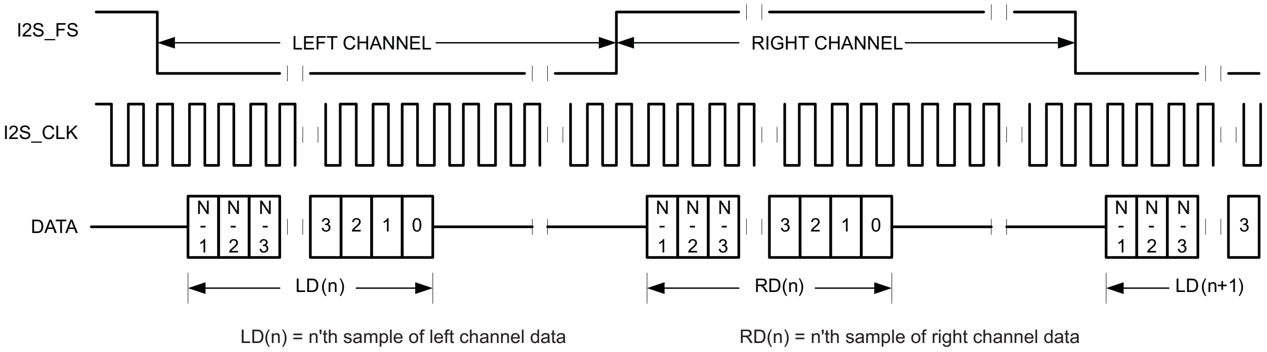

I2S FS
LEFT CHANNEL
RIGHT CHANNEL
I2S_CLK
DATA
N-1 N-2 N-3 3 2 1 0
LD(n)
RD(n)
LD(n+1)
LD(n) = n'th sample of left channel data
RD(n) = n'th sample of right channel data

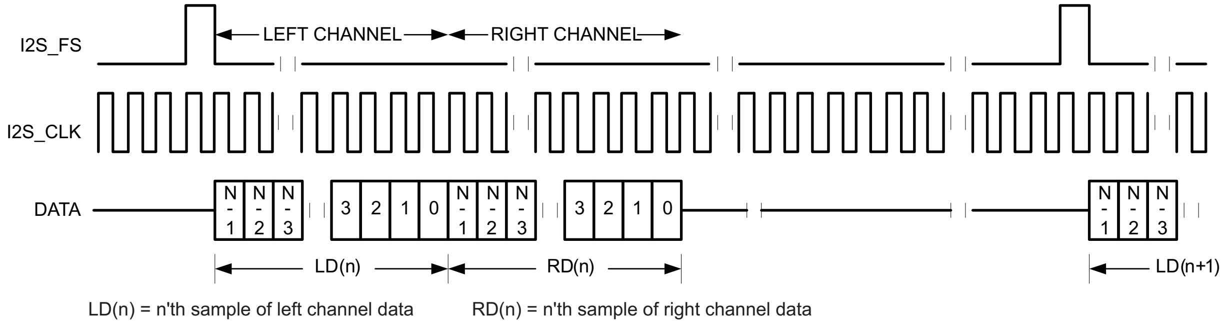
Figure 10-9. Timing Diagram for I2S Mode with Inverse Bit-Clock Polarity


I2S_FS
LEFT CHANNEL
RIGHT CHANNEL
I2S_CLK
DATA
N-1 N-2 N-3 3 2 1 0
LD(n)
RD(n)
LD(n+1)
LD(n) = n'th sample of left channel data
RD(n) = n'th sample of right channel data

10.2.5.2 DSP Format

In DSP format, the trailing edge of the frame-synchronization pulse, I2S*n*_FS, starts the data transfer with the left channel data first and immediately followed by the right channel data. Each data bit is valid on the trailing edge of the bit-clock, I2S*n*_CLK. The first data sample can be delayed by 1 bit or 2 bits after the trailing edge of I2S*n*_FS. With one bit delay, the MSB coincides with the trailing edge of I2S*n*_FS. With two bit delay, the MSB follows the trailing edge of I2S*n*_FS after one I2S*n*_CLK. Figure 10-10 illustrates DSP format operation with a one-bit data delay.

Figure 10-10. Timing Diagram for DSP Mode With One-Bit Delay


I2S_FS
LEFT CHANNEL
RIGHT CHANNEL
I2S_CLK
DATA
N-1
N-2
N-3
3
2
1
0
LD(n)
RD(n)
LD(n+1)
LD(n) = n'th sample of left channel data
RD(n) = n'th sample of right channel data

NOTE:

- The I2S*n*_DX and I2S*n*_RX pins are tri-stated during unused bit clocks in a frame.
- In I2S/Left-justified format:
  - Mono operation is not supported due to the 50% duty cycle restriction on the frame-synchronization clock.
  - The number of I2S*n*_CLKs should be greater than or equal to twice the configured data word-length.
- In DSP format:
  - The number of I2S*n*_CLKs should be greater than or equal to twice the configured data word-length for stereo operation.
  - The number of I2S*n*_CLKs should be greater than or equal to the configured data word-length for mono operation.

10.2.6 I2S Data Transfer and Control Behavior

When the I2S module is enabled, MSB-first data transfer starts when the appropriate level is detected on the frame-sync clock. Data in the Transmit Shift Register is shifted out serially to the I2Sn_DX while data bits are shifted in serially from the I2Sn_RX pin to the Receive Shift Register on the falling or leading edge of the bit-clock as programmed in the CLKPOL field of the I2SSCTRL. Data for the left channel is transferred first followed by the right channel data.

The module generates the transmit interrupt/event (the transmit and receive interrupts should be enabled in the I2SINTMASK register if CPU transfers are desired; if DMA is used to transfer data these interrupts should be disabled – see Sections 2.8 and 2.9) to indicate that the Transmit Left/Right Data1/0 registers should be filled with valid data for the next set of transfers. The CPU or DMA servicing this interrupt/event writes the next valid data to the above mentioned registers. Failure to do so before the next frame-sync cycle will result in the OUERROR being flagged in the I2SINTFL register (assuming that error detection has been enabled in the I2SINTMASK register). At the next frame-sync cycle, data from the Transmit Left/Right Data1/0 registers into the Transmit Buffer register and then to the Transmit Shift register.

NOTE: Data should be written into the Transmit Left/Right Data1/0 registers only on a transmit interrupt. Data can not be preloaded into these registers before enabling the I2S module or before receiving the first transmit interrupt.

When the required number of data words is received in the Receive Shift register (one for mono or two for stereo), the data is moved into the Receive Buffer register and ultimately into the Receive Left/Right Data1/0 registers. A receive interrupt/event is generated at this point. The CPU or DMA servicing this interrupt reads the register into memory. Failure to do so before the next frame-sync cycle will result in the OUERROR being flagged in the I2SINTFL register (assuming that error detection has been enabled in the I2SINTMASK register).

Transmit and receive interrupts/events are continuously generated after every transfer from the transmit data registers to the transmit buffer register and from the receive buffer register to the receive data registers respectively. If packed mode is enabled (PACK = 1 in I2SSCTRL), interrupts/events are generated after all the required number of data words have been transmitted/received (see Section 10.2.8).

10.2.7 I2S Data Transfer Latency

Due to the buffered nature of the I2S module, there exists some latency in the transmit and receive paths (from the Transmit Data registers to the I2S_DX pin and I2S_RX pin to Receive Data registers) which is dependent on the desired configuration of the module. The latency may not be of consequence when the I2S module in its intended scope of a streaming audio peripheral. However, it is documented here in the interests of completeness. It will also help explain observed loopback data (LOOPBACK=1 in I2SSCTRL) as the latency is also present in loopback mode.

10.2.7.1 Transmit Path Latency

After the I2S is enabled, the first valid data sample on the I2Sn_DX pin will appear after:

- Five frame-sync clocks if PACK mode is used (PACK=1 in I2SSCTRL) or,
- Three frame-sync clocks if PACK mode is not used (PACK=0 in I2SSCTRL)

Hence there is a latency of three or five samples (for each channel) before the first data sample that is written to the Transmit Left/Right Data 1/0 registers after the first transmit interrupt/event. During this time, the I2S transmits zeros that should be discarded or ignored.

10.2.7.2 Receive Path Latency

After the I2S is enabled, the receive path starts receiving data after:

- 1 or 2 frame-sync clocks for 8-, 18-, 20-, 24-, or 32-bit data depending on other configuration
- 1, 2 or 3 frame-sync clocks for 10-, 12-, 14-, or 16-bit data depending on other configuration

10.2.7.3 Loopback Path Latency

The internal loopback mode (LOOPBACK=1 in I2SSCTRL) can be used as a debug tool to verify the user's program to service I2S interrupts/events. This is different from an external loopback which would require the I2Sn_DX pin to be connected to the I2Sn_RX pin. In the internal loopback mode, data from the Transmit Shift register is directly routed to the Receive Shift register, changing the data latency as given below:

- If pack mode is used (PACK=1 in I2SSCTRL)
  - Ignore the first 6 samples received for 8-bit data and FSDIV=000 in I2SSRATE
  - Ignore the first 5 samples received for 8-bit data and FSDIV > 000 in I2SSRATE
  - Ignore the first 6 samples received for 10-, 12-, 14- or 16-bit data
- If pack mode is not used (PACK=0 in I2SSCTRL), ignore the first 2 samples received.

10.2.8 Data Packing and Sign Extension Options

The I2S bus supports the use of packed (multiple) and/or sign extended data words in its data registers to reduce software overheads in servicing interrupts for every data sample and for ease of data handling in software algorithms.

NOTE: Using the Pack or Sign Extend options does not affect transmission of data samples over the serial I2S bus; it only affects how data words are arranged in the Receive and Transmit Data Registers.

10.2.8.1 Data Pack Mode

Setting the PACK bit field in the I2SSCTRL register enables data packing in the 32-bit I2S data registers for word-lengths of 8, 10, 12, 14 and 16 bits. This mode can be used in the following scenario:

- Using DMA to transfer data samples to make better use of data buffers.
- Using CPU to transfer data samples to reduce interrupt overheads.

During the receive operation, the I2S bus puts successive data samples into the I2S*n* Receive Left/Right Data *n* Registers (Data 1 register first, Data 0 register next) before generating the interrupt/event. The transmit data is expected in a similar format in the I2S*n* Transmit Left/Right Data *n* Registers. Four 8-bit data samples or two 10, 12, 14 or 16-bit data samples can be packed in the data registers, as shown in Section 10.2.8.3.

The advantages of using the PACK mode can be seen as given below:

- Reduces the number of I2S interrupts/events, which results in reducing interrupt overheads and the better use of bus bandwidth.
- Efficient use of internal/on-chip memory if DMA is used for transferring data between I2S and main memory. Since the DMA transfers a double-word (all 32-bits of the I2S*n* Transfer Left/Right Data *n* Registers) during each transaction, using packed I2S*n* Receive/Transmit Left/Right Data *n* Registers results in efficient transfer of data samples. Hence, for a given number of samples, size of data buffers is reduced by a factor of two for 10-, 12-, 14- or 16-bit word length or by a factor of four for 8-bit word length.

Figure 10-11 and Figure 10-12 illustrate packed and unpacked data receive behavior for mono transmission of four 12-bit data samples (sign extension is not enabled). When pack mode is not used, the I2S module stores 12-bit data left-justified in MSW (with trailing zeros) and generates DMA event resulting in zeros stored in alternate memory locations. When pack mode is used, the module packs 12-bit data left-justified in MSW first and then LSW (with trailing zeros) and generates DMA event once every two transfers resulting in better utilization of memory.

Figure 10-11. Example of Unpacked 12-Bit Data Receive

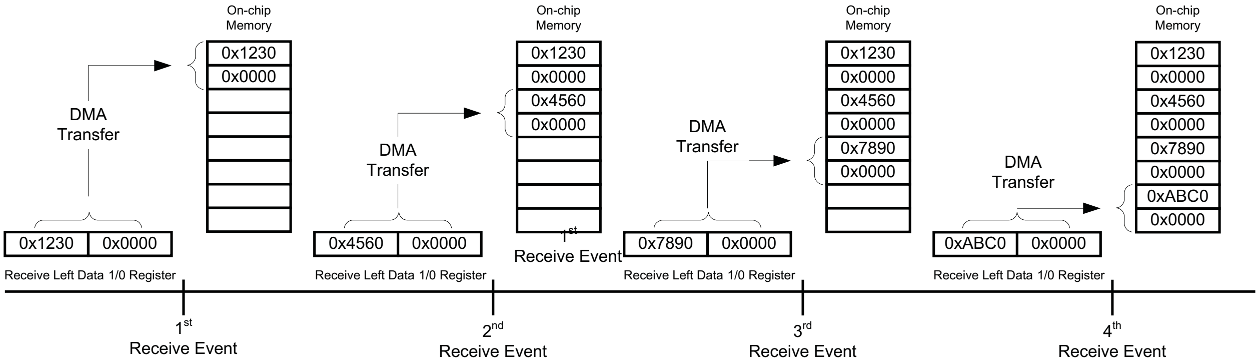
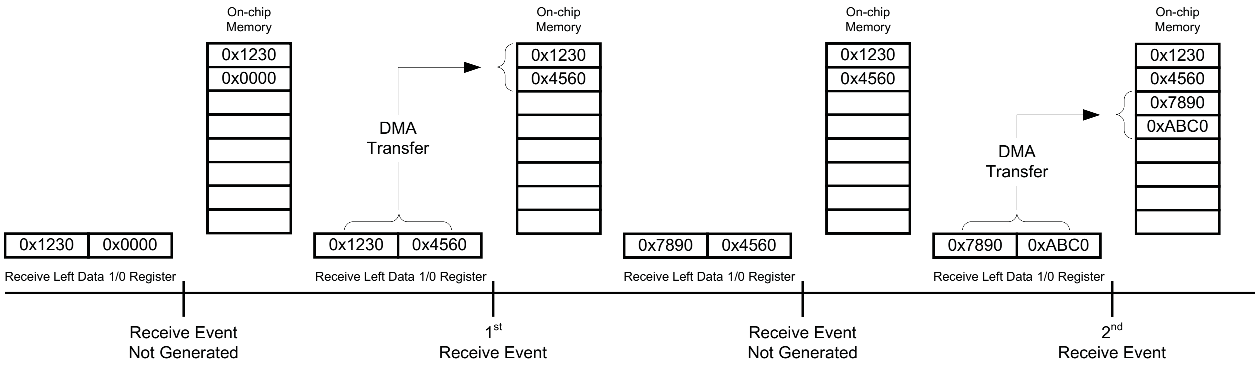


Figure 10-12. Example of Packed 12-Bit Data Receive



10.2.8.2 Data Sign Extend Mode

The I2S peripherals can be configured to work with sign extended data samples for ease of use in commonly used S16 or S32 data representations in signal processing algorithms. Data words of lengths 8-, 10-, 12- or 14-bits are sign-extended (right-justified) to 16 bits (8-bit data cannot be sign extended in packed mode as four samples are packed into the 32-bit data registers). Data of word lengths 18-, 20- or 24-bits are sign extended (right-justified) to 32 bits.

The choice of PACK, SIGN_EXT and WDLNGTH bit options in the I2SSCTRL register affects the arrangement of received data in the I2S*n* Receive Left/Right Data *n* Registers. Similarly, the I2S peripheral expects data to be arranged in a similar fashion in the I2S*n* Transmit Left/Right Data *n* Registers for the correct data value to be shifted out to the I2S*n*_DX pin.

Table 10-2 illustrates sign extension behavior with an example.

Table 10-2. Example of Sign Extension Behavior

		Data Register With SIGN_EXT=0		Data Register With SIGN_EXT=1	
Word Length	Data on I2S Bus	Data 1 Register	Data 0 Register	Data 1 Register	Data 0 Register
8	0xA1	0xA100	0x0000	0xFFA1	0x0000
10	0x2B7	0xAD80	0x0000	0xFEB7	0x0000
12	0x6AA	0x6AA0	0x0000	0x06AA	0x0000
14	0x3999	0xE666	0x0000	0xF999	0x0000
16	0x29CC	0x29CC	0x0000	0x29CC	0x0000
18	0x19EFA	0x67BE	0x8000	0x0001	0x9EFA
20	0x7ADE1	0x7ADE	0x1000	0x0007	0xADE1
24	0x98A311	0x98A3	0x1100	0xFF98	0xA311
32	0xD16AEE09	0xD16A	0xEE09	0xD16A	0xEE09

10.2.8.3 PACK and Sign Extend Data Arrangement for Various Word Lengths

The tables in this section describe sign extended and packed data arrangement for each configurable word length. Data samples are indicated by x[0], x[1], x[2] and x[3] in increasing temporal order (x[3] is the most recent sample).

10.2.8.3.1 8-Bit Word Length

Table 10-3. PACK and Sign Extend Data Arrangement for 8-Bit Word Length

	I2S Receive/Transmit Left or Right Data 1 Register			I2S Receive/Transmit Left or Right Data 0 Register	
PACK = 0 SIGN_EXT = 0	15 8	7 0		15	0
	x(0)	0		0	
PACK = 0 SIGN_EXT = 1	15 8	7 0		15	0
	Sign Bits (Bit #7)	x(0)		0	
PACK = 1 SIGN_EXT = 0	15 8	7 0		15 8	7 0
	x(0)	x(1)		x(2)	x(3)
PACK = 1 SIGN_EXT = 1	15 8	7 0		15 8	7
	x(0)	x(1)		x(2)	x(3)

10.2.8.3.2 10-Bit Word Length

Table 10-4. PACK and Sign Extend Data Arrangement for 10-Bit Word Length

	I2S Receive/Transmit Left or Right Data 1 Register		I2S Receive/Transmit Left or Right Data 0 Register	

PACK = 0
SIGN_EXT = 0
	15 6	5 0	15 0	

	x(0)	0	0	

PACK = 0
SIGN_EXT = 1
	15 10	9 0	15 0	

	Sign Bits (Bit #9)	x(0)	0	

PACK = 1
SIGN_EXT = 0
	15 6	5 0	15 6	5 0

	x(0)	0	x(1)	0

PACK = 1
SIGN_EXT = 1
	15 10	9 0	15 10	9

	Sign Bits (Bit #9)	x(0)	Sign Bits (Bit #9)	x(1)

10.2.8.3.3 12-Bit Word Length

Table 10-5. PACK and Sign Extend Data Arrangement for 12-Bit Word Length

	I2S Receive/Transmit Left or Right Data 1 Register		I2S Receive/Transmit Left or Right Data 0 Register	

PACK = 0
SIGN_EXT = 0
	15 4	3 0	15 0	

	x(0)	0	0	

PACK = 0
SIGN_EXT = 1
	15 12	11 0	15 0	

	Sign Bits (Bit #11)	x(0)	0	

PACK = 1
SIGN_EXT = 0
	15 4	3 0	15 4	3 0

	x(0)	0	x(1)	0

PACK = 1
SIGN_EXT = 1
	15 12	11 0	15 12	11

	Sign Bits (Bit #11)	x(0)	Sign Bits (Bit #11)	x(1)

10.2.8.3.4 14-Bit Word Length

Table 10-6. PACK and Sign Extend Data Arrangement for 14-Bit Word Length

	I2S Receive/Transmit Left or Right Data 1 Register		I2S Receive/Transmit Left or Right Data 0 Register	

PACK = 0
SIGN_EXT = 0
	15 2	1 0	15 0	

	x(0)	0	0	

PACK = 0
SIGN_EXT = 1
	15 14	13 0	15 0	

	Sign Bits (Bit #13)	x(0)	0	

PACK = 1
SIGN_EXT = 0
	15 2	1 0	15 2	1 0

	x(0)	0	x(1)	0

PACK = 1
SIGN_EXT = 1
	15 14	13 0	15 14	13

	Sign Bits (Bit #13)	x(0)	Sign Bits (Bit #13)	x(1)

10.2.8.3.5 16-Bit Word Length

Table 10-7. PACK and Sign Extend Data Arrangement for 16-Bit Word Length

	I2S Receive/Transmit Left or Right Data 1 Register			I2S Receive/Transmit Left or Right Data 0 Register	
PACK = 0 SIGN_EXT = 0	15	0		15	0
	x(0)			0	
PACK = 0 SIGN_EXT = 1	15	0		15	0
	x(0)			0	
PACK = 1 SIGN_EXT = 0	15	0		15	0
	x(0)			x(1)	
PACK = 1 SIGN_EXT = 1	15	0		15	
	x(0)			x(1)	

10.2.8.3.6 18-Bit Word Length

Table 10-8. PACK and Sign Extend Data Arrangement for 18-Bit Word Length

	I2S Receive/Transmit Left or Right Data 1 Register			I2S Receive/Transmit Left or Right Data 0 Register	
PACK = 0 SIGN_EXT = 0	15	0		15 14	13 0
	x(0) Bits[17-2]			x(0) Bits[1-0]	0
PACK = 0 SIGN_EXT = 1	15 2	1 0		15	0
	Sign Bits(Bit #17)	x(0) Bits[17-16]		x(0) Bits[15-0]	
PACK = 1 SIGN_EXT = 0	Not Supported			Not Supported	
PACK = 1 SIGN_EXT = 1	Not Supported			Not Supported	

10.2.8.3.7 20-Bit Word Length

Table 10-9. PACK and Sign Extend Data Arrangement for 20-Bit Word Length

	I2S Receive/Transmit Left or Right Data 1 Register			I2S Receive/Transmit Left or Right Data 0 Register	
PACK = 0 SIGN_EXT = 0	15	0		15 12	11 0
	x(0) Bits[19-4]			x(0) Bits[3-0]	0
PACK = 0 SIGN_EXT = 1	15 4	3 0		15	0
	Sign Bits(Bit #19)	x(0) Bits[19-16]		x(0) Bits[15-0]	
PACK = 1 SIGN_EXT = 0	Not Supported			Not Supported	
PACK = 1 SIGN_EXT = 1	Not Supported			Not Supported	

10.2.8.3.8 24-Bit Word Length

Table 10-10. PACK and Sign Extend Data Arrangement for 24-Bit Word Length

	I2S Receive/Transmit Left or Right Data 1 Register		I2S Receive/Transmit Left or Right Data 0 Register	

PACK = 0
SIGN_EXT = 0
	15	0	15 8	7 0

	x(0) Bits[23-8]		x(0) Bits[7-0]	0

PACK = 0
SIGN_EXT = 1
	15 8	7 0	15	0

	Sign Bits(Bit #23)	x(0) Bits[23-16]	x(0) Bits[15-0]	

PACK = 1
SIGN_EXT = 0
	Not Supported		Not Supported	

PACK = 1
SIGN_EXT = 1
	Not Supported		Not Supported	

10.2.8.3.9 32-Bit Word Length

Table 10-11. PACK and Sign Extend Data Arrangement for 32-Bit Word Length

	I2S Receive/Transmit Left or Right Data 1 Register		I2S Receive/Transmit Left or Right Data 0 Register	

PACK = 0
SIGN_EXT = 0
	15	0	15	0

	x(0) Bits[32-16]		x(0) Bits[15-0]	

PACK = 0
SIGN_EXT = 1
	15	0	15	0

	x(0) Bits[32-16]		x(0) Bits[15-0]	

PACK = 1
SIGN_EXT = 0
	Not Supported		Not Supported	

PACK = 1
SIGN_EXT = 1
	Not Supported		Not Supported	

10.2.9 Reset Considerations

The I2S bus has two reset sources: software reset and hardware reset.

10.2.9.1 Software Reset Considerations

The I2S bus can be reset by software through the Peripheral Reset Control Register (PRCR). The software reset is very similar to hardware reset with the only exception being that the data in shift registers are not reset. For more details on PRCR, see the device-specific DSP system guide.

10.2.9.2 Hardware Reset Considerations

A hardware reset is always initiated during a full chip reset. When a hardware reset occurs, all the registers of the I2S bus are set to their default values.

10.2.10 Interrupt Support

Every I2S bus supports transmit and receive data-transfer interrupts/events to CPU or DMA and flags two error conditions. These are enabled or disabled by writing to the I2SINTMASK register. There are separate data-transfer interrupt mask bits for stereo or mono operating modes. The interrupts are also flagged in the I2SINTFL.

For Stereo operating mode (MONO=0 in I2SSCTRL) and with the corresponding STEREO interrupts enabled, transmit/receive interrupts are generated after the right channel data has been transferred (since right channel data is always transmitted/received last). In Mono mode (MONO=1 in I2SSCTRL) , only left channel data is transferred, hence the interrupts are generated after the transmit/receive of left channel data. Mono mode can only be used with DSP format (FRMT = 1 in I2SSCRTL).

NOTE: I2S bus behavior is not defined if stereo and mono interrupts are simultaneously enabled. The I2S module flags an OUERROR (if enabled in I2SINTMASK) incorrectly when the module is enabled for the first time. The user program should ignore this error.

10.2.10.1 Interrupt Multiplexing

Interrupts to the CPU of two I2S peripherals, I2S0 and I2S1, are multiplexed with MMC/SD0 and MMC/SD1 interrupts. Configuring Serial Port 0 or Serial Port 1 fields in the External Bus Selection register to route I2S signals to the external pins, also routes the corresponding I2S interrupts to the CPU. For details, see the device-specific DSP system guide.

10.2.11 DMA Event Support

Each I2S bus has access to one of the four DMA peripherals on the DSP as shown in Table 10-12. To enable seamless transfers using the DMA, the I2S bus generates data-transfer events to DMA irrespective of the value configured in the I2SINTMASK register. Hence, it is recommended to disable CPU data-transfer interrupts in the I2SINTMASK register when using the DMA for data transfers. For Stereo operating mode (MONO=0 in I2SSCTRL), transmit/receive events are generated after the right channel data has been transferred (since right channel data is always transmitted/received last). In Mono mode (MONO=1 in I2SSCTRL) , only left channel data is transferred, hence the events are generated after the transmit/receive of left channel data. Mono mode can only be used with DSP format (FRMT = 1 in I2SSCRTL). For details on DMA configurations options, see the device-specific DSP system guide.

Table 10-12. DMA Access to I2S

DMA Engine	I2S Bus
DMA0	I2S0
DMA1	I2S2
DMA2	I2S3
DMA3	I2S1

NOTE: The DMA can be used to transfer data samples in single double-word bursts from the I2S to:

- On-chip memory (DARAM/SARAM)
- NAND/NOR memory via EMIF

10.2.12 Power Management

The I2S peripherals can be put into idle condition to save power consumption. This is achieved by gating the system clock to the I2S peripherals via individual fields in the Peripheral Clock Gating Configuration registers described in the device-specific DSP system guide.

10.2.13 Emulation Considerations

An emulation halt will not stop I2S operation and an over-run/under-run error condition will be flagged (if enabled) in the I2SINTFL register if the CPU or DMA is unable to service I2S interrupts/events as a result of the emulation halt.

Refreshing CPU registers or memory contents in Code Composer Studio when the I2S is running could result in the DSP missing real-time operation due to JTAG communication overheads over the internal data buses. This would result in over-run/under-run errors being flagged (if enabled) in the I2SINTFL register.

10.2.14 Steps for I2S Configuration and I2S Interrupt Service Routine (ISR)

A sequence of steps for configuring an I2S bus and servicing interrupts in an interrupt service routine (ISR) are given below:

10.2.14.1 Initialization and Configuration Steps

- Bring I2S out of idle.
- If using DMA, reset DMA and MPORT idle bit-fields and run idle instruction to force MPORT out of idle.
- Disable DSP global interrupts.
- Clear DSP interrupt flag registers and enable appropriate I2S/DMA interrupts.
- If using DMA, configure DMA and sync DMA channel(s) to I2S sync event(s).
- Configure external I2S-compatible device.
- Enable DSP global interrupts.
- I2S Configuration:
  - Route I2S signals to external pins.
  - If I2S is the master device, configure the I2SSRATE register.
  - If using CPU interrupts, configure I2SINTMASK register to enable stereo receive/transmit interrupts for stereo mode operation or mono receive/transmit interrupts for mono mode operation (Peripheral behavior is not defined if both stereo and mono interrupts are enabled).
    - If the software/application is designed to respond to the detection of the error condition, enable error interrupts (disregard first OUERROR that is generated).
    - Do not enable stereo/mono TX or RX interrupts if DMA is used for data transfers.
  - Write desired configuration value to I2S*n* Serializer Control Register (I2SSCTRL) (If writing to individual bit fields, enable the I2S by setting the ENABLE bit in I2SSCTRL last.
    - The I2S bus now starts data conversion.
    - If configured as master device, I2S bus will generate interrupts/events even if external I2S-compatible device is not configured correctly and hence not executing data transfers.
    - If configured as a slave device, interrupt/event generation depends on proper operation of the external master device.
- If CPU is not performing other operations, the CPU can now be idled for power savings (if DMA is used for data transfers and software/application requires detection of error conditions, CPU may to read the I2S*n* Interrupt Flag Register (I2SINTFL) at regular intervals to check the error flag). For more information on the CPU idle mode, see the device-specific DSP system guide.
- An I2S (or DMA) interrupt will indicate completion of data transfer(s) and CPU is automatically brought of idle and the interrupt is taken.

10.2.14.2 ISR Steps (for CPU transfers)

Transmit and receive interrupts should have distinct interrupt service routines. A common framework is given below:

- Read the I2SINTFL register to reset the flags and if error detection is required, check for error flags.
- Read or write the I2S*n* Receive/Transmit Left/Right Data *n* Registers for receive and transmit interrupts respectively based on the I2S configuration (Mono, PACK, Sign Extend, Word Length options).
- Return from interrupt.

10.3 Registers

Table 10-13 through Table 10-16 lists the registers of the Inter-IC Sound (I2S) Bus. Refer to the sections listed for detailed information on each register.

Table 10-13. I2S0 Register Mapping Summary

CPU Word Address	Acronym	Description	
2800h	I2SSCTRL	I2S Serializer Control Register	Section 10.3.1
2804h	I2SSRATE	I2S Sample Rate Generator Register	Section 10.3.2
2808h	I2STXLT0	I2S Transmit Left Data 0 Register	Section 10.3.3
2809h	I2STXLT1	I2S Transmit Left Data 1 Register	Section 10.3.4
280Ch	I2STXRT0	I2S Transmit Right Data 0 Register	Section 10.3.5
280Dh	I2STXRT1	I2S Transmit Right Data 1 Register	Section 10.3.6
2810h	I2SINTFL	I2S Interrupt Flag Register	Section 10.3.7
2814h	I2SINTMASK	I2S Interrupt Mask Register	Section 10.3.8
2828h	I2SRXLT0	I2S Receive Left Data 0 Register	Section 10.3.9
2829h	I2SRXLT1	I2S Receive Left Data 1 Register	Section 10.3.10
282Ch	I2SRXRT0	I2S Receive Right Data 0 Register	Section 10.3.11
282Dh	I2SRXRT1	I2S Receive Right Data 1 Register	Section 10.3.12

Table 10-14. I2S1 Register Mapping Summary

CPU Word Address	Acronym	Description	
2900h	I2SSCTRL	I2S Serializer Control Register	Section 10.3.1
2904h	I2SSRATE	I2S Sample Rate Generator Register	Section 10.3.2
2908h	I2STXLT0	I2S Transmit Left Data 0 Register	Section 10.3.3
2909h	I2STXLT1	I2S Transmit Left Data 1 Register	Section 10.3.4
290Ch	I2STXRT0	I2S Transmit Right Data 0 Register	Section 10.3.5
290Dh	I2STXRT1	I2S Transmit Right Data 1 Register	Section 10.3.6
2910h	I2SINTFL	I2S Interrupt Flag Register	Section 10.3.7
2914h	I2SINTMASK	I2S Interrupt Mask Register	Section 10.3.8
2928h	I2SRXLT0	I2S Receive Left Data 0 Register	Section 10.3.9
2929h	I2SRXLT1	I2S Receive Left Data 1 Register	Section 10.3.10
292Ch	I2SRXRT0	I2S Receive Right Data 0 Register	Section 10.3.11
292Dh	I2SRXRT1	I2S Receive Right Data 1 Register	Section 10.3.12

Table 10-15. I2S2 Register Mapping Summary

CPU Word Address	Acronym	Description	
2A00h	I2SSCTRL	I2S Serializer Control Register	Section 10.3.1
2A04h	I2SSRATE	I2S Sample Rate Generator Register	Section 10.3.2
2A08h	I2STXLT0	I2S Transmit Left Data 0 Register	Section 10.3.3
2A09h	I2STXLT1	I2S Transmit Left Data 1 Register	Section 10.3.4
2A0Ch	I2STXRT0	I2S Transmit Right Data 0 Register	Section 10.3.5
2A0Dh	I2STXRT1	I2S Transmit Right Data 1 Register	Section 10.3.6
2A10h	I2SINTFL	I2S Interrupt Flag Register	Section 10.3.7
2A14h	I2SINTMASK	I2S Interrupt Mask Register	Section 10.3.8
2A28h	I2SRXLT0	I2S Receive Left Data 0 Register	Section 10.3.9
2A29h	I2SRXLT1	I2S Receive Left Data 1 Register	Section 10.3.10
2A2Ch	I2SRXRT0	I2S Receive Right Data 0 Register	Section 10.3.11
2A2Dh	I2SRXRT1	I2S Receive Right Data 1 Register	Section 10.3.12

Table 10-16. I2S3 Register Mapping Summary

CPU Word Address	Acronym	Description	
2B00h	I2SSCTRL	I2S Serializer Control Register	Section 10.3.1
2B04h	I2SSRATE	I2S Sample Rate Generator Register	Section 10.3.2
2B08h	I2STXLT0	I2S Transmit Left Data 0 Register	Section 10.3.3
2B09h	I2STXLT1	I2S Transmit Left Data 1 Register	Section 10.3.4
2B0Ch	I2STXRT0	I2S Transmit Right Data 0 Register	Section 10.3.5
2B0Dh	I2STXRT1	I2S Transmit Right Data 1 Register	Section 10.3.6
2B10h	I2SINTFL	I2S Interrupt Flag Register	Section 10.3.7
2B14h	I2SINTMASK	I2S Interrupt Mask Register	Section 10.3.8
2B28h	I2SRXLT0	I2S Receive Left Data 0 Register	Section 10.3.9
2B29h	I2SRXLT1	I2S Receive Left Data 1 Register	Section 10.3.10
2B2Ch	I2SRXRT0	I2S Receive Right Data 0 Register	Section 10.3.11
2B2Dh	I2SRXRT1	I2S Receive Right Data 1 Register	Section 10.3.12

10.3.1 I2S*n* Serializer Control Register (I2SSCTRL)

The I2S*n* serializer control register (I2SSCTRL) is shown in Figure 10-13 and described in Table 10-17.

Figure 10-13. I2S*n* Serializer Control Register (I2SSCTRL)

15	14	13	12	11	10	9	8
ENABLE	Reserved		MONO	LOOPBACK	FSPOL	CLKPOL	DATADLY
R/W-0	R-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5			2	1	0
PACK	SIGN_EXT	WDLNGTH				MODE	FRMT
R/W-0	R/W-0	R/W-0				R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 10-17. I2S*n* Serializer Control Register (I2SSCTRL) Field Descriptions

Bit	Field	Value	Description
15	ENABLE		Resets or enables the serializer transmission or reception.
		0	The I2S Bus (Data XFR, clock generation, and event gen logic) is disabled and held in reset state.
		1	I2S enabled.
14-13	Reserved	0	Reserved.
12	MONO		Sets I2S into mono or Stereo mode.
		0	Stereo mode.
		1	Mono mode. Valid only when bit 0, FRMT=1 (DSP Format).
11	LOOPBACK		Routes data from transmit shift register back to receive shift register for internal digital loopback.
		0	Normal operation, no loopback.
		1	Digital Loopback mode enabled.
10	FSPOL		Inverts I2S frame-synchronization polarity.
		0	The following: FRMT (bit 0): Function 0: (I2S/LJ) Frame-synchronization pulse is low for left word and high for right word 1: (DSP) Frame-synchronization is pulsed high
		1	The following: FRMT (bit 0): Function 0: (I2S/LJ) Frame-synchronization pulse is high for left word and low for right word 1: (DSP)Frame-synchronization is pulsed low
9	CLKPOL		Controls I2S clock polarity.
		0	The following: FRMT (bit 0): Function 0: (I2S/LJ) Receive data is sampled on the rising edge and transmit data shifted on the falling edge of the bit clock. 1: (DSP) Receive data is sampled on the falling edge and transmit data shifted on the rising edge of the bit clock.
		1	The following: FRMT (bit 0): Function 0: (I2S/LJ) Receive data is sampled on the falling edge and transmit data shifted on the rising edge of the bit clock. 1: (DSP) Receive data is sampled on the rising edge and transmit data falling on the rising edge of the bit clock.
8	DATADLY		Sets the I2S receive/transmit data delay.
		0	1-bit data delay.
		1	2-bit data delay.

Table 10-17. I2S*n* Serializer Control Register (I2SSCTRL) Field Descriptions (continued)

Bit	Field	Value	Description
7	PACK		Enable data packing. Divides down the generation of interrupts so that data is packed into the 32-bit receive/transmit word registers for each channel (left/right).
		0	Data packing mode disabled.
		1	Data packing mode enabled.
			For more information about data packing, see Section 10.2.8.
6	SIGN_EXT		Enable sign extension of words.
		0	No sign extension.
		1	Received data is sign extended. Transmit data is expected to be sign extended.
			For more information about sign extension, see Section 10.2.8.
5-2	WDLNGTH		Choose serializer word length.
		0	8-bit data word.
		1h	10-bit data word.
		2h	12-bit data word.
		3h	14-bit data word.
		4h	16-bit data word.
		5h	18-bit data word.
		6h	20-bit data word.
		7h	24-bit data word.
		8h	32-bit data word.
		9h-Fh	Reserved.
1	MODE		Sets the serializer in master or slave mode.
		0	Serializer is configured as a slave. I2S*n*_CLK and I2S*n*_FS pins are configured as inputs. The bit-clock and frame-synchronization signals are derived from an external source and are provided directly to the I2S synchronizer without being further divided.
		1	Serializer is configured as a master. I2S*n*_CLK and I2S*n*_FS pins are configured as outputs and driven by the clock generators. The bit-clock and frame-synchronization signals are derived from the internal CPU clock.
0	FRMT		Sets the serializer data format.
		0	I2S/left-justified format.
		1	DSP format.

10.3.2 I2S*n* Sample Rate Generator Register (I2SSRATE)

The I2S*n* sample rate generator register (I2SSRATE) shown below controls the operation and various features of the sample rate generator. In master mode, the serializer generates the required clock signals by dividing the input clock by CLKDIV and additionally by FSDIV bit values programmed in the I2SSRATE register. In slave mode, the clocks are externally derived and fed directly to the serializer without division. Hence, this register is ignored in slave mode.

The I2S*n* sample rate generator register (I2SSRATE) is shown in Figure 10-14 and described in Table 10-18.

Figure 10-14. I2S*n* Sample Rate Generator Register (I2SSRATE)

15 6	5 3	2 0
Reserved	FSDIV	CLKDIV
R-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 10-18. I2S*n* Sample Rate Generator Register (I2SSRATE) Field Descriptions

Bit	Field	Value	Description
15-6	Reserved	0	Reserved.
5-3	FSDIV		Divider to generate I2S*n*_FS (frame-synchronization clock). The I2S*n*_CLK is divided down by the configured value to generate the frame-synchronization clock. (Has no effect when I2S is configured as slave device).
		0	Divide by 8
		1h	Divide by 16
		2h	Divide by 32
		3h	Divide by 64
		4h	Divide by 128
		5h	Divide by 256
		6h	Reserved
		7h	Reserved
2-0	CLKDIV		Divider to generate I2S*n*_CLK (bit-clock). The system clock (or DSP clock) to the I2S is divided down by the configured value to generate the bit clock. (Has no effect when I2S is configured as slave device).
		0	Divide by 2.
		1h	Divide by 4.
		2h	Divide by 8.
		3h	Divide by 16.
		4h	Divide by 32.
		5h	Divide by 64.
		6h	Divide by 128.
		7h	Divide by 256.

10.3.3 I2S*n* Transmit Left Data 0 Register (I2STXLT0)

The I2S*n* transmit left data 0 register (I2STXLT0) is shown in Figure 10-15 and described in Table 10-19.

Each I2S module has two double-word (32-bit) transmit data registers to hold left and right channel data respectively. Each double-word register is accessible as two 16-bit registers by the CPU or as a single double-word register by the DMA for efficient data transfer.

Figure 10-15. I2S*n* Transmit Left Data 0 Register (I2STXLT0)

15	0
DATA	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 10-19. I2S*n* Transmit Left Data 0 Register (I2STXLT0) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Transmit left data lower 16 bits.

10.3.4 I2S*n* Transmit Left Data 1 Register (I2STXLT1)

The I2S*n* transmit left data 1 register (I2STXLT1) is shown in Figure 10-16 and described in Table 10-20.

Figure 10-16. I2S*n* Transmit Left Data 1 Register (I2STXLT1)

15	0
DATA	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 10-20. I2S*n* Transmit Left Data 1 Register (I2STXLT1) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Transmit left data upper 16 bits.

10.3.5 I2S*n* Transmit Right Data 0 Register (I2STXRT0)

TheI2S*n* transmit right data 0 register (I2STXRT0) is shown in Figure 10-17 and described in Table 10-21.

Figure 10-17. I2S*n* Transmit Right Data 0 Register (I2STXRT0)

15	0
DATA	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 10-21. I2S*n* Transmit Right Data 0 Register (I2STXRT0) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Transmit right data lower 16 bits.

10.3.6 I2S*n* Transmit Right Data 1 Register (I2STXRT1)

The I2S*n* transmit right data 1 register (I2STXRT1) is shown in Figure 10-18 and described in Table 10-22.

Figure 10-18. I2S*n* Transmit Right Data 1 Register (I2STXRT1)

15	0
DATA	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 10-22. I2S*n* Transmit Right Data 1 Register (I2STXRT1) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Transmit right data upper 16 bits.

10.3.7 I2Sn Interrupt Flag Register (I2SINTFL)

The I2S*n* interrupt flag register (I2SINTFL) is shown in Figure 10-19 and described in Table 10-23.

Figure 10-19. I2S*n* Interrupt Flag Register (I2SINTFL)

15	8
Reserved	
R-0	

7	6	5	4	3	2	1	0
Reserved		XMITSTFL	XMITMONFL	RCVSTFL	RCVMONFL	FERRFL	OUERR
R-0		R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 10-23. I2S*n* Interrupt Flag Register (I2SINTFL) Field Descriptions

Bit	Field	Value	Description
15-6	Reserved	0	Reserved.
5	XMITSTFL		Stereo data transmit flag. Used only when the MONO bit 12 in the I2SSCTRL register = 0 (Stereo mode). This bit is cleared on read.
		0	No pending stereo transmit interrupt.
		1	Stereo transmit interrupt pending. Write new data value to I2S Transmit Left and Right data 0 and 1 registers.
4	XMITMONFL		Mono data transmit flag. Used only when the MONO bit 12 in the I2SSCTRL register = 1 (Mono mode). This bit is cleared on read.
		0	No pending mono transmit interrupt.
		1	Mono transmit interrupt pending. Write new data value to Transmit Left Data 0 and 1 registers.
3	RCVSTFL		Stereo data receive flag. Used only when the MONO bit 12 in the I2SSCTRL register = 0 (Stereo mode). This bit is cleared on read.
		0	No pending stereo receive interrupt.
		1	Stereo receive interrupt pending. Read Receive Left and Right data 0 and 1 registers.
2	RCVMONFL		Mono data receive flag. Used only when the MONO bit 12 in the I2SSCTRL register = 1 (Mono mode). This bit is cleared on read.
		0	No pending mono receive interrupt.
		1	Mono receive interrupt pending. Read Receive Left data 0 and 1 registers.
1	FERRFL		Frame-synchronization error flag. This bit is cleared on read.
		0	No frame-synchronization errors.
		1	Frame-synchronization error(s) occurred.
0	OUERRFL		Overrun or Underrun condition. This bit is cleared on read.
		0	No overrun/under-run errors.
		1	The data registers were not read from or written to before the receive/transmit buffer was overwritten.

10.3.8 I2S*n* Interrupt Mask Register (I2SINTMASK)

The I2S*n* interrupt mask register (I2SINTMASK) is shown in Figure 10-20 and described in Table 10-24.

Figure 10-20. I2S*n* Interrupt Mask Register (I2SINTMASK)

15	8
Reserved	
R-0	

7	6	5	4	3	2	1	0
Reserved		XMITST	XMITMON	RCVST	RCVMON	FERR	OUERR
R-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 10-24. I2S*n* Interrupt Mask Register (I2SINTMASK) Field Descriptions

Bit	Field	Value	Description
15-6	Reserved	0	Reserved.
5	XMITST		Enable stereo left/right transmit data interrupt. Used only when the MONO bit 12 in the I2SSCTRL register = 0 (Stereo mode). This bit is cleared on read.
		0	Disable stereo TX data interrupt.
		1	Enable stereo TX data interrupt.
4	XMITMON		Enable mono left transmit data interrupt. Used only when the MONO bit 12 in the I2SSCTRL register = 1 (Mono mode). This bit is cleared on read.
		0	Disable mono TX data interrupt.
		1	Enable mono TX data interrupt.
3	RCVST		Enable stereo left/right receive data interrupt. Used only when the MONO bit 12 in the I2SSCTRL register = 0 (Stereo mode). This bit is cleared on read.
		0	Disable stereo RX data interrupt.
		1	Enable stereo RX data interrupt.
2	RCVMON		Enable mono left receive data interrupt. Used only when the MONO bit 12 in the I2SSCTRL register = 1 (Mono mode). This bit is cleared on read.
		0	Disable mono RX data interrupt.
		1	Enable mono RX data interrupt.
1	FERR		Enable frame sync error.
		0	Disable frame-synchronization error interrupt.
		1	Enable frame-synchronization error interrupt.
0	OUERR		Enable overrun or underrun condition.
		0	Disable overrun/underrun error interrupt.
		1	Enable overrun/underrun error interrupt.

10.3.9 *I2Sn Receive Left Data 0 Register (I2SRXLT0)*

The I2S*n* receive left data 0 register (I2SRXLT0) is shown in Figure 10-21 and described in Table 10-25.

Each I2S module has two double-word (32-bit) receive data registers to hold left and right channel data respectively. Each double-word register is accessible as two 16-bit registers by the CPU or as a single double-word register by the DMA for efficient data transfer.

Figure 10-21. I2S*n* Receive Left Data 0 Register (I2SRXLT0)

15 ... 0
DATA
R/W-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 10-25. I2S*n* Receive Left Data 0 Register (I2SRXLT0) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Receive left data lower 16 bits.

10.3.10 *I2Sn Receive Left Data 1 Register (I2SRXLT1)*

The I2S*n* receive left data 1 register (I2SRXLT1) is shown in Figure 10-22 and described in Table 10-26.

Figure 10-22. I2S*n* Receive Left Data 1 Register (I2SRXLT1)

15 ... 0
DATA
R/W-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 10-26. I2S*n* Receive Left Data 1 Register (I2SRXLT1) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Receive left data upper 16 bits.

10.3.11 I2S*n* Receive Right Data 0 Register (I2SRXRT0)

The I2S*n* receive right data 0 register (I2SRXRT0) is shown in Figure 10-23 and described in Table 10-27.

Figure 10-23. I2S*n* Receive Right Data 0 Register (I2SRXRT0)

15	0
DATA	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 10-27. I2S*n* Receive Right Data 0 Register (I2SRXRT0) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Receive right data lower 16 bits.

10.3.12 I2Sn Receive Right Data 1 Register (I2SRXRT1)

The I2S*n* receive right data 1 register (I2SRXRT1) is shown in Figure 10-24 and described in Table 10-28.

Figure 10-24. I2S*n* Receive Right Data 1 Register (I2SRXRT1)

15	0
DATA	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 10-28. I2S*n* Receive Right Data 1 Register (I2SRXRT1) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Receive right data upper 16 bits.

Successive Approximation (SAR) Analog-to-Digital Converter (ADC)

This document provides an overview of the Successive Approximation Register (SAR) analog-to-digital Converter (ADC) on the digital signal processor (DSP). The SAR is a 10-bit ADC using a switched capacitor architecture that converts an analog input signal to a digital value at a maximum rate of 64 ksps for use by the DSP. This SAR module supports six channels that are connected to four general purpose analog pins (GPAIN [3:0]), which can also be used as general-purpose digital outputs.

11.1 Introduction

This following sections provide an overview of the successive approximation (SAR) analog-to-digital converter (ADC) on the digital signal processor (DSP).

11.1.1 Purpose of the 10-bit SAR

The SAR in the device is a 10-bit ADC using a switched capacitor architecture that converts an analog input signal to a digital value at a maximum rate of 64 kilo samples per second (ksps) for use by the DSP. This SAR module supports six channels that are connected to four general-purpose analog pins (GPAIN [3:0]) that can be used as general-purpose outputs.

11.1.2 Features

- Up to 64 ksps
- Single conversion and continuous back-to-back conversion modes
- Interrupt driven or polling conversion or DMA event generation
- Internal configurable reference voltages of: V_{DD_ANA} or bandgap_1.0V or bandgap_0.8V
- Software controlled power down
- Individually configurable general-purpose digital outputs

11.1.3 Supported Use Case Statement

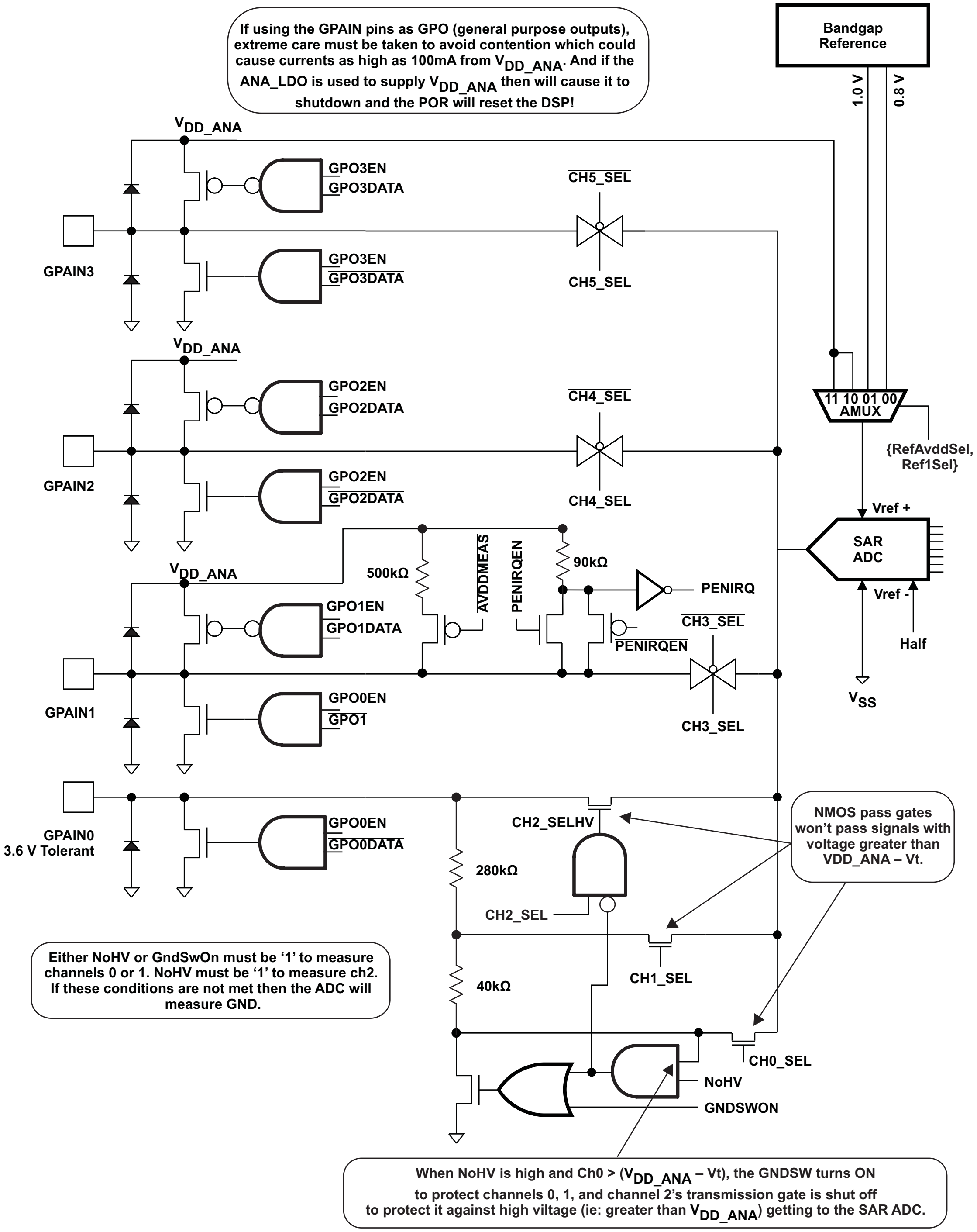
- Measure battery voltage, internal analog voltage (V_{DDA_ANA}), and volume control by measuring across a potentiometer
- 4-wire resistive touch screen coordinate pair measurement and pen down interrupt
- General-purpose outputs that can be driven high or low (except for GPAIN0, which only drives low)
- General-purpose voltage measurement

11.1.4 Industry Standard(s) Compliance Statement

This peripheral is not intended to conform to any specific industry standard.

11.1.5 Functional Block Diagram

Figure 11-1. SAR Converter



If using the GPAIN pins as GPO (general purpose outputs), extreme care must be taken to avoid contention which could cause currents as high as 100mA from V_{DD_ANA}. And if the ANA_LDO is used to supply V_{DD_ANA} then will cause it to shutdown and the POR will reset the DSP!

NMOS pass gates won't pass signals with voltage greater than VDD_ANA – Vt.

Either NoHV or GndSwOn must be '1' to measure channels 0 or 1. NoHV must be '1' to measure ch2. If these conditions are not met then the ADC will measure GND.

When NoHV is high and Ch0 > (V_{DD_ANA} – Vt), the GNDSW turns ON to protect channels 0, 1, and channel 2's transmission gate is shut off to protect it against high viltage (ie: greater than V_{DD_ANA}) getting to the SAR ADC.

11.2 SAR Architecture

The 10-bit successive approximation analog-to-digital module in the device converts an analog input signal to a digital value for use by the DSP. The SAR module supports six channels, VIN[5:0]. These channels are connected to four general purpose analog pins, GPAIN[3:0]. (See Figure 11-1.) All general purpose analog pins can be used as general-purpose outputs by setting the corresponding GPIO bits in the SAR A/D Pin Control Register.

Once a conversion is initiated, the programmer must wait until the conversion completes before selecting another channel or initiating a new conversion. To indicate that a conversion is in progress, the ADCBUSY bit field is set. After the conversion completes, the ADCBUSY bit field changes from 1 to 0, indicating that the conversion data is available. The DSP can then read the data from the ADCDAT bits in the SAR A/D Data Register (SARDATA). The value of the CHSEL bit in SAR A/D Control Register (SARCTRL) is reproduced in the CHAN bit of the SARDATA register, so that the DSP can identify which samples were acquired from which channel.

A DMA event is also generated at the end of every conversion.

11.2.1 SAR Clock Control

The SAR A/D module can operate at a maximum clock rate of 2 MHz (500 ns) and takes 32 clocks cycles to convert a value. This results in a maximum sample rate of 64 ksps. The following equations describe the relationship between the A/D programmable control registers:

SAR A/D Clock Frequency = (System Clock Frequency) / (SystemClkDivisor + 1) ≤ 2 MHz

SAR A/D Conversion Time = (SAR A/D Clock Period * 32)

11.2.2 Memory Map

Table 11-1. SAR Memory Map

Address	Acronym	Description
7012h	SARCTRL	SAR A/D Control Register
7014h	SARDATA	SAR A/D Data Register
7016h	SARCLKCTRL	SAR A/D Clock Control Register
7018h	SARPINCTRL	SAR A/D Reference and Pin Control Register
701Ah	SARGPOCTRL	SAR A/D GPO Control Register

11.2.3 Signal Descriptions

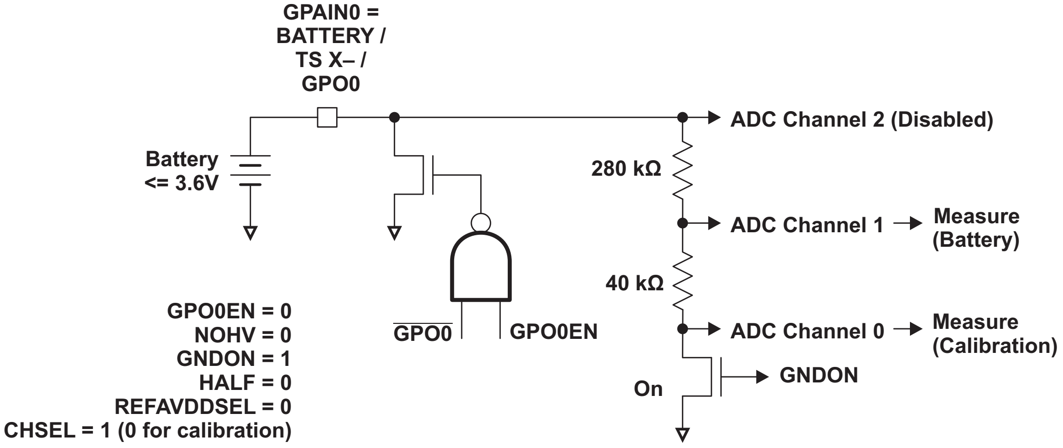
The device's GPAIN[3:0] pins can be configured as inputs to the SAR ADCs or they can be configured as general-purpose outputs that can be driven high or low (excluding GPAIN0 that can only be driven low). The SAR inputs can be used for battery measurement, internal voltage measurement, volume control, and touch screen control. GPAIN[0] is capable of accepting analog input voltage from 0 V up to 3.6 V while GPAIN[1:3] can accept a range of 0 V to V_{DDA_ANA}.

11.2.4 Battery Measurement

The SAR can be configured to measure a battery using GPAIN0.

To measure a battery that has less than 3.6 V, first connect the battery to GPAIN0 and set the ground switch (GNDON) bit of SAR channel 0. Next, calibrate the measurement by sampling the voltage at SAR Channel 0 (set CHAN to Channel 0). Channel 0 should now be tied to ground with GNDON set to 1. If there is an offset on Channel 0, this needs to be applied to the battery reading that can be obtained by switching to channel 1 and sampling the battery voltage through the voltage divider. The voltage divider divides the value from channel 1 by a factor of 8 (see data manual for limits) before being sampled by the SAR ADC. (See Figure 11-2.) After measuring the battery, the ground switch transistor should be shut off to eliminate current draw from the battery.

Figure 11-2. Battery Measurement

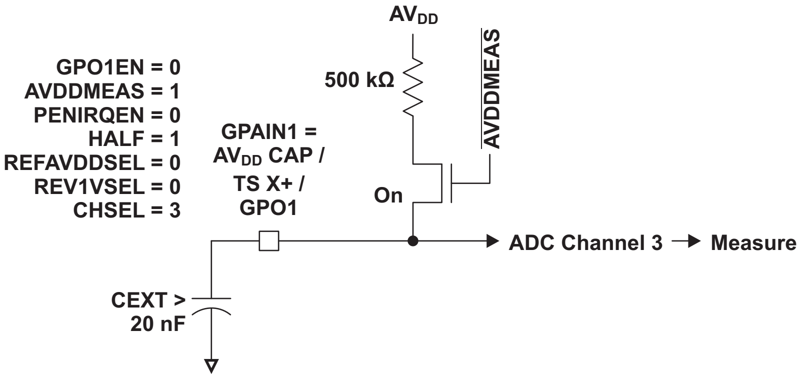


11.2.5 *Internal Voltage Measurement*

Using GPAIN1, the SAR can measure the internal voltage of V_{DDA_ANA} on AVDDMEAS channel 3 of the SAR.

To measure the internal AV_{DD}, set the internal voltage reference by setting in SAR A/D Reference and Pin Control Register (SARPINCTRL). A 20 nF cap is recommended to be connected between GPAIN1 and GND to provide low pass filtering and less measurement noise. Next, sample SAR channel 3. Selecting HALF = 1 has the effect of reducing the ADC's input sampled voltage in half. Therefore, with AV_{DD} (i.e., V_{DDA_ANA}) at its max of 1.43 V, divided by two is 0.715 V. Then, with the ADC's VREF set to bandgap_0.8 V, the dynamic range is optimum. See Figure 11-3.

Figure 11-3. Voltage Measurement

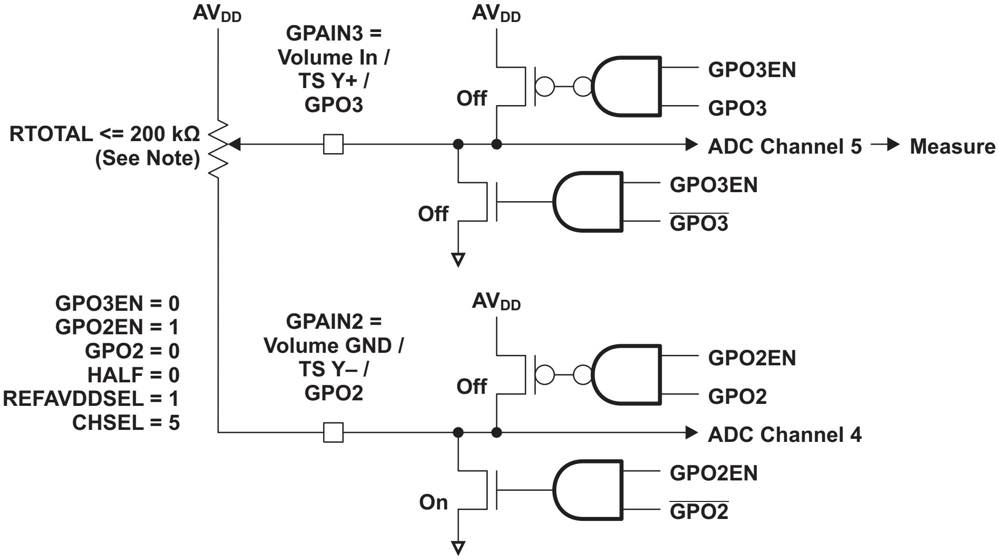


11.2.6 *Volume Control*

The SAR can be used to sample a volume control potentiometer connected across GPAIN2 and GPAIN3. Note that other combinations of the GPAIN[3:0] could be used to perform this function. We have chosen GPAIN2 and GPAIN3 for this example.

To use the SAR for volume control, place a potentiometer across GPAIN3 and GPAIN2, then ground GPAIN2 by clearing GPO2 and sample SAR channel 5 voltage. Use the settings in Figure 11-4.

Figure 11-4. Voltage Control



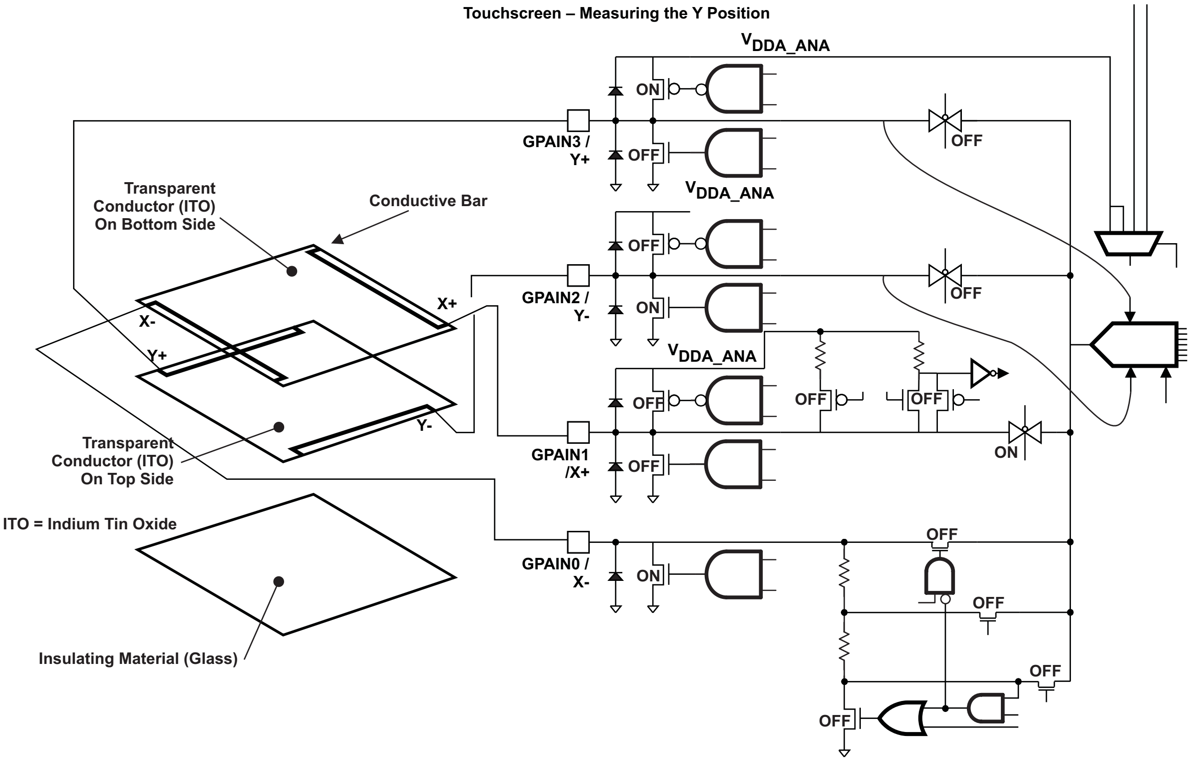
11.2.7 Touch Screen Digitizing

Using all 4 GPAIN pins, the SAR can be used for digitizing touch screen coordinates. With GPAIN3 as Y+, GPAIN2 as Y-, GPAIN1 as X+, and GPAIN0 as X-.

To measure Y position, enable GPAIN3 and GPAIN2 as general-purpose outputs using GPO3EN=1 and GPO2EN=1. Then, ground GPAIN2 by clearing GPO2 and drive GPAIN3 high by setting GPO3. Let the touch panel settle (duration depends on the bypass caps you have at the X+, X-, Y+, Y- terminals of the touch screen) and measure the voltage at GPAIN1 using SAR channel 3.

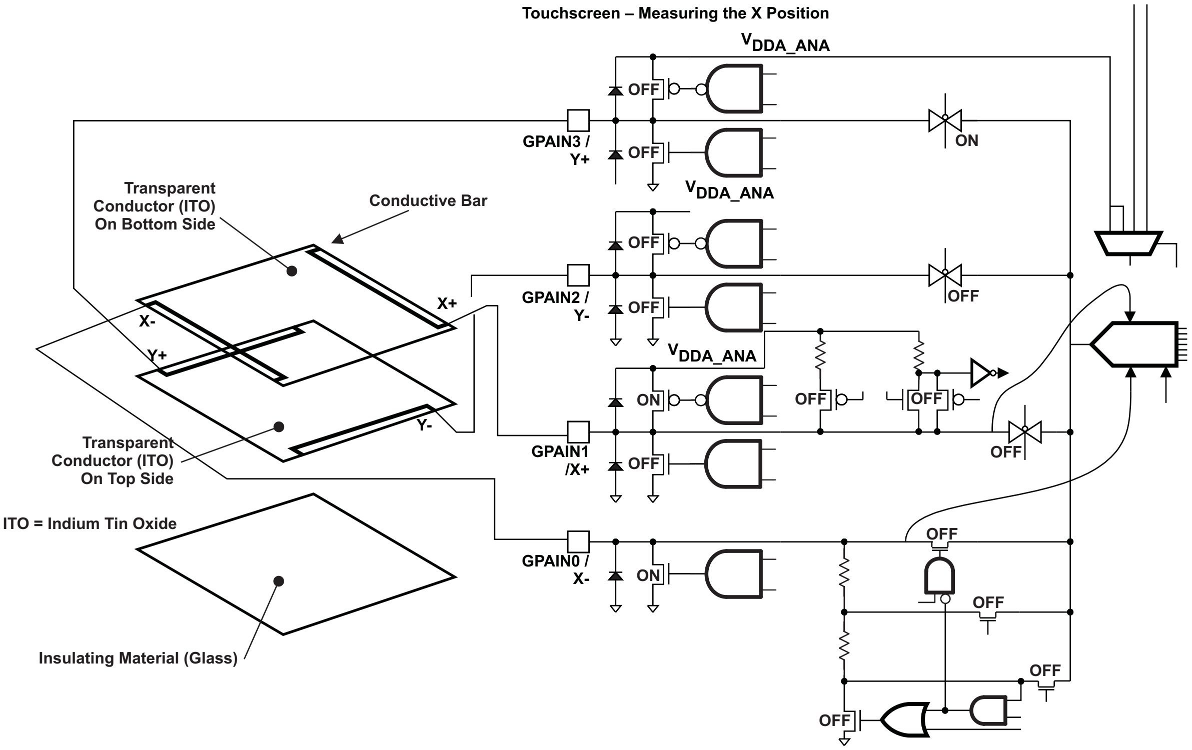
NOTE: It is recommended that an external LDO be used to supply power to V_{DDA_ANA} rather than using ANA_LDO. When ANA_LDO is used to supply power to V_{DDA_ANA}, the pins GPAIN[3:1] cannot be used as general-purpose outputs (driving high) since the maximum current capability of the ANA_LDO can be exceeded. The ISD parameter of the ANA_LDO is too low to drive any realistic load on the GPAIN[3:1] pins while also supplying the PLL through V_{DDA_PLL} and the SAR through V_{DDA_ANA}. Using AVA_LDO to supply power to V_{DDA_ANA} in such a case may result in the on-chip power-on reset (POR) resetting the chip.

Figure 11-5. Y Position



To measure X position, enable GPAIN1 and GPAIN0 using GPO1EN=1 and GPO0EN=1. Then, ground GPAIN0 by clearing GPO0 and drive GPAIN1 high by setting GPO1. Let the touch panel settle, then measure the voltage at GPAIN3 using SAR channel 5.

Figure 11-6. X Position

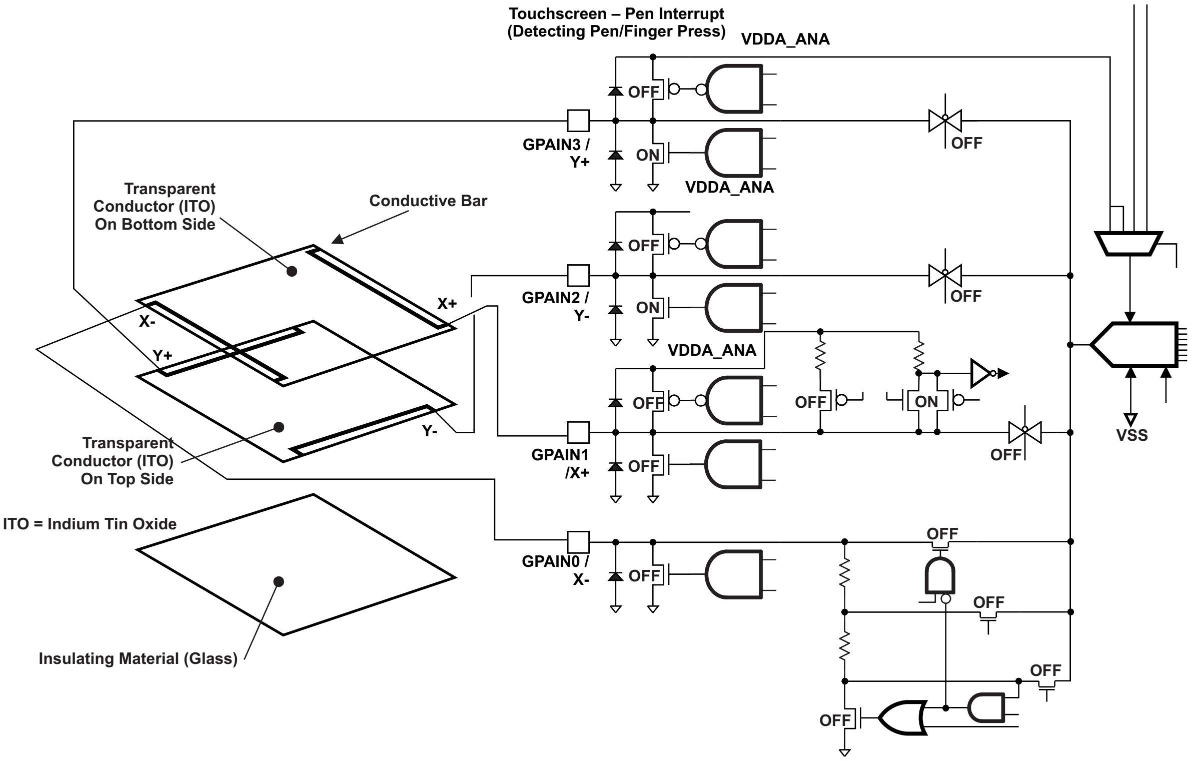


11.2.8 Touch Screen : Pen Press Interrupts

To detect when the touch screen is touched, the SAR peripheral has a PENIRQ feature. This feature makes it possible to detect touch events without continuous ADC polling.

The SAR should be configured as shown in Figure 11-7.

Figure 11-7. Pen Interrupt



The DSP should be configured to allow SAR interrupts. When the touch screen is not pressed, a pullup resistor biases the PENIRQ buffer high so that an interrupt is not generated. When the touch screen is pressed it creates a path to ground that is lower impedance than the pullup resistor. Therefore, the PENIRQ buffer generates an interrupt and the DSP can then take the steps necessary to digitize the X & Y coordinates.

11.2.9 General-Purpose Output

GPAIN[3:0] can be configured as general-purpose outputs. This is accomplished by enabling the GPAIN pin as an output in the SAR A/D GPO Control Register (SARGPOCTRL). After enabling the GPO you can set the output as grounded or driven high. In the case where V_{DDA_ANA} is supplied by the ANA_LDOO, care must be taken to avoid shorting GPAIN pins to ground as this will cause the maximum current of the ANA_LDOO to be exceeded and the POR circuit to reset the DSP. The total current from all GPAIN[3:0] pins to ground should never exceed Imax of the ANA_LDOO. For more information, see the device-specific data manual. In the case where V_{DDA_ANA} is supplied by some source other than the on-chip ANA_LDO, this is not an issue.

11.2.10 Reset Considerations

The SAR can only be reset by a hardware reset.

11.2.10.1 Software Reset Considerations

A software reset (such as a reset generated by the emulator) will not cause the SAR controller registers to be altered. After a software reset, the SAR controller continues to operate as it was configured prior to the reset.

There is no peripheral reset for the SAR.

11.2.10.2 Hardware Reset Considerations

A hardware reset of the processor causes the SAR controller registers to return to their default values after reset.

11.2.11 A/D Conversion

To start an analog to digital conversion the following steps must be executed:

1. Set SAR clock to be less than or equal to 2 MHz in the SAR A/D clock Control Register (SARCLKCTRL) for fastest conversion rate and operation of the A/D module.
   A/D function clock = (Sys Clk)/(ADCCLKDIV+1)
2. Write to the SARPINCTRL register (7018h) to power up the SAR circuits and select the SAR reference voltage.
3. Write a 1 to the ADCSTRT bit in the SARCTRL register and the desired channel for the conversion in the CHSEL bit field.
4. Read the ADCBUSY bit in the SARDATA register to ensure it is set to 1 to indicate the start of conversion. Due to delays between the CPU write instruction and the actual write to the SAR A/D registers the ADCBUSY bit must be set before proceeding.
5. ADCSTRT in the SARCTRL register and ADCBUSY bit in the SARDATA register are set to 0 to indicate the end of the conversion sequence.
6. Once ADCBUSY bit in the SARDATA register is set to 0, the SAR A/D Data Register contains the channel converted in the CHSEL bit field and the actual converted value in the ADCDAT bit field.

11.2.12 Interrupt Support

11.2.12.1 Interrupt Events and Requests

The SAR peripheral generates DSP interrupts every time an ADC conversion is completed and data is available to be read by the DSP. Additionally, when connected to a touch screen device, the SAR can be configured to detect when the touch screen is pressed and generate an interrupt without having to perform continuous conversion to poll the touch screen.

11.2.13 Emulation Considerations

The SAR controller is not affected by emulation halt events (such as breakpoints).

11.3 Registers

Table 11-2 list the memory mapped registers associated with the successive approximation (SAR) analog-to-digital converter (ADC).

Table 11-2. SAR A/D Memory Mapped Registers

CPU Word Address	Acronym	Register Description	Section
7012h	SARCTRL	SAR A/D Control Register	Section 11.3.1
7014h	SARDATA	SAR A/D Data Register	Section 11.3.2
7016h	SARCLKCTRL	SAR A/D Clock Control Register	Section 11.3.3
7018h	SARPINCTRL	SAR A/D Reference and Pin Control Register	Section 11.3.4
701Ah	SARGPOCTRL	SAR A/D GPO Control Register	Section 11.3.5

11.3.1 SAR A/D Control Register (SARCTRL)

The SAR A/D control register (SARCTRL) selects the channel number and indicates the start of a conversion.

The SAR A/D control register (SARCTRL) is shown in Figure 11-8 and described in Table 11-3.

Figure 11-8. SAR A/D Control Register (SARCTRL)

15	14 12	11	10	9 0
ADCSTRT	CHSEL	MULTICH	SNGLCONV	Reserved
RW, +0	RW, +000	RW,+0	RW,+0	RW, +0000000000

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 11-3. SAR A/D Control Register (SARCTRL) Field Descriptions

Bit	Field	Value	Description
15	ADCSTRT		Start Conversion
		0	No conversion.
		1	Start conversion cycle by setting START signal.
14-12	CHSEL		Channel Select
		0	Channel CH0 is selected.
		1h	Channel CH1 is selected.
		2h	Channel CH2 is selected.
		3h	Channel CH3 is selected.
		4h	Channel CH4 is selected.
		5h	Channel CH5 is selected.
		6h-7h	All channels are off.
11	MULTICH		Multi Channel operation.
		0	Normal Mode.
		1	In this mode, the SAR state machine is optimized to give more time to sampling the analog input. This mode could possibly improve measurements in cases where the ADC input has abrupt voltage changes such as when changing from one input channel to another. The additional time given to sampling does not affect the 32 cycles for conversion, but it does come at the expense of settling time for the ADC’s internal comparator. Therefore, this mode should not be used unless the above situation exists and it is determined to improve the measurements.
10	SNGLCONV		Single Conversion mode.
		0	Continuously perform back-to-back conversions, as long as ADCSTRT is set.
		1	Perform one conversion and stop. ADCSTRT must be cleared and then set high to perform another conversion.
9-0	Reserved	0	Reserved.

11.3.2 SAR A/D Data Register (SARDATA)

The SAR A/D data register (SARDATA) indicates if a conversion is in process, the actual digital data converted from the analog signal, and the channel belonging to this conversion.

The SAR A/D data register (SARDATA) is shown in Figure 11-9 and described in Table 11-4.

Figure 11-9. SAR A/D Data Register (SARDATA)

15	14 12	11 10	9 0
ADCBUSY	CHAN	Reserved	ADCDAT
R, +0	R, +111	R, +00	R, +0000000000

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 11-4. SAR A/D Data Register (SARDATA) Field Descriptions

Bit	Field	Value	Description
15	ADCBUSY		ADC Converter Busy. The ADCBUSY bit will be set three SAR clock cycles after the ADCSTRT bit is set.
		0	ADCDAT available.
		1	ADCBUSY performing a conversion. After ADCSTRT is high, the ADCBUSY becomes high.
			This will always read 0 when STATUSMASK=1.
14-12	CHAN		Channel Select
		0	Channel CH0 is selected.
		1h	Channel CH1 is selected.
		2h	Channel CH2 is selected.
		3h	Channel CH3 is selected.
		4h	Channel CH4 is selected.
		5h	Channel CH5 is selected.
		6h-7h	Reserved
			These bits will always read 000 when STATUSMASK = 1.
11-10	Reserved	0	Reserved.
9-0	ADCDAT	0-3FFh	Converter Data.

11.3.3 SAR A/D Clock Control Register (SARCLKCTRL)

The SAR A/D clock control register (SARCLKCTRL) sets the clock divider to control the speed of conversion. The clock rate of the SAR module must not exceed 2MHz.

The SAR A/D clock control register (SARCLKCTRL) is shown in Figure 11-10 and described in Table 11-5.

Figure 11-10. SAR A/D Clock Control Register (SARCLKCTRL)

15	14	0
Reserved	ADCCLKDIV	
R, +0	RW, +111111111111111	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 11-5. SAR A/D Clock Control Register (SARCLKCTRL) Field Descriptions

Bit	Field	Value	Description
15	Reserved	0	Reserved.
14-0	ADCCLKDIV	0-7FFFh	System Clock Divisor This specifies the divider rate of the system clock: F_{SAR_Clock} = (F_{System_Clock}) /(ADCCLKDIV[14:0] + 1) Allows for divide-by-1 up to divide-by-32768.

11.3.4 SAR A/D Reference and Pin Control Register (SARPINCTRL)

The SAR A/D reference and pin control register (SARPINCTRL) controls the SAR's reference voltage and the circuits surrounding the GPAIN pins. The SAR's reference voltage determines the voltage at the ADC input that corresponds to the fullscale output code (ie: 1111111111b). Note, however, that due to the circuitry between the ADC input and the GPAIN[3:0] pins, the voltage at the ADC input isn't necessarily the same as the voltage at the GPAIN[3:0] pins. For example, the voltage divider on GPAIN0/Channel 1 scales the voltage of the signal by a factor of 8 before it arrives at the ADC. It is important to select the best voltage reference according to the voltage range of signal that will be digitized by the ADC so that the best resolution is obtained.

The SAR A/D reference and pin control register (SARPINCTRL) is shown in Figure 11-11 and described in Table 11-6.

Figure 11-11. SAR A/D Reference and Pin Control Register (SARPINCTRL)

15	14	13	12	11	10	9	8
Reserved	STATUSMASK	PWRUPBIAS	SARPWRUP	Reserved	REFBUFFEN	REFLVSEL	REFAVDDSEL
R, +0	Rw,+0	RW,+0	RW,+0	R,+0	RW, +0	RW, +0	RW, +0

7		5	4	3	2	1	0
Reserved			TOUCHSCREENMODE	AVDDMEAS	Reserved	GNDON	HALF
R,+0			RW, +0	RW, +0	RW, +0	RW, +0	RW, +0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 11-6. SAR A/D Reference and Pin Control Register (SARPINCTRL) Field Descriptions

Bit	Field	Value	Description
15	Reserved	0	Reserved.
14	STATUSMASK		Asserting this bit causes bits 12-15 of the SAR_DATA register to be forced to 0000. Those four bits correspond to the ChannelSelected and ADCBusy bits. The purpose for clearing/masking them is so that DMA transfers from the SAR to a memory buffer do not include those status bits and thus post-processing of the memory buffer is not required to strip the status bits out of the sample set. For example, if the SAR & DMA collect a 2k sample set into memory to perform an FFT on the sampled data, you wouldn't want the ChannelSelected status bits to be in the data when the FFT is performed.
		0	The SAR_DATA register includes the status info in bits 12-15.
		1	The SAR_DATA register bits 12-15 are always read as 0000.
13	PWRUPBIAS		Enables or disables the current bias circuit that is needed for the SAR to perform A/D conversions.
		0	Powered Down. Low power setting.
		1	Powered Up. Required setting for performing A/D conversions.
12	SARPWRUP		Enables or disables the analog power to the SAR.
		0	SAR analog Powered down.
		1	SAR analog power present.
11	Reserved	0	Reserved.
10	REFBUFFEN		Reference Buffer enable. The reference buffer can be disabled to save power when the ADC's VREF is set to VDDA_ANA (REFAVDDSEL=1) or when VREF is provided by the TOUCHSCREENMODE pins (TOUCHSCREENMODE=1). The reference buffer must be enabled whenever one of the bandgap reference voltages are used (ie: REFAVDDSEL=0). REFBUFEN can be 0 or 1 when TOUCHSCREENMODE=1.
		0	Reference Buffer is disabled. Low power setting.
		1	Reference Buffer is enabled. Required when using bandgap generated VREF.
9	REFLVSEL		Bandgap-based reference voltage value select. The on-chip bandgap provides two references to the SAR peripheral: 0.8v & 1.0v. This register is used to select which bandgap reference voltage is used when REFAVDDSEL=0. In general, the lowest VREF should be used to get the best resolution from the converter. However, VREF should always be greater than the input signal else clipping will occur.
		0	Bandgap-Based Reference Voltage set to 0.8V.
		1	Bandgap-Based Reference Voltage set to 1V.

Table 11-6. SAR A/D Reference and Pin Control Register (SARPINCTRL) Field Descriptions (continued)

Bit	Field	Value	Description
8	REFAVDDSEL		ADC Reference Voltage Select. When asserted, this register selects VDDA_ANA as the voltage reference for the SAR ADC. Otherwise, one of the two selectable bandgap references will be used. This register has no effect when TOUCHSCREENMODE=1.
		0	Reference Voltage based on Bandgap. The voltage value of the Bandgap reference is dictated by REFLVSEL.
		1	Reference Voltage set to Analog Voltage (VDD_ANA).
7-6	Reserved	0	Reserved.
5	Reserved	0	Reserved must write 0.
4	TOUCHSCREENMODE		Enables Touch Screen Mode. In this mode, the SAR detects which coordinate, X or Y, is being measured based on the GPOxEN and GPOxDATA settings, and switches the ADC's VREF+ and VREF- to the appropriate GPAIN[3:0] pins to reduce offset and gain errors caused by the dc current flowing thru the touch screen and the drop across the GPO output transistors. See Figure 11-5 and Figure 11-6 to see how the VREF+ and VREF- are affected by this mode.
		0	TOUCHSCREENMODE is Disabled.
		1	TOUCHSCREENMODE is Enabled.
3	AVDDMEAS		Enable measurement of internal analog voltage (V_{DD_ANA}) on SAR Channel 3.
		0	PMOS switch on channel 3 is open, thus VDDA_ANA is not connected to channel 3 ADC input.
		1	PMOS switch on channel 3 is closed, thus VDDA_ANA is connected to channel 3 ADC input thru a pullup resistor to enable measuring the internal VDDA_ANA voltage. Note, when measuring VDDA_ANA, an independent voltage reference is needed for the ADC. So one of the two bandgap voltages should be used. Half mode will also be necessary since VDDA_ANA is greater than the two bandgap voltage references.
2	Reserved	0	Reserved.
1	GNDON		Ground SAR Analog Channel 0 and introduce a voltage resistor divider network in SAR Channel 1.
		0	SAR Analog Channel 0 is not grounded.
		1	SAR Analog Channel 0 grounded. Introduces a divider into the SAR Channel 1 input of 1/8 * GPAIN0. See datasheet for tolerance specs on the resistor divider.
0	HALF		Divides the ADC analog input by two before doing the conversion. The attenuation is accomplished by only charging half of the SAR ADC's internal capacitive array during the sample phase, then the whole capacitive array is used for the successive approximation conversion. By sampling with half the capacitance and comparing against VREF with the full capacitance, the input voltage is attenuated by a factor of 2.
		0	A-to-D conversion is based on Vin.
		1	A-to-D conversion is based on Vin / 2.

11.3.5 SAR A/D GPO Control Register (SARGPOCTRL)

The SAR A/D general-purpose output control register (SARGPOCTRL) sets the corresponding GPAIN pins to general-purpose outputs or analog inputs. In general-purpose output mode, the GPAIN pins can be individually driven high or low.

The SAR A/D GPO control register (SARGPOCTRL) is shown in Figure 11-12 and described in Table 11-7.

Figure 11-12. SAR A/D GPO Control Register (SARGPOCTRL)

15	10	9	8
Reserved		PENIRQ	PENIRQEN
R, +00000		RW, +0	RW, +0

7	6	5	4	3	2	1	0
GPO3EN	GPO2EN	GPO1EN	GPO0EN	GPO3	GPO2	GPO1	GPO0
RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 11-7. SAR A/D GPO Control Register (SARGPOCTRL) Field Descriptions

Bit	Field	Value	Description
15-10	Reserved	0	Reserved.
9	PENIRQ		Pen Interrupt Request Status.
		0	No pen input detected.
		1	Pen input detected.
8	PENIRQEN		Pen Interrupt Request Enable.
		0	Disable the pen interrupt.
		1	Enable the pen interrupt and route to the CPU’s SAR interrupt signal.
7	GPO3EN		Enable General Purpose Output on GPAIN3. Allows using GPAIN3 as a general output.
		0	GPAIN3 output driver disabled.
		1	GPAIN3 used as output.
6	GPO2EN		Enable General Purpose Output on GPAIN2. Allows using GPAIN2 as a general output.
		0	GPAIN2 output driver disabled.
		1	GPAIN2 used as output.
5	GPO1EN		Enable General Purpose Output on GPAIN1. Allows using GPAIN1 as a general output.
		0	GPAIN1 output driver disabled.
		1	GPAIN1 used as output.
4	GPO0EN		Enable General Purpose Output on GPAIN0. Allows using GPAIN0 as a general output
		0	GPAIN0 output driver disabled.
		1	GPAIN0 used as output.
3	GPO3		Drive high or low GPAIN3 when set as General Purpose Output.
		0	GPAIN3 grounded.
		1	GPAIN3 driven high.
2	GPO2		Drive high or low GPAIN2 when set as General Purpose Output.
		0	GPAIN2 grounded.
		1	GPAIN2 driven high.
1	GPO1		Drive high or low GPAIN1 when set as General Purpose Output.
		0	GPAIN1 grounded.
		1	GPAIN1 driven high.
0	GPO0		Ground GPAIN0 when set as General Purpose Output.
		0	GPAIN0 grounded.
		1	GPAIN0 driven high.

11.3.6 Conversion Example

To request a conversion the CPU must execute the following sequence of events:

1. Set SAR clock to be less than or equal to 2MHz in SAR Clock Control Register for fastest conversion rate and operation of the A/D module.
2. Write a “1” to the ADCSTRT bit of the SARCTRL register and the desired channel for conversion in the CHAN bit field in the SARDATA register.
3. ADCBUSY bit of the SARDATA register is set to “1” to indicate the start of A/D conversion.
4. Due to delays between the CPU write instruction and the actual write to the SAR A/D Registers, it is recommended to read the SARDATA register and verify the ADCBUSY bit is set to “1” before proceeding with step 6.
5. ADCSTRT and ADCBUSY bits are set to “0” to indicate the end of the conversion sequence. The SAR A/D module enters stand-by mode to conserve power until event 2 occurs over again.
6. Once ADCBUSY bit is set to “0”, the SARDATA register contains the channel converted in the CHAN bit field and the actual converted value in the ADCDAT bit field

Chapter 12

SPRUH87C–August 2011–Revised March 2012

General-Purpose Input/Output (GPIO)

This chapter describes the features and operations of the general-purpose input/output (GPIO).

Topic **Page**

12.1 Introduction

This document describes the general-purpose input/output (GPIO) on the digital signal processor (DSP).

12.1.1 Purpose of the Peripheral

The GPIO peripheral provides general-purpose pins that can be configured as either inputs or outputs. When configured as an output, you can write to an internal register to control the state driven on the output pin. When configured as an input, you can detect the state of the input by reading the state of the internal register. The GPIO can also be used to send interrupts to the CPU.

12.1.2 Features

The GPIO Peripheral consists of the following features:

- 26 GPIOs.
- Output Set/Clear functionality through writing a single output data register.
- All GPIOs can be configured to generate edge detected interrupts to the CPU.

12.1.3 Industry Standard(s) Compliance Statement

The GPIO peripheral connects to external devices. While it is possible that the software implements some standard connectivity protocol over GPIO, the GPIO peripheral itself is not compliant with any such standards.

12.2 Peripheral Architecture

The following sections describe the GPIO peripheral.

12.2.1 Clock Control

The input clock to the GPIO peripheral is driven by the system clock.

12.2.2 Signal Descriptions

The device supports up to 26 signals, GPIO[31:27], GPIO[20:0]. All GPIO pins are muxed with other signals and have an optional internal pull-down resistor. The mux is controlled in the External Bus Selection Register (EBSR) located at port address 0x1C00h. The routing of the signals take place on the next CPU clock cycle. Before modifying the values of EBSR, you must first clock gated all affected peripherals via the Peripheral Clock Gating Control Register (PCGCR1 and PCGCR2) at addresses 0x1C02h and 0x1C03h. The EBSR can be modified only once after boot process is complete. Continuously switching the EBSR is not supported. Pull-downs are disable or enabled by the Pull-down Inhibit Register (0x1C17h, 0x1C18h and 0x1C19h) located in the data manual. These 32 GPIO are muxed with other signals. For more information on the package pinout and muxing of each GPIO signal, refer to the device-specific data manual.

Due to a variety of different technology devices that can be connected to the GPIO, the levels of logic 0 (low) and logic 1 (high) are not fixed and depend on the supply level. See the device-specific data manual for more information.

Table 12-1. GPIO Terminal Functions

Signal		Type [1] [2]	Other [3] [4]	Description
Name	No.			
SD0_CLK/ I2S0_CLK/ GP[0]	M8	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between SD0, I2S0, and GPIO. For GPIO, it is general-purpose input/output pin 0 (GP[0]). Mux control via the SP0MODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR1 register.
SD0_CMD/ I2S0_FS/ GP[1]	M10	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between SD0, I2S0, and GPIO. For GPIO, it is general-purpose input/output pin 1 (GP[1]). Mux control via the SP0MODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR1 register.
SD0_D0/ I2S0_DX/ GP[2]	J1	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between SD0, I2S0, and GPIO. For GPIO, it is general-purpose input/output pin 2 (GP[2]). Mux control via the SP0MODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR1 register.
SD0_D1/ I2S0_RX/ GP[3]	P6	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between SD0, I2S0, and GPIO. For GPIO, it is general-purpose input/output pin 3 (GP[3]). Mux control via the SP0MODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR1 register.
SD0_D2/ GP[4]	N13	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between SD0 and GPIO. For GPIO, it is general-purpose input/output pin 4 (GP[4]). Mux control via the SP0MODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR1 register.
SD0_D3/ GP[5]	P7	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between SD0 and GPIO. For GPIO, it is general-purpose input/output pin 5 (GP[5]). Mux control via the SP0MODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR1 register.
SD1_CLK/ I2S1_CLK/ GP[6]	M14	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between SD1, I2S1, and GPIO. For GPIO, it is general-purpose input/output pin 6 (GP[6]). Mux control via the SP1MODE bits in the EBSR.
SD1_CMD/ I2S1_FS/ GP[7]	L11	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between SD1, I2S1, and GPIO. For GPIO, it is general-purpose input/output pin 7 (GP[7]). Mux control via the SP1MODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR1 register.
SD1_D0/ I2S1_DX/ GP[8]	M13	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between SD1, I2S1, and GPIO. For GPIO, it is general-purpose input/output pin 8 (GP[8]). Mux control via the SP1MODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR1 register.
SD1_D1/ I2S1_RX/ GP[9]	P10	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between SD1, I2S1, and GPIO. For GPIO, it is general-purpose input/output pin 9 (GP[9]). Mux control via the SP1MODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR1 register.
SD1_D2/ GP[10]	L12	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between SD1 and GPIO. For GPIO, it is general-purpose input/output pin 10 (GP[10]). Mux control via the SP1MODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR1 register.
SD1_D3/ GP[11]	M12	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between SD1 and GPIO. For GPIO, it is general-purpose input/output pin 11 (GP[11]). Mux control via the SP1MODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR1 register.

[1] I = Input, O = Output, Z = High impedance, S = Supply voltage, GND = Ground, A = Analog signal, BH = Bus Holder

[2] Input pins of type I, I/O, and I/O/Z are required to be driven at all times. To achieve the lowest power, these pins must not be allowed to float. When configured as input or high-impedance state, and not driven to a known state, they may cause an excessive IO-supply current. If this is the case, enable IPD/IPU, if applicable, or externally terminate the pins.

[3] IPD = Internal pulldown, IPU = Internal pullup. For more detailed information on pullup/pulldown resistors and situations where external pullup/pulldown resistors are required, see , *Pullup/Pulldown Resistors*.

[4] Specifies the operating I/O supply voltage for each signal

Table 12-1. GPIO Terminal Functions (continued)

Signal		Type [1] [2]	Other [3] [4]	Description
Name	No.			
LCD_D[2]/ GP[12]	J2	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between LCD Bridge and GPIO. For GPIO, it is general-purpose input/output pin 12 (GP[12]). Mux control via the PPMODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR3 register.
LCD_D[3]/ GP[13]	N5	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between LCD Bridge and GPIO. For GPIO, it is general-purpose input/output pin 13 (GP[13]). Mux control via the PPMODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR3 register.
LCD_D[4]/ GP[14]	P2	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between LCD Bridge and GPIO. For GPIO, it is general-purpose input/output pin 14 (GP[14]). Mux control via the PPMODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR3 register.
LCD_D[5]/ GP[15]	N7	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between LCD Bridge and GPIO. For GPIO, it is general-purpose input/output pin 15 (GP[15]). Mux control via the PPMODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR3 register.
LCD_D[6]/ GP[16]	P3	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between LCD Bridge and GPIO. For GPIO, it is general-purpose input/output pin 16 (GP[16]). Mux control via the PPMODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR3 register.
LCD_D[7]/ GP[17]	P8	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between LCD Bridge and GPIO. For GPIO, it is general-purpose input/output pin 17 (GP[17]). Mux control via the PPMODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR3 register.
LCD_D8]/ I2S2_CLK/ GP[18]/ SPI_CLK	P5	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between between LCD Bridge and GPIO. For GPIO, it is general-purpose input/output pin 18 (GP[18]). Mux control via the PPMODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR3 register.
LCD_D[9]/ I2S2_FS/ GP[19]/ SPI_CS0	N10	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between LCD Bridge, I2S2 and GPIO. For GPIO, it is general-purpose input/output pin 19 (GP[19]). Mux control via the PPMODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR3 register.
LCD_D[10]/ I2S2_RX/ GP[20]/ SPI_RX	P9	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between LCD Bridge, I2S2, GPIO and SPI. For GPIO, it is general-purpose input/output pin 20 (GP[20]). Mux control via the PPMODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR3 register.
LCD_D[11]/ I2S2_DX/ GP[27]/ SPI_TX	P11	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between LCD Bridge, I2S2, GPIO, and SPI. For GPIO, it is general-purpose input/output pin 27 (GP[27]). Mux control via the PPMODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR3 register.
LCD_D[12]/ UART_RTS/ GP[28]/ I2S3_CLK	N12	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between LCD Bridge, UART, GPIO, and I2S3. For GPIO, it is general-purpose input/output pin 28 (GP[28]). Mux control via the PPMODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR3 register.
LCD_D[13]/ UART_CTS/ GP[29]/ I2S3_FS	P12	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between LCD Bridge, UART, GPIO, and I2S3. For GPIO, it is general-purpose input/output pin 29 (GP[29]). Mux control via the PPMODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR3 register.
LCD_D[14]/ UART_RXD/ GP[30]/ I2S3_RX	P13	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between LCD Bridge, UART, GPIO, and I2S3. For GPIO, it is general-purpose input/output pin 30 (GP[30]). Mux control via the PPMODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR3 register.

Table 12-1. GPIO Terminal Functions (continued)

Signal		Type [1] [2]	Other [3] [4]	Description
Name	No.			
LCD_D[15]/ UART_TXD/ GP[31]/ I2S3_DX	M11	I/O/Z	IPD DV_{DDIO} BH	This pin is multiplexed between LCD Bridge, UART, GPIO, and I2S3. For GPIO, it is general-purpose input/output pin 31 (GP[31]). Mux control via the PPMODE bits in the EBSR. The IPD resistor on this pin can be enabled or disabled via the PDINHIBR3 register.

12.2.3 GPIO Register Structure

The GPIO configuration registers are grouped into two 16-bit registers for each function. Control of the general-purpose I/O is maintained through a set of I/O memory mapped registers. For detailed information on the GPIO registers, see .

12.2.4 Using a GPIO Signal as an Output

GPIO signals are configured to operate as inputs or outputs by writing the appropriate value to the GPIO direction registers (IODIR1 and IODIR2). This section describes using the GPIO signal as an output.

12.2.4.1 Configuring a GPIO Output Signal

To configure a given GPIO signal as an output, set the bit in (IODIR1 and IODIR2) that is associated with the desired GPIO signal. For detailed information on the GPIO direction registers, see Section 12.3.1.

12.2.4.2 Controlling the GPIO Output Signal State

The GPIO output is controlled by the setting or clearing the OUT bit in the GPIO data out registers (IODATAOUT1 or IODATAOUT2) for the desired GPIO. When the GPIO is configured for output, a write of "1" will make the output high and a write of "0" will make the output low. For detailed information on the GPIO data out registers, see Section 12.3.3.

12.2.5 Using a GPIO Signal as an Input

GPIO signals are configured to operate as inputs or outputs by writing the appropriate value to the IODIR1 or IODIR2 registers. This section describes using the GPIO signal as an input.

12.2.5.1 Configuring a GPIO Input Signal

To configure a given GPIO signal as an input, clear the bit in the GPIO direct register (IODIR1 or IODIR2) that is associated with the desired GPIO signal. For detailed information on GPIO direction registers, see Section 12.3.1.

12.2.5.2 Controlling the GPIO Input Signal State

The current state of the GPIO signals are read using the GPIO data in registers (IOINDATA1 & IOINDATA2).

- For GPIO signals configured as inputs, reading input data register returns the state of the input signal synchronized to the GPIO peripheral (system clock).
- For GPIO signals configured as outputs, reading input data register returns the output value being driven by the device.

To use GPIO input signals as interrupt sources, see Section 12.2.7.

12.2.6 Reset Considerations

The GPIO peripheral is only reset by a hardware reset.

12.2.6.1 Software Reset Considerations

A software reset does not modify the configuration and state of the GPIO signals. Software resets include reset initiated through the emulator, the device's software reset instruction, and the Peripheral Reset Control Register (PRCR) (1C05h). For a detailed description, see Chapter 1.

12.2.6.2 Hardware Reset Considerations

A hardware reset will cause the GPIO configuration to return to the default state, including GPIO pin selection (GPIOs default to input), and data registers to their default states, therefore affecting the configuration and state of the GPIO signals.

12.2.7 Interrupt Support

GPIO peripheral can individually generate an interrupt. The GPIO interrupts are falling or rising edge triggered interrupts.

12.2.7.1 Interrupt Events and Requests

The GPIO signals can be configured to generate an interrupt. The device supports interrupts from the GPIO signals. All GPIO are tied a single interrupt. To determine which GPIO caused the interrupt, the GPIO interrupt flag registers (IOINTFLG1 and IOINTFLG2) must be read. For detailed information on the GPIO interrupt flag registers, see Section 12.3.6.

12.2.7.2 Enabling GPIO Interrupt Events

GPIO interrupt events are enabled in GPIO interrupt enable registers (IOINTEN1 and IOINTEN2). Interrupts can be enabled for each of the 32 GPIO. Setting a "1" to the appropriate GPIO will enable external interrupts for this pin. For detailed information on GPIO interrupt enable registers, see Section 12.3.5.

12.2.7.3 Configuring GPIO Interrupt Edge Triggering

Each GPIO interrupt source can be configured to generate an interrupt on the rising or the falling edge. The edge detection is synchronized to the GPIO peripheral module clock. This is controlled in the GPIO interrupt edge trigger enable registers (IOINTEDG1 and IOINTEDG2). Setting a "1" to this register will use Rising edge to trigger the interrupt and "0" sets it to Falling Edge triggered if interrupts are enabled for this GPIO. For detailed information on the GPIO interrupt edge trigger enable registers, see Section 12.3.4.

12.2.7.4 GPIO Interrupt Status

When an interrupt occurs on an enabled GPIO pin, the GPIO interrupt flag registers (IOINTFLG1 and IOINTFLG2) latch the corresponding bit to a "1". The interrupt signal to the CPU will be kept low until all flag bits in the IOINTFLG1 and IOINTFLG2 registers are cleared. To clear the flag you must write a "1" to the corresponding bit.

The status of the GPIO interrupt events can be monitored by reading the IOINTFLG1 and IOINTFLG2 registers. If a GPIO interrupt event has occurred the corresponding bit will be a set to a “1”. In the case of a non-interrupt the corresponding bit will be “0”.

12.2.7.5 Interrupt Multiplexing

No GPIO interrupts are multiplexed with other interrupt functions on the device.

12.3 Registers

Table 12-2 lists the memory-mapped registers for the general-purpose input/output (GPIO).

Table 12-2. Memory-mapped Registers for the General-purpose Input/Output (GPIO)

CPU Word Address	Acronym	Register Description	Section
1C06h	IODIR1	GPIO Direction Register 1	Section 12.3.1
1C07h	IODIR2	GPIO Direction Register 2	Section 12.3.1
1C08h	IOINDATA1	GPIO Data In Register 1	Section 12.3.2
1C09h	IOINDATA2	GPIO Data In Register 2	Section 12.3.2
1C0Ah	IODATAOUT1	GPIO Data Out Register 1	Section 12.3.3
1C0Bh	IODATAOUT2	GPIO Data Out Register 2	Section 12.3.3
1C0Ch	IOINTEDG1	GPIO Interrupt Edge Trigger Enable Register 1	Section 12.3.4
1C0Dh	IOINTEDG2	GPIO Interrupt Edge Trigger Enable Register 2	Section 12.3.4
1C0Eh	IOINTEN1	GPIO Interrupt Enable Register 1	Section 12.3.5
1C0Fh	IOINTEN2	GPIO Interrupt Enable Register 2	Section 12.3.5
1C10h	IOINTFLG1	GPIO Interrupt Flag Register 1	Section 12.3.6
1C11h	IOINTFLG2	GPIO Interrupt Flag Register 2	Section 12.3.6

12.3.1 GPIO Direction Registers (IODIR1 and IODIR2)

The device includes two registers for controlling whether the GPIO is set as a general-purpose Input, or Output. Use the GPIO direction register (IODIR1 and IODIR2) to set the GPIO pin as Input or Output. Each of these registers control 16 of the 32 GPIOs. IODIR1 is used for GPIO[15:0] with bit 0 corresponding to GPIO 0 through bit 15 corresponding to GPIO 15. IODIR2 is used for GPIO[31:16] with bit 0 corresponding to GPIO 16 through bit 15 corresponding to GPIO 31. Writing a "1" to these bits configures the pin as an OUTPUT and writing a "0" configures the pin as an INPUT.

Note: This device supports up to 26 signals: GPIO[31:27] and GPIO[20:0].

The GPIO Direction Registers (IODIR1 and IODIR2) is shown in Figure 12-1 and Figure 12-2 and described in Table 12-3.

Figure 12-1. GPIO Direction Register 1 (IODIR1)

15	0
DIR	
RW+0	

LEGEND: R/W = Read/Write; n = value at reset

Figure 12-2. GPIO Direction Register 2 (IODIR2)

15	0
DIR	
RW+0	

LEGEND: R/W = Read/Write; n = value at reset

Table 12-3. GPIO Direction Register (IODIR1 and IODIR2) Bit Field Description

Bit	Field	Value	Description
15-0	DIR	0-FFFFh	Data direction bits that configure the general-purpose IO pins as either inputs or outputs. Bit 0 of IODIR1 corresponds to GPIO 0 and Bit 0 of IODIR2 corresponds to GPIO 16.
		0	Configure corresponding pin as an INPUT
		1	Configure corresponding pin as an OUTPUT.

12.3.2 GPIO Data In Registers (IOINDATA1 and IOINDATA2)

The device includes two registers for reading in the GPIO values when they are configured as Inputs. Use the GPIO data in registers (IOINDATA1 and IOINDATA2) to read the state of the corresponding GPIO pin. Each of these registers control 16 of the 32 GPIOs. IOINDATA1 is used for GPIO[15:0] with bit 0 corresponding to GPIO 0 through bit 15 corresponding to GPIO15. IOINDATA2 is used for GPIO[31:16] with bit 0 corresponding to GPIO 16 through bit 15 corresponding to GPIO 31.

Note: This device supports up to 26 signals: GPIO[31:27] and GPIO[20:0].

The GPIO Data In Registers (IOINDATA1 and IOINDATA2) is shown in Figure 12-3 and Figure 12-4 and described in Table 12-4.

Figure 12-3. GPIO Data In Register 1 (IOINDATA1)

15	0
IN	
RW+0	

LEGEND: R/W = Read/Write; n = value at reset

Figure 12-4. GPIO Data In Register 2 (IOINDATA2)

15	0
IN	
RW+0	

LEGEND: R/W = Read/Write; n = value at reset

Table 12-4. GPIO Data In Register (IOINDATA1 and IOINDATA2) Bit Field Description

Bit	Field	Value	Description
15-0	IN	0-FFFFh	Data bits that are used to monitor the level of the GPIO pins configured as general-purpose input pins. If DIR = 0, then: • 0 = corresponding I/O pin is read as a LOW. • 1 = corresponding I/O pin is read as a HIGH. If DIR = 1, then: • X = reflects value of output pin.

12.3.3 GPIO Data Out Registers (IOOUTDATA1 and IOOUTDATA2)

The device includes two registers for writing to the GPIO pins when they are configured as outputs. Use the GPIO data out registers (IOOUTDATA1 and IOOUTDATA2) to change the state of the corresponding GPIO pin. Each of these registers control 16 of the 32 GPIOs. IOOUTDATA1 is used for GPIO[15:0] with bit 0 corresponding to GPIO 0 through bit 15 corresponding to GPIO15. IOOUTDATA2 is used for GPIO[31:16] with bit 0 corresponding to GPIO 16 through bit 15 corresponding to GPIO 31.

Note: This device supports up to 26 signals: GPIO[31:27] and GPIO[20:0].

The GPIO Data Out Registers (IOOUTDATA1 and IOOUTDATA2) is shown in Figure 12-5 and Figure 12-6 and described in Table 12-5.

Figure 12-5. GPIO Data Out Register 1 (IODATAOUT1)

15	0
OUT	
RW+0	

LEGEND: R/W = Read/Write; n = value at reset

Figure 12-6. GPIO Data Out Register 2 (IODATAOUT2)

15	0
OUT	
RW+0	

LEGEND: R/W = Read/Write; n = value at reset

Table 12-5. GPIO Data Out Register (IOOUTDATA1 and IOOUTDATA2) Bit Field Description

Bit	Field	Value	Description
15-0	OUT	0-FFFFh	Data bits that are used to control the level of the GPIO pins configured as general-purpose output pins. If DIR = 0, then: • X = value stored on register but not reflected on the output pin. If DIR = 1, then: • 0 = set corresponding I/O pin to LOW. • 1 = set corresponding I/O pin to HIGH.

12.3.4 GPIO Interrupt Edge Trigger Registers (IOINTEDG1 and IOINTEDG2)

The device has two registers for configuring interrupts to trigger on the rising or falling edge of the input signal to the GPIO pins if they are configured as inputs with Interrupt enabled for the chosen GPIO. Use the GPIO interrupt edge trigger registers (IOINTEDG1 and IOINTEDG2) to enable rising or falling edge trigger for the corresponding GPIO pin. Each of these registers control 16 of the 32 GPIOs. IOINTEDG1 is used for GPIO[15:0] with bit 0 corresponding to GPIO 0 through bit 15 corresponding to GPIO15. IOINTEDG2 is used for GPIO[31:16] with bit 0 corresponding to GPIO 16 through bit 15 corresponding to GPIO 31.

Note: This device supports up to 26 signals: GPIO[31:27] and GPIO[20:0].

The GPIO Interrupt Edge Trigger Registers (IOINTEDG1 and IOINTEDG2) is shown in Figure 12-7 and Figure 12-8 and described in Table 12-6.

Figure 12-7. GPIO Interrupt Edge Trigger Enable Register 1 (IOINTEDG1)

15	0
INTEDG	
RW+0	

LEGEND: R/W = Read/Write; n = value at reset

Figure 12-8. GPIO Interrupt Edge Trigger Enable Register 2 (IOINTEDG2)

15	0
INTEDG	
RW+0	

LEGEND: R/W = Read/Write; n = value at reset

Table 12-6. GPIO Interrupt Edge Trigger Enable Register (IOINTEDG1 and IOINTEDG2) Bit Field Description

Bit	Field	Value	Description
15-0	INTEDG	0-FFFFh	Configure the GPIO pin for rising or falling edge.
		0	Corresponding I/O pin has rising edge triggered interrupt capability.
		1	Corresponding I/O pin has falling edge triggered interrupt capability.

12.3.5 GPIO Interrupt Enable Registers (IOINTEN1 and IOINTEN2)

The device has two registers for enabling Interrupts on the GPIO pins if they are configured as Inputs. Use the GPIO interrupt enable registers (IOINTEN1 and IOINTEN2) to enable the interrupt of the corresponding GPIO pin. Each of these registers control 16 of the 32 GPIOs. IOINTEN1 is used for GPIO[15:0] with bit 0 corresponding to GPIO 0 through bit 15 corresponding to GPIO15. IOINTEN2 is used for GPIO[31:16] with bit 0 corresponding to GPIO 16 through bit 15 corresponding to GPIO 31.

Note: This device supports up to 26 signals: GPIO[31:27] and GPIO[20:0].

The GPIO Interrupt Enable Registers (IOINTEN1 and IOINTEN2) is shown in Figure 12-9 and Figure 12-10 and described in Table 12-7.

Figure 12-9. GPIO Interrupt Enable Register 1 (IOINTEN1)

<table>
<tr><td>15</td><td>0</td></tr>
<tr><td colspan="2">INTEN</td></tr>
<tr><td colspan="2">RW+0</td></tr>
</table>

LEGEND: R/W = Read/Write; n = value at reset

Figure 12-10. GPIO Interrupt Enable Register 2 (IOINTEN2)

<table>
<tr><td>15</td><td>0</td></tr>
<tr><td colspan="2">INTEN</td></tr>
<tr><td colspan="2">RW+0</td></tr>
</table>

LEGEND: R/W = Read/Write; n = value at reset

Table 12-7. GPIO Interrupt Enable Register (IOINTEN1 and IOINTEN2) Bit Field Description

Bit	Field	Value	Description
15-0	INTEN	0-FFFFh	Enable or disable interrupt capability of the GPIO:
		0	Corresponding I/O pin has NO interrupt capability.
		1	Corresponding I/O pin is configured as an external interrupt.

12.3.6 GPIO Interrupt Flag Registers (IOINTFLG1 and IOINTFLG2)

The device has two registers that latch an interrupt that occurred with the corresponding GPIO pin. Use the GPIO interrupt flag registers (IOINTFLG1 and IOINTFLG2) to determine which GPIO pin triggered the interrupt. Also, these registers are used to clear the interrupt sent to the CPU. Each of these registers control 16 of the 32 GPIOs. IOINTFLG1 is used for GPIO[15:0] with bit 0 corresponding to GPIO 0 through bit 15 corresponding to GPIO15. IOINTFLG2 is used for GPIO[31:16] with bit 0 corresponding to GPIO 16 through bit 15 corresponding to GPIO 31.

Note: This device supports up to 26 signals: GPIO[31:27] and GPIO[20:0].

The GPIO Interrupt Flag Registers (IOINTFLG1 and IOINTFLG2) is shown in Figure 12-11 and Figure 12-12 and described in Table 12-8.

Figure 12-11. GPIO Interrupt Flag Register 1 (IOINTFLG1)

15 ... 0
INTFLG
RW+0

LEGEND: R/W = Read/Write; n = value at reset

Figure 12-12. GPIO Interrupt Flag Register 2 (IOINTFLG2)

15 ... 0
INTFLG
RW+0

LEGEND: R/W = Read/Write; n = value at reset

Table 12-8. GPIO Interrupt Flag Enable Register (IOINTFLG1 and IOINTFLG2) Bit Field Description

Bit	Field	Value	Description
15-0	INTFLG	0-FFFFh	Register that latches if an interrupt occurred in the corresponding I/O pin, if interrupts were enabled. The flag is cleared by writing a “1” or setting the corresponding bit. The interrupt signal to the CPU will be kept low until all flag bits in this register are cleared.
		0	Corresponding I/O interrupt has not occurred.
		1	Corresponding I/O interrupt occurred. Write of “1” resets the flag.

Universal Serial Bus (USB) Controller

This chapter describes the features and operations of the universal serial bus (USB) controller.

Topic **Page**

13.1 Introduction

This document describes the universal serial bus (USB) controller. The controller complies with the USB 2.0 standard high-speed and full-speed functions. In addition, the four test modes for high-speed operation described in the USB 2.0 specification are supported. It also allows options that allow the USB controller to be forced into full-speed mode and high-speed mode that may be used for debug purposes.

13.1.1 Purpose of the Peripheral

The USB controller provides a low-cost connectivity solution for consumer portable devices by providing a mechanism for data transfer between USB devices up to 480 Mbps. With the USB controller, you can use the DSP to create a high-speed USB slave device that is complaint with the Universal Serial Bus Specification version 2.0.

13.1.2 Features

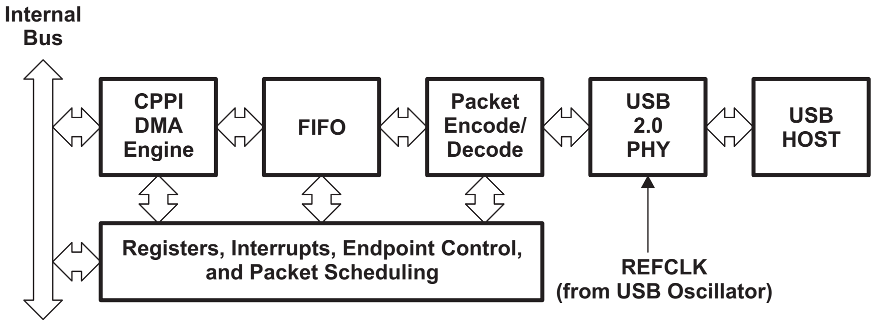
The USB has the following features:

- Operating as a peripheral, it complies with the USB 2.0 standard for high-speed (480 Mbps) and full-speed (12 Mbps) operation with a host
- Supports 4 simultaneous RX and TX endpoints, in addition to control endpoint, more devices can be supported by dynamically switching endpoints states
- Each endpoint (other than endpoint 0) can support all transfer types (control, bulk, interrupt, and isochronous)
- Includes a 4K endpoint FIFO RAM, and supports programmable FIFO sizes
- Includes a DMA controller that supports 4 TX and 4 RX DMA channels
- Includes four types of Communications Port Programming Interface (CPPI) 4.1 DMA compliant transfer modes, Transparent, Generic RNDIS, RNDIS, and Linux CDC mode of DMA for accelerating RNDIS type protocols using short packet termination over USB
- DMA supports single data transfer size up to 4Mbytes

13.1.3 Functional Block Diagram

The USB functional block diagram is shown in Figure 13-1.

Figure 13-1. Functional Block Diagram


Internal Bus
CPPI DMA Engine
FIFO
Packet Encode/ Decode
USB 2.0 PHY
USB HOST
Registers, Interrupts, Endpoint Control, and Packet Scheduling
REFCLK (from USB Oscillator)

13.1.4 Industry Standard(s) Compliance Statement

This device conforms to USB 2.0 Specification.

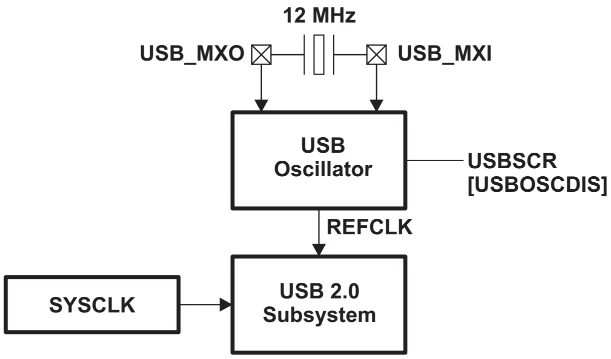
13.2 Architecture

13.2.1 Clock Control

Figure 13-2 shows the clock connections for the USB2.0 module. Note that there is a built-in oscillator that generates a 12 MHz reference clock for the internal PLL of the USB 2.0 subsystem. The USB2.0 subsystem peripheral bus clock is sourced from the system clock (SYSCLK).

NOTE: The device system clock (SYSCLK) must be at least 30 MHz for proper USB operation.

Figure 13-2. USB Clocking Diagram



13.2.2 Signal Descriptions

The USB controller provides the I/O signals listed in Table 13-1.

Table 13-1. USB Terminal Functions

Name	I/O [(1)]	Description
USB_DP	A I/O/Z	USB D+ (differential signal pair)
USB_DM	A I/O/Z	USB D- (differential signal pair)
USB_VBUS	A I	Five volt input that signifies that VBUS is connected.
USB_REXT	A I/O/Z	External resistor connect.
USB_MXI	I	12 MHz crystal oscillator input.
USB_MXO	O	12 MHz crystal oscillator output.
USB_LDOO	PWR	USB module LDO output. This output is regulated to 1.3V.
USB_LDOI	PWR	USB module LDO input. This input handles a voltage range of 1.8V to 3.6V.
VSS_USBOSC	PWR	3.3V USB oscillator power supply.
VDD_USBPLL	PWR	3.3V USB PLL power supply.
VDDA_USBXCVR	PWR	3.3V USB transceiver power supply.
VDDA_USB	PWR	1.3V USB analog power supply.
VDD_USB	PWR	1.3V USB PLL and oscillator digital power supply.
VSS_USBOSC	GND	USB oscillator ground.
VSS_USBPLL	GND	USB PLL ground.
VSSA_USBXCVR	GND	USB transceiver ground.
VSS_USBXCVR	GND	USB ground for reference circuits.
VSSA_USB	GND	USB analog ground.
VSS_USB	GND	USB PLL and oscillator digital ground.

[(1)] I = Input, O = Output, Z = High impedance, GND = Ground, A = Analog signal, PWR = Power supply pin.

13.2.3 Memory Map

The USB controller can access only internal single-access RAM (SARAM). It cannot access dual-access RAM (DARAM). The starting address for SARAM is different from the point-of-view of the CPU and USB controller. The memory map, as seen by the USB controller and the CPU, is shown in Table 13-2.

Table 13-2. USB Controller Memory Map

USB Start *Byte* Address	CPU Start *Word* Address	CPU Memory Map	USB Controller Memory Map
0001 0000h[(1)]	00 0000h[(1)]	DARAM	Reserved
0009 0000h	00 8000h	SARAM	SARAM

[(1)] CPU word addresses 00 0000h - 00 005Fh (which correspond to byte addresses 00 0000h - 00 00BFh) are reserved for the memory-mapped registers (MMRs) of the DSP CPU.

13.2.4 USB_DP/USB_DM Polarity Inversion

The polarity of the USB data pins (USB_DP and USB_DM) can be inverted through the USBDATPOL bit of the USB system control register (USBSCR). Since USB_DP is equal to the inverse of USB_DM (they form a differential pair), inverting these pins allows you to effectively swap their function. This allows flexibility in board design by allowing different USB connector configurations. In particular, this allows for mounting the connector on either side of the board and for arranging the data pins so they do not physically cross each other.

13.2.5 Indexed and Non-Indexed Registers

The USB controller provides two mechanisms of accessing the endpoint control and status registers:

- Indexed Endpoint Control/Status Registers: These registers are located at I/O address 8410h to 841Fh. The endpoint is selected by programming the INDEX register of the controller.
- Non-indexed Endpoint Control/Status Registers: These registers are located at I/O address 8500h to 854Fh. Registers at address 8500h to 850Fh map to Endpoint 0; at address 8510h to 851Fh map to Endpoint 1, and so on.

For detailed information about the USB controller registers, see Section 13.3.

13.2.6 USB PHY Initialization

The general procedure for USB PHY initialization consists of enabling the USB on-chip oscillator, configuring PHY parameters, and finally resetting the PHY. The detailed USB PHY initialization sequence is as follows:

1. The bits USBOSCBIASDIS and USBOSCDIS in the USB system control register (USBSCR) should be cleared to 0 to enable the on-chip USB oscillatory if not enabled already.
2. Wait cycles for the on-chip oscillator to stabilize. Refer to the device-specific data manual for oscillator stabilization time.
3. To configure the PHY for normal operation, the bits USBPWDN, USBSESSEND, and USBPLLEN in USBSCR should be cleared to 0, the USBVBUSDET bit should be set to 1, and the USBDATPOL bit should be set according to the system requirements (set to 1 for normal operation).
4. Enable the USB clock by clearing USBCG to 0 in the peripheral clock gating configuration register 2 (PCGCR2).
5. Set the USBCLKSTPREQ bit.
6. Set COUNT = 20h in the peripheral software reset counter register (PSRCR).
7. Reset the USB controller by setting USB_RST to 1 in the peripheral reset control register (PRCR). This bit will self-clear once the reset has been completed.

For more information on the PCGCR2, CLKSTOP, PSRCR, and PRCR, see Section 1.1, System Control.

During the normal operation, the USB PHY PLL should run at 60 MHz. This clock can be probed on the device. One can monitor different clocks of the device by probing Pin-A7 (CLKOUT) for debugging purposes. Different clocks can be routed to this pin by setting both register CCSSR (0x1C24 at IO space) and CPU register ST3_55.

To monitor the USB PHY PLL, the setting of register CCSSR should be as follows:

- When CCSSR = 0x000F; USB PHY clock (60 MHz) is routed to A7

The setting of bit-2 (CLKOFF) of register ST3_55:

- When CLKOFF = 0, the CLKOUT pin is enabled
- When CLKOFF = 1, the CLKOUT pin is disabled

After these settings, the 60 MHz clock can be probed on CLKOUT pin.

13.2.6.1 USB System Control Register (USBSCR)

The USB system control register is used to disable the USB on-chip oscillator and to power-down the USB.

The USB system control register (USBSCR) is shown in Figure 13-3 and described in Table 13-3.

Figure 13-3. USB System Control Register (USBSCR) [1C32h]

15	14	13	12	11			8
USBPWDN	USBSESSEND	USBVBUSDET	USBPLLEN	Reserved			
R/W-1	R/W-0	R/W-1	R/W-0	R-0			

7	6	5	4	3	2	1	0
Reserved	USBDATPOL	Reserved		USBOSCBIASDIS	USBOSCDIS	BYTEMODE	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-3. USB System Control Register (USBSCR) Field Descriptions

Bit	Field	Value	Description
15	USBPWDN		USB module power.
		0	USB module is powered.
		1	USB module is powered-down.
14	USBSESSEND		USB VBUS session end comparator enable.
		0	USB VBUS session end comparator is disabled.
		1	USB VBUS session end comparator is enabled.
13	USBVBUSDET		USB VBUS detect enable.
		0	USB VBUS detect comparator is disabled.
		1	USB VBUS detect comparator is enabled.
12	USBPLLEN		USB PLL enable.
		0	Normal USB operation.
		1	Override USB suspend end behavior and force release of PLL from suspend state.
11-7	Reserved	0	Reserved. Always write 0 to these bits.
6	USBDATPOL		USB data polarity bit.
		0	Reverse polarity on DP and DM signals.
		1	Normal polarity.
5-4	Reserved	0	Reserved.
3	USBOSCBIASDIS		USB internal oscillator bias resistor disable.
		0	Internal oscillator bias resistor enabled (normal operating mode).
		1	Internal oscillator bias resistor disabled.
2	USBOSCDIS		USB oscillator disable bit.
		0	USB internal oscillator enable.
		1	USB internal oscillator disabled.
1-0	BYTEMODE		USB byte mode select bits.
		0	Word accesses by the CPU are allowed.
		1h	Byte accesses by the CPU are allowed (high byte is selected).
		2h	Byte accesses by the CPU are allowed (low byte is selected).
		3h	Reserved.

13.2.7 Dynamic FIFO Sizing

The USB controller supports a total of 4K RAM to dynamically allocate FIFO to all endpoints. The allocation of FIFO space to the different endpoints requires the specification for each Tx and Rx endpoint of:

- The start address of the FIFO within the RAM block
- The maximum size of packet to be supported
- Whether double-buffering is required.

These details are specified through four registers, which are added to the indexed area of the memory map. That is, the registers for the desired endpoint are accessed after programming the INDEX register with the desired endpoint value. Section 13.3.48, Section 13.3.49, Section 13.3.50, and Section 13.3.52 provide details of these registers.

NOTE: The option of setting FIFO sizes dynamically only applies to Endpoints 1 to 4. Endpoint 0 FIFO has a fixed size (64 bytes) and a fixed location (start address 0).

It is the responsibility of the firmware to ensure that all the Tx and Rx endpoints that are active in the current USB configuration have a block of RAM assigned exclusively to that endpoint that is at least as large as the maximum packet size set for that endpoint.

13.2.8 USB Controller Peripheral Mode Operation

The USB controller can be used as a high-speed or a full-speed USB peripheral device attached to a conventional USB host (such as a PC).

The USB2.0 controller will transition to session when it sees power (must be greater or equal to 4.01V) on the USB0_VBUS pin, assuming that the firmware has set the SOFTCONN bit in the POWER register and has enabled the data lines and there is an external host sourcing power on the USB0_VBUS line. The USB 2.0 controller will then set the SESSION bit upon detecting the power on the USB0_VBUS line and it will connect its 1.5Kohm pull-up resistor so it signifies to the external host out it is a Full-Speed device. Note that even when operating as a High-Speed; it has to first come up as Full-Speed. The USB2.0 controller will then wait for a reset signal from the host.

13.2.8.1 USB Interrupts

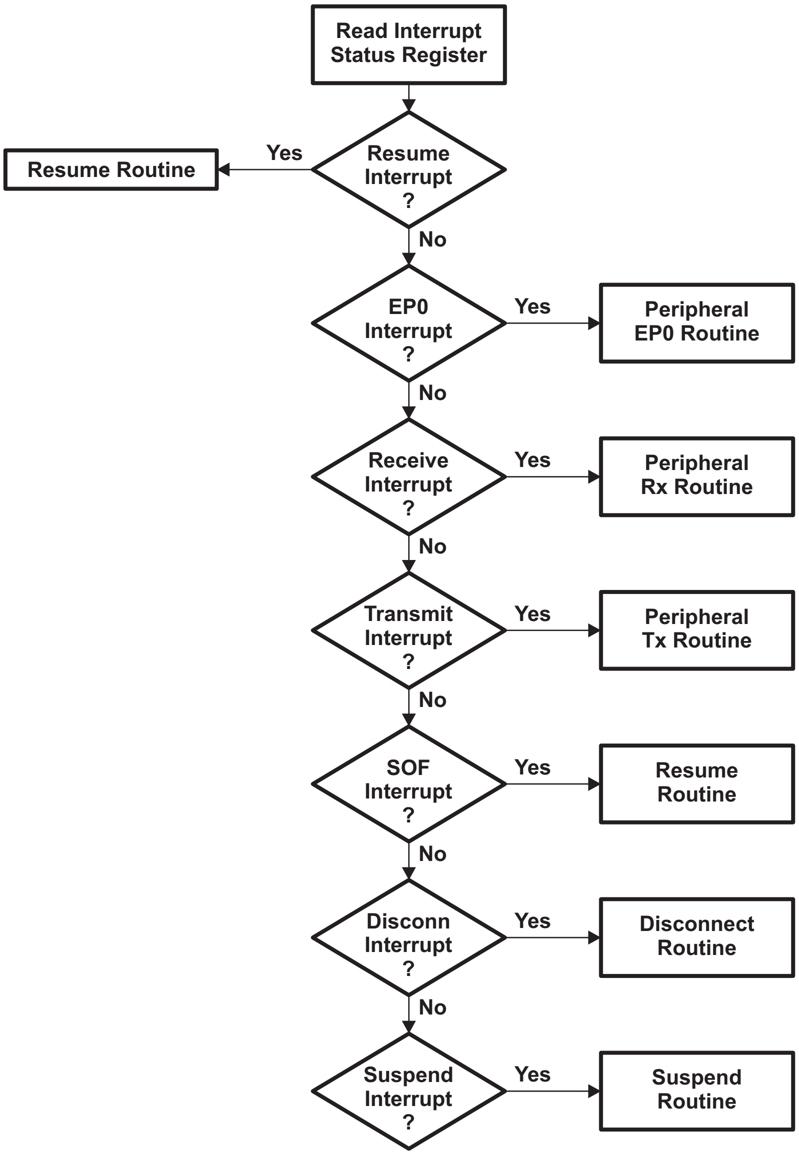
The USB controller interrupts the CPU on completion of the data transfer on any of the endpoints or on detecting reset, resume, suspend, connect, disconnect, or start-of-frame (SOF) on the bus.

When the CPU is interrupted with a USB interrupt, it needs to read the interrupt status register to determine the endpoints that have caused the interrupt and jump to the appropriate routine. If multiple endpoints have caused the interrupt, endpoint 0 should be serviced first followed by the other endpoints. The suspend interrupt should be serviced last.

The flowchart in Figure 13-4 describes the interrupt service routine for the USB module.

The following sections describe the programming of USB controller in peripheral mode.

Figure 13-4. Interrupt Service Routine Flow Chart



13.2.8.2 Connect, Suspend Mode, and Reset Signaling

The following sections describe the operation of the USB controller during connect, suspend mode, and USB reset.

13.2.8.2.1 Soft Connect

After a reset, the SOFTCONN bit in the POWER register is cleared to 0. The controller will therefore appear disconnected until the software has set the SOFTCONN bit to 1. The application software can then choose when to set the PHY into its normal mode. Systems with a lengthy initialization procedure may use this to ensure that initialization is complete and the system is ready to perform enumeration before connecting to the USB.

Once the SOFTCONN bit of the POWER register has been set, the software can also simulate a disconnect by clearing this bit to 0.

13.2.8.2.2 Suspend Mode

The controller monitors activity on the bus and when no activity has occurred for 3 ms, it goes into Suspend mode. If the Suspend interrupt has been enabled, an interrupt will be generated.

At this point, the controller can be left active (and hence able to detect when Resume signaling occurs on the USB), or the application may arrange to disable the controller by stopping its clock. However, the controller will not then be able to detect Resume signaling on the USB. As a result, some external hardware will be needed to detect Resume signaling (by monitoring the DM and DP signals) so that the clock to the controller can be restarted.

When Resume signaling occurs on the bus, first the clock to the controller must be restarted if necessary. Then the controller will automatically exit Suspend mode. If the Resume interrupt is enabled, an interrupt will be generated.

If the software wants to initiate a remote wake-up while the controller is in Suspend mode, it should write to the POWER register to set the RESUME bit to 1. The software should then leave this bit set for approximately 10 ms (minimum of 2 ms, a maximum of 15 ms) before resetting it to 0.

NOTE: No resume interrupt will be generated when the software initiates a remote wake-up.

13.2.8.2.3 Reset Signaling

If the HSENA bit in the POWER register was set, the controller also tries to negotiate for high-speed operation.

Whether high-speed operation is selected is indicated by the HSMODE bit of the POWER register.

When the application software receives a reset interrupt, it should close any open pipes and wait for bus enumeration to begin.

13.2.8.3 Control Transactions

Endpoint 0 is the main control endpoint of the core. The software is required to handle all the standard device requests that may be sent or received via endpoint 0. These are described in Universal Serial Bus Specification, Revision 2.0, Chapter 9. The protocol for these device requests involves different numbers and types of transactions per transfer. To accommodate this, the software needs to take a state machine approach to command decoding and handling.

The Standard Device Requests received by a USB peripheral device can be divided into three categories: Zero Data Requests (in which all the information is included in the command), Write Requests (in which the command will be followed by additional data), and Read Requests (in which the device is required to send data back to the host).

This section looks at the sequence of actions that the software must perform to process these different types of device request.

NOTE: The Setup packet associated with any standard device request should include an 8-byte command. Any setup packet containing a command field of anything other than 8 bytes will be automatically rejected by the controller.

13.2.8.3.1 Zero Data Requests

Zero data requests have all their information included in the 8-byte command and require no additional data to be transferred. Examples of Zero Data standard device requests are:

- SET_FEATURE
- CLEAR_FEATURE
- SET_ADDRESS
- SET_CONFIGURATION
- SET_INTERFACE

The sequence of events will begin, as with all requests, when the software receives an endpoint 0 interrupt. The RXPKTRDY bit of the PERI_CSR0 register will also have been set. The 8-byte command should then be read from the endpoint 0 FIFO, decoded, and the appropriate action taken.

For example, if the command is SET_ADDRESS, the 7-bit address value contained in the command should be written to the FADDR register. The PERI_CSR0 register should then be written to set the SERV_RXPKTRDY bit (indicating that the command has been read from the FIFO) and to set the DATAEND bit (indicating that no further data is expected for this request). The interval between setting the SERV_RXPKTRDY bit and setting the DATAEND bit should be very small to avoid getting a SetupEnd error condition.

When the host moves to the status stage of the request, a second endpoint 0 interrupt will be generated to indicate that the request has completed. No further action is required from the software. The second interrupt is just a confirmation that the request completed successfully. For SET_ADDRESS command, the address should be set in the FADDR register only after the status stage interrupt is received.

If the command is an unrecognized command, or for some other reason cannot be executed, then when it has been decoded, the PERI_CSR0 register should be written to set the SERV_RXPKTRDY bit and to set the SENDSTALL bit. When the host moves to the status stage of the request, the controller will send a STALL to tell the host that the request was not executed. A second endpoint 0 interrupt will be generated and the SENTSTALL bit in the PERI_CSR0 register will be set.

If the host sends more data after the DATAEND bit has been set, then the controller will send a STALL. An endpoint 0 interrupt will be generated and the SENTSTALL bit in the PERI_CSR0 register will be set.

NOTE: DMA is not supported for endpoint 0, so the command should be read by accessing the endpoint 0 FIFO register.

13.2.8.3.2 Write Requests

Write requests involve an additional packet (or packets) of data being sent from the host after the 8-byte command. An example of a Write standard device request is: SET_DESCRIPTOR.

The sequence of events will begin, as with all requests, when the software receives an endpoint 0 interrupt. The RXPKTRDY bit of PERI_CSR0 will also have been set. The 8-byte command should then be read from the Endpoint 0 FIFO and decoded.

As with a zero data request, the PERI_CSR0 register should then be written to set the SERV_RXPKTRDY bit (indicating that the command has been read from the FIFO) but in this case the DATAEND bit should not be set (indicating that more data is expected).

When a second endpoint 0 interrupt is received, the PERI_CSR0 register should be read to check the endpoint status. The RXPKTRDY bit in the PERI_CSR0 register should be set to indicate that a data packet has been received. The COUNT0 register should then be read to determine the size of this data packet. The data packet can then be read from the endpoint 0 FIFO.

If the length of the data associated with the request (indicated by the wLength field in the command) is greater than the maximum packet size for endpoint 0, further data packets will be sent. In this case, PERI_CSR0 should be written to set the SERV_RXPKTRDY bit, but the DATAEND bit should not be set.

When all the expected data packets have been received, the PERI_CSR0 register should be written to set the SERV_RXPKTRDY bit and to set the DATAEND bit (indicating that no more data is expected).

When the host moves to the status stage of the request, another endpoint 0 interrupt will be generated to indicate that the request has completed. No further action is required from the software, the interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when it has been decoded, the PERI_CSR0 register should be written to set the SERV_RXPKTRDY bit and to set the SENDSTALL bit. When the host sends more data, the controller will send a STALL to tell the host that the request was not executed. An endpoint 0 interrupt will be generated and the SENTSTALL bit in the PERI_CSR0 register will be set.

If the host sends more data after the DATAEND bit has been set, then the controller will send a STALL. An endpoint 0 interrupt will be generated and the SENTSTALL bit will be set.

13.2.8.3.3 Read Requests

Read requests have a packet (or packets) of data sent from the function to the host after the 8-byte command. Examples of Read Standard Device Requests are:

- GET_CONFIGURATION
- GET_INTERFACE
- GET_DESCRIPTOR
- GET_STATUS
- SYNCH_FRAME

The sequence of events will begin, as with all requests, when the software receives an endpoint 0 interrupt. The RXPKTRDY bit in the PERI_CSR0 register will also have been set. Next, the 8-byte command should be read from the endpoint 0 FIFO and decoded. The PERI_CSR0 register should then be written to set the SERV_RXPKTRDY bit (indicating that the command has read from the FIFO).

The data to be sent to the host should be written to the endpoint 0 FIFO. If the data to be sent is greater than the maximum packet size for endpoint 0, only the maximum packet size should be written to the FIFO. The PERI_CSR0 register should then be written to set the TXPKTRDY bit (indicating that there is a packet in the FIFO to be sent). When the packet has been sent to the host, another endpoint 0 interrupt will be generated and the next data packet can be written to the FIFO.

When the last data packet has been written to the FIFO, the PERI_CSR0 register should be written to set the TXPKTRDY bit and to set the DATAEND bit (indicating that there is no more data after this packet).

When the host moves to the status stage of the request, another endpoint 0 interrupt will be generated to indicate that the request has completed. No further action is required from the software: the interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when it has been decoded, the PERI_CSR0 register should be written to set the SERV_RXPKTRDY bit and to set the SENDSTALL bit. When the host requests data, the controller will send a STALL to tell the host that the request was not executed. An endpoint 0 interrupt will be generated and the SENTSTALL bit in the PERI_CSR0 register will be set.

If the host requests more data after the DATAEND bit has been set, then the controller will send a STALL. An endpoint 0 interrupt will be generated and the SENTSTALL bit will be set.

13.2.8.3.4 Endpoint 0 States

The endpoint 0 control needs three modes – IDLE, TX, and RX – corresponding to the different phases of the control transfer and the states endpoint 0 enters for the different phases of the transfer (described in later sections).

The default mode on power-up or reset should be IDLE. The RXPKTRDY bit in the PERI_CSR0 register becoming set when endpoint 0 is in IDLE state indicates a new device request. Once the device request is unloaded from the FIFO, the controller decodes the descriptor to find whether there is a data phase and, if so, the direction of the data phase of the control transfer (in order to set the FIFO direction). See Figure 13-5.

Depending on the direction of the data phase, endpoint 0 goes into either TX state or RX state. If there is no Data phase, endpoint 0 remains in IDLE state to accept the next device request.

The actions that the CPU needs to take at the different phases of the possible transfers (for example, loading the FIFO, setting TXPKTRDY) are indicated in Figure 13-6 .

NOTE: The controller changes the FIFO direction, depending on the direction of the data phase independently of the CPU.

Figure 13-5. CPU Actions at Transfer Phases

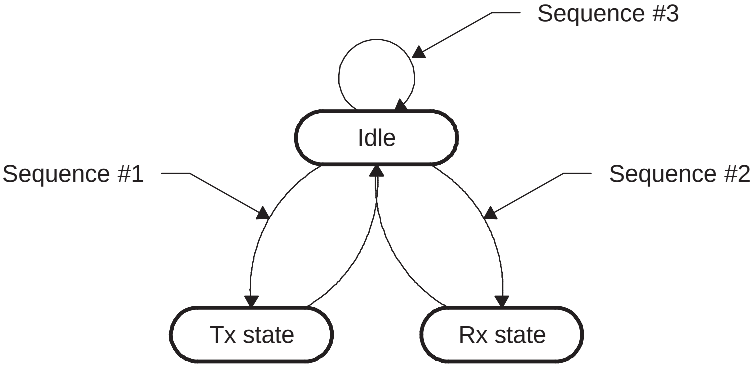
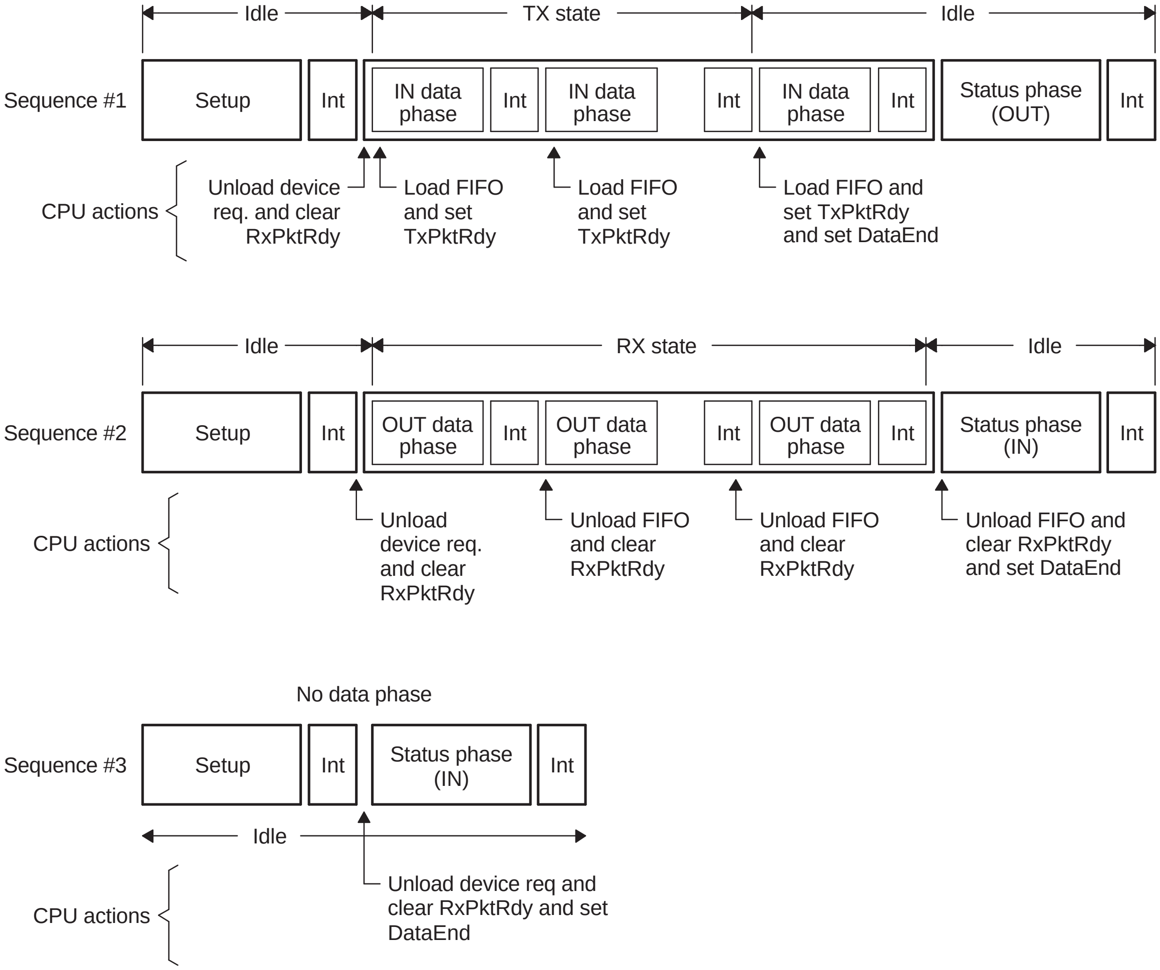


Figure 13-6. Sequence of Transfer


Idle
TX state
Idle
Sequence #1
Setup
Int
IN data phase
Int
IN data phase
Int
IN data phase
Int
Status phase (OUT)
Int
CPU actions
Unload device req. and clear RxPktRdy
Load FIFO and set TxPktRdy
Load FIFO and set TxPktRdy
Load FIFO and set TxPktRdy and set DataEnd
Idle
RX state
Idle
Sequence #2
Setup
Int
OUT data phase
Int
OUT data phase
Int
OUT data phase
Int
Status phase (IN)
Int
CPU actions
Unload device req. and clear RxPktRdy
Unload FIFO and clear RxPktRdy
Unload FIFO and clear RxPktRdy
Unload FIFO and clear RxPktRdy and set DataEnd
No data phase
Sequence #3
Setup
Int
Status phase (IN)
Int
Idle
CPU actions
Unload device req and clear RxPktRdy and set DataEnd

13.2.8.3.5 Endpoint 0 Service Routine

An Endpoint 0 interrupt is generated when:

- The controller sets the RXPKTRDY bit in the PERI_CSR0 register after a valid token has been received and data has been written to the FIFO.
- The controller clears the TXPKTRDY bit of PERI_CSR0 after the packet of data in the FIFO has been successfully transmitted to the host.
- The controller sets the SENTSTALL bit of PERI_CSR0 after a control transaction is ended due to a protocol violation.
- The controller sets the SETUPEND bit of PERI_CSR0 because a control transfer has ended before DATAEND is set.

Whenever the endpoint 0 service routine is entered, the software must first check to see if the current control transfer has been ended due to either a STALL condition or a premature end of control transfer. If the control transfer ends due to a STALL condition, the SENTSTALL bit would be set. If the control transfer ends due to a premature end of control transfer, the SETUPEND bit would be set. In either case, the software should abort processing the current control transfer and set the state to IDLE.

Once the software has determined that the interrupt was not generated by an illegal bus state, the next action taken depends on the endpoint state. Figure 13-7 shows the flow of this process.

If endpoint 0 is in IDLE state, the only valid reason an interrupt can be generated is as a result of the controller receiving data from the bus. The service routine must check for this by testing the RXPKTRDY bit of PERI_CSR0. If this bit is set, then the controller has received a SETUP packet. This must be unloaded from the FIFO and decoded to determine the action the controller must take. Depending on the command contained within the SETUP packet, endpoint 0 will enter one of three states:

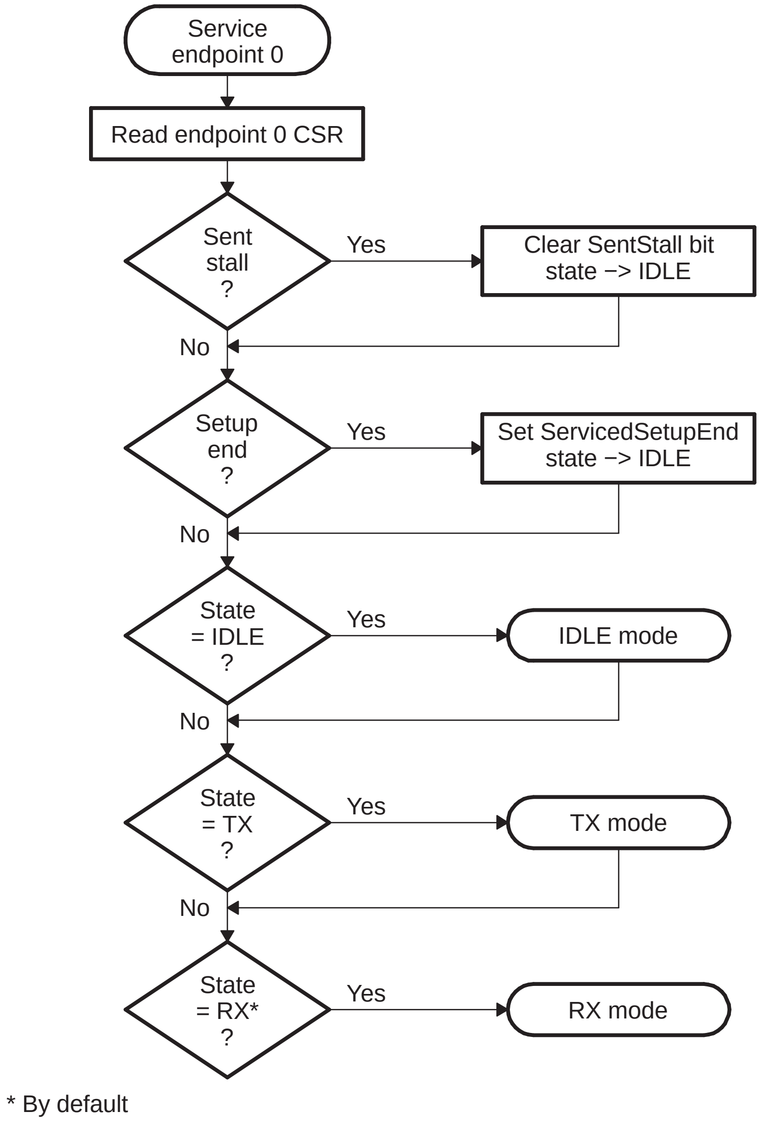
- If the command is a single packet transaction (SET_ADDRESS, SET_INTERFACE etc.) without any data phase, the endpoint will remain in IDLE state.
- If the command has an OUT data phase (SET_DESCRIPTOR etc.), the endpoint will enter RX state.
- If the command has an IN data phase (GET_DESCRIPTOR etc.), the endpoint will enter TX state.

If the endpoint 0 is in TX state, the interrupt indicates that the core has received an IN token and data from the FIFO has been sent. The software must respond to this either by placing more data in the FIFO if the host is still expecting more data or by setting the DATAEND bit to indicate that the data phase is complete. Once the data phase of the transaction has been completed, endpoint 0 should be returned to IDLE state to await the next control transaction.

NOTE: All command transactions include a field that indicates the amount of data the host expects to receive or is going to send.

If the endpoint is in RX state, the interrupt indicates that a data packet has been received. The software must respond by unloading the received data from the FIFO. The software must then determine whether it has received all of the expected data. If it has, the software should set the DATAEND bit and return endpoint 0 to IDLE state. If more data is expected, the firmware should set the SERV_RXPKTRDY bit of PERI_CSR0 to indicate that it has read the data in the FIFO and leave the endpoint in RX state.

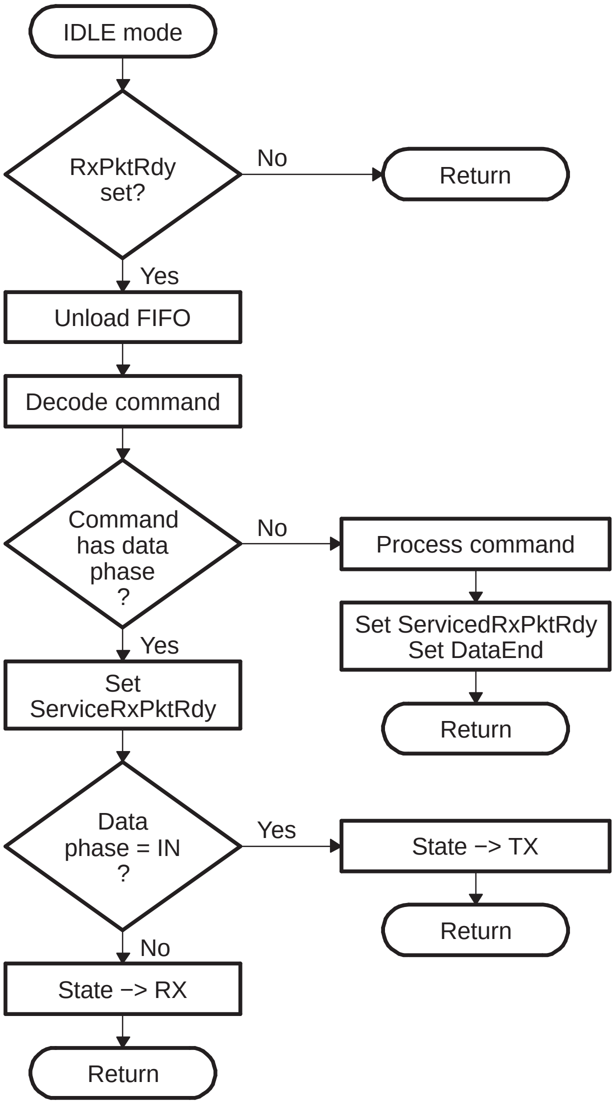
Figure 13-7. Service Endpoint 0 Flow Chart



13.2.8.3.5.1 IDLE Mode

IDLE mode is the mode the endpoint 0 control must select at power-on or reset and is the mode to which the endpoint 0 control should return when the RX and TX modes are terminated. It is also the mode in which the SETUP phase of control transfer is handled (as outlined in Figure 13-8).

Figure 13-8. IDLE Mode Flow Chart



13.2.8.3.5.2 TX Mode

When the endpoint is in TX state all arriving IN tokens need to be treated as part of a data phase until the required amount of data has been sent to the host. If either a SETUP or an OUT token is received while the endpoint is in the TX state, this will cause a SetupEnd condition to occur as the core expects only IN tokens. See Figure 13-9.

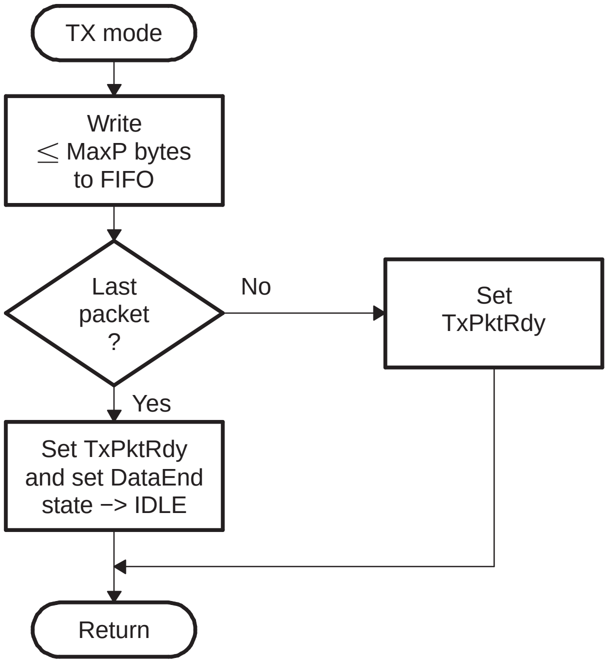
Three events can cause TX mode to be terminated before the expected amount of data has been sent:

1. The host sends an invalid token causing a SETUPEND condition (bit 4 of PERI_CSR0 set).
2. The software sends a packet containing less than the maximum packet size for endpoint 0.
3. The software sends an empty data packet.

Until the transaction is terminated, the software simply needs to load the FIFO when it receives an interrupt that indicates a packet has been sent from the FIFO. (An interrupt is generated when TXPKTRDY is cleared.)

When the software forces the termination of a transfer (by sending a short or empty data packet), it should set the DATAEND bit of PERI_CSR0 (bit 3) to indicate to the core that the data phase is complete and that the core should next receive an acknowledge packet.

Figure 13-9. TX Mode Flow Chart



13.2.8.3.5.3 RX Mode

In RX mode, all arriving data should be treated as part of a data phase until the expected amount of data has been received. If either a SETUP or an IN token is received while the endpoint is in RX state, a SetupEnd condition will occur as the controller expects only OUT tokens.

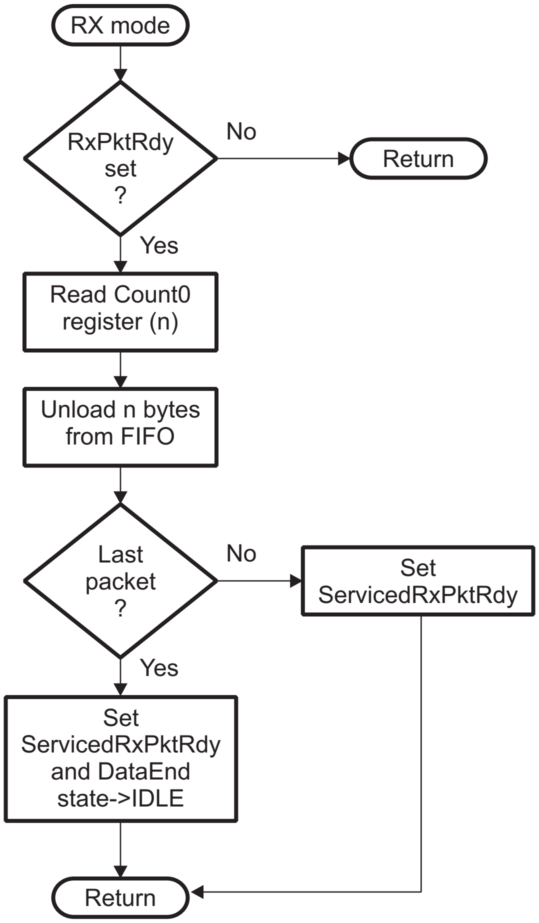
Three events can cause RX mode to be terminated before the expected amount of data has been received as shown in Figure 13-10:

1. The host sends an invalid token causing a SETUPEND condition (setting bit 4 of PERI_CSR0).
2. The host sends a packet which contains less than the maximum packet size for endpoint 0.
3. The host sends an empty data packet.

Until the transaction is terminated, the software unloads the FIFO when it receives an interrupt that indicates new data has arrived (setting RXPKTRDY bit of PERI_CSR0) and to clear RXPKTRDY by setting the SERV_RXPKTRDY bit of PERI_CSR0 (bit 6).

When the software detects the termination of a transfer (by receiving either the expected amount of data or an empty data packet), it should set the DATAEND bit (bit 3 of PERI_CSR0) to indicate to the controller that the data phase is complete and that the core should receive an acknowledge packet next.

Figure 13-10. RX Mode Flow Chart



13.2.8.3.5.4 Error Handling

A control transfer may be aborted due to a protocol error on the USB, the host prematurely ending the transfer, or if the software wishes to abort the transfer (for example, because it cannot process the command).

The controller automatically detects protocol errors and sends a STALL packet to the host under the following conditions:

- The host sends more data during the OUT Data phase of a write request than was specified in the command. This condition is detected when the host sends an OUT token after the DATAEND bit (bit 3 of PERI_CSR0) has been set.
- The host requests more data during the IN Data phase of a read request than was specified in the command. This condition is detected when the host sends an IN token after the DATAEND bit in the PERI_CSR0 register has been set.
- The host sends more than Max Packet Size data bytes in an OUT data packet.
- The host sends a non-zero length DATA1 packet during the STATUS phase of a read request.

When the controller has sent the STALL packet, it sets the SENTSTALL bit (bit 2 of PERI_CSR0) and generates an interrupt. When the software receives an endpoint 0 interrupt with the SENTSTALL bit set, it should abort the current transfer, clear the SENTSTALL bit, and return to the IDLE state.

If the host prematurely ends a transfer by entering the STATUS phase before all the data for the request has been transferred, or by sending a new SETUP packet before completing the current transfer, then the SETUPEND bit (bit 4 of PERI_CSR0) will be set and an endpoint 0 interrupt generated. When the software receives an endpoint 0 interrupt with the SETUPEND bit set, it should abort the current transfer, set the SERV_SETUPEND bit (bit 7 of PERI_CSR0), and return to the IDLE state. If the RXPKTRDY bit (bit 0 of PERI_CSR0) is set this indicates that the host has sent another SETUP packet and the software should then process this command.

If the software wants to abort the current transfer, because it cannot process the command or has some other internal error, then it should set the SENDSTALL bit (bit 5 of PERI_CSR0). The controller will then send a STALL packet to the host, set the SENTSTALL bit (bit 2 of PERI_CSR0) and generate an endpoint 0 interrupt.

13.2.8.3.5.5 Additional Conditions

The controller automatically responds to certain conditions on the USB bus or actions by the host. The details are:

- Stall Issued to Control Transfers
  - The host sends more data during an OUT Data phase of a Control transfer than was specified in the device request during the SETUP phase. This condition is detected by the controller when the host sends an OUT token (instead of an IN token) after the software has unloaded the last OUT packet and set DataEnd.
  - The host requests more data during an IN data phase of a Control transfer than was specified in the device request during the SETUP phase. This condition is detected by the controller when the host sends an IN token (instead of an OUT token) after the software has cleared TXPKTRDY and set DataEnd in response to the ACK issued by the host to what should have been the last packet.
  - The host sends more than MaxPktSize data with an OUT data token.
  - The host sends the wrong PID for the OUT Status phase of a Control transfer.
  - The host sends more than a zero length data packet for the OUT Status phase.
- Zero Length Out Data Packets In Control Transfer
  - A zero length OUT data packet is used to indicate the end of a Control transfer. In normal operation, such packets should only be received after the entire length of the device request has been transferred (i.e., after the software has set DataEnd). If, however, the host sends a zero length OUT data packet before the entire length of device request has been transferred, this signals the premature end of the transfer. In this case, the controller will automatically flush any IN token loaded by software ready for the Data phase from the FIFO and set SETUPEND bit (bit 4 of PERI_CSR0).

13.2.8.4 Bulk Transactions

13.2.8.4.1 Bulk In Transactions

A Bulk IN transaction is used to transfer non-periodic data from the USB peripheral device to the host.

The following optional features are available for use with a Tx endpoint for Bulk IN transactions:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO awaiting transmission to the host. Double packet buffering is enabled by setting the DPB bit of TXFIFOSZ register (bit 4).
- DMA: If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint is able to accept another packet in its FIFO. This feature allows the DMA controller to load packets into the FIFO without processor intervention.

  When DMA is enabled and DMAMODE bit of PERI_TXCSR is set, an endpoint interrupt is not generated for completion of the packet transfer. An endpoint interrupt is generated only in the error conditions.

13.2.8.4.1.1 Setup

In configuring a TX endpoint for bulk transactions, the TXMAXP register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the endpoint and the PERI_TXCSR register should be set as shown in Table 13-4 when using DMA:

Table 13-4. PERI_TXCSR Register Bit Configuration for Bulk IN Transactions

Bit Position	Bit Field Name	Configuration
Bit 14	ISO	Cleared to 0 for bulk mode operation.
Bit 13	MODE	Set to 1 to make sure the FIFO is enabled (only necessary if the FIFO is shared with an RX endpoint).
Bit 12	DMAEN	Set to 1 if DMA requests must be enabled.
Bit 11	FRCDATATOG	Cleared to 0 to allow normal data toggle operations.
Bit 10	DMAMODE	Set to 1 when DMA is enabled and EP interrupt is not needed for each packet transmission.

When the endpoint is first configured (following a SET_CONFIGURATION or SET_INTERFACE command on Endpoint 0), the lower byte of PERI_TXCSR should be written to set the CLRDATATOG bit (bit 6). This will ensure that the data toggle (which is handled automatically by the controller) starts in the correct state.

Also if there are any data packets in the FIFO, indicated by the FIFONOTEMPTY bit (bit 1 of PERI_TXCSR) being set, they should be flushed by setting the FLUSHFIFO bit (bit 3 of PERI_TXCSR).

NOTE: It may be necessary to set this bit twice in succession if double buffering is enabled.

13.2.8.4.1.2 Operation

When data is to be transferred over a Bulk IN pipe, a data packet needs to be loaded into the FIFO and the PERI_TXCSR register written to set the TXPKTRDY bit (bit 0). When the packet has been sent, the TXPKTRDY bit will be cleared by the USB controller and an interrupt generated so that the next packet can be loaded into the FIFO. If double packet buffering is enabled, then after the first packet has been loaded and the TXPKTRDY bit set, the TXPKTRDY bit will immediately be cleared by the USB controller and an interrupt generated so that a second packet can be loaded into the FIFO. The software should operate in the same way, loading a packet when it receives an interrupt, regardless of whether double packet buffering is enabled or not.

In the general case, the packet size must not exceed the size specified by the lower 11 bits of the TXMAXP register. This part of the register defines the payload (packet size) for transfers over the USB and is required by the USB Specification to be either 8, 16, 32, 64 (Full-Speed or High-Speed) or 512 bytes (High-Speed only).

The host may determine that all the data for a transfer has been sent by knowing the total amount of data that is expected. Alternatively it may infer that all the data has been sent when it receives a packet which is smaller than the stated payload (TXMAXP[10-0]). In the latter case, if the total size of the data block is a multiple of this payload, it will be necessary for the function to send a null packet after all the data has been sent. This is done by setting TXPKTRDY when the next interrupt is received, without loading any data into the FIFO.

If large blocks of data are being transferred, then the overhead of calling an interrupt service routine to load each packet can be avoided by using DMA.

13.2.8.4.1.3 Error Handling

If the software wants to shut down the Bulk IN pipe, it should set the SENDSTALL bit (bit 4 of PERI_TXCSR). When the controller receives the next IN token, it will send a STALL to the host, set the SENTSTALL bit (bit 5 of PERI_TXCSR) and generate an interrupt.

When the software receives an interrupt with the SENTSTALL bit (bit 5 of PERI_TXCSR) set, it should clear the SENTSTALL bit. It should however leave the SENDSTALL bit set until it is ready to re-enable the Bulk IN pipe.

NOTE: If the host failed to receive the STALL packet for some reason, it will send another IN token, so it is advisable to leave the SENDSTALL bit set until the software is ready to re-enable the Bulk IN pipe. When a pipe is re-enabled, the data toggle sequence should be restarted by setting the CLRDATATOG bit in the PERI_TXCSR register (bit 6).

13.2.8.4.2 Bulk OUT Transactions

A Bulk OUT transaction is used to transfer non-periodic data from the host to the function controller.

The following optional features are available for use with an Rx endpoint for Bulk OUT transactions:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO on reception from the host. Double packet buffering is enabled by setting the DPB bit of the RXFIFOSZ register (bit 4).
- DMA: If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint has a packet in its FIFO. This feature can be used to allow the DMA controller to unload packets from the FIFO without processor intervention.

  When DMA is enabled, endpoint interrupt will not be generated for completion of packet reception. Endpoint interrupt will be generated only in the error conditions.

13.2.8.4.2.1 Setup

In configuring an Rx endpoint for Bulk OUT transactions, the RXMAXP register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the endpoint. In addition, the relevant interrupt enable bit in the INTRRXE register should be set (if an interrupt is required for this endpoint) and the PERI_RXCSR register should be set as shown in Table 13-5.

Table 13-5. PERI_RXCSR Register Bit Configuration for Bulk OUT Transactions

Bit Position	Bit Field Name	Configuration
Bit 14	ISO	Cleared to 0 to enable Bulk protocol.
Bit 13	DMAEN	Set to 1 if a DMA request is required for this endpoint.
Bit 12	DISNYET	Cleared to 0 to allow normal PING flow control. This will affect only high speed transactions.
Bit 11	DMAMODE	Always clear this bit to 0.

When the endpoint is first configured (following a SET_CONFIGURATION or SET_INTERFACE command on Endpoint 0), the lower byte of PERI_RXCSR should be written to set the CLRDATATOG bit (bit 7). This will ensure that the data toggle (which is handled automatically by the USB controller) starts in the correct state.

Also if there are any data packets in the FIFO (indicated by the RXPKTRDY bit (bit 0 of PERI_RXCSR) being set), they should be flushed by setting the FLUSHFIFO bit (bit 4 of PERI_RXCSR).

NOTE: It may be necessary to set this bit twice in succession if double buffering is enabled.

13.2.8.4.2.2 Operation

When a data packet is received by a Bulk Rx endpoint, the RXPKTRDY bit (bit 0 of PERI_RXCSR) is set and an interrupt is generated. The software should read the RXCOUNT register for the endpoint to determine the size of the data packet. The data packet should be read from the FIFO, then the RXPKTRDY bit should be cleared.

The packets received should not exceed the size specified in the RXMAXP register (as this should be the value set in the wMaxPacketSize field of the endpoint descriptor sent to the host). When a block of data larger than wMaxPacketSize needs to be sent to the function, it will be sent as multiple packets. All the packets will be wMaxPacketSize in size, except the last packet which will contain the residue. The software may use an application specific method of determining the total size of the block and hence when the last packet has been received. Alternatively it may infer that the entire block has been received when it receives a packet which is less than wMaxPacketSize in size. (If the total size of the data block is a multiple of wMaxPacketSize, a null data packet will be sent after the data to signify that the transfer is complete.)

In the general case, the application software will need to read each packet from the FIFO individually. If large blocks of data are being transferred, the overhead of calling an interrupt service routine to unload each packet can be avoided by using DMA.

13.2.8.4.2.3 Error Handling

If the software wants to shut down the Bulk OUT pipe, it should set the SENDSTALL bit (bit 5 of PERI_RXCSR). When the controller receives the next packet it will send a STALL to the host, set the SENTSTALL bit (bit 6 of PERI_RXCSR) and generate an interrupt.

When the software receives an interrupt with the SENTSTALL bit (bit 6 of PERI_RXCSR) set, it should clear this bit. It should however leave the SENDSTALL bit set until it is ready to re-enable the Bulk OUT pipe.

NOTE: If the host failed to receive the STALL packet for some reason, it will send another packet, so it is advisable to leave the SENDSTALL bit set until the software is ready to re-enable the Bulk OUT pipe. When a Bulk OUT pipe is re-enabled, the data toggle sequence should be restarted by setting the CLRDATATOG bit (bit 7) in the PERI_RXCSR register.

13.2.8.5 Interrupt Transactions

An Interrupt IN transaction uses the same protocol as a Bulk IN transaction and can be used the same way. Similarly, an Interrupt OUT transaction uses almost the same protocol as a Bulk OUT transaction and can be used the same way.

Tx endpoints in the USB controller have one feature for Interrupt IN transactions that they do not support in Bulk IN transactions. In Interrupt IN transactions, the endpoints support continuous toggle of the data toggle bit.

This feature is enabled by setting the FRCDATATOG bit in the PERI_TXCSR register (bit 11). When this bit is set, the controller will consider the packet as having been successfully sent and toggle the data bit for the endpoint, regardless of whether an ACK was received from the host.

Another difference is that interrupt endpoints do not support PING flow control. This means that the controller should never respond with a NYET handshake, only ACK/NAK/STALL. To ensure this, the DISNYET bit in the PERI_RXCSR register (bit 12) should be set to disable the transmission of NYET handshakes in high-speed mode.

Though DMA can be used with an interrupt OUT endpoint, it generally offers little benefit as interrupt endpoints are usually expected to transfer all their data in a single packet.

13.2.8.6 Isochronous Transactions

13.2.8.6.1 Isochronous IN Transactions

An Isochronous IN transaction is used to transfer periodic data from the function controller to the host.

The following optional features are available for use with a Tx endpoint for Isochronous IN transactions:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO awaiting transmission to the host. Double packet buffering is enabled by setting the DPB bit of TXFIFOSZ register (bit 4).

NOTE: Double packet buffering is generally advisable for Isochronous transactions in order to avoid Underrun errors as described in later section.

- DMA: If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint is able to accept another packet in its FIFO. This feature allows the DMA controller to load packets into the FIFO without processor intervention.

  However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size and the PERI_TXCSR register needs to be accessed following every packet to check for Underrun errors.

  When DMA is enabled and DMAMODE bit of PERI_TXCSR is set, endpoint interrupt will not be generated for completion of packet transfer. Endpoint interrupt will be generated only in the error conditions.

13.2.8.6.1.1 Setup

In configuring a Tx endpoint for Isochronous IN transactions, the TXMAXP register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the endpoint. In addition, the relevant interrupt enable bit in the INTRTXE register should be set (if an interrupt is required for this endpoint) and the PERI_TXCSR register should be set as shown in Table 13-6.

Table 13-6. PERI_TXCSR Register Bit Configuration for Isochronous IN Transactions

Bit Position	Bit Field Name	Configuration
Bit 14	ISO	Set to 1 to enable Isochronous transfer protocol.
Bit 13	MODE	Set to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).
Bit 12	DMAEN	Set to 1 if DMA Requests have to be enabled.
Bit 11	FRCDATATOG	Ignored in Isochronous mode.
Bit 10	DMAMODE	Set it to 1, when DMA is enabled and EP interrupt is not needed for each packet transmission.

13.2.8.6.1.2 Operation

An Isochronous endpoint does not support data retries, so if data underrun is to be avoided, the data to be sent to the host must be loaded into the FIFO before the IN token is received. The host will send one IN token per frame (or microframe in High-speed mode), however the timing within the frame (or microframe) can vary. If an IN token is received near the end of one frame and then at the start of the next frame, there will be little time to reload the FIFO. For this reason, double buffering of the endpoint is usually necessary.

An interrupt is generated whenever a packet is sent to the host and the software may use this interrupt to load the next packet into the FIFO and set the TXPKTRDY bit in the PERI_TXCSR register (bit 0) in the same way as for a Bulk Tx endpoint. As the interrupt could occur almost any time within a frame(/microframe), depending on when the host has scheduled the transaction, this may result in irregular timing of FIFO load requests. If the data source for the endpoint is coming from some external hardware, it may be more convenient to wait until the end of each frame(/microframe) before loading the FIFO as this will minimize the requirement for additional buffering. This can be done by using either the SOF interrupt or the external SOF_PULSE signal from the controller to trigger the loading of the next data packet. The SOF_PULSE is generated once per frame(/microframe) when a SOF packet is received. (The controller also maintains an external frame(/microframe) counter so it can still generate a SOF_PULSE when the SOF packet has been lost.) The interrupts may still be used to set the TXPKTRDY bit in PERI_TXCSR (bit 0) and to check for data overruns/underruns.

Starting up a double-buffered Isochronous IN pipe can be a source of problems. Double buffering requires that a data packet is not transmitted until the frame(/microframe) after it is loaded. There is no problem if the function loads the first data packet at least a frame(/microframe) before the host sets up the pipe (and therefore starts sending IN tokens). But if the host has already started sending IN tokens by the time the first packet is loaded, the packet may be transmitted in the same frame(/microframe) as it is loaded, depending on whether it is loaded before, or after, the IN token is received. This potential problem can be avoided by setting the ISOUPDATE bit in the POWER register (bit 7). When this bit is set, any data packet loaded into an Isochronous Tx endpoint FIFO will not be transmitted until after the next SOF packet has been received, thereby ensuring that the data packet is not sent too early.

13.2.8.6.1.3 Error Handling

If the endpoint has no data in its FIFO when an IN token is received, it will send a null data packet to the host and set the UNDERRUN bit in the PERI_TXCSR register (bit 2). This is an indication that the software is not supplying data fast enough for the host. It is up to the application to determine how this error condition is handled.

If the software is loading one packet per frame(/microframe) and it finds that the TXPKTRDY bit in the PERI_TXCSR register (bit 0) is set when it wants to load the next packet, this indicates that a data packet has not been sent (perhaps because an IN token from the host was corrupted). It is up to the application how it handles this condition: it may choose to flush the unsent packet by setting the FLUSHFIFO bit in the PERI_TXCSR register (bit 3), or it may choose to skip the current packet.

13.2.8.6.2 Isochronous OUT Transactions

An Isochronous OUT transaction is used to transfer periodic data from the host to the function controller.

Following optional features are available for use with an Rx endpoint for Isochronous OUT transactions:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO on reception from the host. Double packet buffering is enabled by setting the DPB bit of RXFIFOSZ register (bit 4).

NOTE: Double packet buffering is generally advisable for Isochronous transactions in order to avoid Overrun errors.

- DMA: If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint has a packet in its FIFO. This feature can be used to allow the DMA controller to unload packets from the FIFO without processor intervention.

  However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size and the PERI_RXCSR register needs to be accessed following every packet to check for Overrun or CRC errors.

  When DMA is enabled, endpoint interrupt will not be generated for completion of packet reception. Endpoint interrupt will be generated only in the error conditions.

13.2.8.6.2.1 Setup

In configuring an Rx endpoint for Isochronous OUT transactions, the RXMAXP register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the endpoint. In addition, the relevant interrupt enable bit in the INTRRXE register should be set (if an interrupt is required for this endpoint) and the PERI_RXCSR register should be set as shown in Table 13-7.

Table 13-7. PERI_RXCSR Register Bit Configuration for Isochronous OUT Transactions

Bit Position	Bit Field Name	Configuration
Bit 14	ISO	Set to 1 to enable isochronous protocol.
Bit 13	DMAEN	Set to 1 if a DMA request is required for this endpoint.
Bit 12	DISNYET	Ignored in isochronous transfers.
Bit 11	DMAMODE	Always clear this bit to 0.

13.2.8.6.2.2 Operation

An Isochronous endpoint does not support data retries so, if a data overrun is to be avoided, there must be space in the FIFO to accept a packet when it is received. The host will send one packet per frame (or microframe in High-speed mode); however, the time within the frame can vary. If a packet is received near the end of one frame(/microframe) and another arrives at the start of the next frame, there will be little time to unload the FIFO. For this reason, double buffering of the endpoint is usually necessary.

An interrupt is generated whenever a packet is received from the host and the software may use this interrupt to unload the packet from the FIFO and clear the RXPKTRDY bit in the PERI_RXCSR register (bit 0) in the same way as for a Bulk Rx endpoint. As the interrupt could occur almost any time within a frame(/microframe), depending on when the host has scheduled the transaction, the timing of FIFO unload requests will probably be irregular. If the data sink for the endpoint is going to some external hardware, it may be better to minimize the requirement for additional buffering by waiting until the end of each frame(/microframe) before unloading the FIFO. This can be done by using either the SOF interrupt or the external SOF_PULSE signal from the controller to trigger the unloading of the data packet. The SOF_PULSE is generated once per frame(/microframe) when a SOF packet is received. (The controller also maintains an external frame(/microframe) counter so it can still generate a SOF_PULSE when the SOF packet has been lost.) The interrupts may still be used to clear the RXPKTRDY bit in PERI_RXCSR and to check for data overruns/underruns.

13.2.8.6.2.3 Error Handling

If there is no space in the FIFO to store a packet when it is received from the host, the OVERRUN bit in the PERI_RXCSR register (bit 2) will be set. This is an indication that the software is not unloading data fast enough for the host. It is up to the application to determine how this error condition is handled.

If the controller finds that a received packet has a CRC error, it will still store the packet in the FIFO and set the RXPKTRDY bit (bit 0 of PERI_RXCSR) and the DATAERROR bit (bit 3 of PERI_RXCSR). It is left up to the application how this error condition is handled.

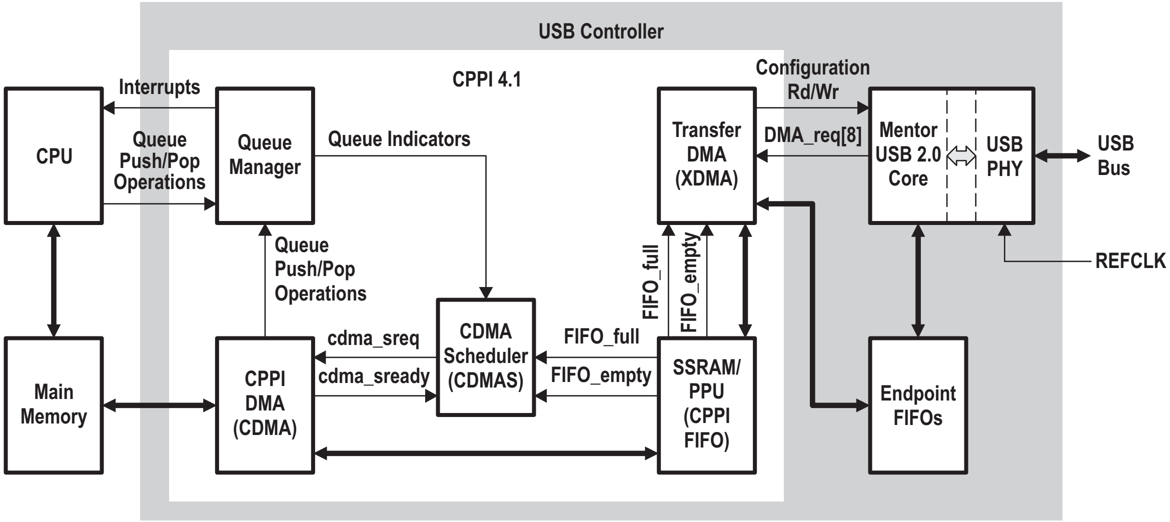
13.2.9 Communications Port Programming Interface (CPPI) 4.1 DMA Overview

The CPPI DMA module supports the transmission and reception of USB packets. The CPPI DMA is designed to facilitate the segmentation and reassembly of CPPI compliant packets to/from smaller data blocks that are natively compatible with the specific requirements of each networking port. Multiple Tx and Rx channels are provided within the DMA which allow multiple segmentation or reassembly operations to be effectively performed in parallel (but not actually simultaneously). The DMA controller maintains state information for each of the ports/channels which allows packet segmentation and reassembly operations to be time division multiplexed between channels in order to share the underlying DMA hardware. A DMA scheduler is used to control the ordering and rate at which this multiplexing occurs.

The CPPI (version 4.1) DMA controller sub-module is a common 4 dual-port DMA Controller. It supports 4 Tx and 4 Rx Ports and each port attaches to the associated endpoint in the controller. Port 1 maps to endpoint 1 and Port 2 maps to endpoint 2 and Port 3 maps to endpoint 3 and Port 4 maps to endpoint 4, while endpoint 0 can not utilize the DMA and the firmware is responsible to load or offload the endpoint 0 FIFO via CPU.

Figure 13-11 displays the USB controller block diagram.

Figure 13-11. USB Controller Block Diagram



Host— The host is an intelligent system resource that configures and manages each communications control module. The host is responsible for allocating memory, initializing all data structures, and responding to port interrupts.

Main Memory— The area of data storage managed by the CPU. The CPPI DMA (CDMA) reads and writes CPPI packets from and to main memory. This memory can exist internal or external from the device.

Queue Manager (QM)— The QM is responsible for accelerating management of a variety of Packet Queues and Free Descriptor / Buffer Queues. It provides status indications to the CDMA Scheduler when queues are empty or full.

CPPI DMA (CDMA)— The CDMA is responsible for transferring data between the CPPI FIFO and Main Memory. It acquires free Buffer Descriptor from the QM (Receive Submit Queue) for storage of received data, posts received packets pointers to the Receive Completion Queue, transmits packets stored on the Transmit Submit Queue (Transmit Queue) , and posts completed transmit packets to the Transmit Completion Queue.

CDMA Scheduler (CDMAS)— The CDMAS is responsible for scheduling CDMA transmit and receive operations. It uses Queue Indicators from the QM and the CDMA to determine the types of operations to schedule.

CPPI FIFO— The CPPI FIFO provides 8 FIFO interfaces (one for each of the 4 transmit and receive endpoints). Each FIFO contains two 64-byte memory storage elements (ping-pong buffer storage).

Transfer DMA (XDMA)— The XDMA receives DMA requests from the Mentor USB 2.0 Core and initiates DMAs to the CPPI FIFO.

Endpoint FIFOs— The Endpoint FIFOs are the USB packet storage elements used by the Mentor USB 2.0 Core for packet transmission or reception. The XDMA transfers data between the CPPI FIFO and the Endpoint FIFOs for transmit operations and between the Endpoint FIFOs and the CPPI FIFO for receive operations.

Mentor USB 2.0 Core— This controller is responsible for processing USB bus transfers (control, bulk, interrupt, and isochronous). It supports 4 transmit and 4 receive endpoints in addition to endpoint 0 (control).

13.2.9.1 CPPI Terminology

The following terms are important in the discussion of DMA CPPI.

Port— A port is the communications module (peripheral hardware) that contains the control logic for Direct Memory Access for a single transmit/receive interface or set of interfaces. Each port may have multiple communication channels that transfer data using homogenous or heterogeneous protocols. A port is usually subdivided into transmit and receive pairs which are independent of each other. Each endpoint, excluding endpoint 0, has its own dedicated port.

Channel— A channel refers to the sub-division of information (flows) that is transported across ports. Each channel has associated state information. Channels are used to segregate information flows based on the protocol used, scheduling requirements (example: CBR, VBR, ABR), or concurrency requirements (that is, blocking avoidance). All four ports have dedicated single channels, channel 0, associated for their use in a USB application.

Data Buffer— A data buffer is a single data structure that contains payload information for transmission to or reception from a port. A data buffer is a byte aligned contiguous block of memory used to store packet payload data. A data buffer may hold any portion of a packet and may be linked together (via descriptors) with other buffers to form packets. Data buffers may be allocated anywhere within the device memory map. The Buffer Length field of the packet descriptor indicates the number of valid data bytes in the buffer. There may be from 1 to 4M-1 valid data bytes in each buffer.

Host Buffer Descriptor— A buffer descriptor is a single data structure that contains information about one or more data buffers. This type of descriptor is required when more than one descriptor is needed to define an entire packet, i.e., it either defines the middle of a packet or end of a packet.

Host Packet Descriptor— A packet descriptor is another name for the first buffer descriptor within a packet. Some fields within a data buffer descriptor are only valid when it is a packet descriptor including the tags, packet length, packet type, and flags. This type of descriptor is always used to define a packet since it provides packet level information that is useful to both the ports and the Host in order to properly process the packet. It is the only descriptor used when single descriptor solely defines a packet. When multiple descriptors are needed to define a packet, the packet descriptor is the first descriptor used to define a packet.

Free Descriptor/Buffer Queue— A free descriptor/buffer queue is a hardware managed list of available descriptors with pre-linked empty buffers that are to be used by the receive ports for host type descriptors. Free Descriptor/Buffer Queues are implemented by the Queue Manager.

Teardown Descriptor— Teardown Descriptor is a special structure which is not used to describe either a packet or a buffer but is instead used to describe the completion of a channel halt and teardown event. Channel teardown is an important function because it ensures that when a connection is no longer needed that the hardware can be reliably halted and any remaining packets which had not yet been transmitted can be reclaimed by the Host without the possibility of losing buffer or descriptor references (which results in a memory leak).

Packet Queue— A packet queue is hardware managed list of valid (i.e. populated) packet descriptors that is used for forwarding a packet from one entity to another for any number of purposes.

Queue Manager— The queue manager is a hardware module that is responsible for accelerating management of the packet queues. Packets are added to a packet queue by writing the 32-bit descriptor address to a particular memory mapped location in the Queue Manager module. Packets are de-queued by reading the same location for that particular queue. A single Queue Manager is used for a USB application.

NOTE: All descriptors (regardless of type) must be allocated at addresses that are naturally aligned to the smallest power of 2 that is equal to or greater than the descriptor size.

13.2.9.2 Host Packet Descriptor (SOP Descriptor)

Host Packet Descriptors are designed to be used when USB like application requires support for true, unlimited fragment count scatter/gather type operations. The Host Packet Descriptor is the first descriptor on multiple descriptors setup or the only descriptor in a single descriptors setup. The Host Packet Descriptor contains the following information:

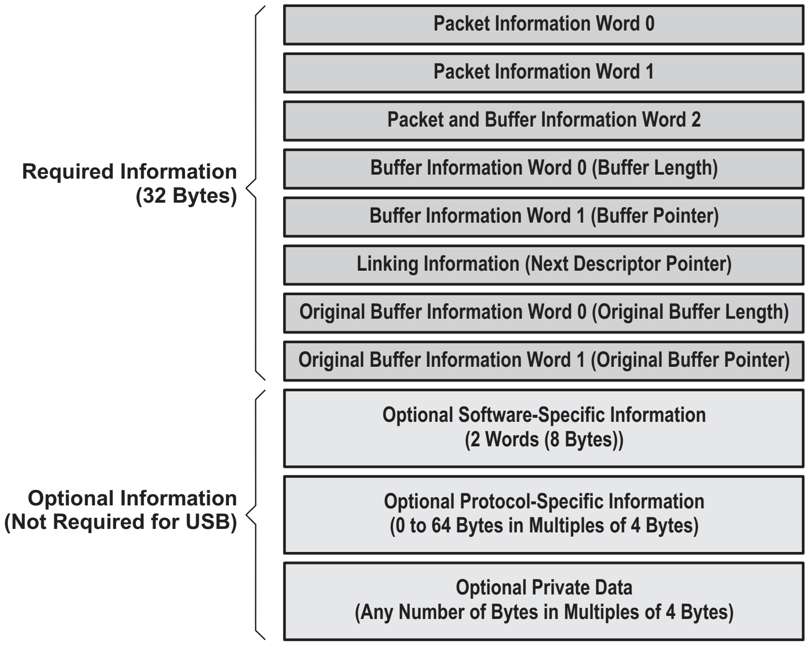
- Indicator which identifies the descriptor as a Host Packet Descriptor (always 10h)
- Source and Destination Tags (Reserved)
- Packet Type
- Packet Length
- Protocol Specific Region Size
- Protocol Specific Control/Status Bits
- Pointer to the first valid byte in the SOP data buffer
- Length of the SOP data buffer
- Pointer to the next buffer descriptor in the packet

Host Packet Descriptors can vary in size of their defined fields from 32 bytes up to 104 bytes. Within this range, Host Packet Descriptors always contain 32 bytes of required information and may also contain 8 bytes of software specific tagging information and up to 64 bytes (indicated in 4 byte increments) of protocol specific information. How much protocol specific information (and therefore the allocated size of the descriptors) is application dependent.

NOTE: Descriptors can be located anywhere within the 16MB address space of the device (except for DARAM, which is not accessible by the USB controller). However, all descriptors must be placed in a single contiguous block of up to 64KW.

The Host Packet Descriptor layout is shown in Figure 13-12.

Figure 13-12. Host Packet Descriptor Layout


Required Information
(32 Bytes)
Packet Information Word 0
Packet Information Word 1
Packet and Buffer Information Word 2
Buffer Information Word 0 (Buffer Length)
Buffer Information Word 1 (Buffer Pointer)
Linking Information (Next Descriptor Pointer)
Original Buffer Information Word 0 (Original Buffer Length)
Original Buffer Information Word 1 (Original Buffer Pointer)
Optional Information
(Not Required for USB)
Optional Software-Specific Information
(2 Words (8 Bytes))
Optional Protocol-Specific Information
(0 to 64 Bytes in Multiples of 4 Bytes)
Optional Private Data
(Any Number of Bytes in Multiples of 4 Bytes)

13.2.9.3 Host Buffer Descriptor (Non-SOP Descriptor)

The Host Buffer Descriptor is identical in size and organization to a Host Packet Descriptor but does not include valid information in the packet level fields and does not include a populated region for protocol specific information. The packet level fields is not needed since the SOP descriptor contain this information and additional copy of this data is not needed/necessary.

Host Buffer Descriptors are designed to be linked onto a Host Packet Descriptor or another Host Buffer Descriptor to provide support for unlimited scatter / gather type operations. Host Buffer Descriptors provide information about a single corresponding data buffer. Every Host buffer descriptor stores the following information:

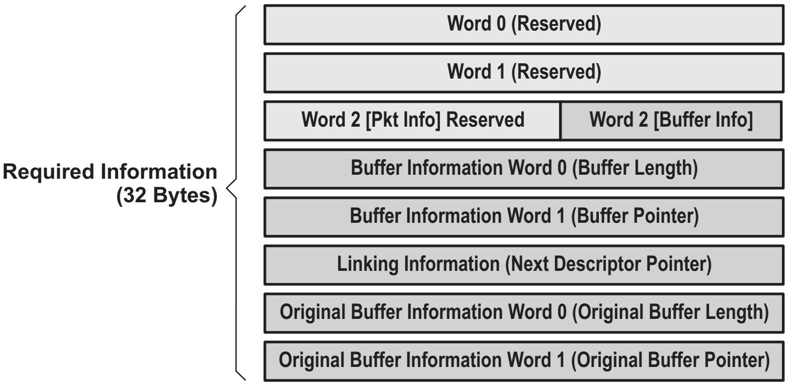
- Pointer to the first valid byte in the data buffer
- Length of the data buffer
- Pointer to the next buffer descriptor in the packet

Host Buffer Descriptors always contain 32 bytes of required information. Since it is a requirement that it is possible to convert a Host descriptor between a Buffer Descriptor and a Packet Descriptor (by filling in the appropriate fields) in practice, Host Buffer Descriptors will be allocated using the same sizes as Host Packet Descriptors. In addition, since the 5 LSBs of the Descriptor Pointers are used in CPPI 4.1 for the purpose of indicating the length of the descriptor, the minimum size of a descriptor is always 32 bytes. (For more information on Descriptor Size, see Section 13.3.82).

NOTE: Descriptors can be located anywhere within the 16MB address space of the device (except for DARAM, which is not accessible by the USB controller). However, all descriptors must be placed in a single contiguous block of up to 64KW.

The descriptor layout is shown in Figure 13-13.

Figure 13-13. Host Buffer Descriptor Layout


Required Information
(32 Bytes)
Word 0 (Reserved)
Word 1 (Reserved)
Word 2 [Pkt Info] Reserved
Word 2 [Buffer Info]
Buffer Information Word 0 (Buffer Length)
Buffer Information Word 1 (Buffer Pointer)
Linking Information (Next Descriptor Pointer)
Original Buffer Information Word 0 (Original Buffer Length)
Original Buffer Information Word 1 (Original Buffer Pointer)

13.2.9.4 Teardown Descriptor

The Teardown Descriptor is not like the Host Packet or Buffer Descriptors since it is not used to describe either a packet or a buffer. The Teardown Descriptor is always 32 bytes long and is comprised of 4 bytes of actual teardown information and 28 bytes of pad (see Figure 13-14). Since the 5 LSBs of the Descriptor Pointers are used in CPPI 4.1 for the purpose of indicating the length of the descriptor, the minimum size of a descriptor is 32 bytes.

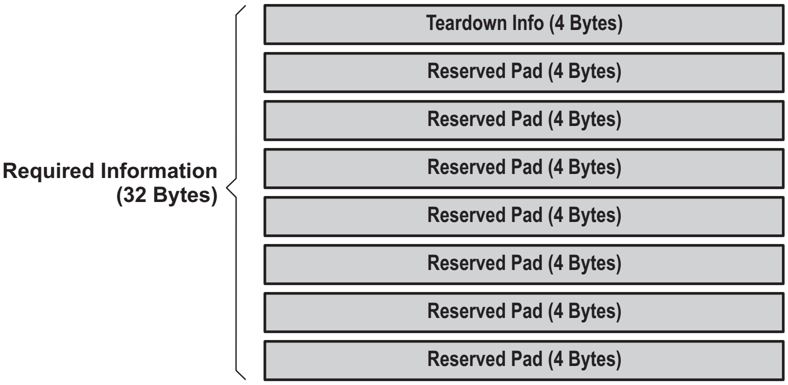
Teardown Descriptor is used to describe a channel halt and teardown event. Channel teardown ensures that when a connection is no longer needed that the hardware can be reliably halted and any remaining packets which had not yet been transmitted can be reclaimed by the Host without the possibility of losing buffer or descriptor references (which results in a memory leak).

NOTE: Descriptors can be located anywhere within the 16MB address space of the device (except for DARAM, which is not accessible by the USB controller). However, all descriptors must be placed in a single contiguous block of up to 64KW.

The Teardown Descriptor contains the following information:

- Indicator which identifies the descriptor as a Teardown Packet Descriptor
- DMA Controller Number where teardown occurred
- Channel number within DMA where teardown occurred
- Indicator of whether this teardown was for the Tx or Rx channel

Figure 13-14. Teardown Descriptor Layout



Teardown operation of an endpoint requires three operations. The teardown register in the CPPI DMA must be written, the corresponding endpoint bit in TEARDOWN of the USB module must be set, and the FlushFIFO bit in the Mentor USB controller TX/RXCSR register must be set. Writing to TEARDOWN in the USB2.0 module resets the CPPI FIFO occupancy value and pointers to 0. It also resets the current state of the XDMA for the endpoint selected, after any current operations have been completed. Note that due to VBUSP bridge latency, the CPPI FIFO occupancy values will not be reset immediately upon writing of TEARDOWN.

13.2.9.5 Queues

Several types of queues exist (a total of 64 queues) within the CPPI 4.1 DMA. Regardless of the type of queue a queue is, queues are used to hold pointers to host or buffer packet descriptors while they are being passed between the Host and / or any of the ports in the system. All queues are maintained within the Queue Manager module.

The following type of Queues exist:

- Receive Free Descriptor/Buffer Queue
- Receive Completion (Return) Queue
- Transmit Submit Queue (also referred as Transmit Queue)
- Transmit Completion (Return) Queue
- Free Descriptor Queue (Unassigned: Can be used for Completion or Application Specific purposes)

Table 13-8 displays the allocation (partition) of the available Queues.

Table 13-8. Allocation of Queues

Starting Queue Number	Number of Queues	Function
0	16	RX +Free Descriptor/Buffer (submit) queues
16	2	USB Endpoint 1 TX (submit) queues
18	2	USB Endpoint 2 TX (submit) queues
20	2	USB Endpoint 3 TX (submit) queues
22	2	USB Endpoint 4 TX (submit) queues
24	2	TX Completion (return) queues
26	2	RX Completion (return) queues
28	36	Unassigned (application-defined) queues

13.2.9.5.1 Queuing Packets

Prior to queuing packets, the host/firmware should construct data buffer as well host packet/buffer descriptors within the 16MB address space of the device (except for DARAM which is not accessible by the USB controller).

NOTE: Descriptors must be placed in a single contiguous block of up to 64KW anywhere in the 16MB address space of the device, except DARAM which is not accessible by the USB controller.

Queuing of packets onto a packet queue is accomplished by writing a pointer to the Packet Descriptor into a specific address within the selected queue (Register D of Queue N). Packet is always queued onto the tail of the queue. The Queue Manager provides a unique set of addresses for adding packets for each queue that it manages.

NOTE: The control register D for each queue is split up into two registers (CTRL1D and CTRL2D). To load a descriptor pointer into a queue, use a single double word write to CTRL1D.

13.2.9.5.2 De-Queuing Packets

De-queuing of packets from a packet queue is accomplished by reading the head packet pointer from a specific address within the selected queue (Register D of Queue N). After the head pointer has been read, the Queue Manager will invalidate the head pointer and will replace it with the next packet pointer in the queue. This functionality, which is implemented in the Queue Manager, prevents the ports from needing to traverse linked lists and allows for certain optimizations to be performed within the Queue Manager.

NOTE: The control register D for each queue is split up into two registers (CTRL1D and CTRL2D). To unload a descriptor pointer into a queue, use a single word read from CTRL1D. The return value will be the lower 16 bits of the descriptor address. Since all descriptors must be within a 64KW memory range, a read from CTRL2D is not necessary.

13.2.9.5.3 Type of Queues

Several types of queues exist and all are managed by the Queue Manager which is part of the CPPI 4.1 DMA. All accesses to the queues are through memory mapped registers and no external memory setup is required by the firmware.

13.2.9.5.3.1 Receive Free Descriptor/Buffer (Submit) Queue

Receive ports use queues referred to as "receive free descriptor / buffer queues" to forward completed receive packets to the host or another peer port entity. The entries on the Free Descriptor / Buffer Queues have pre-attached empty buffers whose size and location are described in the "original buffer information" fields in the descriptor. The host is required to allocate both the descriptor and buffer and pre-link them prior to adding (submitting) a descriptor to one of the available receive free descriptor / buffer queue. The first 16 queues (Queue 0 up to Queue 15) are reserved for all four receive ports to handle incoming packets.

13.2.9.5.3.2 Transmit (Submit) Queue

Transmit ports use packet queues referred to as "transmit (submit) queues" to store the packets that are waiting to be transmitted. Each port has dedicated queues (2 queues per port) that are reserved exclusively for a use by a single port. Multiple queues per port/channel are allocated to facilitate Quality of Service (QoS) for applications that require QoS. Queue 16 and 17 are allocated for port 1, Queue 18 and 19 are allocated for port 2 and Queue 20 and Queue 21 are allocated for port 3 and Queue 22 and 23 are allocated for port 4.

13.2.9.5.3.3 Transmit Completion Queue

Transmit ports also use packet queues referred to as "transmit completion queues" to return packets to the host after they have been transmitted. Even though, non-allocated queues can be used for this purpose, a total of two dedicated queues (Queue 24 and Queue 25), that is to be shared amongst all four transmit ports, have been reserved for returning transmit packets after end of transmit operation when the firmware desires to receive interrupt when transmission completes.

13.2.9.5.3.4 Receive Completion Queue

Receive ports also use packet queues referred to as "receive completion queues" to return packets to the port after they have been received. Even though, non-allocated queues can be used for this purpose, a total of two dedicated queues (Queue 26 and Queue 27), that is to be shared amongst all four transmit ports, have been reserved for returning received packets to the receive ports after end of receive operation when the firmware desires to receive interrupt when transmission completes.

13.2.9.5.3.5 Unassigned (Application Defined) Queue

Thirty-six additional queues (Queue 28 to Queue 63) exist that have not been dedicated for exclusive use. The user can use these queues as a Completion Queues or Free Descriptor/Buffer queue.

When these queues are used as Completion Queues, interrupt will not be generated. However, the queues will have the list of descriptor pointers for the packets that have completed transmission or reception. The firmware can use polling method by continually performing the de-queuing technique onto the particular unassigned queue used to identify if the reception or transmission has completed.

When unassigned queues are used as free descriptor/buffer queue, the user can use these queues to queue/store available descriptors for future receive and transmit operations by the firmware popping the respective assigned queue and retrieving and populating descriptor prior to submitting the updated descriptor.

13.2.9.5.3.6 Teardown Queue

The Teardown Queue is used by the DMA to communicate a completion of a channel teardown after a channel teardown is invoked on to a channel. The pointer to the teardown descriptor is written to the teardown queue, which is also the Completion Queue, when the channel teardown completes.

13.2.9.5.3.7 Diverting Queue Packets from one Queue to Another

The host can move the entire contents of one queue to another queue by writing the source queue number and the destination queue number to the Queue Diversion Register. When diverting packets, the host can choose whether the source queue contents should be pushed onto the tail of the destination queue.

13.2.9.6 Memory Regions and Linking RAM

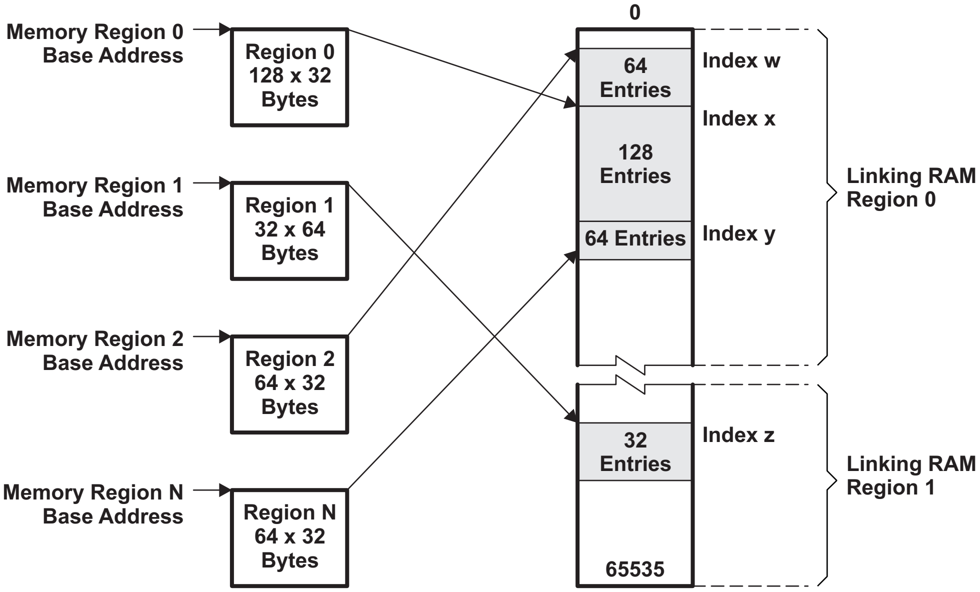
In addition to allocating memory for raw data, the host is responsible for allocating additional memory for exclusive use of the CPPI DMA as well as the Queue Manager. The Queue Manager has the capability of managing up to 16 Memory Regions. These Memory regions are used to allocate descriptors of variable sizes. The total number of descriptors that can be managed by the Queue Manager should not exceed 64K. Each memory region has descriptors of one configurable size. These 64K descriptors are referenced internally in the queue manager by a 16-bit quantity index.

The queue manager uses a linking RAM to store information (16-bit quantity index) about how the descriptors are logically connected to one another. A total of two Linking RAMs exists to be used by all Memory Regions. Each location in the linking RAM stores information for one descriptor index. The linking information for all descriptors in a given memory region is stored in a contiguous fashion in the linking RAM. The host, when it initializes the memory regions, also writes the index number corresponding to the first descriptor in a given region.

This information is used by the queue manager to compute where exactly in memory a particular descriptor is stored. The size of the linking RAM to be allotted by the host/firmware should be large enough to contain information for the amount of descriptor defined within the total Memory Regions. A total of 4 bytes of RAM is required for each descriptor. Figure 13-15 illustrates the relationship between memory regions and linking RAM.

NOTE: The reason for the existence of the two Linking RAMs is for the case when the user desires to allocate Linking RAMs in both internal and external memory. There is no restriction as to the placement of the Linking RAM.

Figure 13-15. Relationship Between Memory Regions and Linking RAM



13.2.9.7 Zero Length Packets

A special case is the handling of null packets with the CPPI 4.1 compliant DMA controller. Upon receiving a zero length USB packet, the XFER DMA will send a data block to the DMA controller with byte count of zero and the zero byte packet bit of INFO Word 2 set. The DMA controller will then perform normal End of Packet termination of the packet, without transferring data.

If a zero-length USB packet is received, the XDMA will send the CDMA a data block with a byte count of 0 and this bit set. The CDMA will then perform normal EOP termination of the packet without transferring data. For transmit, if a packet has this bit set, the XDMA will ignore the CPPI packet size and send a zero-length packet to the USB controller.

13.2.9.8 CPPI DMA Scheduler

The CPPI DMA scheduler is responsible for controlling the rate and order between the different Tx and Rx threads that are provided in the CPPI DMA controller. The scheduler table RAM exists within the scheduler.

13.2.9.8.1 CPPI DMA Scheduler Initialization

Before the scheduler can be used, the host is required to initialize and enable the block. This initialization is performed as follows:

1. The Host initializes entries within an internal memory array in the scheduler. This array contains up to 256 entries and each entry consists of a DMA channel number and a bit indicating if this is a Tx or Rx opportunity. These entries represent both the order and frequency that various Tx and Rx channels will be processed. A table size of 256 entries allows channel bandwidth to be allocated with a maximum precision of 1/256th of the total DMA bandwidth. The more entries that are present for a given channel, the bigger the slice of the bandwidth that channel will be given. Larger tables can be accommodated to allow for more precision. This array can only be written by the Host, it cannot be read.
2. If the application does not need to use the entire 256 entries, firmware can initialize the portion of the 256 entries and indicate the last entry used by writing to the LAST_ENTRY bits in the CDMA Scheduler Control Register 1 (DMA_SCHED_CTRL1) in the scheduler.
3. The host writes to the ENABLE bit in DMA_SCHED_CTRL1 to enable the scheduler. The scheduler is not required to be disabled in order to change the scheduler array contents.

13.2.9.8.2 Scheduler Operation

Once the scheduler is enabled it will begin processing the entries in the table and when appropriate passing credits to the DMA controller to perform a Tx or Rx operation. The operation of the DMA controller is as follows:

1. After the DMA scheduler is enabled it begins with the table index set to 0.
2. The scheduler reads the entry pointed to by the index and checks to see if the channel in question is currently in a state where a DMA operation can be accepted. The following must both be true:
   - The DMA channel must be enabled.
   - The CPPI FIFO that the channel talks to has free space on TX (FIFO full signal is not asserted) or a valid block on Rx (FIFO empty signal is not asserted).
3. If the DMA channel is capable of processing a credit to transfer a block, the DMA scheduler will issue that credit via the DMA scheduling interface. These are the steps:
   (a) The DMA controller may not be ready to accept the credit immediately and it may stall the scheduler until it can accept the credit. The DMA controller only accepts credits when it is in the IDLE state.
   (b) Once a credit has been accepted, the scheduler will increment the index to the next entry and will start at step 2.
4. If the channel in question is not currently capable of processing a credit, the scheduler will increment the index in the scheduler table to the next entry and will start at step 2.
5. When the scheduler attempts to increment its index to the value programmed in the table size register, the index will reset to 0.

13.2.9.9 CPPI DMA Transfer Interrupt Handling

The CPPI DMA 4.1 Interrupt handling mechanism does not go through the PDR Interrupt handler built into the core. The DMA interrupt line is directly routed to the Interrupt Dispatcher in a PDR compliant manner. The DMA interrupt is not maskable. The firmware needs to use queues not reserved by H/W as Completion Queues if require for DMA interrupt to be generated on a completion of a transfer.

Queues 24 and 25 are reserved by H/W for DMA transmit operations and queues 26 and 27 are reserved by H/W for DMA receive operations. If firmware uses these queues as completion queues, interrupt will be generated when the transfer completes. If need not to generate an interrupt, firmware is required to use queues that are not reserved as completion queues (queues 28 to 67).

13.2.9.10 DMA State Registers

The port must store and maintain state information for each transmit and receive port/channel. The state information is referred to as the Tx DMA State and Rx DMA State.

13.2.9.10.1 Transmit DMA State Registers

The Tx DMA State is a combination of control fields and protocol specific port scratchpad space used to manipulate data structures and transmit packets. Each transmit channel has two queues. Each queue has a one head descriptor pointer and one completion pointer. There are four Tx DMA State registers; one for each port/channel.

The following information is stored in the Tx DMA State:

- Tx Queue Head Descriptor Pointer(s)
- Tx Completion Pointer(s)
- Protocol specific control/status (port scratchpad)

13.2.9.10.2 Receive DMA State Registers

The Rx DMA State is a combination of control fields and protocol specific port scratchpad space used to manipulate data structures in order to receive packets. Each receive channel has only one queue. Each channel queue has one head descriptor pointer and one completion pointer. There are four Rx DMA State registers; one for each port/channel.

The following information is stored in the Rx DMA State:

- Rx Queue Head Descriptor Pointer
- Rx Queue Completion Pointer
- Rx Buffer Offset

13.2.9.11 USB DMA Protocols Supported

Four different type of DMA transfers are supported by the CPPI 4.1 DMA; Transparent, RNDIS, Generic RNDIS, and Linux CDC. The following sections will outline the details on these DMA transfer types.

13.2.9.11.1 Transparent DMA

Transparent Mode DMA operation is the default DMA mode where DMA interrupt is generated whenever a DMA packet is transferred. In the transparent mode, DMA packet size cannot be greater than USB MaxPktSize for the endpoint. This transfer type is ideal for transfer (not packet) sizes that are less than a max packet size.

Transparent DMA Transfer Setup

The following will configure all four ports/channels for Transparent DMA Transfer type.

- Make sure that RNDIS Mode is disabled globally. RNDIS bit in the control register (CTRLR) is cleared to 0.
- Configure the endpoint mode control fields in the DMA Mode Registers (MODE1 and MODE2) for Transparent Mode (RX*n*_MODE and TX*n*_MODE = 0h).

13.2.9.11.2 RNDIS

RNDIS mode DMA is used for large transfers (i.e., total data size to be transferred is greater than USB MaxPktSize where the MzxPktSize is a multiple of 64 bytes) that requires multiple USB packets. This is accomplished by breaking the larger packet into smaller packets, where each packet size being USB MaxPktSize except the last packet where its size is less than USB MaxPktSize, including zero bytes. This implies that multiple USB packets of MaxPktSize will be received and transferred together as a single large DMA transfer and the DMA interrupt is generated only at the end of the complete reception of DMA transfer. The protocol defines the end of the complete transfer by receiving a short USB packet (smaller than USB MaxPktSize as mentioned in USB specification 2.0). If the DMA packet size is an exact multiple of USB MaxPktSize, the DMA controller waits for a zero byte packet at the end of complete transfer to signify the completion of the transfer.

NOTE: RNDIS Mode DMA is supported only when USB MaxPktSize is an integral multiple of 64 bytes.

RNDIS DMA Transfer Setup

The following will configure all four ports/channels for RNDIS DMA Transfer type. If all endpoints are to be configured with the same RNDIS DMA transfer type, then you can enable for RNDIS mode support from the Control Register and the content of the Mode Register will be ignored.

If you need to enable RNDIS support globally.

- Enable RNDIS Mode globally. RNDIS bit in the control register (CTRLR) is set to 1.

If you need to enable RNDIS support at the port/channel (endpoint) level.

- Disable RNDIS Mode globally. RNDIS bit in the control register (CTRLR) is cleared to 0.
- Configure the endpoint mode control fields in the DMA Mode Registers (MODE1 and MODE2) for RNDIS Mode (RX*n*_MODE and TX*n*_MODE = 1h).

The above two setups yield the same result.

13.2.9.11.3 Generic RNDIS

Generic RNDIS DMA transfer mode is identical to the normal RNDIS mode in nearly all respects, except for the exception case where the last packet of the transfer can either be a short packet or the MaxPktSize. Generic RNDIS transfer makes use of a RNDIS EP Size register (there exists a register for each endpoint) that must be programmed with a value that is an integer multiple of the endpoint size for the DMA to know the end of the transfer when the last packet size is equal to the USB MaxPktSize. For example, it the Tx/RxMaxP is programmed with a value of 64, the Generic RNDIS EP Size register for that endpoint must be programmed with a value that is an integer multiple of 64 (for example, 64, 128, 192, 256, etc.).

In other words, when using Generic RNDIS mode and the DMA is tasked to transfer data transfer size that is less than a value programmed within the RNDIS EP Size register and this transfer will be resulting with a short packet, the DMA will terminate the transfer when encountering the short packet behaving exactly as the RNDIS DMA transfer type.

This means that Generic RNDIS mode will perform data transfer in the same manner as RNDIS mode, closing the CPPI packet when a USB packet is received that is less than the USB MaxPktSize size. Otherwise, the packet will be closed when the value in the Generic RNDIS EP Size register is reached.

Using RNDIS EP Size register, a packet of up to 64K bytes can be transferred. This is to allow the host software to program the USB module to transfer data that is an exact multiple of the USB MaxPktSize (Tx/RxMaxP programmed value) without having to send an additional short packet to terminate.

NOTE: As in RNDIS mode, the USB max packet size of any Generic RNDIS mode enabled endpoints must be a multiple of 64 bytes. Generic RNDIS acceleration should not be enabled for endpoints where the max packet size is not a multiple of 64 bytes. Only transparent mode should be used for such endpoints.

Generic RNDIS DMA Transfer Setup

The following will configure all four ports/channels for Generic RNDIS DMA Transfer type.

- Disable RNDIS Mode globally. RNDIS bit in the control register (CTRLR) is cleared to 0.
- Configure the endpoint mode control fields in the DMA Mode Registers (MODE1 and MODE2) for Generic RNDIS Mode (RX*n*_MODE and TX*n*_MODE = 3h).

13.2.9.11.4 Linux CDC

Linux CDC DMA transfer mode acts in the same manner as RNDIS packets, except for the case where the last data matches the max USB packet size. If the last data packet of a transfer is a short packet where the data size is greater than zero and less the USB MaxPktSize, then the behavior of the Linux CDC DMA transfer type is identical with the RNDIS DMA transfer type. The only exception is when the short packet length terminating the transfer is a Null Packet. In this case, instead of transferring the Null Packet, it will transfer a data packet of size 1 byte with the data value of 0h.

In transmit operation, if an endpoint is configured or CDC Linux mode, upon receiving a Null Packet from the CPPI DMA, the XFER DMA will then generate a packet containing 1 byte of data, whose value is 0h, indicating the end of the transfer. During receive operation, the XFER DMA will recognize the one byte zero packet as a termination of the data transfer, and sends a block of data with the EOP indicator set and a byte count of one to the CPPI DMA controller. The CPPI DMA realizing the end of the transfer termination will not update/increase the packet size count of the Host Packet Descriptor.

Linux CDC DMA Transfer Setup

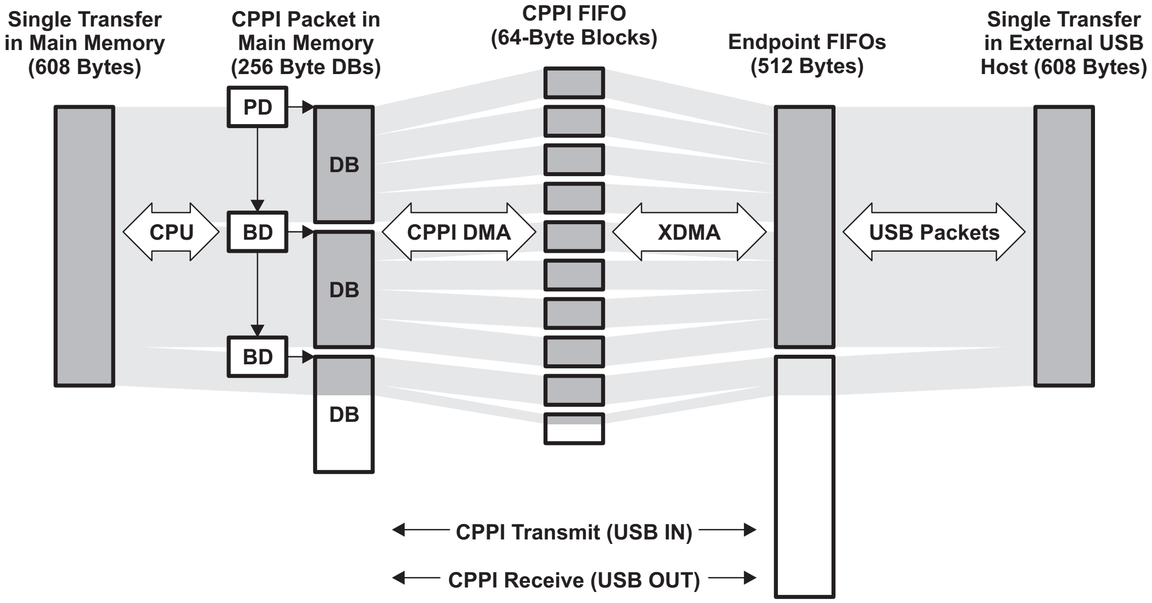
The following will configure all four ports/channels for Linux CDC DMA Transfer type.

- Disable RNDIS Mode globally. RNDIS bit in the control register (CTRLR) is cleared to 0.
- Configure the endpoint mode control fields in the DMA Mode Registers (MODE1 and MODE2) for Linux CDC Mode (RX*n*_MODE and TX*n*_MODE = 2h).

13.2.9.12 USB Data Flow Using DMA

The necessary steps required to perform a USB data transfer using the CPPI 4.1 DMA is expressed using an example for both transmit and receive cases. Assume a device is ready to perform a data transfer of size 608 bytes (see Figure 13-16).

Figure 13-16. High-Level Transmit and Receive Data Transfer Example


Single Transfer in Main Memory (608 Bytes)
CPPI Packet in Main Memory (256 Byte DBs)
CPPI FIFO (64-Byte Blocks)
Endpoint FIFOs (512 Bytes)
Single Transfer in External USB Host (608 Bytes)
PD
DB
CPU
BD
DB
BD
DB
CPPI DMA
XDMA
USB Packets
CPPI Transmit (USB IN)
CPPI Receive (USB OUT)

Example assumptions:

- The CPPI data buffers are 256 bytes in length.
- The USB endpoint 1 Tx and Rx endpoint 1 size are 512 bytes.
- A single transfer length is 608 bytes.
- The SOP offset is 0.

This translates to the following:

- Transmit Case:
  - 1 Host Packet Descriptor with Packet Length field of 608 bytes and a Data Buffer of size 256 Bytes linked to the 1st Host Buffer Descriptor.
  - First Host Buffer Descriptor with a Data Buffer size of 256 Bytes linked to the 2nd Buffer Descriptor.
  - Second Host Buffer Descriptor with a Data Buffer size of 96 bytes (can be greater, the Packet Descriptor contain the size of the packet) linked with its link word set to Null.
- Receive Case:
  - Two Host Buffer Descriptors with 256 bytes of Data Buffer Size
  - One Host Buffer Descriptor with 96 bytes (can be greater) of Data Buffer size

Within the rest of this section, the following nomenclature is used.

BD— Host Buffer Descriptor

DB— Data Buffer Size of 256 Bytes

PBD— Pointer to Host Buffer Descriptor

PD— Host Packet Descriptor

PPD— Pointer to Host Packet Descriptor

RXCQ— Receive Completion Queue or Receive Return Queue (for all Rx EPs, use 26 or 27)

RXSQ— Receive Free Packet/Buffer Descriptor Queue or Receive Submit Queue. (for all Rx EPs, use 0 to 15)

TXCQ— Transmit Completion Queue or Transmit Return Queue (for all Tx EPs, use 24 or 25)

TXSQ— Transmit Queue or Transmit Submit Queue (for EP1, use 16 or 17)

13.2.9.12.1 Transmit USB Data Flow Using DMA

The transmit descriptors and queue status configuration prior to the transfer taking place is shown in Figure 13-17. An example of initialization for a transmit USB data flow is shown in Figure 13-18.

Figure 13-17. Transmit Descriptors and Queue Status Configuration

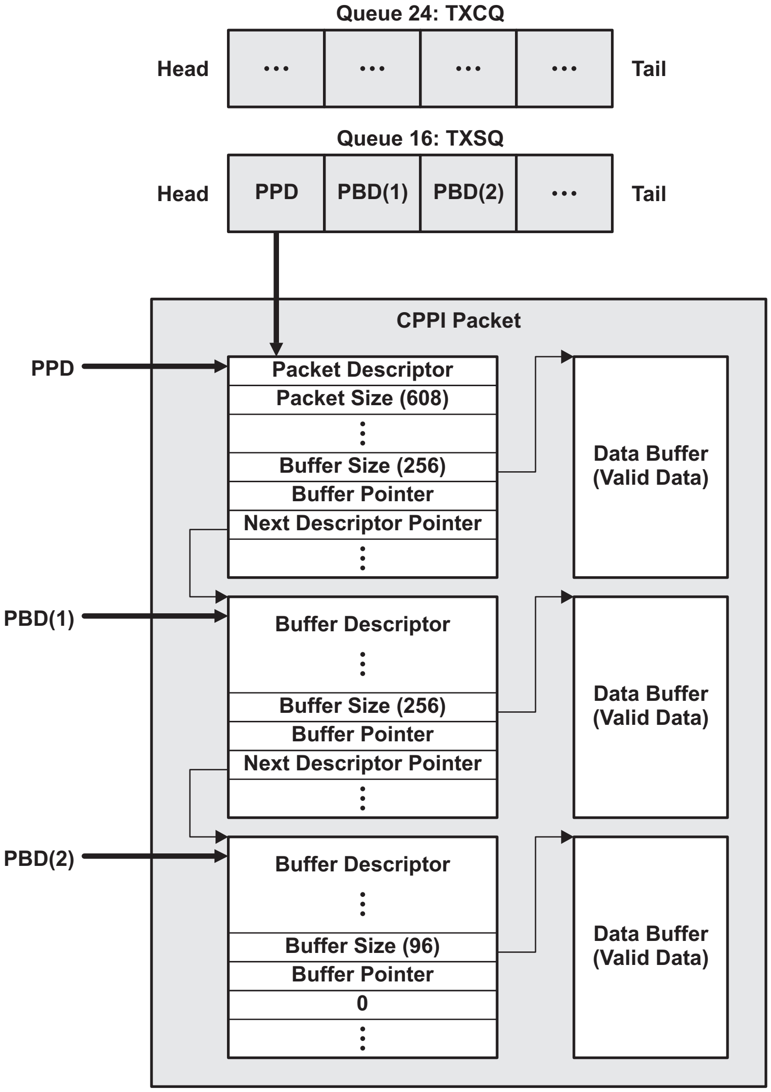
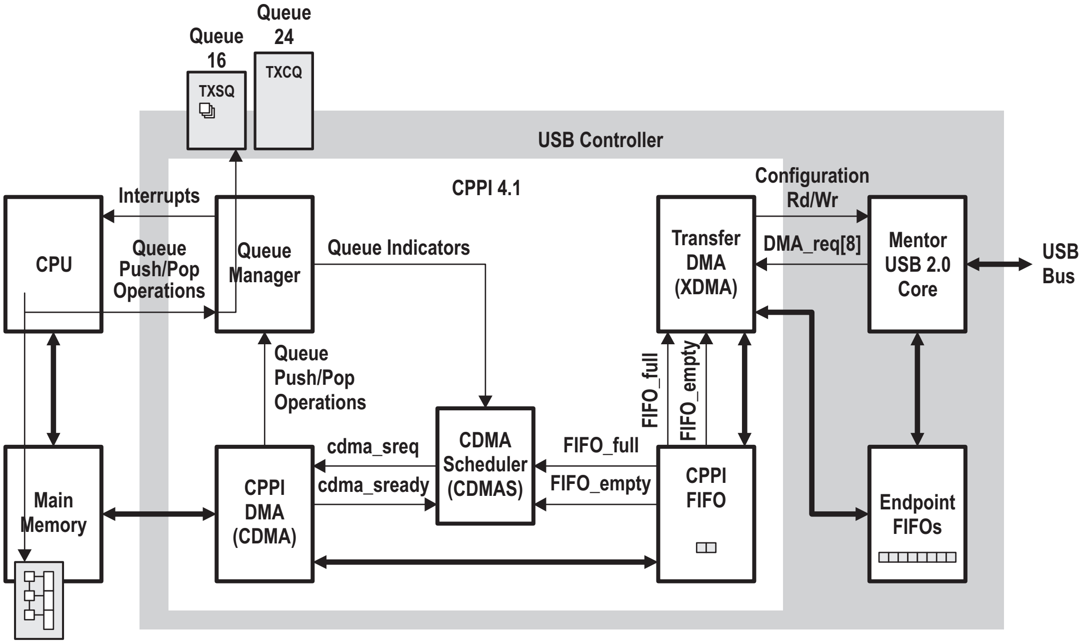


Figure 13-18. Transmit USB Data Flow Example (Initialization)



Step 1 (Initialization for Tx):

1. The CPU initializes Queue Manager with the Memory Region 0 base address and Memory Region 0 size, Link RAM0 Base address, Link RAM0 data size, and Link RAM1 Base address.
2. The CPU creates PD, BDs, and DBs in main memory and link as indicated in Figure 13-18.
3. It then initializes and configures the Queue Manager, Channel Setup, DMA Scheduler, and Mentor USB 2.0 Core.
4. It then adds (pushes) the PPD and the two PBDs to the TXSQ

NOTE: You can create more BD/DB pairs and push them on one of the unassigned queues. The firmware can pop a BD/DP pair from this chosen queue and can create its HPD or HBDs and pre link them prior to submitting the pointers to the HPD and HBD on to the TXSQ.

Step 2 (CDMA and XDMA transfers packet data into Endpoint FIFO for Tx):

1. The Queue Manager informs the CDMAS that the TXSQ is not empty.
2. CDMAS checks that the CPPI FIFO FIFO_full is not asserted, then issues a credit to the CDMA.
3. CDMA reads the packet descriptor pointer and descriptor size hint from the Queue Manager.
4. CMDA reads the packet descriptor from memory.
5. For each 64-byte block of data in the packet data payload:
   (a) The CDMA transfers a max burst of 64-byte block from the data to be transferred in main memory to the CPPI FIFO.
   (b) The XDMA sees FIFO_empty not asserted and transfers 64-byte block from CPPI FIFO to Endpoint FIFO.
   (c) The CDMA performs the above 2 steps 3 more times since the data size of the HPD is 256 bytes.
6. The CDMA reads the first buffer descriptor pointer.
7. CDMA reads the buffer descriptor from memory.

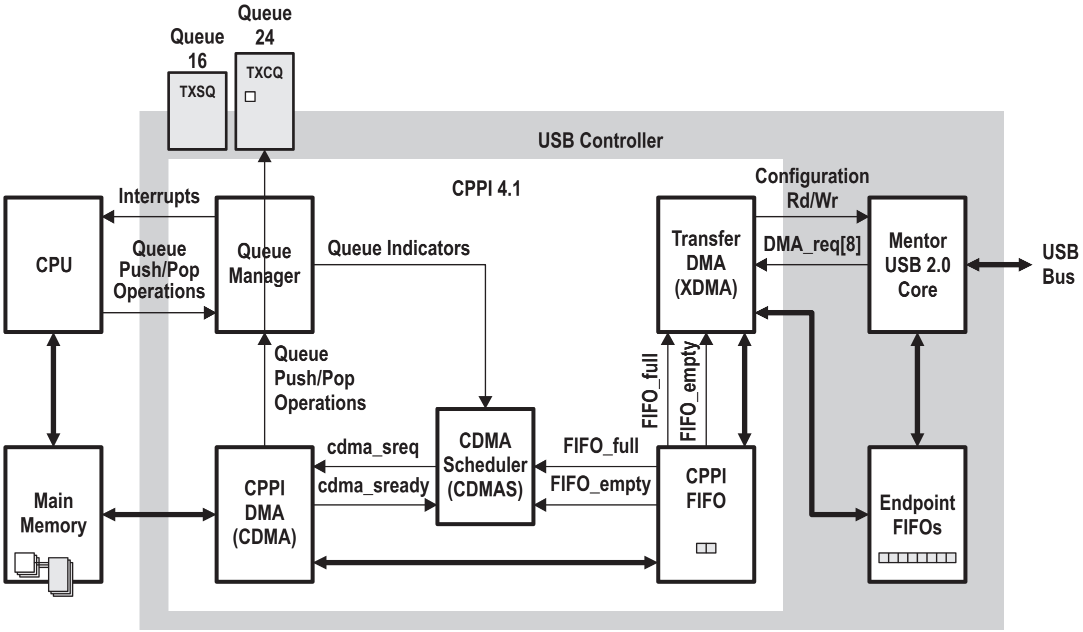
8. For each 64-byte block of data in the packet data payload:
   (a) The CDMA transfers a max burst of 64-byte block from the data to be transferred in main memory to the CPPI FIFO.
   (b) The XDMA sees FIFO_empty not asserted and transfers 64-byte block from CPPI FIFO to Endpoint FIFO.
   (c) The CDMA performs the above 2 steps 2 more times since data size of the HBD is 256 bytes.
9. The CDMA reads the second buffer descriptor pointer.
10. CDMA reads the buffer descriptor from memory.
11. For each 64-byte block of data in the packet data payload:
   (a) The CDMA transfers a max burst of 64-byte block from the data to be transferred in main memory to the CPPI FIFO.
   (b) The XDMA sees FIFO_empty not asserted and transfers 64-byte block from CPPI FIFO to Endpoint FIFO.
   (c) The CDMA transfers the last remaining 32-byte from the data to be transferred in main memory to the CPPI FIFO.
   (d) The XDMA sees FIFO_empty not asserted and transfers 32-byte block from CPPI FIFO to Endpoint FIFO.

Step 3 (Mentor USB 2.0 Core transmits USB packets for Tx):

1. Once the XDMA has transferred enough 64-byte blocks of data from the CPPI FIFO to fill the Endpoint FIFO, it signals the Mentor USB 2.0 Core that a TX packet is ready (sets the endpoint’s TxPktRdy bit).
2. The Mentor USB 2.0 Core will transmit the packet from the Endpoint FIFO out on the USB BUS when it receives a corresponding IN request from the attached USB Host.
3. After the USB packet is transferred, the Mentor USB 2.0 Core issues a TX DMA_req to the XDMA.
4. This process is repeated until the entire packet has been transmitted. The XDMA will also generate the required termination packet depending on the termination mode configured for the endpoint.

An example of the completion for a transmit USB data flow is shown in Figure 13-19.

Figure 13-19. Transmit USB Data Flow Example (Completion)



Step 4 (Return packet to completion queue and interrupt CPU for Tx):

1. After all data for the packet has been transmitted (as specified by the packet size field), the CDMA will write the pointer to the packet descriptor to the TX Completion Queue specified in the return queue manager / queue number fields of the packet descriptor.
2. The Queue Manager then indicates the status of the TXSQ (empty) to the CDMAS and the TXCQ to the CPU via an interrupt.

13.2.9.12.2 Receive USB Data Flow Using DMA

The receive descriptors and queue status configuration prior to the transfer taking place is shown in Figure 13-20. An example of initialization for a receive USB data flow is shown in Figure 13-21.

Figure 13-20. Receive Descriptors and Queue Status Configuration

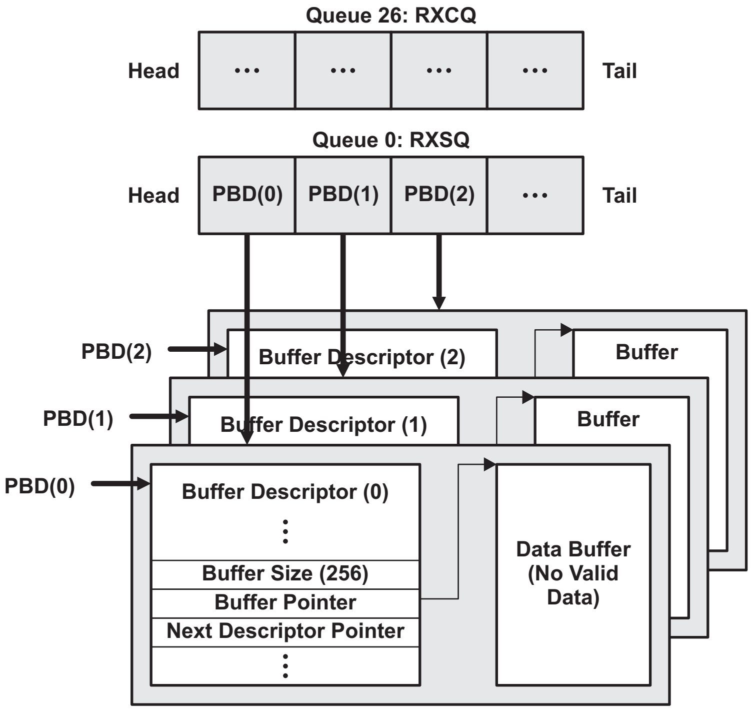
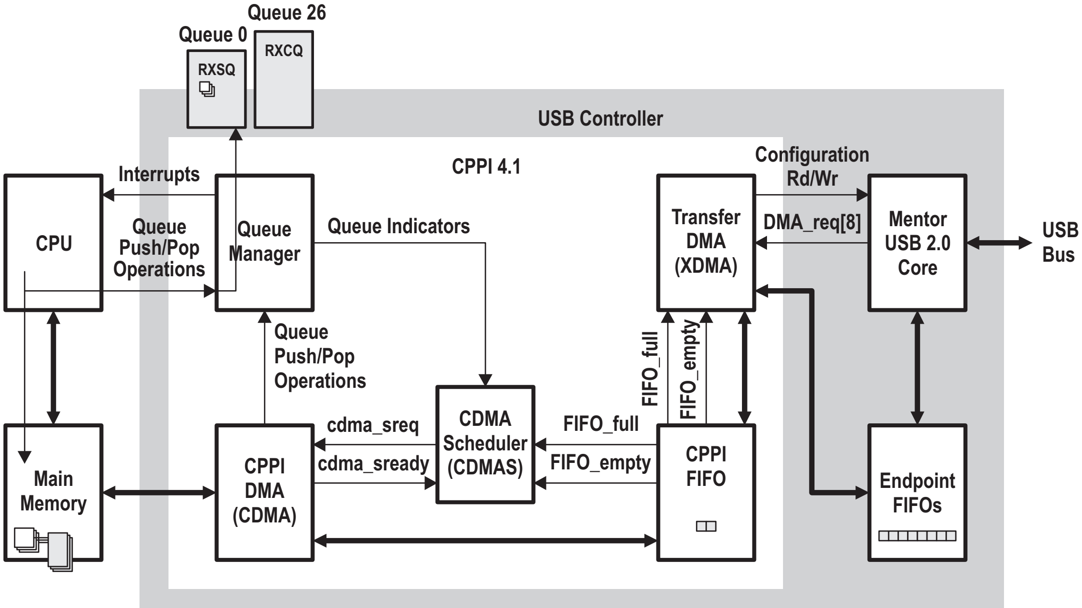


Figure 13-21. Receive USB Data Flow Example (Initialization)



Step 1 (Initialization for Rx):

1. The CPU initializes Queue Manager with the Memory Region 0 base address and Memory Region 0 size, Link RAM0 Base address, Link RAM0 data size, and Link RAM1 Base address.
2. The CPU creates BDs, and DBs in main memory and link them as indicated in Figure 13-21.
3. It then initializes the RXCQ queue and configures the Queue Manager, Channel Setup, DMA Scheduler, and Mentor USB 2.0 Core.
4. It then adds (pushes) the address of the three PHDs into the RXSQ.

Step 2 (Mentor USB 2.0 Core receives a packet, XDMA starts data transfer for Receive):

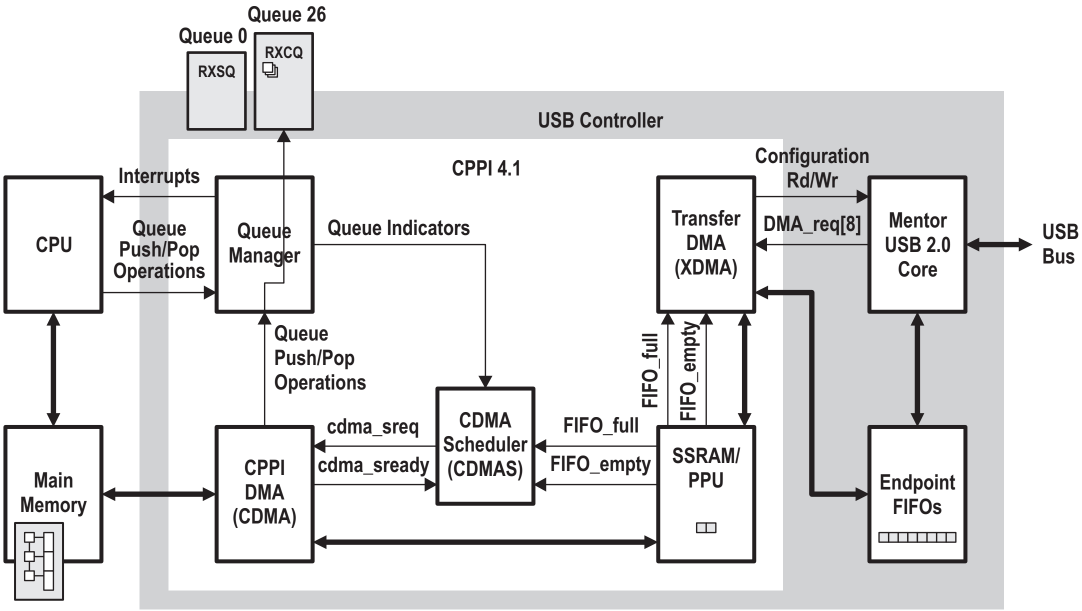
1. The Mentor USB 2.0 Core receives a USB packet from the USB Host and stores it in the Endpoint FIFO.
2. It then asserts a DMA_req to the XDMA informing it that data is available in the Endpoint FIFO.
3. The XDMA verifies the corresponding CPPI FIFO is not full via the FIFO_full signal, then starts transferring 64-byte data blocks from the Endpoint FIFO into the CPPI FIFO.

Step 3 (CDMA transfers data from SSRAM / PPU to main memory for Receive):

1. The CDMAS see FIFO_empty de-asserted (there is RX data in the FIFO) and issues a transaction credit to the CDMA.
2. The CDMA begins packet reception by fetching the first PBD from the Queue Manager using the Free Descriptor / Buffer Queue 0 (Rx Submit Queue) index that was initialized in the RX port DMA state for that channel.
3. The CDMA will then begin writing the 64-byte block of packet data into this DB.
4. The CDMA will continue filling the buffer with additional 64-byte blocks of data from the CPPI FIFO and will fetch additional PBD as needed using the Free Descriptor / Buffer Queue 1, 2, and 3 indexes for the 2nd, 3rd, and remaining buffers in the packet. After each buffer is filled, the CDMA writes the buffer descriptor to main memory.

An example of the completion for a receive USB data flow is shown in Figure 13-22 .

Figure 13-22. Receive USB Data Flow Example (Completion)



Step 4 (CDMA completes the packet transfer for Receive):

1. After the entire packet has been received, the CDMA writes the packet descriptor to main memory.
2. The CDMA then writes the packet descriptor to the RXCQ specified in the Queue Manager / Queue Number fields in the RX Global Configuration Register.
3. The Queue Manager then indicates the status of the RXCQ to the CPU via an interrupt.
4. The CPU can then process the received packet by popping the received packet information from the RXCQ and accessing the packet's data from main memory.

13.2.9.13 Interrupt Handling

Table 13-9 lists the interrupts generated by the USB controller.

Table 13-9. Interrupts Generated by the USB Controller

Interrupt	Description
Tx Endpoint [4-0]	Tx endpoint ready or error condition. For endpoints 4 to 0. (Rx and Tx for endpoint 0)
Rx Endpoint [4-1]	Rx endpoint ready or error condition. For endpoints 4 to 1. (Endpoint 0 has interrupt status in Tx interrupt)
USB Core[3-0]	Interrupts for 4 USB conditions
DMA Tx Completion [3-0]	Tx DMA completion interrupt for channel 3 to 0 using Queues 24 and 25
DMA Rx Completion [3-0]	Rx DMA completion interrupt for channel 3 to 0 using Queues 26 and 27

Whenever any of these interrupt conditions are generated, the host processor is interrupted. The software needs to read the different interrupt status registers (discussed in later section) to determine the source of the interrupt.

The USB interrupt conditions are listed in Table 13-10.

Table 13-10. USB Interrupt Conditions

Interrupt	Description
USB[3]	SOF started
USB[2]	Reset Signaling detected
USB[1]	Resume signaling detected
USB[0]	Suspend Signaling detected

13.2.9.13.1 USB Core Interrupts

Interrupt status can be determined using the INTSRCR (interrupt source) registers. These registers are non-masked. To clear the interrupt source, set the corresponding interrupt bit in INTCLRR registers. For debugging purposes, interrupt can be set manually through INTSETR registers.

The interrupt controller provides the option of masking the interrupts. A mask can be set using INTMSKSETR registers and can be cleared by setting the corresponding bit in the INTMSKCLRR registers. The mask can be read from INTMSKR registers. The masked interrupt status is determined using the INTMASKEDR registers.

The host processor software should write to the End Of Interrupt Register (EOIR) to acknowledge the completion of an interrupt.

NOTE: While EOIR is not written, the interrupt from the USB controller remains asserted.

13.2.10 BYTEMODE Bits of the USB System Control Register

The CPU cannot generate 8-bit accesses to its data or I/O space. This presents a problem given that some USB controller I/O registers are only 8 bits in width.

For these situations, the BYTEMODE bits of the USB system control register (USBSCR) can be used to program the DSP switched central resource (SCR) such that a CPU word access generates single byte access when reading or writing from USB controller I/O registers.

Table 13-11 summarizes the effect of the BYTEMODE bits for different CPU operations. For more details on USBSCR, please refer to Section 13.2.6.1.

Table 13-11. Effect of USBSCR BYTEMODE Bits on USB Access

BYTEMODE Setting	CPU Access to USB Register
BYTEMODE = 0h (16-bit word access)	Entire register contents are accessed.
BYTEMODE = 1h (8-bit access with high byte selected)	Only the upper byte of the register is accessed.
BYTEMODE = 2h (8-bit access with low byte selected)	Only the lower byte of the register is accessed.

13.2.11 Reset Considerations

The USB controller has two reset sources: hardware reset and the soft reset.

13.2.11.1 Software Reset Considerations

The USB controller can be reset by software through the RESET bit in the control register (CTRLR) or through the USB_RST bit in the peripheral reset control register (PCR).

When the RESET bit in the control register (CTRLR) is set, all the USB controller registers and DMA operations are reset. The bit is cleared automatically.

When USB_RST is set to 1, a hardware reset is forced on the USB controller. The effects of a hardware reset are described in the next section. Please note that the USB input clock must be enabled when using USB_RST (see Section 13.2.1).

13.2.11.2 Hardware Reset Considerations

A hardware reset is always initiated during a full chip reset. Alternatively, software can force an USB controller hardware reset through the USB_RST bits of the peripheral reset control register (PRCR). For more details on PRCR, see Section 1.1, System Control.

When a hardware reset is asserted, all the registers are set to their default values.

13.2.12 Interrupt Support

The USB controller is capable of interrupting the CPU. For more information on the mapping of interrupts, see Section 1.1, System Control.

13.2.13 DMA Event Support

The USB is an internal bus master peripheral and does not utilize system DMA events. The USB has its own dedicated DMA, CPPI 4.1 DMA, that it utilizes for DMA driven data transfer.

13.2.14 Power Management

The USB controller can be clock gated to conserve power during periods of no activity. The clock gating the peripheral is controlled by the CPU. For detailed information on power management procedures, see Section 1.1, System Control.

One of the main improvements of the device is that a dedicated LDO for USB has been integrated on the chip. This simplifies the power supply design on the system level. In order to use the internal USB LDO, the pin connections in Table 13-12 are required. The external LDO can also be used; the pin connections for using external LDO are included in Table 13-12 as well.

Table 13-12. LDO Pin Connections

Pin Number	Pin Name	Internal LDO	External LDO
A12	RSV0	L	L
B12	RSV3	L	L
C13	DSP_LDO_EN	L	H
B13	RSV16	L	L

Notes:

- *L* in the above table means voltage is at ground-level
- *H* in the above table means voltage is at the corresponding power supply level
- To use the internal USB LDO, the bit0 of register LDOCNTL (LDO Control Register) at 0x7004 need to be set as “1”
- Internal USB LDO is dedicated for USB module only; it should not be used for any other modules or external devices
- If an application requires the device to boot from an external USB device, then the external LDO must be used to power the USB module

13.3 Registers

13.3.1 USB Controller Register Summary

The following sections summarize the registers for the universal serial bus (USB) controller. Please note that the USB controller includes an USB2.0 mentor core and a communication part programming interface (CPPI) DMA, each with its own set of registers.

Due to the fact that the CDMA (CPPI DMA) is a native 32-bit module, and the device is a 16-bit DSP, it cannot do 32-bit read/write in one operation cycle. So, some cautions are needed when the device is accessing this module.

- When the device reads a 32-bit self-clean register, only the first read returns a valid 16-bit data. This is because when a reading action is applied to a 32-bit self-clean register; all 32 bit of the register is cleaned after the current read.
- When one needs to write a 32-bit data to a pair of 16-bit registers, which are split from a native 32-bit register, these two 16-bit write operations need to be consecutive and all the interrupts need to be disabled during these two write operations.

13.3.1.1 Universal Serial Bus (USB) Controller Registers

Table 13-13 lists the registers of the USB controller. Refer to the sections listed for detailed information on each register.

NOTE: Some USB controller registers are 8-bits wide. However, the CPU cannot generate 8-bit accesses to its data or I/O space. When accessing these registers, program the BYTEMODE bits of the USB system control register (USBSCR) to mask the upper or lower byte of of a word access. The BYTEMODE bits should be set to 00b (16-bit access) when accessing any other register. See Section 13.2.10 for more details on the BYTEMODE bits.

Table 13-13. Universal Serial Bus (USB) Registers

CPU Word Address	Acronym	Register Description	Section
8000h	REVID1	Revision Identification Register 1	Section 13.3.2
8001h	REVID2	Revision Identification Register 2	Section 13.3.2
8004h	CTRLR	Control Register	Section 13.3.3
8008h	STATR	Status Register	Section 13.3.4
800Ch	EMUR	Emulation Register	Section 13.3.5
8010h	MODE1	Mode Register 1	Section 13.3.6
8011h	MODE2	Mode Register 2	Section 13.3.6
8014h	AUTOREQ	Auto Request Register	Section 13.3.7
8018h	SRPFIXTIME1	SRP Fix Time Register 1	Section 13.3.8
8019h	SRPFIXTIME2	SRP Fix Time Register 2	Section 13.3.8
801Ch	TEARDOWN1	Teardown Register 1	Section 13.3.9
801Dh	TEARDOWN2	Teardown Register 2	Section 13.3.9
8020h	INTSRCR1	USB Interrupt Source Register 1	Section 13.3.10
8021h	INTSRCR2	USB Interrupt Source Register 2	Section 13.3.10
8024h	INTSETR1	USB Interrupt Source Set Register 1	Section 13.3.11
8025h	INTSETR2	USB Interrupt Source Set Register 2	Section 13.3.11
8028h	INTCLRR1	USB Interrupt Source Clear Register 1	Section 13.3.12
8029h	INTCLRR2	USB Interrupt Source Clear Register 2	Section 13.3.12
802Ch	INTMSKR1	USB Interrupt Mask Register 1	Section 13.3.13
802Dh	INTMSKR2	USB Interrupt Mask Register 2	Section 13.3.13
8030h	INTMSKSETR1	USB Interrupt Mask Set Register 1	Section 13.3.14

Table 13-13. Universal Serial Bus (USB) Registers (continued)

CPU Word Address	Acronym	Register Description	Section
8031h	INTMSKSETR2	USB Interrupt Mask Set Register 2	Section 13.3.14
8034h	INTMSKCLRR1	USB Interrupt Mask Clear Register 1	Section 13.3.15
8035h	INTMSKCLRR2	USB Interrupt Mask Clear Register 2	Section 13.3.15
8038h	INTMASKEDR1	USB Interrupt Source Masked Register 1	Section 13.3.16
8039h	INTMASKEDR2	USB Interrupt Source Masked Register 2	Section 13.3.16
803Ch	EOIR	USB End of Interrupt Register	Section 13.3.17
8040h	INTVECTR1	USB Interrupt Vector Register 1	Section 13.3.18
8041h	INTVECTR2	USB Interrupt Vector Register 2	Section 13.3.18
8050h	GREP1SZR1	Generic RNDIS EP1Size Register 1	Section 13.3.19
8051h	GREP1SZR2	Generic RNDIS EP1Size Register 2	Section 13.3.19
8054h	GREP2SZR1	Generic RNDIS EP2 Size Register 1	Section 13.3.20
8055h	GREP2SZR2	Generic RNDIS EP2 Size Register 2	Section 13.3.20
8058h	GREP3SZR1	Generic RNDIS EP3 Size Register 1	Section 13.3.21
8059h	GREP3SZR2	Generic RNDIS EP3 Size Register 2	Section 13.3.21
805Ch	GREP4SZR1	Generic RNDIS EP4 Size Register 1	Section 13.3.22
805Dh	GREP4SZR2	Generic RNDIS EP4 Size Register 2	Section 13.3.22

13.3.1.2 Mentor USB2.0 Core Registers

This section lists the registers of the Mentor USB2.0 core integrated in the USB controller.

NOTE: Some USB controller registers are 8-bits wide. However, the CPU cannot generate 8-bit accesses to its data or I/O space. When accessing these registers, program the BYTEMODE bits of the USB system control register (USBSCR) to mask the upper or lower byte of of a word access. The BYTEMODE bits should be set to 00b (16-bit access) when accessing any other register. See Section 13.2.10 for more details on the BYTEMODE bits.

13.3.1.2.1 Common USB Registers

Table 13-15 lists the common USB registers. Some common USB registers are 8-bit wide and share a word address with other 8-bit registers. Table 13-14 describes how the common USB registers are laid out in memory.

Table 13-14. Common USB Register Layout

<table>
<tr><th rowspan="2">CPU Word Address</th><th colspan="2">Register</th></tr>
<tr><th>Byte 1</th><th>Byte 0</th></tr>
<tr><td>8401h</td><td>POWER</td><td>FADDR</td></tr>
<tr><td>8402h</td><td colspan="2">INTRTX</td></tr>
<tr><td>8405h</td><td colspan="2">INTRRX</td></tr>
<tr><td>8406h</td><td colspan="2">INTRTXE</td></tr>
<tr><td>8409h</td><td colspan="2">INTRRXE</td></tr>
<tr><td>840Ah</td><td>INTRUSBE</td><td>INTRUSB</td></tr>
<tr><td>840Dh</td><td colspan="2">FRAME</td></tr>
<tr><td>840Eh</td><td>TESTMODE</td><td>INDEX</td></tr>
</table>

Table 13-15. Common USB Registers

CPU Word Address	Acronym	Register Description	Section
8401h	FADDR_POWER	Function Address Register, Power Management Register	Section 13.3.23
8402h	INTRTX	Interrupt Register for Endpoint 0 plus Transmit Endpoints 1 to 4	Section 13.3.25
8405h	INTRRX	Interrupt Register for Receive Endpoints 1 to 4	Section 13.3.26
8406h	INTRTXE	Interrupt enable register for INTRTX	Section 13.3.27
8409h	INTRRXE	Interrupt Enable Register for INTRRX	Section 13.3.28
840Ah	INTRUSB_INTRUSBE	Interrupt Register for Common USB Interrupts, Interrupt Enable Register	Section 13.3.29
840Dh	FRAME	Frame Number Register	Section 13.3.31
840Eh	INDEX_TESTMODE	Index Register for Selecting the Endpoint Status and Control Registers, Register to Enable the USB 2.0 Test Modes	Section 13.3.32

13.3.1.2.2 Indexed Registers

Table 13-18 lists the index registers. These registers operate on the endpoint selected by the index register. (The index register is the low-8 bits of the INDEX_TESTMODE 16 bits register). Table 13-16 describes how the indexed USB registers are laid out in memory when endpoint 0 is selected in the index register (INDEX = 0). Similarly, Table 13-17 shows the layout of the indexed registers when endpoints 1-4 are selected in the index register (INDEX = 1 or 2 or 3 or 4).

Table 13-16. USB Indexed Register Layout when Index Register Set to Select Endpoint 0

<table>
<tr><th rowspan="2">CPU Word Address</th><th colspan="2">Register</th></tr>
<tr><th>Byte 1</th><th>Byte 0</th></tr>
<tr><td>8411h</td><td colspan="2">Reserved</td></tr>
<tr><td>8412h</td><td colspan="2">PERI_CSR0</td></tr>
<tr><td>8415h</td><td colspan="2">Reserved</td></tr>
<tr><td>8416h</td><td colspan="2">Reserved</td></tr>
<tr><td>8419h</td><td colspan="2">COUNT0</td></tr>
<tr><td>841Ah</td><td colspan="2">Reserved</td></tr>
<tr><td>841Dh</td><td colspan="2">Reserved</td></tr>
<tr><td>841Eh</td><td>CONFIGDATA_INDX</td><td>Reserved</td></tr>
</table>

Table 13-17. USB Indexed Register Layout when Index Register Set to Select Endpoint 1-4

<table>
<tr><th rowspan="2">CPU Word Address</th><th colspan="2">Register</th></tr>
<tr><th>Byte 1</th><th>Byte 0</th></tr>
<tr><td>8411h</td><td colspan="2">TXMAXP</td></tr>
<tr><td>8412h</td><td colspan="2">PERI_TXCSR</td></tr>
<tr><td>8415h</td><td colspan="2">RXMAXP</td></tr>
<tr><td>8416h</td><td colspan="2">PERI_RXCSR</td></tr>
<tr><td>8419h</td><td colspan="2">RXCOUNY</td></tr>
<tr><td>841Ah</td><td colspan="2">Reserved</td></tr>
<tr><td>841Dh</td><td colspan="2">Reserved</td></tr>
<tr><td>841Eh</td><td colspan="2">Reserved</td></tr>
</table>

Table 13-18. USB Indexed Registers

CPU Word Address	Acronym	Register Description	Section
8411h	TXMAXP_MAP	Maximum Packet Size for Peripheral/Host Transmit Endpoint. (Index register set to select Endpoints 1-4)	Section 13.3.34

Table 13-18. USB Indexed Registers (continued)

CPU Word Address	Acronym	Register Description	Section
8412h	PERI_CSR0	Control Status Register for Peripheral Endpoint 0. (Index register set to select Endpoint 0)	Section 13.3.35
	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint. (Index register set to select Endpoints 1-4)	Section 13.3.36
8415h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint. (Index register set to select Endpoints 1-4)	Section 13.3.37
8416h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint. (Index register set to select Endpoints 1-4)	Section 13.3.38
8419h	COUNT0	Number of Received Bytes in Endpoint 0 FIFO. (Index register set to select Endpoint 0)	Section 13.3.39
	RXCOUNT	Number of Bytes in Host Receive Endpoint FIFO. (Index register set to select Endpoints 1- 4)	Section 13.3.40
841Ah	-	Reserved	
841Dh	-	Reserved	
841Eh	CONFIGDATA_INDC (Upper byte of 841Dh)	Returns details of core configuration. (index register set to select Endpoint 0)	Section 13.3.41

13.3.1.2.3 FIFO Registers

Table 13-19 lists the FIFO registers of the USB2.0 Mentor core.

Table 13-19. USB FIFO Registers

CPU Word Address	Acronym	Register Description	Section
8421h	FIFO0R1	Transmit and Receive FIFO Register 1 for Endpoint 0	Section 13.3.42
8422h	FIFO0R2	Transmit and Receive FIFO Register 2 for Endpoint 0	Section 13.3.42
8425h	FIFO1R1	Transmit and Receive FIFO Register 1 for Endpoint 1	Section 13.3.43
8426h	FIFO1R2	Transmit and Receive FIFO Register 2 for Endpoint 1	Section 13.3.43
8429h	FIFO2R1	Transmit and Receive FIFO Register 1 for Endpoint 2	Section 13.3.44
842Ah	FIFO2R2	Transmit and Receive FIFO Register 2 for Endpoint 2	Section 13.3.44
842Dh	FIFO3R1	Transmit and Receive FIFO Register 1 for Endpoint 3	Section 13.3.45
842Eh	FIFO3R2	Transmit and Receive FIFO Register 2 for Endpoint 3	Section 13.3.45
8431h	FIFO4R1	Transmit and Receive FIFO Register 1 for Endpoint 4	Section 13.3.46
8432h	FIFO4R2	Transmit and Receive FIFO Register 2 for Endpoint 4	Section 13.3.46

13.3.1.2.4 Dynamic FIFO Control Registers

Table 13-21 lists the dynamic FIFO control registers of the US2.0 Mentor core. Some common USB registers are 8-bit wide and share a word address with other 8-bit registers. Table 13-20 describes how the common USB registers are laid out in memory.

Table 13-20. Dynamic FIFO Control Register Layout

<table>
<tr><th rowspan="2">CPU Word Address</th><th colspan="2">Register</th></tr>
<tr><th>Byte 1</th><th>Byte 0</th></tr>
<tr><td>8461h</td><td colspan="2">Reserved</td></tr>
<tr><td>8462h</td><td>RXFIFOSZ</td><td>TXFIFOSZ</td></tr>
<tr><td>8465h</td><td colspan="2">TXFIFOADDR</td></tr>
<tr><td>8466h</td><td colspan="2">RXFIFOADDR</td></tr>
<tr><td>846Dh</td><td colspan="2">HWVERS</td></tr>
</table>

Table 13-21. Dynamic FIFO Control Registers

CPU Word Address	Acronym	Register Description	Section
8461h	-	Reserved	
8462h	TXFIFOSZ_RXFIFOSZ	Transmit Endpoint FIFO Size, Receive Endpoint FIFO Size (Index register set to select Endpoints 1-4)	Section 13.3.48
8465h	TXFIFOADDR	Transmit Endpoint FIFO Address (Index register set to select Endpoints 1-4)	Section 13.3.50
8466h	RXFIFOADDR	Receive Endpoint FIFO Address (Index register set to select Endpoints 1-4)	Section 13.3.52
846Dh	HWVERS	Hardware Version Register	Section 13.3.51

13.3.1.2.5 Control and Status Registers for Endpoints 0-4

Table 13-22 lists the control and status registers for endpoints 0-4 of the USB2.0 Mentor Core.

Table 13-22. Control and Status Registers for Endpoints 0-4

CPU Word Address	Acronym	Register Description	Section
		Control and Status Register for Endpoint 1	
8511h	TXMAXP	Maximum Packet Size for Peripheral/Host Transmit Endpoint	Section 13.3.34
8512h	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint (peripheral mode)	Section 13.3.36
8515h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint	Section 13.3.37
8516h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint (peripheral mode)	Section 13.3.38
8519h	RXCOUNT	Number of Bytes in Host Receive endpoint FIFO	Section 13.3.40
851Ah	-	Reserved	
851Dh	-	Reserved	
851Eh	-	Reserved	
		Control and Status Register for Endpoint 2	
8521h	TXMAXP	Maximum Packet Size for Peripheral/Host Transmit Endpoint	Section 13.3.34
8522h	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint (peripheral mode)	Section 13.3.36
8525h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint	Section 13.3.37
8526h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint (peripheral mode)	Section 13.3.38
8529h	RXCOUNT	Number of Bytes in Host Receive endpoint FIFO	Section 13.3.40
852Ah	-	Reserved	
852Dh	-	Reserved	
852Eh	-	Reserved	
		Control and Status Register for Endpoint 3	
8531h	TXMAXP	Maximum Packet Size for Peripheral/Host Transmit Endpoint	Section 13.3.34
8532h	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint (peripheral mode)	Section 13.3.36
8535h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint	Section 13.3.37
8536h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint (peripheral mode)	Section 13.3.38
8539h	RXCOUNT	Number of Bytes in Host Receive endpoint FIFO	Section 13.3.40
853Ah	-	Reserved	
853Dh	-	Reserved	
853Eh	-	Reserved	
		Control and Status Register for Endpoint 4	

Table 13-22. Control and Status Registers for Endpoints 0-4 (continued)

CPU Word Address	Acronym	Register Description	Section
8541h	TXMAXP	Maximum Packet Size for Peripheral/Host Transmit Endpoint	Section 13.3.34
8542h	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint (peripheral mode)	Section 13.3.36
8545h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint	Section 13.3.37
8546h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint (peripheral mode)	Section 13.3.38
8549h	RXCOUNT	Number of Bytes in Host Receive endpoint FIFO	Section 13.3.40
854Ah	-	Reserved	
854Dh	-	Reserved	
854Eh	-	Reserved	

13.3.1.3 Communications Port Programming Interface (CPPI) 4.1 DMA Registers

This section lists the registers of the communications port programming interface (CPPI) DMA. Refer to the sections listed for detailed information on each register.

13.3.1.3.1 CPPI DMA (CMDA) Registers

Table 13-23 lists the register of the CPPI DMA (CMDA).

Table 13-23. CPPI DMA (CMDA) Registers

CPU Word Address	Acronym	Register Description	Section
9000h	DMAREVID1	CDMA Revision Identification Register 1	Section 13.3.53
9001h	DMAREVID2	CDMA Revision Identification Register 2	Section 13.3.53
9004h	TDFDQ	CDMA Teardown Free Descriptor Queue Control Register	Section 13.3.54
9008h	DMAEMU	CDMA Emulation Control Register	Section 13.3.55
9800h	TXGCR1[0]	Transmit Channel 0 Global Configuration Register 1	Section 13.3.56
9801h	TXGCR2[0]	Transmit Channel 0 Global Configuration Register 2	Section 13.3.56
9808h	RXGCR1[0]	Receive Channel 0 Global Configuration Register 1	Section 13.3.57
9809h	RXGCR2[0]	Receive Channel 0 Global Configuration Register 2	Section 13.3.57
980Ch	RXHPCR1A[0]	Receive Channel 0 Host Packet Configuration Register 1 A	Section 13.3.58
980Dh	RXHPCR2A[0]	Receive Channel 0 Host Packet Configuration Register 2 A	Section 13.3.58
9810h	RXHPCR1B[0]	Receive Channel 0 Host Packet Configuration Register 1 B	Section 13.3.59
9811h	RXHPCR2B[0]	Receive Channel 0 Host Packet Configuration Register 2 B	Section 13.3.59
9820h	TXGCR1[1]	Transmit Channel 1 Global Configuration Register 1	Section 13.3.56
9821h	TXGCR2[1]	Transmit Channel 1 Global Configuration Register 2	Section 13.3.56
9828h	RXGCR1[1]	Receive Channel 1 Global Configuration Register 1	Section 13.3.57
9829h	RXGCR2[1]	Receive Channel 1 Global Configuration Register 2	Section 13.3.57
982Ch	RXHPCR1A[1]	Receive Channel 1 Host Packet Configuration Register 1 A	Section 13.3.58
982Dh	RXHPCR2A[1]	Receive Channel 1 Host Packet Configuration Register 2 A	Section 13.3.58
9830h	RXHPCR1B[1]	Receive Channel 1 Host Packet Configuration Register 1 B	Section 13.3.59
9831h	RXHPCR2B[1]	Receive Channel 1 Host Packet Configuration Register 2 B	Section 13.3.59
9840h	TXGCR1[2]	Transmit Channel 2 Global Configuration Register 1	Section 13.3.56
9841h	TXGCR2[2]	Transmit Channel 2 Global Configuration Register 2	Section 13.3.56
9848h	RXGCR1[2]	Receive Channel 2 Global Configuration Register 1	Section 13.3.57
9849h	RXGCR2[2]	Receive Channel 2 Global Configuration Register 2	Section 13.3.57
984Ch	RXHPCR1A[2]	Receive Channel 2 Host Packet Configuration Register 1 A	Section 13.3.58
984Dh	RXHPCR2A[2]	Receive Channel 2 Host Packet Configuration Register 2 A	Section 13.3.58

Table 13-23. CPPI DMA (CMDA) Registers (continued)

CPU Word Address	Acronym	Register Description	Section
9850h	RXHPCR1B[2]	Receive Channel 2 Host Packet Configuration Register 1 B	Section 13.3.59
9851h	RXHPCR2B[2]	Receive Channel 2 Host Packet Configuration Register 2 B	Section 13.3.59
9860h	TXGCR1[3]	Transmit Channel 3 Global Configuration Register 1	Section 13.3.56
9861h	TXGCR2[3]	Transmit Channel 3 Global Configuration Register 2	Section 13.3.56
9868h	RXGCR1[3]	Receive Channel 3 Global Configuration Register 1	Section 13.3.57
9869h	RXGCR2[3]	Receive Channel 3 Global Configuration Register 2	Section 13.3.57
986Ch	RXHPCR1A[3]	Receive Channel 3 Host Packet Configuration Register 1 A	Section 13.3.58
986Dh	RXHPCR2A[3]	Receive Channel 3 Host Packet Configuration Register 2 A	Section 13.3.58
9870h	RXHPCR1B[3]	Receive Channel 3 Host Packet Configuration Register 1 B	Section 13.3.59
9871h	RXHPCR2B[3]	Receive Channel 3 Host Packet Configuration Register 2 B	Section 13.3.59
A000h	DMA_SCHED_CTRL1	CDMA Scheduler Control Register 1	Section 13.3.60
A001h	DMA_SCHED_CTRL2	CDMA Scheduler Control Register 1	Section 13.3.60
A800h + 4 × *N*	ENTRYLSW[*N*]	CDMA Scheduler Table Word *N* Registers LSW (*N* = 0 to 63)	Section 13.3.61
A801h + 4 × *N*	ENTRYMSW[*N*]	CDMA Scheduler Table Word *N* Registers MSW (*N* = 0 to 63)	Section 13.3.61

13.3.1.3.2 Queue Manager (QMGR) Registers

Table 13-24 lists the registers of the queue manager.

Table 13-24. Queue Manager (QMGR) Registers

CPU Word Address	Acronym	Register Description	Section
C000h	QMGRREVID1	Queue Manager Revision Identification Register 1	Section 13.3.62
C001h	QMGRREVID2	Queue Manager Revision Identification Register 2	Section 13.3.62
C008h	DIVERSION1	Queue Manager Queue Diversion Register 1	Section 13.3.63
C009h	DIVERSION2	Queue Manager Queue Diversion Register 2	Section 13.3.63
C020h	FDBSC0	Queue Manager Free Descriptor/Buffer Starvation Count Register 0	Section 13.3.64
C021h	FDBSC1	Queue Manager Free Descriptor/Buffer Starvation Count Register 1	Section 13.3.65
C024h	FDBSC2	Queue Manager Free Descriptor/Buffer Starvation Count Register 2	Section 13.3.66
C025h	FDBSC3	Queue Manager Free Descriptor/Buffer Starvation Count Register 3	Section 13.3.67
C028h	FDBSC4	Queue Manager Free Descriptor/Buffer Starvation Count Register 4	Section 13.3.68
C029h	FDBSC5	Queue Manager Free Descriptor/Buffer Starvation Count Register 5	Section 13.3.69
C02Ch	FDBSC6	Queue Manager Free Descriptor/Buffer Starvation Count Register 6	Section 13.3.70
C02Dh	FDBSC7	Queue Manager Free Descriptor/Buffer Starvation Count Register 7	Section 13.3.71
C080h	LRAM0BASE1	Queue Manager Linking RAM Region 0 Base Address Register 1	Section 13.3.72
C081h	LRAM0BASE2	Queue Manager Linking RAM Region 0 Base Address Register 2	Section 13.3.72
C084h	LRAM0SIZE	Queue Manager Linking RAM Region 0 Size Register	Section 13.3.73
C085h	-	Reserved	
C088h	LRAM1BASE1	Queue Manager Linking RAM Region 1 Base Address Register 1	Section 13.3.74
C089h	LRAM1BASE2	Queue Manager Linking RAM Region 1 Base Address Register 2	Section 13.3.74
C090h	PEND0	Queue Manager Queue Pending 0	Section 13.3.75
C091h	PEND1	Queue Manager Queue Pending 1	Section 13.3.76
C094h	PEND2	Queue Manager Queue Pending 2	Section 13.3.77
C095h	PEND3	Queue Manager Queue Pending 3	Section 13.3.78
C098h	PEND4	Queue Manager Queue Pending 4	Section 13.3.79
C099h	PEND5	Queue Manager Queue Pending 5	Section 13.3.80

Table 13-24. Queue Manager (QMGR) Registers (continued)

CPU Word Address	Acronym	Register Description	Section
D000h + 16 × *R*	QMEMRBASE1[*R*]	Queue Manager Memory Region *R* Base Address Register 1 (*R* = 0 to 15)	Section 13.3.81
D001h + 16 × *R*	QMEMRBASE2[*R*]	Queue Manager Memory Region *R* Base Address Register 2 (*R* = 0 to 15)	Section 13.3.81
D004h + 16 × *R*	QMEMRCTRL1[*R*]	Queue Manager Memory Region *R* Control Register (*R* = 0 to 15)	Section 13.3.82
D005h + 16 × *R*	QMEMRCTRL2[*R*]	Queue Manager Memory Region *R* Control Register (*R* = 0 to 15)	Section 13.3.82
E00Ch + 16 × *N*	CTRL1D	Queue Manager Queue *N* Control Register 1 D (*N* = 0 to 63)	Section 13.3.83
E00Dh + 16 × *N*	CTRL2D	Queue Manager Queue *N* Control Register 2 D (*N* = 0 to 63)	Section 13.3.83
E800h + 16 × *N*	QSTATA	Queue Manager Queue *N* Status Register A (*N* = 0 to 63)	Section 13.3.84
E804h + 16 × *N*	QSTAT1B	Queue Manager Queue *N* Status Register 1 B (*N* = 0 to 63)	Section 13.3.85
E805h + 16 × *N*	QSTAT2B	Queue Manager Queue *N* Status Register 2 B (*N* = 0 to 63)	Section 13.3.85
E808h + 16 × *N*	QSTATC	Queue Manager Queue *N* Status Register C (*N* = 0 to 63)	Section 13.3.86

13.3.2 *Revision Identification Registers (REVID1 and REVID2)*

The revision identification registers (REVID1 and REVID2) contain the revision for the USB 2.0 controller module. The REVID1 is shown in Figure 13-23 and described in Table 13-25. The REVID2 is shown in Figure 13-24 and described in Table 13-26.

Figure 13-23. Revision Identification Register 1 (REVID1)

15	0
REVLSB	
R- 0800h	

LEGEND: R = Read only; -*n* = value after reset

Figure 13-24. Revision Identification Register 2 (REVID2)

15	0
REVMSB	
R-4EA1h	

LEGEND: R = Read only; -*n* = value after reset

Table 13-25. Revision Identification Register 1 (REVID1) Field Descriptions

Bit	Field	Value	Description
15-0	REVLSB	0-FFFFh	Least significant bits of the revision ID of the USB module.

Table 13-26. Revision Identification Register 2 (REVID2) Field Descriptions

Bit	Field	Value	Description
15-0	REVMSB	0-FFFFh	Most significant bits of the revision ID of the USB module.

13.3.3 Control Register (CTRLR)

The control register (CTRLR) allows the CPU to control various aspects of the module. The CTRLR is shown in Figure 13-25 and described in Table 13-27.

Figure 13-25. Control Register (CTRLR)

15 5	4	3	2	1	0
Reserved	RNDIS	UINT	Reserved	CLKFACK	RESET
R-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-27. Control Register (CTRLR) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved
4	RNDIS		Global RNDIS mode enable for all endpoints.
		0	Global RNDIS mode is disabled.
		1	Global RNDIS mode is enabled.
3	UINT		USB non-PDR interrupt handler enable.
		0	PDR interrupt handler is enabled.
		1	PDR interrupt handler is disabled.
2	Reserved	0	Reserved
1	CLKFACK		Clock stop fast ACK enable.
		0	Clock stop fast ACK is disabled.
		1	Clock stop fast ACK is enabled.
0	RESET		Soft reset.
		0	No effect.
		1	Writing a 1 starts a module reset.

13.3.4 Status Register (STATR)

The status register (STATR) allows the CPU to check various aspects of the module. The STATR is shown in Figure 13-26 and described in Table 13-28.

Figure 13-26. Status Register (STATR)

15 1	0
Reserved	DRVVBUS
R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 13-28. Status Register (STATR) Field Descriptions

Bit	Field	Value	Description
15-1	Reserved	0	Reserved.
0	DRVVBUS		Current DRVVBUS value.
		0	DRVVBUS value is logic 0.
		1	DRVVBUS value is logic 1.

13.3.5 Emulation Register (EMUR)

The emulation register (EMUR) allows the CPU to configure the CBA 3.0 emulation interface. The EMUR is shown in Figure 13-27 and described in Table 13-29.

Figure 13-27. Emulation Register (EMUR)

15 3	2	1	0
Reserved	RT_SEL	SOFT	FREERUN
R-0	R/W-0	R/W-1	R/W-1

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-29. Emulation Register (EMUR) Field Descriptions

Bit	Field	Value	Description
15-3	Reserved	0	Reserved.
2	RT_SEL		Real-time enable.
		0	Enable.
		1	No effect.
1	SOFT		Soft stop.
		0	No effect.
		1	Soft stop enable.
0	FREERUN		Free run.
		0	No effect.
		1	Free run enable.

13.3.6 Mode Registers (MODE1 and MODE2)

The mode registers (MODE1 and MODE2) allow the CPU to individually enable RNDIS/Generic/CDC modes for each endpoint. Using the global RNDIS bit in the control register (CTRLR) overrides this register and enables RNDIS mode for all endpoints. The MODE1 is shown in Figure 13-28 and described in Table 13-30. The MODE2 is shown in Figure 13-29 and described in Table 13-31.

Figure 13-28. Mode Register 1 (MODE1)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		TX4_MODE		Reserved		TX3_MODE		Reserved		TX2_MODE		Reserved		TX1_MODE	
R		R/W		R		R/W		R		R/W		R		R/W	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 13-29. Mode Register 2 (MODE2)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		RX4_MODE		Reserved		RX3_MODE		Reserved		RX2_MODE		Reserved		RX1_MODE	
R		R/W		R		R/W		R		R/W		R		R/W	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-30. Mode Register 1 (MODE1) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-12	TX4_MODE	0-3h	Transmit endpoint 4 mode control.
		0	Transparent mode on Transmit endpoint 4.
		1h	RNDIS mode on Transmit endpoint 4.
		2h	CDC mode on Transmit endpoint 4.
		3h	Generic RNDIS mode on Transmit endpoint 4.
11-10	Reserved	0	Reserved.
9-8	TX3_MODE	0-3h	Transmit endpoint 3 mode control.
		0	Transparent mode on Transmit endpoint 3.
		1h	RNDIS mode on Transmit endpoint 3.
		2h	CDC mode on Transmit endpoint 3.
		3h	Generic RNDIS mode on Transmit endpoint 3.
7-6	Reserved	0	Reserved.
5-4	TX2_MODE	0-3h	Transmit endpoint 2 mode control.
		0	Transparent mode on Transmit endpoint 2.
		1h	RNDIS mode on Transmit endpoint 2.
		2h	CDC mode on Transmit endpoint 2.
		3h	Generic RNDIS mode on Transmit endpoint 2.
3-2	Reserved	0	Reserved.
1-0	TX1_MODE	0-3h	Transmit endpoint 1 mode control.
		0	Transparent mode on Transmit endpoint 1.
		1h	RNDIS mode on Transmit endpoint 1.
		2h	CDC mode on Transmit endpoint 1.
		3h	Generic RNDIS mode on Transmit endpoint 1.

Table 13-31. Mode Register 2 (MODE2) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.

Table 13-31. Mode Register 2 (MODE2) Field Descriptions (continued)

Bit	Field	Value	Description
13-12	RX4_MODE	0-3h	Receive endpoint 4 mode control.
		0	Transparent mode on Receive endpoint 4.
		1h	RNDIS mode on Receive endpoint 4.
		2h	CDC mode on Receive endpoint 4.
		3h	Generic RNDIS mode on Receive endpoint 4.
11-10	Reserved	0	Reserved.
9-8	RX3_MODE	0-3h	Receive endpoint 3 mode control.
		0	Transparent mode on Receive endpoint 3.
		1h	RNDIS mode on Receive endpoint 3.
		2h	CDC mode on Receive endpoint 3.
		3h	Generic RNDIS mode on Receive endpoint 3.
7-6	Reserved	0	Reserved.
5-4	RX2_MODE	0-3h	Receive endpoint 2 mode control.
		0	Transparent mode on Receive endpoint 2.
		1h	RNDIS mode on Receive endpoint 2.
		2h	CDC mode on Receive endpoint 2.
		3h	Generic RNDIS mode on Receive endpoint 2.
3-2	Reserved	0	Reserved.
1-0	RX1_MODE	0-3h	Receive endpoint 1 mode control.
		0	Transparent mode on Receive endpoint 1.
		1h	RNDIS mode on Receive endpoint 1.
		2h	CDC mode on Receive endpoint 1.
		3h	Generic RNDIS mode on Receive endpoint 1.

13.3.7 Auto Request Register (AUTOREQ)

The auto request register (AUTOREQ) allows the CPU to enable an automatic IN token request generation for host mode RX operation per each RX endpoint. This feature has the DMA set the REQPKT bit in the control status register for host receive endpoint (HOST_RXCSR) when it clears the RXPKTRDY bit after reading out a packet. The REQPKT bit is used by the core to generate an IN token to receive data. By using this feature, the host can automatically generate an IN token after the DMA finishes receiving data and empties an endpoint buffer, thus receiving the next data packet as soon as possible from the connected device. Without this feature, the CPU will have to manually set the REQPKT bit for every USB packet.

There are two modes that auto request can function in: always or all except an EOP. The always mode sets the REQPKT bit after every USB packet the DMA receives thus generating a new IN token after each USB packet. The EOP mode sets the REQPKT bit after every USB packet that is not an EOP (end of packet) in the CPPI descriptor. For RNDIS, CDC, and Generic RNDIS modes, the auto request stops when the EOP is received (either via a short packet for RNDIS, CDC, and Generic RNDIS or the count is reached for Generic RNDIS), making it useful for starting a large RNDIS packet and having it auto generate IN tokens until the end of the RNDIS packet. For transparent mode, every USB packet is an EOP CPPI packet so the auto request never functions and acts like auto request is disabled.

The AUTOREQ is shown in Figure 13-30 and described in Table 13-32.

Figure 13-30. Auto Request Register (AUTOREQ)

15 8	7 6	5 4	3 2	1 0
Reserved	RX4_AUTREQ	RX3_AUTREQ	RX2_AUTREQ	RX1_AUTREQ
R-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-32. Auto Request Register (AUTOREQ) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved.
7-6	RX4_AUTREQ	0-3h	Receive endpoint 4 auto request enable.
		0	No auto request.
		1h	Auto request on all but EOP.
		2h	Reserved.
		3h	Auto request always.
5-4	RX3_AUTREQ	0-3h	Receive endpoint 3 auto request enable.
		0	No auto request.
		1h	Auto request on all but EOP.
		2h	Reserved.
		3h	Auto request always.
3-2	RX2_AUTREQ	0-3h	Receive endpoint 2 auto request enable.
		0	No auto request.
		1h	Auto request on all but EOP.
		2h	Reserved.
		3h	Auto request always.
1-0	RX1_AUTREQ	0-3h	Receive endpoint 1 auto request enable.
		0	No auto request.
		1h	Auto request on all but EOP.
		2h	Reserved.
		3h	Auto request always.

13.3.8 SRP Fix Time Registers (SRPFIXTIME1 and SRPFIXTIME2)

The SRP fix time registers (SRPFIXTIME1 and SRPFIXTIME2) allow the CPU to configure the maximum amount of time the SRP fix logic blocks the Avalid from the PHY to the Mentor core. The SRPFIXTIME1 is shown in Figure 13-31 and described in Table 13-33. The SRPFIXTIME2 is shown in Figure 13-32 and described in Table 13-34.

Figure 13-31. SRP Fix Time Register 1 (SRPFIXTIME1)

15 ... 0
SRPFIXTIMELSB
R/W-DE80h

LEGEND: R/W = Read/Write; -*n* = value after reset

Figure 13-32. SRP Fix Time Register 2 (SRPFIXTIME2)

15 ... 0
SRPFIXTIMEMSB
R/W-0280h

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 13-33. SRP Fix Time Register 1 (SRPFIXTIME1) Field Descriptions

Bit	Field	Value	Description
15-0	SRPFIXTIMELSB	0-FFFFh	SRP fix maximum time in 60 MHz cycles. Together, SRPFIXTIME1 and SRPFIXTIME2 specify a 32 bit value. Default is 700 ms (280 DE80h).

Table 13-34. SRP Fix Time Register 2 (SRPFIXTIME2) Field Descriptions

Bit	Field	Value	Description
15-0	SRPFIXTIMEMSB	0-FFFFh	SRP fix maximum time in 60 MHz cycles. Together, SRPFIXTIME1 and SRPFIXTIME2 specify a 32 bit value. Default is 700 ms (280 DE80h).

13.3.9 Teardown Registers (TEARDOWN1 and TEARDOWN2)

The teardown registers (TEARDOWN1 and TEARDOWN2) control the tearing down of receive and transmit FIFOs in the USB controller. When a 1 is written to a valid bit in TEARDOWN1 or TEARDOWN2, the CPPI FIFO pointers for that endpoint are cleared. TEARDOWN1 and TEARDOWN2 must be used in conjunction with the CPPI DMA teardown mechanism. The Host should also write the FLUSHFIFO bits in the TXCSR and RXCSR registers to ensure a complete teardown of the endpoint.

The TEARDOWN1 is shown in Figure 13-33 and described in Table 13-35. The TEARDOWN2 is shown in Figure 13-34 and described in Table 13-36.

Figure 13-33. Teardown Register 1 (TEARDOWN1)

15 5	4 1	0
Reserved	RX_TDOWN	Reserved
R-0	R/W-0	R-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 13-34. Teardown Register 2 (TEARDOWN2)

15 5	4 1	0
Reserved	TX_TDOWN	Reserved
R-0	R/W-0	R-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-35. Teardown Register 1 (TEARDOWN1) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved.
4-1	RX_TDOWN		Receive endpoint teardown.
		0	Disable.
		1	Enable.
0	Reserved	0	Reserved.

Table 13-36. Teardown Register 2 (TEARDOWN2) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved.
4-1	TX_TDOWN		Transmit endpoint teardown.
		0	Disable.
		1	Enable.
0	Reserved	0	Reserved.

13.3.10 USB Interrupt Source Registers (INTSRCR1 and INTSRCR2)

The USB interrupt source registers (INTSRCR1 and INTSRCR2) contain the status of the interrupt sources generated by the USB core (not the DMA). The INTSRCR1 is shown in Figure 13-35 and described in Table 13-37. The INTSRCR2 is shown in Figure 13-36 and described in Table 13-38.

Figure 13-35. USB Interrupt Source Register 1 (INTSRCR1)

15 13	12 9	8 5	4 0
Reserved	RX	Reserved	TX
	R-0		R-0

LEGEND: R = Read only; -*n* = value after reset

Figure 13-36. USB Interrupt Source Register 2 (INTSRCR2)

15 9	8 0
Reserved	USB
R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 13-37. USB Interrupt Source Register 1 (INTSRCR1) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12	RX4	0/1	Receive interrupt source for EndPoint4
11	RX3	0/1	Receive interrupt source for EndPoint3
10	RX2	0/1	Receive interrupt source for EndPoint2
9	RX1	0/1	Receive interrupt source for EndPoint1
8-5	Reserved	0	
4	TX4	0/1	Transmit interrupt source for EndPoint4
3	TX3	0/1	Transmit interrupt source for EndPoint3
2	TX2	0/1	Transmit interrupt source for EndPoint2
1	TX1	0/1	Transmit interrupt source for EndPoint1
0	RX1/TX1	0/1	Both Receive and Transmit interrupt source for EndPoint0

Table 13-38. USB Interrupt Source Register 2 (INTSRCR2) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved.
8-0	USB	0-1FFh	USB interrupt sources. (Please see Figure 13-67 for the definition of each bit here.)

13.3.11 USB Interrupt Source Set Registers (INTSETR1 and INTSETR2)

The USB interrupt source set registers (INTSETR1 and INTSETR2) allow the USB interrupt sources to be manually triggered. A read of this register returns the USB interrupt source register value. The INTSETR1 is shown in Figure 13-37 and described in Table 13-39. The INTSETR2 is shown in Figure 13-38 and described in Table 13-40.

Figure 13-37. USB Interrupt Source Set Register 1 (INTSETR1)

15 13	12 9	8 5	4 0
Reserved	RX	Reserved	TX
R-0	R/W-0	R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 13-38. USB Interrupt Source Set Register 2 (INTSETR2)

15 9	8 0
Reserved	USB
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-39. USB Interrupt Source Set Register 1 (INTSETR1) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12-8	RX	0-Fh	Write a 1 to set equivalent Receive endpoint interrupt source. Allows the USB interrupt sources to be manually triggered.
7-5	Reserved	0	Reserved.
4-0	TX	0-1Fh	Write a 1 to set equivalent Transmit endpoint interrupt source. Allows the USB interrupt sources to be manually triggered.

Table 13-40. USB Interrupt Source Set Register 2 (INTSETR2) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved.
8-0	USB	0-1FFh	Write a 1 to set equivalent USB interrupt source. Allows the USB interrupt sources to be manually triggered.

13.3.12 USB Interrupt Source Clear Registers (INTCLRR1 and INTCLRR2)

The USB interrupt source clear registers (INTCLRR1 and INTCLRR2) allow the CPU to acknowledge an interrupt source and turn it off. A read of this register returns the USB interrupt source register value. The INTCLRR1 is shown in Figure 13-39 and described in Table 13-41. The INTCLRR2 is shown in Figure 13-40 and described in Table 13-42.

Figure 13-39. USB Interrupt Source Clear Register 1 (INTCLRR1)

15 13	12 9	8 5	4 0
Reserved	RX	Reserved	TX
R-0	R/W-0	R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 13-40. USB Interrupt Source Clear Register 2 (INTCLRR2)

15 9	8 0
Reserved	USB
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-41. USB Interrupt Source Clear Register 1 (INTCLRR1) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12-8	RX	0-Fh	Write a 1 to clear equivalent Receive endpoint interrupt source. Allows the CPU to acknowledge an interrupt source and turn it off.
7-5	Reserved	0	Reserved.
4-0	TX	0-1Fh	Write a 1 to clear equivalent Transmit endpoint interrupt source. Allows the CPU to acknowledge an interrupt source and turn it off.

Table 13-42. USB Interrupt Source Clear Register 2 (INTCLRR2) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved.
8-0	USB	0-1FFh	Write a 1 to clear equivalent USB interrupt source. Allows the CPU to acknowledge an interrupt source and turn it off.

13.3.13 USB Interrupt Mask Registers (INTMSKR1 and INTMSKR2)

The USB interrupt mask registers (INTMSKR1 and INTMSKR2) contain the masks of the interrupt sources generated by the USB core (not the DMA). These masks are used to enable or disable interrupt sources generated on the masked source interrupts (the raw source interrupts are never masked). The bit positions are maintained in the same position as the interrupt sources in the USB interrupt source register.

The INTMSKR1 is shown in Figure 13-41 and described in Table 13-43. The INTMSKR2 is shown in Figure 13-42 and described in Table 13-44.

Figure 13-41. USB Interrupt Mask Register 1 (INTMSKR1)

15 13	12 9	8 5	4 0
Reserved	RX	Reserved	TX
R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Figure 13-42. USB Interrupt Mask Register 2 (INTMSKR2)

15 9	8 0
Reserved	USB
R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 13-43. USB Interrupt Mask Register 1 (INTMSKR1) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12-8	RX	0-Fh	Receive endpoint interrupt source masks.
7-5	Reserved	0	Reserved.
4-0	TX	0-1Fh	Transmit endpoint interrupt source masks.

Table 13-44. USB Interrupt Mask Register 2 (INTMSKR2) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved.
8-0	USB	0-1FFh	USB interrupt source masks.

13.3.14 USB Interrupt Mask Set Registers (INTMSKSETR1 and INTMSKSETR2)

The USB interrupt mask set registers (INTMSKSETR1 and INTMSKSETR2) allow the USB masks to be individually enabled. A read to this register returns the USB interrupt mask register value. The INTMSKSETR1 is shown in Figure 13-43 and described in Table 13-45. The INTMSKSETR2 is shown in Figure 13-44 and described in Table 13-46.

Figure 13-43. USB Interrupt Mask Set Register 1 (INTMSKSETR1)

15 13	12 9	8 5	4 0
Reserved	RX	Reserved	TX
R-0	R/W-0	R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 13-44. USB Interrupt Mask Set Register 2 (INTMSKSETR2)

15 9	8 0
Reserved	USB
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-45. USB Interrupt Mask Set Register 1 (INTMSKSETR1) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12-8	RX	0-Fh	Write a 1 to set equivalent Receive endpoint interrupt mask.
7-5	Reserved	0	Reserved.
4-0	TX	0-1Fh	Write a 1 to set equivalent Transmit endpoint interrupt mask.

Table 13-46. USB Interrupt Mask Set Register 2 (INTMSKSETR2) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved.
8-0	USB	0-1FFh	Write a 1 to set equivalent USB interrupt mask.

13.3.15 USB Interrupt Mask Clear Registers (INTMSKCLRR1 and INTMSKCLRR2)

The USB interrupt mask clear registers (INTMSKCLRR1 and INTMSKCLRR2) allow the USB interrupt masks to be individually disabled. A read to this register returns the USB interrupt mask register value. The INTMSKCLRR1 is shown in Figure 13-45 and described in Table 13-47. The INTMSKCLRR2 is shown in Figure 13-46 and described in Table 13-48.

Figure 13-45. USB Interrupt Mask Clear Register 1 (INTMSKCLRR1)

15 13	12 9	8 5	4 0
Reserved	RX	Reserved	TX
R-0	R/W-0	R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 13-46. USB Interrupt Mask Clear Register 2 (INTMSKCLRR2)

15 9	8 0
Reserved	USB
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-47. USB Interrupt Mask Clear Register 1 (INTMSKCLRR1) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12-8	RX	0-Fh	Write a 1 to clear equivalent Receive endpoint interrupt mask.
7-5	Reserved	0	Reserved.
4-0	TX	0-1Fh	Write a 1 to clear equivalent Transmit endpoint interrupt mask.

Table 13-48. USB Interrupt Mask Clear Register 2 (INTMSKCLRR2) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved.
8-0	USB	0-1FFh	Write a 1 to clear equivalent USB interrupt mask.

13.3.16 USB Interrupt Source Masked Registers (INTMASKEDR1 and INTMASKEDR2)

The USB interrupt source masked registers (INTMASKEDR1 and INTMASKEDR2) contain the status of the interrupt sources generated by the USB core masked by the USB interrupt mask register values. The INTMASKEDR1 is shown in Figure 13-47 and described in Table 13-49. The INTMASKEDR2 is shown in Figure 13-48 and described in Table 13-50.

Figure 13-47. USB Interrupt Source Masked Register 1 (INTMASKEDR1)

15 13	12 9	8 5	4 0
Reserved	RX	Reserved	TX
R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Figure 13-48. USB Interrupt Source Masked Register 2 (INTMASKEDR2)

15 9	8 0
Reserved	USB
R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 13-49. USB Interrupt Source Masked Register 1 (INTMASKEDR1) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12-8	RX	0-Fh	Receive endpoint interrupt sources masked.
7-5	Reserved	0	Reserved.
4-0	TX	0-1Fh	Transmit endpoint interrupt sources masked.

Table 13-50. USB Interrupt Source Masked Register 2 (INTMASKEDR2) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved.
8-0	USB	0-1FFh	USB interrupt sources masked.

13.3.17 USB End of Interrupt Register (EOIR)

The USB end of interrupt register (EOIR) allows the CPU to acknowledge completion of an interrupt by writing 0 to the EOI_VECTOR bit. The EOIR is shown in Figure 13-49 and described in Table 13-51.

Figure 13-49. USB End of Interrupt Register (EOIR)

15 8	7 0
Reserved	EOI_VECTOR
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-51. USB End of Interrupt Register (EOIR) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved.
7-0	EOI_VECTOR	0-FFh	EOI Vector.

13.3.18 USB Interrupt Vector Registers (INTVECTR1 and INTVECTR2)

The USB interrupt vector registers (INTVECTR1 and INTVECTR2) recycle the Interrupt Vector input to be read by the CPU. The INTVECTR1 is shown in Figure 13-50 and described in Table 13-52. The INTVECTR2 is shown in Figure 13-51 and described in Table 13-53.

Figure 13-50. USB Interrupt Vector Register 1 (INTVECTR1)

15   0
VECTORLSB
R-0

LEGEND: R = Read only; -*n* = value after reset

Figure 13-51. USB Interrupt Vector Register 2 (INTVECTR2)

15   0
VECTORMSB
R-0

LEGEND: R = Read only; -*n* = value after reset

Table 13-52. USB Interrupt Vector Register 1 (INTVECTR1) Field Descriptions

Bit	Field	Value	Description
15-0	VECTORLSB	0-FFFFh	Input Interrupt Vector. Together, INTVECTR1 and INTVECTR2 specify a 32 bit value.

Table 13-53. USB Interrupt Vector Register 2 (INTVECTR2) Field Descriptions

Bit	Field	Value	Description
15-0	VECTORMSB	0-FFFFh	Input Interrupt Vector. Together, INTVECTR1 and INTVECTR2 specify a 32 bit value.

13.3.19 Generic RNDIS EP1 Size Registers (GREP1SZR1 and GREP1SZR2)

The generic RNDIS EP1 size registers (GREP1SZR1 and GREP1SZR2) are programmed with a RNDIS packet size in bytes. When EP1 is in Generic RNDIS mode, the received USB packets are collected into a single CPPI packet that is completed when the number of bytes equal to the value of this register have been received, or a *short* packet is received. The packet size must be an integer multiple of the endpoint size. The maximum packet size that can be used is 10000h, or 65536.

The GREP1SZR1 is shown in Figure 13-52 and described in Table 13-54. The GREP1SZR2 is shown in Figure 13-53 and described in Table 13-55.

Figure 13-52. Generic RNDIS EP1 Size Register 1 (GREP1SZR1)

15 — 0
EP1_SIZE_LSB
R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 13-53. Generic RNDIS EP1 Size Register 2 (GREP1SZR2)

15 — 1	0
Reserved	EP1_SIZE_MSB
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-54. Generic RNDIS EP1 Size Register 1 (GREP1SZR1) Field Descriptions

Bit	Field	Value	Description
15-0	EP1_SIZE_LSB	0-FFFFh	Generic RNDIS packet size. Together, GREP1SZR1 and GREP1SZR2 specify the packet size.

Table 13-55. Generic RNDIS EP1 Size Register 2 (GREP1SZR2) Field Descriptions

Bit	Field	Value	Description
15-1	Reserved	0	Reserved.
0	EP1_SIZE_MSB	0-1	Generic RNDIS packet size. Together, GREP1SZR1 and GREP1SZR2 specify the packet size.

13.3.20 Generic RNDIS EP2 Size Registers (GREP2SZR1 and GREP2SZR2)

The generic RNDIS EP2 size registers (GREP2SZR1 and GREP2SZR2) are programmed with a RNDIS packet size in bytes. When EP2 is in Generic RNDIS mode, the received USB packets are collected into a single CPPI packet that is completed when the number of bytes equal to the value of this register have been received, or a *short* packet is received. The packet size must be an integer multiple of the endpoint size. The maximum packet size that can be used is 10000h, or 65536.

The GREP2SZR1 is shown in Figure 13-54 and described in Table 13-56. The GREP2SZR2 is shown in Figure 13-55 and described in Table 13-57.

Figure 13-54. Generic RNDIS EP2 Size Register 1 (GREP2SZR1)

15 — 0
EP2_SIZE_LSB
R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 13-55. Generic RNDIS EP2 Size Register 2 (GREP2SZR2)

15 — 1	0
Reserved	EP2_SIZE_MSB
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-56. Generic RNDIS EP2 Size Register 1 (GREP2SZR1) Field Descriptions

Bit	Field	Value	Description
15-0	EP2_SIZE_LSB	0-FFFFh	Generic RNDIS packet size. Together, GREP2SZR1 and GREP2SZR2 specify the packet size.

Table 13-57. Generic RNDIS EP2 Size Register 2 (GREP2SZR2) Field Descriptions

Bit	Field	Value	Description
15-1	Reserved	0	Reserved.
0	EP2_SIZE_MSB	0-1	Generic RNDIS packet size. Together, GREP2SZR1 and GREP2SZR2 specify the packet size.

13.3.21 Generic RNDIS EP3 Size Registers (GREP3SZR1 and GREP3SZR2)

The generic RNDIS EP3 size registers (GREP3SZR1 and GREP3SZR2) are programmed with a RNDIS packet size in bytes. When EP3 is in Generic RNDIS mode, the received USB packets are collected into a single CPPI packet that is completed when the number of bytes equal to the value of this register has been received, or a *short* packet is received. The packet value must be an integer multiple of the endpoint size. The maximum packet size that can be used is 10000h, or 65536.

The GREP3SZR1 is shown in Figure 13-56 and described in Table 13-58. The GREP3SZR2 is shown in Figure 13-57 and described in Table 13-59.

Figure 13-56. Generic RNDIS EP3 Size Register 1 (GREP3SZR1)

15 0
EP3_SIZE_LSB
R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 13-57. Generic RNDIS EP3 Size Register 2 (GREP3SZR2)

15 1	0
Reserved	EP3_SIZE_MSB
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-58. Generic RNDIS EP3 Size Register 1 (GREP3SZR1) Field Descriptions

Bit	Field	Value	Description
15-0	EP3_SIZE_LSB	0-FFFFh	Generic RNDIS packet size. Together, GREP3SZR1 and GREP3SZR2 specify the packet size.

Table 13-59. Generic RNDIS EP3 Size Register 2 (GREP3SZR2) Field Descriptions

Bit	Field	Value	Description
15-1	Reserved	0	Reserved.
0	EP3_SIZE_MSB	0-1	Generic RNDIS packet size. Together, GREP3SZR1 and GREP3SZR2 specify the packet size.

13.3.22 Generic RNDIS EP4 Size Registers (GREP4SZR1 and GREP4SZR2)

The generic RNDIS EP4 size registers (GREP4SZR1 and GREP4SZR2) are programmed with a RNDIS packet size in bytes. When EP4 is in Generic RNDIS mode, the received USB packets are collected into a single CPPI packet that is completed when the number of bytes equal to the value of this register has been received, or a *short* packet is received. The packet size must be an integer multiple of the endpoint size. The maximum packet size that can be used is 10000h, or 65536.

The GREP4SZR1 is shown in Figure 13-58 and described in Table 13-60. The GREP4SZR2 is shown in Figure 13-59 and described in Table 13-61.

Figure 13-58. Generic RNDIS EP4 Size Register 1 (GREP4SZR1)

15 0
EP4_SIZE_LSB
R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 13-59. Generic RNDIS EP4 Size Register 2 (GREP4SZR2)

15 1	0
Reserved	EP4_SIZE_MSB
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-60. Generic RNDIS EP4 Size Register 1 (GREP4SZR1) Field Descriptions

Bit	Field	Value	Description
15-0	EP4_SIZE_LSB	0-FFFFh	Generic RNDIS packet size. Together, GREP4SZR1 and GREP4SZR2 specify the packet size.

Table 13-61. Generic RNDIS EP4 Size Register 2 (GREP4SZR2) Field Descriptions

Bit	Field	Value	Description
15-1	Reserved	0	Reserved
0	EP4_SIZE_MSB	0-1	Generic RNDIS packet size. Together, GREP4SZR1 and GREP4SZR2 specify the packet size.

13.3.23 Function Address Register (FADDR)

The function address register (FADDR) is shown in Figure 13-60 and described in Table 13-62.

Figure 13-60. Function Address Register (FADDR)

7	6	0
Reserved	FUNCADDR	
R-0	R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 13-62. Function Address Register (FADDR) Field Descriptions

Bit	Field	Value	Description
7	Reserved	0	Reserved.
6-0	FUNCADDR	0-7Fh	7_bit address of the peripheral part of the transaction. This register should be written with the address received through a SET_ADDRESS command, which will then be used for decoding the function address in subsequent token packets. When used in Host mode, this register should be set to the value sent in a SET_ADDRESS command during device enumeration as the address for the peripheral device.

13.3.24 Power Management Register (POWER)

The power management register (POWER) is shown in Figure 13-61 and described in Table 13-63.

Figure 13-61. Power Management Register (POWER)

7	6	5	4	3	2	1	0
ISOUPDATE	SOFTCONN	HSEN	HSMODE	RESET	RESUME	SUSPENDM	ENSUSPM
R/W-0	R/W-0	R/W-1	R-0	R-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-63. Power Management Register (POWER) Field Descriptions

Bit	Field	Value	Description
7	ISOUPDATE	0-1	When set, the USB controller will wait for an SOF token from the time TxPktRdy is set before sending the packet. If an IN token is received before an SOF token, then a zero length data packet will be sent. This bit only affects endpoints performing Isochronous transfers.
6	SOFTCONN	0-1	If Soft Connect/Disconnect feature is enabled, then the USB D+/D- lines are enabled when this bit is set and tri-stated when this bit is cleared.
5	HSEN	0-1	When set, the USB controller will negotiate for high-speed mode when the device is reset by the hub. If not set, the device will only operate in full-speed mode.
4	HSMODE	0-1	This bit is set when the USB controller has successfully negotiated for high-speed mode.
3	RESET	0-1	This bit is set when Reset signaling is present on the bus. Note: this bit is read-only.
2	RESUME	0-1	Set to generate Resume signaling when the controller is in Suspend mode. The bit should be cleared after 10 ms (a maximum of 15 ms) to end Resume signaling. In Host mode, this bit is also automatically set when Resume signaling from the target is detected while the USB controller is suspended.
1	SUSPENDM	0-1	This bit is set on entry into Suspend mode. It is cleared when the interrupt register is read, or the RESUME bit is set.
0	ENSUSPM	0-1	Set to enable the SUSPENDM output.

13.3.25 Interrupt Register for Endpoint 0 Plus Transmit Endpoints 1 to 4 (INTRTX)

The interrupt register for endpoint 0 plus transmit endpoints 1 to 4 (INTRTX) is shown in Figure 13-62 and described in Table 13-64.

Figure 13-62. Interrupt Register for Endpoint 0 Plus Tx Endpoints 1 to 4 (INTRTX)

15		8
	Reserved	
	R-0	

7	5	4	3	2	1	0
Reserved		EP4TX	EP3TX	EP2TX	EP1TX	EP0
R-0		R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 13-64. Interrupt Register for Endpoint 0 Plus Transmit Endpoints 1 to 4 (INTRTX) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved.
4	EP4TX	0-1	Transmit Endpoint 4 interrupt active.
3	EP3TX	0-1	Transmit Endpoint 3 interrupt active.
2	EP2TX	0-1	Transmit Endpoint 2 interrupt active.
1	EP1TX	0-1	Transmit Endpoint 1 interrupt active.
0	EP0	0-1	Endpoint 0 interrupt active.

13.3.26 Interrupt Register for Receive Endpoints 1 to 4 (INTRRX)

The interrupt register for receive endpoints 1 to 4 (INTRRX) is shown in Figure 13-63 and described in Table 13-65.

Figure 13-63. Interrupt Register for Receive Endpoints 1 to 4 (INTRRX)

15		8
	Reserved	
	R-0	

7	5	4	3	2	1	0
Reserved		EP4RX	EP3RX	EP2RX	EP1RX	Reserved
R-0		R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 13-65. Interrupt Register for Receive Endpoints 1 to 4 (INTRRX) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved.
4	EP4RX	0-1	Receive Endpoint 4 interrupt active.
3	EP3RX	0-1	Receive Endpoint 3 interrupt active.
2	EP2RX	0-1	Receive Endpoint 2 interrupt active.
1	EP1RX	0-1	Receive Endpoint 1 interrupt active.
0	Reserved	0	Reserved.

13.3.27 Interrupt Enable Register for INTRTX (INTRTXE)

The interrupt enable register for INTRTX (INTRTXE) is shown in Figure 13-64 and described in Table 13-66.

Figure 13-64. Interrupt Enable Register for INTRTX (INTRTXE)

15	8
Reserved	
R-0	

7	5	4	3	2	1	0
Reserved		EP4TX	EP3TX	EP2TX	EP1TX	EP0
R-0		R/W-1	R/W-1	R/W-1	R/W-1	R/W-1

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-66. Interrupt Enable Register for INTRTX (INTRTXE) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved.
4	EP4TX	0-1	Transmit Endpoint 4 interrupt active.
3	EP3TX	0-1	Transmit Endpoint 3 interrupt active.
2	EP2TX	0-1	Transmit Endpoint 2 interrupt active.
1	EP1TX	0-1	Transmit Endpoint 1 interrupt active.
0	EP0	0-1	Endpoint 0 interrupt active.

13.3.28 Interrupt Enable Register for INTRRX (INTRRXE)

The interrupt enable register for INTRRX (INTRRXE) is shown in Figure 13-65 and described in Table 13-67.

Figure 13-65. Interrupt Enable Register for INTRRX (INTRRXE)

15	8
Reserved	
R-0	

7	5	4	3	2	1	0
Reserved		EP4RX	EP3RX	EP2RX	EP1RX	Reserved
R-0		R/W-1	R/W-1	R/W-1	R/W-1	R-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-67. Interrupt Enable Register for INTRRX (INTRRXE) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved.
4	EP4RX	0-1	Receive Endpoint 4 interrupt active.
3	EP3RX	0-1	Receive Endpoint 3 interrupt active.
2	EP2RX	0-1	Receive Endpoint 2 interrupt active.
1	EP1RX	0-1	Receive Endpoint 1 interrupt active.
0	Reserved	0	Reserved.

13.3.29 Interrupt Register for Common USB Interrupts (INTRUSB)

The interrupt register for common USB interrupts (INTRUSB) is shown in Figure 13-66 and described in Table 13-68.

NOTE: Unless the UINT bit of CTRLR is set, do not read or write this register directly. Use the INTSRCR register instead.

Figure 13-66. Interrupt Register for Common USB Interrupts (INTRUSB)

7	6	5	4	3	2	1	0
VBUSERR	SESSREQ	DISCON	CONN	SOF	RESET_BABBLE	RESUME	SUSPEND
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 13-68. Interrupt Register for Common USB Interrupts (INTRUSB) Field Descriptions

Bit	Field	Value	Description
7	VBUSERR	0-1	Set when VBus drops below the VBus valid threshold during a session. Only valid when the USB controller is 'A' device. All active interrupts will be cleared when this register is read.
6	SESSREQ	0-1	Set when session request signaling has been detected. Only valid when USB controller is 'A' device.
5	DISCON	0-1	Set when a session ends.
4	CONN	0-1	Set when a device connection is detected. Only valid in host mode.
3	SOF	0-1	Set when a new frame starts.
2	RESET_BABBLE	0-1	Set when reset signaling is detected on the bus.
1	RESUME	0-1	Set when resume signaling is detected on the bus while the USB controller is in suspend mode.
0	SUSPEND	0-1	Set when suspend signaling is detected on the bus.

13.3.30 Interrupt Enable Register for INTRUSB (INTRUSBE)

The interrupt enable register for INTRUSB (INTRUSBE) is shown in Figure 13-67 and described in Table 13-69.

NOTE: Unless the UINT bit of CTRLR is set, do not read or write this register directly. Use the INTSETR/INTCLRR registers instead.

Figure 13-67. Interrupt Enable Register for INTRUSB (INTRUSBE)

7	6	5	4	3	2	1	0
VBUSERR	SESSREQ	DISCON	CONN	SOF	RESET_BABBLE	RESUME	SUSPEND
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 13-69. Interrupt Enable Register for INTRUSB (INTRUSBE) Field Descriptions

Bit	Field	Value	Description
7	VBUSERR	0-1	Vbus error interrupt enable.
6	SESSREQ	0-1	Session request interrupt enable.
5	DISCON	0-1	Disconnect interrupt enable.
4	CONN	0-1	Connect interrupt enable.
3	SOF	0-1	Start of frame interrupt enable.
2	RESET_BABBLE	0-1	Reset interrupt enable.
1	RESUME	0-1	Resume interrupt enable.
0	SUSPEND	0-1	Suspend interrupt enable.

13.3.31 Frame Number Register (FRAME)

The frame number register (FRAME) is shown in Figure 13-68 and described in Table 13-70.

Figure 13-68. Frame Number Register (FRAME)

15	11	10	0
Reserved		FRAMENUMBER	
R-0		R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 13-70. Frame Number Register (FRAME) Field Descriptions

Bit	Field	Value	Description
15-11	Reserved	0	Reserved.
10-0	FRAMENUMBER	0-7FFh	Last received frame number.

13.3.32 Index Register for Selecting the Endpoint Status and Control Registers (INDEX)

The index register for selecting the endpoint status and control registers (INDEX) is shown in Figure 13-69 and described in Table 13-71.

Figure 13-69. Index Register for Selecting the Endpoint Status and Control Registers (INDEX)

7 4	3 0
Reserved	EPSEL
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-71. Index Register for Selecting the Endpoint Status and Control Registers (INDEX) Field Descriptions

Bit	Field	Value	Description
7-4	Reserved	0	Reserved.
3-0	EPSEL	0-Fh	Each transmit endpoint and each receive endpoint have their own set of control/status registers. EPSEL determines which endpoint control/status registers are accessed. Before accessing an endpoint's control/status registers, the endpoint number should be written to the Index register to ensure that the correct control/status registers appear in the memory-map.

13.3.33 Register to Enable the USB 2.0 Test Modes (TESTMODE)

The register to enable the USB 2.0 test modes (TESTMODE) is shown in Figure 13-70 and described in Table 13-72.

Figure 13-70. Register to Enable the USB 2.0 Test Modes (TESTMODE)

7	6	5	4	3	2	1	0
FORCE_HOST	FIFO_ACCESS	FORCE_FS	FORCE_HS	TEST_PACKET	TEST_K	TEST_J	TEST_SE0_NAK
R/W-0	W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; W = Write only; -*n* = value after reset

Table 13-72. Register to Enable the USB 2.0 Test Modes (TESTMODE) Field Descriptions

Bit	Field	Value	Description
7	FORCE_HOST	0-1	Set this bit to forcibly put the USB controller into Host mode when SESSION bit is set, regardless of whether it is connected to any peripheral. The controller remains in Host mode until the Session bit is cleared, even if a device is disconnected. And if the FORCE_HOST but remains set, it will re-enter Host mode next time the SESSION bit is set. The operating speed is determined using the FORCE_HS and FORCE_FS bits.
6	FIFO_ACCESS	0-1	Set this bit to transfer the packet in EP0 Tx FIFO to EP0 Receive FIFO. It is cleared automatically.
5	FORCE_FS	0-1	Set this bit to force the USB controller into full-speed mode when it receives a USB reset.
4	FORCE_HS	0-1	Set this bit to force the USB controller into high-speed mode when it receives a USB reset.
3	TEST_PACKET	0-1	Set this bit to enter the Test_Packet test mode. In this mode, the USB controller repetitively transmits a 53-byte test packet on the bus, the form of which is defined in the Universal Serial Bus Specification Revision 2.0. Note: The test packet has a fixed format and must be loaded into the Endpoint 0 FIFO before the test mode is entered.
2	TEST_K	0-1	Set this bit to enter the Test_K test mode. In this mode, the USB controller transmits a continuous K on the bus.
1	TEST_J	0-1	Set this bit to enter the Test_J test mode. In this mode, the USB controller transmits a continuous J on the bus.
0	TEST_SE0_NAK	0-1	Set this bit to enter the Test_SE0_NAK test mode. In this mode, the USB controller remains in high-speed mode, but responds to any valid IN token with a NAK.

13.3.34 Maximum Packet Size for Peripheral/Host Transmit Endpoint (TXMAXP)

The maximum packet size for peripheral/host transmit endpoint (TXMAXP) is shown in Figure 13-71 and described in Table 13-73.

Figure 13-71. Maximum Packet Size for Peripheral/Host Transmit Endpoint (TXMAXP)

15	11	10	0
Reserved		MAXPAYLOAD	
R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-73. Maximum Packet Size for Peripheral/Host Transmit Endpoint (TXMAXP) Field Descriptions

Bit	Field	Value	Description
15-11	Reserved	0	Reserved.
10-0	MAXPAYLOAD	0-FFh	The maximum payload transmitted in a single transaction. The value set can be up to 1024 bytes, but is subject to the constraints placed by the USB Specification on packet sizes for Bulk, Interrupt, and Isochronous transfers in full-speed and high-speed operations. The value written to this register should match the wMaxPacketSize field of the Standard Endpoint Descriptor for the associated endpoint. A mismatch could cause unexpected results.

13.3.35 Control Status Register for Peripheral Endpoint 0 (PERI_CSR0)

The control status register for peripheral endpoint 0 (PERI_CSR0) is shown in Figure 13-72 and described in Table 13-74.

Figure 13-72. Control Status Register for Peripheral Endpoint 0 (PERI_CSR0)

15 - 9	8
Reserved	FLUSHFIFO
R-0	W-0

7	6	5	4	3	2	1	0
SERV_SETUPEND	SERV_RXPKTRDY	SENDSTALL	SETUPEND	DATAEND	SENTSTALL	TXPKTRDY	RXPKTRDY
W-0	W-0	W-0	R-0	W-0	R/W-0	R/W-0	R-0

LEGEND: R/W = Read/Write; R = Read only; W = Write only; -*n* = value after reset

Table 13-74. Control Status Register for Peripheral Endpoint 0 (PERI_CSR0) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved.
8	FLUSHFIFO	0-1	Set this bit to flush the next packet to be transmitted/read from the Endpoint 0 FIFO. The FIFO pointer is reset and the TXPKTRDY/RXPKTRDY bit is cleared. Note: FLUSHFIFO has no effect unless TXPKTRDY/RXPKTRDY is set.
7	SERV_SETUPEND	0-1	Set this bit to clear the SETUPEND bit. It is cleared automatically.
6	SERV_RXPKTRDY	0-1	Set this bit to clear the RXPKTRDY bit. It is cleared automatically.
5	SENDSTALL	0-1	Set this bit to terminate the current transaction. The STALL handshake will be transmitted and then this bit will be cleared automatically.
4	SETUPEND	0-1	This bit will be set when a control transaction ends before the DATAEND bit has been set. An interrupt will be generated, and the FIFO will be flushed at this time. The bit is cleared by the writing a 1 to the SERV_SETUPEND bit.
3	DATAEND	0-1	Set this bit to 1: a. When setting TXPKTRDY for the last data packet. b. When clearing RXPKTRDY after unloading the last data packet. c. When setting TXPKTRDY for a zero length data packet. It is cleared automatically.
2	SENTSTALL	0-1	This bit is set when a STALL handshake is transmitted. This bit should be cleared.
1	TXPKTRDY	0-1	Set this bit after loading a data packet into the FIFO. It is cleared automatically when the data packet has been transmitted. An interrupt is generated (if enabled) when the bit is cleared.
0	RXPKTRDY	0-1	This bit is set when a data packet has been received. An interrupt is generated when this bit is set. This bit is cleared by setting the SERV_RXPKTRDY bit.

13.3.36 Control Status Register for Peripheral Transmit Endpoint (PERI_TXCSR)

The control status register for peripheral transmit endpoint (PERI_TXCSR) is shown in Figure 13-73 and described in Table 13-75.

Figure 13-73. Control Status Register for Peripheral Transmit Endpoint (PERI_TXCSR)

15	14	13	12	11	10	9 – 7
AUTOSET	ISO	MODE	DMAEN	FRCDATATOG	DMAMODE	Reserved
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0

6	5	4	3	2	1	0
CLRDATATOG	SENTSTALL	SENDSTALL	FLUSHFIFO	UNDERRUN	FIFONOTEMPTY	TXPKTRDY
W-0	R/W-0	R/W-0	W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; W = Write only; -*n* = value after reset

Table 13-75. Control Status Register for Peripheral Transmit Endpoint (PERI_TXCSR) Field Descriptions

Bit	Field	Value	Description
15	AUTOSET	0	DMA Mode: The CPU needs to set the AUTOSET bit prior to enabling the Tx DMA.
		1	CPU Mode: If the CPU sets the AUTOSET bit, the TXPKTRDY bit will be automatically set when data of the maximum packet size (value in TXMAXP) is loaded into the Tx FIFO. If a packet of less than the maximum packet size is loaded, then the TXPKTRDY bit will have to be set manually.
14	ISO	0-1	Set this bit to enable the Tx endpoint for Isochronous transfers, and clear it to enable the Tx endpoint for Bulk or Interrupt transfers.
13	MODE	0-1	Set this bit to enable the endpoint direction as Tx, and clear the bit to enable it as Rx. Note: This bit has any effect only where the same endpoint FIFO is used for both Transmit and Receive transactions.
12	DMAEN	0-1	Set this bit to enable the DMA request for the Tx endpoint.
11	FRCDATATOG	0-1	Set this bit to force the endpoint data toggle to switch and the data packet to be cleared from the FIFO, regardless of whether an ACK was received. This can be used by Interrupt Tx endpoints that are used to communicate rate feedback for Isochronous endpoints.
10	DMAMODE	0-1	Set to 1 when DMA is enabled and EP interrupt is not needed for each packet transmission.
9-7	Reserved	0	Reserved.
6	CLRDATATOG	0-1	Write a 1 to this bit to reset the endpoint data toggle to 0.
5	SENTSTALL	0-1	This bit is set automatically when a STALL handshake is transmitted. The FIFO is flushed and the TXPKTRDY bit is cleared. You should clear this bit.
4	SENDSTALL	0-1	Write a 1 to this bit to issue a STALL handshake to an IN token. Clear this bit to terminate the stall condition. Note: This bit has no effect where the endpoint is being used for Isochronous transfers.
3	FLUSHFIFO	0-1	Write a 1 to this bit to flush the next packet to be transmitted from the endpoint Tx FIFO. The FIFO pointer is reset and the TXPKTRDY bit is cleared. Note: FlushFIFO has no effect unless the TXPKTRDY bit is set. Also note that, if the FIFO is double-buffered, FlushFIFO may need to be set twice to completely clear the FIFO.
2	UNDERRUN	0-1	This bit is set automatically if an IN token is received when TXPKTRDY is not set. You should clear this bit.
1	FIFONOTEMPTY	0-1	This bit is set when there is at least 1 packet in the Tx FIFO. You should clear this bit.
0	TXPKTRDY	0-1	Set this bit after loading a data packet into the FIFO. It is cleared automatically when a data packet has been transmitted. An interrupt is generated (if enabled) when the bit is cleared.

13.3.37 Maximum Packet Size for Peripheral Receive Endpoint (RXMAXP)

The maximum packet size for peripheral receive endpoint (RXMAXP) is shown in Figure 13-74 and described in Table 13-76.

Figure 13-74. Maximum Packet Size for Peripheral Receive Endpoint (RXMAXP)

15 11	10 0
Reserved	MAXPAYLOAD
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-76. Maximum Packet Size for Peripheral Receive Endpoint (RXMAXP) Field Descriptions

Bit	Field	Value	Description
15-11	Reserved	0	Reserved.
10-0	MAXPAYLOAD	0-FFh	Defines the maximum amount of data that can be transferred through the selected Receive endpoint in a single frame/microframe (high-speed transfers). The value set can be up to 1024 bytes, but is subject to the constraints placed by the USB Specification on packet sizes for Bulk, Interrupt, and Isochronous transfers in full-speed and high-speed operations. The value written to this register should match the wMaxPacketSize field of the Standard Endpoint Descriptor for the associated endpoint. A mismatch could cause unexpected results.

13.3.38 Control Status Register for Peripheral Receive Endpoint (PERI_RXCSR)

The control status register for peripheral receive endpoint (PERI_RXCSR) is shown in Figure 13-75 and described in Table 13-77.

Figure 13-75. Control Status Register for Peripheral Receive Endpoint (PERI_RXCSR)

15	14	13	12	11	10		8
AUTOCLEAR	ISO	DMAEN	DISNYET	DMAMODE	Reserved		
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0		

7	6	5	4	3	2	1	0
CLRDATATOG	SENTSTALL	SENDSTALL	FLUSHFIFO	DATAERROR	OVERRUN	FIFOFULL	RXPKTRDY
W-0	R/W-0	R/W-0	W-0	R-0	R/W-0	R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; W = Write only; -*n* = value after reset

Table 13-77. Control Status Register for Peripheral Receive Endpoint (PERI_RXCSR) Field Descriptions

Bit	Field	Value	Description
15	AUTOCLEAR	0	DMA Mode: The CPU sets the AUTOCLEAR bit prior to enabling the Rx DMA.
		1	CPU Mode: If the CPU sets the AUTOCLEAR bit, then the RXPKTRDY bit will be automatically cleared when a packet of RXMAXP bytes has been unloaded from the Receive FIFO. When packets of less than the maximum packet size are unloaded, RXPKTRDY will have to be cleared manually.
14	ISO	0-1	Set this bit to enable the Receive endpoint for Isochronous transfers, and clear it to enable the Receive endpoint for Bulk/Interrupt transfers.
13	DMAEN	0-1	Set this bit to enable the DMA request for the Receive endpoints.
12	DISNYET	0	DISNYET: Applies only for Bulk/Interrupt Transactions: The CPU sets this bit to disable the sending of NYET handshakes. When set, all successfully received Rx packets are ACK'd including at the point at which the FIFO becomes full. Note: This bit only has any effect in high-speed mode, in which mode it should be set for all Interrupt endpoints.
		1	PID_ERROR: Applies only for ISO Transactions: The core sets this bit to indicate a PID error in the received packet.
11	DMAMODE	0-1	The CPU clears the DMAMODE bit prior to enabling the Rx DMA.
10-8	Reserved	0	Reserved.
7	CLRDATATOG	0-1	Write a 1 to this bit to reset the endpoint data toggle to 0.
6	SENTSTALL	0-1	This bit is set when a STALL handshake is transmitted. The FIFO is flushed and the TXPKTRDY bit is cleared. You should clear this bit.
5	SENDSTALL	0-1	Write a 1 to this bit to issue a STALL handshake. Clear this bit to terminate the stall condition. Note: This bit has no effect where the endpoint is being used for Isochronous transfers.
4	FLUSHFIFO	0-1	Write a 1 to this bit to flush the next packet to be read from the endpoint Receive FIFO. The FIFO pointer is reset and the RXPKTRDY bit is cleared. Note: FLUSHFIFO has no effect unless RXPKTRDY is set. Also note that, if the FIFO is double-buffered, FLUSHFIFO may need to be set twice to completely clear the FIFO.
3	DATAERROR	0-1	This bit is set when RXPKTRDY is set if the data packet has a CRC or bit-stuff error. It is cleared when RXPKTRDY is cleared. Note: This bit is only valid when the endpoint is operating in ISO mode. In Bulk mode, it always returns zero.
2	OVERRUN	0-1	This bit is set if an OUT packet cannot be loaded into the Receive FIFO. You should clear this bit. Note: This bit is only valid when the endpoint is operating in ISO mode. In Bulk mode, it always returns zero.
1	FIFOFULL	0-1	This bit is set when no more packets can be loaded into the Receive FIFO.
0	RXPKTRDY	0-1	This bit is set when a data packet has been received. You should clear this bit when the packet has been unloaded from the Receive FIFO. An interrupt is generated when the bit is set.

13.3.39 Count 0 Register (COUNT0)

The count 0 register (COUNT0) is shown in Figure 13-76 and described in Table 13-78.

Figure 13-76. Count 0 Register (COUNT0)

15 7	6 0
Reserved	EP0RXCOUNT
R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 13-78. Count 0 Register (COUNT0) Field Descriptions

Bit	Field	Value	Description
15-7	Reserved	0	Reserved.
6-0	EP0RXCOUNT	0-7Fh	Indicates the number of received data bytes in the Endpoint 0 FIFO. The value returned changes as the contents of the FIFO change and is only valid while RXPKTRDY of PERI_CSR0 is set.

13.3.40 Receive Count Register (RXCOUNT)

The receive count register (RXCOUNT) is shown in Figure 13-77 and described in Table 13-79.

Figure 13-77. Receive Count Register (RXCOUNT)

15 13	12 0
Reserved	EPRXCOUNT
R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 13-79. Receive Count Register (RXCOUNT) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12-0	EPRXCOUNT	0-1FFFh	Holds the number of received data bytes in the packet in the Receive FIFO. The value returned changes as the contents of the FIFO change and is only valid while RXPKTRDY of PERI_RXCSR or HOST_RXCSR is set.

13.3.41 Configuration Data Register (CONFIGDATA)

The configuration data register (CONFIGDATA) is shown in Figure 13-78 and described in Table 13-80.

Figure 13-78. Configuration Data Register (CONFIGDATA)

7	6	5	4	3	2	1	0
MPRXE	MPTXE	BIGENDIAN	HBRXE	HBTXE	DYNFIFO	SOFTCONE	UTMIDATAWIDTH
R-0	R-0	R-0	R-0	R-0	R-1	R-1	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 13-80. Configuration Data Register (CONFIGDATA) Field Descriptions

Bit	Field	Value	Description
7	MPRXE		Indicates automatic amalgamation of bulk packets.
		0	Automatic amalgamation of bulk packets is not selected.
		1	Automatic amalgamation of bulk packets is selected.
6	MPTXE		Indicates automatic splitting of bulk packets.
		0	Automatic splitting of bulk packets is not selected.
		1	Automatic splitting of bulk packets is selected.
5	BIGENDIAN		Indicates endian ordering.
		0	Little-endian ordering is selected.
		1	Big-endian ordering is selected.
4	HBRXE		Indicates high-bandwidth Rx ISO endpoint support.
		0	High-bandwidth Rx ISO endpoint support is not selected.
		1	High-bandwidth Rx ISO endpoint support is selected.
3	HBTXE		Indicates high-bandwidth Tx ISO endpoint support.
		0	High-bandwidth Tx ISO endpoint support is not selected.
		1	High-bandwidth Tx ISO endpoint support is selected.
2	DYNFIFO		Indicates dynamic FIFO sizing.
		0	Dynamic FIFO sizing option is not selected.
		1	Dynamic FIFO sizing option is selected.
1	SOFTCONE		Indicates soft connect/disconnect.
		0	Soft connect/disconnect option is not selected.
		1	Soft connect/disconnect option is selected.
0	UTMIDATAWIDTH		Indicates selected UTMI data width.
		0	8 bits.
		1	16 bits.

13.3.42 Transmit and Receive FIFO Registers for Endpoint 0 (FIFO0R1 and FIFO0R2)

The transmit and receive FIFO register 1 for endpoint 0 (FIFO0R1) is shown in Figure 13-79 and described in Table 13-81. The transmit and receive FIFO register 2 for endpoint 0 (FIFO0R2) is shown in Figure 13-80 and described in Table 13-82.

Figure 13-79. Transmit and Receive FIFO Register 1 for Endpoint 0 (FIFO0R1)

15	0
DATA	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Figure 13-80. Transmit and Receive FIFO Register 2 for Endpoint 0 (FIFO0R2)

15	0
DATA	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 13-81. Transmit and Receive FIFO Register 1 for Endpoint 0 (FIFO0R1) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to this address loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

Table 13-82. Transmit and Receive FIFO Register 2 for Endpoint 0 (FIFO0R2) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to this address loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

13.3.43 Transmit and Receive FIFO Registers for Endpoint 1 (FIFO1R1 and FIFO1R2)

The transmit and receive FIFO register 1 for endpoint 1 (FIFO1R1) is shown in Figure 13-81 and described in Table 13-83. The transmit and receive FIFO register 2 for endpoint 1 (FIFO1R2) is shown in Figure 13-82 and described in Table 13-84.

Figure 13-81. Transmit and Receive FIFO Register 1 for Endpoint 1 (FIFO1R1)

15 … 0
DATA
R/W-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Figure 13-82. Transmit and Receive FIFO Register 2 for Endpoint 1 (FIFO1R2)

15 … 0
DATA
R/W-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 13-83. Transmit and Receive FIFO Register 1 for Endpoint 1 (FIFO1R1) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

Table 13-84. Transmit and Receive FIFO Register 2 for Endpoint 1 (FIFO1R2) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

13.3.44 Transmit and Receive FIFO Registers for Endpoint 2 (FIFO2R1 and FIFO2R2)

The transmit and receive FIFO register 1 for endpoint 2 (FIFO2R1) is shown in Figure 13-83 and described in Table 13-85. The transmit and receive FIFO register 2 for endpoint 2 (FIFO2R2) is shown in Figure 13-84 and described in Table 13-86.

Figure 13-83. Transmit and Receive FIFO Register 1 for Endpoint 2 (FIFO2R1)

15 — 0
DATA
R/W-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Figure 13-84. Transmit and Receive FIFO Register 2 for Endpoint 2 (FIFO2R2)

15 — 0
DATA
R/W-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 13-85. Transmit and Receive FIFO Register 1 for Endpoint 2 (FIFO2R1) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

Table 13-86. Transmit and Receive FIFO Register 2 for Endpoint 2 (FIFO2R2) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

13.3.45 Transmit and Receive FIFO Registers for Endpoint 3 (FIFO3R1 and FIFO3R2)

The transmit and receive FIFO register 1 for endpoint 3 (FIFO3R1) is shown in Figure 13-85 and described in Table 13-87. The transmit and receive FIFO register 2 for endpoint 3 (FIFO3R2) is shown in Figure 13-86 and described in Table 13-88.

Figure 13-85. Transmit and Receive FIFO Register 1 for Endpoint 3 (FIFO3R1)

15	0
DATA	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Figure 13-86. Transmit and Receive FIFO Register 2 for Endpoint 3 (FIFO3R2)

15	0
DATA	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 13-87. Transmit and Receive FIFO Register 1 for Endpoint 3 (FIFO3R1) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

Table 13-88. Transmit and Receive FIFO Register 2 for Endpoint 3 (FIFO3R2) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

13.3.46 Transmit and Receive FIFO Registers for Endpoint 4 (FIFO4R1 and FIFO4R2)

The transmit and receive FIFO register 1 for endpoint 4 (FIFO4R1) is shown in Figure 13-87 and described in Table 13-89. The transmit and receive FIFO register 2 for endpoint 4 (FIFO4R2) is shown in Figure 13-88 and described in Table 13-90.

Figure 13-87. Transmit and Receive FIFO Register 1 for Endpoint 4 (FIFO4R1)

15	0
DATA	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Figure 13-88. Transmit and Receive FIFO Register 2 for Endpoint 4 (FIFO4R2)

15	0
DATA	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 13-89. Transmit and Receive FIFO Register 1 for Endpoint 4 (FIFO4R1) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

Table 13-90. Transmit and Receive FIFO Register 2 for Endpoint 4 (FIFO4R2) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

13.3.47 Device Control Register (DEVCTL)

The device control register (DEVCTL) is shown in Figure 13-89 and described in Table 13-91.

Figure 13-89. Device Control Register (DEVCTL)

7	6	5	4	3	2	1	0
BDEVICE	FSDEV	LSDEV	VBUS		HOSTMODE	Reserved	SESSION
R-0	R-0	R-0	R-0		R-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-91. Device Control Register (DEVCTL) Field Descriptions

Bit	Field	Value	Description
7	BDEVICE		This read-only bit indicates whether the USB controller is operating as the 'A' device or the 'B' device.
		0	A device.
		1	B device.
			Only valid while a session is in progress.
6	FSDEV	0-1	This read-only bit is set when a full-speed or high-speed device has been detected being connected to the port (high-speed devices are distinguished from full-speed by checking for high-speed chirps when the device is reset). Only valid in Host mode. Host mode is not supported on the device.
5	LSDEV	0-1	This read-only bit is set when a low-speed device has been detected being connected to the port. Only valid in Host mode. Host mode is not supported on the device.
	VBUS	0-3h	These read-only bits encode the current VBus level as follows:
		0	Below Session End.
		1 h	Above Session End, below AValid.
		2h	Above AValid, below VBusValid.
		3h	Above VBusValid.
2	HOSTMODE	0-1	This read-only bit is set when the USB controller is acting as a Host. Host mode is not supported on the device.
1	Reserved	0	Reserved.
0	SESSION	0-1	When operating as an 'A' device, you must set or clear this bit start or end a session. When operating as a 'B' device, this bit is set/cleared by the USB controller when a session starts/ends. You must also set this bit to initiate the Session Request Protocol. When the USB controller is in Suspend mode, you may clear the bit to perform a software disconnect. A special software routine is required to perform SRP. Details will be made available in a later document version.

13.3.48 Transmit Endpoint FIFO Size (TXFIFOSZ)

Section 13.2.7 describes dynamically setting endpoint FIFO sizes. The option of dynamically setting endpoint FIFO sizes only applies to Endpoints 1-4. The Endpoint 0 FIFO has a fixed size (64 bytes) and a fixed location (start address 0). It is the responsibility of the firmware to ensure that all the Tx and Rx endpoints that are active in the current USB configuration have a block of RAM assigned exclusively to that endpoint. The RAM must be at least as large as the maximum packet size set for that endpoint.

The transmit endpoint FIFO size (TXFIFOSZ) is shown in Figure 13-90 and described in Table 13-92.

Figure 13-90. Transmit Endpoint FIFO Size (TXFIFOSZ)

7 5	4	3 0
Reserved	DPB	SZ
R-0	R/W-0	R-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-92. Transmit Endpoint FIFO Size (TXFIFOSZ) Field Descriptions

Bit	Field	Value	Description
7-5	Reserved	0	Reserved.
4	DPB		Double packet buffering enable.
		0	Single packet buffering is supported.
		1	Double packet buffering is enabled.
3-0	SZ	0-Fh	Maximum packet size to be allowed (before any splitting within the FIFO of Bulk packets prior to transmission). If m = SZ, the FIFO size is calculated as $2^{(m+3)}$ for single packet buffering and $2^{(m+4)}$ for dual packet buffering.

13.3.49 Receive Endpoint FIFO Size (RXFIFOSZ)

Section 13.2.7 describes dynamically setting endpoint FIFO sizes. The option of dynamically setting endpoint FIFO sizes only applies to Endpoints 1-4. The Endpoint 0 FIFO has a fixed size (64 bytes) and a fixed location (start address 0). It is the responsibility of the firmware to ensure that all the Tx and Rx endpoints that are active in the current USB configuration have a block of RAM assigned exclusively to that endpoint. The RAM must be at least as large as the maximum packet size set for that endpoint.

The receive endpoint FIFO size (RXFIFOSZ) is shown in Figure 13-91 and described in Table 13-93.

Figure 13-91. Receive Endpoint FIFO Size (RXFIFOSZ)

7 5	4	3 0
Reserved	DPB	SZ
R-0	R/W-0	R-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-93. Receive Endpoint FIFO Size (RXFIFOSZ) Field Descriptions

Bit	Field	Value	Description
7-5	Reserved	0	Reserved.
4	DPB		Double packet buffering enable.
		0	Single packet buffering is supported.
		1	Double packet buffering is enabled.
3-0	SZ	0-Fh	Maximum packet size to be allowed (before any splitting within the FIFO of Bulk packets prior to transmission). If m = SZ, the FIFO size is calculated as $2^{(m+3)}$ for single packet buffering and $2^{(m+4)}$ for dual packet buffering.

13.3.50 Transmit Endpoint FIFO Address (TXFIFOADDR)

The transmit endpoint FIFO address (TXFIFOADDR) is shown in Figure 13-92 and described in Table 13-94.

Figure 13-92. Transmit Endpoint FIFO Address (TXFIFOADDR)

15 13	12 0
Reserved	ADDR
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-94. Transmit Endpoint FIFO Address (TXFIFOADDR) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12-0	ADDR	0-1FFFh	Start Address of endpoint FIFO in units of 8 bytes. If m = ADDR, then the start address is 8 × m.

13.3.51 Hardware Version Register (HWVERS)

The hardware version register (HWVERS) contains the RTL major and minor version numbers for the USB 2.0 controller module. The RTL version number is REVMAJ.REVMIN. The HWVERS is shown in Figure 13-93 and described in Table 13-95.

Figure 13-93. Hardware Version Register (HWVERS)

15	14 10	9 0
RC	REVMAJ	REVMIN
R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 13-95. Hardware Version Register (HWVERS) Field Descriptions

Bit	Field	Value	Description
15	RC	0-1	Set to 1 if RTL is used from a Release Candidate, rather than from a full release of the core.
14-10	REVMAJ	0-1Fh	Major version of RTL. Range is 0-3.1.
9-0	REVMIN	0-3E7h	Minor version of RTL. Range is 0-999.

13.3.52 Receive Endpoint FIFO Address (RXFIFOADDR)

The receive endpoint FIFO address (RXFIFOADDR) is shown in Figure 13-94 and described in Table 13-96.

Figure 13-94. Receive Endpoint FIFO Address (RXFIFOADDR)

15 13	12 0
Reserved	ADDR
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-96. Receive Endpoint FIFO Address (RXFIFOADDR) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12-0	ADDR	0-1FFFh	Start Address of endpoint FIFO in units of 8 bytes. If m = ADDR, then the start address is 8 × m.

13.3.53 CDMA Revision Identification Registers (DMAREVID1 and DMAREVID2)

The CDMA revision identification registers (DMAREVID1 and DMAREVID2) contain the revision for the module. The DMAREVID1 is shown in Figure 13-95 and described in Table 13-97. The DMAREVID2 is shown in Figure 13-96 and described in Table 13-98.

Figure 13-95. CDMA Revision Identification Register 1 (DMAREVID1)

15 0
REV_LSB
R-1900h

LEGEND: R = Read only; -*n* = value after reset

Figure 13-96. CDMA Revision Identification Register 2 (DMAREVID2)

15 0
REV_MSB
R-0053h

LEGEND: R = Read only; -*n* = value after reset

Table 13-97. CDMA Revision Identification Register 1 (DMAREVID1) Field Descriptions

Bit	Field	Value	Description
15-0	REV_LSB	0-FFFFh	Revision ID of the CPPI DMA (CDMA) module. Least significant bits.

Table 13-98. CDMA Revision Identification Register 2 (DMAREVID2) Field Descriptions

Bit	Field	Value	Description
15-0	REV_MSB	0-FFFFh	Revision ID of the CPPI DMA (CDMA) module. Most significant bits.

13.3.54 CDMA Teardown Free Descriptor Queue Control Register (TDFDQ)

The CDMA teardown free descriptor queue control register (TDFDQ) is used to inform the DMA of the location in memory or descriptor array which is to be used for signaling of a teardown complete for each transmit and receive channel. The CDMA teardown free descriptor queue control register (TDFDQ) is shown in Figure 13-97 and described in Table 13-99.

Figure 13-97. CDMA Teardown Free Descriptor Queue Control Register (TDFDQ)

15 14	13 12	11 0
Reserved	TD_DESC_QMGR	TD_DESC_QNUM
R-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-99. CDMA Teardown Free Descriptor Queue Control Register (TDFDQ) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-12	TD_DESC_QMGR	0-3h	Controls which of the four queue managers the DMA accesses to allocate a channel teardown descriptor from the teardown descriptor queue.
11-0	TD_DESC_QNUM	0-FFFh	Controls which of the 2K queues in the indicated queue manager should be read to allocate the channel teardown descriptors.

13.3.55 CDMA Emulation Control Register (DMAEMU)

The CDMA emulation controls the behavior of the DMA when an emulation suspend signal is asserted. The CDMA emulation control register (DMAEMU) is shown in Figure 13-98 and described in Table 13-100.

Figure 13-98. CDMA Emulation Control Register (DMAEMU)

15 2	1	0
Reserved	SOFT	FREE
R-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-100. CDMA Emulation Control Register (DMAEMU) Field Descriptions

Bit	Field	Value	Description
15-2	Reserved	0	Reserved.
1	SOFT	0-1	
0	FREE	0-1	

13.3.56 CDMA Transmit Channel n Global Configuration Registers (TXGCR1[*n*] and TXGCR2[*n*])

The transmit channel *n* configuration registers (TXGCR2[*n*] and TXGCR1[*n*]) initialize the behavior of each of the transmit DMA channels. There are four configuration register pairs, one for each transmit DMA channel.

The transmit channel *n* configuration registers TXGCR1[*n*]) and (TXGCR2[*n*] are shown in Figure 13-99 and Figure 13-100and described in Table 13-101 and Table 13-102. .

Figure 13-99. CDMA Transmit Channel *n* Global Configuration Register 1 (TXGCR1[*n*])

15 14	13 12	11 0
Reserved	TX_DEFAULT_QMGR	TX_DEFAULT_QNUM
R-0	W-0	W-0

LEGEND: R/W = Read/Write; R = Read only; W = Write only; -*n* = value after reset

Figure 13-100. CDMA Transmit Channel *n* Global Configuration Register 2 (TXGCR2[*n*])

15	14	13 0
TX_ENABLE	TX_TEARDOWN	Reserved
R/W-0	R/W-0	R-0

LEGEND: R/W = Read/Write; R = Read only; W = Write only; -*n* = value after reset

Table 13-101. CDMA Transmit Channel *n* Global Configuration Register 1 (TXGCR1[*n*]) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-12	TX_DEFAULT_QMGR	0-3h	Controls the default queue manager number that is used to queue teardown descriptors back to the host.
11-0	TX_DEFAULT_QNUM	0-FFFh	Controls the default queue number within the selected queue manager onto which teardown descriptors are queued back to the host.

Table 13-102. CDMA Transmit Channel *n* Global Configuration Register 2 (TXGCR2[*n*]) Field Descriptions

Bit	Field	Value	Description
15	TX_ENABLE		Channel control. The TX_ENABLE field is cleared after a channel teardown is complete.
		0	Disables channel.
		1	Enables channel.
14	TX_TEARDOWN	0-1	Setting this bit requests the channel to be torn down. The TX_TEARDOWN field remains set after a channel teardown is complete.
13-0	Reserved	0	Reserved.

13.3.57 CDMA Receive Channel n Global Configuration Registers (RXGCR1[n] and RXGCR2[n])

The receive channel *n* global configuration registers (RXGCR1[*n*] and RXGCR2[*n*]) initialize the global (non-descriptor-type specific) behavior of each of the receive DMA channels. There are four configuration register pairs, one for each receive DMA channel. If the enable bit is being set, the receive channel *n* global configuration register should only be written after all of the other receive configuration registers have been initialized.

The receive channel *n* global configuration registers (RXGCR1[*n*] and RXGCR2[*n*]) are shown in Figure 13-101 and Figure 13-102 and are described in Table 13-103 and Table 13-104.

Figure 13-101. CDMA Receive Channel *n* Global Configuration Register 1 (RXGCR1[*n*])

15 14	13 12	11 0
RX_DEFAULT_DESC_TYPE	RX_DEFAULT_RQ_QMGR	RX_DEFAULT_RQ_QNUM
R-0	W-0	W-0

LEGEND: R/W = Read/Write; R = Read only; W = Write only; -*n* = value after reset

Figure 13-102. CDMA Receive Channel *n* Global Configuration Register 2 (RXGCR2[*n*])

15	14	13 9	8	7 0
RX_ENABLE	RX_TEARDOWN	Reserved	RX_ERROR_HANDLING	RX_SOP_OFFSET
R/W-0	R/W-0	R-0	W-0	W-0

LEGEND: R/W = Read/Write; R = Read only; W = Write only; -*n* = value after reset

Table 13-103. CDMA Receive Channel *n* Global Configuration Register 1 (RXGCR1[*n*]) Field Descriptions

Bit	Field	Value	Description
15-14	RX_DEFAULT_DESC_TYPE	0-3h	Indicates the default descriptor type to use. The actual descriptor type that is used for reception can be overridden by information provided in the CPPI FIFO data block.
		0	Reserved.
		1h	Host.
		2h-3h	Reserved.
13-12	RX_DEFAULT_RQ_QMGR	0-3h	Indicates the default receive queue manager that this channel should use. The actual receive queue manager index can be overridden by information provided in the CPPI FIFO data block.
11-0	RX_DEFAULT_RQ_QNUM	0-FFFh	Indicates the default receive queue that this channel should use. The actual receive queue that is used for reception can be overridden by information provided in the CPPI FIFO data block.

Table 13-104. CDMA Receive Channel *n* Global Configuration Register 2 (RXGCR2[*n*]) Field Descriptions

Bit	Field	Value	Description
15	RX_ENABLE		Channel control. Field is cleared after a channel teardown is complete.
		0	Disables channel.
		1	Enables channel.
14	RX_TEARDOWN	0-1	Indicates whether a receive operation is complete. Field should be cleared when a channel is initialized. Field is set after a channel teardown is complete.
13-9	Reserved	0	Reserved.

Table 13-104. CDMA Receive Channel *n* Global Configuration Register 2 (RXGCR2[*n*]) Field Descriptions (continued)

Bit	Field	Value	Description
8	RX_ERROR_HANDLING		Controls the error handling mode for the channel and is only used when channel errors (i.e. descriptor or buffer starvation occur):
		0	Starvation errors result in dropping packet and reclaiming any used descriptor or buffer resources back to the original queues/pools they were allocated to.
		1	Starvation errors result in subsequent retry of the descriptor allocation operation. In this mode, the DMA will return to the IDLE state without saving its internal operational state back to the internal state RAM and without issuing an advance operation on the FIFO interface. This results in the DMA re-initiating the FIFO block transfer at a later time with the intention that additional free buffers and/or descriptors will have been added.
7-0	RX_SOP_OFFSET	0–FFh	Specifies the number of bytes that are to be skipped in the SOP buffer before beginning to write the payload. This value must be less than the minimum size of a buffer in the system.

13.3.58 CDMA Receive Channel n Host Packet Configuration Registers A (RXHPCR1A[n] and RXHPCR2A[n])

The receive channel *n* host packet configuration registers A (RXHPCR1A[n] and RXHPCR2A[n]) initialize the behavior of each of the receive DMA channels for reception of host type packets. There are four configuration A registers, one for each receive DMA channel.

The receive channel *n* host packet configuration register 1 A (RXHPCR1A[n]) are shown in Figure 13-103 and described in Table 13-105. The receive channel *n* host packet configuration register 2 A (RXHPCR2A[n]) is shown in Figure 13-104 and described in Table 13-106.

Figure 13-103. Receive Channel *n* Host Packet Configuration Register 1 A (RXHPCR1A[*n*])

15 14	13 12	11 0
Reserved	RX_HOST_FDQ0_QMGR	RX_HOST_FDQ0_QNUM
R-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Figure 13-104. Receive Channel *n* Host Packet Configuration Register 2 A (RXHPCR2A[*n*])

15 14	13 12	11 0
Reserved	RX_HOST_FDQ1_QMGR	RX_HOST_FDQ1_QNUM
R-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Table 13-105. Receive Channel *n* Host Packet Configuration Register 1 A (RXHPCR1A[*n*]) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-12	RX_HOST_FDQ0_QMGR	0-3h	Specifies which buffer manager should be used for the first receive buffer in a host type packet.
11-0	RX_HOST_FDQ0_QNUM	0-FFFh	Specifies which free descriptor/buffer pool should be used for the first receive buffer in a host type packet.

Table 13-106. Receive Channel *n* Host Packet Configuration Register 2 A (RXHPCR2A[*n*]) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-12	RX_HOST_FDQ1_QMGR	0-3h	Specifies which buffer manager should be used for the second receive buffer in a host type packet.
11-0	RX_HOST_FDQ1_QNUM	0-FFFh	Specifies which free descriptor/buffer pool should be used for the second receive buffer in a host type packet.

13.3.59 CDMA Receive Channel n Host Packet Configuration Registers B (RXHPCR1B[n] and RXHPCR2B[n])

The receive channel *n* host packet configuration registers B (RXHPCR1B[n] and RXHPCR2B[n]) initialize the behavior of each of the receive DMA channels for reception of host type packets. There are four configuration B register pairs, one for each receive DMA channel.

The receive channel *n* host packet configuration register 1 B (RXHPCR1B[*n*]) is shown in Figure 13-105 and described in Table 13-107. The receive channel *n* host packet configuration register 2 B (RXHPCR2B[*n*]) is shown in Figure 13-106 and described in Table 13-108.

Figure 13-105. Receive Channel *n* Host Packet Configuration Register 1 B (RXHPCR1B[*n*])

15 14	13 12	11 0
Reserved	RX_HOST_FDQ2_QMGR	RX_HOST_FDQ2_QNUM
R-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Figure 13-106. Receive Channel *n* Host Packet Configuration Register 2 B (RXHPCR2B[*n*])

15 14	13 12	11 0
Reserved	RX_HOST_FDQ3_QMGR	RX_HOST_FDQ3_QNUM
R-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Table 13-107. Receive Channel *n* Host Packet Configuration Register 1 B (RXHPCR1B[*n*]) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-12	RX_HOST_FDQ2_QMGR	0-3h	Specifies which buffer manager should be used for the third receive buffer in a host type packet.
11-0	RX_HOST_FDQ2_QNUM	0-FFFh	Specifies which free descriptor/buffer pool should be used for the third receive buffer in a host type packet.

Table 13-108. Receive Channel *n* Host Packet Configuration Register 2 B (RXHPCR2B[*n*]) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-12	RX_HOST_FDQ3_QMGR	0-3h	Specifies which buffer manager should be used for the fourth or later receive buffer in a host type packet.
11-0	RX_HOST_FDQ3_QNUM	0-FFFh	Specifies which free descriptor/buffer pool should be used for the fourth or later receive buffer in a host type packet.

13.3.60 CDMA Scheduler Control Register (DMA_SCHED_CTRL1 and DMA_SCHED_CTRL2)

The CDMA scheduler control registers (DMA_SCHED_CTRL1 and DMA_SCHED_CTRL2) enable the scheduler and indicate the last entry in the scheduler table. The CDMA scheduler control register 1 (DMA_SCHED_CTRL1) is shown in Figure 13-107 and described in Table 13-109. The CDMA scheduler control register 2 (DMA_SCHED_CTRL2) is shown in Figure 13-108 and described in Table 13-110.

Figure 13-107. CDMA Scheduler Control Register 1 (DMA_SCHED_CTRL1)

15 8	7 0
Reserved	LAST_ENTRY
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 13-108. CDMA Scheduler Control Register 2 (DMA_SCHED_CTRL2)

15	14 0
ENABLE	Reserved
R/W-0	R-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-109. CDMA Scheduler Control Register 1 (DMA_SCHED_CTRL1) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved.
7-0	LAST_ENTRY	0-FFh	Indicates the last valid entry in the scheduler table. There are 64 words in the table and there are 4 entries in each word. The table can be programmed with any integer number of entries from 1 to 256. The corresponding encoding for this field is as follows:
		0	1 entry.
		1h	2 entries.
		2h-FFh	3 entries to 256 entries.

Table 13-110. CDMA Scheduler Control Register 2 (DMA_SCHED_CTRL2) Field Descriptions

Bit	Field	Value	Description
15	ENABLE		This is the enable bit for the scheduler and is encoded as follows:
		0	Scheduler is disabled and will no longer fetch entries from the scheduler table or pass credits to the DMA controller.
		1	Scheduler is enabled. This bit should only be set after the table has been initialized.
14-0	Reserved	0	Reserved.

13.3.61 CDMA Scheduler Table Word n Registers (ENTRYLSW[n]-ENTRYMSW[n])

The CDMA scheduler table word *n* registers (ENTRYLSW[n]-ENTRYMSW[n]) provide information about the scheduler. The CDMA scheduler table word *n* registers (ENTRYLSW[*n*]) are shown in Figure 13-109 and described in Table 13-111. The CDMA scheduler table word *n* registers (ENTRYMSW[*n*]) are shown in Figure 13-110 and described in Table 13-112.

Figure 13-109. CDMA Scheduler Table Word *n* Registers (ENTRYLSW[*n*])

15	14 12	11 8	7	6 4	3 0
ENTRY1_RXTX	Reserved	ENTRY1_CHANNEL	ENTRY0_RXTX	Reserved	ENTRY0_CHANNEL
W-0	R-0	W-0	W-0	R-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Figure 13-110. CDMA Scheduler Table Word *n* Registers (ENTRYMSW[*n*])

15	14 12	11 8	7	6 4	3 0
ENTRY3_RXTX	Reserved	ENTRY3_CHANNEL	ENTRY2_RXTX	Reserved	ENTRY2_CHANNEL
W-0	R-0	W-0	W-0	R-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Table 13-111. CDMA Scheduler Table Word *n* Registers (ENTRYLSW[*n*]) Field Descriptions

Bit	Field	Value	Description
15	ENTRY1_RXTX		This entry is for a transmit or a receive channel.
		0	Transmit channel
		1	Receive channel
14-12	Reserved	0	Reserved
11-8	ENTRY1_CHANNEL	0-Fh	Indicates the channel number that is to be given an opportunity to transfer data. If this is a transmit entry, the DMA will be presented with a scheduling credit for that exact transmit channel. If this is a receive entry, the DMA will be presented with a scheduling credit for the receive FIFO that is associated with this channel. For receive FIFOs which carry traffic for more than one receive DMA channel, the exact channel number that is given in the receive credit will actually be the channel number which is currently on the head element of that Rx FIFO, which is not necessarily the channel number given in the scheduler table entry.
7	ENTRY0_RXTX		This entry is for a transmit or a receive channel.
		0	Transmit channel
		1	Receive channel
6-4	Reserved	0	Reserved
3-0	ENTRY0_CHANNEL	0-Fh	Indicates the channel number that is to be given an opportunity to transfer data. If this is a transmit entry, the DMA will be presented with a scheduling credit for that exact transmit channel. If this is a receive entry, the DMA will be presented with a scheduling credit for the receive FIFO that is associated with this channel. For receive FIFOs which carry traffic for more than one receive DMA channel, the exact channel number that is given in the receive credit will actually be the channel number which is currently on the head element of that Rx FIFO, which is not necessarily the channel number given in the scheduler table entry.

Table 13-112. CDMA Scheduler Table Word *n* Registers (ENTRYMSW[*n*]) Field Descriptions

Bit	Field	Value	Description
15	ENTRY3_RXTX		This entry is for a transmit or a receive channel.
		0	Transmit channel
		1	Receive channel
14-12	Reserved	0	Reserved

Table 13-112. CDMA Scheduler Table Word *n* Registers (ENTRYMSW[*n*]) Field Descriptions (continued)

Bit	Field	Value	Description
11-8	ENTRY3_CHANNEL	0-Fh	Indicates the channel number that is to be given an opportunity to transfer data. If this is a transmit entry, the DMA will be presented with a scheduling credit for that exact transmit channel. If this is a receive entry, the DMA will be presented with a scheduling credit for the receive FIFO that is associated with this channel. For receive FIFOs which carry traffic for more than one receive DMA channel, the exact channel number that is given in the receive credit will actually be the channel number which is currently on the head element of that Rx FIFO, which is not necessarily the channel number given in the scheduler table entry.
7	ENTRY2_RXTX		This entry is for a transmit or a receive channel.
		0	Transmit channel
		1	Receive channel
6-4	Reserved	0	Reserved
3-0	ENTRY2_CHANNEL	0-Fh	Indicates the channel number that is to be given an opportunity to transfer data. If this is a transmit entry, the DMA will be presented with a scheduling credit for that exact transmit channel. If this is a receive entry, the DMA will be presented with a scheduling credit for the receive FIFO that is associated with this channel. For receive FIFOs which carry traffic for more than one receive DMA channel, the exact channel number that is given in the receive credit will actually be the channel number which is currently on the head element of that Rx FIFO, which is not necessarily the channel number given in the scheduler table entry.

13.3.62 Queue Manager Revision Identification Registers (QMGRREVID1 and QMGRREVID2)

The queue manager revision identification registers (QMGRREVID1 and QMGRREVID2) contain the major and minor revisions for the module. The QMGRREVID1 is shown in Figure 13-111 and described in Table 13-113. The QMGRREVID2 is shown in Figure 13-112 and described in Table 13-114.

Figure 13-111. Queue Manager Revision Identification Register 1 (QMGRREVID1)

15 … 0
REV_LSB
R-1200h

LEGEND: R = Read only; -*n* = value after reset

Figure 13-112. Queue Manager Revision Identification Register 2 (QMGRREVID2)

15 … 0
REV_MSB
R-0052h

LEGEND: R = Read only; -*n* = value after reset

Table 13-113. Queue Manager Revision Identification Register 1 (QMGRREVID1) Field Descriptions

Bit	Field	Value	Description
15-0	REV_LSB	0-FFFFh	Revision ID of the queue manager. Least-significant bits.

Table 13-114. Queue Manager Revision Identification Register 2 (QMGRREVID2) Field Descriptions

Bit	Field	Value	Description
15-0	REV_MSB	0-FFFFh	Revision ID of the queue manager. Most-significant bits.

13.3.63 Queue Manager Queue Diversion Registers (DIVERSION1 and DIVERSION2)

The queue manager queue diversion registers (DIVERSION1 and DIVERSION2) are used to transfer the contents of one queue onto another queue. It does not support byte accesses. The queue manager queue diversion register 1 (DIVERSION1) is shown in Figure 13-113 and described in Table 13-115. The queue manager queue diversion register 2 (DIVERSION2) is shown in Figure 13-114 and described in Table 13-116.

Figure 13-113. Queue Manager Queue Diversion Register 1 (DIVERSION1)

15 14	13 0
Reserved	SOURCE_QNUM
R-0	W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 13-114. Queue Manager Queue Diversion Register 2 (DIVERSION2)

15	14	13 0
HEAD_TAIL	Reserved	DEST_QNUM
W-0	R-0	W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-115. Queue Manager Queue Diversion Register 1 (DIVERSION1) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-0	SOURCE_QNUM	0-3FFFh	Source Queue Number.

Table 13-116. Queue Manager Queue Diversion Register 2 (DIVERSION2 Field Descriptions

Bit	Field	Value	Description
15	HEAD_TAIL		Indicates whether queue contents should be merged on to the head or tail of the destination queue.
		0	Head.
		1	Tail.
14	Reserved	0	Reserved.
13-0	DEST_QNUM	0-3FFFh	Destination Queue Number.

13.3.64 Queue Manager Free Descriptor/Buffer Starvation Count Register 0 (FDBSC0)

The free descriptor/buffer queue starvation count register (FDBSC0) provides statistics about how many starvation events are occurring on the receive free descriptor/buffer queues. These registers do not support byte accesses. The free descriptor/buffer queue starvation count register (FDBSC0) is shown in Figure 13-115 and described in Table 13-117.

Figure 13-115. Queue Manager Free Descriptor/Buffer Starvation Count Register 0 (FDBSC0)

15 8	7 0
FDBQ1_STARVE_CNT	FDBQ0_STARVE_CNT
RC-0	RC-0

LEGEND: RC = Cleared on read; -*n* = value after reset

Table 13-117. Queue Manager Free Descriptor/Buffer Starvation Count Register 0 (FDBSC0) Field Descriptions

Bit	Field	Value	Description
15-8	FDBQ1_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 1 is read while it is empty. This field is cleared when read by CPU.
7-0	FDBQ0_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 0 is read while it is empty. This field is cleared when read by CPU.

13.3.65 Queue Manager Free Descriptor/Buffer Starvation Count Register 1 (FDBSC1)

The free descriptor/buffer queue starvation count register (FDBSC1) provides statistics about how many starvation events are occurring on the receive free descriptor/buffer queues. These registers do not support byte accesses. The free descriptor/buffer queue starvation count register (FDBSC1) is shown in Figure 13-116 and described in Table 13-118.

Figure 13-116. Queue Manager Free Descriptor/Buffer Starvation Count Register 1 (FDBSC1)

15 8	7 0
FDBQ3_STARVE_CNT	FDBQ2_STARVE_CNT
RC-0	RC-0

LEGEND: RC = Cleared on read; -*n* = value after reset

Table 13-118. Queue Manager Free Descriptor/Buffer Starvation Count Register 1 (FDBSC1) Field Descriptions

Bit	Field	Value	Description
15-8	FDBQ3_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 3 is read while it is empty. This field is cleared when readby CPU.
7-0	FDBQ2_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 2 is read while it is empty. This field is cleared when readby CPU.

13.3.66 Queue Manager Free Descriptor/Buffer Starvation Count Register 2 (FDBSC2)

The free descriptor/buffer queue starvation count register 2 (FDBSC2) provides statistics about how many starvation events are occurring on the receive free descriptor/buffer queues. These registers do not support byte accesses. The free descriptor/buffer queue starvation count register 2 (FDBSC2) is shown in Figure 13-117 and described in Table 13-119.

Figure 13-117. Queue Manager Free Descriptor/Buffer Starvation Count Register 2 (FDBSC2)

15 8	7 0
FDBQ5_STARVE_CNT	FDBQ4_STARVE_CNT
RC-0	RC-0

LEGEND: RC = Cleared on read; -*n* = value after reset

Table 13-119. Queue Manager Free Descriptor/Buffer Starvation Count Register 2 (FDBSC2) Field Descriptions

Bit	Field	Value	Description
15-8	FDBQ5_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 5 is read while it is empty. This field is cleared when read by CPU.
7-0	FDBQ4_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 4 is read while it is empty. This field is cleared when read by CPU.

13.3.67 Queue Manager Free Descriptor/Buffer Starvation Count Register 3 (FDBSC3)

The free descriptor/buffer queue starvation count register 3 (FDBSC3) provides statistics about how many starvation events are occurring on the receive free descriptor/buffer queues. These registers do not support byte accesses. The free descriptor/buffer queue starvation count register 3 (FDBSC3) is shown in Figure 13-118 and described in Table 13-120.

Figure 13-118. Queue Manager Free Descriptor/Buffer Starvation Count Register 3 (FDBSC3)

15 8	7 0
FDBQ7_STARVE_CNT	FDBQ6_STARVE_CNT
RC-0	RC-0

LEGEND: RC = Cleared on read; -*n* = value after reset

Table 13-120. Queue Manager Free Descriptor/Buffer Starvation Count Register 3 (FDBSC3) Field Descriptions

Bit	Field	Value	Description
15-8	FDBQ7_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 7 is read while it is empty. This field is cleared when read by CPU.
7-0	FDBQ6_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 6 is read while it is empty. This field is cleared when read by CPU.

13.3.68 Queue Manager Free Descriptor/Buffer Starvation Count Register 4 (FDBSC4)

The free descriptor/buffer queue starvation count register 4 (FDBSC4) provides statistics about how many starvation events are occurring on the receive free descriptor/buffer queues. These registers do not support byte accesses. The free descriptor/buffer queue starvation count register 4 (FDBSC4) is shown in Figure 13-119 and described in Table 13-121.

Figure 13-119. Queue Manager Free Descriptor/Buffer Starvation Count Register 4 (FDBSC4)

15 8	7 0
FDBQ9_STARVE_CNT	FDBQ8_STARVE_CNT
RC-0	RC-0

LEGEND: RC = Cleared on read; -*n* = value after reset

Table 13-121. Queue Manager Free Descriptor/Buffer Starvation Count Register 4 (FDBSC4) Field Descriptions

Bit	Field	Value	Description
15-8	FDBQ9_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 9 is read while it is empty. This field is cleared when read by CPU.
7-0	FDBQ8_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 8 is read while it is empty. This field is cleared when read by CPU.

13.3.69 Queue Manager Free Descriptor/Buffer Starvation Count Register 5 (FDBSC5)

The free descriptor/buffer queue starvation count register 5 (FDBSC5) provides statistics about how many starvation events are occurring on the receive free descriptor/buffer queues. These registers do not support byte accesses. The free descriptor/buffer queue starvation count register 5 (FDBSC5) is shown in Figure 13-120 and described in Figure 13-120.

Figure 13-120. Queue Manager Free Descriptor/Buffer Starvation Count Register 5 (FDBSC5)

15 8	7 0
FDBQ11_STARVE_CNT	FDB10_STARVE_CNT
RC-0	RC-0

LEGEND: RC = Cleared on read; -*n* = value after reset

Table 13-122. Queue Manager Free Descriptor/Buffer Starvation Count Register 5 (FDBSC5) Field Descriptions

Bit	Field	Value	Description
15-8	FDBQ11_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 11 is read while it is empty. This field is cleared when read by CPU.
7-0	FDBQ10_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 10 is read while it is empty. This field is cleared when read by CPU.

13.3.70 Queue Manager Free Descriptor/Buffer Starvation Count Register 6 (FDBSC6)

The free descriptor/buffer queue starvation count register 6 (FDBSC6) provides statistics about how many starvation events are occurring on the receive free descriptor/buffer queues. These registers do not support byte accesses. The free descriptor/buffer queue starvation count register 6 (FDBSC6) is shown in Figure 13-121 and described in Table 13-123.

Figure 13-121. Queue Manager Free Descriptor/Buffer Starvation Count Register 6 (FDBSC6)

15 8	7 0
FDBQ13_STARVE_CNT	FDBQ12_STARVE_CNT
RC-0	RC-0

LEGEND: RC = Cleared on read; -*n* = value after reset

Table 13-123. Queue Manager Free Descriptor/Buffer Starvation Count Register 6 (FDBSC6) Field Descriptions

Bit	Field	Value	Description
15-8	FDBQ13_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 13 is read while it is empty. This field is cleared when read by CPU.
7-0	FDBQ12_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 12 is read while it is empty. This field is cleared when read by CPU.

13.3.71 Queue Manager Free Descriptor/Buffer Starvation Count Register 7 (FDBSC7)

The free descriptor/buffer queue starvation count register 7 (FDBSC7) provides statistics about how many starvation events are occurring on the receive free descriptor/buffer queues. The registers do not support byte accesses. The free descriptor/buffer queue starvation count register 7 (FDBSC7) is shown in Figure 13-122 and described in Table 13-124.

Figure 13-122. Queue Manager Free Descriptor/Buffer Starvation Count Register 7 (FDBSC7)

15 8	7 0
FDBQ15_STARVE_CNT	FDB14_STARVE_CNT
RC-0	RC-0

LEGEND: RC = Cleared on read; -*n* = value after reset

Table 13-124. Queue Manager Free Descriptor/Buffer Starvation Count Register 7 (FDBSC7) Field Descriptions

Bit	Field	Value	Description
15-8	FDBQ15_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 15 is read while it is empty. This field is cleared when read by CPU.
7-0	FDBQ14_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 14 is read while it is empty. This field is cleared when read by CPU.

13.3.72 Queue Manager Linking RAM Region 0 Base Address Registers (LRAM0BASE1 and LRAM0BASE2)

The queue manager linking RAM region 0 base address registers (LRAM0BASE1 and LRAM0BASE2) set the base address for the first portion of the Linking RAM. This address must be 32-bit aligned. It is used by the Queue Manager to calculate the 32-bit linking address for a given descriptor index. These registers do not support byte accesses.

The queue manager linking RAM region 0 base address register 1 (LRAM0BASE1) is shown in Figure 13-123 and described in Table 13-125. The queue manager linking RAM region 0 base address register 2 (LRAM0BASE2) is shown in Figure 13-124 and described in Table 13-126.

Figure 13-123. Queue Manager Linking RAM Region 0 Base Address Register 1 (LRAM0BASE1)

15 … 0
REGION0_BASE_LSB
R/W-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Figure 13-124. Queue Manager Linking RAM Region 0 Base Address Register 2 (LRAM0BASE2)

15 … 0
REGION0_BASE_MSB
R/W-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 13-125. Queue Manager Linking RAM Region 0 Base Address Register 1 (LRAM0BASE1) Field Descriptions

Bit	Field	Value	Description
15-0	REGION0_BASE_LSB	0-FFFFh	This field stores the 16 least significant bits of the base address for the first region of the linking RAM. This may be anywhere in 32-bit address space but would be typically located in on-chip memory.

Table 13-126. Queue Manager Linking RAM Region 0 Base Address Register 2 (LRAM0BASE2) Field Descriptions

Bit	Field	Value	Description
15-0	REGION0_BASE_MSB	0-FFFFh	This field stores the 16 most significant bits of the base address for the first region of the linking RAM. This may be anywhere in 32-bit address space but would be typically located in on-chip memory.

13.3.73 Queue Manager Linking RAM Region 0 Size Register (LRAM0SIZE)

The queue manager linking RAM region 0 size register (LRAM0SIZE) sets the size of the array of linking pointers that are located in Region 0 of Linking RAM. The size specified the number of descriptors for which linking information is stored in this region. It does not support byte accesses. The queue manager linking RAM region 0 size register (LRAM0SIZE) is shown in Figure 13-125 and described in Table 13-127.

Figure 13-125. Queue Manager Linking RAM Region 0 Size Register (LRAM0SIZE)

15	14	13	0
Reserved		REGION0_SIZE	
R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-127. Queue Manager Linking RAM Region 0 Size Register (LRAM0SIZE) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-0	REGION0_SIZE	0-3FFh	This field indicates the number of entries that are contained in the linking RAM region 0. A descriptor with index less than region0_size value has its linking location in region 0. A descriptor with index greater than region0_size has its linking location in region 1. The queue manager will add the index (left shifted by 2 bits) to the appropriate regionX_base_addr to get the absolute 32-bit address to the linking location for a descriptor.

13.3.74 Queue Manager Linking RAM Region 1 Base Address Registers (LRAM1BASE1 and LRAM1BASE2)

The queue manager linking RAM region 1 base address registers (LRAM1BASE1 and LRAM1BASE2) are used to set the base address for the first portion of the Linking RAM. This address must be 32-bit aligned. These registers are used by the Queue Manager to calculate the 32-bit linking address for a given descriptor index. These registers do not support byte accesses.

The queue manager linking RAM region 1 base address register (LRAM1BASE1) is shown in Figure 13-126 and described in Table 13-128. The queue manager linking RAM region 1 base address register (LRAM1BASE2) is shown in Figure 13-127 and described in Table 13-129.

Figure 13-126. Queue Manager Linking RAM Region 1 Base Address Register 1 (LRAM1BASE1)

15	0
REGION1_BASE_LSB	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Figure 13-127. Queue Manager Linking RAM Region 1 Base Address Register 2 (LRAM1BASE2)

15	0
REGION1_BASE_MSB	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 13-128. Queue Manager Linking RAM Region 1 Base Address Register 1 (LRAM1BASE1) Field Descriptions

Bit	Field	Value	Description
15-0	REGION1_BASE_LSB	0-FFFFh	This field stores the least significant bits of the base address for the second region of the linking RAM. This may be anywhere in 32-bit address space but would be typically located in off-chip memory.

Table 13-129. Queue Manager Linking RAM Region 1 Base Address Register (LRAM1BASE2) Field Descriptions

Bit	Field	Value	Description
15-0	REGION1_BASE_MSB	0-FFFFh	This field stores the most significant bits of the base address for the second region of the linking RAM. This may be anywhere in 32-bit address space but would be typically located in off-chip memory.

13.3.75 Queue Manager Queue Pending Register 0 (PEND0)

The queue pending register 0 (PEND0) can be read to find the pending status for queues 15 to 0. It does not support byte accesses. The queue pending register 0 (PEND0) is shown in Figure 13-128 and described in Table 13-130.

Figure 13-128. Queue Manager Queue Pending Register 0 (PEND0)

15 0
QPEND0
R-0

LEGEND: R = Read only; -*n* = value after reset

Table 13-130. Queue Manager Queue Pending Register 0 (PEND0) Field Descriptions

Bit	Field	Value	Description
15-0	QPEND0	0-FFFFh	This field indicates the queue pending status for queues 15-0.

13.3.76 Queue Manager Queue Pending Register 1 (PEND1)

The queue pending register 1 (PEND1) can be read to find the pending status for queues 31 to 16. It does not support byte accesses. The queue pending register 1 (PEND1) is shown in Figure 13-129 and described in Table 13-131.

Figure 13-129. Queue Manager Queue Pending Register 1 (PEND1)

15 0
QPEND1
R-0

LEGEND: R = Read only; -*n* = value after reset

Table 13-131. Queue Manager Queue Pending Register 1 (PEND1) Field Descriptions

Bit	Field	Value	Description
15-0	QPEND1	0-FFFFh	This field indicates the queue pending status for queues 31-16.

13.3.77 Queue Manager Queue Pending Register 2 (PEND2)

The queue pending register 2 (PEND2) can be read to find the pending status for queues 47 to 32. It does not support byte accesses. The queue pending register 2 (PEND2) is shown in Figure 13-130 and described in Table 13-132.

Figure 13-130. Queue Manager Queue Pending Register 2 (PEND2)

15	0
QPEND2	
R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 13-132. Queue Manager Queue Pending Register 2 (PEND2) Field Descriptions

Bit	Field	Value	Description
15-0	QPEND2	0-FFFFh	This field indicates the queue pending status for queues 47-32.

13.3.78 Queue Manager Queue Pending Register 3 (PEND3)

The queue pending register 3 (PEND3) can be read to find the pending status for queues 63 to 48. It does not support byte accesses. The queue pending register 3 (PEND3) is shown in Figure 13-131 and described in Table 13-133.

Figure 13-131. Queue Manager Queue Pending Register 3 (PEND3)

15	0
QPEND3	
R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 13-133. Queue Manager Queue Pending Register 3 (PEND3) Field Descriptions

Bit	Field	Value	Description
15-0	QPEND3	0-FFFFh	This field indicates the queue pending status for queues 63-48.

13.3.79 Queue Manager Queue Pending Register 4 (PEND4)

The queue pending register 4 (PEND4) can be read to find the pending status for queues 79 to 64. It does not support byte accesses. The queue pending register 4 (PEND4) is shown in Figure 13-132 and described in Table 13-134.

Figure 13-132. Queue Manager Queue Pending Register 4 (PEND4)

15 … 0
QPEND4
R-0

LEGEND: R = Read only; -*n* = value after reset

Table 13-134. Queue Manager Queue Pending Register 4 (PEND4) Field Descriptions

Bit	Field	Value	Description
15-0	QPEND4	0-FFFFh	This field indicates the queue pending status for queues 79-64.

13.3.80 Queue Manager Queue Pending Register 5 (PEND5)

The queue pending register 5 (PEND5) can be read to find the pending status for queues 95 to 80. It does not support byte accesses. The queue pending register 5 (PEND5) is shown in Figure 13-133 and described in Table 13-135.

Figure 13-133. Queue Manager Queue Pending Register 5 (PEND5)

15 … 0
QPEND5
R-0

LEGEND: R = Read only; -*n* = value after reset

Table 13-135. Queue Manager Queue Pending Register 5 (PEND5) Field Descriptions

Bit	Field	Value	Description
15-0	QPEND5	0-FFFFh	This field indicates the queue pending status for queues 95-80.

13.3.81 Queue Manager Memory Region *R* Base Address Registers (QMEMRBASE1[R] and QMEMRBASE2[R])

The memory region *R* base address registers (QMEMRBASE1[R] and QMEMRBASE2[R]) are written by the host to set the base address of memory region *R*, where *R* is 0-15. This memory region will store a number of descriptors of a particular size as determined by the memory region *R* control register. These registers do not support byte accesses.

The memory region *R* base address register (QMEMRBASE1[*R*]) is shown in Figure 13-134 and described in Table 13-136. The memory region *R* base address register (QMEMRBASE2[*R*]) is shown in Figure 13-135 and described in Table 13-137.

Figure 13-134. Queue Manager Memory Region *R* Base Address Register 1 (QMEMRBASE1[*R*])

15		0
	REG_LSB	
	R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Figure 13-135. Queue Manager Memory Region *R* Base Address Register 2 (QMEMRBASE2[*R*])

15		0
	REG_MSB	
	R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 13-136. Queue Manager Memory Region *R* Base Address Register 1 (QMEMRBASE1[*R*]) Field Descriptions

Bit	Field	Value	Description
15-0	REG_LSB	0-FFFFh	This field contains the least-significant bits of the base address of the memory region R.

Table 13-137. Queue Manager Memory Region *R* Base Address Register 2 (QMEMRBASE2[*R*]) Field Descriptions

Bit	Field	Value	Description
15-0	REG_MSB	0-FFFFh	This field contains the most-significant bits of the base address of the memory region R.

13.3.82 Queue Manager Memory Region *R* Control Registers (QMEMRCTRL1[R] and QMEMRCTRL2[R])

The memory region *R* control registers (QMEMRCTRL1[R] and QMEMRCTRL2[R]) are written by the host to configure various parameters of memory region *R*, where *R* is 0-15. These registers do not support byte accesses.

The memory region *R* control register (QMEMRCTRL1[*R*])) is shown in Figure 13-136 and described in Table 13-138. The memory region *R* control register (QMEMRCTRL2[*R*])) is shown in Figure 13-137 and described in Table 13-139.

Figure 13-136. Queue Manager Memory Region *R* Control Register 1 (QMEMRCTRL1[*R*])

15 12	11 8	7 3	2 0
Reserved	DESC_SIZE	Reserved	REG_SIZE
R-0	R/W-0	R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 13-137. Queue Manager Memory Region *R* Control Register 2 (QMEMRCTRL2[*R*])

15 14	13 0
Reserved	START_INDEX
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 13-138. Queue Manager Memory Region *R* Control Register 1 (QMEMRCTRL1[*R*]) Field Descriptions

Bit	Field	Value	Description
15-12	Reserved	0	Reserved.
11-8	DESC_SIZE	0-Fh	This field indicates the size of each descriptor in this memory region.
		0	32.
		1h	64.
		2h	128.
		3h	256.
		4h	512.
		5h	1K.
		6h	2K.
		7h	4K.
		8h	8K.
		9h-Fh	Reserved.
7-3	Reserved	0	Reserved.
2-0	REG_SIZE	0-7h	This field indicates the size of the memory region (in terms of number of descriptors).
		0	32.
		1h	64.
		2h	128.
		3h	256.
		4h	512.
		5h	1K.
		6h	2K.
		7h	4K.

Table 13-139. Queue Manager Memory Region *R* Control Register 2 (QMEMRCTRL2[*R*]) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-0	START_INDEX	0-3FFFh	This field indicates where in linking RAM the descriptor linking information corresponding to memory region R starts.

13.3.83 Queue Manager Queue N Control Register D (CTRL1D[N] and CTRL2D[N])

The queue manager queue *N* control registers D (CTRL1D[N] and CTRL2D[N]) are written to add a packet to the queue and read to pop a packets off a queue. The packet is only pushed or popped to/from the queue when the queue manager queue *N* control register D is written. These registers do not support byte accesses.

The queue manager queue *N* control register 1 D (CTRL1D[*N*]) is shown in Figure 13-138 and described in Table 13-140. The queue manager queue *N* control register 2 D (CTRL2D[*N*]) is shown in Figure 13-139 and described in Table 13-141.

Figure 13-138. Queue Manager Queue *N* Control Register 1 D (CTRL1D[*N*])

15 5	4 0
DESC_PTR_LSB	DESC_SIZE
R/W-0	R/W-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Figure 13-139. Queue Manager Queue *N* Control Register 2 D (CTRL2D[*N*])

15 0
DESC_PTR_MSB
R/W-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 13-140. Queue Manager Queue *N* Control Register 1 D (CTRL1D[*N*]) Field Descriptions

Bit	Field	Value	Description
15-5	DESC_PTR_LSB		Descriptor Pointer (Least significant bits).
		0	Queue is empty.
		1	Indicates a 32-bit aligned address that points to a descriptor.
4-0	DESC_SIZE	0-1Fh	The descriptor size is encoded in 4-byte increments. This field returns a 0 when an empty queue is read.
		0	24 bytes.
		1h	28 bytes.
		2h	32 bytes.
		3h-1Fh	36 bytes to 148 bytes.

Table 13-141. Queue Manager Queue *N* Control Register 2 D (CTRL2D[*N*]) Field Descriptions

Bit	Field	Value	Description
15-0	DESC_PTR_MSB		Descriptor Pointer (Most significant bits).
		0	Queue is empty.
		1	Indicates a 32-bit aligned address that points to a descriptor.

13.3.84 Queue Manager Queue *N* Status Register A (QSTATA[N])

The queue manager queue *N* status register A (QSTATA[*N*]) is an optional register that is only implemented for a queue if the queue supports entry/byte count feature. The entry count feature provides a count of the number of entries that are currently valid in the queue. It does not support byte accesses. The queue manager queue *N* status register A (QSTATA[*N*]) is shown in Figure 13-140 and described in Table 13-142.

Figure 13-140. Queue Manager Queue *N* Status Register A (QSTATA[*N*])

15 14	13 0
Reserved	QUEUE_ENTRY_COUNT
R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 13-142. Queue Manager Queue *N* Status Register A (QSTATA[*N*]) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-0	QUEUE_ENTRY_COUNT	0-3FFFh	This field indicates how many packets are currently queued on the queue.

13.3.85 Queue Manager Queue *N* Status Registers B (QSTAT1B[N] and QSTAT2B[N])

The queue manager queue *N* status registers B (QSTAT1B[N] and QSTAT2B[N]) are optional registers that are only implemented for a queue if the queue supports a total byte count feature. The total byte count feature provides a count of the total number of bytes in all of the packets that are currently valid in the queue. The registers do not support byte accesses.

The queue manager queue *N* status register 1 B (QSTAT1B[*N*]) is shown in Figure 13-141 and described in Table 13-143. The queue manager queue *N* status register 2 B (QSTAT2B[*N*]) is shown in Figure 13-142 and described in Table 13-144.

Figure 13-141. Queue Manager Queue *N* Status Register 1 B (QSTAT1B[*N*])

15 0
QUEUE_BYTE_COUNT_LSB
R-0

LEGEND: R = Read only; -*n* = value after reset

Figure 13-142. Queue Manager Queue *N* Status Register 2 B (QSTAT2B[*N*])

15 12	11 0
Reserved	QUEUE_BYTE_COUNT_MSB

LEGEND: R = Read only; -*n* = value after reset

Table 13-143. Queue Manager Queue *N* Status Register 1 B (QSTAT1B[*N*]) Field Descriptions

Bit	Field	Value	Description
15-0	QUEUE_BYTE_COUNT_LSB	0-FFFFh	Together, QUEUE_BYTE_COUNT_MSB and QUEUE_BYTE_COUNT_LSB indicate how many bytes total are contained in all of the packets which are currently queued on this queue.

Table 13-144. Queue Manager Queue *N* Status Register 2 B (QSTAT2B[*N*]) Field Descriptions

Bit	Field	Value	Description
15-12	Reserved	0	Reserved.

Table 13-144. Queue Manager Queue *N* Status Register 2 B (QSTAT2B[*N*]) Field Descriptions (continued)

Bit	Field	Value	Description
11-0	QUEUE_BYTE_COUNT_MSB	0-FFFh	Together, QUEUE_BYTE_COUNT_MSB and QUEUE_BYTE_COUNT_LSB indicate how many bytes total are contained in all of the packets which are currently queued on this queue.

13.3.86 Queue Manager Queue N *Status Register C (QSTATC[N])*

The queue manager queue *N* status register C (QSTATC[*N*]) specifies the packet size for the head element of a queue. It does not support byte accesses. The queue manager queue *N* status register C (QSTATC[*N*]) is shown in Figure 13-143 and described in Table 13-145.

Figure 13-143. Queue Manager Queue *N* Status Register C (QSTATC[*N*])

15	14	13	0
Reserved		PACKET_SIZE	
R-0		R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 13-145. Queue Manager Queue *N* Status Register C (QSTATC[*N*]) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-0	PACKET_SIZE	0-3FFFh	This field indicates how many packets are currently queued on the queue.

Chapter 14

SPRUH87C – August 2011 – Revised March 2012

Liquid Crystal Display Controller (LCDC)

This chapter describes the features and operations of the liquid crystal display (LCD) controller.

Topic **Page**

14.1 Introduction

The liquid crystal display controller (LCDC) supports asynchronous (memory-mapped) LCD interfaces.

14.1.1 Purpose of the Peripheral

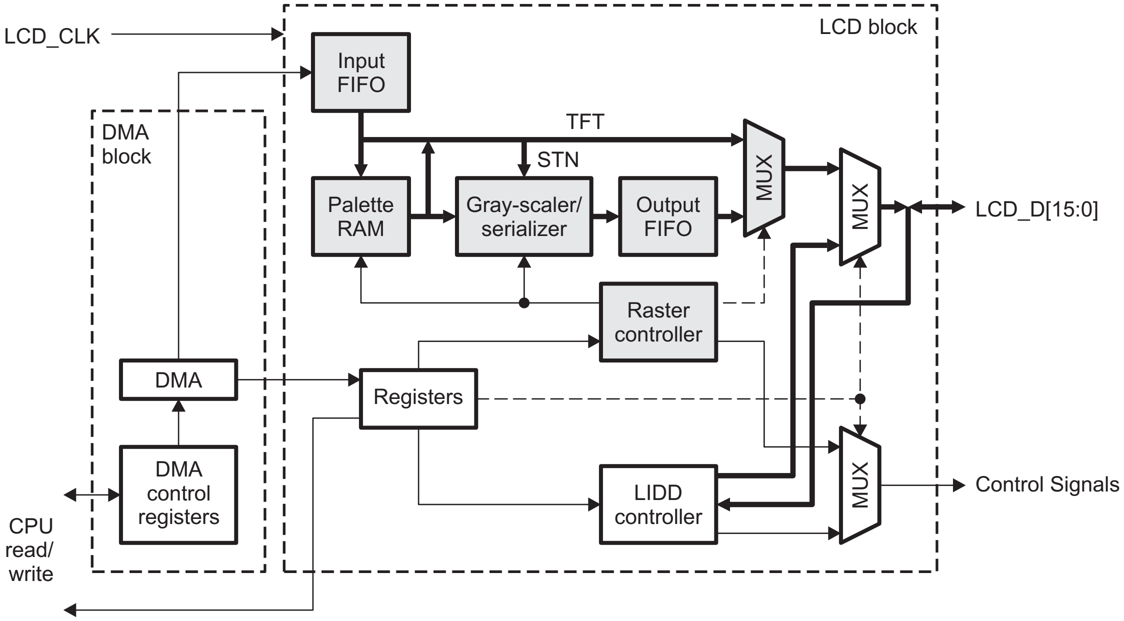
The LCD controller consists of two independent controllers, the Raster Controller and the LCD Interface Display Driver (LIDD) controller. Each controller operates independently from the other and only one of them is active at any given time. The Raster Controller is currently not supported and is only for feature enhancement for the future.

The LIDD Controller supports the asynchronous LCD interface. It provides full-timing programmability of control signals and output data.

Figure 14-1 shows internal blocks of the LCD controller. Please note that Figure 1 shows LIDD controller and Raster controller as well but the blocks in gray belong to Raster Controller and are not supported. The solid, thick lines in Figure 1 indicate the data path. The LIDD controllers are responsible for generating the correct external timing. The DMA engine provides a constant flow of data from the frame buffer(s) to the external LCD panel via the LIDD Controllers. In addition, CPU access is provided to read and write registers.

The blocks in gray belong to the Raster Controller that is not supported.

Figure 14-1. LCD Controller



14.1.2 Terminology Used in this Document

Term	Meaning
RDB	Read Strobe
WRB	Write Strobe
RS	Register Select
CS	Chip select
OE	Output enable
WE	Write enable

14.1.2.1 LCD Clock

The LCD controller has an internal clock (LCD_CLK) which is derived from system clock. The LCD_CLK determines the minimum cycle of configuration of the control signals .

$$\text{LCD\_CLK} = \text{SYSTEM\_CLK when CLK\_DIV} = 0$$

$$\frac{\text{LCD\_CLK} = \text{SYSTEM\_CLK}}{\text{LCK\_DIV}} \text{ when } 0 < \text{CLK\_DIV} < 256$$

14.1.3 LCD External I/O Signals

Table 14-1 shows the details of the LCD controller external signals.

Table 14-1. LCD External I/O Signals

Signal	Type	Description
LCD_RS	OUT	HD44780U mode: Register Select (RS) MPU6800 Sync mode: Command/Data Select (C/D) MPU6800 Async mode: Command/Data Select (C/D) MPU80 Sync mode: Command/Data Select (C/D) MPU80 Async mode: Command/Data Select (C/D)
LCD_RW_WRB	OUT	HD44780U mode: Read/Write Select (R/W) MPU6800 Sync mode: Read/Write Select (R/W) MPU6800 Async mode: Read/Write Select (R/W) MPU80 Sync mode: Write Strobe (WRB) MPU80 Async mode: Write Strobe (WRB)
LCD_EN_RDB	OUT	HD44780U mode: not used MPU6800 Sync mode: Read or Write Enable (EN) MPU6800 Async mode: Read or Write Enable (EN) MPU80 Sync mode: Read Strobe (RDB) MPU80 Async mode: Read Strobe (RDB)
LCD_CS0_E0	OUT	HD44780U mode: Start Data Read/Write (E0) MPU6800 Sync mode: Primary Chip Select (CS0) MPU6800 Async mode: Primary Chip Select (CS0) MPU80 Sync mode: Primary Chip Select (CS0) MPU80 Async mode: Primary Chip Select (CS0)

Table 14-1. LCD External I/O Signals (continued)

Signal	Type	Description
LCD_CS1_E1	OUT	HD44780U mode: Start Data Read/Write (E1)
		MPU6800 Sync mode: LCD_CLK output
		MPU6800 Async mode: Secondary Chip Select (CS1)
		MPU80 Sync mode: LCD_CLK output
		MPU80 Async mode: Secondary Chip Select (CS1)
LCD_D[15:0]	LIDD: OUT/IN	HD44780U mode: Read and write the command and Data
		MPU6800 Sync mode: Read and write command and data
		MPU6800 Async mode: Read and write command and data
		MPU80 Sync mode: Read and write command and data
		MPU80 Async mode: Read and write command and data

14.1.4 LCD Interface Display Driver Details (LIDD) Controller

The LIDD Controller is designed to support LCD panels with a memory-mapped interface. The types of displays range from low-end character monochrome LCD panels to high-end TFT smart LCD panels.

LIDD mode (and the use of this logic) is enabled by clearing the MODESEL bit in the LCD control register (LCD_CTRL).

LIDD Controller operation is summarized as follows:

- During initialization, the LCD LIDD CS0/CS1 configuration registers (LIDD_CS0_CONF and LIDD_CS1_CONF) are configured to match the requirements of the LCD panel being used.
- During normal operation, the CPU writes display data to the LCD data registers (LIDD_CS0_DATA and LIDD_CS1_DATA). The LIDD interface converts the CPU write into the proper signal transition sequence for the display, as programmed earlier. Note that the first CPU write should send the beginning address of the update to the LCD panel and the subsequent writes update data at display locations starting from the first address and continuing sequentially. Note that DMA may be used instead of CPU.
- The LIDD Controller is also capable of reading back status or data from the LCD panel, if the latter has this capability. This is set up and activated in a similar manner to the write function described above.

NOTE: If an LCD panel is not used, this interface can be used to control any MCU-like peripheral.

Table 14-2 describes how the signals are used to interface external LCD modules, which are configured by the LIDD_CTRL register.

Table 14-2. LIDD I/O Name Map

Display Type	Interface Type	Data Bits	LIDD_CTRL [2:0]	I/O Name	Display I/O Name	Comment
Character Display	HD44780 Type	4	100	LCD_D[7:4]	DATA[7:4]	Data Bus (length defined by Instruction)
				LCD_CS0_E1	E0	Enable Strobe (first display)
				LCD_RW_WRB	R/$\overline{W}$	Read/$\overline{\text{Write}}$
				LCD_RS	RS	Register Select (Data/not Instruction)
				LCD_CS1_E1	E1	Enable Strobe (second display optional)

Table 14-2. LIDD I/O Name Map (continued)

Display Type	Interface Type	Data Bits	LIDD_CTRL [2:0]	I/O Name	Display I/O Name	Comment
Character Display	HD44780 Type	8	100	LCD_D[7:0]	DATA[7:0]	Data Bus (length defined by Instruction)
				LCD_CS0_E0	E0	Enable Strobe (first display)
				LCD_RW_WRB	R/$\overline{W}$	Read/$\overline{\text{Write}}$
				LCD_RS	RS	Register Select (Data/not Instruction)
				LCD_CS1_E1	E1	Enable Strobe (second display optional)
Micro Interface Graphic Display	6800 Family	Up To 16	001	LCD_D[7:0]	DATA[7:0]	Data Bus
				LCD_EN_RDB	E	Enable Clock
				LCD_RW_WRB	R/$\overline{W}$	Read/$\overline{\text{Write}}$
				LCD_RS	A0	Address/Data Select
				LCD_CS0_E0	CS (or CS0)	Chip Select (first display)
				LCD_CS1_E1	CS1	Chip Select (second display optional)
			000	LCD_CS1_E1	LCD_CLK	LCD_CLK Output for Synchronous Clock
Micro Interface Graphic Display	8080 Family	Up To 16	011	LCD_D[7:0]	DATA[7:0]	Data Bus
				LCD_EN_RDB	RD	Read Strobe
				LCD_RW_WRB	WR	Write Strobe
				LCD_RS	A0	Address/Data Select
				LCD_CS1_E0	CS (or CS0)	Chip Select (first display)
				LCD_CS1_E1	CS1	Chip Select (second display optional)
			010	LCD_CLK	LCD_CLK	LCD_CLK Output for Synchronous Clock

The timing parameters are defined by the LIDD_CS0_CONF and LIDD_CS1_CONF registers.

The timing configuration is based on an internal reference clock, LCD_CLK. LCD_CLK is generated out of System Clock , which is determined by the CLKDIV bit in the LCD_CTRL register.

$$\text{LCD\_CLK} = \text{SYSTEM\_CLK when CLK\_DIV} = 0$$

$$\frac{\text{LCD\_CLK} = \text{SYSTEM\_CLK}}{\text{LCK\_DIV}} \text{ when } 0 < \text{CLK\_DIV} < 256$$

14.1.5 LIDD Controller Timing

Figure 14-2. LIDD Mode HD44780 Write Timing Diagram

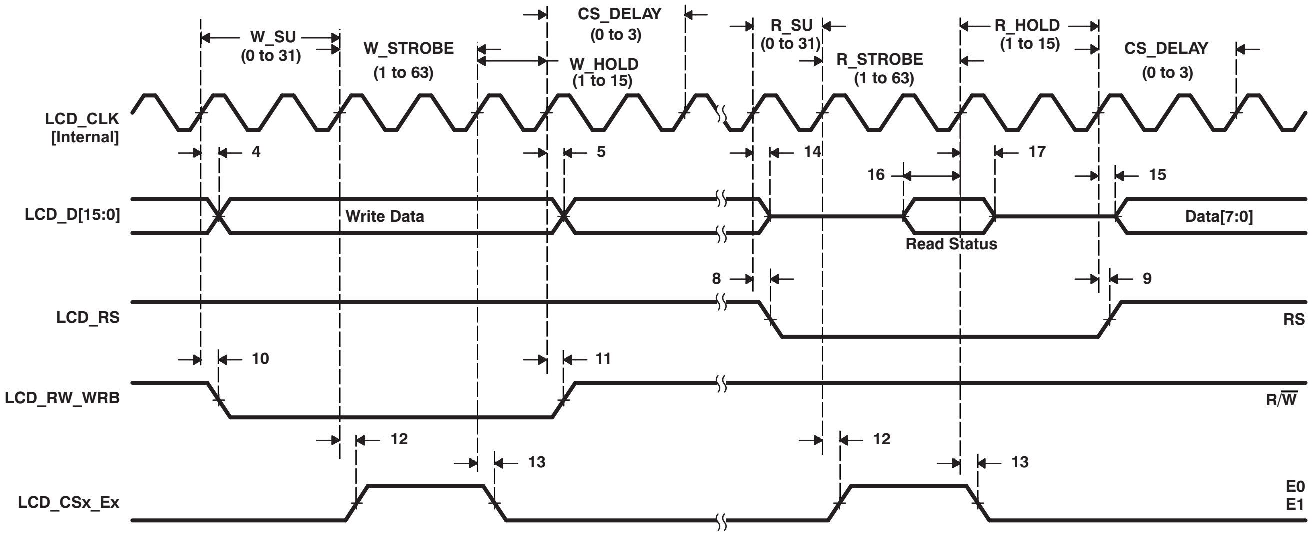
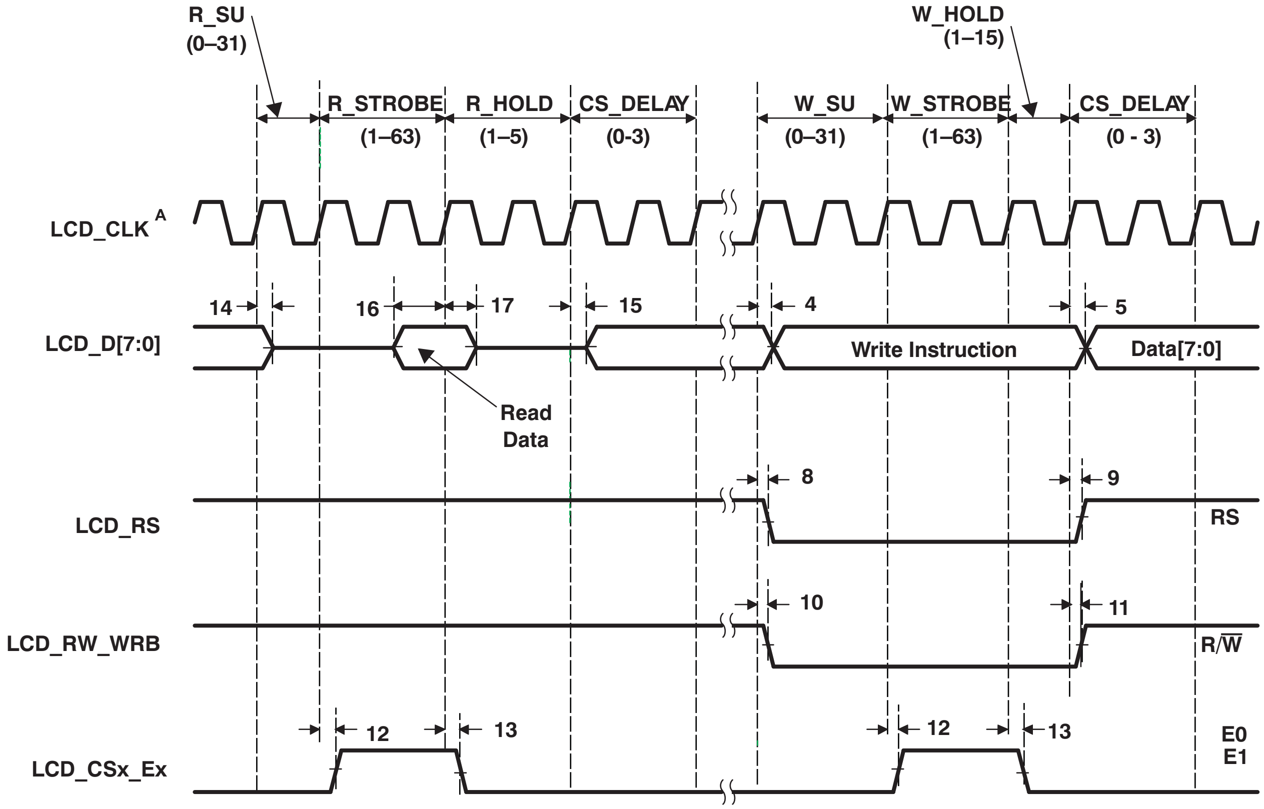
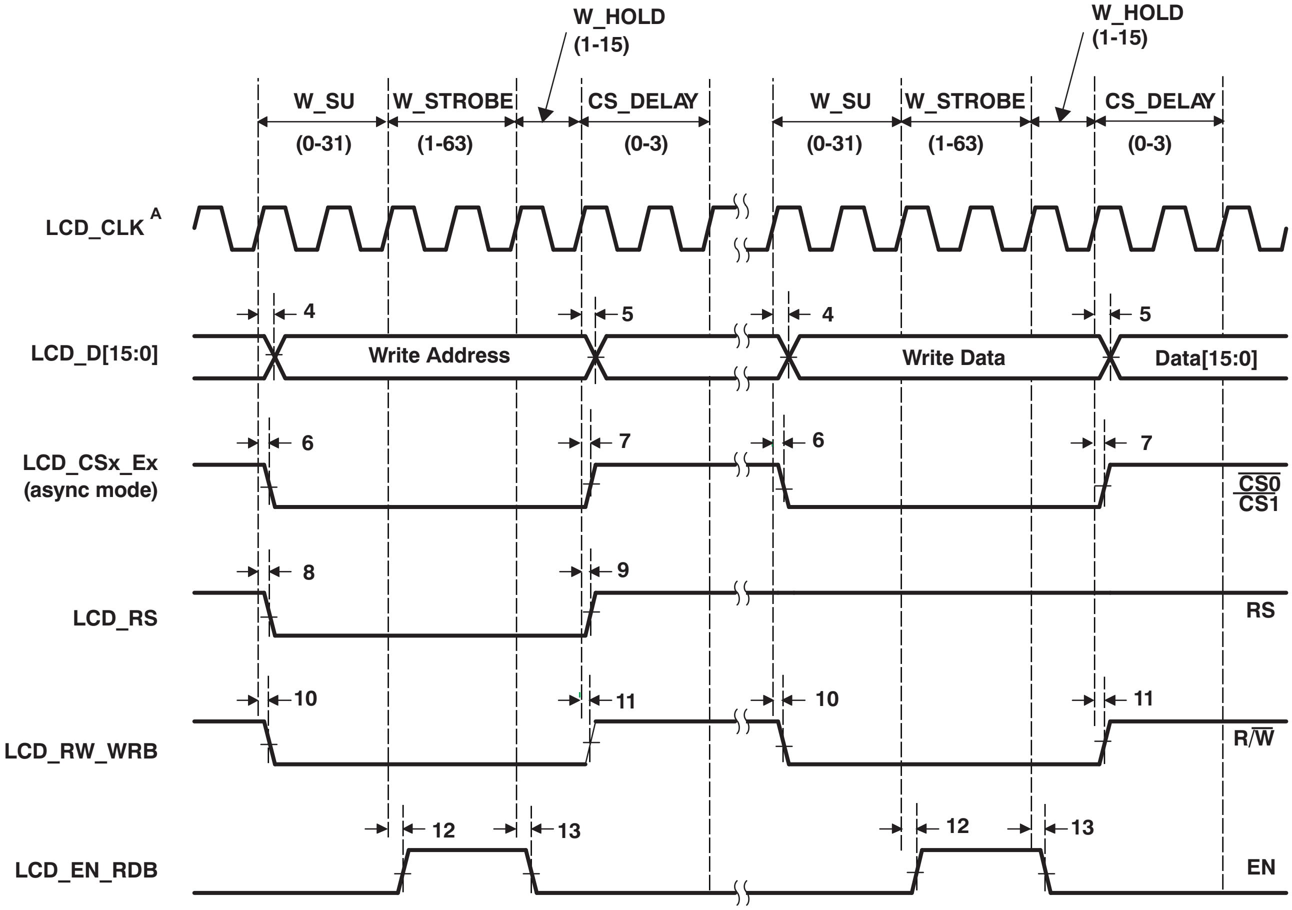


Figure 14-3. LIDD Mode HD44780 Read Timing Diagram



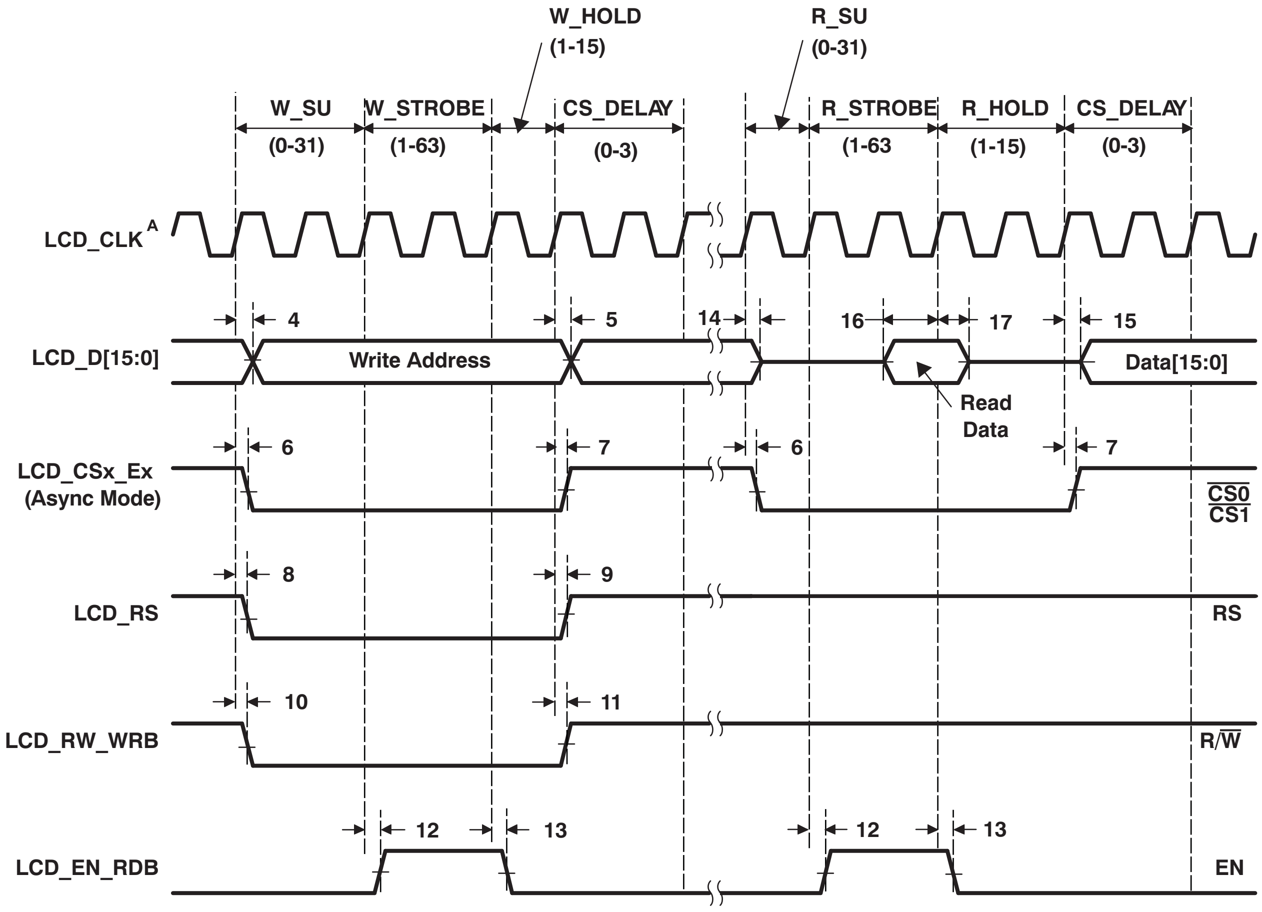
A In 6800 Async mode, LCD_CLK is internal
In 6800 Sync mode, LCD_CLK is output via LCD_CS1_E1

Figure 14-4. LIDD Mode 6800 Write Timing Diagram



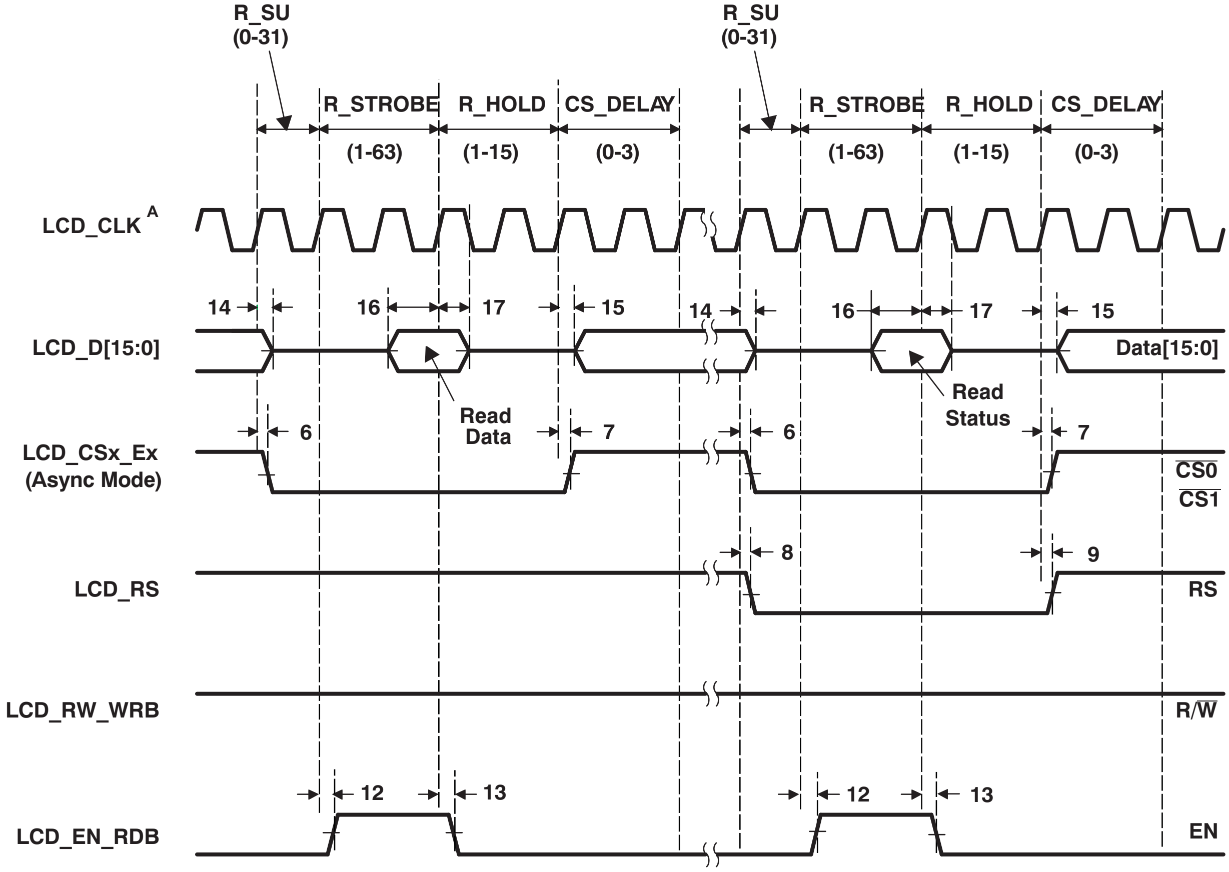
A In 6800 Async mode, LCD_CLK is internal
In 6800 Sync mode, LCD_CLK is output via LCD_CS1_E1

Figure 14-5. LIDD Mode 6800 Read Timing Diagram



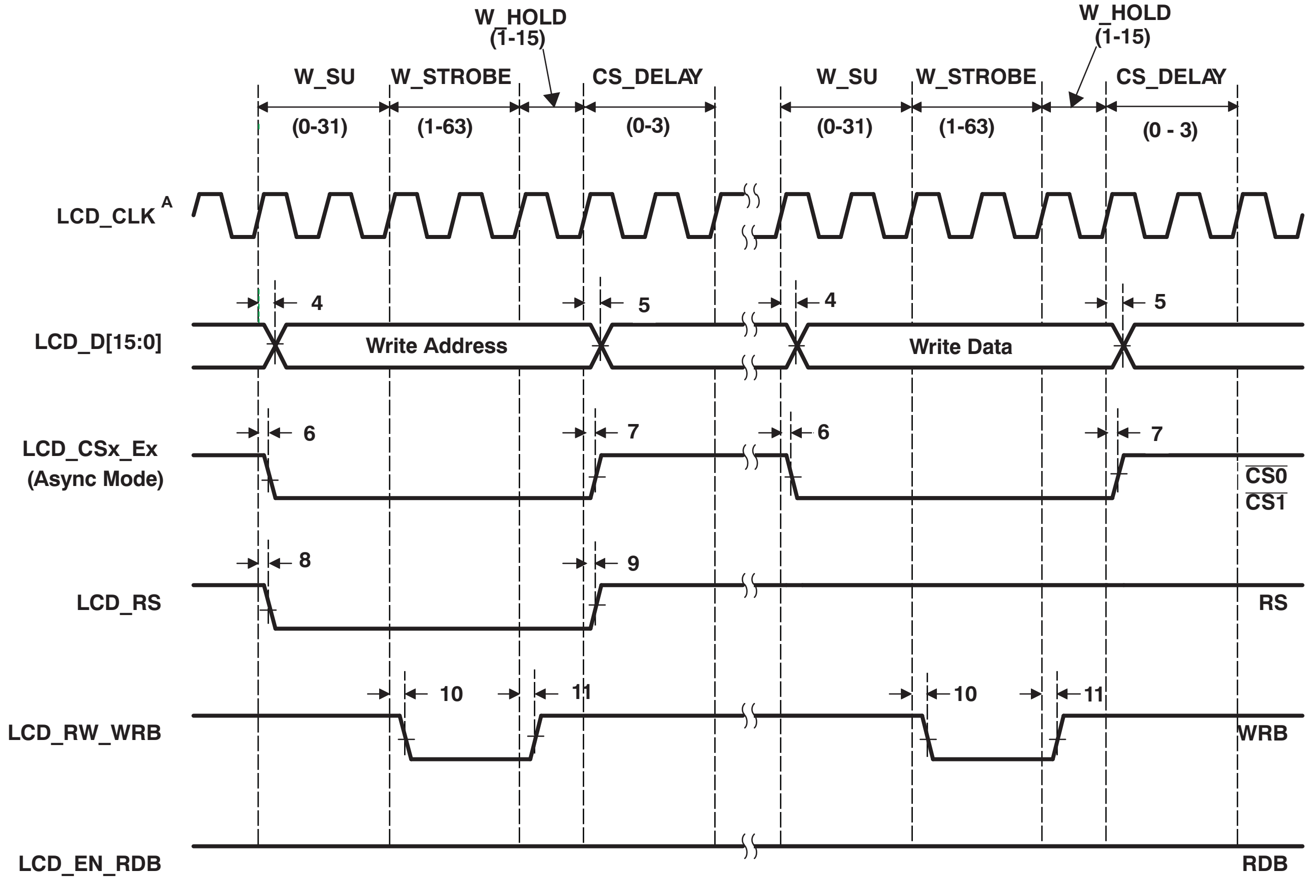
A In 6800 Async mode, LCD_CLK is internal
In 6800 Sync mode, LCD_CLK is output via LCD_CS1_E1

Figure 14-6. LIDD Mode 6800 Status Timing Diagram



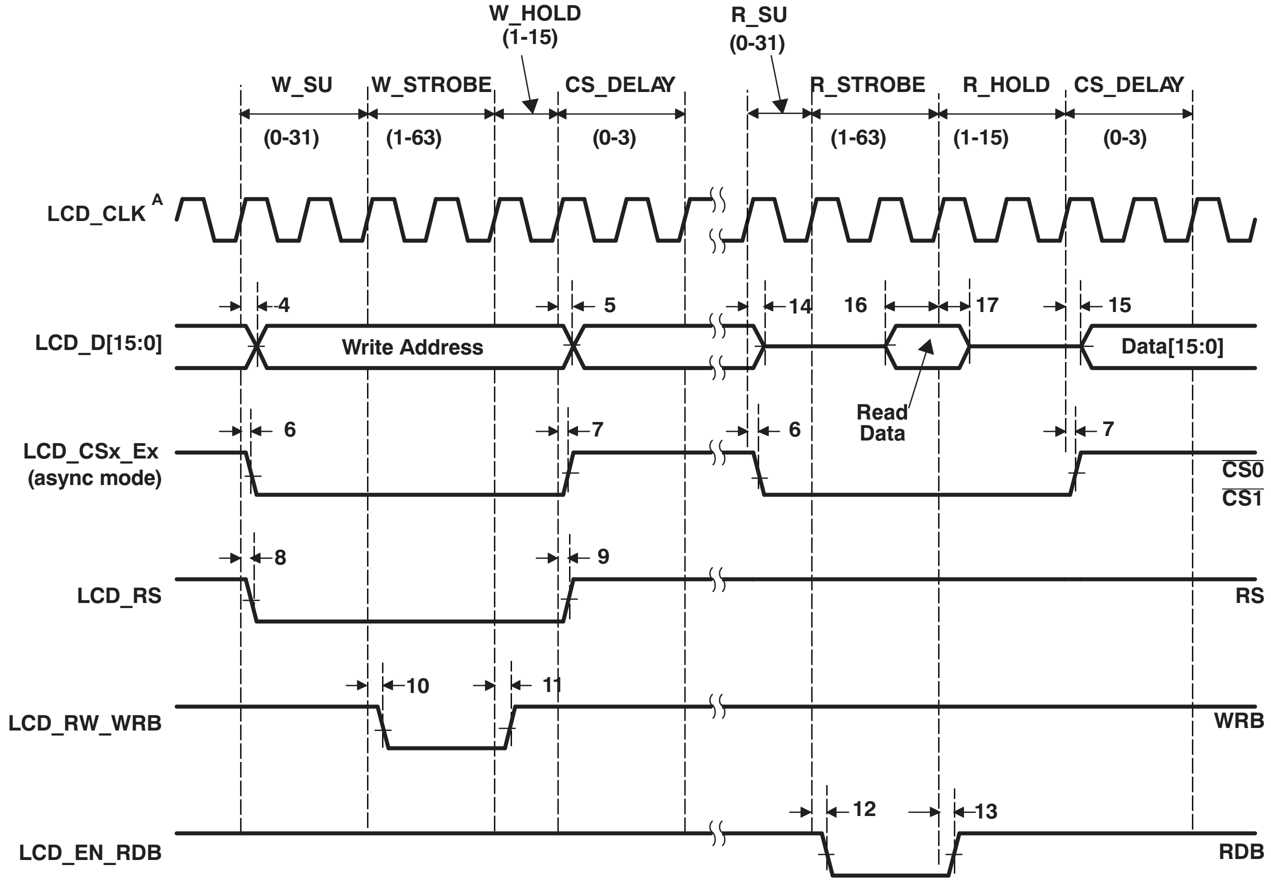
A In 6800 Async mode, LCD_CLK is internal
In 6800 Sync mode, LCD_CLK is output via LCD_CS1_E1

Figure 14-7. LIDD Mode 8080 Write Timing Diagram



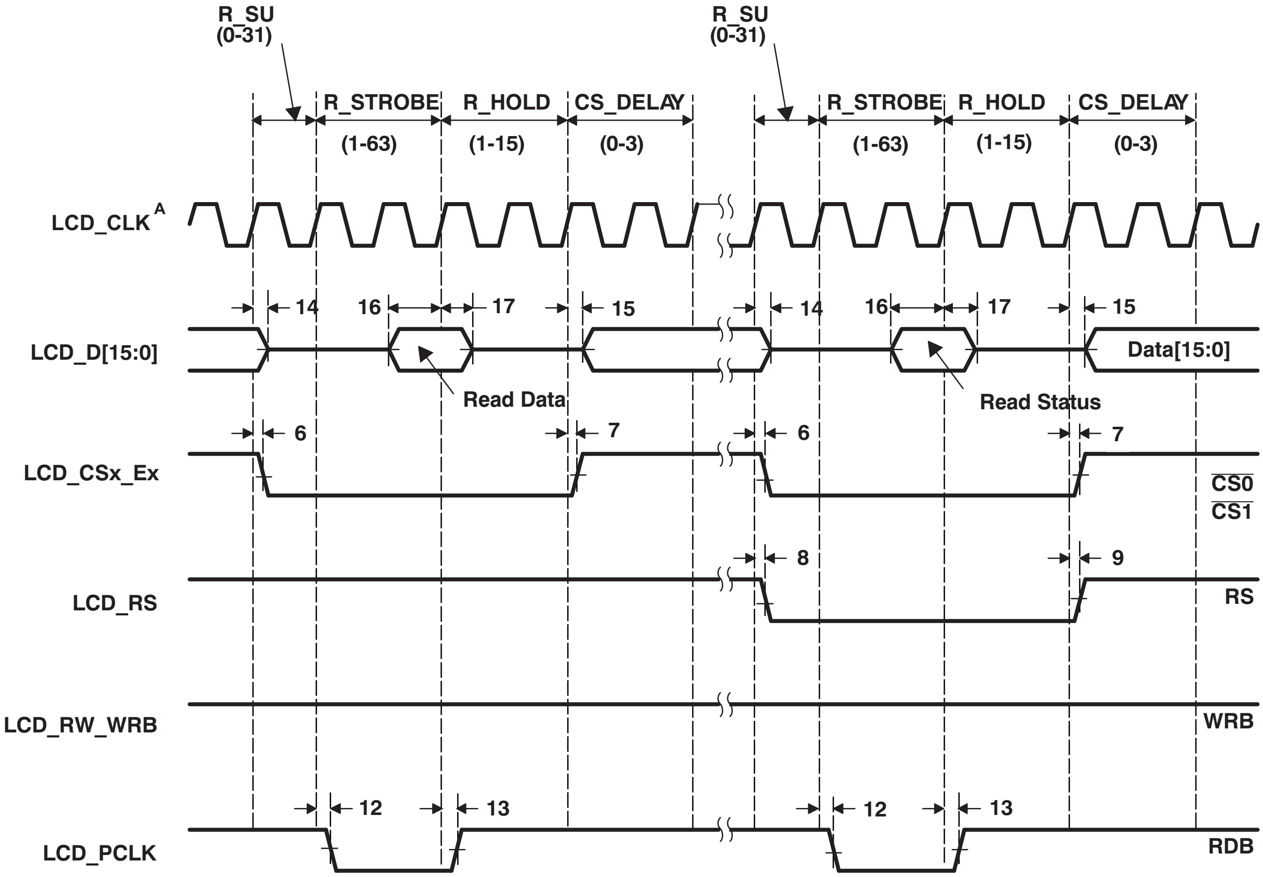
A In 8080 Async mode, LCD_CLK is internal
In 8080 Sync mode, LCD_CLK is output via LCD_CS1_E1

Figure 14-8. LIDD Mode 8080 Read Timing Diagram



A In 8080 Async mode, LCD_CLK is internal
In 8080 Sync mode, LCD_CLK is output via LCD_CS1_E1

Figure 14-9. LIDD Mode 8080 Status Timing Diagram



A In 8080 Async mode, LCD_CLK is internal
In 8080 Sync mode, LCD_CLK is output via LCD_CS1_E1

14.1.6 DMA Engine

The DMA engine provides the capability to output data constantly, without burdening the CPU, via interrupts or a firmware timer. It operates on one or two frame buffers, which are set up during initialization. Using two frame buffers (ping-pong buffers) enables the simultaneous operation of outputting the current video frame to the external display and updating the next video frame. The ping-pong buffering approach is preferred in most applications.

The DMA engine accesses the LIDD Controller's address and/or data registers.

To program DMA engine, configure the following registers, as shown in Table 14-3.

Table 14-3. Register Configuration for DMA Engine Programming

Register	Configuration
LCDDMA_CTRL	Configure DMA data format
LCDDMA_FB0_BASE LCDDMA_FB0_CEILING	Configure frame buffer 0
LCDDMA_FB1_BASE LCDDMA_FB1_CEILING	Configure frame buffer 1 (If only one frame buffer is used, these two registers will not be used.)

In addition, the LIDD_CTRL register (for LIDD Controller) should also be configured appropriately, along with all the timing registers.

To enable DMA transfers, the LIDD_DMA_EN bit (in the LIDD_CTRL register) should be written with 1.

14.1.6.1 Interrupts

Interrupts in this LCD module are related to DMA engine operation. Two registers are closely related to this subject:

- The LIDD_CTRL enables or disables .
- The LCD_STAT register collects all the interrupt status information.

The DMA engine generates one interrupt signal every time the specified frame buffer has been transferred completely.

- The DONE_INT_EN bit in the LIDD_CTRL register specifies if the interrupt signal is delivered to the system interrupt controller, which in turn may or may not generate an interrupt to CPU.
- The EOF1, EOF0, and DONE bits in the LCD_STAT register reflect the interrupt signal, regardless of being delivered to the system interrupt controller or not.

14.1.6.1.1 Interrupt Handling

Refer the device-specific data manual for information about LCD interrupt number to CPU .The interrupt service routine needs to determine the interrupt source by examining the LCD_STAT register and clearing the interrupt properly.

14.2 LCD Port Mapping

The DSP uses pin multiplexing to accommodate a larger number of peripheral functions in the smallest possible package, providing the ultimate flexibility for end applications. The external bus selection register (EBSR) controls all the pin multiplexing functions on the device. LCD ports can be 8-bit LCD ports, 16-bit LCD ports, or disabled via the EBSR register. For more details on the EBSR register, see Section 1.1, *System Control*.

14.3 Registers

Table 14-4 lists the memory-mapped registers for the LCD Controller (LCDC). See the device-specific data manual for the memory address of these registers.

Table 14-4. LCD Controller (LCDC) Registers

CPU Word Address	Acronym	Register Description	Section
2E00h	LCDREVMIN	LCD Minor Revision Register	Section 14.3.1
2E01h	LCDREVMAJ	LCD Major Revision Register	Section 14.3.2
2E04h	LCDCR	LCD Control Register	Section 14.3.3
2E08h	LCDSR	LCD Status Register	Section 14.3.4
2E0Ch	LCDLIDDCR	LCD LIDD Control Register	Section 14.3.5
2E10h	LCDLIDDCS0CONFIG0	LCD LIDD CS0 Configuration Register 0	Section 14.3.6
2E11h	LCDLIDDCS0CONFIG1	LCD LIDD CS0 Configuration Register 1	Section 14.3.7
2E14h	LCDLIDDCS0ADDR	LCD LIDD CS0 Address Read/Write Register	Section 14.3.8
2E18h	LCDLIDDCS0DATA	LCD LIDD CS0 Data Read/Write Register	Section 14.3.9
2E1Ch	LCDLIDDCS1CONFIG0	LCD LIDD CS1 Configuration Register 0	Section 14.3.6
2E1Dh	LCDLIDDCS1CONFIG1	LCD LIDD CS1 Configuration Register 1	Section 14.3.7
2E20h	LCDLIDDCS1ADDR	LCD LIDD CS1 Address Read/Write Register	Section 14.3.8
2E24h	LCDLIDDCS1DATA	LCD LIDD CS1 Data Read/Write Register	Section 14.3.9
2E28h – 2E3Ah	-	Reserved	-
2E40h	LCDDMACR	LCD DMA Control Register	Section 14.3.10
2E44h	LCDDMAFB0BAR0	LCD DMA Frame Buffer 0 Base Address Register 0	Section 14.3.11
2E45h	LCDDMAFB0BAR1	LCD DMA Frame Buffer 0 Base Address Register 1	Section 14.3.12
2E48h	LCDDMAFB0CAR0	LCD DMA Frame Buffer 0 Ceiling Address Register 0	Section 14.3.13
2E49h	LCDDMAFB0CAR1	LCD DMA Frame Buffer 0 Ceiling Address Register 1	Section 14.3.14
2E4Ch	LCDDMAFB1BAR0	LCD DMA Frame Buffer 1 Base Address Register 0	Section 14.3.15
2E4Dh	LCDDMAFB1BAR1	LCD DMA Frame Buffer 1 Base Address Register 1	Section 14.3.16
2E50h	LCDDMAFB1CAR0	LCD DMA Frame Buffer 1 Ceiling Address Register 0	Section 14.3.17
2E51h	LCDDMAFB1CAR1	LCD DMA Frame Buffer 1 Ceiling Address Register 1	Section 14.3.18

14.3.1 LCD Minor Revision Register (LCDREVMIN)

The LCD minor revision register (LCDREVMIN) is shown in Figure 14-10 and described in Table 14-5.

Figure 14-10. LCD Minor Revision Register (LCDREVMIN)

15	0
REVMIN	
R- 0000h	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 14-5. LCD Minor Revision Register (LCDREVMIN) Field Descriptions

Bit	Field	Value	Description
15-0	REVMIN	0000h	Minor Revision.

14.3.2 LCD Major Revision Register (LCDREVMAJ)

The LCD major revision register (LCDREVMAJ) is shown in Table 14-6 and described in Table 14-6.

Figure 14-11. LCD Major Revision Register (LCDREVMAJ)

15	0
REVMAJ	
R-0001h	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 14-6. LCD Major Revision Register (LCDREVMAJ) Field Descriptions

Bit	Field	Value	Description
15-0	REVMAJ	0001h	Major Revision.

14.3.3 LCD Control Register (LCDCR)

The LCD control register (LCDCR) is shown in Figure 14-12 and described in Table 14-7.

Figure 14-12. LCD Control Register (LCDCR)

15	8	7	1	0
CLKDIV		Reserved		MODESEL
RW-0		R-0		RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 14-7. LCD Control Register (LCDCR) Field Descriptions

Bit	Field	Value	Description
15-8	CLKDIV	0-FFh	Clock Divisor Value (from 0–255) is used to specify the frequency of LCD_CLK .
7-1	Reserved	0	Reserved.
0	MODESEL		LCD Mode Select. MODESEL should be set to 0 (default) all the time. Raster mode is not supported.
		0	LCD Controller in LIDD mode.
		1	LCD Controller in Raster mode (not supported).

The 8-bit clock divider (CLKDIV) field is used to select the frequency LCD_CLK.

$$\text{LCD\_CLK} = \text{SYSTEM\_CLK when CLK\_DIV} = 0$$

$$\text{LCD\_CLK} = \frac{\text{SYSTEM\_CLK}}{\text{CLK\_DIV}} \text{ when } 0 < \text{CLK\_DIV} < 256$$

14.3.4 LCD Status Register (LCDSR)

The LCD status register (LCD_STAT) contains bits that signal status to the processor. Each of the LCD status bits signals an interrupt request as long as the bit is set AND the interrupt enable for that bit is also set (see the LCD DMA control register for these enables). Writing a 1 to each bit clears it; once the bit is cleared, the interrupt is cleared.

The LCD status register (LCDSR) is shown in Figure 14-13 and described in Table 14-8.

Figure 14-13. LCD Status Register (LCDSR)

15	10	9	8
Reserved		EOF1	EOF0
R-0		RW-0	RW-0

7	1	0
Reserved		DONE
R-0		RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 14-8. LCD Status Register (LCDSR) Field Descriptions

Bit	Field	Value	Description
15-10	Reserved	0	Reserved.
9	EOF1		End of Frame 1
		0	No end of frame 1 detected
		1	End of frame 1 detected.
8	EOF0		End of Frame 0
		0	No end of frame 0 detected.
		1	End of frame 0 detected.
7-1	Reserved	0	Reserved.
0	DONE		LIDD Frame Done.
		0	Raster engine enabled.
		1	Raster engine disabled.

14.3.5 LCD LIDD Control Register (LCDLIDDCR)

The LCD LIDD control register (LIDD_CTRL) contains the polarity controls for LIDD output signals (to account for variety in the external LCD display/peripheral signal requirements), and the LIDD type select bits.

NOTE: To activate DMA to drive LIDD interface, all other control bit-fields must be programmed before setting LIDD_DMA_EN = 1 and must also disable LIDD_DMA_EN bit when changing the state of any control bit within the LCD controller.

The LCD LIDD control register (LCDLIDDCR) is shown in Figure 14-14 and described in Table 14-9.

Figure 14-14. LCD LIDD Control Register (LCDLIDDCR)

15	11	10	9	8
Reserved		DONE_INT_EN	DMA_CS0_CS1	LIDD_DMA_EN
R-0		RW-0	RW-0	RW-0

7	6	5	4	3	2	0
CS1_E1_POL	CS0_E0_POL	WS_DIR_POL	RS_EN_POL	RSPOL	LIDD_MODE_SEL	
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 14-9. LCD LIDD Control Register (LCDLIDDCR) Field Descriptions

Bit	Field	Value	Description
15-11	Reserved	0	Reserved
10	DONE_INT_EN		LIDD Frame Done Interrupt Enable
		0	Disable LIDD Frame Done interrupt.
		1	Enable LIDD Frame Done interrupt (seen on LCD Status Reg bit 0).
9	DMA_CS0_CS1		CS0/CS1 Select for LIDD DMA writes
		0	DMA writes to LIDD CS0.
		1	DMA writes to LIDD CS1.
8	LIDD_DMA_EN		LIDD DMA Enable
		0	Deactivate DMA control of LIDD interface; DMA control is released upon completion of transfer of the currentframe of data (LIDD Frame Done) after this bit is cleared. The MPU has direct read/write access to the panel in this mode .
		1	Activate DMA to drive LIDD interface to support streaming data to "smart" panels. The MPU cannot access the panel directly in this mode.
7	CS1_E1_POL		Chip Select 1/Enable 1 (Secondary) Polarity Control.
		0	Do Not Invert Chip Select 1/Enable 1
		1	Invert Chip Select 1/Enable 1 Chip Select 1 is active low by default; Enable 1 is active high by default.
6	CS0_E0_POL		Chip Select 0/Enable 0 (Primary) Polarity Control.
		0	Do Not Invert Chip Select 0/Enable 0
		1	Invert Chip Select 0/Enable 0 Chip Select 0 is active low by default; Enable 0 is active high by default.
5	WS_DIR_POL		Write Strobe/Direction Polarity Control
		0	Do Not Invert Write Strobe/Direction
		1	Invert Write Strobe/Direction Write Strobe/Direction is active low/write low by default.
4	RS_EN_POL		Read Strobe/Enable Polarity Control
		0	Do Not Invert Read Strobe/Enable
		1	IInvert Read Strobe/Enable Read Strobe is active low by default; Enable is active high by default
3	RSPOL		Register Select Polarity Control
		0	Do not invert RS
		1	Invert RS. RS is active low by default

Table 14-9. LCD LIDD Control Register (LCDLIDDCR) Field Descriptions (continued)

<table>
<tr><th>Bit</th><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td>2-0</td><td>LIDD_MODE_SEL</td><td>0-7h</td><td>LIDD Mode Select. Selects type of LCD interface for the LIDD to drive. LIDD_MODE_SEL defines the function of LCD external pins as follows:
<table>
<tr><th>Pin</th><th>000b</th><th>001b</th><th>010b</th><th>011b</th><th>100b</th></tr>
<tr><td>LCD_EN_RDB</td><td>EN</td><td>EN</td><td>RDB</td><td>RDB</td><td>N/A</td></tr>
<tr><td>LCD_RW_WRB</td><td>R/W</td><td>R/W</td><td>WRB</td><td>WRB</td><td>R/W</td></tr>
<tr><td>LCD_RS</td><td>D/C(RS)</td><td>D/C (RS)</td><td>D/C (RS)</td><td>D/C (RS)</td><td>RS</td></tr>
<tr><td>LCD_CS0_E0</td><td>CS0</td><td>CS0</td><td>CS0</td><td>CS0</td><td>E0</td></tr>
<tr><td>LCD_CS1_E1</td><td>LCD_CLK</td><td>CS1</td><td>LCD_CLK</td><td>CS1</td><td>E1</td></tr>
</table></td></tr>
<tr><td></td><td></td><td>0</td><td>MPU6800 Sync mode</td></tr>
<tr><td></td><td></td><td>1h</td><td>MPU 6800 Async mode</td></tr>
<tr><td></td><td></td><td>2h</td><td>MPU80 Sync mode</td></tr>
<tr><td></td><td></td><td>3h</td><td>MPU80 Async mode</td></tr>
<tr><td></td><td></td><td>4h</td><td>Hitachi Async (HD44780)mode</td></tr>
</table>

14.3.6 LCD LIDD CS0 and CS1 Configuration Register 0 (LCDLIDDCS0CONFIG0 and LCDLIDDCS1CONFIG0)

The LCD LIDD CS0 and CS1 Configuration Register 0 (LCDLIDDCS0CONFIG0 and LCDLIDDCS1CONFIG0) provides the capability to configure write and read strobe timing parameters to meet a variety of interface timing requirements for the chip select 0 (primary) device and chip select 1 (secondary) device, respectively. These values are in LCD_CLK cycles; LCD_CLK is divided down from system clock as defined by the CLKDIV field in the LCD control register.

The LLCD LIDD CS0 and CS1 Configuration Register 0 (LCDLIDDCS0CONFIG0 and LCDLIDDCS1CONFIG0) is shown in Figure 14-15 and described in Table 14-10.

Figure 14-15. LCD LIDD CS0 and CS1 Configuration Register 0 (LCDLIDDCS0CONFIG0 and LCDLIDDCS1CONFIG0)

15 12	11 6	5 2	1 0
R_SU0	R_STROBE	R_HOLD	TA
RW-0	RW-1	RW-1	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 14-10. LCD LIDD CS0 and CS1 Configuration Register 0 (LCDLIDDCS0CONFIG0 and LCDLIDDCS1CONFIG0) Field Descriptions

Bit	Field	Value	Description
15-12	R_SU0	0-Fh	Read Strobe Set-Up cycles (lower 4 bits upper bit is in LIDD CSO config reg 1 at bit-0) Field value defines number of LCD_CLK cycles after Data Bus/Pad Output Enable, ALE, Direction bit and Chip Select 0 have been set-up before the Read Strobe is asserted when performing a read access.
11-6	R_STROBE	0-3Fh	Read Strobe Duration cycles Field value defines number of LCD_CLK cycles for which the Read Strobe is held active when performing a read access.
5-2	R_HOLD	0-Fh	Read Strobe Hold cycles Field value defines number of LCD_CLK cycles for which Data Bus/Pad Output Enable, ALE, Direction bit and Chip Select 0 are held after the Read Strobe is deasserted when performing a read access.
1-0	TA	0-3h	Field value defines number of LCD_CLK cycles between the end of one CS0 device access and the start of another CS0 device access unless the two accesses are both reads, in which case this delay is not incurred.

14.3.7 LCD LIDD CS0 and CS1 Configuration Register 1 (LCDLIDDCS0CONFIG1 and LCDLIDDCS1CONFIG1)

The LCD LIDD CS0 and CS1 Configuration Register 1 (LCDLIDDCS0CONFIG1 and LCDLIDDCS1CONFIG1) provides the capability to configure write and read strobe timing parameters to meet a variety of interface timing requirements for the chip select 0 (primary) device and chip select 1 (secondary) device, respectively. These values are in LCD_CLK cycles; LCD_CLK is divided down from system clock as defined by the CLKDIV field in the LCD control register.

The LCD LIDD CS0 and CS1 Configuration Register 1 (LCDLIDDCS0CONFIG1 and LCDLIDDCS1CONFIG1) is shown in Figure 14-16 and described in Table 14-11.

Figure 14-16. LCD LIDD CS0 and CS1 Configuration Register 1 (LClidd_cs1_1DLIDDCS0CONFIG1 and LCDLIDDCS1CONFIG1)

15 11	10 5	4 1	0
W_SU	W_STROBE	W_HOLD	R_SU1
RW-0	RW-1	RW-1	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 14-11. LCD LIDD CS0 and CS1 Configuration Register 1 (LCDLIDDCS0CONFIG1 and LCDLIDDCS1CONFIG1) Field Descriptions

Bit	Field	Value	Description
15-11	W_SU	0-1Fh	Write Strobe Set-Up cycles Field value defines number of LCD_CLK cycles after Data Bus/Pad Output Enable, ALE, Direction bit and Chip Select 0 have been set-up before the Write Strobe is asserted when performing a write access.
10-5	W_STROBE	0-3Fh	Write Strobe Duration cycles Field value defines number of LCD_CLK cycles for which the Write Strobe is held active when performing a write access.
4-1	W_HOLD	0-Fh	Write Strobe Hold cycles Field value defines number of LCD_CLK cycles for which Data Bus/Pad Output Enable, ALE, Direction bit and Chip Select 0 are held after the Write Strobe is deasserted when performing a write access.
0	R_SU1	1-0	Most Significant Bit for Read Strobe Set-Up cycles (upper bit. The lower 4 bits are located in LIDD CSO confir reg 0) Field value defines number of LCD_CLK cycles after Data Bus/Pad Output Enable, ALE, Direction bit and Chip Select 0 have been set-up before the Read Strobe is asserted when performing a read access.

14.3.8 LCD LIDD CS0 and CS1 Address Read/Write Register (LCDLIDDCS0ADDR and LCDLIDDCS1ADDR)

The LCD LIDD CS0 and SC1 Address Read/Write Register (LCDLIDDCS0ADDR and LCDLIDDCS1ADDR) are accessed by the processor to perform the address/index read or write operations on the CS0 and CS1 device respectively. Writing to LCDLIDDCS0ADDR asserts CS0 and Address Latch Enable, which loads the ADR_INDX field of this register into the address generator of the peripheral device. Likewise, reading from LCDLIDDCS0ADDR asserts CS0 and Address Latch Enable, which loads status information from the peripheral device into the ADR_INDX field of this register. Similarly writing to LCDLIDDCS1ADDR asserts CS1 and Address Latch Enable, which loads the ADR_INDX field of this register into the address generator of the peripheral device. Likewise, reading from LCDLIDDCS1ADDR asserts CS1 and Address Latch Enable, which loads status information from the peripheral device into the ADR_INDX field of this register.

The LCD LIDD CS0 and SC1 Address Read/Write Register (LCDLIDDCS0ADDR and LCDLIDDCS1ADDR) is shown in Figure 14-17 and described in Table 14-12.

Figure 14-17. LCD LIDD CS0 and CS1 Address Read/Write Register (LCDLIDDCS0ADDR and LCDLIDDCS1ADDR)

15	0
ADR_INDX	
RW-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 14-12. LCD LIDD CS0 and CS1 Address Read/Write Register (LCDLIDDCS0ADDR and LCDLIDDCS1ADDR) Field Descriptions

Bit	Field	Value	Description
15-0	ADR_INDX	0-FFFFh	Peripheral Device Address/Index value. On writes, this field is loaded into the CSn peripheral device's address generator. On reads, this field contains the CSn peripheral device's status

14.3.9 LCD LIDD CS0 and CS1 Data Read/Write Register (LCDLIDDCS0DATA and LCDLIDDCS1DATA)

The LCD LIDD CS0 and CS1 Data Read/Write Register (LCDLIDDCS0DATA and LCDLIDDCS1DATA) are accessed by the processor to perform the data read or write operations on the CS0 and CS1 device respectively . Writing to LCDLIDDCS0DATA asserts CS0 and deasserts Address Latch Enable, which loads the DATA field of this register into the peripheral device. Likewise, reading from this register asserts CS0 and deasserts Address Latch Enable, which loads data from the peripheral device into the DATA field of this register. Similarly writing to LCDLIDDCS1DATA asserts CS1 and deasserts Address Latch Enable, which loads the DATA field of this register into the peripheral device. Likewise, reading from LCDLIDDCS1DATA asserts CS1 and deasserts Address Latch Enable, which loads data from the peripheral device into the DATA field of this register.

The LCD LIDD CS0 and CS1 Data Read/Write Register (LCDLIDDCS0DATA and LCDLIDDCS1DATA) is shown in Figure 14-18 and described in Table 14-13.

Figure 14-18. LCD LIDD CS0 and CS1 Data Read/Write Register (LCDLIDDCS0DATA and LCDLIDDCS1DATA)

15 - 0
DATA
RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 14-13. LCD LIDD CS0 and CS1 Data Read/Write Register (LCDLIDDCS0DATA and LCDLIDDCS1DATA) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Peripheral Device Data value. On writes, this field is loaded into the CSn peripheral device. On reads, this field contains the CSn peripheral device's data.

14.3.10 LCD DMA Control Register (LCDDMACR)

The LCD DMA control register (LCDDMACR) is shown in Figure 14-19 and described in Table 14-14.

Figure 14-19. LCD DMA Control Register (LCDDMACR)

15	8
Reserved	
R-0	

7	6	4	3	2	1	0
Reserved	BURST_SIZE		Reserved	EOF_INTEN	BIGENDIAN	FRAME_MODE
R-0	RW-0		R-0	RW-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 14-14. LCD DMA Control Register (LCDDMACR) Field Descriptions

Bit	Field	Value	Description
15-7	Reserved	0	Reserved.
6-4	BURST_SIZE		Burst Size setting for DMA transfers (all DMA transfers are 32 bits wide).
		0	Burst size of 1
		1	Burst size of 2
		2h	Burst size of 4
		3h	Burst size of 8
		4h	Burst size of 16
		5h-7h	Reserved.
3	Reserved	0	Reserved.
2	EOF_INTEN		End of Frame interrupt enable Setting this bit allows the End of Frame 0 or 1 Status bits in the LCD Status Register to trigger an interrupt 0: End of Frame 0/1Interrupt disabled 1: End of Frame 0/1Interrupt enabled.
		0	EoF Interrupt disabled.
		1	EoF Interrupt enabled.
1	BIGENDIAN		Big Endian enable Use this bit when the processor is operating in Big Endian mode and writes to the frame buffer(s) are less than 32 bits wide; only in this scenario we need to change the byte alignment for data coming into the FIFO from the frame buffer(s) 0: Big Endian data reordering disabled 1: Big Endian data reordering enabled.
		0	Big Endian disabled.
		1	Big Endian enabled.
0	FRAME_MODE		Frame Mode 0: one frame buffer (FB0 only) used 1: two frame buffers used; DMA ping-pongs between FB0 and FB1 in this mode
		0	One Frame buffer used
		1	Two frame buffers used.

14.3.11 LCD DMA Frame Buffer 0 Base Address Register 0 (LCDDMAFB0BAR0)

The LCD DMA frame buffer 0 base address register 0 is shown in Figure 14-20 and described in Table 14-15.

Figure 14-20. LCD DMA Frame Buffer 0 Base Address Register 0 (LCDDMAFB0BAR0)

15	0
FB0_BASE0	
RW-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 14-15. LCD DMA Frame Buffer 0 Base Address Register 0 (LCDDMAFB0BAR0) Field Descriptions

Bit	Field	Value	Description
15-0	FB0_BASE0	0-FFFFh	Frame Buffer 0 Base Address pointer LSW; 2 LSBs are hardwired to be 00.

14.3.12 LCD DMA Frame Buffer 0 Base Address Register 1 (LCDDMAFB0BAR1)

The LCD DMA frame buffer 0 base address register 1 is shown in Figure 14-21 and described in Table 14-16.

Figure 14-21. LCD DMA Frame Buffer 0 Base Address Register 1 (LCDDMAFB0BAR1)

15	0
FB0_BASE1	
RW-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 14-16. LCD DMA Frame Buffer 0 Base Address Register 1 (LCDDMAFB0BAR1) Field Descriptions

Bit	Field	Value	Description
15-0	FB0_BASE1	0-FFFFh	Frame Buffer 0 Base Address pointer MSW.

14.3.13 LCD DMA Frame Buffer 0 Ceiling Address Register 0 (LCDDMAFB0CAR0)

The LCD DMA frame buffer 0 ceiling address register 0 is shown in Figure 14-22 and described in Table 14-17.

Figure 14-22. LCD DMA Frame Buffer 0 Ceiling Address Register 0 (LCDDMAFB0CAR0)

15	0
FB0_CEIL0	
RW-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 14-17. LCD DMA Frame Buffer 0 Ceiling Address Register 0 (LCDDMAFB0CAR0) Field Descriptions

Bit	Field	Value	Description
15-0	FB0_CEIL0	0-FFFFh	Frame Buffer 0 Ceiling Address pointer LSW; 2 LSBs are hardwired to be 00.

14.3.14 LCD DMA Frame Buffer 0 Ceiling Address Register 1 (LCDDMAFB0CAR1)

The LCD DMA frame buffer 0 ceiling address register 1 is shown in Figure 14-23 and described in Table 14-18.

Figure 14-23. LCD DMA Frame Buffer 0 Ceiling Address Register 1 (LCDDMAFB0CAR1)

15	0
FB0_CEIL1	
RW-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 14-18. LCD DMA Frame Buffer 0 Ceiling Address Register 1 (LCDDMAFB0CAR1) Field Descriptions

Bit	Field	Value	Description
15-0	FB0_CEIL1	0-FFFFh	Frame Buffer 0 Ceiling Address pointer MSW.

14.3.15 LCD DMA Frame Buffer 1 Base Address Register 0 (LCDDMAFB1BAR0)

The LCD DMA frame buffer 1 base address register 0 is shown in Figure 14-24 and described in Table 14-19.

Figure 14-24. LCD DMA Frame Buffer 1 Base Address Register 0 (LCDDMAFB1BAR0)

15 … 0
FB1_BASE0
RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 14-19. LCD DMA Frame Buffer 1 Base Address Register 0 (LCDDMAFB1BAR0) Field Descriptions

Bit	Field	Value	Description
15-0	FB1_BASE0	0-FFFFh	Frame Buffer 1 Base Address pointer LSW; 2 LSBs are hardwired to be 00.

14.3.16 LCD DMA Frame Buffer 1 Base Address Register 1 (LCDDMAFB1BAR1)

The LCD DMA frame buffer 1 base address register 1 is shown in Figure 14-25 and described in Table 14-20.

Figure 14-25. LCD DMA Frame Buffer 1 Base Address Register 1 (LCDDMAFB1BAR1)

15 … 0
FB1_BASE1
RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 14-20. LCD DMA Frame Buffer 1 Base Address Register 1 (LCDDMAFB1BAR1) Field Descriptions

Bit	Field	Value	Description
15-0	FB1_BASE1	0-FFFFh	Frame Buffer 1 Base Address pointer MSW.

14.3.17 LCD DMA Frame Buffer 1 Ceiling Address Register 0 (LCDDMAFB1CAR0)

The LCD DMA frame buffer 1 ceiling address register 0 is shown in Figure 14-26 and described in Table 14-21.

Figure 14-26. LCD DMA Frame Buffer 1 Ceiling Address Register 0 (LCDDMAFB1CAR0)

<table>
<tr><td>15</td><td align="right">0</td></tr>
<tr><td colspan="2" align="center">FB1_CEIL0</td></tr>
<tr><td colspan="2" align="center">RW-0</td></tr>
</table>

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 14-21. LCD DMA Frame Buffer 1 Ceiling Address Register 0 (LCDDMAFB1CAR0) Field Descriptions

Bit	Field	Value	Description
15-0	FB1_CEIL0	0-FFFFh	Frame Buffer 1 Ceiling Address pointer LSW; 2 LSBs are hardwired to be 00.

14.3.18 LCD DMA Frame Buffer 1 Ceiling Address Register 1 (LCDDMAFB1CAR1)

The LCD DMA frame buffer 1 ceiling address register 1 is shown in Figure 14-27 and described in Table 14-22.

Figure 14-27. LCD DMA Frame Buffer 1 Ceiling Address Register 1 (LCDDMAFB1CAR1)

<table>
<tr><td>15</td><td align="right">0</td></tr>
<tr><td colspan="2" align="center">FB1_CEIL1</td></tr>
<tr><td colspan="2" align="center">RW-0</td></tr>
</table>

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 14-22. LCD DMA Frame Buffer 1 Ceiling Address Register 1 (LCDDMAFB1CAR1) Field Descriptions

Bit	Field	Value	Description
15-0	FB1_CEIL1	0-FFFFh	Frame Buffer 1 Ceiling Address pointer MSW.

Revision History

Table A-1 lists the changes made to the SPRUH87B version of this document to make it a SPRUH87C revision.

Table A-1. Document Revision History

Reference	Additions/Modifications/Deletions
Chapter 1, *System Control*	Changed description of Table 1-7, *CLKOUT Control Source Select Register (CCSSR) Field Descriptions*.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Mobile Processors	www.ti.com/omap		
Wireless Connectivity	www.ti.com/wirelessconnectivity		
		TI E2E Community Home Page	e2e.ti.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265