TMS320C55x DSP Peripherals Overview Reference Guide

Preliminary Draft

Literature Number: SPRU317F
October 2003

TEXAS INSTRUMENTS

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Read This First

About This Manual

This manual is for the peripherals that are on the digital signal processors (DSPs) in the TMS320C55x™ (C55x™) DSP generation.

This manual is in transition. Some peripheral information has been revised and moved from chapters in this manual to separate documents. These separate documents are referenced in Chapter 1. The peripheral information that is still in this manual is being revised and will be in separate documents in the future. When the revisions are complete, this manual is to remain as an overview that points to all of the separate documents. In some cases, information has been moved from this manual to the device-specific data manuals.

Notational Conventions

This document uses the following conventions:

❑ If an underscore is appended to the name of a signal (for example, RESET_), the signal is active low.

❑ In most cases, hexadecimal numbers are shown with the suffix h. For example, the following number is a hexadecimal 40 (decimal 64):

  40h

  Similarly, binary numbers usually are shown with the suffix b. For example, the following number is the decimal number 4 shown in binary form:

  0100b

❑ Register bits are sometimes referenced with the following notations:

Notation	Description	Example
Register(n−m)	Bits n through m of Register	A(15−0) represents bits 15 through 0 of the register A.
E[n:m]	Range of bits En through Em	E[7:0] represents E7, E6, E5, E4, E3, E2, E1, and E0.

- The following terms are used to name portions of data:

Term	Description	Example
LSByte	Least significant byte	In A(15−0), bits 7−0 are the LSByte.
MSByte	Most significant byte	In A(15−0), bits 15−8 are the MSByte.

Related Documentation From Texas Instruments

The following documents describe the C55x devices and related support tools. Copies of these documents are available on the Internet at www.ti.com. *Tip:* Enter the literature number in the search box provided at www.ti.com.

TMS320VC5501 Fixed-Point Digital Signal Processor Data Manual (literature number SPRS206) describes the features of the TMS320VC5501 fixed-point DSP and provides signal descriptions, pinouts, electrical specifications, and timings for the device.

TMS320VC5502 Fixed-Point Digital Signal Processor Data Manual (literature number SPRS166) describes the features of the TMS320VC5502 fixed-point DSP and provides signal descriptions, pinouts, electrical specifications, and timings for the device.

TMS320VC5509 Fixed-Point Digital Signal Processor Data Manual (literature number SPRS163) describes the features of the TMS320VC5509 fixed-point DSP and provides signal descriptions, pinouts, electrical specifications, and timings for the device.

TMS320VC5509A Fixed-Point Digital Signal Processor Data Manual (literature number SPRS205) describes the features of the TMS320VC5509A fixed-point DSP and provides signal descriptions, pinouts, electrical specifications, and timings for the device.

TMS320VC5510 Fixed-Point Digital Signal Processor Data Manual (literature number SPRS076) describes the features of the TMS320VC5510 fixed-point DSP and provides signal descriptions, pinouts, electrical specifications, and timings for the device.

TMS320C55x Technical Overview (literature number SPRU393). This overview is an introduction to the TMS320C55x DSPs, the latest generation of fixed-point DSPs in the TMS320C5000™ DSP platform. Like the previous generations, this processor is optimized for high performance and low-power operation. This book describes the CPU architecture, low-power enhancements, and embedded emulation features.

TMS320C55x DSP CPU Reference Guide (literature number SPRU371) describes the architecture, registers, and operation of the CPU for the TMS320C55x DSPs.

TMS320C55x DSP Algebraic Instruction Set Reference Guide (literature number SPRU375) describes the TMS320C55x DSP algebraic instructions individually. Also includes a summary of the instruction set, a list of the instruction opcodes, and a cross-reference to the mnemonic instruction set.

TMS320C55x DSP Mnemonic Instruction Set Reference Guide (literature number SPRU374) describes the TMS320C55x DSP mnemonic instructions individually. Also includes a summary of the instruction set, a list of the instruction opcodes, and a cross-reference to the algebraic instruction set.

TMS320C55x Optimizing C/C++ Compiler User's Guide (literature number SPRU281) describes the TMS320C55x™ C/C++ Compiler. This C/C++ compiler accepts ISO standard C and C++ source code and produces assembly language source code for TMS320C55x devices.

TMS320C55x Assembly Language Tools User's Guide (literature number SPRU280) describes the assembly language tools (assembler, linker, and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for TMS320C55x devices.

TMS320C55x DSP Programmer's Guide (literature number SPRU376) describes ways to optimize C and assembly code for the TMS320C55x DSPs and explains how to write code that uses special features and instructions of the DSPs.

Trademarks

TMS320C5000, TMS320C55x, and C55x are trademarks of Texas Instruments.

MultiMediaCard is a trademark of the MultiMediaCard Association.

Other trademarks are the property of their respective owners.

This page is intentionally left blank.

Contents

1 TMS320C55x DSP Peripherals Overview 1-1

Gives an overview of which peripherals are available on the various TMS320C55x DSPs. Directs you to a chapter in this manual or to another document for details about each peripheral.

2 Analog-to-Digital Converter (ADC) 2-1

[The ADC is described in the TMS320VC5509 DSP Analog-to-Digital Converter (ADC) Reference Guide *(SPRU586).]*

3 Clock Generator 3-1

Describes the clock generator that is in TMS320VC5509, TMS320VC5509A, and TMS320VC5510 DSPs. [For information about clock generation in TMS320VC5501 and TMS320VC5502 DSPs, see the device-specific data manuals.]

3.1 Introduction to the DSP Clock Generator 3-2
3.2 Operational Flow of the DSP Clock Generator 3-3
3.3 Bypass Mode 3-5
3.3.1 Entering and Exiting the Bypass Mode 3-5
3.3.2 CLKMD Bits Used in the Bypass Mode 3-5
3.3.3 Setting the Output Frequency for the Bypass Mode 3-5
3.4 Lock Mode 3-6
3.4.1 Entering and Exiting the Lock Mode 3-6
3.4.2 CLKMD Bits Used in the Lock Mode 3-6
3.4.3 Setting the Output Frequency for the Lock Mode 3-7
3.4.4 Calculating the Lock Time 3-7
3.5 Idle (Low-Power) Mode 3-8
3.6 The CLKOUT Pin and the Associated Clock Divider 3-9
3.7 DSP Reset Conditions of the DSP Clock Generator 3-10
3.7.1 Clock Generator During Reset 3-10
3.7.2 Clock Generator After Reset 3-10
3.8 Clock Mode Register 3-12

4 Direct Memory Access (DMA) Controller 4-1

[The DMA controller that is in TMS320VC5501 and TMS320VC5502 DSPs is described in the TMS320VC5501/5502 DSP Direct Memory Access (DMA) Controller Reference Guide *(SPRU613). The DMA controller that is in TMS320VC5509, TMS320VC5509A, and TMS320VC5510 DSPs is described in the* TMS320VC5509/5510 DSP Direct Memory Access (DMA) Controller Reference Guide *(SPRU587).]*

5 External Memory Interface (EMIF) 5-1

[For the EMIF information that applies to your C55x device, see the corresponding device-specific reference guide: TMS320VC5501/5502 DSP External Memory Interface (EMIF) Reference Guide *(SPRU621),* TMS320VC5509 DSP External Memory Interface (EMIF) Reference Guide *(SPRU670), or* TMS320VC5510 DSP External Memory Interface (EMIF) Reference Guide *(SPRU590).]*

6 General-Purpose I/O Port 6-1

[Information about the general-purpose I/O port has been moved to the device-specific data manuals.]

7 Host Port Interface (HPI) 7-1

[For the HPI information that applies to your C55x device, see the corresponding device-specific reference guide: TMS320VC5501/5502 DSP Host Port Interface (HPI) Reference Guide *(SPRU620),* TMS320VC5509 DSP Host Port Interface (HPI) Reference Guide *(SPRU619), or* TMS320VC5510 DSP Host Port Interface (HPI) Reference Guide *(SPRU588).]*

8 Idle Configurations 8-1

Describes how to deactivate and reactivate specific domains within a TMS320C55x DSP. [Check the device-specific data manual for additional information about deactivating and reactivating these domains.]

8.1 Idle Domains 8-2
8.2 Idle Configuration Process 8-4
8.3 Valid Idle Configurations 8-5
8.4 To Change Idle Configurations (Key Conditions) 8-6
8.4.1 Condition 1: CLKGEN and CPU Domains Active 8-6
8.4.2 Condition 2: CLKGEN Domain Active, CPU Domain Idle 8-7
8.4.3 Condition 3: CLKGEN Domain Idle 8-7
8.5 Interrupt Handling When the CPU Is Reactivated 8-8
8.6 Effect of a DSP Reset on the Idle Domains 8-8
8.7 Idle Registers 8-9

9 Instruction Cache 9-1

[The instruction cache that is in TMS320VC5501 and TMS320VC5502 DSPs is described in the TMS320VC5501/5502 DSP Instruction Cache Reference Guide *(SPRU630). The instruction cache that is in TMS320VC5510 DSPs is described in the* TMS320VC5510 DSP Instruction Cache Reference Guide *(SPRU576).]*

10 Inter-Integrated Circuit (I2C) Module 10-1
[Information about the I2C module is in the TMS320VC5501/5502/5509 DSP Inter-Integrated Circuit (I2C) Module Reference Guide *(SPRU146).]*

11 Multichannel Buffered Serial Port (McBSP) 11-1
[McBSP information is in the TMS320VC5501/5502/5509/5510 DSP Multichannel Buffered Serial Port (McBSP) Reference Guide *(SPRU592).]*

12 MultiMediaCard / SD Card Controller 12-1
[Information about the MultiMediaCard / SD card controller is in the TMS320VC5509 DSP MultiMediaCard / SD Card Controller Reference Guide *(SPRU593).]*

13 Real-Time Clock (RTC) 13-1
[Information about the RTC is in the TMS320VC5509 DSP Real-Time Clock (RTC) Reference Guide *(SPRU594).]*

14 System Control Registers 14-1
[Information about the system control registers has been moved to the device-specific data manuals.]

15 Timer (General-Purpose) 15-1
[The general-purpose timer that is in TMS320VC5501 and TMS320VC5502 DSPs is described in the TMS320VC5501/5502 DSP Timers Reference Guide *(SPRU618). The general-purpose timer that is in TMS320VC5509, TMS320VC5509A, and TMS320VC5510 DSPs is described in the* TMS320VC5509/5510 DSP Timers Reference Guide *(SPRU595).]*

16 Universal Asynchronous Receiver/Transmitter (UART) 16-1
[UART information is in the TMS320VC5501/5502 DSP Universal Asynchronous Receiver/Transmitter (UART) Reference Guide *(SPRU597).]*

17 Universal Serial Bus (USB) Module 17-1
Describes the USB module that is in TMS320VC5509 and TMS320VC5509A DSPs.

17.1 USB Concepts Overview 17-2
17.1.1 Terminology 17-2
17.1.2 Data Toggle Mechanism 17-4
17.2 Introduction to the USB Module 17-5
17.2.1 Block Diagram of the USB Module 17-6
17.2.2 Transferring Data Between the USB Host and the DSP Memory 17-9
17.2.3 Clock Generation for the USB Module 17-9
17.3 USB Buffer Manager (UBM) 17-11

17.4 USB DMA Controller 17-13
17.4.1 Advantage of Using the USB DMA Controller 17-13
17.4.2 Things To Consider 17-13
17.4.3 Interaction Between the CPU and the USB DMA Controller 17-14
17.4.4 Automatic Alternating Accesses of the X and Y Buffers 17-18
17.4.5 DMA Reload Operation (Automatic Register Swapping) 17-18
17.4.6 Transfer Count Saved to DSP Memory For an OUT Transfer 17-19
17.4.7 Configuring the USB DMA Controller 17-20
17.4.8 Monitoring CPU-Initiated DMA Transfers 17-24
17.4.9 USB DMA State Tables and State Diagrams 17-26
17.5 Host-DMA Mode: Host-Initiated Direct Memory Accesses 17-40
17.5.1 Protocol Header For the Host-DMA Mode 17-40
17.5.2 Initiating a Host Read Operation (Data to Host) 17-42
17.5.3 Initiating a Host Write Operation (Data to DSP Memory) 17-42
17.5.4 Configuring the USB Module for the Host-DMA Mode 17-42
17.6 Interrupt Activity in the USB Module 17-46
17.6.1 USB Bus Interrupt Requests 17-47
17.6.2 Endpoint Interrupt Requests 17-48
17.6.3 USB DMA Interrupt Requests 17-50
17.6.4 Host-DMA Mode Interrupt Requests 17-52
17.7 Power, Emulation, and Reset Considerations 17-53
17.7.1 Putting the USB Module into Its Idle Mode 17-53
17.7.2 USB Module Indirectly Affected By Certain Idle Configurations 17-53
17.7.3 USB Module During Emulation 17-53
17.7.4 Resetting the USB Module 17-54
17.8 USB Module Registers 17-55
17.8.1 High-Level Summary of USB Module Registers 17-55
17.8.2 DMA Context Registers 17-59
17.8.3 Descriptor Registers for IN and OUT Endpoints 1−7 17-68
17.8.4 Descriptor Registers for IN and OUT Endpoints 0 17-82
17.8.5 Interrupt Registers 17-85
17.8.6 Host-DMA Mode Registers 17-93
17.8.7 General Control and Status Registers 17-95

18 Watchdog Timer 18-1

[The watchdog timer that is in TMS320VC5501 and TMS320VC5502 DSPs is described in the TMS320VC5501/5502 DSP Timers Reference Guide *(SPRU618). The watchdog timer that is in TMS320VC5509 and TMS320VC5509A DSPs is described in the* TMS320VC5509/5510 DSP Timers Reference Guide *(SPRU595).]*

A Revision History A-1

Describes the notable changes that were made to this document since its last revision.

Figures

3−1 Operational Flow of the DSP Clock Generator 3-3
3−2 Dividing the CPU Clock for the CLKOUT Pin 3-9
3−3 Clock Mode Register (CLKMD) 3-12
8−1 Idle Configuration Process 8-4
8−2 Idle Configuration Register (ICR) and Idle Status Register (ISTR) 8-9
17−1 Conceptual Block Diagram of the USB Module 17-6
17−2 Path for Data Transferred Between the Host and the DSP Memory 17-9
17−3 Clock Generation for the USB Module 17-10
17−4 USB Clock Mode Register (USBCLKMD) 17-10
17−5 Role of a NAK bit in UBM Activity (at an Individual Endpoint) 17-12
17−6 Activity for CPU-Initiated DMA Transfers 17-16
17−7 Storage of Transfer Count For an OUT Transfer 17-20
17−8 The Effect of END = 1 on USB DMA Transfers 17-23
17−9 Missing Packet Response for Isochronous IN DMA Transfer
(Supports Table 17−5 on Page 17-31) 17-38
17−10 Missing Packet Response for Isochronous OUT DMA Transfer
(Supports Table 17−6 on Page 17-35) 17-39
17−11 Bytes of the Host-DMA Protocol Header 17-41
17−12 Possible Sources of a USB Interrupt Request 17-46
17−13 Enable Paths of USB Bus Interrupt Requests 17-48
17−14 Enable Paths for the Endpoint Interrupt Requests 17-49
17−15 Enable Paths for the USB DMA Interrupt Requests 17-51
17−16 Enable Paths for the Host-DMA Mode Interrupt Requests 17-52
17−17 USB DMA Control Register (USBxDCTLn) 17-61
17−18 USB DMA Address Registers (USBxDADLn and USBxDADHn) 17-65
17−19 DMA Size Register (USBxDSIZn) 17-65
17−20 DMA Count Register (USBxDCTn) 17-66
17−21 DMA Reload-Address Registers (USBxDRALn and USBxDRAHn) 17-67
17−22 DMA Reload-Size Register (USBxDRSZn) 17-68
17−23 IN Endpoint n Configuration Register (USBICNFn) 17-70
17−24 OUT Endpoint n Configuration Register (USBOCNFn) 17-72
17−25 Endpoint Buffer Base Address Registers 17-74
17−26 Endpoint n Count Registers 17-75
17−27 IN Endpoint n Extended Count Values in the Isochronous Mode (ISO = 1) 17-77
17−28 OUT Endpoint n Extended Count Values in the Isochronous Mode (ISO = 1) 17-78
17−29 Endpoint n X-/Y-Buffer Size Register (USBxSIZn) 17-78
17−30 Extended Size Values in the Isochronous Mode (ISO = 1) 17-80

17−31 Endpoint Buffer Size and Count Extension Registers 17-81
17−32 Endpoint 0 Configuration Register (USBxCNF0) 17-82
17−33 Count Register for IN Endpoint 0 (USBICT0) 17-84
17−34 Count Register for OUT Endpoint 0 (USBOCT0) 17-84
17−35 Interrupt Source Register (USBINTSRC) 17-86
17−36 Endpoint Interrupt Flag Register for OUT Endpoints (USBOEPIF) 17-88
17−37 Endpoint Interrupt Flag Register for IN Endpoints (USBIEPIF) 17-88
17−38 Endpoint Interrupt Enable Register for OUT Endpoints (USBOEPIE) 17-89
17−39 Endpoint Interrupt Enable Register for IN Endpoints (USBIEPIE) 17-89
17−40 DMA GO Interrupt Flag Register for OUT Endpoints (USBODGIF) 17-90
17−41 DMA GO Interrupt Flag Register for IN Endpoints (USBIDGIF) 17-90
17−42 DMA RLD Interrupt Flag Register for OUT Endpoints (USBODRIF) 17-91
17−43 DMA RLD Interrupt Flag Register for IN Endpoints (USBIDRIF) 17-91
17−44 DMA Interrupt Enable Register for OUT Endpoints (USBODIE) 17-92
17−45 DMA Interrupt Enable Register for IN Endpoints (USBIDIE) 17-92
17−46 Host Control Register (USBHCTL) 17-93
17−47 Host Endpoint Select Register (USBHEPSEL) 17-94
17−48 Host Status Register (USBHSTAT) 17-94
17−49 Global Control Register (USBGCTL) 17-95
17−50 Frame Number (High Part) Register (USBFNUMH) 17-96
17−51 Frame Number (Low Part) Register (USBFNUML) 17-96
17−52 PSOF Interrupt Timer Counter (USBPSOFTMR) 17-96
17−53 USB Control Register (USBCTL) 17-97
17−54 USB Interrupt Flag Register (USBIF) 17-99
17−55 USB Interrupt Enable Register (USBIE) 17-100
17−56 USB Device Address Register (USBADDR) 17-101
17−57 USB Idle Control Register (USBIDLECTL) 17-102

Tables

1−1 TMS320C55x DSP Peripherals 1-1
3−1 Operational States Shown in Figure 3−1 3-4
3−2 CLKMD Bits Used in the Bypass Mode 3-5
3−3 CLKMD Bits Used in the Lock Mode 3-6
3−4 Examples of Selecting a Lock Mode Frequency 3-7
3−5 Effect of CLKDIV Bits on CLKOUT Frequency 3-9
3−6 Reset Values of CLKMD Bits and The Effects 3-11
3−7 Bit Field Descriptions for the Clock Mode Register (CLKMD) 3-12
8−1 Idle Domains in the DSP 8-2
8−2 Changing Idle Configurations 8-6
8−3 CPU Response After Reactivation 8-8
8−4 ICR Bit Descriptions 8-9
8−5 ISTR Bit Descriptions 8-11
17−1 CPU-Initiated DMA Transfers 17-15
17−2 Primary USB DMA Size and Address Registers and the Corresponding Reload Registers 17-18
17−3 State Table: Non-Isochronous IN DMA Transfer 17-27
17−4 State Table: Non-Isochronous OUT DMA Transfer 17-29
17−5 State Table: Isochronous IN DMA Transfer 17-31
17−6 State Table: Isochronous OUT DMA Transfer 17-35
17−7 The Fields of Byte 1 of the Host-DMA Protocol Header 17-41
17−8 Descriptions of the USB Bus Interrupt Requests 17-47
17−9 High-Level Summary of the USB Module Registers 17-56
17−10 USB DMA Context Registers for Each OUT or IN Endpoint n (n = 1, 2, 3, 4, 5, 6, or 7) 17-60
17−11 USBxDCTLn Bit Descriptions 17-61
17−12 USBxDADLn and USBxDADHn Bit Descriptions 17-65
17−13 USBxDSIZn Bit Description 17-66
17−14 USBxDCTn Bit Description 17-66
17−15 USBxDRALn and USBxDRAHn Bit Descriptions 17-67
17−16 USBxDRSZn Bit Description 17-68
17−17 Descriptor Registers for Each OUT or IN Endpoint n (n = 1, 2, 3, 4, 5, 6, or 7) 17-69
17−18 USBICNFn Bit Descriptions 17-70
17−19 USBOCNFn Bit Descriptions 17-72
17−20 Endpoint Buffer Base Address Register Bit Descriptions 17-74
17−21 Endpoint n Count Register Bit Descriptions 17-76

17−22 Endpoint Size Register Bit Descriptions 17-79
17−23 Endpoint Size and Count Extension Register Bit Descriptions 17-82
17−24 USBxCNF0 Bit Descriptions 17-83
17−25 USBxCT0 Bit Descriptions 17-84
17−26 Interrupt Source Register (USBINTSRC) Field Values 17-86
17−27 Interrupt Sources Matched to INTSRC Values 17-87
17−28 Endpoint Interrupt Flag Register (USBxEPIF) Field Values 17-88
17−29 Endpoint Interrupt Enable Register (USBxEPIE) Field Values 17-89
17−30 DMA GO Interrupt Flag Register (USBxDGIF) Field Values 17-90
17−31 DMA RLD Interrupt Flag Register (USBxDRIF) Field Values 17-91
17−32 DMA Interrupt Enable Register (USBxDIE) Field Values 17-92
17−33 Host Control Register (USBHCTL) Field Values 17-93
17−34 Host Endpoint Select Register (USBHEPSEL) Field Values 17-94
17−35 Host Status Register (USBHSTAT) Field Values 17-94
17−36 Global Control Register (USBGCTL) Field Values 17-95
17−37 Frame Number Register (USBFNUMH and USBFNUML) Field Values 17-96
17−38 PSOF Interrupt Timer Counter (USBPSOFTMR) Field Values 17-97
17−39 USB Control Register (USBCTL) Field Values 17-97
17−40 USB Interrupt Flag Register (USBIF) Field Values 17-99
17−41 USB Interrupt Enable Register (USBIE) Field Values 17-100
17−42 USB Device Address Register (USBADDR) Field Values 17-102
17−43 USB Idle Control Register (USBIDLECTL) Field Values 17-102

Examples

17−1 DMA Reload Operation for OUT Endpoint 3 17-19
17−2 Loading the Endpoint Buffer Base Addresses 17-75

This page is intentionally left blank.

Chapter 1

TMS320C55x DSP Peripherals Overview

Table 1−1 lists the peripherals of the TMS320C55x™ (C55x™) DSP generation and indicates how many copies of each peripheral are on the specific C55x devices. The column labeled “5509” applies to both TMS320VC5509 and TMS320VC5509A devices. On a given device, some peripherals may share pins, making the peripherals’ use mutually exclusive; see the device-specific data manual for details.

For a detailed description of a peripheral, see the chapter or document listed in the last column of the table. If a peripheral has its own reference guide, the table shows the literature number (SPRUxxx) for that reference guide. If you are viewing the table online, you can click the literature number to view or download a portable document format (PDF) file. Otherwise, you can find the PDF files on the Internet at www.ti.com. *Tip:* Enter the literature number in the search box provided at www.ti.com.

All information about general-purpose I/O pins and system control registers has been moved from this document to the device-specific data manuals.

Table 1−1. TMS320C55x DSP Peripherals

Peripheral	**5501**	**5502**	**5509**	**5510**	**For Details, See ...**
Analog-to-digital converter (ADC)			1		SPRU586
Clock generator with PLL			1	1	Chapter 3 and the device-specific data manual
	1	1			Device-specific data manual
Direct memory access (DMA) controller			1	1	SPRU587
	1	1			SPRU613
External memory interface (EMIF)				1	SPRU590
			1		SPRU670
	1	1			SPRU621

Table 1−1. TMS320C55x DSP Peripherals (Continued)

Peripheral	5501	5502	5509	5510	For Details, See ...
Host port interface (HPI)				1	SPRU588
			1		SPRU619
	1	1			SPRU620
Instruction cache				1	SPRU576
	1	1			SPRU630
Inter-integrated circuit (I2C) module	1	1	1		SPRU146
Multichannel buffered serial port (McBSP)	2	3	3	3	SPRU592
MultiMediaCard™ / SD card controller			2		SPRU593
Power management / Idle configurations	1	1	1	1	Chapter 8 and the device-specific data manual
Real-time clock (RTC)			1		SPRU594
Timer, general-purpose			2	2	SPRU595
	2	2			SPRU618
Timer, watchdog			1		SPRU595
	1	1			SPRU618
Universal Asynchronous Receiver/Transmitter (UART)	1	1			SPRU597
Universal Serial Bus (USB) module			1		Chapter 17

Chapter 2

Analog-to-Digital Converter (ADC)

The ADC is described in the *TMS320VC5509 DSP Analog-to-Digital Converter (ADC) Reference Guide* (SPRU586).

This page is intentionally left blank.

Chapter 3

Clock Generator

This chapter describes the clock generator that is in TMS320VC5509, TMS320VC5509A, and TMS320VC5510 DSPs. This clock generator accepts an input clock at the CLKIN pin and enables you to modify that signal internally to produce an output clock with the desired frequency. The clock generator passes this output clock (the CPU clock) to the CPU, to peripherals, and to other modules inside the C55x™ DSP. The CPU clock is also passed through a programmable clock divider to the CLKOUT pin. Check the device-specific data manual for additional clock-generation information.

Note:

For information about clock generation in TMS320VC5501 and TMS320VC5502 DSPs, see the device-specific data manuals.

Topic **Page**

3.1 Introduction to the DSP Clock Generator 3-2

3.2 Operational Flow of the DSP Clock Generator 3-3

3.3 Bypass Mode 3-5

3.4 Lock Mode 3-6

3.5 Idle (Low-Power) Mode 3-8

3.6 The CLKOUT Pin and the Associated Clock Divider 3-9

3.7 DSP Reset Conditions of the DSP Clock Generator 3-10

3.8 Clock Mode Register 3-12

3.1 Introduction to the DSP Clock Generator

The DSP clock generator supplies the DSP with a clock signal that is based on an input clock signal connected at the CLKIN pin. Included in the clock generator is a digital phase-lock loop (PLL), which can be enabled or disabled. You can configure the clock generator to create a CPU clock signal that has the desired frequency.

The clock generator has a clock mode register, CLKMD (see section 3.8 on page 3-12), for controlling and monitoring the activity of the clock generator. For example, you can write to the PLL ENABLE bit in CLKMD to toggle between the two main modes of operation:

- ❑ In the bypass mode (see section 3.3 on page 3-5), the PLL is bypassed, and the frequency of the output clock signal is equal to the frequency of the input clock signal divided by 1, 2, or 4. Because the PLL is disabled, this mode can be used to save power.
- ❑ In the lock mode (see section 3.4 on page 3-6), the input frequency 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. The lock mode is entered if the PLL ENABLE bit of the clock mode register is set and the phase-locking sequence is complete. (During the phase-locking sequence, the clock generator is kept in the bypass mode.)

The clock generator also has an idle mode (see section 3.5 on page 3-8) for power conservation. You place the clock generator into its idle mode by turning off the CLKGEN idle domain. For information on turning on and off idle domains, see Chapter 8, *Idle Configurations*.

The output of the clock generator or a divided down version of that output can be seen on the CLKOUT pin. For details, see section 3.6*, The CLKOUT Pin and the Associated Clock Divider*, on page 3-9.

3.2 Operational Flow of the DSP Clock Generator

Figure 3−1 and Table 3−1 describe the operational states (A−F) of the DSP clock generator. The clock mode register (CLKMD) is loaded by software or by a DSP reset. If the write to CLKMD enables the PLL, the PLL begins its phase-locking sequence (state A). If the write disables the PLL, the clock generator enters its bypass mode (state D).

Figure 3−1. Operational Flow of the DSP Clock Generator

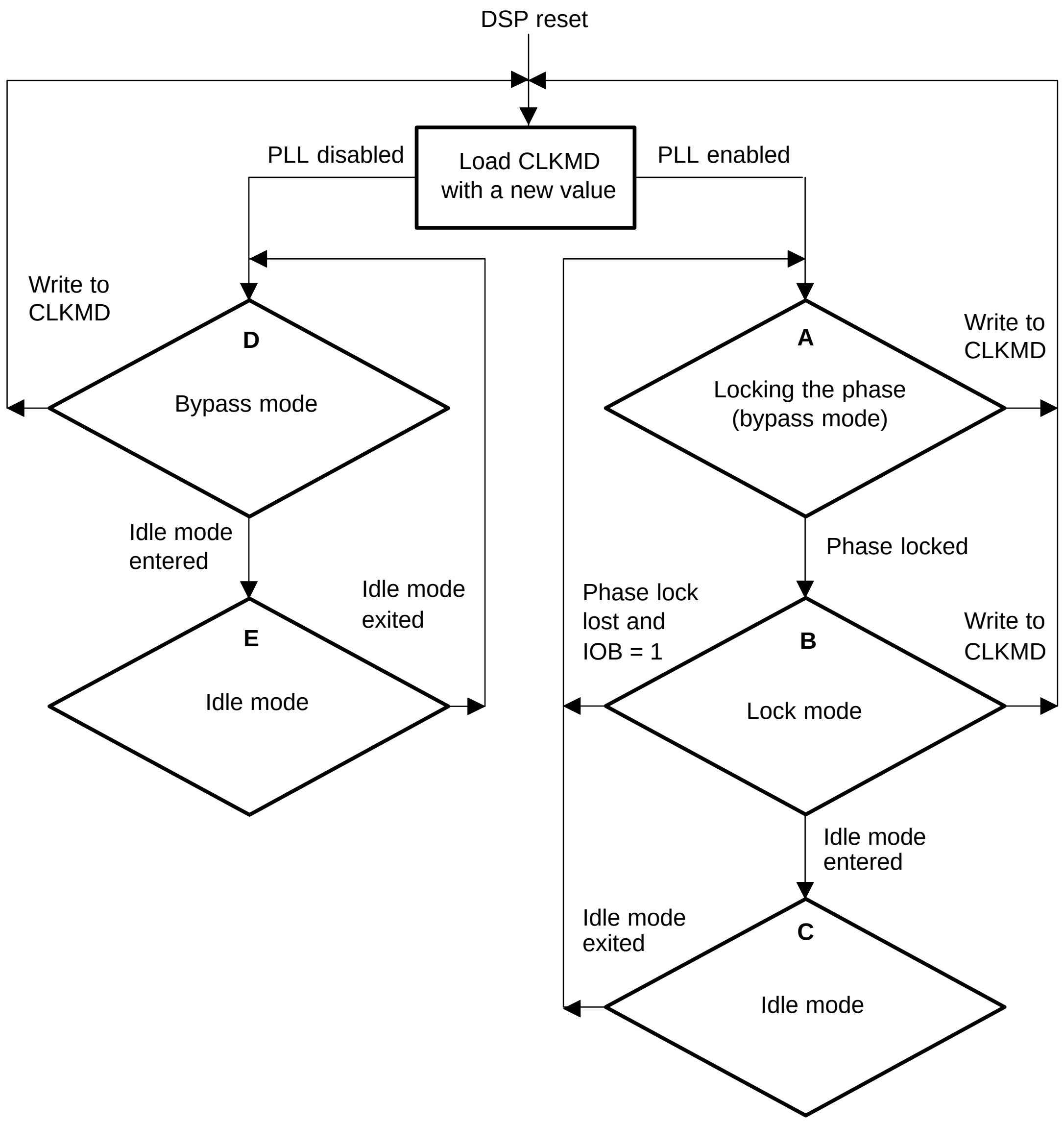


Table 3−1. Operational States Shown in Figure 3−1

State	Description
A	**Locking the phase.** The clock generator enters the bypass mode, and the PLL locks the phase of the output clock signal to that of the input clock signal. Once the phase is locked and the output signal is at the frequency defined by the PLL MULT bits and the PLL DIV bits of CLKMD, the clock generator enters its lock mode (state B). You can reconfigure the clock generator by writing to CLKMD.
B	**Lock mode.** In the lock mode, the PLL is generating an output signal with the selected frequency. The output signal is phase-locked to the input signal. If the PLL loses the lock and the IOB bit of CLKMD is 1, the clock generator returns to the bypass mode and reacquires the lock (state A); if the IOB bit is 0, the clock generator does not reacquire the lock. An idle instruction can place the clock generator into its idle mode (state C). To change to the bypass mode or to reconfigure the clock generator in other ways, you can write to CLKMD.
C	**Idle mode (entered from the lock mode).** An idle instruction has placed the clock generator into its idle mode. If the idle mode is properly exited, the clock generator starts again and reacquires the phase lock (state A). The method used to reacquire the lock depends on the IAI bit of CLKMD.
D	**Bypass mode.** The PLL is disabled, and the clock generator is in the bypass mode. The divider within the clock generator produces an output clock signal at the frequency defined by the BYPASS DIV bits of CLKMD. An idle instruction can place the clock generator into its idle mode (state E). To change to the lock mode or to reconfigure the clock generator in other ways, you can write to CLKMD.
E	**Idle mode (entered from the bypass mode).** An idle instruction has placed the clock generator into its idle mode. If the idle mode is properly exited, the clock generator starts again in the bypass mode.

3.3 Bypass Mode

When the DSP clock generator is in the bypass mode and the phase-lock loop (PLL) is disabled, the frequency of the output clock signal is equal to the frequency of the input clock signal divided by 1, 2, or 4.

3.3.1 Entering and Exiting the Bypass Mode

To enter the bypass mode, write a 0 to the PLL ENABLE bit in the clock mode register (CLKMD). The PLL will be disabled.

To exit the bypass mode, write a 1 to the PLL ENABLE bit. The PLL will start up and enter its phase-locking sequence. After the PLL is generating the configured output frequency and the phase of the output clock signal is locked to the phase of the input clock signal, the clock generator enters the lock mode. Until then, the clock generator stays in the bypass mode.

If the clock generator is in the lock mode and the PLL must reacquire its phase lock (IOB = 1), the clock generator enters the bypass mode until the phase is locked again.

3.3.2 CLKMD Bits Used in the Bypass Mode

Table 3−2 describes the bits of the clock mode register (CLKMD) that are used in the bypass mode. The reserved bits in CLKMD (Rsvd and TEST) should not be used in either the bypass mode or the lock mode. For a detailed description of CLKMD, see section 3.8 on page 3-12.

Table 3−2. CLKMD Bits Used in the Bypass Mode

CLKMD Bit Field	Role In The Bypass Mode
PLL ENABLE	Allows you to switch to the lock mode
BYPASS DIV	Determines how the input clock frequency is divided (if at all) to produce the output clock frequency
LOCK	Is 0 in the bypass mode

3.3.3 Setting the Output Frequency for the Bypass Mode

The output frequency is determined by the input frequency and the value in the BYPASS DIV bits. Load BYPASS DIV as required to divide the input frequency by 1, 2, or 4.

3.4 Lock Mode

In the lock mode, the input frequency 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.

3.4.1 Entering and Exiting the Lock Mode

To enter the lock mode, write a 1 to the PLL ENABLE bit in the clock mode register (CLKMD). The PLL will start up and will enter its phase-locking sequence. After the PLL is generating the configured output frequency and the phase of the output clock signal is locked to the phase of the input clock signal, the clock generator enters the lock mode. Until then, the clock generator stays in the bypass mode.

If the clock generator is in the lock mode and the PLL must reacquire its phase lock (IOB = 1 in CLKMD), the clock generator will enter the bypass mode until the phase is locked again.

To exit the lock mode (enter the bypass mode), write a 0 to the PLL ENABLE bit. The PLL will be disabled.

3.4.2 CLKMD Bits Used in the Lock Mode

Table 3−3 describes the bits of the clock mode register (CLKMD) that are used in the lock mode. The reserved bits (Rsvd and TEST) in CLKMD should not be used in either the lock mode or the bypass mode. For a detailed description of CLKMD, see section 3.8 on page 3-12.

Table 3−3. CLKMD Bits Used in the Lock Mode

CLKMD Bit Field(s)	Role In The Lock Mode
PLL ENABLE	Allows you to switch to the bypass mode (disable the PLL)
PLL MULT and PLL DIV	Determine how the input clock frequency is modified (if at all) to produce the output clock frequency
IAI	Determines whether the PLL returns to the beginning of the phase-locking sequence when the clock generator exits its idle mode
BREAKLN	Indicates when the phase lock has been broken
IOB	Determines whether the PLL will reacquire a lost phase lock
LOCK	Is 1 in the lock mode

3.4.3 Setting the Output Frequency for the Lock Mode

The input frequency is multiplied by the PLL MULT value of CLKMD and is divided according to the PLL DIV value of CLKMD. PLL MULT can be a value from 2 to 31. PLL DIV can be a value from 0 (divide by 1) to 3 (divide by 4). The output frequency can be calculated with the following equation:

$$\text{Output frequency} = \frac{\text{PLL MULT}}{(\text{PLL DIV} + 1)} \times \text{Input frequency}$$

Table 3−4 shows some examples of using PLL MULT and PLL DIV to select an output frequency.

Table 3−4. Examples of Selecting a Lock Mode Frequency

PLL MULT	**PLL DIV**	Output Frequency
31	0 (divide by1)	31 × Input frequency (maximum frequency)
10	1 (divide by 2)	5 × Input frequency
2	2 (divide by 3)	2/3 × Input frequency
2	3 (divide by 4)	1/2 × Input frequency (minimum frequency)

3.4.4 Calculating the Lock Time

The time needed for phase locking (the lock time) depends on the PLL MULT and PLL DIV values and on the output clock frequency. The lock time in number of **input** clock cycles is given by Equation 3−1.

Equation 3−1. Lock Time for the PLL

$$\text{Lock time} = 4 \times (\text{PLL DIV} + 1) \times \left[10D + 30(\text{PLL MULT} + 1) + 2\right]$$

where $D = 1 + \log_2\left(\frac{200}{\text{PLL MULT} \times \text{Output frequency}}\right)$ rounded to the nearest integer.

3.5 Idle (Low-Power) Mode

To save power, you can put the DSP clock generator into its idle mode by loading an idle configuration that turns off the CLKGEN idle domain. When the clock generator is idle, the output clock is stopped and held high. For more details, see Chapter 8, *Idle Configurations*.

When the clock generator exits its idle mode, the reaction of the clock generator depends on several factors. If the clock generator was in its bypass mode before the idle instruction was executed, the PLL returns to the bypass mode. If the clock generator was in its lock mode before the idle instruction was executed, the clock generator switches to its bypass mode, reacquires the phase lock, and then returns to the lock mode. The method used for reacquiring the phase lock depends on the IAI bit of CLKMD:

IAI	Method Used For Reacquiring The Phase Lock
0	The PLL reacquires the phase lock using the same process that was underway before the idle mode was entered.
1	The PLL restarts the phase-locking sequence.

3.6 The CLKOUT Pin and the Associated Clock Divider

The DSP clock generator generates the CPU clock that is supplied to the CPU, to peripherals, and to other modules inside the DSP. As shown in Figure 3−2, the CPU clock is also passed to a clock divider that supplies a signal (CLKOUT) to the CLKOUT pin. The frequency of CLKOUT depends on the CLKDIV bits of the system register, SYSR (see Table 3−5).

Figure 3−2. Dividing the CPU Clock for the CLKOUT Pin

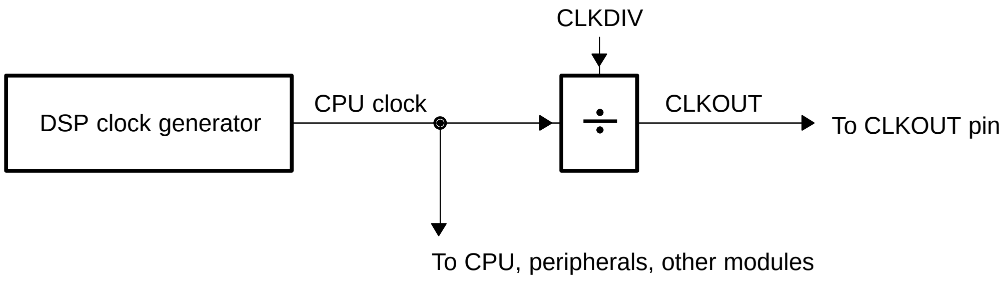


Table 3−5. Effect of CLKDIV Bits on CLKOUT Frequency

CLKDIV	Frequency of CLKOUT
000b	1/1 × CPU clock frequency
001b	1/2 × CPU clock frequency
010b	1/3 × CPU clock frequency
011b	1/4 × CPU clock frequency
100b	1/5 × CPU clock frequency
101b	1/6 × CPU clock frequency
110b	1/7 × CPU clock frequency
111b	1/8 × CPU clock frequency

3.7 DSP Reset Conditions of the DSP Clock Generator

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

3.7.1 Clock Generator During Reset

The DSP can make use of the output clock signal during reset. While the DSP reset signal is held low:

- ❑ The clock generator is in the bypass mode.
- ❑ The output clock frequency is determined by the level of the signal on the CLKMD input pin:

CLKMD Signal	Output Frequency
Low	Input frequency
High	1/2 $\times$ Input frequency

3.7.2 Clock Generator After Reset

On the rising edge of the DSP reset signal (when reset is deasserted), the clock mode register is loaded with a value determined by the level on the CLKMD pin:

CLKMD Signal	Output Frequency
Low	2002h
High	2006h

Table 3−6 summarizes the effects of this load to the clock mode register.

Table 3−6. Reset Values of CLKMD Bits and The Effects

Reset Value	Effect
IAI = 0	Only applicable in the lock mode. Initialize-after-idle is not selected. After the idle mode is exited, the PLL reacquires the phase lock using the same process that was underway before the idle mode was entered (the phase-locking sequence is not restarted).
IOB= 1	Only applicable in the lock mode. Initialize-on-break is selected. Any time the PLL loses its phase lock, the clock generator switches to its bypass mode and starts a new phase-locking sequence.
PLL MULT= 00000b PLL DIV= 00b	Only applicable in the lock mode. The output frequency is equal to the input frequency.
PLL ENABLE = 0	The PLL is disabled. The clock generator is in its bypass mode.
If CLKMD signal is low BYPASS DIV= 00b If CLKMD signal is high BYPASS DIV= 01b	If CLKMD signal is low Output frequency = Input frequency If CLKMD signal is high Output frequency = 1/2 $\times$ Input frequency
BREAKLN = 1	The break-lock indicator is reset.
LOCK = 0	The lock-mode indicator reflects the fact that the clock generator is in the bypass mode.

3.8 Clock Mode Register

You control the DSP clock generator with the clock mode register, CLKMD. Figure 3−3 and Table 3−7 describe the contents of CLKMD, which is accessible in I/O space. After the DSP reset signal becomes inactive, the CLKMD register is initialized with a predetermined value dependent only upon the state of the CLKMD input pin (the difference is in the BYPASS DIV bits):

CLKMD Signal Level at Reset	CLKMD Register Reset Value
Low	2002h
High	2006h

Figure 3−3. Clock Mode Register (CLKMD)

15	14	13	12	11−7
Rsvd	IAI	IOB	TEST (keep 0)	PLL MULT
	R/W − 0	R/W − 1		R/W − 00000

6−5	4	3−2	1	0
PLL DIV	PLL ENABLE	BYPASS DIV	BREAKLN	LOCK
R/W − 00	R/W − 0	R/W − pin	R − 1	R − 0

Legend:

R	Read-only access
R/W	Read/write access
− X	X is the value after a DSP reset. X = pin indicates that the reset value depends on the signal level on the CLKMD pin.

Table 3−7. Bit Field Descriptions for the Clock Mode Register (CLKMD)

Bit(s)	Name	Description	Reset Value
15	Rsvd	This bit is reserved; it is not available for your use. This bit is always 0.	−
14	IAI	Initialize after idle bit. IAI determines how the PLL reacquires the phase lock after the clock generator exits its idle mode (when the CLKGEN idle domain is reactivated): 0 The PLL locks using the same process that was underway before the idle mode was entered. 1 The PLL restarts the phase-locking sequence.	0

Table 3−7. Bit Field Descriptions for the Clock Mode Register (CLKMD) (Continued)

Bit(s)	Name	Description	Reset Value
13	IOB	Initialize on break bit. IOB determines whether the clock generator initializes the PLL phase-locking sequence whenever the phase lock is broken. If the PLL indicates a break in the phase lock: 0 The clock generator does not interrupt the PLL. The clock generator stays in the lock mode, and the PLL continues to output the current clock signal. 1 The clock generator switches to its bypass mode and restarts the PLL phase-locking sequence.	1
12	TEST	This reserved test bit is cleared during a DSP reset and your program must be keep it 0 for proper operation of the clock generator. Make sure that whenever your program modifies CLKMD, it writes a 0 to bit 12.	0
11−7	PLL MULT	PLL multiply value. When the PLL is enabled (PLL ENABLE = 1), the frequency of the input clock signal is multiplied according to the value in PLL MULT. PLL MULT can be a value from 2 to 31. The input clock is multiplied by the unsigned integer in PLL MULT and is divided according to the value in the PLL DIV bits. The maximum frequency for the PLL output clock signal is 31 times the frequency of the input clock signal. To obtain this maximum frequency, load PLL MULT with 31 (multiply by 31), and load PLL DIV with 0 (divide by 1). The minimum frequency for the output clock signal is 1/2 the frequency of the input clock signal. To obtain this minimum frequency, load PLL MULT with 2 (multiply by 2) and load PLL DIV with 3 (divide by 4).	00000b

Table 3−7. Bit Field Descriptions for the Clock Mode Register (CLKMD) (Continued)

Bit(s)	Name	Description	Reset Value
6−5	PLL DIV	PLL divide value. When the PLL is enabled (PLL ENABLE = 1), the two PLL DIV bits select one of four divide options listed in the following table. The PLL also uses the multiply value supplied by the PLL MULT bits. To program the minimum or maximum frequency, see the description for the PLL MULT bits. 00b No division/divide by 1 The input frequency is not divided. 01b Divide by 2 The input frequency is divided by 2. 10b Divide by 3 The input frequency is divided by 3. 11b Divide by 4 The input frequency is divided by 4.	00b
4	PLL ENABLE	PLL enable bit. Write to PLL ENABLE to enable or disable the PLL. When you set PLL ENABLE, you request the clock generator to enter the lock mode. The clock generator does not enter the lock mode until the PLL is creating a phase-locked signal with the frequency selected by the PLL MULT bits and the PLL DIV bits. 0 Disable the PLL (enter the bypass mode). 1 Enable the PLL and, when the correct output clock signal is generated, enter the lock mode.	0
3−2	BYPASS DIV	Bypass-mode divide value. In the bypass mode, BYPASS DIV determines the frequency of the output clock signal. During a DSP reset, if the level on the CLKMD pin is low, BYPASS DIV is reset to 00b (no division). If the level on CLKMD is high, BYPASS DIV is reset to 01b (divide by 2). 00b No division/divide by 1 The frequency of the output clock signal is the same as the frequency of the input clock signal. 01b Divide by 2 The frequency of the output clock signal is 1/2 the frequency of the input clock signal. 10b or 11b Divide by 4 The frequency of the output clock signal is 1/4 the frequency of the input clock signal.	00b if CLKMD signal is low 01b if CLKMD signal is high

Table 3−7. Bit Field Descriptions for the Clock Mode Register (CLKMD) (Continued)

Bit(s)	Name	Description	Reset Value
1	BREAKLN	Break-lock indicator. BREAKLN indicates whether the PLL has broken the phase lock. In addition, if you write to CLKMD, BREAKLN is forced to 1. 0 The PLL has broken the phase lock. 1 The phase lock is restored, or a write to CLKMD has occurred.	1
0	LOCK	Lock-mode indicator. LOCK indicates whether the clock generator is in its lock mode: 0 The clock generator is in the bypass mode. The output clock signal has the frequency determined by the BYPASS DIV bits, or the PLL is in the process of getting a phase lock. 1 The clock generator is in the lock mode. The PLL has a phase lock, and the output clock has the frequency determined by the PLL MULT bits and the PLL DIV bits.	0

This page is intentionally left blank.

Chapter 4

Direct Memory Access (DMA) Controller

The DMA controller that is in TMS320VC5501 and TMS320VC5502 DSPs is described in the *TMS320VC5501/5502 DSP Direct Memory Access (DMA) Controller Reference Guide* (SPRU613). The DMA controller that is in TMS320VC5509, TMS320VC5509A, and TMS320VC5510 DSPs is described in the *TMS320VC5509/5510 DSP Direct Memory Access (DMA) Controller Reference Guide* (SPRU587).

This page is intentionally left blank.

Chapter 5

External Memory Interface (EMIF)

For the EMIF information that applies to your C55x device, see the corresponding device-specific reference guide:

- ❑ *TMS320VC5501/5502 DSP External Memory Interface (EMIF) Reference Guide* (SPRU621),
- ❑ *TMS320VC5509 DSP External Memory Interface (EMIF) Reference Guide* (SPRU670), or
- ❑ *TMS320VC5510 DSP External Memory Interface (EMIF) Reference Guide* (SPRU590).

This page is intentionally left blank.

Chapter 6

General-Purpose I/O Port

Information about the general-purpose I/O port has been moved to the device-specific data manuals.

This page is intentionally left blank.

Chapter 7

Host Port Interface (HPI)

For the HPI information that applies to your C55x device, see the corresponding device-specific reference guide:

- *TMS320VC5501/5502 DSP Host Port Interface (HPI) Reference Guide* (SPRU620),
- *TMS320VC5509 DSP Host Port Interface (HPI) Reference Guide* (SPRU619), or
- *TMS320VC5510 DSP Host Port Interface (HPI) Reference Guide* (SPRU588).

The name “enhanced host port interface (EHPI)” has been reduced to “host port interface (HPI)” to align with a new peripheral naming convention.

This page is intentionally left blank.

Chapter 8

Idle Configurations

The TMS320C55x DSP is divided into the idle domains described in this chapter. To minimize power consumption, you can choose which domains are active and which domains are idle at any given time. The current state of all domains is collectively called the *idle configuration*.

Note:

Check the device-specific data manual for additional information about deactivating and reactivating idle domains of the DSP.

Topic **Page**

8.1 Idle Domains 8-2

8.2 Idle Configuration Process 8-4

8.3 Valid Idle Configurations 8-5

8.4 To Change Idle Configurations (Key Conditions) 8-6

8.5 Interrupt Handling When the CPU is Reactivated 8-8

8.6 Effect of a DSP Reset on the Idle Domains 8-8

8.7 Idle Registers 8-9

8.1 Idle Domains

The DSP is divided into the idle domains described in Table 8−1. You can control which of these idle domains are active and which are idle at any given time, as described in section 8.2.

Notes:

1) The peripheral bus controller and the host port interface (HPI) on the DSP are not part of any idle domain. The only way to turn these modules off is to put the clock generator into its idle mode (make the CLKGEN domain idle).

2) The internal memory blocks (SARAM and DARAM) and the external memory are shared by two domains (CPU and DMA). When both domains are idle, memory accesses are disabled.

Table 8−1. Idle Domains in the DSP

Domain	Contents of the Domain	Configurability
CPU	CPU and CPU buses	When the idle instruction is executed, the CPU remains active or becomes idle, depending on the chosen idle configuration. Regardless of this domain’s state before a DSP reset, it is active after a DSP reset.
DMA	DMA controller and DMA buses	When the idle instruction is executed, the DMA (direct memory access) controller remains active or becomes idle, depending on the chosen idle configuration. Regardless of this domain’s state before a DSP reset, it is active after a DSP reset.
CACHE	Instruction cache	When the idle instruction is executed, the instruction cache remains active or becomes idle, depending on the chosen idle configuration. Regardless of this domain’s state before a DSP reset, it is active after a DSP reset.
PERIPH	Timers, serial ports, and other peripherals	Each of the peripherals in this domain has an idle enable bit that determines whether the peripheral can be placed in its idle mode when the idle instruction is executed. If the PERIPH domain is configured to be idle and an idle enable bit is 1, the corresponding peripheral is placed in its idle mode. Regardless of this domain’s state before a DSP reset, it is active after a DSP reset.

Table 8−1. Idle Domains in the DSP (Continued)

Domain	Contents of the Domain	Configurability
CLKGEN	Clock generator, including the phase-lock loop (PLL) circuitry	When the idle instruction is executed, the clock generator remains active or becomes idle, depending on the chosen idle configuration. When the clock generator is in its idle mode, no clocking is available for the CPU or the DMA controller. If the clock generator is configured to be idle and the CPU, the DMA controller, or the cache is configured to be active, a bus error interrupt request is sent to the CPU. If properly enabled, the interrupt will force the CLKGEN domain to be reactivated. Peripherals that do not depend on the DSP clock signal are not affected by the state of the CLKGEN domain. Regardless of this domain's state before a DSP reset, it is active after a DSP reset.
EMIF	External memory interface (EMIF)	When the idle instruction is executed, the EMIF is disabled or enabled, depending on the chosen idle configuration. Regardless of this domain's state before a DSP reset, it is active after a DSP reset.

8.2 Idle Configuration Process

The idle configuration indicates which idle domains will be idle, and which idle domains will be active, the next time the idle instruction is executed. The basic steps to the idle configuration process are:

1) Define a new idle configuration by writing to the bits in the idle configuration register (ICR). Make sure that you use a valid idle configuration (see section 8.3).

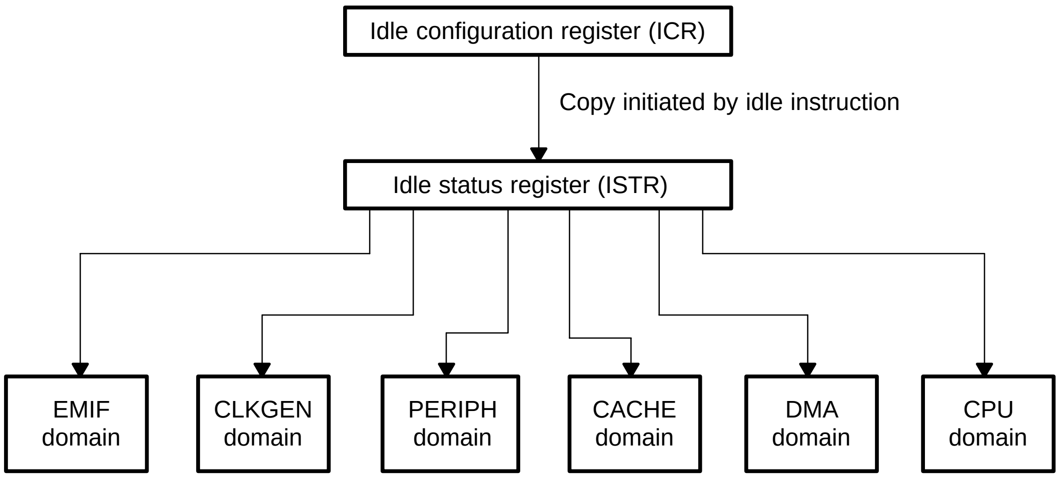
2) Apply the new idle configuration by executing the idle instruction. The effects are shown in the following figure. The content of ICR is copied to the idle status register (ISTR). The bits of ISTR are then propagated through the system to enable or disable each of the chosen domains.

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

Notes:

If you intend to switch among multiple idle configurations, make sure that your system has the means to change from one idle configuration to the next. For important considerations, see section 8.4.

Figure 8−1. Idle Configuration Process


Idle configuration register (ICR)
Copy initiated by idle instruction
Idle status register (ISTR)
EMIF domain
CLKGEN domain
PERIPH domain
CACHE domain
DMA domain
CPU domain

8.3 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, when the CLKGEN domain is idle (the DSP clock generator is disabled), the DMA controller, the CPU, and any peripherals that do not have their own external clocks cannot operate.

8.4 To Change Idle Configurations (Key Conditions)

Before you use the idle instruction, make sure that there is a method for the DSP to change the idle configuration afterward. Table 8−2 summarizes the methods available under three key conditions. The table also describes the effects on the idle registers: the idle status register (ISTR) and the idle configuration register (ICR). For more details about these idle registers, see section 8.7 on page 8-9.

Table 8−2. Changing Idle Configurations

Condition	Available Methods For Changing Idle Configuration	ISTR After Change	ICR After Change
1. CLKGEN domain is active CPU domain is active	A. Write a new configuration to the idle configuration register (ICR), and then execute the idle instruction.	A. Modified by the idle instruction; contains a copy of the new ICR value	A. Contains the new value that was loaded by the program
(see section 8.4.1)	B. Initiate a DSP hardware reset.	B. Cleared (all 0s)	B. Cleared (all 0s)
2. CLKGEN domain is active CPU domain is idle	A. Use an unmasked hardware interrupt or the nonmaskable hardware interrupt called NMI_.	A. CLKGENIS and CPUIS bits are 0. No other bits were modified.	A. Not modified
(see section 8.4.2)	B. Initiate a DSP hardware reset.	B. Cleared (all 0s)	B. Cleared (all 0s)
3. CLKGEN domain is idle (see section 8.4.3)	A. Use an unmasked hardware interrupt or the nonmaskable hardware interrupt called NMI_.	A. CLKGENIS and CPUIS bits are 0. No other bits were modified.	A. Not modified
	B. Initiate a DSP hardware reset.	B. Cleared (all 0s)	B. Cleared (all 0s)

8.4.1 Condition 1: CLKGEN and CPU Domains Active

When the CLKGEN domain is active (the DSP clock generator is enabled) and the CPU domain is active (the DSP CPU is running), program flow continues. In this case, there are two methods of changing idle configurations:

❑ Write a new idle configuration to the idle configuration register (ICR), and then execute the idle instruction. The idle instruction copies the content of the ICR to the idle status register (ISTR), and the ISTR bit values are propagated to the idle domains. After the domains change states, the value in ISTR matches the value in ICR.

❑ Initiate a DSP hardware reset at the DSP reset pin. When the DSP resets, all domains are made active.

8.4.2 Condition 2: CLKGEN Domain Active, CPU Domain Idle

When the CPU domain is idle, program flow is halted. It is not possible to write a new value to the idle configuration register (ICR) or to execute the idle instruction. Two other methods are available for changing the idle configuration:

❑ Use an unmasked interrupt or the nonmaskable interrupt called NMI_. The interrupt clears the CLKGENIS and CPUIS bits of the idle status register (ISTR). The change to CPUIS reactivates the CPU domain, and the change to CLKGENIS ensures that the CLKGEN domain is also active. The content of the idle configuration register (ICR) is not modified. To learn how the CPU responds to the interrupt, see section 8.5.

❑ Initiate a DSP hardware reset at the DSP reset pin. When the DSP resets, all domains are made active.

Once program flow has begun again, you can reactivate or deactivate other domains by writing a new idle configuration to ICR and then executing the idle instruction.

8.4.3 Condition 3: CLKGEN Domain Idle

When the CLKGEN domain is idle (the DSP clock generator is disabled), no internal clocks are active, including the CPU clock. With the CPU halted, it is not possible to write a new value to the idle configuration register (ICR) or to execute the idle instruction. Two other methods are available for waking the DSP:

❑ Use an unmasked interrupt or the nonmaskable interrupt called NMI_. The interrupt clears the CLKGENIS and CPUIS bits of the idle status register (ISTR). The change to CPUIS reactivates the CPU domain, and the change to CLKGENIS ensures that the CLKGEN domain is also active. The content of the idle configuration register (ICR) is not modified. To learn how the CPU responds to the interrupt, see section 8.5.

❑ Initiate a DSP hardware reset at the DSP reset pin. When the DSP resets, all domains are made active.

Once program flow has begun again, you can reactivate or deactivate other domains by writing a new idle configuration to ICR and then executing the idle instruction.

8.5 Interrupt Handling When the CPU Is Reactivated

If the CPU has been halted by an idle configuration, it can be reactivated by a nonmaskable interrupt (NMI_ or RESET_) or by a maskable interrupt that is enabled in an interrupt enable register (IER0 or IER1). A maskable interrupt request will also set the corresponding interrupt flag bit in an interrupt flag register (IFR0 or IFR1). Table 8−3 summarizes how the CPU responds after being reactivated by maskable and nonmaskable interrupts. INTM is the global interrupt mask bit in status register ST1_55.

Table 8−3. CPU Response After Reactivation

Interrupt	**CPU Response After Reactivation ...**
A maskable interrupt	If INTM = 0: The CPU executes the interrupt service routine, executes the instruction that follows the idle instruction, and continues from there. If INTM = 1: The CPU executes the instruction that follows the idle instruction and then continues from there. The interrupt service routine cannot be executed until the interrupt has been enabled by INTM.
NMI_ (nonmaskable)	The CPU executes the interrupt service routine, executes the instruction that follows the idle instruction, and continues from there.
RESET_ (nonmaskable)	The DSP is reset. (During a DSP reset, all idle domains are made active.)

8.6 Effect of a DSP Reset on the Idle Domains

Driving the DSP reset signal low starts a DSP reset. During a DSP reset, all idle domains are made active.

8.7 Idle Registers

Two registers provide the means for you to individually configure and monitor each of the idle domains: the idle configuration register (ICR) and the idle status register (ISTR). These registers (see Figure 8−2) are part of the peripheral bus controller and are accessible to the CPU. Table 8−4 describes the read/write bits of ICR, and Table 8−5 describes the read-only bits of ISTR.

ICR lets you configure how each idle 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 idle domains. Making the clock generator idle (CLKGENI = 1) provides the lowest level of power consumption in the DSP by stopping all the system clocks.

Figure 8−2. Idle Configuration Register (ICR) and Idle Status Register (ISTR)

	15−6	5	4	3	2	1	0
ICR	Reserved	EMIFI	CLKGENI	PERI	CACHEI	DMAI	CPUI
		R/W − 0	R/W − 0	R/W − 0	R/W − 0	R/W − 0	R/W − 0

	15−6	5	4	3	2	1	0
ISTR	Reserved	EMIFIS	CLKGENIS	PERIS	CACHEIS	DMAIS	CPUIS
		R − 0	R − 0	R − 0	R − 0	R − 0	R − 0

Legend:

R	Read access
W	Write access
− 0	This bit is cleared by a DSP reset.
Reserved	A write to this bit field has no effect, and the bit or bits in this field always appear as 0s during read operations.

Table 8−4. ICR Bit Descriptions

Bit(s)	Name	Description	Reset Value
15−6	Reserved	These bits are not available for your use. They are read-only bits and return 0s when read.	−
5	EMIFI	EMIF-domain idle configuration bit. EMIFI determines whether the external memory interface (EMIF) will be idle after the next execution of the idle instruction: 0 EMIF will be active 1 EMIF will be idle	0

Table 8−4. ICR Bit Descriptions (Continued)

Bit(s)	Name	Description	Reset Value
4	CLKGENI	CLKGEN-domain idle configuration bit. CLKGENI determines whether the DSP clock generator will be idle after the next execution of the idle instruction: 0 Clock generator will be active 1 Clock generator will be idle **Important:** For a proper power-down, when you set CLKGEN = 1, make sure you also set CPUI = 1, DMAI = 1, and CACHEI = 1. If you do not, a bus error interrupt (BERRINT) request is sent to the CPU.	0
3	PERI	PERIPH-domain idle configuration bit. Peripherals that are in the PERIPH domain each have an idle enable bit. PERI, in conjunction with the idle enable bits, determines which of the peripherals in the domain will be idle after the next execution of the idle instruction: 0 All peripherals in the domain will be active. 1 For each peripheral in the domain: If the idle enable bit is 1, the peripheral will be in its idle mode. If the idle enable bit is 0, the peripheral will be active.	0
2	CACHEI	CACHE-domain idle configuration bit. CACHEI determines whether the cache will be idle after the next execution of the idle instruction: 0 Cache will be active 1 Cache will be idle	0
1	DMAI	DMA-domain idle configuration bit. DMAI determines whether the DMA controller will be idle after the next execution of the idle instruction: 0 DMA controller will be active 1 DMA controller will be idle	0
0	CPUI	CPU-domain idle configuration bit. CPUI determines whether the DSP CPU will be idle after the next execution of the idle instruction: 0 CPU will be active 1 CPU will be idle	0

Table 8−5. ISTR Bit Descriptions

Bit(s)	Name	Description	Reset Value
15−6	Reserved	These bits are not available for your use. They are read-only bits and return 0s when read.	−
5	EMIFIS	EMIF-domain idle status bit. EMIFIS is a copy of EMIFI made during the execution of the idle instruction. EMIFIS reflects the current idle status of the external memory interface (EMIF): 0 EMIF is active 1 EMIF is idle	0
4	CLKGENIS	CLKGEN-domain idle status bit. CLKGENIS is a copy of CLKGENI made during the execution of the idle instruction. CLKGENIS reflects the current idle status of the DSP clock generator: 0 Clock generator is active 1 Clock generator is idle	0
3	PERIS	PERIPH-domain idle status bit. PERIS is a copy of PERI made during the execution of the idle instruction. PERIS reflects the current idle status of the peripherals in the PERIPH domain: 0 All peripherals in the domain are active 1 For each peripheral in the domain: If the idle enable bit is 1, the peripheral is idle. If the idle enable bit is 0, the peripheral is active.	0
2	CACHEIS	CACHE-domain idle status bit. CACHEIS is a copy of CACHEI made during the execution of the idle instruction. CACHEIS reflects the current idle status of the cache: 0 Cache is active 1 Cache is idle	0
1	DMAIS	DMA-domain idle status bit. DMAIS is a copy of DMAI made during the execution of the idle instruction. DMAIS reflects the current idle status of the DMA controller: 0 DMA controller is active 1 DMA controller is idle	0

Table 8−5. ISTR Bit Descriptions (Continued)

Bit(s)	Name	Description	Reset Value
0	CPUIS	CPU-domain idle status bit. CPUIS is a copy of CPUI made during the execution of the idle instruction. CPUIS reflects the current idle status of the DSP CPU: 0 CPU is active 1 CPU is idle	0

Chapter 9

Instruction Cache

The instruction cache that is in TMS320VC5501 and TMS320VC5502 DSPs is described in the *TMS320VC5501/5502 DSP Instruction Cache Reference Guide* (SPRU630). The instruction cache that is in TMS320VC5510 DSPs is described in the *TMS320VC5510 DSP Instruction Cache Reference Guide* (SPRU576).

This page is intentionally left blank.

Chapter 10

Inter-Integrated Circuit (I2C) Module

Information about the I2C module is in the *TMS320VC5501/5502/5509 DSP Inter-Integrated Circuit (I2C) Module Reference Guide* (SPRU146).

This page is intentionally left blank.

Chapter 11

Multichannel Buffered Serial Port (McBSP)

McBSP information is in the *TMS320VC5501/5502/5509/5510 DSP Multichannel Buffered Serial Port (McBSP) Reference Guide* (SPRU592).

This page is intentionally left blank.

Chapter 12

MultiMediaCard / SD Card Controller

TMS320VC5509 and TMS320VC5509A devices the TMS320C55x contain a peripheral for controlling memory cards that conform to the MultiMediaCard™ standard or the SD (Secure Digital) Memory Card standard. Information about the MultiMediaCard / SD card controller (MMC controller) is in the *TMS320VC5509 DSP MultiMediaCard / SD Card Controller Reference Guide* (SPRU593).

This page is intentionally left blank.

Chapter 13

Real-Time Clock (RTC)

Information about the RTC is in the *TMS320VC5509 DSP Real-Time Clock (RTC) Reference Guide* (SPRU594).

This page is intentionally left blank.

Chapter 14

System Control Registers

Information about the system control registers has been moved to the device-specific data manuals.

This page is intentionally left blank.

Chapter 15

Timer (General-Purpose)

The general-purpose timer that is in TMS320VC5501 and TMS320VC5502 DSPs is described in the *TMS320VC5501/5502 DSP Timers Reference Guide* (SPRU618). The general-purpose timer that is in TMS320VC5509, TMS320VC5509A, and TMS320VC5510 DSPs is described in the *TMS320VC5509/5510 DSP Timers Reference Guide* (SPRU595).

This page is intentionally left blank.

Chapter 16

Universal Asynchronous Receiver/Transmitter (UART)

UART information is in the *TMS320VC5501/5502 DSP Universal Asynchronous Receiver/Transmitter (UART) Reference Guide* (SPRU597).

This page is intentionally left blank.

Chapter 17

Universal Serial Bus (USB) Module

This chapter describes the USB module that is in TMS320VC5509 and TMS320VC5509A DSPs. With its USB module, each of these DSPs can be used to create a full speed USB slave device that is compliant with Universal Serial Bus Specification Version 1.1.

Topic **Page**

17.1 USB Concepts Overview 17-2

17.2 Introduction to the USB Module 17-5

17.3 USB Buffer Manager (UBM) 17-11

17.4 USB DMA Controller 17-13

17.5 Host-DMA Mode: Host-Initiated Direct Memory Accesses 17-40

17.6 Interrupt Activity in the USB Module 17-46

17.7 Power, Emulation, and Reset Considerations 17-53

17.8 USB Module Registers 17-55

17.1 USB Concepts Overview

This section explains USB concepts and terminology used in this chapter.

17.1.1 Terminology

In a USB system, the host is the master. The host initiates all data transfers between itself and attached USB devices. Therefore, the direction of a data transfer is described relative to the host:

OUT transfer	A transfer of data from the host to a device: Host $\rightarrow$ Device
IN transfer	A transfer of data from a device to the host: Host $\leftarrow$ Device

Each IN or OUT transfer can be one of the following types. The types of transfers on a USB are:

Control transfer	A data transfer that is used by the USB host to send commands to a USB device, including commands to enumerate the device when it is first attached. Control transfers include error checking.
Bulk transfer	A data transfer that is used by the host for large amounts of data that are not time-critical. Can use when transfer time is not critical. The host only allocates bus time for bulk transfers when the time is not need by transfers of the other types. Bulk transfers include error checking. A device such as a printer is a good application for this type of transfer.
Interrupt transfer	The data transfer used when a USB device must send or receive moderate amounts of data periodically with minimum latency. Interrupt transfers include error checking. Typical devices that use this type of transfer are keyboards and joysticks.
Isochronous transfer	A data transfer available to support USB devices that need to send or receive data in real time at a constant rate. Isochronous transfers can handle more data than interrupt transfers, but no error checking is performed. A device such as a digital speaker is a typical application for isochronous transfers.

Note:

From an implementation standpoint, bulk and interrupt transfers are treated the same way in the C55x USB module. The only difference is that, in the case of an interrupt transfer, the transfer is initiated periodically by the host, whereas a bulk transfer is initiated by the host whenever the bus is not used for other transfers.

For data transfer between a USB host and a USB device, the data passes through an endpoint in the device:

Endpoint	A designated storage location within a USB device. Each endpoint in a device is uniquely identified by its number and its direction (IN or OUT).
OUT Endpoint	An endpoint that holds data received from the USB host. To use data from the host, the USB device must read the data from an OUT endpoint. Each device must have an OUT endpoint 0 to be used for control transfers.
IN Endpoint	An endpoint that holds data to be sent to the USB host. To send data to the host, the USB device must write to an IN endpoint. Each device must have an IN endpoint 0 to be used for control transfers.

The overall characteristics of the USB device and the type of each endpoint must be reported to the host when the device is attached to the bus for the first time. This process is called enumeration.

The USB bandwidth is shared by multiple USB devices. Data is transferred on the bus at regular (1-millisecond) intervals. Each of these intervals is called a **frame**, and the host divides up the frame for all the devices on the bus. As each new USB device is recognized and successfully configured by the host, it gains a portion of the frame. The size of the portion depends on factors such as the type of transfer (for example, isochronous versus bulk) and the amount of bandwidth that is not being used by other devices that are already on the bus.

17.1.2 Data Toggle Mechanism

For non-isochronous transfers, the USB uses a data toggle mechanism to detect transmission errors, to ensure that the transmitter and the receiver of USB data are synchronizing throughout a transfer. The data toggle mechanism requires two data packet types (DATA0 and DATA1) and two toggle bits (one in the transmitter and one in the receiver). Each packet transmitted is a DATA0 packet or a DATA1 packet, depending on the value of the transmitter’s toggle bit (0 = DATA0; 1 = DATA1). If the receiver is synchronized, its toggle bit matches that of the transmitter, and the receiver expects the data type that was transmitted. Once the packet is successfully received, the receiver complements its toggle bit and sends an acknowledgement to the transmitter. When the acknowledgement arrives at the transmitter, the transmitter complements its toggle bit.

A USB transfer may be comprised of a number of transactions, but the first packet of the first transaction is a DATA0 packet. Subsequent packets alternate in type (DATA1, DATA0, DATA1, and so on).

17.2 Introduction to the USB Module

The USB module is a USB 1.1-compliant, full speed slave device.

The C55x USB module has 16 endpoints:

❑ Two control endpoints (for control transfers only): OUT endpoint 0 and IN endpoint 0.

❑ Fourteen general-purpose endpoints (for other types of transfers): OUT endpoints 1−7 and IN endpoints 1−7. Each of these endpoints has:

  ■ Support for bulk, interrupt, and isochronous transfers.

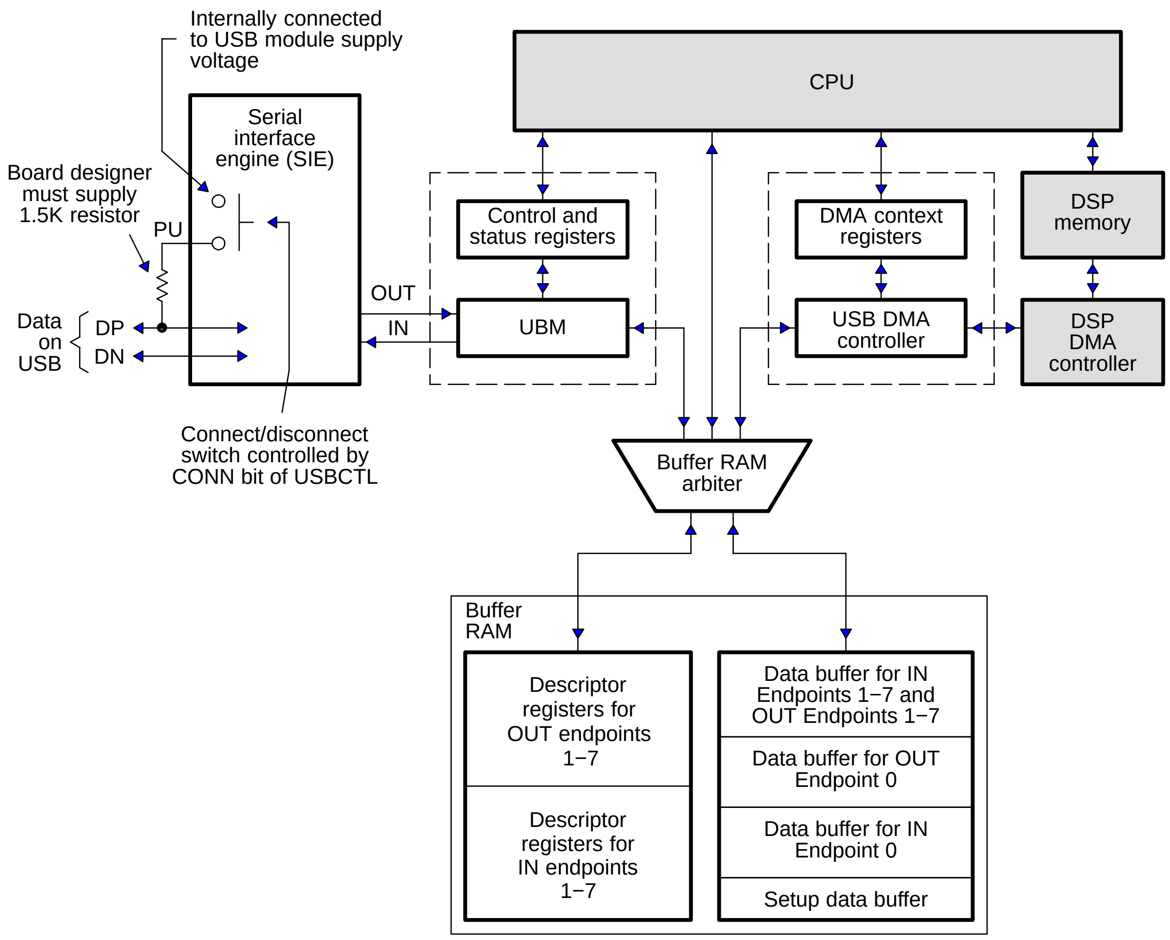
  ■ An optional double-buffer scheme for fast data throughput.

  ■ A dedicated DMA channel. A DMA controller inside the USB module can pass data between the general-purpose endpoints and the DSP memory while the CPU performs other tasks. (This DMA controller does not access the control endpoints.)

17.2.1 Block Diagram of the USB Module

Figure 17−1 contains a conceptual block diagram of the USB module. The shaded blocks in the figure are outside the USB module. Following the figure is a list that describes each of the main components of the module.

Figure 17−1. Conceptual Block Diagram of the USB Module


Internally connected to USB module supply voltage
CPU
Serial interface engine (SIE)
Board designer must supply 1.5K resistor
PU
Data on USB
DP
DN
OUT
IN
Control and status registers
UBM
DMA context registers
USB DMA controller
DSP memory
DSP DMA controller
Connect/disconnect switch controlled by CONN bit of USBCTL
Buffer RAM arbiter
Buffer RAM
Descriptor registers for OUT endpoints 1−7
Descriptor registers for IN endpoints 1−7
Data buffer for IN Endpoints 1−7 and OUT Endpoints 1−7
Data buffer for OUT Endpoint 0
Data buffer for IN Endpoint 0
Setup data buffer

- **Interface pins:**

Pin	Description
DP	Connect this pin to the line of the USB connector that carries the positive differential data.
DN	Connect this pin to the line of the USB connector that carries the negative differential data.
PU	Use this pin to connect a 1.5 Kohm pullup resistor to the DP line. A software-controlled switch can connect the pullup resistor to an internal 3.6-V source.
	When the CPU sets the connect bit of USBCTL (CONN = 1), the switch closes and completes the pullup circuit, causing the USB host to detect the USB module as a new device on the bus, and to start the enumeration process.
	To disconnect the device from the USB system, clear the CONN bit. The switch will open and disconnect the pullup resistor.

- **Serial interface engine (SIE).** The SIE is the USB protocol handler. It parses the USB bit stream for data packets that are meant for the USB device. For an OUT transfer, the SIE converts the serial data to parallel data and passes them to the USB buffer manager. For an IN transfer, the SIE converts parallel data from the UBM to serial data, and transmits them over the USB.

  The SIE also handles error-checking tasks. For an OUT transfer, the SIE does the error checking and transfers only the good data to the UBM. For an IN transfer, the SIE generates the necessary error-checking information before sending the data on the bus.

- **USB buffer manager (UBM) and the control and status registers.** The UBM controls data flow between the SIE and the buffer RAM. Most of the control registers are used to control the behavior of the UBM, and most of the status registers are modified by the UBM, to notify the CPU when any events occur.

❑ **Buffer RAM.** The buffer RAM contains registers that are mapped in the DSP I/O space. In the RAM are:

- Relocatable buffer space for each of the general-purpose endpoints (3.5K bytes). A general-purpose endpoint can have one data buffer (X buffer) or two data buffers (X buffer and Y buffer).
- A fixed data buffer for OUT endpoint 0 (64 bytes)
- A fixed data buffer for IN endpoint 0 (64 bytes)
- A fixed data buffer for a setup packet (8 bytes)
- Descriptor registers. For each of the general-purpose endpoints, there are eight registers that determine the endpoint characteristics.

❑ **USB DMA controller and its context registers.** This DMA controller can transfer data between the DSP memory and the X and Y buffers of the general-purpose endpoints. Each of these endpoints has a dedicated DMA channel and a dedicated set of DMA context registers for controlling and monitoring activity in that channel. The CPU can read from or write to each of these context registers by accessing the appropriate 16-bit address in I/O space.

The USB DMA controller accesses memory via the auxiliary port of the DSP DMA controller (see Chapter 4). This auxiliary port is shared by the USB DMA controller and the host port interface (HPI), but the USB DMA controller is given the higher priority.

A state machine in the USB DMA controller handles data transfers in the host-DMA mode (see section 17.5 on page 17-40).

❑ **Buffer RAM arbiter.** The 8-bit-wide buffer RAM can be accessed by the UBM, by the USB DMA controller, and by the DSP CPU. The buffer RAM arbiter provides a fair access scheme to arbitrate accesses from these three requesters.

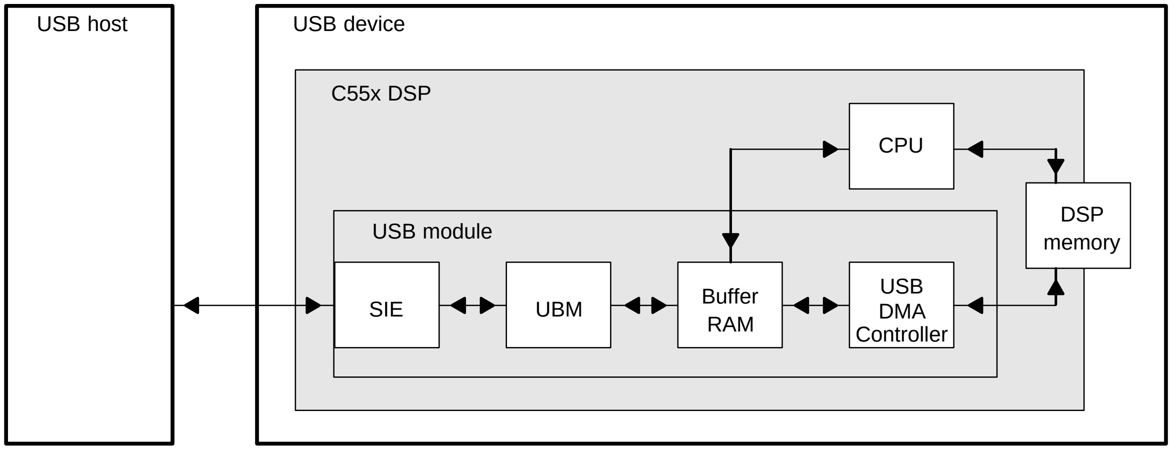
The USB DMA controller only accesses the X and Y buffers of the general-purpose endpoints. The controller uses 24-bit byte addresses to access DSP memory.

The CPU can access the buffer RAM, including the descriptor registers, via I/O space. The CPU writes 16-bit values to I/O space. However, when the CPU writes to the RAM, the high eight bits are ignored, and when the CPU reads from the RAM, the high eight bits are 0s.

17.2.2 Transferring Data Between the USB Host and the DSP Memory

Figure 17−2 shows, at a high-level, how data travels between the USB host and the DSP memory when a C55x DSP handles the USB activity for a USB device. During IN transfers, the SIE (serial interface engine) converts the parallel data from the UBM into a serial data stream for the host. During OUT transfers, the SIE converts the host’s serial data into a parallel format for the UBM. The UBM either moves data from the SIE to the buffer RAM or from the buffer RAM to the SIE. Before the UBM transfers data to the SIE, the CPU or the USB DMA controller must put the data into the buffer RAM. When the CPU or the DMA controller is ready to move data to the DSP memory, it must wait for the UBM to move the data from the SIE to the buffer RAM.

Figure 17−2. Path for Data Transferred Between the Host and the DSP Memory



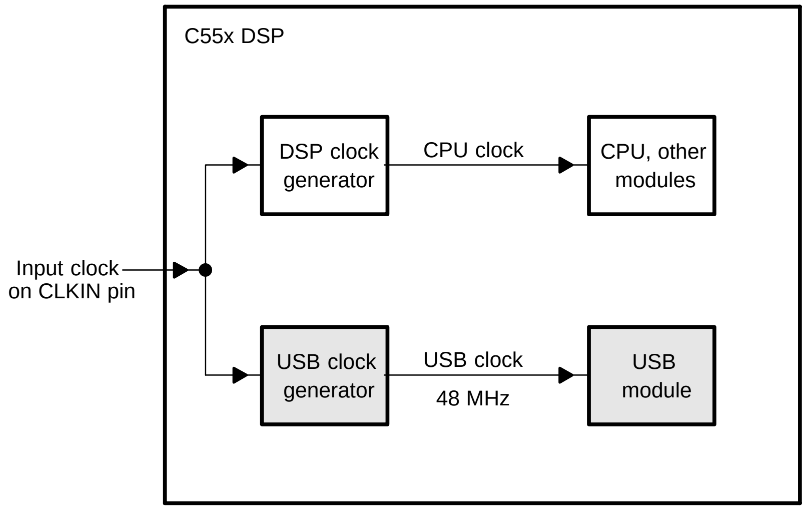
17.2.3 Clock Generation for the USB Module

As shown in Figure 17−3, the USB module has a dedicated clock generator that is independent of the DSP clock generator. Both generators receive their input from the CLKIN pin. The DSP clock generator supplies the CPU clock that is used by the CPU and most of the other modules inside the DSP. The USB clock generator supplies the clock needed to operate the USB module. Because the generators are independent, if an IDLE instruction turns off the DSP clock generator, the USB module can keep running.

Note:

The USB module requires a 48-MHz clock. The clock on the CLKIN pin may vary, but you must program the USB clock generator to produce a 48-MHz clock.

Figure 17−3. Clock Generation for the USB Module



To understand the USB clock generator, see the description of the DSP clock generator in Chapter 3. The two clock generators are nearly identical in function. The USB clock mode register, USBCLKMD (shown in Figure 17−4), has the same bit fields of the DSP clock mode register (CLKMD), which is described in section 3.8 (page 3-12). However, the reset value of the BYPASS DIV bits in USBCLKMD is *not* determined by a pin. These bits are always reset to 01b, providing a USB clock that is half the speed of the input clock. For the I/O address of USBCLKMD, check the data manual for your C55x DSP.

Figure 17−4. USB Clock Mode Register (USBCLKMD)

15	14	13	12	11−7
Rsvd	IAI	IOB	TEST (keep 0)	PLL MULT
	R/W − 0	R/W − 1		R/W − 00000

6−5	4	3−2	1	0
PLL DIV	PLL ENABLE	BYPASS DIV	BREAKLN	LOCK
R/W − 00	R/W − 0	R/W − 01	R − 1	R − 0

Legend:

R	Read-only access
R/W	Read/write access
− X	X is the value after a DSP reset.

17.3 USB Buffer Manager (UBM)

When data is to be moved to or from the buffer RAM, the UBM accesses one of the following buffers in the buffer RAM:

Buffers For Transfers To DSP Memory	**Buffers Transfers From DSP Memory**
Control endpoint buffers	
OUT endpoint 0 buffer	IN endpoint 0 buffer
General-purpose endpoint buffers	
OUT endpoint 1 buffer (X or Y)	IN endpoint 1 buffer (X or Y)
OUT endpoint 2 buffer (X or Y)	IN endpoint 2 buffer (X or Y)
OUT endpoint 3 buffer (X or Y)	IN endpoint 3 buffer (X or Y)
OUT endpoint 4 buffer (X or Y)	IN endpoint 4 buffer (X or Y)
OUT endpoint 5 buffer (X or Y)	IN endpoint 5 buffer (X or Y)
OUT endpoint 6 buffer (X or Y)	IN endpoint 6 buffer (X or Y)
OUT endpoint 7 buffer (X or Y)	IN endpoint 7 buffer (X or Y)

Each of the general-purpose endpoints can be configured to have a single buffer (X) or a double buffer (two buffers, X and Y). This is controlled by the double buffer mode (DBUF) bit in USBxCNFn. If there are two buffers, the UBM keeps track of which buffer to use. If the endpoint is in the non-isochronous mode, the UBM uses the X buffer for a DATA0 packet and the Y buffer for a DATA1 packet.

Each of the endpoint buffers is associated with a programmable count register of the following format:

7	6−0
NAK	CT (bytes)

The NAK bit corresponds to the negative acknowledgement (NAK) of the USB protocol. While the NAK bit is set (NAK = 1), the UBM sends a NAK in response to a host request at that particular endpoint. The UBM does not access the buffer until NAK is cleared (NAK = 0). The role of the NAK bit is summarize in the flow chart of Figure 17−5.

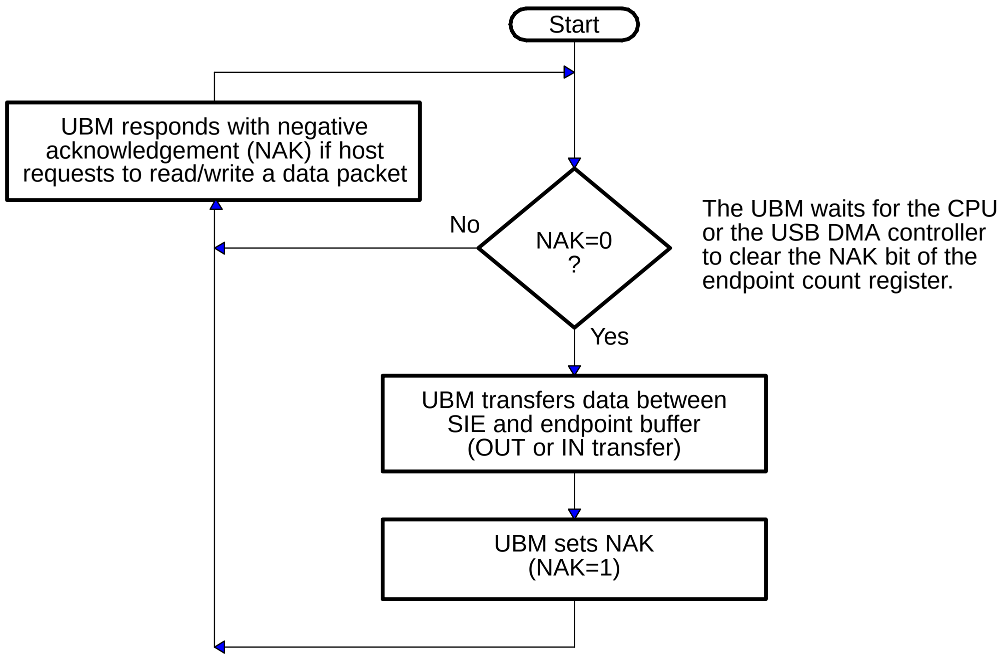
When an OUT packet is received from the serial interface engine (SIE), the UBM writes the incoming data to the appropriate endpoint buffer. Then the UBM sets the NAK bit to prevent the host from overwriting the packet before it is read by the CPU or the USB DMA controller. When an IN token is received, the UBM transfers data from the buffer to the SIE. Then the UBM sets the NAK bit so that the host will not receive the same packet multiple times before a new packet is loaded into the buffer. When NAK is cleared by the CPU or by the USB DMA controller, the UBM can access the buffer again.

The CT (count) field indicates the number of bytes in a transfer between the SIE and an endpoint buffer. For an IN transfer, you must initialize the CT field to tell the UBM how many bytes to read from the buffer. In addition, you must clear NAK when you want the UBM to start. In the case of an OUT transfer, the UBM updates the CT field after moving a new data packet from the SIE to the endpoint buffer.

Note:

In isochronous transfers, the count can be as large as 1023 bytes, requiring a 10-bit CT field. Thus, for isochronous transfers, the count value is extended by three high bits from another register (see section 17.8.3.4 on page 17-75).

Figure 17−5. Role of a NAK bit in UBM Activity (at an Individual Endpoint)


Start
UBM responds with negative acknowledgement (NAK) if host requests to read/write a data packet
No
NAK=0 ?
The UBM waits for the CPU or the USB DMA controller to clear the NAK bit of the endpoint count register.
Yes
UBM transfers data between SIE and endpoint buffer (OUT or IN transfer)
UBM sets NAK (NAK=1)

17.4 USB DMA Controller

The USB module contains a dedicated DMA controller that can transfer data between the DSP memory and the data buffers of the general-purpose endpoints (OUT endpoints 1−7 and IN endpoints 1−7). This DMA controller cannot access the control endpoints (OUT endpoint 0 and IN endpoint 0).

Note:

The information in this section assumes that you have *not* enabled the host-DMA mode. For details about this mode, see section 17.5 on page 17-40.

17.4.1 Advantage of Using the USB DMA Controller

The USB DMA controller transfers data between the endpoint buffers and the DSP memory with minimal CPU involvement. The CPU tells the USB DMA controller to begin a data transfer; then it can continue with other tasks while the controller moves the data. The controller notifies the CPU of the transfer status via GO and RLD status flags and interrupts (see section 17.6.3 on page 17-50).

17.4.2 Things To Consider

Keep the following facts in mind when you use the USB DMA controller:

❑ Each of the general-purpose endpoints must be in double-buffer mode (DBUF = 1 in USBOCNF1−USBOCNF7 and in USBICNF1−USBICNF7). The USB DMA controller assumes there an X buffer and a Y buffer for each general-purpose endpoint, and it accesses the buffers alternately, beginning with the X buffer.

❑ The USB DMA controller accesses the DSP memory via the auxiliary port of the DSP DMA controller (see Chapter 4). This auxiliary port is also used by the host port interface (HPI). The USB module has the higher priority and thus can delay HPI memory accesses.

❑ The DSP DMA controller must share the external memory interface (EMIF) with other parts of the DSP. The EMIF handles requests from throughout the DSP according to a preset priority ranking. When the USB DMA controller must access external memory, the DSP DMA controller sends a request to the EMIF and waits to be serviced.

If the USB DMA controller is not used:

- ❏ Make sure the software does not write to the DMA control register (USBODCTLn or USBIDCTLn) of any endpoint. Writing 1 to the GO bit of any DMA control register initiates a DMA transfer in the controller. In addition, if the controller finishes a DMA transfer and finds that the RLD bit is 1, the controller will perform another transfer.
- ❏ Do not enable RLD and GO interrupt requests in the registers USBODIE and USBIDIE.

17.4.3 Interaction Between the CPU and the USB DMA Controller

Table 17−1 shows how the CPU and the USB DMA controller interact. Each action of the CPU, of course, depends on the instructions in your code. Figure 17−6 (a) (page 17-16) shows how DMA activity is affected by the GO and RLD bits set by the CPU and how it is affected by the NAK bit in an endpoint buffer count register. Figure 17−6 (b) (page 17-17) shows how the CPU (via your code) can handle GO and RLD reports from the controller.

After a DMA transfer, the GO bit of USBxDCTLn is cleared if the RLD bit of USBxDCTLn is 0. If RLD is 1, and neither OVF nor STP is set in USBxDCTLn, the controller performs a DMA reload operation (see section 17.4.5 on page 17-18): The contents of the primary address and size registers are swapped with the contents of the reload address and size registers. After a reload operation, the controller automatically starts a new DMA transfer.

Table 17−1. CPU-Initiated DMA Transfers

Action of the CPU (Executing Your Code)	Action of the USB DMA Controller
Initialize the DMA context registers. (Each general-purpose endpoint has eight DMA context registers; see section 17.8.2 on page 17-59).	Behave according to the contents of the DMA context registers.
Issue a go command (set the GO bit). The GO bit is in the DMA control register for the endpoint (USBODCTLn or USBIDCTLn). Before initiating a new transfer, poll the GO bit to make sure the previous transfer (or series of transfers) is complete (GO = 0).	Respond to a go command. When the CPU sets the GO bit, the controller begins polling the NAK bit in the X-/Y-buffer count register. When NAK = 1, the controller begins the DMA transfer, unless the endpoint is in the isochronous mode (ISO = 1) . When ISO = 1, the controller also waits for a start-of-frame packet (SOF) on the bus.
Set or clear the RLD (reload) bit as desired. The RLD bit is in the DMA control register for the endpoint. Set RLD if you want to tell the controller to begin another transfer after the current transfer is complete. Make sure you initialize the reload address and size registers first.	Behave according to the value of RLD. Once a DMA transfer is complete, the controller checks the RLD bit. If RLD = 0, the controller stops, clears GO, and waits for the CPU to set GO again. If RLD = 1, the controller performs a DMA reload operation, clears RLD, and begins another transfer (if NAK = 1).
Issue a stop command (optional). To stop the controller before it would normally stop itself, set the STP bit (STP = 1) in the DMA control register for the endpoint.	Respond to a stop command. The controller normally stops when it has completed a transfer and the RLD bit is 0. However, if the CPU sets the STP bit for the endpoint, the controller stops its activity on the next packet boundary or at the end of the current DMA transfer, whichever happens first. As it stops, the controller clears the STP and GO bits.
Enable/disable interrupts, and respond to interrupts. Using the GO and RLD interrupt enable registers, individually enable/disable GO and RLD interrupts. If an interrupt is enabled, it is passed to the CPU as a USB interrupt. The interrupt service routine (ISR) can read USBINTSRC (see section 17.8.5.1 on page 17-85) to determine the interrupt source. Then the ISR can execute the appropriate subroutine.	Generate interrupts. When the controller completes a transfer and RLD = 0, the controller clears the GO bit and sets the GO interrupt flag. The RLD interrupt flag is set when the controller completes a reload operation and clears the RLD bit. When an interrupt flag is set, the corresponding interrupt (if enabled) is sent to the CPU. For information about the GO and RLD flags and interrupts, see section 17.6.3 on page 17-50.
Read status information. To monitor the activity of the controller, read the status bits in the DMA control register and the flag bits in the interrupt flag registers.	Record status information for the CPU. The controller modifies bits in the DMA control register and in the interrupt flag registers to notify the CPU of specific actions or errors.

Figure 17−6. Activity for CPU-Initiated DMA Transfers

(a) USB DMA Controller Flow (at an Individual Endpoint)

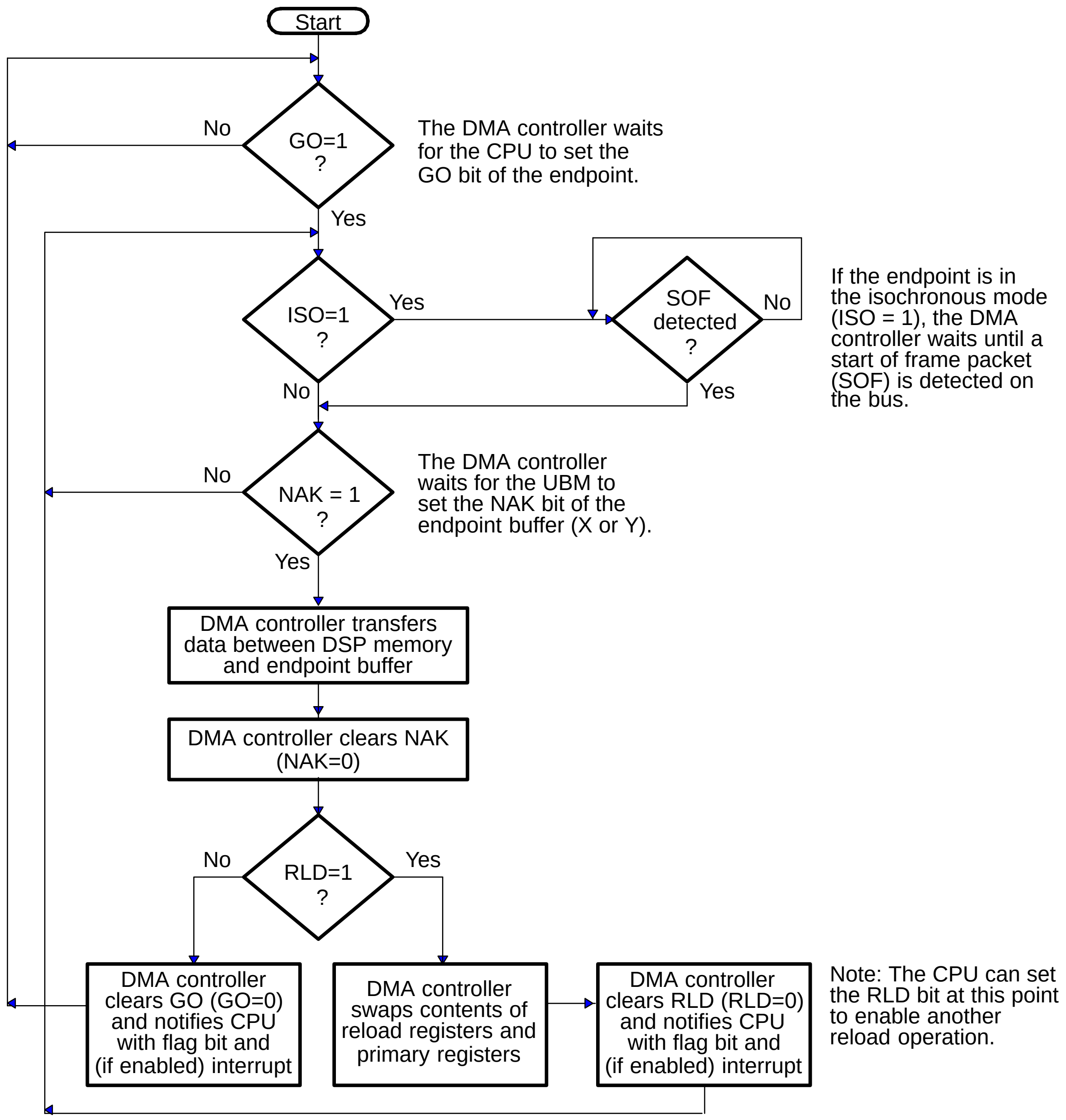
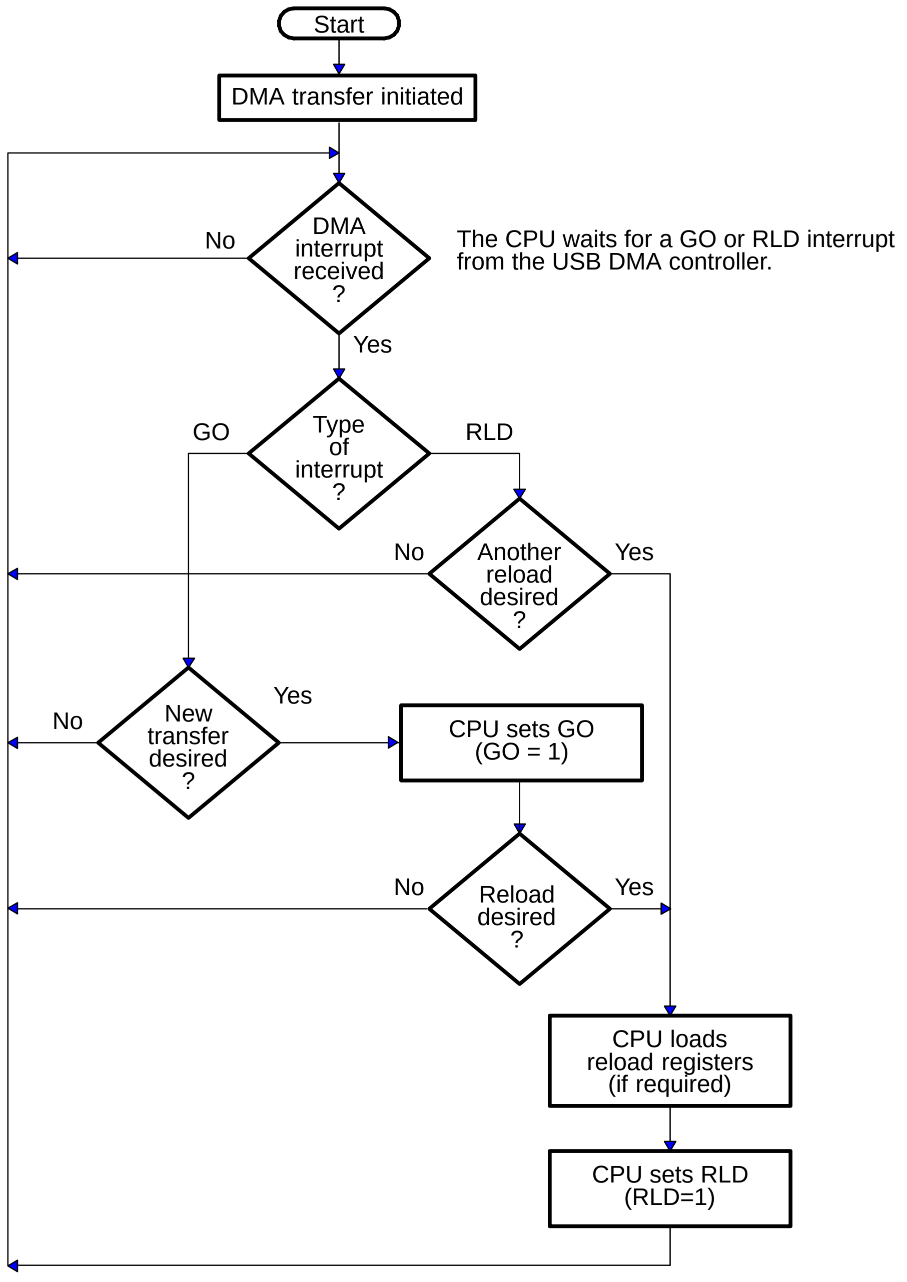

Start
GO=1 ?
No
Yes
The DMA controller waits for the CPU to set the GO bit of the endpoint.
ISO=1 ?
Yes
No
SOF detected ?
No
Yes
If the endpoint is in the isochronous mode (ISO = 1), the DMA controller waits until a start of frame packet (SOF) is detected on the bus.
NAK = 1 ?
No
Yes
The DMA controller waits for the UBM to set the NAK bit of the endpoint buffer (X or Y).
DMA controller transfers data between DSP memory and endpoint buffer
DMA controller clears NAK (NAK=0)
RLD=1 ?
No
Yes
DMA controller clears GO (GO=0) and notifies CPU with flag bit and (if enabled) interrupt
DMA controller swaps contents of reload registers and primary registers
DMA controller clears RLD (RLD=0) and notifies CPU with flag bit and (if enabled) interrupt
Note: The CPU can set the RLD bit at this point to enable another reload operation.

Figure 17−6. Activity for CPU-Initiated DMA Transfers (Continued)

(b) CPU Flow (at an Individual Endpoint)



17.4.4 Automatic Alternating Accesses of the X and Y Buffers

For non-isochronous USB transfers, the USB DMA controller automatically tracks the data packet type and determines which of the endpoint's buffers to access, X or Y:

Data Packet Type	USB DMA Controller Accesses ...
DATA0	X buffer OUT transfer: The controller reads data from the X buffer. IN transfer: The controller writes data to the X buffer.
DATA1	Y buffer OUT transfer: The controller reads data from the Y buffer. IN transfer: The controller writes data to the Y buffer.

For isochronous USB transfers, the USB DMA controller uses the X buffer first and then alternates between the Y buffer and the X buffer.

17.4.5 DMA Reload Operation (Automatic Register Swapping)

For each endpoint n (n = 1, 2, 3, 4, 5, 6, or 7), the USB DMA controller has a set of primary registers and a set of reload registers for the DMA transfer size and the DSP memory address (see Table 17−2). The primary registers are used for the current DMA transfer, and the reload registers are used to queue up an address and size for the next transfer.

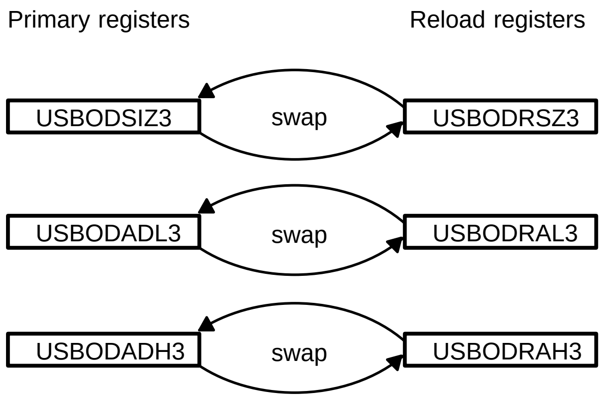
Table 17−2. Primary USB DMA Size and Address Registers and the Corresponding Reload Registers

Endpoint	Primary Register	Register Contains ...	Reload Register
OUT endpoint n	USBODSIZn	DMA transfer size in bytes	USBODRSZn
	USBODADLn	Low 16 bits of DSP memory address	USBODRALn
	USBODADHn	High 8 bits of DSP memory address	USBODRAHn
IN endpoint n	USBIDSIZn	DMA transfer size in bytes	USBIDRSZn
	USBIDADLn	Low 16 bits of DSP memory address	USBIDRALn
	USBIDADHn	High 8 bits of DSP memory address	USBIDRAHn

As mentioned in Table 17−1 on page 17-15, when the USB DMA controller completes a DMA transfer, it checks the RLD (reload) bit of the appropriate DMA control register. If RLD = 1, the controller performs a DMA reload operation: The controller swaps the contents of the primary registers and the reload registers. Example 17−1 shows a reload operation involving the registers for OUT endpoint 3.

This register swapping saves CPU time if you repeatedly toggle between the same two blocks in memory. Rather than putting new values into the reload registers between transfers, you can set the reload registers once and initiate a reload operation each time you want the controller to access the other block.

Example 17−1. DMA Reload Operation for OUT Endpoint 3



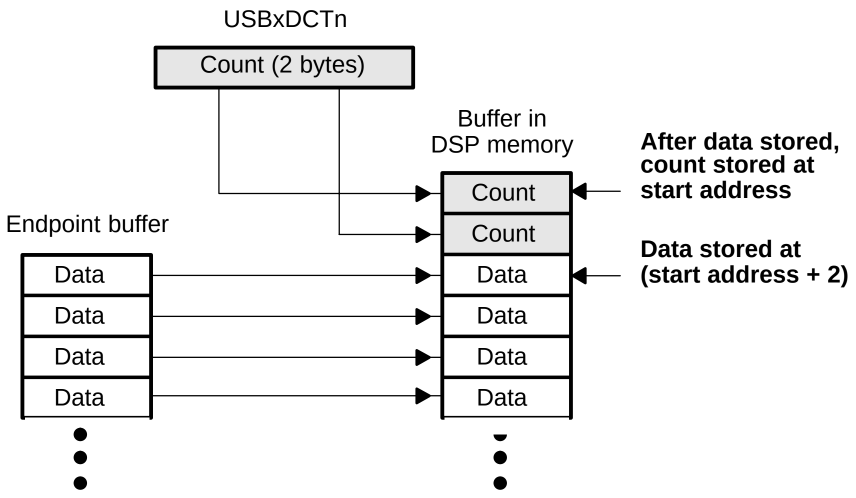
17.4.6 Transfer Count Saved to DSP Memory For an OUT Transfer

For each new DMA transfer, the USB DMA controller ensures that the transfer count for the endpoint starts at 0. When you give a go command (GO = 1), the controller clears the endpoint’s count register (USBxDCTn) before moving the data. Likewise, after the controller completes a DMA reload operation (see section 17.4.5), it clears the count register before beginning the next DMA transfer.

At times, an OUT transfer will end with a short packet. If the USB DMA controller performs a DMA reload operation and immediately starts the next transfer, the count register is cleared before you can read the number of bytes in the packet. To prevent this loss of information, the controller copies the count to the DSP memory after every read from an endpoint buffer.

Figure 17−7 shows the positions of the data and the count in DSP memory. The start address is the address programmed in the primary address registers (USBxDADHn and USBDADLn). When the controller moves the data, it begins writing at (start address + 2). When all of the data has been moved, the controller stores the 2-byte count at the start address.

Figure 17−7. Storage of Transfer Count For an OUT Transfer



To properly read data from an OUT transfer, follow these guidelines:

- When you define the size of the buffer in DSP memory, include an additional two bytes for the DMA transfer count. Specifically, the buffer must be two bytes larger than the size you programmed in USBxDSIZn.
- When you read the data, keep in mind that the data starts two bytes after the start address you specified in USBxDADHn and USBxDADLn.

17.4.7 Configuring the USB DMA Controller

To configure an endpoint that will be accessed with CPU-initiated DMA transfers, use the instructions in the following paragraphs. In the register and bit names that appear in these paragraphs, a lowercase x can be O (for OUT) or I (for IN), and a lowercase n can be 1, 2, 3, 4, 5, 6, or 7 (indicating the endpoint number). For example, one of the possible values for USBxDCTLn is USBIDCTL4, which represents the DMA control register for IN endpoint 4.

17.4.7.1 Set the Transfer Size

Register(Field)	*symval*	Value	Description
USBxDSIZn(15−0)		1−65535	Number of bytes to be transferred
USBxDCTn(15−0)		1−65535	Number of bytes that have been transferred

For an endpoint n, you must tell the USB DMA controller how many bytes to transfer between the DSP memory and the endpoint. Write the number of bytes (up to 64K bytes) to USBxDSIZn.

The count value in USBxDCTn is cleared before each new DMA transfer and is updated with the number of bytes transferred at the end of the transfer. If you specified a DMA reload operation (RLD = 1), the controller automatically clears USBxDCTn before beginning the next DMA transfer.

17.4.7.2 Set the DSP Memory Address

Register(Field)	*symval*	Value	Description
USBxDADHn(15−0)		0000h−00FFh	High 8 bits of the DSP memory address
USBxDADLn(15−0)		0000h−FFFFh	Low 16 bits of the DSP memory address

Because each endpoint has a dedicated DMA channel, the USB DMA controller knows the location of the buffer for endpoint n, but you must tell the controller which address to use when accessing the DSP memory. The controller accesses bytes in the endpoint buffer.

The address you specify must be a **byte address** with 24 bits. Load the 8 high bits of the address into USBxDADHn. (Bits 15−8 of USBxDADHn must contain 0s). Load the 16 low bits of the address into USBxDADLn.

In addition, the address must be **16-bit aligned**. Make sure the least significant bit (LSB) of USBxDADL is 0.

17.4.7.3 Enable/Disable a DMA Reload Operation and, If Necessary, Initialize the Reload Registers

Register(Field)	*symval*	Value	Description
USBxDCTLn(RLD)		0	No pending DMA reload operation. (Writing 0 to RLD has no effect.)
		1	Enable DMA reload operation.
USBxDRSZn(15−0)		0−65535	Reload-size value for USBxDSIZn
USBxDRAHn(15−0)		0000h−00FFh	Reload-address value for USBxDADHn
USBxDRALn(15−0)		0000h−FFFFh	Reload-address value for USBxDADLn

The USB DMA controller checks RLD at the end of each DMA transfer to determine whether to stop or to begin another transfer. If you want the controller to begin another transfer after the first, initialize the reload registers (USBxDRSZn, USBxDRAHn, and USBxDRALn) and then set the RLD bit. When the controller is done with the first transfer and it finds RLD = 1, it performs a DMA reload operation (see section 17.4.5 on page 17-18) and begins the next transfer. If the controller is stopped (GO = 0), setting RLD has no effect.

Each time the controller performs a DMA reload operation, it clears RLD and notifies the CPU. To notify the CPU, the controller sets the endpoint’s RLD interrupt flag bit in USBxDRIF. In addition, if the endpoint's DMA interrupts are enabled in USBxDIE, the controller sends an interrupt request to the CPU. To keep DMA transfers continuous, the CPU can set the RLD bit again before the end of each DMA transfer (that is, before the GO bit is cleared to 0).

17.4.7.4 If Desired, Enable DMA Interrupt Requests

Register(Field)	*symval*	Value	Description
USBxDIE(xEn)		0	Disable DMA GO and RLD interrupt requests.
		1	Enable DMA GO and RLD interrupt requests.

If DMA interrupts for an endpoint are enabled, the USB DMA controller can generate a GO interrupt request each time it clears the GO bit of USBxDCTLn; that is, it can notify the CPU that the controller has stopped. Similarly, the controller can use a RLD interrupt request to notify the CPU that a DMA reload operation is done (when the controller clears the RLD bit of USBxDCTLn).

Both of these DMA interrupt requests are enabled or disabled by a bit in one of the DMA interrupt enable registers. The interrupt enable bits for OUT endpoints 1−7 are in USBODIE; those for IN endpoints 1−7 are in USBIDIE. You enable GO and RLD interrupt requests for an endpoint by writing to the corresponding interrupt enable bit. For example, to enable DMA interrupt requests for IN endpoint 6, write a 1 to USBIDIE(IE6). For OUT endpoint 2, write a 1 to USBODIE(OE2).

For more details about the DMA interrupt requests, see section 17.6.3 on page 17-50.

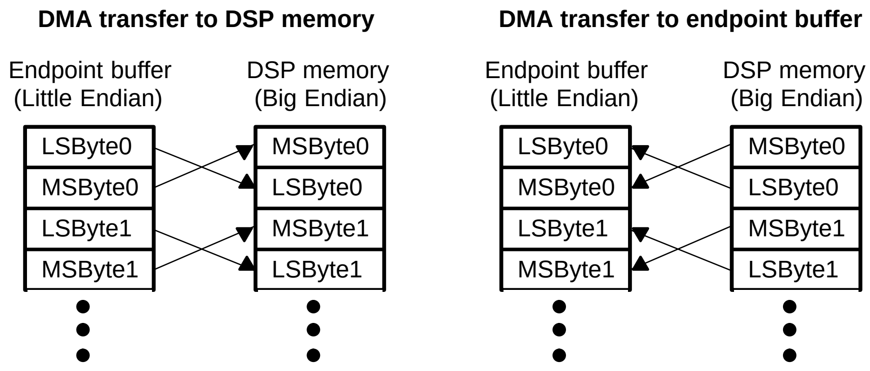
17.4.7.5 Determine Whether to Reverse the Endianness (Orientation) of Data During DMA Transfers

Register(Field)	*symval*	Value	Description
USBxDCTLn(END)		0	Do not change the order of the bytes in the next DMA transfer.
		1	Reverse the endianness of each word moved in the next DMA transfer.

In the Big Endian data orientation for words, the first byte is the most significant byte (MSByte) of the word. In the Little Endian orientation, the first byte is the least significant byte (LSByte). The C55x CPU assumes that data in memory has the Big Endian orientation. When the UBM transfers data between the SIE and the endpoint buffer, the UBM does not change the order of any data bytes. However, by using the END bit, you can tell the USB DMA controller to swap the byte orientation before writing to the endpoint buffer or after reading from the endpoint buffer.

Figure 17−8 shows the effect of making END = 1 for an endpoint: The USB DMA controller reverses the endianness of the words it transfers between the DSP memory and the endpoint buffer. When END = 0 for an endpoint, the USB DMA controller does not change the order of the bytes.

Figure 17−8. The Effect of END = 1 on USB DMA Transfers



17.4.7.6 Enable/Disable Concatenation

Register(Field)	*symval*	Value	Description
USBxDCTLn(CAT)		0	Disable concatenation.
		1	Enable concatenation.

When the packet sent or received at an endpoint is larger than the DMA transfer size, you may want to set CAT = 1, so that the USB DMA controller does multiple DMA transfers for a single OUT or IN transfer on the USB. For more details, see the description for the CAT bit in section 17.8.2.1 (page 17-60).

17.4.7.7 Select Whether to Require Short Packets

Register(Field)	*symval*	Value	Description
USBxDCTLn(SHT)		0	Short packets not required
		1	Short packets required

Typically, a USB transfer ends with a short packet, a packet that is shorter than the maximum allowable size for the endpoint. If a maximum-size packet is the last packet, the transmitter (host or slave device) can send one more packet, of zero length, to indicate that there is no more data. The SHT bit tells the USB DMA controller whether to wait for/generate 0-byte packets when the transfer ends with a maximum-size packet. For more details, see the description for the SHT bit in section 17.8.2.1.

17.4.7.8 Select Whether a Missing Packet is an Error During Isochronous Transfers

Register(Field)	*symval*	Value	Description
USBxDCTLn(EM)		0	Do not halt the USB DMA controller in response to a missing packet. Treat a missing packet as a zero-length packet.
		1	A missing packet is an error and will stop the USB DMA controller.

If the USB DMA controller is handling data packets for an isochronous endpoint, this bit determines how the controller will respond if no packet is received in/transmitted from the endpoint buffer during the current USB frame. For more details, see the description for the EM bit in section 17.8.2.1.

17.4.8 Monitoring CPU-Initiated DMA Transfers

To monitor an endpoint that is involved in CPU-initiated DMA transfers, use the instructions in the following paragraphs. In the register and bit names that appear in these paragraphs, a lowercase x can be O (for OUT) or I (for IN), and a lowercase n can be 1, 2, 3, 4, 5, 6, or 7 (indicating the endpoint number). For example, one of the possible values for USBxDCTLn is USBODCTL5, which represents the DMA control register for OUT endpoint 5.

17.4.8.1 Checking the Transfer Count (USBxDCTn)

Register(Field)	*symval*	Value	Description
USBxDCTn(15−0)		0−65535	Indicates how many bytes have been transferred

The USB DMA controller clears USBxDCTn before each new DMA transfer, including those transfers following DMA reload operations. USBxDCTn is updated with the number of bytes transferred at the end of a transfer. You can read this register to determine how many bytes were moved.

17.4.8.2 Determining Whether a DMA Transfer is In Progress/Done

Register(Field)	*symval*	Value	Description
USBxDCTLn(GO)		0	The USB DMA controller is idling (ready for the next DMA transfer).
		1	A DMA transfer is in progress.
USBxDGIF(xEn)		0	No DMA GO interrupt is pending.
		1	The USB DMA controller has completed the current DMA transfers or series of transfers and has cleared the GO bit. This event also generates a GO interrupt if the interrupt is enabled by USBxDIE(xEn).

When the CPU sets the GO bit, the DMA controller begins a DMA transfer. At the end of the transfer, if the RLD bit is 1, the controller does not clear GO. Instead the controller performs a DMA reload operation (see section 17.4.5 on page 17-18) and begins a new transfer with the new address and size. By using repeated reload operations, you can have the controller perform a series of transfers.

When the controller completes a transfer and finds RLD = 0, it clears GO to 0. In addition, it sets the endpoint's GO flag in USBxDGIF and generate an interrupt request. For example, if the controller is done at OUT endpoint 4, it sets the OE4 bit in USBODGIF. If the corresponding interrupt request is enabled by the OE4 bit in USBODIE, an interrupt request is generated.

17.4.8.3 Determining Whether a DMA Reload Operation is In Progress/Done

Register(Field)	*symval*	Value	Description
USBxDCTLn(RLD)		0	USB DMA controller is done with the previously requested reload operation.
		1	USB DMA controller is waiting to complete a reload operation.
USBxDRIF(xEn)		0	No DMA RLD interrupt is pending.
		1	The USB DMA controller has completed the DMA reload operation and has cleared RLD. This event also generates a RLD interrupt if the interrupt is enabled by USBxDIE(xEn).

When the USB DMA controller completes a DMA transfer, it checks the RLD bit. If RLD = 1, the controller performs a DMA reload operation. When the DMA reload operation is done, the controller clears RLD. In addition, it sets the endpoint's RLD flag in USBxDRIF and can generate an interrupt request. For example, if the controller complete a DMA reload operation for OUT endpoint 4, it sets the OE4 bit in USBODRIF. If the corresponding interrupt request is enabled by the OE4 bit in USBODIE, an interrupt request is generated.

17.4.8.4 Checking for an Overflow or Underflow Condition

Register(Field)	*symval*	Value	Description
USBxDCTLn(OVF)		0	No overflow/underflow detected
		1	Overflow/underflow detected

Essentially, an overflow condition occurs when too many bytes are arriving in an endpoint buffer, and an underflow condition occurs when not enough bytes are available to be read from the endpoint buffer. For more details, see the description for the OVF bit in section 17.8.2.1 (page 17-60).

17.4.8.5 Watching for Missed Packets During Isochronous Transfers

Register(Field)	*symval*	Value	Description
USBxDCTLn(PM)		0	No missing packet.
		1	Missing packet: A packet did not arrive in the previous USB frame for the endpoint.

If the USB DMA controller is handling data packets for an isochronous endpoint, and you can determine how the controller will respond if no packet is received in/transmitted from the endpoint buffer during the current USB frame. If you want the controller to consider a missing packet an error condition, set the EM bit in USBxDCTLn. If EM = 1, you can watch for missing packets by monitoring the PM bit in USBxDCTLn. EM and PM are described in section 17.8.2.1 (page 17-60).

17.4.9 USB DMA State Tables and State Diagrams

This section contains the following state tables to summarize the status of the USB DMA controller under various conditions:

- Table 17−3: Non-isochronous IN DMA Transfer (page 17-27)
- Table 17−4: Non-isochronous OUT DMA Transfer (page 17-29)
- Table 17−5: Isochronous IN DMA Transfer (page 17-31)
- Table 17−6: Isochronous OUT DMA Transfer (page 17-35)

This section also includes the following state diagrams to support the isochronous transfer state tables:

- Figure 17−9: Missing Packet Response for Isochronous IN DMA Transfer (page 17-38)
- Figure 17−10: Missing Packet Response for Isochronous OUT DMA Transfer (page 17-39)

Table 17−3. State Table: Non-Isochronous IN DMA Transfer

Description		**Initial State**						**End State**						
DMA transfer state	**Programmer view**	**Bytes free in endpt buffer**	**Bytes in DMA transfer**	**STP**	**RLD**	**CAT**	**SHT**	**End of current DMA transfer**	**Reload and swap**	**Reset GO**	**Reset STP**	**Update DMA count**	**Send packet (clear NAK)**	**DMA transfer activity**
Normal transfer in progress		> 0	> 0	x	x	x	x							In progress
Endpoint buffer full	Stop requested	0	x	1	x	x	x	1		1	1			Idle
DMA transfer completion	Stop requested	> 0	0	1	x	x	x	1		1	1			Idle
Endpoint buffer full, more data remaining in DMA transfer		0	> 0	0	x	x	x					1	Max	Idle
DMA transfer completion, endpoint buffer full		0	0	0	0	1	x	1	0	1		1	Max	Idle
					1				1	0				
DMA transfer completion, endpoint buffer full		0	0	0	0	0	0	1	0	1		1	Max	Idle
					1				1	0				
DMA transfer completion, endpoint buffer full	Short packet requested	0	0	0	x	0	1					1	Max	Idle until current packet moves out, then prepare a 0-byte packet

Table 17−3. State Table: Non-Isochronous IN DMA Transfer (Continued)

Description		**Initial State**						**End State**						
DMA transfer state	**Programmer view**	**Bytes free in endpt buffer**	**Bytes in DMA transfer**	**STP**	**RLD**	**CAT**	**SHT**	**End of current DMA transfer**	**Reload and swap**	**Reset GO**	**Reset STP**	**Update DMA count**	**Send packet (clear NAK)**	**DMA transfer activity**
DMA transfer completion, endpoint buffer **not** full		> 0	0	0	0	0	x	1	0	1		1	Short	Idle
					1				1	0				
DMA transfer completion, endpoint buffer **not** full	CAT requested, next buffer is **not** ready yet	> 0	0	0	0	1	x	1		1		1		Pause
DMA transfer completion, endpoint buffer **not** full	CAT requested, next buffer is ready	> 0	0	0	1	1	x	1	1			1		In progress

Table 17−4. State Table: Non-Isochronous OUT DMA Transfer

Description		**Initial State**							**End State**							
DMA transfer state	**Programmer view**	**Received packet size**	**Bytes in endpt buffer**	**Bytes in DMA transfer**	**STP**	**RLD**	**CAT**	**SHT**	**End of current DMA transfer**	**Reload and swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA count**	**Clear NAK for next packet**	**DMA transfer activity**
Normal transfer in progress		x	> 0	> 0	x	x	x	x								In progress
Packet transfer completion	Stop requested	x	0	x	1	x	x	x	1		1	1		1	1	Idle
DMA transfer completion	Stop requested	x	> 0	0	1	x	x	x	1		1	1				Idle
Packet transfer completion, more data remaining in DMA transfer		Max	0	> 0	0	x	x	x						1	1	Idle
Packet transfer completion, more data remaining in DMA transfer		Short	0	> 0	0	0	x	x	1	0	1			1	1	Idle
						1				1	0					
DMA transfer completion, packet fits exactly		Max	0	0	0	0	x	0	1	0	1			1	1	Idle
						1				1	0					
DMA transfer completion, packet fits exactly		Max	0	0	0	0	1	x	1	0	1			1	1	Idle
						1				1	0					

Table 17−4. State Table: Non-Isochronous OUT DMA Transfer (Continued)

Description		**Initial State**							**End State**							
DMA transfer state	**Programmer view**	**Received packet size**	**Bytes in endpt buffer**	**Bytes in DMA transfer**	**STP**	**RLD**	**CAT**	**SHT**	**End of current DMA transfer**	**Reload and swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA count**	**Clear NAK for next packet**	**DMA transfer activity**
DMA transfer completion, packet fits exactly	Short packet requested	Max	0	0	0	x	0	1						1	1	Idle, expecting a 0−byte packet
DMA transfer completion, packet fits exactly	Short packet requested	Short	0	0	0	0	x	x	1	0	1			1	1	Idle
						1				1	0					
DMA transfer completion, more data remaining in endpoint buffer	Overflow condition	x	> 0	0	0	x	0	x	1		1		1	1		Idle
DMA transfer completion, more data remaining in endpoint buffer	CAT requested, next buffer is not ready yet	x	> 0	0	0	0	1	x	1		1			1		Pause
DMA transfer completion, more data remaining in endpoint buffer	CAT requested, next buffer is ready	x	> 0	0	0	1	1	x	1	1				1		In progress

Table 17−5. State Table: Isochronous IN DMA Transfer

Description		Initial State								Final State							
DMA Transfer State	**Program-mer view**	**DMA transfer complete before SOF**	**Bytes free in Endpt Buffer**	**Bytes in DMA transfer**	**Missing Packet Error †**	**STP**	**RLD**	**CAT**	**SHT**	**End of Current DMA transfer**	**Reload and Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Send Packet (clear NAK)**	**DMA Transfer Activity**
DMA failed to keep up with USB		0	x	x	x	x	x	x	x	1		1		1			Idle
Normal transfer in progress		1	> 0	> 0	x	x	x	x	x								In progress
Endpoint buffer full, more data remaining in DMA transfer	Stop requested	1	0	> 0	x	1	x	x	x	1		1	1		1	Max	Idle
DMA transfer completion, endpoint buffer is **not** full	Stop requested	1	> 0	0	x	1	x	x	x	1		1	1				Idle
DMA transfer completion, endpoint buffer is full	Stop requested	1	0	0	x	1	x	x	x	1		1	1		1	Max	Idle

† Entries in this column are taken from the missing packet response state diagram of Figure 17−9 (page 17-38).

Table 17−5. State Table: Isochronous IN DMA Transfer (Continued)

Description		**Initial State**								**Final State**							
DMA Transfer State	**Program-mer view**	**DMA transfer complete before SOF**	**Bytes free in Endpt Buffer**	**Bytes in DMA transfer**	**Missing Packet Error †**	**STP**	**RLD**	**CAT**	**SHT**	**End of Current DMA transfer**	**Reload and Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Send Packet (clear NAK)**	**DMA Transfer Activity**
Endpoint buffer full, more data remaining in DMA transfer, host missed an IN request earlier		1	0	> 0	1	0	x	x	x	1		1			0	Max	Idle
Endpoint buffer full, more data remaining in DMA transfer		1	0	> 0	0	0	x	x	x						1	Max	Idle
DMA transfer completion, endpoint buffer is **not** full	Current buffer is the last of the transfer	1	> 0	0	0	0	0	0	x	1	0	1			1	Non-max packet	Idle
							1				1	0					

† Entries in this column are taken from the missing packet response state diagram of Figure 17−9 (page 17-38).

Table 17−5. State Table: Isochronous IN DMA Transfer (Continued)

Description		**Initial State**								**Final State**							
DMA Transfer State	**Program-mer view**	**DMA transfer complete before SOF**	**Bytes free in Endpt Buffer**	**Bytes in DMA transfer**	**Missing Packet Error †**	**STP**	**RLD**	**CAT**	**SHT**	**End of Current DMA transfer**	**Reload and Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Send Packet (clear NAK)**	**DMA Transfer Activity**
DMA transfer completion, endpoint buffer is **not** full	CAT requested, next buffer is not ready yet (underflow condition)	1	> 0	0	0	0	0	1	x	1		1		1	1	Non-max packet	Idle
DMA transfer completion, endpoint buffer is **not** full	CAT requested, next buffer is ready	1	> 0	0	0	0	1	1	x	1					1		Start next transfer, fill up rest of endpoint buffer
DMA transfer completion, endpoint buffer full, missing packet error seen		1	0	0	1	0	x	x	x	1		1					Idle
DMA transfer completion, endpoint buffer full		1	0	0	0	0	0	0	0	1	0	1			1	Max	Idle
							1				1	0					

† Entries in this column are taken from the missing packet response state diagram of Figure 17−9 (page 17-38).

Table 17−5. State Table: Isochronous IN DMA Transfer (Continued)

Description		**Initial State**								**Final State**							
DMA Transfer State	**Program-mer view**	**DMA transfer complete before SOF**	**Bytes free in Endpt Buffer**	**Bytes in DMA transfer**	**Missing Packet Error** †	**STP**	**RLD**	**CAT**	**SHT**	**End of Current DMA transfer**	**Reload and Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Send Packet (clear NAK)**	**DMA Transfer Activity**
DMA transfer completion, endpoint buffer full	Short (0-byte) packet requested	1	0	0	0	0	x	0	1						1	Short (zero)	Idle
DMA transfer completion, endpoint buffer full	CAT requested	1	0	0	0	0	0	1	x	1	0	1			1	Max	Idle
							1				1	0					

† Entries in this column are taken from the missing packet response state diagram of Figure 17−9 (page 17-38).

Table 17−6. State Table: Isochronous OUT DMA Transfer

Description		Initial State								Final State							
DMA Transfer State	**Program-mer view**	**DMA transfer complete before SOF**	**Bytes in Endpt Buffer**	**Bytes in DMA transfer**	**Normal, Short, Ignore, Missing Error** †	**STP**	**RLD**	**CAT**	**SHT**	**End of Current DMA transfer**	**Reload and Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Receive Packet (Clear NAK)**	**DMA Transfer Activity**
DMA failed to keep up with USB		0	x	x	x	x	x	x	x	1		1		1			Idle
Normal transfer in progress		1	> 0	> 0	x	x	x	x	x								In Progress
Endpoint buffer is empty	Stop requested	1	0	x	x	1	x	x	x	1		1	1		1	1	Idle
Endpoint buffer empty, more data remaining in DMA transfer		1	0	> 0	Normal	0	x	x	x						1	1	Idle
		1	0	> 0	Short	0	0	x	x	1	0	1			1	1	Idle
							1				1	0					
		1	0	> 0	Ignore	0	x	x	x							1	Idle
		1	0	> 0	Missing Error	0	x	x	x	1		1			1	1	Idle

† Entries in this column are taken from the missing packet response state diagram of Figure 17−10 (page 17-39).

Table 17−6. State Table: Isochronous OUT DMA Transfer (Continued)

Description		**Initial State**								**Final State**							
DMA Transfer State	**Program-mer view**	**DMA transfer complete before SOF**	**Bytes in Endpt Buffer**	**Bytes in DMA transfer**	**Normal, Short, Ignore, Missing Error** †	**STP**	**RLD**	**CAT**	**SHT**	**End of Current DMA transfer**	**Reload and Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Receive Packet (Clear NAK)**	**DMA Transfer Activity**
DMA transfer completion, more data remaining in the endpoint buffer.	Stop requested	1	> 0	0	Normal	1	x	x	x	1		1	1			1	Idle
	Overflow condition	1	> 0	0	Normal	0	x	0	x	1		1		1	1	1	Idle
	CAT requested, next buffer is not ready yet (overflow condition)	1	> 0	0	Normal	0	0	1	x	1		1		1	1	1	Idle
	CAT requested, next buffer is ready	1	> 0	0	Normal	0	1	1	x	1	1				1	0	In progress

† Entries in this column are taken from the missing packet response state diagram of Figure 17−10 (page 17-39).

Table 17−6. State Table: Isochronous OUT DMA Transfer (Continued)

Description		**Initial State**								**Final State**							
DMA Transfer State	**Program-mer view**	**DMA transfer complete before SOF**	**Bytes in Endpt Buffer**	**Bytes in DMA transfer**	**Normal, Short, Ignore, Missing Error** †	**STP**	**RLD**	**CAT**	**SHT**	**End of Current DMA transfer**	**Reload and Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Receive Packet (Clear NAK)**	**DMA Transfer Activity**
DMA transfer completion, endpoint buffer is empty	Stop requested	1	0	0	x	1	x	x	x	1		1	1		1	1	Idle
	CAT requested	1	0	0	Normal	0	0	1	x	1	0	1			1	1	Idle
							1				1	0					
		1	0	0	Normal	0	0	0	0	1	0	1			1	1	Idle
							1				1	0					
	Short (0-byte) Packet expected	1	0	0	Normal	0	x	0	1						1	1	Expect a short packet next
DMA expecting a short (0-byte) packet to end the transfer		1	0	0	Missing Error	0	x	x	x	1		1			1	1	Idle
		1	0	0	Short	0	0	x	x	1	0	1			1	1	Idle
							1				1	0					
		1	0	0	Ignore	0	x	x	x							1	In progress

† Entries in this column are taken from the missing packet response state diagram of Figure 17−10 (page 17-39).

Figure 17−9. Missing Packet Response for Isochronous IN DMA Transfer (Supports Table 17−5 on Page 17-31)

GO: Go command to USB DMA controller
PM: Previous packet missed
EM: Error if missing packet

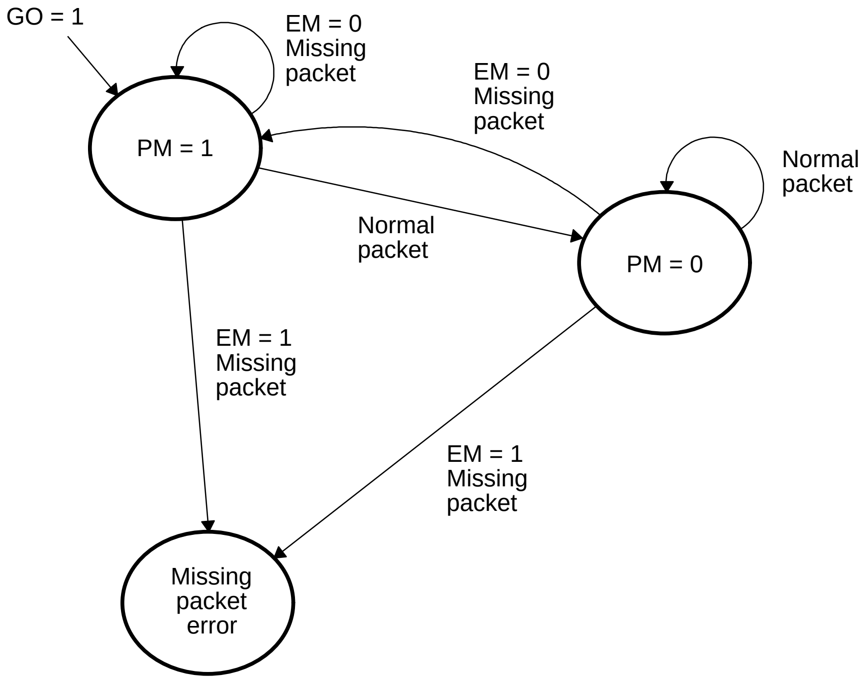
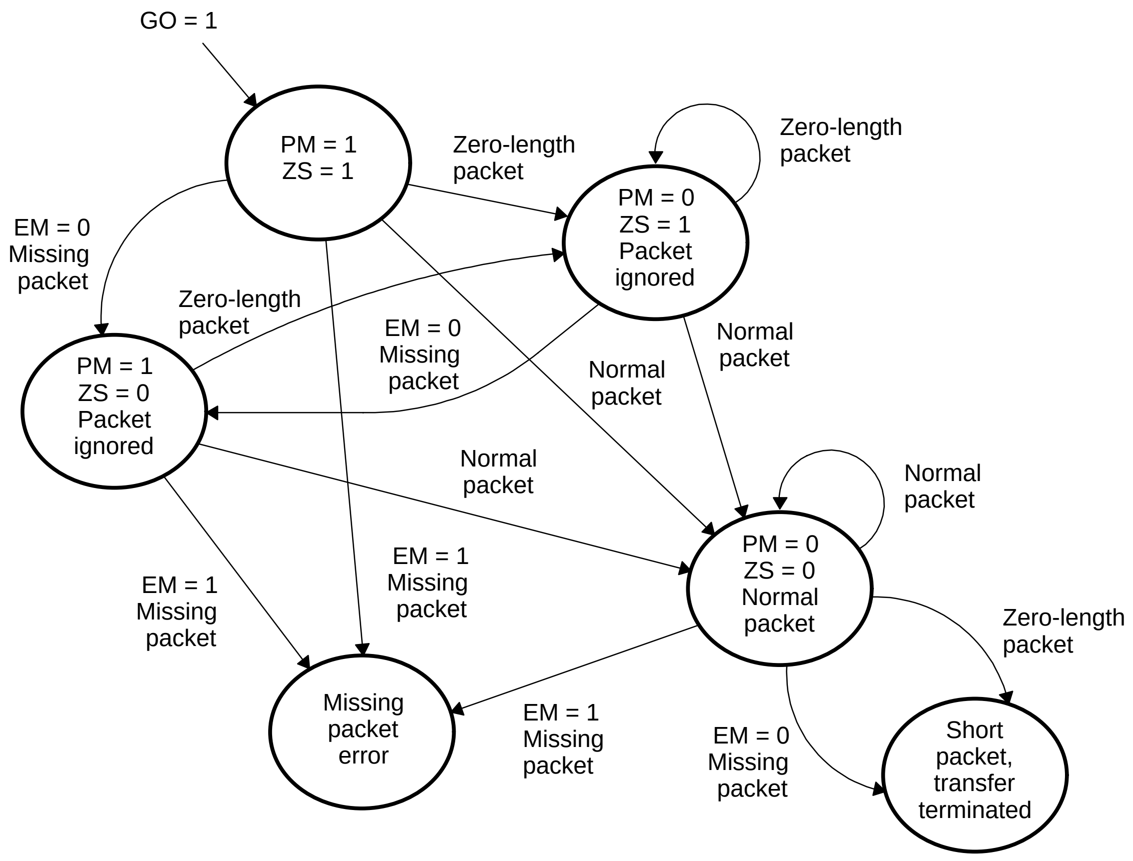


Figure 17−10. Missing Packet Response for Isochronous OUT DMA Transfer (Supports Table 17−6 on Page 17-35)

GO: Go command to USB DMA controller
PM: Previous packet missed
EM: Error if missing packet
ZS: Zero-length packet status



17.5 Host-DMA Mode: Host-Initiated Direct Memory Accesses

In the host-DMA mode, the host can initiate direct memory accesses through two general-purpose endpoints (one OUT endpoint and one IN endpoint). On the USB, the host-DMA endpoints appear as bulk endpoints (endpoints that expect bulk transfers). During enumeration, the device should report that these endpoints are bulk endpoints, and the host should communicate with these endpoints as it would with any bulk endpoint.

Each host-DMA transfer is performed on a packet-by-packet basis. Each transfer begins with an OUT packet containing a 5 byte host-DMA protocol header (described in section 17.5.1). If data is moving from the host to the device, the data being transferred follows the protocol header in the same packet. If the data is moving from the device to the host, the data being transferred is sent in a subsequent IN packet.

When the CPU has relinquished DMA control to the host, the CPU should not write to the GO bit or the RLD bit. However, the CPU can be notified by interrupt flags and (if enabled) interrupts when a host-DMA transfer is complete or if an error has occurred. You do not need to initialize any of the DMA context registers because the details of each transfer are in the header protocol sent by the host.

Note:

Once the two bulk endpoints are selected for this purpose, the endpoints cannot be used for any other purpose until a USB software reset (write 1 to the SOFTRST bit of USBGCTL) or a DSP reset (drive the $\overline{\text{RESET}}$ pin low) is performed.

17.5.1 Protocol Header For the Host-DMA Mode

Each time the host communicates with host-DMA endpoints, the host must first send a 5-byte protocol header to describe the desired transfer to the USB DMA controller. Figure 17−11 shows the values the host must send in the header. Byte 1 tells the USB DMA controller about the data transfer that is requested. Table 17−7 gives the details about this first byte. Bytes 2−5 must contain the DSP memory address as shown in the following figure. Because a C55x DSP uses 24-bit addresses, byte 5 of the header protocol must be 0s.

Figure 17−11. Bytes of the Host-DMA Protocol Header

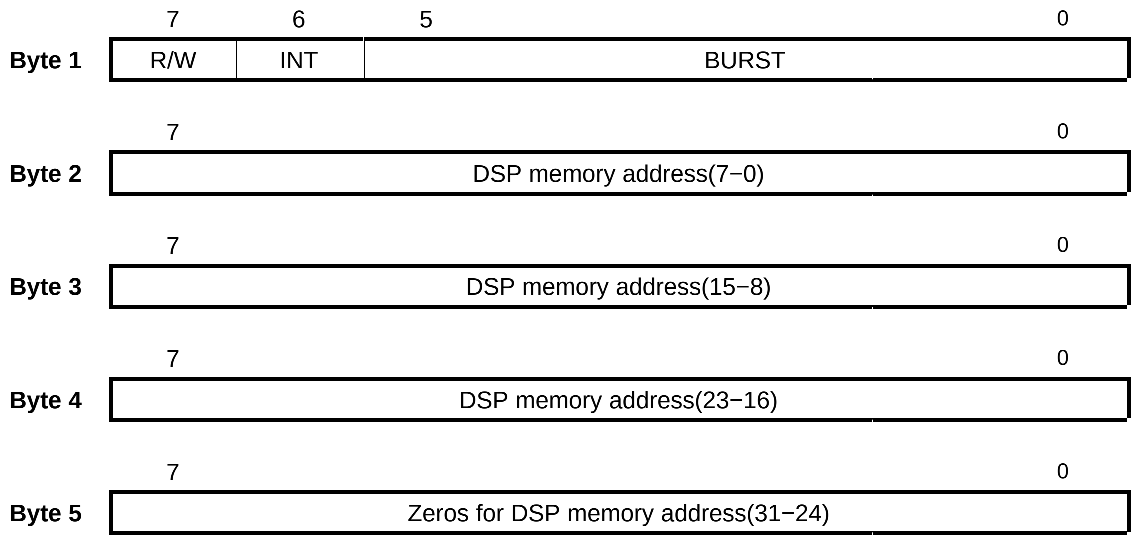


Table 17−7. The Fields of Byte 1 of the Host-DMA Protocol Header

Bit	Field	Value	Function
7	R/W	0	Host write operation: The host wants the USB DMA controller to write data to the DSP memory.
		1	Host read operation: The host wants the USB DMA controller to read data from the DSP memory.
6	INT	0	No host interrupt
		1	Host interrupt requested: The host wants the USB DMA controller to generate a host interrupt (HINT) at the end of the transfer.
5−0	BURST	n =1−59	The host wants n bytes transferred by the USB DMA controller. The host can send as few as one byte (BURST = 1) and as many as 59 bytes (BURST = 59).

17.5.2 Initiating a Host Read Operation (Data to Host)

When the host wants to read data from the DSP memory, the host must initiate two bulk transfers on the USB in the following order:

1) **OUT transfer to program the DMA state machine.** In the first transfer, the host sends a protocol header (5 bytes), which includes a bit to indicate that the host wants to read data. The UBM receives the header from the SIE and then passes the header to the selected OUT endpoint buffer. The USB DMA controller reads the header, configures itself, and then moves the requested data from the memory to the selected IN endpoint buffer.

2) **IN transfer to get the data.** In the second transfer, the host sends a request for the data. The SIE passes up to 64 bytes to the host. The first 5 bytes is a copy of the protocol header that was sent in the previous OUT transfer. The next 1 to 59 bytes are the data the UBM has retrieved from the selected IN endpoint buffer.

17.5.3 Initiating a Host Write Operation (Data to DSP Memory)

To write data to the DSP memory, the host must initiate one bulk transfer: an OUT transfer that sends the 5-byte protocol header followed by up to 59 bytes of data (up to 64 bytes total). After the UBM has passed all the bytes to the selected OUT endpoint buffer, the USB DMA controller interprets the header and then moves the 1 to 59 data bytes from the buffer to the DSP memory.

17.5.4 Configuring the USB Module for the Host-DMA Mode

The following tables and paragraphs explain the steps to follow to prepare the USB module for the host-DMA mode. First, choose two of the general-purpose endpoints to serve as the host-DMA endpoints, and program these two endpoints as bulk endpoints with the double buffer mode enabled. Second, select the endpoints in USBHEPSEL and (if desired) enabled the interrupts for the host-DMA mode. Finally, enable the host-DMA mode.

17.5.4.1 Make Sure the Chosen Endpoints are in the Non-Isochronous Mode

Register(Field)	Value	Description
USBxCNFn(ISO)	0	Non-isochronous mode
	1	Isochronous mode

The host-DMA endpoints must be configured as bulk endpoints, and for bulk transfers, an endpoint must be in the non-isochronous mode. Clear the ISO bit of the appropriate endpoint configuration registers. For example, if your host-DMA endpoints are OUT endpoint 4 and IN endpoint 4, make sure that USBOCNF4(ISO) = 0 and USBICNF4(ISO) = 0. The endpoint configuration registers are described on pages 17-70 (for IN endpoints) and 17-72 (for OUT endpoints).

17.5.4.2 Make Sure the Chosen Endpoints are in the Double Buffer Mode

Register(Field)	Value	Description
USBxCNFn(DBUF)	0	Single buffer mode
	1	Double buffer mode

Whenever the USB DMA controller accesses an endpoint buffer, the controller assumes that the endpoint buffer is equally divided between an X buffer and a Y buffer. To meet this requirement for your host-DMA endpoints, set the DBUF bits in the appropriate endpoint configuration registers. For example, if your host-DMA endpoints are OUT endpoint 1 and IN endpoint 3, make sure that USBOCNF1(DBUF) = 1 and USBICNF3(DBUF) = 1.

17.5.4.3 Set the Maximum Packet Size for the Chosen Endpoints (64 Bytes Recommended)

Register(Field)	Value	Description
USBxSIZn(SIZ)	0−64	Size for buffer X/Y (maximum packet size)

In the double buffer mode (required for the USB DMA controller), the USB DMA controller accesses one of the buffers at a time (X or Y), based on the data toggle sequence (DATA0, DATA1, DATA0, and so on). The 7-bit SIZ field in USBxSIZn determines the maximum packet size: the number of bytes the active buffer (X or Y) can hold at one time. Because the host-DMA mode allows transfers up to 64 bytes long, it is recommended that you load 64 into SIZ for each of the host-DMA endpoints. If you want more than 59 data bytes transferred, send them in multiple host-DMA transfers. The fields of the buffer size register are described in section 17.8.3.5 (page 17-78).

17.5.4.4 Select the Chosen Endpoints in USBHEPSEL

Register(Field)	Value	Description
USBHEPSEL(IEP)	1−7	Indicates which IN endpoint will be used as the host-DMA endpoint for IN transfers.
USBHEPSEL(OEP)	1−7	Indicates which OUT endpoint will be used as the host-DMA endpoint for OUT transfers.

With USBHEPSEL you must select one of the OUT endpoints and one of the IN endpoints to be used for host-initiated DMA transfers. Once you select the host-DMA endpoints in this register, these two endpoints cannot be used for any other purpose until you initiate a USB software reset (write 1 to the SOFTRST bit of USBGCTL) or a DSP reset (drive the $\overline{\text{RESET}}$ pin low). The fields of USBHEPSEL are described in section 17.8.6.2 (page 17-94).

17.5.4.5 Enable or Disable the Interrupts Associated with the Host-DMA Mode

Register(Field)	Value	Description
USBHCTL(HIE)	0	Disable host interrupt requests.
	1	Enable host interrupt requests.
USBHCTL(HERRIE)	0	Disable host error interrupt requests.
	1	Enable host error interrupt requests.
USBHSTAT(HIF)	0	Host-DMA transfer not done
	1	Host-DMA transfer done. The USB DMA controller has completed a DMA transfer between the DSP memory and a host-DMA endpoint. This flag is set only if the host requests the interrupt by setting the INT bit in the first byte of the protocol header.
USBHSTAT(HERRIF)	0	No host error
	1	A host error has occurred: During an OUT transfer, the size of the data transferred from the host did not match the size specified in the protocol header.

If you want the host to be able to generate an interrupt request at the end of a host-DMA transfer, set the host interrupt enable (HIE) bit of the host control register (USBHCTL). Likewise, if you want the CPU to be notified of a host transfer error, set the host error interrupt enable (HERRIE) bit of USBHCTL.

The host status register (USBHSTAT) has flags that the CPU can poll regardless of whether the interrupts are enabled. The HIF bit indicates that the USB DMA controller has completed a host-DMA transfer and the host requested an interrupt at the end of the DMA transfer (in the protocol header). The HERRIF bit indicates that a host error has occurred.

17.5.4.6 Enable the Host-DMA Mode

Register(Field)	Value	Description
USBHCTL(EN)	0	Disable host-DMA mode.
	1	Enable host-DMA mode (allow the host to initiate DMA transfers at the endpoints selected in USBHEPSEL).
USBHSTAT(DIS)	0	Host-DMA mode enabled.
	1	Host-DMA mode disabled.

A reset clears the EN bit of USBHCTL. Set the EN bit to enable the host-DMA mode. To verify that the mode is on, you can check the DIS bit of USBHSTAT. If DIS = 1, the mode is disabled. The fields of USBHCTL and USBHSTAT are described in on pages 17-93 and 17-94, respectively.

17.6 Interrupt Activity in the USB Module

The interrupt requests generated by the USB module can be grouped into the following main categories:

- ❑ USB bus interrupt requests (see section 17.6.1 on page 17-47)
- ❑ Endpoint interrupt requests (see section 17.6.2 on page 17-48)
- ❑ USB DMA interrupt requests (see section 17.6.3 on page 17-50)
- ❑ Host-DMA mode interrupt requests (see section 17.6.4 on page 17-52)

As shown in Figure 17−12, all requests are multiplexed through an arbiter to a single USB interrupt request for the CPU. When the arbiter receives multiple interrupt requests at the same time, it services them one at a time according to a predefined priority ranking. The priority of each request is included in the description of the interrupt source register (USBINTSRC) in section 17.8.5.1 (page 17-85).

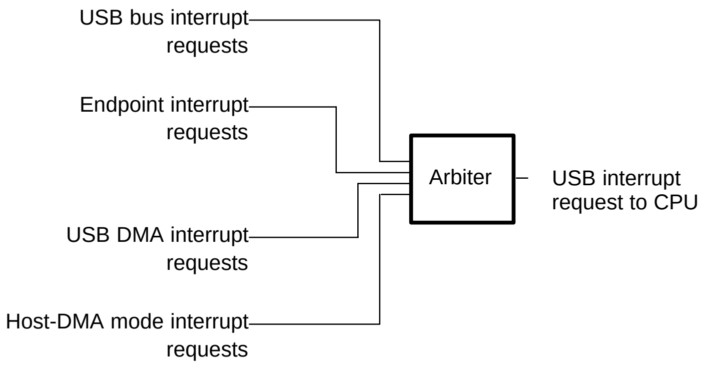
The USB interrupt is one of the maskable interrupts of the CPU. As with any maskable interrupt request, if it is properly enabled in the CPU, the CPU executes the corresponding interrupt service routine (ISR). The ISR for the USB interrupt can determine the interrupt source by reading the interrupt source register. Then the ISR can branch to the appropriate subroutine.

After the CPU reads USBINTSRC, the following events occur:

1) The interrupt flag for the source interrupt is cleared in the corresponding interrupt flag register. *Exception:* The STPOW and SETUP bits in USBIF are not cleared when USBINTSRC is read. To clear one of these bits, write a 1 to it.

2) The arbiter determines which of the remaining interrupt requests has the highest priority, writes the code for that interrupt to USBINTSRC, and forwards the interrupt request to the CPU.

Figure 17−12. Possible Sources of a USB Interrupt Request


USB bus interrupt requests
Endpoint interrupt requests
USB DMA interrupt requests
Host-DMA mode interrupt requests
Arbiter
USB interrupt request to CPU

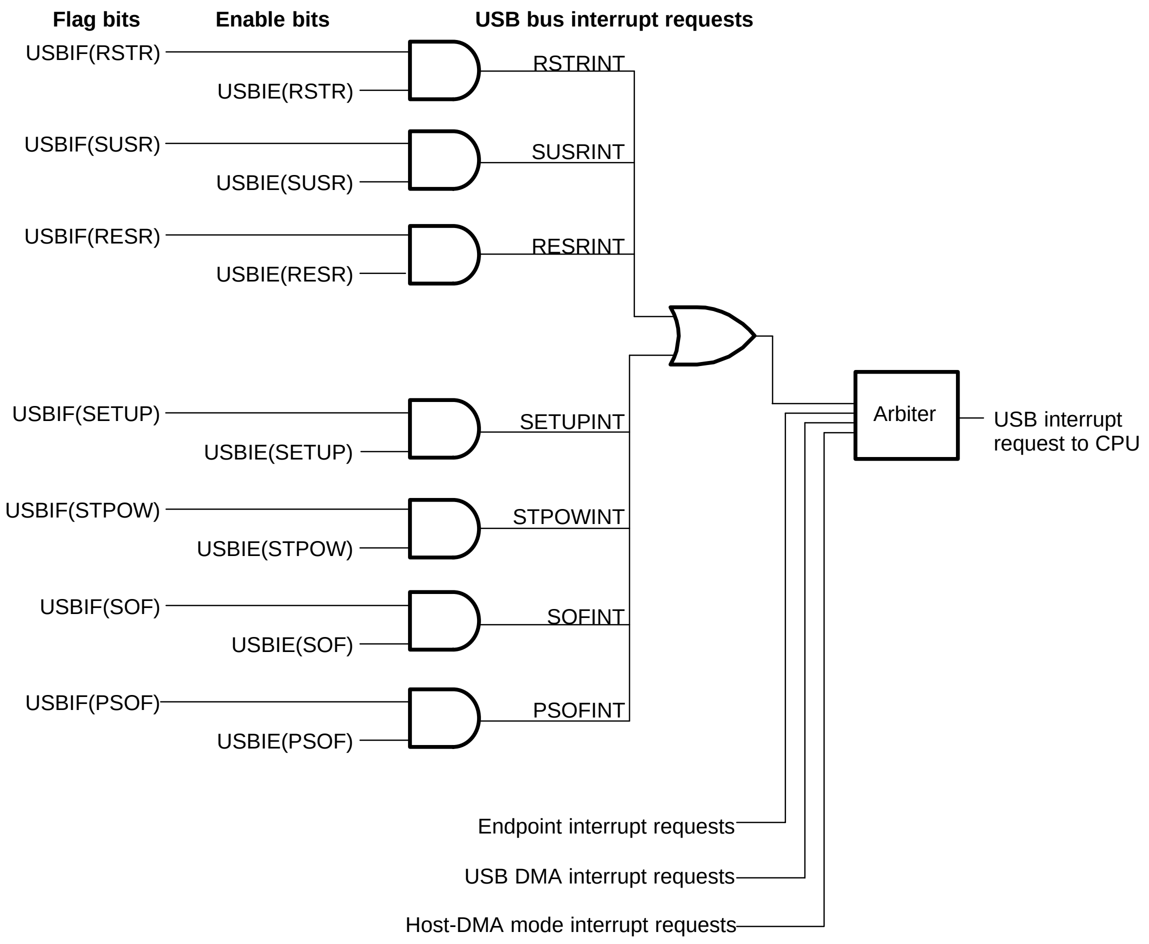
17.6.1 USB Bus Interrupt Requests

The USB module can generate a number of interrupt requests that are related to activity on the USB (see Table 17−8). As shown in Figure 17−13, each of the interrupt requests has a flag bit in the USB interrupt flag register (USBIF) and an enable bit in the USB interrupt enable register (USBIE). When one of the specified events occurs, its flag bit is set. If the corresponding enable bit is 0, the the interrupt request is blocked. If the enable bit is 1, the request is forwarded to the CPU as a USB interrupt.

Table 17−8. Descriptions of the USB Bus Interrupt Requests

USB Bus Interrupt Request	**Interrupt Source**
RSTRINT	A reset condition is detected on the USB.
SUSRINT	A suspend condition is detected on the USB.
RESRINT	Activity on the USB resumes, ending a suspend condition.
SETUPINT	A setup packet arrived. (Setup data is stored in the setup packet buffer.)
STPOWINT	A setup overwrite has occurred; that is, a new setup packet arrived before the previous setup packet was read from the setup packet buffer.
SOFINT	A start-of-frame (SOF) packet is detected on the USB.
PSOFINT	The pre-SOF (PSOF) timer has finished counting down. If you want an interrupt to occur n (1 to 255) clock cycles before each SOF packet, load n into the pre-SOF interrupt timer count register, USBPSOFTMR (see section 17.8.7.3 on page 17-96). The counter runs at 750 kHz (12MHz/16).

Figure 17−13. Enable Paths of USB Bus Interrupt Requests



17.6.2 Endpoint Interrupt Requests

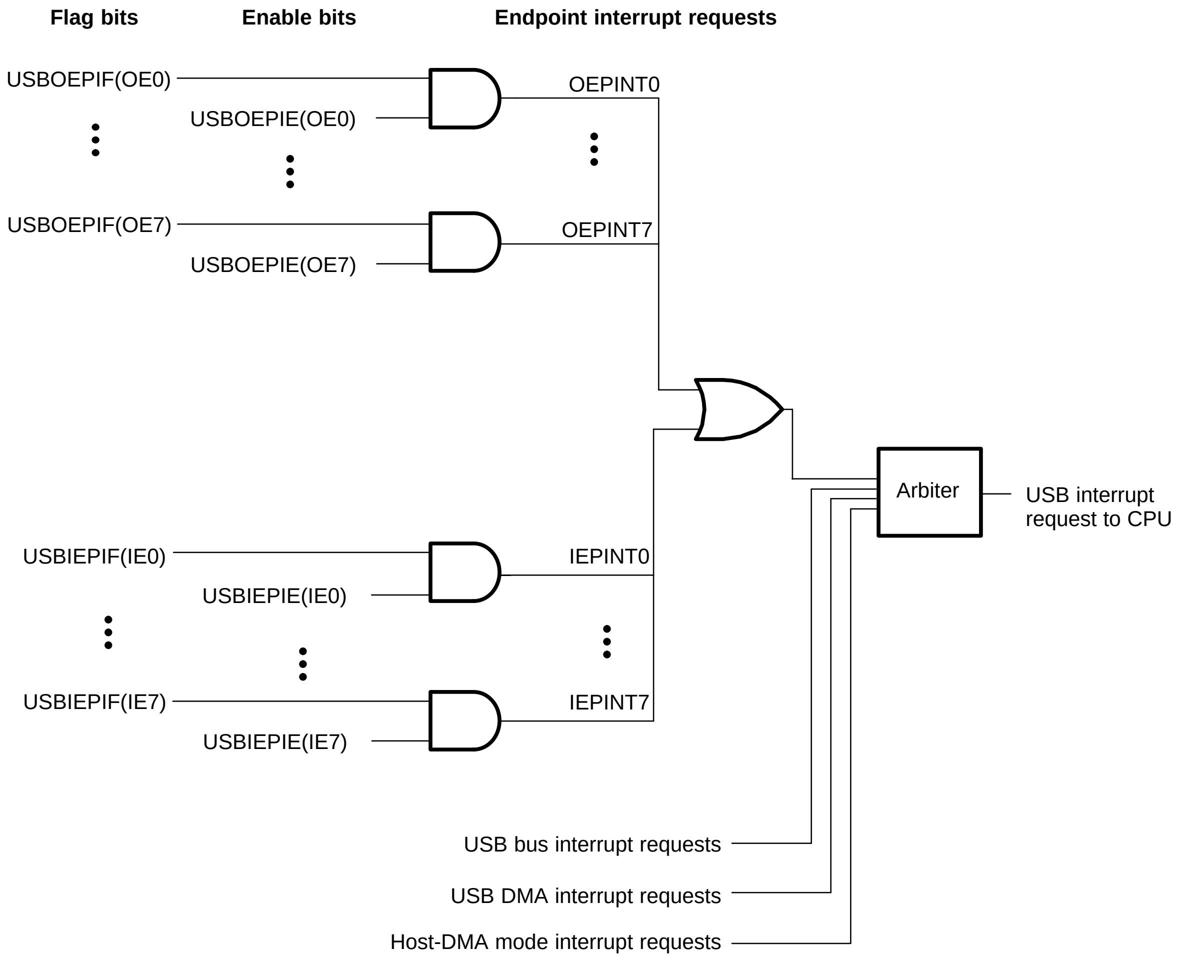
For each endpoint, the UBM can generate an interrupt request every time data moves in or out of the endpoint buffer. As shown in Figure 17−14:

- ❑ Each OUT endpoint has a flag bit in the OUT endpoint interrupt flag register (USBOEPIF) and an enable bit in the OUT endpoint interrupt enable register (USBOEPIE).
- ❑ Each IN endpoint has a flag bit in the IN endpoint interrupt flag register (USBIEPIF) and an enable bit in the IN endpoint interrupt enable register (USBIEPIE).
- ❑ For either type of endpoint, when both the flag bit and the enable bit are set, an interrupt request is passed to the CPU.

Software must set the enable bit, but the flag bit is set by the UBM when a specific event occurs:

- **For an OUT endpoint (data from host):** When the UBM receives a valid data packet, it writes the data to the appropriate OUT endpoint buffer. The UBM then sets the NAK bit of the endpoint's count register, to keep the host from writing to the buffer before the data is read. When the NAK bit is set, the associated interrupt flag bit is also set.
- **For an IN endpoint (data to host):** When the USB module receives an IN packet, the UBM reads the data from the appropriate IN endpoint buffer. Then the UBM sets the NAK bit of the endpoint's count register, to keep the host from reading again before new data is placed in the buffer. When the NAK bit is set, the associated interrupt flag bit is also set.

Figure 17−14. Enable Paths for the Endpoint Interrupt Requests



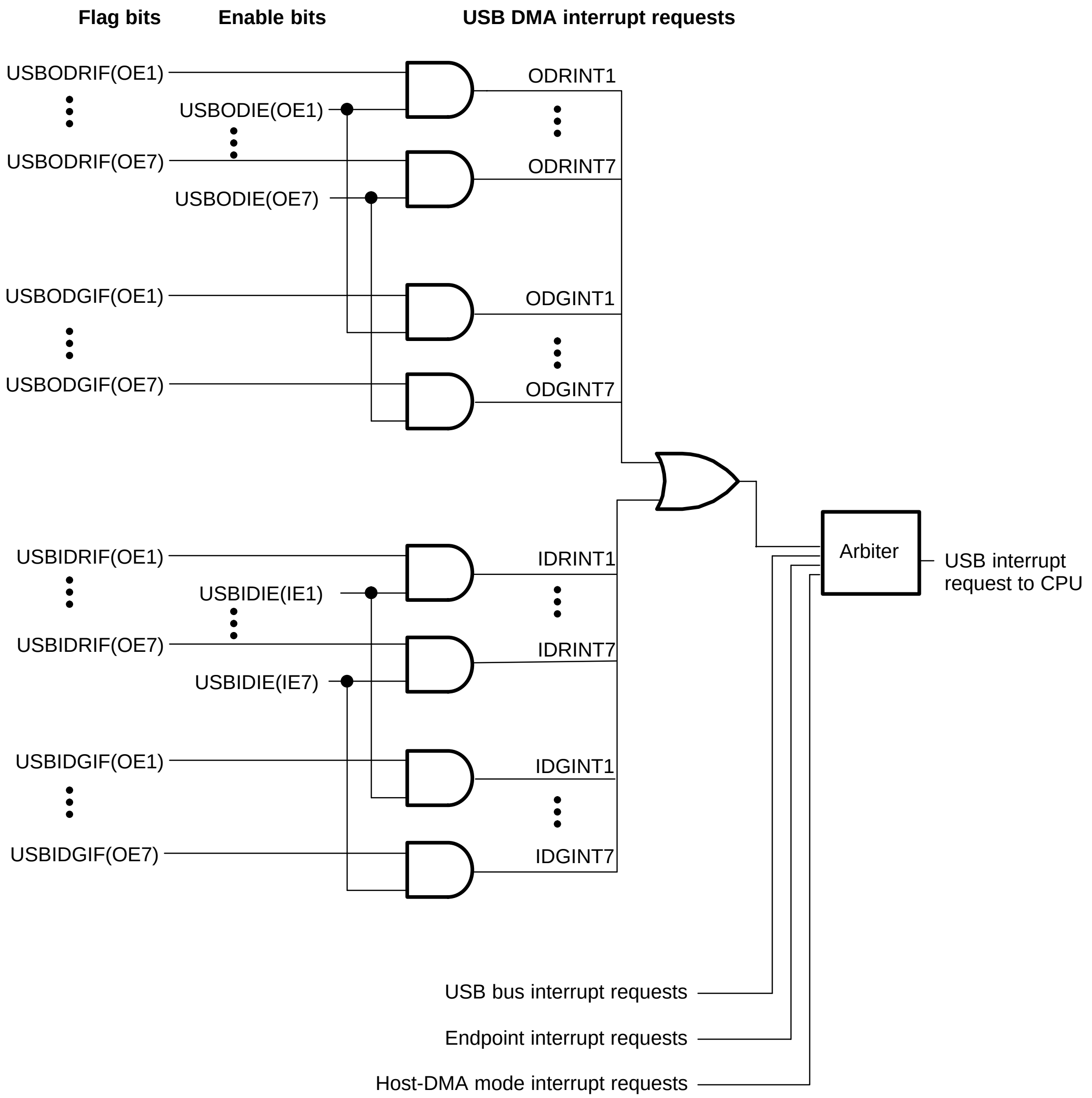
17.6.3 USB DMA Interrupt Requests

The USB module can generate an interrupt request every time the USB DMA controller clears the GO bit or RLD (reload) bit for one of the general-purpose endpoints (OUT endpoints 1−7 and IN endpoints 1−7). As shown in Figure 17−15:

- Each OUT endpoint has:
  - One flag bit in the OUT endpoint DMA GO interrupt flag register (USBODGIF).
  - Another flag bit in the OUT endpoint DMA RLD interrupt flag register (USBODRIF).
  - A single enable bit in the OUT endpoint DMA interrupt enable register (USBODIE). This bit enables or disables both GO and RLD interrupt requests.
- Each IN endpoint has:
  - One flag bit in the IN endpoint DMA GO interrupt flag register (USBIDGIF)
  - Another flag bit in the IN endpoint DMA RLD interrupt flag register (USBIDRIF)
  - A single enable bit in the OUT endpoint DMA interrupt enable register (USBIDIE). This bit enables or disables both GO and RLD interrupt requests.
- For either type of endpoint, when either or both of the flag bits are set and the enable bit is set, an interrupt request is passed to the CPU.

Software must set the enable bit, but the flag bit is set by the DMA controller for a specific event. To understand how the GO and RLD bits are used to control DMA activity, see section 17.4.3 on page 17-14.

Figure 17−15.   Enable Paths for the USB DMA Interrupt Requests

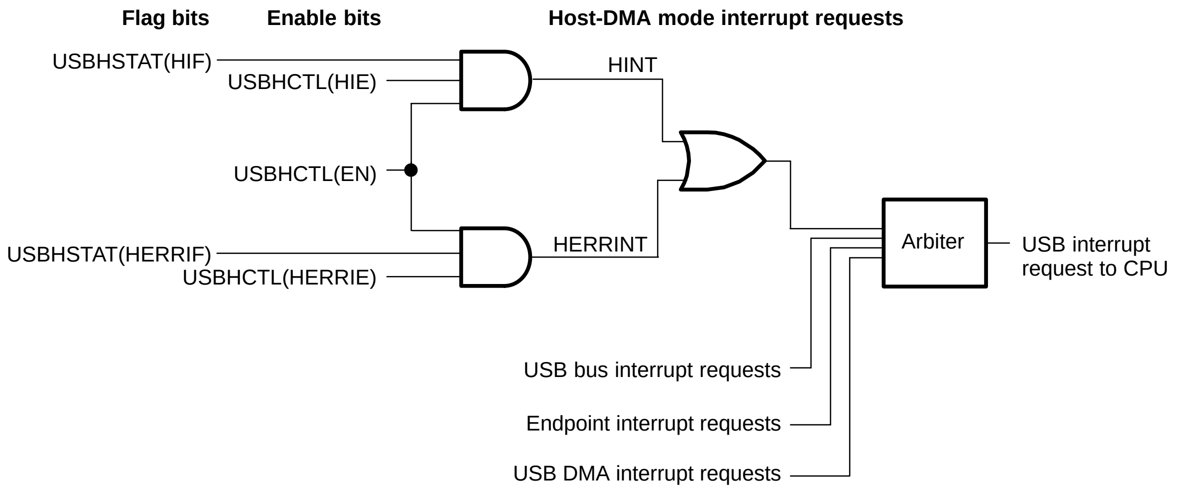


17.6.4 Host-DMA Mode Interrupt Requests

If the host-DMA mode (see section 17.5 on page 17-40) is enabled, a special state machine in the USB DMA controller can generate the following types of interrupts. The flag bits, enable bits, and interrupt requests are shown in Figure 17−16. Notice these interrupt requests are dependent on the EN bit in USBHCTL; when EN = 1, the host-DMA mode is enabled.

- **Host interrupt (HINT).** If the USB DMA controller completes a transfer between the USB host and the DSP memory (via an endpoint buffer) and the host requested an interrupt at the end of the transfer, the state machine sets the HIF flag bit in the host status register (USBHSTAT). If the HIE enable bit is 1 in the host control register (USBHCTL), an interrupt request is passed to the CPU.
- **Host error interrupt (HERRINT).** During an OUT transfer, if the size of the data transferred from the host does not match the size specified in the protocol header, the state machine sets the HERRIF flag bit in USBHSTAT. If the HERRIE enable bit is 1 in USBHCTL, an interrupt request goes to the CPU.

Figure 17−16. Enable Paths for the Host-DMA Mode Interrupt Requests


Flag bits
Enable bits
Host-DMA mode interrupt requests
USBHSTAT(HIF)
USBHCTL(HIE)
HINT
USBHCTL(EN)
USBHSTAT(HERRIF)
USBHCTL(HERRIE)
HERRINT
Arbiter
USB interrupt request to CPU
USB bus interrupt requests
Endpoint interrupt requests
USB DMA interrupt requests

17.7 Power, Emulation, and Reset Considerations

This section is a summary of the effects of power control, emulation, and reset operations on the USB module.

17.7.1 Putting the USB Module into Its Idle Mode

The USB module is one of the peripheral devices in the PERIPH idle domain. For details on controlling the various idle domains of the DSP, see Chapter 8, *Idle Configurations*. If you want the USB module to become idle in response to an IDLE instruction, make the following preparations:

1) Write 1 to the idle enable (IDLEEN) bit in USBIDLECTL (see section 17.8.7.8 on page 17-102). This tells the DSP to make the USB module idle when the PERIPH domain becomes idle.

2) Write a 1 to the PERI bit in ICR (see section 8.7 on page 8-9). This tells the DSP to make the PERIPH domain idle in response to an IDLE instruction.

17.7.2 USB Module Indirectly Affected By Certain Idle Configurations

As mentioned in section 17.7.1, the USB module can be affected by any idle configuration that turns off the PERIPH idle domain. In addition, activity in the USB module can be affected by other idle configurations. For example:

❑ Idle configurations that turn off the CPU, preventing the CPU from controlling and monitoring USB activity. (If enabled, an interrupt from the USB module will wake the CPU.)

❑ Idle configurations that turn off the DSP DMA controller, preventing the USB module from accessing the DSP memory

❑ Idle configurations that turn off the EMIF, preventing the DSP DMA controller from accessing external memory

For more details about idle configurations, see Chapter 8.

17.7.3 USB Module During Emulation

During emulation, the USB module is not halted by a breakpoint. However, the CPU is halted and, therefore, unable to respond to USB interrupts or other requests.

In addition, the USB DMA controller cannot access memory if the DSP DMA controller is programmed to halt when a breakpoint is encountered in the debugger software. The FREE bit of DMA_GCR controls the emulation behavior of the DSP DMA controller (see Chapter 4). If FREE = 0 (the reset value), a breakpoint suspends DMA transfers. If FREE = 1, DMA transfers are not interrupted by a breakpoint.

17.7.4 Resetting the USB Module

There are three ways to reset the USB module:

- Write 1 to the USB software reset bit (SOFTRST) in the USB global control register (USBGCTL). This resets the USB module but does not hold it in reset. Immediately after the reset operation, the USB module is free to run. The reset operation disconnects the USB module from the bus. **Note:** The reset triggered by setting the SOFTRST bit does not affect the USB control register (USBCTL).

- Write 0 to the USB reset bit (USBRST) in the USB idle control register (USBIDLECTL). This resets the USB module and holds it in reset until you write 1 to USBRST. During the reset operation, all of the USB module registers assume their power-on default values (shown in the register figures of section 17.8). One important effect is that the USB module is disconnected from the USB (CONN = 0 in USBCTL).

- Initiate a DSP reset by driving the $\overline{\text{RESET}}$ pin low. The entire DSP is reset and is held in the reset state until you drive the pin high. When all DSP registers assume their reset values, the USBRST bit is forced to 0, which causes a USB reset.

17.8 USB Module Registers

This section covers the following topics:

Topic	See ...
High-level summary of the USB registers	Section 17.8.1
DMA context registers	Section 17.8.2 on page 17-59
Descriptor registers for IN and OUT Endpoints 1−7	Section 17.8.3 on page 17-68
Descriptor registers for IN and OUT Endpoints 0	Section 17.8.4 on page 17-82
Interrupt registers	Section 17.8.5 on page 17-85
Host-DMA mode registers	Section 17.8.6 on page 17-93
General control and status registers	Section 17.8.7 on page 17-95

17.8.1 High-Level Summary of USB Module Registers

Table 17−9 lists the registers that are part of the USB module. These registers are in the I/O space of the C55x DSP. There are two additional registers that also reside in I/O space but are not part of the USB module:

❑ USBCLKMD, which controls the operation of the the dedicated USB clock generator (see section 17.2.3 on page 17-9)

❑ USBIDLECTL, which contains bits to put the USB module into its idle mode or into reset (see section 17.8.7.8 on page 17-102)

On each C55x DSP that contains a USB module, the set of USB module registers may start at a different base address, but the individual registers are at the same offset from the base address. To form a register's address, add the base address and the offset shown in the first column of Table 17−9. For example, the base address on a TMS320VC5509 DSP is 5800h, and the OUT endpoint 1 DMA context registers begin at I/O address 5808h.

Table 17−9. High-Level Summary of the USB Module Registers

I/O Address (Word Address)	Number of Registers	Width (Bits)	Description
Base address + 0000h			Reserved
Base address + 0008h	8	16	OUT endpoint 1 DMA context block
Base address + 0010h	8	16	OUT endpoint 2 DMA context block
Base address + 0018h	8	16	OUT endpoint 3 DMA context block
Base address + 0020h	8	16	OUT endpoint 4 DMA context block
Base address + 0028h	8	16	OUT endpoint 5 DMA context block
Base address + 0030h	8	16	OUT endpoint 6 DMA context block
Base address + 0038h	8	16	OUT endpoint 7 DMA context block
Base address + 0040h			Reserved
Base address + 0048h	8	16	IN endpoint 1 DMA context block
Base address + 0050h	8	16	IN endpoint 2 DMA context block
Base address + 0058h	8	16	IN endpoint 3 DMA context block
Base address + 0060h	8	16	IN endpoint 4 DMA context block
Base address + 0068h	8	16	IN endpoint 5 DMA context block
Base address + 0070h	8	16	IN endpoint 6 DMA context block
Base address + 0078h	8	16	IN endpoint 7 DMA context block

Table 17−9. High-Level Summary of the USB Module Registers (Continued)

I/O Address (Word Address)	Number of Registers	Width (Bits)	Description
Base address + 0080h	3584	8	Space for X and Y data buffers for OUT endpoints 1−7 and IN endpoints 1−7
Base address + 0E80h	64	8	OUT endpoint 0 buffer
Base address + 0EC0h	64	8	IN endpoint 0 buffer
Base address + 0F00h	8	8	Setup packet buffer
Base address + 0F08h	8	8	OUT endpoint 1 descriptor block
Base address + 0F10h	8	8	OUT endpoint 2 descriptor block
Base address + 0F18h	8	8	OUT endpoint 3 descriptor block
Base address + 0F20h	8	8	OUT endpoint 4 descriptor block
Base address + 0F28h	8	8	OUT endpoint 5 descriptor block
Base address + 0F30h	8	8	OUT endpoint 6 descriptor block
Base address + 0F38h	8	8	OUT endpoint 7 descriptor block
Base address + 0F40h			Reserved
Base address + 0F48h	8	8	IN endpoint 1 descriptor block
Base address + 0F50h	8	8	IN endpoint 2 descriptor block
Base address + 0F58h	8	8	IN endpoint 3 descriptor block
Base address + 0F60h	8	8	IN endpoint 4 descriptor block
Base address + 0F68h	8	8	IN endpoint 5 descriptor block
Base address + 0F70h	8	8	IN endpoint 6 descriptor block
Base address + 0F78h	8	8	IN endpoint 7 descriptor block

Table 17−9. High-Level Summary of the USB Module Registers (Continued)

I/O Address (Word Address)	Number of Registers	Width (Bits)	Description
Base address + 0F80h	1		IN endpoint 0 configuration register
Base address + 0F81h	1	8	IN endpoint 0 count register
Base address + 0F82h	1	8	OUT endpoint 0 configuration register
Base address + 0F83h	1	8	OUT endpoint 0 count register
Base address + 0F84h			Reserved
Base address + 0F91h	1	8	Global control register
Base address + 0F92h	1	8	Interrupt source register
Base address + 0F93h	1	8	Endpoint interrupt flag register for IN endpoints
Base address + 0F94h	1	8	Endpoint interrupt enable register for OUT endpoints
Base address + 0F95h	1	8	DMA RLD (reload) interrupt flag register for IN endpoints
Base address + 0F96h	1	8	DMA RLD interrupt flag register for OUT endpoints
Base address + 0F97h	1	8	DMA GO interrupt flag register for IN endpoints
Base address + 0F98h	1	8	DMA GO interrupt flag register for OUT endpoints
Base address + 0F99h	1	8	DMA interrupt enable register for IN endpoints
Base address + 0F9Ah	1	8	DMA interrupt enable register for OUT endpoints
Base address + 0F9Bh	1	8	Endpoint interrupt enable register for IN endpoints
Base address + 0F9Ch	1	8	Endpoint interrupt enable register for OUT endpoints
Base address + 0F9Dh			Reserved
Base address + 0FA0h	1	8	Host control register
Base address + 0FA1h	1	8	Host endpoint select register
Base address + 0FA2h	1	8	Host status register
Base address + 0FA3h			Reserved

Table 17−9. High-Level Summary of the USB Module Registers (Continued)

I/O Address (Word Address)	Number of Registers	Width (Bits)	Description
Base address + 0FF8h	1	8	Frame number register, low part
Base address + 0FF9h	1	8	Frame number register, high part
Base address + 0FFAh	1	8	Pre-SOF interrupt timer register
Base address + 0FFBh			Reserved
Base address + 0FFCh	1	8	USB control register
Base address + 0FFDh	1	8	USB interrupt enable register
Base address + 0FFEh	1	8	USB interrupt flag register
Base address + 0FFFh	1	8	USB device address register

17.8.2 DMA Context Registers

Each of the general-purpose endpoints (OUT endpoints 1−7 and IN endpoints 1−7) has a dedicated DMA channel and a dedicated block of DMA context registers for controlling and monitoring transfer activities in that channel. This section describes the function of each of the context registers, which are summarized in Table 17−10.

For each endpoint, the block of DMA context registers starts at a different base address, but the individual registers are at the same offset from the base address. The first column of Table 17−10 shows the offsets.

Table 17−10. USB DMA Context Registers for Each OUT or IN Endpoint n (n = 1, 2, 3, 4, 5, 6, or 7)

Offset From Context Block's Base Address (Words)	**USB DMA Context Register**		**Description**
	OUT endpoint n	**IN endpoint n**	
0	USBODCTLn	USBIDCTLn	Control register
1	USBODSIZn	USBIDSIZn	Size register (transfer size in bytes)
2	USBODADLn	USBIDADLn	Address register, low part (byte address for a location in DSP memory)
3	USBODADHn	USBIDADHn	Address register, high part (byte address for a location in DSP memory)
4	USBODCTn	USBIDCTn	Count register (transfer count in bytes)
5	USBODRSZn	USBIDRSZn	Reload-size register (reload value for USBxDSIZn, x = O or I)
6	USBODRALn	USBIDRALn	Reload-address register, low part (reload value for USBxDADLn, x = O or I)
7	USBODRAHn	USBIDRAHn	Reload-address register, high part (reload value for USBxDADHn, x = O or I)

17.8.2.1 USB DMA Control Register (USBxDCTLn) (x = O or I; n = 1, 2, 3, 4, 5, 6, or 7)

This register controls the operation of the endpoint DMA channel. The control bits in this register affect the DMA state changes described in section 17.4.9 (page 17-26).

The state of bits 4−6 is captured when a 1 is written to the GO bit and is not captured during the DMA transfer. When the DMA transfer completes, if the RLD bit is set, the state of bits 4−6 is captured again and a new transfer is started. If the RLD bit is set when a DMA transfer ends, the captured USBxDADLn, USBxDADHn, and USBxDSIZn values are swapped with the values stored in USBxDRALn, USBxDRAHn, and USBxDRSZn, respectively.

Figure 17−17. USB DMA Control Register (USBxDCTLn)

USBxDCTLn

15−9	8
Reserved	PM
	R−U

7	6	5	4	3	2	1	0
EM	SHT	CAT	END	OVF	RLD	STP	GO
R/W−U	R/W−U	R/W−U	R/W−U	R/W1C−0	R/W−0	R/W−0	R/W−0

Legend:

R	Read-only access
R/W	Read access/Write access
R/W1C	Read access/Write 1 to clear this bit
−X	X is the value after a DSP reset (X = U indicates that the DSP reset value is undefined).

Table 17−11. USBxDCTLn Bit Descriptions

Bit	Field	Value	Description
15−9	Reserved		These bits are not available for use.
8	PM		Previous packet missing. This status bit indicates that a packet did not occur during the previous frame. The software should consider this bit as a don’t care when EM=0.
		0	A packet did occur on the previous frame for this endpoint.
		1	A packet did NOT occur on the previous frame for this endpoint.
7	EM		Error on missing packet. This control bit determines if, during an isochronous transfer, missing a packet during a frame should be considered an error condition.
		0	Missing packets are treated the same as zero-length packets.
		1	Missing packets will cause the GO bit to be cleared and the DMA to be halted. The error status will show in the PM bit. This event will only occur when PM goes from 0 to 1.

Table 17−11. USBxDCTLn Bit Descriptions (Continued)

Bit	Field	Value	Description
6	SHT		Short packet control. This bit only takes effect on a start or reload condition.
			For IN transfers:
		0	If the size of the last packet in the transfer matches the maximum packet size, do not insert an additional, 0-byte packet.
		1	If the size of the last packet in the transfer matches the maximum packet size, insert a zero-length (0-byte) packet to terminate the transaction with a short packet.
			For OUT transfers:
		0	If the size of the last packet in the transfer matches the maximum packet size, do not wait for a 0-byte packet to indicate the end of the transfer.
		1	If the size of the last packet in the transfer matches the maximum packet size, wait for an additional, 0-byte packet as an indication of the end of the transfer.
5	CAT		Concatenation control. This bit only takes effect on a start or reload condition.
			For IN transfers:
		0	If the transfer size is not enough to fill a maximum-size packet, allow the USB module to transfer a short packet.
		1	Concatenate DMA transfers. If the transfer size is not enough to fill a packet, then perform the next DMA transfer to fill the packet before allowing the USB module to send the data out.
			For OUT transfers:
		0	If the packet size exceeds the number of bytes remaining in the DMA transfer, record an overflow in the OVF bit.
		1	Concatenate DMA transfers. If the packet size exceeds the number of bytes remaining in the DMA transfer, do not record an overflow. Instead, record the current position in the buffer. When the next DMA transfer starts, read the rest of the data, beginning at the recorded position.
4	END		Endianness (byte orientation). This bit only takes effect on a start or reload condition.
		0	Little Endian (first byte is least significant byte in word)
		1	Big Endian (first byte is most significant byte in word)

Table 17−11. USBxDCTLn Bit Descriptions (Continued)

Bit	Field	Value	Description
3	OVF		Overflow/Underflow. For conditions that set this flag, see the state tables in section 17.4.9 (page 17-26).
			For isochronous IN transfers:
		0	Read – No underflow condition
		1	Read – Underflow condition Write – Write 1 to clear this flag.
			For isochronous and non-isochronous OUT transfers:
		0	Read – No overflow condition
		1	Read – Overflow condition Write – Write 1 to clear this flag.
2	RLD		Reload control. User writes a 1 to reload the address and size registers from reload registers when there are no pending transfers. The current address and size are automatically swapped with the reload address and size
		0	Do not use the reload registers.
		1	Reload and swap the address and size registers.
1	STP		Stop DMA transfer. This bit stops the DMA transfer on the next packet boundary. The data in the UBM Buffer is not affected.
		0	DMA functions normally
		1	DMA stops on the next packet boundary or at the end of the current DMA transfer (whichever occurs first), without clearing the UBM state or sending the current packet. At this time the GO and STP bits are reset.

Table 17−11. USBxDCTLn Bit Descriptions (Continued)

Bit	Field	Value	Description
0	GO		Start DMA transfer. When written with 1, this bit starts the DMA transfer for the endpoint. Writes of 0 have no effect. GO is cleared when a DMA transfer is no longer active.
		0	The USB DMA controller is idling (the controller is available for a new transfer).
		1	Read 1 − The USB DMA controller is performing a transfer. Write 1 − Start the endpoint DMA transfer (STP bit must be 0).

17.8.2.2 USB DMA Address Registers (USBxDADHn and USBxDADLn) (x = O or I; n = 1, 2, 3, 4, 5, 6, or 7)

The USB DMA controller handles transfers between an endpoint buffer and the DSP memory. Because each of the USB DMA channels is dedicated to a particular endpoint, the start address for the endpoint buffer is known. Software must supply only a start address for the DSP memory.

The start address must be a **byte address**. Load the high 8 bits of the byte address to USBxDADHn and the low 16 bits to USBxDADLn. The DMA controller concatenates the two values to form a 24-bit address:

DSP memory address: DADH:DADL

In addition the address must be **16-bit aligned**. Make sure that the least significant bit (LSB) is 0.

The DMA starts an OUT transfer from (DADH:DADL) + 2 and continues to (DADH:DADL) + DSIZ + 2 or the transfer is otherwise terminated (for example, by a stop command via the STP bit or by a short OUT packet). The 16 bit word at (DADH:DADL) is then updated with the count of bytes actually transferred. The byte order of this word is architecture dependent and not dependent on the state of the END bit.

IN transfers start from (DADH:DADL) and continue to (DADH:DADL) + DSIZ.

Figure 17−18. USB DMA Address Registers (USBxDADLn and USBxDADHn)

	15−0
USBxDADLn	DADL (byte address)
USBxDADHn	DADH (byte address)
	R/W−U

Legend:

R/W	Read/write access
−U	The contents of the registers is undefined after a DSP reset.

Table 17−12. USBxDADLn and USBxDADHn Bit Descriptions

Bit	Field	Value	Description
USBxDADLn(15−0)	DADL	0000h−FFFFh	Low part of the DSP memory start address. The start address must be 16-bit aligned; therefore, make sure bit 0 of this register is 0.
USBxDADHn(7−0)	DADH	0000h−00FFh	High part of the DSP memory start address. C55x DSP memory addresses have 24 bits; therefore, load the 8 high bits of DADH with 0s.

17.8.2.3 USB DMA Size Register (USBxDSIZn) (x = O or I; n = 1, 2, 3, 4, 5, 6, or 7)

USBxDSIZn specifies the number of bytes for the DMA controller to transfer in a single DMA transfer. During a DMA reload operation (see section 17.4.5 on page 17-18), the content of USBxDSIZn is swapped with the content of USBxDRSZn. USBxDRSZn is described in section 17.8.2.6 (page 17-68).

Figure 17−19. DMA Size Register (USBxDSIZn)

	15−0
USBxDSIZn	DSIZ (bytes)
	R/W−U

Legend:

R/W	Read/write access
−U	The content of the register is undefined after a DSP reset.

Table 17−13. USBxDSIZn Bit Description

Bit	Field	Value	Description
15−0	DSIZ	1−65535	Number of bytes for the DMA controller to transfer

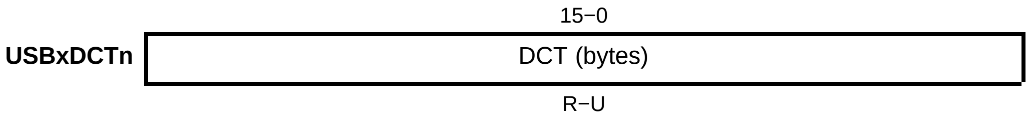
17.8.2.4 DMA Count Register (USBxDCTn) (x = O or I; n = 1, 2, 3, 4, 5, 6, or 7)

USBxDCTn counts up, to track the number of bytes that have been transferred for OUT or IN endpoint n. The USB DMA controller automatically loads this register with 0 before beginning each DMA transfer. This includes the transfer that follows a DMA reload operation.

Note:

When the USB DMA controller stores data from an OUT transfer, it also stores USBxDCTn to the DSP memory (see section 17.4.6 on page 17-19).

Figure 17−20. DMA Count Register (USBxDCTn)



Legend:

R	Read-only register
−U	The content of the register is undefined after a DSP reset.

Table 17−14. USBxDCTn Bit Description

Bit	Field	Value	Description
15−0	DCT	0−65535	Indicates the number of bytes that have been transferred by the USB DMA controller **Note:** Before starting each new DMA transfer, the controller loads this register with 0.

17.8.2.5 USB DMA Reload-Address Registers (USBxDRALn and USBxDRAHn) (x = O or I; n = 1, 2, 3, 4, 5, 6, or 7)

USBxDRALn specifies the low 16 bits of the reload address, and USBxDRAHn specifies the high 8 bits of the reload address. If the RLD bit is set when the current DMA transfer is completed, then the contents of these reload registers are swapped with the contents of their corresponding primary registers:

❑ The content of USBxDRALn is swapped with the content of USBxDADLn.

❑ The content of USBxDRAHn is swapped with the content of USBxDADHn.

This register swapping is part of the DMA reload operation described in section 17.4.5 (page 17-18).

Figure 17−21. DMA Reload-Address Registers (USBxDRALn and USBxDRAHn)

	15−0
USBxDRALn	DRAL (byte address)
USBxDRAHn	DRAH (byte address)
	R/W−U

Legend:

R/W	Read/write access
−U	The contents of the registers is undefined after a DSP reset.

Table 17−15. USBxDRALn and USBxDRAHn Bit Descriptions

Bit	Field	Value	Description
USBxDRALn(15−0)	DRAL	0000h−FFFFh	Reload value for DADL The addresses used by the USB DMA controller must be 16-bit aligned; therefore, make sure bit 0 of this register is 0.
USBxDRAHn(15−0)	DRAH	0000h−00FFh	Reload value for DADH C55x DSP memory addresses have 24 bits; therefore, load the 8 high bits of DRAH with 0s.

17.8.2.6 DMA Reload-Size Register (USBxDRSZn) (x = O or I; n = 1, 2, 3, 4, 5, 6, or 7)

Specifies the reload size. If the RLD bit is set when the current DMA transfer is completed, the content of USBxDRSZn is swapped with the content of USBxDSIZn. This register swapping is part of the DMA reload operation described in section 17.4.5 (page 17-18).

Figure 17−22. DMA Reload-Size Register (USBxDRSZn)

	15−0
USBxDRSZn	DRSZ (bytes)
	R/W−U

Legend:

R/W	Read/write access
−U	The content of the register is undefined after a DSP reset.

Table 17−16. USBxDRSZn Bit Description

Bit	Field	Value	Description
15−0	DRSZ	1−65535	Reload value for DSIZ

17.8.3 Descriptor Registers for IN and OUT Endpoints 1−7

The general-purpose endpoints (IN endpoints 1−7 and OUT endpoints 1−7) each have a block of eight descriptor registers to define the endpoint characteristics for the UBM. Table 17−17 shows the descriptor registers available for an OUT endpoint and for an IN endpoint. To access a descriptor register, find the base address of the descriptor block and add the offset shown in the first column of Table 17−17. The actual addresses can be found in the data manual of the respective C55x DSP.

Table 17−17. Descriptor Registers for Each OUT or IN Endpoint n (n = 1, 2, 3, 4, 5, 6, or 7)

Offset From the Descriptor Block's Base Address (Words)	**Endpoint Descriptor Register**		**Description**
	OUT Endpoint n	**IN Endpoint n**	
0	USBOCNFn	USBICNFn	Endpoint n configuration register
1	USBOBAXn	USBIBAXn	X-buffer base address register (bits 11−4 of a byte address)
2	USBOCTXn	USBICTXn	X-buffer count register (transfer count in bytes)
3	USBOCTXHn	USBISIZHn	**OUT endpoint:** X-buffer count extension register (used for isochronous transfers only) **IN endpoint:** X-/Y-buffer size extension register (used for isochronous transfers only)
4	USBOSIZn	USBISIZn	X-/Y-buffer size register (transfer size in bytes)
5	USBOBAYn	USBIBAYn	Y-buffer base address register (bits 11−4 of a byte address)
6	USBOCTYn	USBICTYn	Y-buffer count register (transfer count in bytes)
7	USBOCTYHn	Reserved	**OUT endpoint:** Y-buffer count extension register (used for isochronous transfers only) **IN endpoint:** Not available for use

17.8.3.1 IN Endpoint n Configuration Register (USBICNFn) (n = 1, 2, 3, 4, 5, 6, or 7)

As shown in Figure 17−23, the function of bits 5−0 of USBICNFn depend on whether you have programmed the endpoint for non-isochronous transfers or for isochronous transfers. Table 17−18 describes the bits of USBICNFn, taking into account the optional function of bits 5−0.

Figure 17−23. IN Endpoint n Configuration Register (USBICNFn)

USBICNFn in Non-Isochronous Mode (ISO = 0)

7	6	5	4	3	2−0
UBME	ISO = 0	TOGGLE	DBUF	STALL	Reserved
R/W−U	R/W−U	R/W−U	R/W−U	R/W−U	

USBICNFn in Isochronous Mode (ISO = 1)

7	6	5−3	2−0
UBME	ISO = 1	CTXH	CTYH
R/W−U	R/W−U	R/W−U	R/W−U

Legend:

R/W	Read access/Write access
−U	The content of these bits is undefined after a DSP reset.

Table 17−18. USBICNFn Bit Descriptions

Bit	Field	Value	Description
7	UBME		UBM access enable
		0	The UBM cannot access this endpoint (the endpoint is inactive).
		1	The UBM can access this endpoint (the endpoint is active).
6	ISO		Isochronous mode enable
		0	Non-isochronous mode
		1	Isochronous mode
Bits 5−0 in Non-Isochronous Mode (ISO = 0)			
5	TOGGLE		Endpoint data toggle. This bit reflects the data toggle sequence (see section 17.1.2 on page 17-4). **Note:** You do not need to write to this bit; it is maintained by the UBM.
		0	The next data packet is DATA0.
		1	The next data packet is DATA1.

Table 17−18. USBICNFn Bit Descriptions (Continued)

Bit	Field	Value	Description
4	DBUF		Double buffer mode enable **Note:** The USB DMA controller requires the double buffer mode. If the DMA controller will be servicing the endpoint, make sure DBUF = 1 before you start the controller.
		0	Single buffer used (X buffer only)
		1	Double buffer mode. The USB DMA controller tracks the data toggle sequence to determine the active buffer. For a DATA0 packet, the controller uses the X buffer; for a DATA1 packet, the controller uses the Y buffer.
3	STALL		Endpoint stall. Set this bit to tell the USB host that the endpoint is stalled.
		0	No stall
		1	Endpoint stalled. A STALL handshake will be initiated in response to host access requests until the STALL bit is cleared.
2−0	Reserved		Write 0s to these bits.
Bits 5−0 in Isochronous Mode (ISO = 1)			
5−3	CTXH	000b−111b	IN endpoint X-buffer byte count − high bits.
2−0	CTYH	000b−111b	IN endpoint Y-buffer byte count − high bits.

17.8.3.2 OUT Endpoint n Configuration Register (USBOCNFn) (n = 1, 2, 3, 4, 5, 6, or 7)

As shown in Figure 17−24, the function of bits 5−0 of USBOCNFn depend on whether you have programmed the endpoint for non-isochronous transfers or for isochronous transfers. Table 17−19 describes the bits of USBOCNFn, taking into account the optional function of bits 5−0.

Figure 17−24. OUT Endpoint n Configuration Register (USBOCNFn)

USBOCNFn in Non-Isochronous Mode (ISO = 0)

7	6	5	4	3	2−0
UBME	ISO = 0	TOGGLE	DBUF	STALL	Reserved
R/W−U	R/W−U	R/W−U	R/W−U	R/W−U	

USBOCNFn in Isochronous Mode (ISO = 1)

7	6	5−3	2−0
UBME	ISO = 1	Reserved	SIZH
R/W−U	R/W−U	R/W−U	R/W−U

Legend:

R/W	Read access/Write access
− U	The content of these bits is undefined after a DSP reset.

Table 17−19. USBOCNFn Bit Descriptions

Bit	Field	Value	Description
7	UBME		UBM access enable
		0	The UBM cannot access this endpoint (the endpoint is inactive).
		1	The UBM can access this endpoint (the endpoint is active).
6	ISO		Isochronous mode enable
		0	Non-isochronous mode
		1	Isochronous mode
Bits 5−0 in Non-Isochronous Mode (ISO = 0)			
5	TOGGLE		Endpoint data toggle. This bit reflects the data toggle sequence (see section 17.1.2 on page 17-4). **Note:** You do not need to write to this bit; it is maintained by the UBM.
		0	The next data packet is DATA0.
		1	The next data packet is DATA1.

Table 17−19. USBOCNFn Bit Descriptions (Continued)

Bit	Field	Value	Description
4	DBUF		Double buffer mode enable **Note:** The USB DMA controller requires the double buffer mode. If the DMA controller will be servicing the endpoint, make sure DBUF = 1 before you start the controller.
		0	Single buffer used (X buffer only)
		1	Double buffer mode. The USB DMA controller tracks the data toggle sequence to determine the active buffer. For a DATA0 packet, the controller uses the X buffer; for a DATA1 packet, the controller uses the Y buffer.
3	STALL		Endpoint stall. Set this bit to tell the USB host that the endpoint is stalled.
		0	No stall
		1	Endpoint stalled. A STALL handshake will be initiated in response to host access requests until the STALL bit is cleared.
2−0	Reserved		These bits are not available for use.
Bits 5−0 in Isochronous Mode (ISO = 1)			
5−3	Reserved		Write 0s to these bits.
2−0	SIZH	000h−111b	OUT endpoint X-/Y-buffer size − high bits

17.8.3.3 Endpoint n Buffer Base Address Registers (USBxBAXn, USBxBAYn) (x = O or I; n = 1, 2, 3, 4, 5, 6, or 7)

Each general-purpose endpoint has two buffer base address registers: one for its X buffer and one for its Y buffer (see Figure 17−25 and Table 17−20). By writing to one of these registers, you provide bits 11−4 of a 12-bit relative address. The USB module adds 0s for the bits 3−0 of the relative address. The address is relative to the start address of the USB module registers.

Consider Example 17−2, which follows Table 17−20. Rather than the absolute address, you load the offset shifted right by 4 bits. The 4-bit shift is required because the buffer base address registers must hold the 8 high bits. When the USB module uses those 8 bits, it extends them with four least significant 0s.

Figure 17−25. Endpoint Buffer Base Address Registers

	7−0
USBxBAXn	BAX (bits 11−4 of offset)
USBxBAYn	BAY (bits 11−4 of offset)
	R/W − U

Legend:

R/W	Read/write access
− U	The contents of the registers is undefined after a DSP reset.

Table 17−20. Endpoint Buffer Base Address Register Bit Descriptions

Bit	**Field**	**Value**	**Description**
For IN endpoint n:			
USBIBAXn(7−0)	BAX	00h−FFh	Bits 11−4 of the X-buffer base address for IN endpoint n. BIts 3−0 are 0s.
USBIBAYn(7−0)	BAY	00h−FFh	Bits 11−4 of the Y-buffer base address for IN endpoint n. Bits 3−0 are 0s.
For OUT endpoint n:			
USBOBAXn(7−0)	BAX	00h−FFh	Bits 11−4 of the X-buffer base address for OUT endpoint n. Bits 3−0 are 0s.
USBOBAYn(7−0)	BAY	00h−FFh	Bits 11−4 of the Y-buffer base address for OUT endpoint n. Bits 3−0 are 0s.

Example 17−2. Loading the Endpoint Buffer Base Addresses

IN endpoint 1, X buffer: Assigned to the 1st 64 bytes of the buffer RAM
IN endpoint 1, Y buffer: Assigned to the 2nd 64 bytes of the buffer RAM

Buffer	I/O Address Seen by CPU	Value Loaded into Buffer Base Address Register
X	USB module registers base address + Offset for top of buffer RAM (80h)	USBIBAX1 = (80 >>4)
Y	USB module registers base address + Offset for 64 bytes further (C0h)	USBIBAY1 = (C0 >>4)

17.8.3.4 Endpoint n Count Registers (USBxCTXn, USBxCTYn) (x = O or I; n = 1, 2, 3, 4, 5, 6, or 7)

Each general-purpose endpoint has two count registers (see Figure 17−26 and Table 17−21): one for its X buffer and one for its Y buffer. The NAK bit corresponds to the negative acknowledgement (NAK) of the USB protocol. While the NAK bit is set (NAK = 1), the UBM sends a NAK in response to host data requests at the endpoint. For more details about the role of the NAK bit, see section 17.3, *USB Buffer Manager (UBM)*, on page 17-11.

Each buffer (X or Y) needs a count register to determine how many bytes the UBM should move out of the buffer (for an IN endpoint) or to record how many bytes the UBM has moved into the buffer (for an OUT endpoint). If the endpoint is in the non-isochronous mode (ISO = 0), the CTX/CTY field is the full count register. If the endpoint is in the isochronous mode (ISO = 1), the CTX/CTY field is the 7 low bits of a 10-bit count register. The 3 high bits come from another register, as described in Table 17−21.

Figure 17−26. Endpoint n Count Registers

	7	6−0
USBxCTXn	NAK	CTX (bytes)
USBxCTYn	NAK	CTY (bytes)
	R/W − U	R/W − U

Legend:

R/W	Read/write access
− U	The contents of these bits are undefined after a DSP reset.

Table 17−21. Endpoint n Count Register Bit Descriptions

Bit	Field	Value	Description
7	NAK		Negative acknowledgement
			For IN endpoint n:
		0	Data in the endpoint buffer is ready for an IN transfer.
		1	Data in the endpoint buffer is not ready. The UBM will send a NAK in response to an IN token.
			For OUT endpoint n:
		0	The endpoint buffer is ready for an OUT transfer.
		1	Either the endpoint buffer is not ready or it contains a valid data packet from the previous transfer. The UBM will send a NAK in response to an OUT token.
6−0	CTX/CTY		Count bits for buffer b (b = X or Y).
			In the non-isochronous mode:
		0−64	**For IN endpoint n:** When NAK = 0, this value indicates the number of bytes the UBM should move out of buffer b in response to an IN token. **For OUT endpoint n:** This value is a running count of the bytes the UBM has stored to buffer b. **Note:** If CTb holds a value greater than 64, the results are unpredictable.
			In the isochronous mode:
		000 0000b−111 1111b	**For IN endpoint n:** These bits are the 7 low bits of the 10-bit byte count for buffer b. As shown in Figure 17−27, the 3 high bits are taken from USBICNFn. Load into these 10 bits the number of bytes you want the UBM to read from buffer b in response to an IN token. When an IN token arrives and NAK = 0, the UBM reads that many bytes, passing them to the serial interface engine (SIE) for the host. **For OUT endpoint n:** These bits are the 7 low bits of the 10-bit byte counter for buffer b. As shown in Figure 17−28, the 3 high bits are taken from USBOCTXHn/USBOCTYHn. The 10-bit counter keeps a running count of the bytes the UBM has written to buffer b.

Figure 17−27. IN Endpoint n Extended Count Values in the Isochronous Mode (ISO = 1)

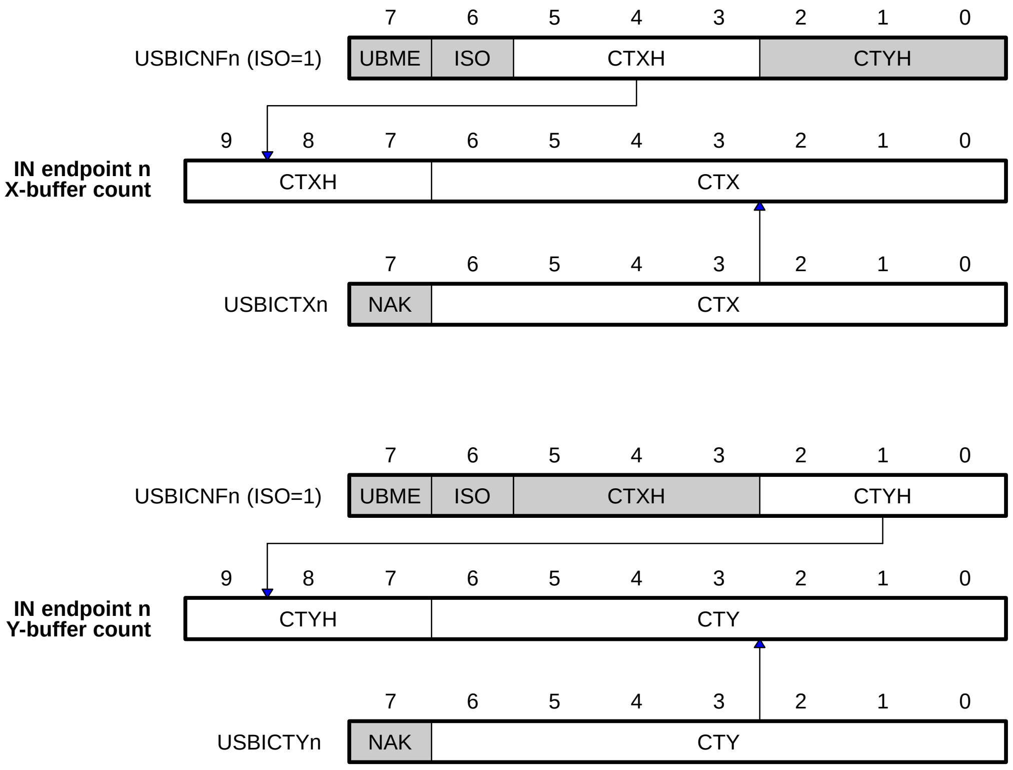
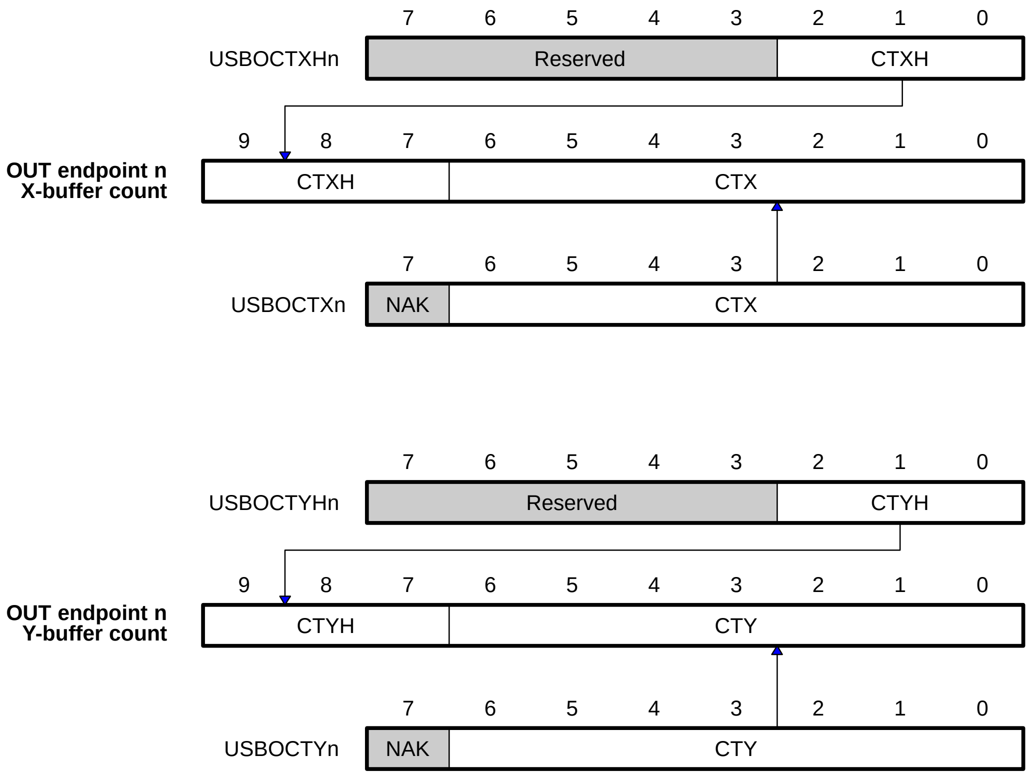


Figure 17−28. OUT Endpoint n Extended Count Values in the Isochronous Mode (ISO = 1)



17.8.3.5 Endpoint n X-/Y-Buffer Size Register (USBxSIZn) (x = O or I; n = 1, 2, 3, 4, 5, 6, or 7)

The SIZ field in USBxSIZn determines the maximum packet size: the number of bytes the endpoint buffer can hold at one time. In the double buffer mode (required for the USB DMA controller), the X buffer and the Y buffer are of equal size, and SIZ defines that size.

Figure 17−29. Endpoint n X-/Y-Buffer Size Register (USBxSIZn)

	7	6−0
USBxSIZn	Reserved	SIZ (bytes)
		R/W − U

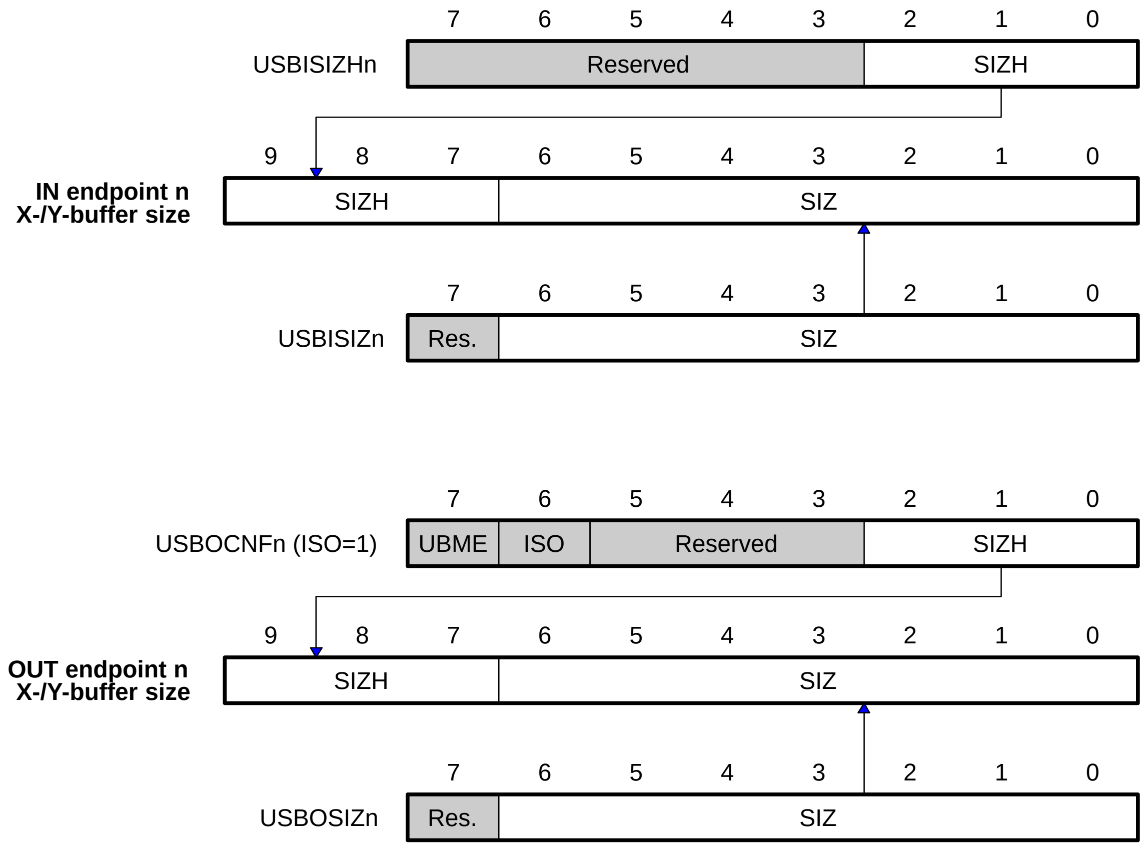
Legend:

R/W	Read/write access
− U	The contents of these bits are undefined after a DSP reset.

Table 17−22. Endpoint Size Register Bit Descriptions

Bit	Field	Value	Description
7	Reserved		This bit is not available for use.
6−0	SIZ		X-/Y-buffer size
			In the non-isochronous mode:
		8, 16, 32, or 64	The number of bytes in the buffer RAM allocated for the X buffer. In the double buffer mode (DBUF = 1), the same number of bytes is allocated for the Y buffer. **Note:** If SIZ holds a value greater than 64, the results are unpredictable.
			In the isochronous mode:
		000 0001b− 111 1111b	The 7 low bits of the 10-bit buffer size. As shown in Figure 17−30, the 3 high bits are taken from USBISIZHn for an IN endpoint or from USBOCNFn for an OUT endpoint. The 10-bit buffer size is used for the X buffer and, in the double buffer mode, is also used for the Y buffer.

Figure 17−30. Extended Size Values in the Isochronous Mode (ISO = 1)



17.8.3.6 Endpoint n Buffer Size and Count Extension Registers: USBISIZHn, USBICTXHn, USBOCTYHn (n = 1, 2, 3, 4, 5, 6, or 7)

These registers provide buffer size and count extensions for isochronous transfers between the USB module and a host processor. Specifically:

- ❑ If IN endpoint n is using the isochronous mode (ISO = 1 in USBICNFn), USBISIZHn supplies 3 high bits to extend the X-/Y-buffer size from a 7-bit value (SIZ) to a 10-bit value (SIZH:SIZ). The formation of this extended size value is shown in Figure 17−30 on page 17-80.
- ❑ If OUT endpoint n is using the isochronous mode (ISO = 1 in USBOCNFn):
  - ■ USBOCTXHn supplies 3 high bits to extend the X-buffer byte count from a 7-bit value (CTX) to a 10-bit value (CTXH:CTX).
  - ■ USBOCTYHn supplies 3 high bits to extend the Y-buffer byte count from a 7-bit value (CTY) to a 10-bit value (CTYH:CTY).

  The formation of these extended count values is shown in Figure 17−28 on page 17-78.

Figure 17−31. Endpoint Buffer Size and Count Extension Registers

USBISIZHn

7−3	2−0
Reserved	SIZH
	R/W − U

USBOCTXHn

7−3	2−0
Reserved	CTXH
	R/W − U

USBOCTYHn

7−3	2−0
Reserved	CTYH
	R/W − U

Legend:

R/W	Read access/Write access
− U	The content of these bits is undefined after a DSP reset.

Table 17−23. Endpoint Size and Count Extension Register Bit Descriptions

Bit	Field	Value	Description
USBISIZHn(2−0)	SIZH	000h−111h	In the isochronous mode (ISO = 1), these bits are the 3 high bits of the IN endpoint X-/Y-buffer byte count. The 7 low bits are in the SIZ bits of USBISIZn (see section 17.8.3.5 on page 17-78).
USBOCTXHn(2−0)	CTXH	000h−111h	In the isochronous mode (ISO = 1), these bits are the 3 high bits of the OUT endpoint X-buffer byte count. The 7 low bits are in the CTX bits of USBOCTXn (see section 17.8.3.4 on page 17-75).
USBOCTYHn(2−0)	CTYH	000h−111h	In the isochronous mode (ISO = 1), these bits are the 3 high bits of the OUT endpoint Y-buffer byte count. The 7 low bits are in the CTY bits of USBOCTYn (see section 17.8.3.4 on page 17-75).

17.8.4 Descriptor Registers for IN and OUT Endpoints 0

The control endpoints (IN endpoint 0 and OUT endpoint 0) each have two descriptor registers to define them: a configuration register and a count register, which are described here.

17.8.4.1 Endpoint 0 Configuration Register (USBxCNF0) (x = O or I)

IN endpoint 0 and OUT endpoint 0 each have a configuration register of the form shown in Figure 17−32. Table 17−24 describes the bit fields of this regiser.

Figure 17−32. Endpoint 0 Configuration Register (USBxCNF0)

USBxCNF0

7	6	5	4	3	2−0
UBME	Reserved	TOGGLE	Reserved	STALL	Reserved
R/W − 0		R − 0		R/W − 0	

Legend:

R	Read-only access
R/W	Read/write access
− 0	A DSP reset forces these bits to 0.

Table 17−24. USBxCNF0 Bit Descriptions

Bit	Field	Value	Description
7	UBME		UBM access enable
		0	The UBM cannot access this endpoint (the endpoint is inactive).
		1	The UBM can access this endpoint (the endpoint is active).
6	Reserved		This bit is not available for use.
5	TOGGLE		Endpoint data toggle. This bit reflects the data toggle sequence (see section 17.1.2 on page 17-4).
		0	The next data packet is DATA0.
		1	The next data packet is DATA1.
4	Reserved		This bit is not available for use.
3	STALL		Endpoint stall. Set this bit to tell the USB host that the endpoint is stalled.
		0	No stall
		1	Endpoint stalled. A STALL handshake will be initiated in response to host access requests until the STALL bit is cleared. The STALL bit is automatically cleared when the next setup packet arrives.
2−0	Reserved		These bits are not available for use.

17.8.4.2 Endpoint 0 Count Register (USBxCT0) (x = O or I)

IN endpoint 0 and OUT endpoint 0 each has one count register. As shown in Figure 17−33 and Figure 17−34, the two count registers have the same form but different reset values for their NAK bits and different accessibility for their CT0 (count) fields.

Table 17−25 describes the bit fields of an endpoint 0 count register. The NAK bit corresponds to the negative acknowledgement (NAK) of the USB protocol. While the NAK bit is set (NAK = 1), the UBM sends a NAK in response to host requests at the endpoint. For more details about the role of the NAK bit, see section 17.3, *USB Buffer Manager (UBM)*, on page 17-11. The CT0 field determines how many bytes the UBM should move out of the endpoint buffer (for IN endpoint 0) or records how many bytes the UBM has moved into the endpoint buffer (for OUT endpoint 0).

Figure 17−33. Count Register for IN Endpoint 0 (USBICT0)

	7	6−0
USBICT0	NAK	CT0 (bytes)
	R/W − 1	R/W − 0

Legend:

R/W	Read/write access
− X	X is the value after a DSP reset.

Figure 17−34. Count Register for OUT Endpoint 0 (USBOCT0)

	7	6−0
USBOCT0	NAK	CT0 (bytes)
	R/W − 0	R − 0

Legend:

R	Read-only access
R/W	Read/write access
− X	X is the value after a DSP reset.

Table 17−25. USBxCT0 Bit Descriptions

Bit	Field	Value	Description
7	NAK		Negative acknowledgement
			For IN endpoint 0:
		0	Data in the endpoint buffer is ready for an IN transfer.
		1	Data in the endpoint buffer is not ready. The UBM will send a NAK in response to an IN token.
			For OUT endpoint 0:
		0	The endpoint buffer is ready for an OUT transfer.
		1	Either the endpoint buffer is not ready or it contains a valid data packet from the previous transfer. The UBM will send a NAK in response to an OUT token.

Table 17−25. USBxCT0 Bit Descriptions (Continued)

Bit	Field	Value	Description
6−0	CT0		Count bits
		0−64	**For IN endpoint 0:** When NAK = 0, this value indicates the number of bytes the UBM should move out of the endpoint buffer in response to an IN token.
			For OUT endpoint 0: This value is a running count of the bytes the UBM has stored to the endpoint buffer.
			Note: If CT0 holds a value greater than 64, the results are unpredictable.

17.8.5 Interrupt Registers

This section describes registers that identify the current USB interrupt source (USBINTSRC), hold flags for interrupt events (USBxEPIF, USBxDGIF, and USBxDRIF), enable/disable interrupt requests (USBxEPIE and USBxDIE).

17.8.5.1 Interrupt Source Register (USBINTSRC)

All interrupt requests generated in the USB module are multiplexed through an arbiter to a single USB interrupt request for the CPU. The interrupt service routine can determine the interrupt source by reading the interrupt source register (USBINTSRC). Then the ISR can branch to the appropriate subroutine.

When the CPU reads the INTSRC field (see Figure 17−35), it obtains a 7-bit interrupt source code. Table 17−27 shows the valid INTSRC codes and the corresponding interrupt sources.

When the interrupt arbiter receives multiple interrupt requests at the same time, it services them one at a time according to a predefined priority ranking. The INTSRC value also identifies the priority of an interrupt source (02h is highest, 52h is lowest).

For more details about the interrupt sources and how USBINTSRC is used, see section 17.6 on page 17-46.

Figure 17−35. Interrupt Source Register (USBINTSRC)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								INTSRC							
								R−0							

Table 17−26. Interrupt Source Register (USBINTSRC) Field Values

Bit	Field	Value	Function
15−8	Reserved		These bits are not available for use.
7−0	INTSRC	00h−52h	These 8 bits indicate the interrupt source, the event that caused an interrupt to the CPU. In addition, these bits indicate the priority of each interrupt source. The interrupt source corresponding to INTSRC = 02h has the highest priority, and the interrupt source corresponding to INTSRC = 52h has the lowest priority.

Table 17−27. Interrupt Sources Matched to INTSRC Values

INTSRC Value/ Priority	Interrupt Source	INTSRC Value/ Priority	Interrupt Source
00h	(No interrupt)	36h	OUT endpoint 3 DMA Reload
02h	OUT endpoint 0	37h	OUT endpoint 3 DMA Go
04h	IN endpoint 0	38h	OUT endpoint 4 DMA Reload
06h	RSTR interrupt	39h	OUT endpoint 4 DMA Go
08h	SUSR interrupt	3Ah	OUT endpoint 5 DMA Reload
0Ah	RESR interrupt	3Bh	OUT endpoint 5 DMA Go
0Ch	SETUP packet received	3Ch	OUT endpoint 6 DMA Reload
0Eh	SETUP packet overwrite	3Dh	OUT endpoint 6 DMA Go
10h	SOF	3Eh	OUT endpoint 7 DMA Reload
11h	PSOF	3Fh	OUT endpoint 7 DMA Go
12h	OUT endpoint 1	42h	IN endpoint 1 DMA Reload
14h	OUT endpoint 2	43h	IN endpoint 1 DMA Go
16h	OUT endpoint 3	44h	IN endpoint 2 DMA Reload
18h	OUT endpoint 4	45h	IN endpoint 2 DMA Go
1Ah	OUT endpoint 5	46h	IN endpoint 3 DMA Reload
1Ch	OUT endpoint 6	47h	IN endpoint 3 DMA Go
1Eh	OUT endpoint 7	48h	IN endpoint 4 DMA Reload
22h	IN endpoint 1	49h	IN endpoint 4 DMA Go
24h	IN endpoint 2	4Ah	IN endpoint 5 DMA Reload
26h	IN endpoint 3	4Bh	IN endpoint 5 DMA Go
28h	IN endpoint 4	4Ch	IN endpoint 6 DMA Reload
2Ah	IN endpoint 5	4Dh	IN endpoint 6 DMA Go
2Ch	IN endpoint 6	4Eh	IN endpoint 7 DMA Reload
2Eh	IN endpoint 7	4Fh	IN endpoint 7 DMA Go
32h	OUT endpoint 1 DMA Reload	50h	Host interrupt
33h	OUT endpoint 1 DMA Go	52h	Host error
34h	OUT endpoint 2 DMA Reload	other	Reserved
35h	OUT endpoint 2 DMA Go		

17.8.5.2 Endpoint Interrupt Flag Register (USBxEPIF) (x = O or I)

For each endpoint, the USB module sets an interrupt flag in USBxEPIF every time the UBM moves data in/out of the endpoint's buffer. For example, if the UBM has finished moving data into the buffer for OUT endpoint 2, the OE2 flag is set in USBOEPIF. In addition to setting a flag, the USB module can generate an endpoint interrupt request. For more details about endpoint interrupt requests, see section 17.6.2 on page 17-48.

Figure 17−36. Endpoint Interrupt Flag Register for OUT Endpoints (USBOEPIF)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
OE7	OE6	OE5	OE4	OE3	OE2	OE1	OE0
R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0

Figure 17−37. Endpoint Interrupt Flag Register for IN Endpoints (USBIEPIF)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
IE7	IE6	IE5	IE4	IE3	IE2	IE1	IE0
R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0

Table 17−28. Endpoint Interrupt Flag Register (USBxEPIF) Field Values

Bit	Field	Value	Function
15−8	Reserved	0	These bits are not available for use.
7−0	xE[7:0]		x = O (for OUT) or I (for IN).
		0	No interrupt pending
		1	Indicates that the corresponding endpoint generated an interrupt. It is set by the hardware and cleared either when the CPU reads the interrupt source register (USBINTSRC) with INTSRC equal to the corresponding interrupt or when the CPU writes a 1 to this bit.

17.8.5.3 Endpoint Interrupt Enable Register (USBxEPIE) (x = O or I)

If a bit gets set in the endpoint interrupt flag register (USBxEPIF) and the corresponding bit is set in USBxEPIE, an endpoint interrupt request is generated. For example, if the IE2 bit of USBIEPIF gets set and the IE2 bit of USBIEPIE is 1, an IN endpoint 2 interrupt request is generated. For more details about endpoint interrupt requests, see section 17.6.2 on page 17-48.

Figure 17−38. Endpoint Interrupt Enable Register for OUT Endpoints (USBOEPIE)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
OE7	OE6	OE5	OE4	OE3	OE2	OE1	OE0
R/W−0	R/W−0	R/W−0	R/W−0	R/W−0	R/W−0	R/W−0	R/W−0

Figure 17−39. Endpoint Interrupt Enable Register for IN Endpoints (USBIEPIE)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
IE7	IE6	IE5	IE4	IE3	IE2	IE1	IE0
R/W−0	R/W−0	R/W−0	R/W−0	R/W−0	R/W−0	R/W−0	R/W−0

Table 17−29. Endpoint Interrupt Enable Register (USBxEPIE) Field Values

Bit	Field	Value	Function
15−8	Reserved	0	These bits are not available for use.
7−0	xE[7:0]		Endpoint interrupt enable. x = O (for OUT) or I (for IN).
		0	Endpoint interrupt requests are disabled for the endpoint.
		1	Endpoint interrupt requests are enabled for the endpoint.

17.8.5.4 DMA GO Interrupt Flag Register (USBxDGIF) (x = O or I)

At the completion of a DMA transfer, if RLD = 0, the USB DMA controller clears the GO bit of the endpoint, and the corresponding GO interrupt flag is set in USBxDGIF. For example, when the controller goes idle after servicing OUT endpoint 6, the OE6 bit is set in USBODGIF. In addition to setting the flag, the USB module can generate a DMA GO interrupt request. For details on GO interrupt requests, see section 17.6.3 on page 17-50.

Figure 17−40. DMA GO Interrupt Flag Register for OUT Endpoints (USBODGIF)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
OE7	OE6	OE5	OE4	OE3	OE2	OE1	Reserved
R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	

Figure 17−41. DMA GO Interrupt Flag Register for IN Endpoints (USBIDGIF)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
IE7	IE6	IE5	IE4	IE3	IE2	IE1	Reserved
R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	

Table 17−30. DMA GO Interrupt Flag Register (USBxDGIF) Field Values

Bit	Field	Value	Function
15−8	Reserved	0	These bits are not available for use.
7−1	xE[7:1]		x = O (for OUT) or I (for IN). The USB DMA controller sets the DMA GO flag bit to indicate that the controller is ready for a new DMA transfer. The flag bit is cleared either when the CPU reads the interrupt source register (USBINTSRC) with INTSRC equal to the corresponding interrupt or when the CPU writes a 1 to this bit.
		0	No GO interrupt pending
		1	GO interrupt pending
0	Reserved	0	This bit is not available for use.

17.8.5.5 DMA RLD Interrupt Flag Register (USBxDRIF) (x = O or I)

At the completion of a DMA transfer, if RLD = 1, the USB DMA controller clears the RLD bit of the endpoint, and the corresponding RLD interrupt flag is set in USBxDRIF. For example, when the controller performs a DMA reload operation for IN endpoint 7, the IE7 bit is set in USBIDRIF. In addition to setting the flag, the USB module can generate a DMA RLD interrupt request. For details on RLD interrupt requests, see section 17.6.3 on page 17-50.

Figure 17−42. DMA RLD Interrupt Flag Register for OUT Endpoints (USBODRIF)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
OE7	OE6	OE5	OE4	OE3	OE2	OE1	Reserved
R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	

Figure 17−43. DMA RLD Interrupt Flag Register for IN Endpoints (USBIDRIF)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
IE7	IE6	IE5	IE4	IE3	IE2	IE1	Reserved
R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	

Table 17−31. DMA RLD Interrupt Flag Register (USBxDRIF) Field Values

Bit	Field	Value	Function
15−8	Reserved	0	These bits are not available for use.
7−1	xE[7:1]		x = O (for OUT) or I (for IN). The DMA RLD flag bit is set when the USB DMA controller completes a DMA reload operation (see section 17.4.5 on page 17-18). The flag bit is cleared either when the CPU reads the interrupt source register (USBINTSRC) with INTSRC equal to the corresponding interrupt or when the CPU writes a 1 to this bit.
		0	No RLD interrupt pending
		1	RLD interrupt pending
0	Reserved		This bit is not available for use.

17.8.5.6 DMA Interrupt Enable Register (USBxDIE) (X = O or I)

USBxDIE enables or disables both DMA GO interrupt requests and DMA RLD interrupt requests. Consider DMA interrupt activity for IN endpoint 2. If the USB module sets the IE2 GO flag in USBIDGIF and IE2 = 1 in USBIDIE, a DMA GO interrupt request is generated for the endpoint. Likewise, if the IE2 RLD flag is set in USBIDRIF and IE2 = 1 in USBIDIE, a DMA RLD interrupt is generated for the endpoint. For more details about DMA interrupt requests, see section 17.6.3 on page 17-50.

Figure 17−44. DMA Interrupt Enable Register for OUT Endpoints (USBODIE)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
OE7	OE6	OE5	OE4	OE3	OE2	OE1	Reserved
R/W−0	R/W−0	R/W−0	R/W−0	R/W−0	R/W−0	R/W−0	

Figure 17−45. DMA Interrupt Enable Register for IN Endpoints (USBIDIE)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
IE7	IE6	IE5	IE4	IE3	IE2	IE1	Reserved
R/W−0	R/W−0	R/W−0	R/W−0	R/W−0	R/W−0	R/W−0	

Table 17−32. DMA Interrupt Enable Register (USBxDIE) Field Values

Bit	Field	Value	Function
15−8	Reserved	0	These bits are not available for use.
7−1	xE[7:1]		x = O (for OUT) or I (for IN). DMA interrupt enable. This bit enables or disables both the RLD interrupt and the GO interrupt for the endpoint.
		0	RLD and GO interrupt requests are disabled for the endpoint.
		1	RLD and GO interrupt requests are enabled for the endpoint.
0	Reserved	0	This bit is not available for use.

17.8.6 Host-DMA Mode Registers

This section describes the registers used to configure the host-DMA mode of the USB module and to monitor host-DMA transfers. For more information about this mode, see section 17.5 on page 17-40.

17.8.6.1 Host Control Register (USBHCTL)

USBHCTL has bits to enable or disable the following: the host-DMA mode, the host interrupt, and the host error interrupt.

Figure 17−46. Host Control Register (USBHCTL)

7	6	5	4	3	2	1	0
Reserved					HERRIE	HIE	EN
					R/W−0	R/W−0	R/W−0

Table 17−33. Host Control Register (USBHCTL) Field Values

Bit	Field	Value	Function
7−3	Reserved		This bits are not available for use.
2	HERRIE		Host error interrupt enable
		0	Host error interrupt disabled
		1	Host error interrupt enabled
1	HIE		Host interrupt enable
		0	Host interrupt disabled
		1	Host interrupt enabled
0	EN		Host-DMA mode enable
		0	Host-DMA mode disabled
		1	Host-DMA mode enabled

17.8.6.2 Host Endpoint Select Register (USBHEPSEL)

USBHEPSEL determines which two of the 14 general-purpose endpoints are to be used by the USB DMA controller for host-DMA transfers.

Figure 17−47. Host Endpoint Select Register (USBHEPSEL)

7	6	5	4	3	2	1	0
Reserved		IEP		Reserved		OEP	
		R/W−0				R/W−0	

Table 17−34. Host Endpoint Select Register (USBHEPSEL) Field Values

Bit	Field	Value	Function
7	Reserved		This bit is not available for use.
6−4	IEP	1−7	IN endpoint number. IEP determines which of the general-purpose IN endpoints will be the host-DMA endpoint for IN transfers in the host-DMA mode.
3	Reserved		This bit is not available for your use.
2−0	OEP	1−7	OUT endpoint number. OEP determines which of the general-purpose OUT endpoints will be the host-DMA endpoint for OUT transfers in the host-DMA mode.

17.8.6.3 Host Status Register (USBHSTAT)

Check USBHSTAT to determine whether the host-DMA mode is disabled (DIS bit). USBHSTAT also provides interrupt flag bits for the host interrupt and the host error interrupt.

Figure 17−48. Host Status Register (USBHSTAT)

7	6	5	4	3	2	1	0
		Reserved			HERRIF	HIF	DIS
					R/W1C−0	R/W1C−0	R−1

Table 17−35. Host Status Register (USBHSTAT) Field Values

Bit	Field	Value	Function
7−3	Reserved		These bits are not available for use.
2	HERRIF		Host error interrupt flag
		0	No host error interrupt pending
		1	A host error has occurred: During an OUT transfer, the size of the data transferred from the host did not match the size specified in the protocol header.

Table 17−35. Host Status Register (USBHSTAT) Field Values (Continued)

Bit	Field	Value	Function
1	HIF		Host interrupt flag
		0	No host interrupt pending
		1	The USB DMA controller has completed a DMA transfer between the DSP memory and a host-DMA endpoint. This flag is set only if the host requests the interrupt by setting the INT bit in the first byte of the protocol header.
0	DIS		Host-DMA mode disabled flag
		0	Host-DMA mode not disabled
		1	Host-DMA mode disabled

17.8.7 General Control and Status Registers

This section describes the USB registers that perform general control and status functions. The bits in these registers allow such functions as resetting the USB module, shutting down the USB module, responding to specific conditions on the USB, and tracking USB frames for isochronous transfers.

17.8.7.1 Global Control Register (USBGCTL)

The SOFTRST bit in USBGCTL enables you to reset the USB module without resetting the entire DSP.

Figure 17−49. Global Control Register (USBGCTL)

7 … 1	0
Reserved	SOFTRST
	W−0

Table 17−36. Global Control Register (USBGCTL) Field Values

Bit	Field	Value	Function
7−1	Reserved		These bits are not available for use.
0	SOFTRST		Software reset
		1	Reset the USB module. The effects are the same as a hardware DSP reset: All of the USB module registers assume their power-on default values except for USBCTL. Two effects are: (1) The USB module goes inactive, and (2) The USB module is disconnected from the USB. As soon as the reset is complete, SOFTRST is cleared to 0, and the USB is free to run.

17.8.7.2 Frame Number Registers (USBFNUMH, USBFNUML)

See Figure 17−50 and Figure 17−51. The register formed by the concatenation of the bits in USBFNUMH and USBFNUML can help you with isochronous transfers. This register reports the current USB frame number. The 11-bit value rolls over to 0 if it grows larger than 2047.

Figure 17−50. Frame Number (High Part) Register (USBFNUMH)

7	6	5	4	3	2	1	0
Reserved					FNUMH		
					R−0		

Figure 17−51. Frame Number (Low Part) Register (USBFNUML)

7	6	5	4	3	2	1	0
FNUML							
R−0							

Table 17−37. Frame Number Register (USBFNUMH and USBFNUML) Field Values

Register(Bit)	Field	Value	Function
USBFNUMH(7−3)	Reserved	0	Reserved. Reads as 0.
USBFNUMH(2−0)	FNUMH	0h−3h	Most significant 3 bits of the current USB frame number.
USBFNUML(7−0)	FNUML	00h−FFh	Least significant 8 bits of the current USB frame number.

17.8.7.3 PSOF Interrupt Timer Counter (USBPSOFTMR)

USBPSOFTMR is shown in Figure 17−52 and described in Table 17−38.

A start-of-frame (SOF) token is expected on the USB every 1 millisecond. If an endpoint is placed in the isochronous mode, the SOF token triggers a pending DMA transfer for that endpoint. To provide the minimum latency between the preparation of data buffers and the availability of those buffers to the USB DMA controller, the USB module can give advance notice of each SOF token. Advance notice can come from a pre-SOF (PSOF) interrupt request. If you load USBPSOFTMR with a value *n* and enable PSOF interrupt requests, the USB module generates a PSOF interrupt request n 750-kHz clock cycles ahead of the expected arrival of the SOF token.

Figure 17−52. PSOF Interrupt Timer Counter (USBPSOFTMR)

7	6	5	4	3	2	1	0
PSOFTMR							
R/W−0							

Table 17−38. PSOF Interrupt Timer Counter (USBPSOFTMR) Field Values

Bit	Field	Value	Function
7−0	PSOFTMR	0−255	Indicates the number of clocks a PSOF interrupt should proceed each SOF token. The clock is 750 kHz (USB 12 MHz clock divided by 16). **Note:** The time of the next SOF token is predicted and the prediction is not guaranteed to be precise.

17.8.7.4 USB Control Register (USBCTL)

USBCTL enables and controls the features that are described in Table 17−39. This register is not affected during a reset operation that is initiated with the SOFTRST bit. This register is affected by other reset operations.

Figure 17−53. USB Control Register (USBCTL)

7	6	5	4	3	2	1	0
CONN	FEN	RWUP	FRSTE	Reserved		SETUP	DIR
R/W−0	R/W−1	R/W−0	R/W−1			R/W−0	R/W−0

Table 17−39. USB Control Register (USBCTL) Field Values

Bit	Field	Value	Function
7	CONN		Connect/disconnect
		0	Upstream port is disconnected. The pull-up disabled.
		1	Upstream port is connected. Pull up enabled
6	FEN		Function enable
		0	The USB module is inactive.
		1	The USB module is active. If connected to the bus (see the description for the CONN bit), the module will communicate with the host.
5	RWUP		Device remote wakeup request. Writing a 1 to this bit generates a remote wake up signal on the bus. The USB module clears RWUP after the signal is sent.
		0	Writing a 0 to this bit has no effect.
		1	The CPU has asked the USB module to generate a remote wakeup signal on the bus.

Table 17−39. USB Control Register (USBCTL) Field Values (Continued)

Bit	Field	Value	Function
4	FRSTE		Function reset/USB reset connection enable
		0	Function (module) reset is not connected to a USB reset. If a USB reset request is detected on the bus, the RSTR interrupt request is generated, but the USB module is not reset.
		1	Function (module) reset is connected to a USB reset. If a USB reset request is detected on the bus, the RSTR interrupt request is generated and the USB module is reset. All pending interrupts are cleared except the RSTR interrupt. The USB module is not disconnected from the bus.
3−2	Reserved	0	These bits are not available for use.
1	SETUP		Setup interrupt status. Software can write a 1 to this bit to indicate when a SETUP interrupt is being serviced. A write of 0 has no effect.
		0	The CPU has cleared this SETUP bit by writing 1 to the SETUP bit of the USB interrupt flag register (USBIF). A new setup packet will be accepted.
		1	Do not accept a new setup packet.The USB module will send a STALL in response to any setup packet until the SETUP bit is 0.
0	DIR		Endpoint 0 data direction. When a setup packet arrives, the CPU must decode the packet and set or clear the DIR bit to reflect the direction of data flow. This bit also determines the endpoint’s response to a 0-byte handshake packet. The USB module will not generate an endpoint interrupt upon completion of a control transfer handshake (a 0-byte transfer). The USB module stalls endpoint 0 if an OUT packet is expected (DIR = 0, OUT NAK = 0, IN NAK = 1) and an IN token arrives (early handshake), or the other way around.
		0	An OUT control transaction is expected. If an IN token (early handshake) arrives, respond with STALL.
		1	An IN control transaction is expected. If an OUT token (early handshake) arrives, respond with STALL.

17.8.7.5 USB Interrupt Flag Register (USBIF)

USBIF indicates the current status of the USB bus interrupts.

Notes:

1) The STPOW and SETUP bits are not automatically cleared when the CPU reads the interrupt source register (USBINTSRC). To clear one of these bits, write a 1 to it. The other bits in USBIF are automatically cleared when the corresponding interrupt value is read from USBINTSRC.

2) If a new setup token is received before SETUP is cleared, the USB module sets STPOW and an STPOW interrupt request (if enabled) is generated.

Figure 17−54. USB Interrupt Flag Register (USBIF)

7	6	5	4	3	2	1	0
RSTR	SUSR	RESR	SOF	PSOF	SETUP	Reserved	STPOW
R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0	R/W1C−0		R/W1C−0

Table 17−40. USB Interrupt Flag Register (USBIF) Field Values

Bit	Field	Value	Function
7	RSTR		Function reset request. This bit is set in response to host initiating a port reset. This bit is not affected by a USB module reset.
		0	No reset condition detected on the USB
		1	Reset condition detected on the USB
6	SUSR		Function suspend request
		0	No suspend condition detected on the USB
		1	Suspend condition detected on the USB
5	RESR		Function resume request
		0	No resume condition detected on the USB
		1	Activity on the USB resumes after a suspend condition.
4	SOF		Start of frame (SOF) notification
		0	No SOF packet detected on the USB
		1	SOF packet detected on the USB

Table 17−40. *USB Interrupt Flag Register (USBIF) Field Values (Continued)*

Bit	Field	Value	Function
3	PSOF	0	Pre-SOF (PSOF) notification. This bit is set a multiple of 16 USB clock cycles ahead of when the SOF token is expected. This allows the user to provide a transfer buffer for the USB DMA controller in time to prevent overflow or underflow conditions. This is especially helpful for isochronous transfers. The timing of this event is controlled by the content of the USBPSOFTMR register.
		0	No PSOF notification received
		1	PSOF notification received from PSOF timer
2	SETUP	0	SETUP packet received. Clearing this bit also clears the SETUP bit in the USB control register (USBCTL) and allows a new setup packet to move into the setup packet buffer.
		0	No setup packet received
		1	A new setup packet has arrived.
1	Reserved		This bit is not available for use.
0	STPOW		Setup overwrite
		0	No setup overwrite
		1	A setup overwrite has occurred; that is, SETUP = 1 in USBIF and another setup packet has arrived.

17.8.7.6 USB Interrupt Enable Register (USBIE)

The bits in USBIE enable or disable each of the interrupts associated with the bits in the USB interrupt flag register, USBIF (see section 17.8.7.5).

Figure 17−55. *USB Interrupt Enable Register (USBIE)*

7	6	5	4	3	2	1	0
RSTR	SUSR	RESR	SOF	PSOF	SETUP	Reserved	STPOW
R/W−0	R/W−0	R/W−0	R/W−0	R/W−0	R/W−0		R/W−0

Table 17−41. *USB Interrupt Enable Register (USBIE) Field Values*

Bit	Field	Value	Function
7	RSTR		Function reset interrupt enable. This bit is not affected by a USB reset.
		0	Function reset interrupt disabled
		1	Function reset interrupt enabled

Table 17−41. USB Interrupt Enable Register (USBIE) Field Values (Continued)

Bit	Field	Value	Function
6	SUSR		Function suspend interrupt enable
		0	Function suspend interrupt disabled
		1	Function suspend interrupt enabled
5	RESR		Function resume interrupt enable
		0	Function resume interrupt disabled
		1	Function resume interrupt enabled
4	SOF		Start-of-frame (SOF) interrupt enable
		0	SOF interrupt disabled
		1	SOF interrupt enabled
3	PSOF		Pre-SOF (PSOF) interrupt enable
		0	PSOF interrupt disabled
		1	PSOF interrupt enabled
2	SETUP		SETUP interrupt enable
		0	SETUP interrupt disabled
		1	SETUP interrupt enabled
1	Reserved		This bit is not available for use.
0	STPOW		Setup overwrite interrupt enable
		0	STPOW interrupt disabled
		1	STPOW interrupt enabled

17.8.7.7 USB Device Address Register (USBADDR)

This register contains the address that uniquely identifies the USB module to the USB host.

Figure 17−56. USB Device Address Register (USBADDR)

7	6	5	4	3	2	1	0
Reserved	ADDR						
	R/W−0						

Table 17−42. USB Device Address Register (USBADDR) Field Values

Bit	Field	Value	Function
7	Reserved	0	Reserved = 0
6−0	ADDR	00h−3Fh	These seven bits hold the USB address assigned to the USB module. The CPU writes the address to this register as a result of a Set Address request from the host.

17.8.7.8 USB Idle Control Register (USBIDLECTL)

This register, shown in Figure 17−57, is not part of the USB module, but it contains the idle enable bit that enables the CPU to put the USB module in its idle mode. It also contains an idle status bit that indicates when the USB module is in the idle mode. Finally, USBIDLECTL contains the USBRST bit, which the CPU can use to hold the USB module in reset. The details about these bits are in Table 17−43.

This register resides in I/O space.

Figure 17−57. USB Idle Control Register (USBIDLECTL)

7–3	2	1	0
Reserved	USBRST	IDLESTAT	IDLEEN
	R/W−0	R−0	R/W−0

Table 17−43. USB Idle Control Register (USBIDLECTL) Field Values

Bit	Field	Value	Function
7−3	Reserved		These bits are not available for use.
2	USBRST		USB module reset
		0	Hold the USB module in reset.
		1	Bring the USB module out of reset.
1	IDLESTAT		Idle status
		0	The USB module has not been turned off by an IDLE instruction.
		1	The USB module is idle following an IDLE instruction.
0	IDLEEN		Idle enable
		0	The USB module cannot be affected by an IDLE instruction.
		1	Allow the USB module to be deactivated by an IDLE instruction. If the PERIPH domain has been deactivated by an IDLE instruction (PERIS = 1 in the idle status register), the USB module is inactive.

Chapter 18

Watchdog Timer

The watchdog timer that is in TMS320VC5501 and TMS320VC5502 DSPs is described in the *TMS320VC5501/5502 DSP Timers Reference Guide* (SPRU618). The watchdog timer that is in TMS320VC5509 and TMS320VC5509A DSPs is described in the *TMS320VC5509/5510 DSP Timers Reference Guide* (SPRU595).

This page is intentionally left blank.

Appendix A

Revision History

This document was revised to SPRU317F from SPRU317E, which was released in June 2003. Notable changes made since the last revision are listed in the following table.

Page	Additions/Modifications/Deletions
Global	The name of the MultiMediaCard controller has been changed to MultiMediaCard / SD card controller.
Global	Information about the following peripherals has been updated and moved to separate reference guides: ❑ External memory interface (EMIF) of the TMS320VC5509, TMS320VC5509A, and TMS320VC5510 DSPs ❑ MultiMediaCard / SD card controller of the TMS320VC5509 and TMS320VC5509A DSPs
1-1	Modified Chapter 1, *Overview*.
17-6	Corrected Figure 17−1.

This page is intentionally left blank.

Index

A

analog-to-digital converter (ADC) 2-1

B

BREAKLN bit (break-lock indicator) of CLKMD 3-12
BYPASS DIV (bypass-mode divide value) of CLKMD 3-12
bypass mode 3-5

C

cache 9-1
CACHE idle domain 8-2
CACHE-domain idle configuration bit (CACHEI) 8-9
CACHE-domain idle status bit (CACHEIS) 8-9
CACHEI bit of ICR 8-9
CACHEIS bit of ISTR 8-9
CLKGEN idle domain 8-2
CLKGEN-domain idle configuration bit (CLKGENI) 8-9
CLKGEN-domain idle status bit (CLKGENIS) 8-9
CLKGENI bit of ICR 8-9
CLKGENIS bit of ISTR 8-9
CLKMD 3-12
CLKOUT pin 3-9
clock divider for CLKOUT 3-9
clock generator
  bypass mode 3-5
  CLKOUT pin 3-9
  clock mode register (CLKMD) 3-12
  DSP reset conditions 3-10
  idle mode 3-8
  introduction 3-2
  lock mode 3-6
  operational flow 3-3
clock mode register (CLKMD) 3-12
CPU idle domain 8-2
CPU-domain idle configuration bit (CPUI) 8-9
CPU-domain idle status bit (CPUIS) 8-9
CPUI bit of ICR 8-9
CPUIS bit of ISTR 8-9

D

direct memory access (DMA) controller 4-1
dividing down CPU clock for CLKOUTpin 3-9
DMA controller 4-1
DMA idle domain 8-2
DMA-domain idle configuration bit (DMAI) 8-9
DMA-domain idle status bit (DMAIS) 8-9
DMAI bit of ICR 8-9
DMAIS bit of ISTR 8-9
DSP reset
  effects on clock generator 3-10
  effects on idle domains 8-8
  effects on USB module 17-54

E

EHPI 7-1
EMIF 5-1
EMIF idle domain 8-2
EMIF-domain idle configuration bit (EMIFI) 8-9

EMIF-domain idle status bit (EMIFIS) 8-9
EMIFI bit of ICR 8-9
EMIFIS bit of ISTR 8-9
enhanced host port interface (EHPI) 7-1
external memory interface (EMIF) 5-1

G

general-purpose I/O port 6-1
general-purpose timer 15-1

H

history of this document since previous revision A-1
host port interface (HPI) 7-1

I

I2C module 10-1
IAI bit of CLKMD 3-12
ICR 8-9
idle configuration register (ICR) 8-9
idle configurations
  definition 8-1
  interrupt handling when CPU is reactivated 8-8
  process 8-4
  to change 8-6
  valid 8-5
idle domains
  description 8-2
  effects of reset 8-8
idle instruction used to change idle configurations 8-4
idle modes
  clock generator 3-8
  USB module 17-53
idle status register (ISTR) 8-9
initialize-after-idle bit (IAI) 3-12
initialize-on-break bit (IOB) 3-12
instruction cache 9-1
inter-integrated circuit (I2C) module 10-1
interrupt handling when CPU is reactivated 8-8
interrupts used to wake the CPU and clock generator 8-6
IOB bit of CLKMD 3-12
ISTR 8-9

L

LOCK (lock-mode indicator) bit of CLKMD 3-12
lock mode 3-6

M

McBSP 11-1
memory interface (EMIF) 5-1
MMC controller 12-1
multichannel buffered serial port (McBSP) 11-1
MultiMediaCard / SD card controller 12-1

N

notational conventions iii

O

overview of TMS320C55x peripherals 1-1

P

PERI bit of ICR 8-9
PERIPH idle domain 8-2
PERIPH-domain idle configuration bit (PERI) 8-9
PERIPH-domain idle status bit (PERIS) 8-9
peripherals overview for TMS320C55x DSPs 1-1
PERIS bit of ISTR 8-9
pins/signals
  clock generator (CLKOUT pin) 3-9
  USB module 17-7
PLL DIV (PLL divide) bits of CLKMD 3-12
PLL ENABLE bit of CLKMD 3-12
PLL MULT (PLL multiply) bits of CLKMD 3-12
power conservation with idle configurations 8-1

R

real-time clock (RTC)   13-1
registers
  clock mode register (CLKMD)   3-12
  idle   8-9
  system control   14-1
  USB module   17-55
related documentation from Texas Instruments   iv
reset
  effects on clock generator   3-10
  effects on idle domains   8-8
  effects on USB module   17-54
revision history of this document   A-1
RTC   13-1

S

serial port (McBSP)   11-1
signals/pins
  clock generator (CLKOUT pin)   3-9
  USB module   17-7
system control registers   14-1

T

TEST bit of CLKMD   3-12
timer, general-purpose   15-1
timer, watchdog   18-1
trademarks   v

U

universal asynchronous receiver/transmitter (UART)   16-1
Universal Serial Bus (USB) module
  block diagram   17-6
  buffer manager (UBM)   17-11
  DMA controller   17-13
  emulation   17-53
  host-DMA mode   17-40
  interrupts   17-46
  introduction   17-5
  overview   17-2
  power control   17-53
  registers   17-55
  reset   17-53
  signals/pins   17-7

V

valid idle configurations   8-5

W

watchdog timer   18-1