TMS320VC5507/5509 DSP Universal Serial Bus (USB) Module Reference Guide

Literature Number: SPRU596A
June 2004

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

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

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

Preface

Read This First

About This Manual

This manual describes the features and operation of the Universal Serial Bus (USB) module that is available on the TMS320VC5507, TMS320VC5509, and TMS320VC5509A digital signal processors (DSPs) in the TMS320C55x™ (C55x™) DSP generation.

Notational Conventions

This document uses the following conventions:

- 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 often are shown with the suffix b. For example, the following number is the decimal number 4 shown in binary form:

  0100b

- If a signal or pin is active low, it has an overbar. For example, the $\overline{\text{RESET}}$ signal is active low.
- Register bits are sometimes referenced with the following notations:

Notation	Description	Example
Register(n–m)	Bits n through m of Register	R(15–0) represents the 16 least significant bits of register R.
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
LSB	Least significant bit	In AC0(15–0), bit 0 is the LSB.
MSB	Most significant bit	In AC0(15–0), bit15 is the MSB.
LSByte	Least significant byte	In AC0(15–0), bits 7–0 are the LSByte.
MSByte	Most significant byte	In AC0(15–0), bits15–8 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.

TMS320VC5507 Fixed-Point Digital Signal Processor Data Manual (literature number SPRS244) describes the features of the TMS320VC5507 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.

TMS320C55x Technical Overview (literature number SPRU393) introduces 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 Peripherals Overview Reference Guide (literature number SPRU317) introduces the peripherals, interfaces, and related hardware that are available on 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

This page is intentionally left blank.

Contents

Figures

Tables

Examples

Universal Serial Bus (USB) Module

With the USB module, you can use the DSP to create a full speed USB slave device that is compliant with Universal Serial Bus Specification Version 2.0. This chapter explains the architecture of the module and how to program the module.

1 USB Concepts Overview

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

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 → Device

IN transfer A transfer of data from a device to the host:

Host ← Device

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

Control transfer The data transfer used by the 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 The data transfer used by the host to send or receive a large amount of non-time-critical data. The data transfer can be used 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 the interrupt transfer are keyboards and joysticks.

Isochronous transfer The data transfer used by USB devices to send or receive data in real time at a constant rate. The isochronous transfers can handle more data than interrupt transfers, but no error checking is performed. A typical candidate for isochronous transfers is a digital speaker system.

Note:

From an implementation standpoint, bulk and interrupt transfers are treated the same way in the C55x USB module. The only difference is that the interrupt 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 endpoint OUT0 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 endpoint IN0 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 can be shared by multiple USB devices. Data is transferred on the bus at regular (1-ms) 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 enumerated 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.

1.2 Data Toggle Mechanism

For non-isochronous transfers, USB devices use a data toggle mechanism to detect transmission errors and to keep the transmitter and the receiver of USB data synchronized 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.

The first packet of a USB transfer is a DATA0 packet. Subsequent packets alternate in type (DATA1, DATA0, DATA1, and so on).

2 Introduction to the USB Module

The USB module described in this section is a USB 2.0-compliant, full-speed (12 Mbps) slave module.

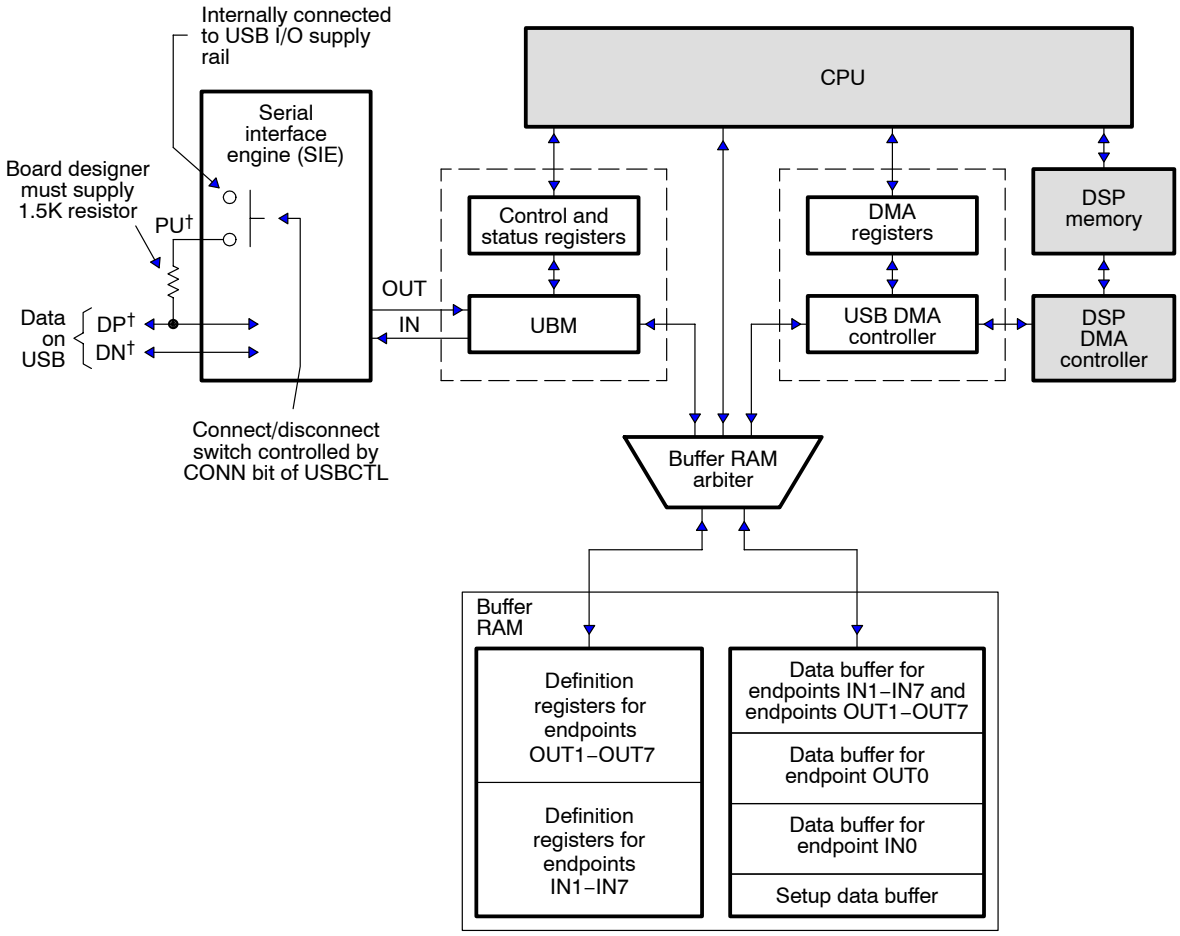
The C55x USB module has 16 endpoints:

- Two control endpoints (for control transfers only): OUT0 and IN0.
- Fourteen general-purpose endpoints (for other types of transfers): OUT1−OUT7 and IN1−IN7. Each of these endpoints can support:
  - 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. (The USB DMA controller cannot access the control endpoints.)

2.1 Block Diagram of the USB Module

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

Figure 1. Conceptual Block Diagram of the USB Module



† A more detailed figure of the pin connections is provided in section 2.2 (page 22).

- **Interface pins.** The following table introduces the three pins shown in Figure 1. More detailed pin connections are shown in Figure 2 (page 22).

Pin	Description
DP	Connect this pin to the USB connector terminal that carries the positive differential data.
DN	Connect this pin to the USB connector terminal that carries the negative differential data.
PU	Use this pin to connect a 1.5-Ω pullup resistor to the DP line. A software-controlled switch connects the pullup resistor to the USB I/O supply rail internally. 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 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 performs error-checking. 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 event occurs.

- **Buffer RAM.** The buffer RAM contains registers that are mapped in the DSP I/O space. The buffer RAM consist of:
  - 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-length (64-byte) data buffer for endpoint OUT0
  - A fixed-length (64-byte) data buffer for endpoint IN0
  - A fixed-length (8-byte) data buffer for a setup packet
  - Definition registers. Each of the general-purpose endpoints has eight definition registers that determine the endpoint characteristics.

- **USB DMA controller and the DMA registers.** The USB 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 registers for controlling and monitoring activity in that channel. The CPU can read from or write to each of these registers.

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

- **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 by which these three requesters share the buffer RAM.

  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.

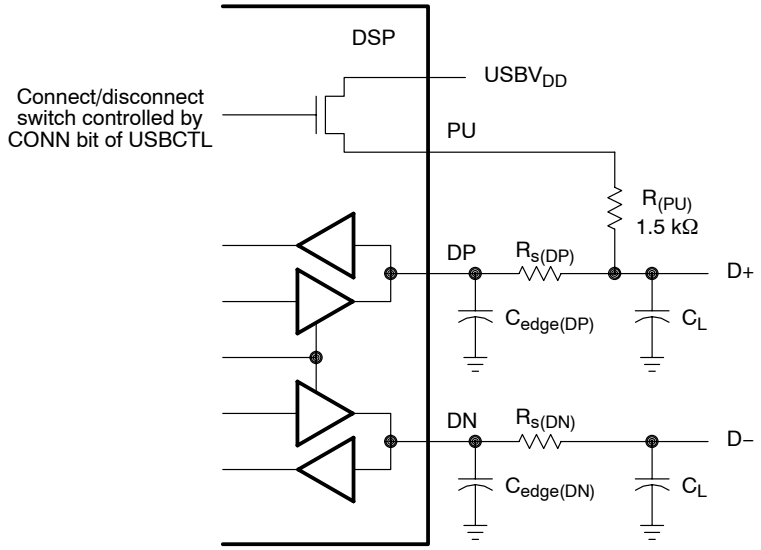
  The CPU can access the buffer RAM, including the definition registers, via I/O space. The CPU writes 16-bit values to I/O space. However, when the CPU writes to the RAM, the high eight bits are ignored, and when the CPU reads from the RAM, the high eight bits are don’t cares.

2.2 Connection of the USB Module to the Bus

Figure 2 shows the bus connections for the USB module. $R_{s(DP)}$ and $R_{s(DN)}$ are series resistors required to match the output impedance of the drivers for the differential data pins (DP and DN). Consult the device-specific data manual for the required values of $R_{s(DP)}$ and $R_{s(DN)}$, as well as $C_{edge(DP)}$, $C_{edge(DN)}$, and C_L. The 1.5 kΩ pullup resistor, $R_{(PU)}$, is a USB specification requirement for a full-speed device.

Setting the connect (CONN) bit of USBCTL connects the pullup (PU) pin to the power supply pin, $USBV_{DD}$. As a result, the USB host detects the USB module on the bus and starts the enumeration process. Clearing the CONN bit disconnects the two pins, resulting in a device-removal condition on the bus.

Figure 2. Connection of the USB Module to the Bus (Full-Speed Connection)

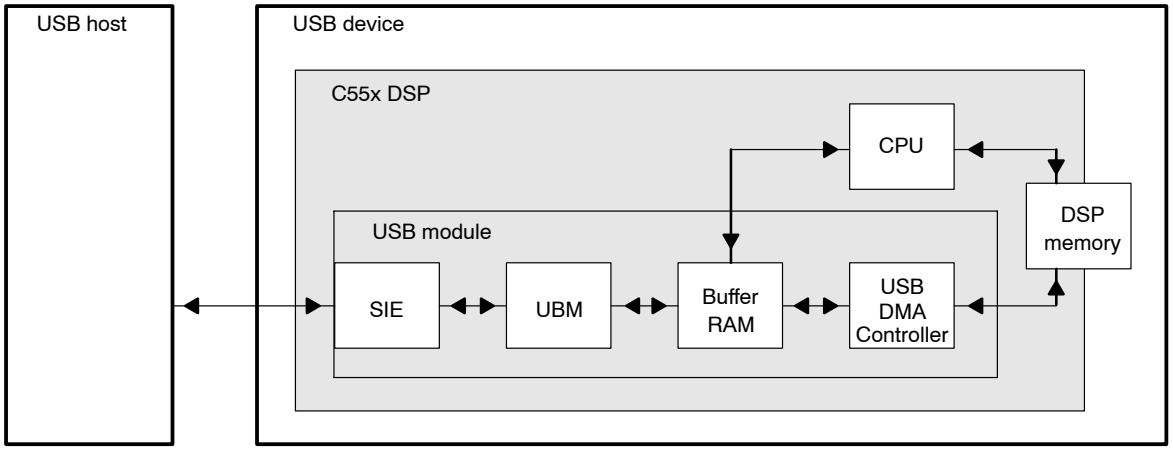


† Consult the device-specific data manual for the required values of $R_{s(DP)}$, $R_{s(DN)}$, $C_{edge(DP)}$, $C_{edge(DN)}$, and C_L.

2.3 Transfer of Data Between the USB Host and the DSP Memory

Figure 3 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 serial data from the host into 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 USB 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 3. Path for Data Transferred Between the Host and the DSP Memory



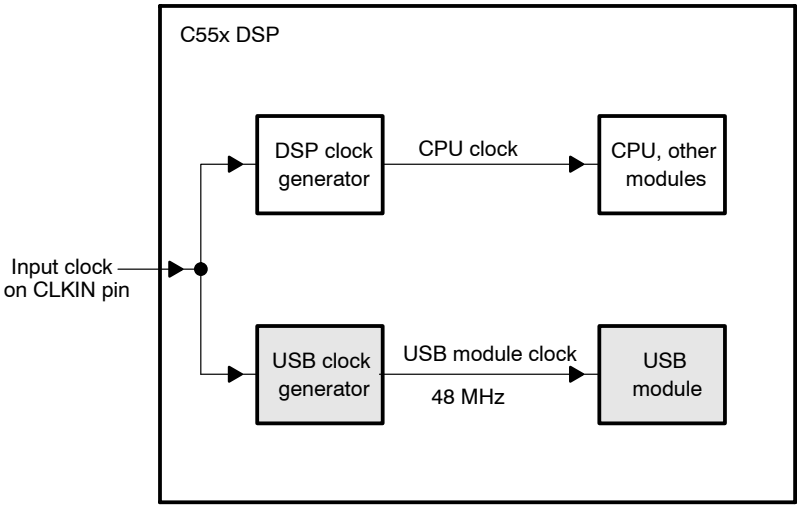
2.4 Clock Generation for the USB Module

As shown in Figure 4, 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.

Note:

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

Figure 4. Clock Generation for the USB Module



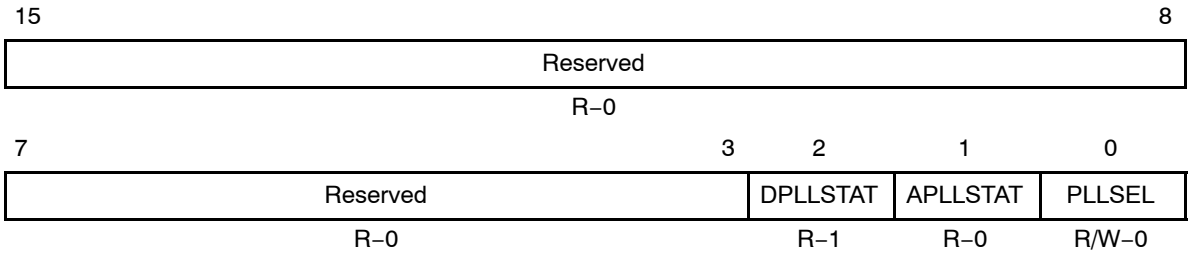
2.4.1 USB Clock Generator on TMS320VC5509 Devices Versus TMS320VC5507/5509A Devices

The USB clock generator on TMS320VC5509 devices includes only a digital phase-locked loop circuit (DPLL). On TMS320VC5507/5509A devices, the USB clock generator includes both a DPLL and an analog phase-locked loop circuit (APLL). The choice of DPLL or APLL is made with the USB PLL selection register (USBPLLSEL), which is shown in Figure 5 and described in Table 1. This register is at address 1E80h in the I/O space of a TMS320VC5507/5509A device.

Write to the PLLSEL bit to indicate which PLL circuit is to drive the USB module. Then poll the status bits, DPLLSTAT and APLLSTAT. When a stable clock signal is available from the selected PLL, the corresponding status bit is set and the other status bit is cleared.

At device power-up, the DPLL is selected by default. If software selects the APLL but the APLLSTAT bit remains 0, the APLL is not providing a stable clock signal and should be restarted. Until the APLL is ready, the DPLL is used.

Figure 5. USB PLL Selection Register (USBPLLSEL)



Legend: R = Read; W = Write; -*n* = Value after reset

Table 1. Bits of the USB PLL Selection Register (USBPLLSEL)

Bit	Field	Value	Description
15–3	Reserved		These read-only bits return 0s when read.
2	DPLLSTAT		DPLL status bit
		0	The DPLL is not the clock source for USB module.
		1	The DPLL is the clock source for the USB module.
1	APLLSTAT		APLL status bit
		0	The APLL is not the clock source for the USB module.
		1	The APLL is the clock source for the USB module.
0	PLLSEL		PLL selection bits
		0	The DPLL is selected as the USB module clock source.
		1	The APLL is selected as the USB module clock source.

2.4.2 DPLL Operation

TMS320VC5509 devices have a DPLL only. On TMS320VC5507/5509A devices, the DPLL is selected if PLLSEL = 0 in USBPLLSEL. The DPLL functions like the DSP clock generator. To find details about the DSP clock generator, see the *TMS320C55x DSP Peripherals Overview Reference Guide* (literature number SPRU317).

The DPLL has two modes: the lock mode and the bypass mode. In the lock mode, the input clock is multiplied and/or divided, and in the bypass mode, the input clock is divided only. At device power-up, the bypass mode is selected.

The lock mode or bypass mode can be selected with the PLLENABLE bit of the USB digital PLL control register (USBDPLL). This register, shown in Figure 6 and described in Table 2, contains all the bits for controlling the DPLL. USBDPLL has the same fields as the DSP clock mode register (CLKMD). USBDPLL is at address 1E00h in the I/O space of the DSP.

Figure 6. USB Digital PLL Control Register (USBDPLL)

15	14	13	12	11			
Reserved	IAI	IOB	Reserved‡	PLLMULT			
R−0	R/W−0	R/W−1	R/W−0	R/W−0			
7	6	5	4	3	2	1	0
PLLMULT	PLLDIV		PLLENABLE	BYPASSDIV		BREAKLN	LOCK
R/W−0	R/W−00		R/W−0	R/W−01		R−1	R−0

Legend: R = Read; W = Write; -*n* = Value after reset
‡ Always write 0 to this reserved bit.

Table 2. Bits of the USB Digital PLL Control Register (USBDPLL)

Bit	Field	Value	Description
15	Reserved		This read-only reserved bit returns 0 when read.
14	IAI		Initialize after idle bit. IAI determines how the DPLL reacquires the phase lock after the DPLL exits its idle mode (when the CLKGEN idle domain is reactivated).
		0	The DPLL does not restart the phase-locking sequence. Instead the DPLL reacquires the lock using the same lock settings that were in use just before the idle mode was entered.
		1	The DPLL restarts the phase-locking sequence. This option is recommended if the input clock has or may have changed while the clock generator was idle.

Table 2. Bits of the USB Digital PLL Control Register (USBDPLL) (Continued)

Bit	Field	Value	Description
13	IOB		Initialize on break bit. IOB determines whether the DPLL initializes the phase-locking sequence whenever the phase lock is broken.
			If the DPLL indicates a break in the phase lock:
		0	The DPLL is not interrupted. The DPLL stays in the lock mode, and continues to output the current clock signal.
		1	The DPLL switches to its bypass mode and restarts the phase-locking sequence.
12	Reserved	0	Always write 0 to this reserved bit.
11–7	PLLMULT	2 to 31	PLL multiply value. When PLLENABLE = 1 and the DPLL is locked, the input clock is multiplied by the unsigned integer in PLLMULT and is divided according to the value in the PLLDIV bits. The options are summarized in Table 3.
6–5	PLLDIV		PLL divide value. When PLLENABLE = 1 and the DPLL is locked, the input clock is multiplied by the unsigned integer in PLLMULT and is divided according to the value in the PLLDIV bits. The options are summarized in Table 3.
		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.
4	PLLENABLE		PLL enable bit. Write to PLLENABLE to enable or disable phase locking. When PLLENABLE is set, the DPLL initiates the phase-locking sequence. When the DPLL acquires the phase lock, the frequency of the DPLL output clock is determined by the PLLMULT and PLLDIV bits.
		0	Disable phase locking (enter the bypass mode).
		1	Enable phase locking and, when the correct output clock signal is generated, enter the lock mode.

Table 2. Bits of the USB Digital PLL Control Register (USBDPLL) (Continued)

Bit	Field	Value	Description
3–2	BYPASSDIV		Bypass-mode divide value. In the bypass mode, BYPASSDIV determines the frequency of the output clock signal. The options are summarized in Table 3.
		00b or 01b	No division/divide by 1 The frequency of the output clock signal is the same as the frequency of the input clock signal.
		10b	Divide by 2 The frequency of the output clock signal is 1/2 the frequency of the input clock signal.
		11b	Divide by 4 The frequency of the output clock signal is 1/4 the frequency of the input clock signal.
1	BREAKLN		Break-lock indicator. BREAKLN indicates whether the DPLL has broken the phase lock. In addition, if you write to CLKMD, BREAKLN is forced to 1.
		0	The DPLL has broken the phase lock.
		1	The phase lock is restored, or a write to CLKMD has occurred.
0	LOCK		Lock-mode indicator. LOCK indicates whether the DPLL is in its lock mode.
		0	The DPLL is in the bypass mode. The output clock signal has the frequency determined by the BYPASSDIV bits, or the DPLL is in the process of getting a phase lock.
		1	The DPLL is in the lock mode. The DPLL has a phase lock, and the output clock has the frequency determined by the PLLMULT bits and the PLLDIV bits.

Table 3. DPLL Options for the USB Module Clock Frequency

PLLENABLE	BYPASSDIV†	PLLDIV†	PLLMULT†	DPLL Option	USB Module Clock Frequency‡
0	0 or 1	X	X	Bypass mode; divide by 1	Input clock frequency × 1/1
0	2	X	X	Bypass mode; divide by 2	Input clock frequency × 1/2
0	3	X	X	Bypass mode; divide by 4	Input clock frequency × 1/4
1	X	0	K = 2 to 31	Lock mode; multiply by K	Input clock frequency × K/1
1	X	1	K = 2 to 31	Lock mode; multiply by K/2	Input clock frequency × K/2
1	X	2	K = 2 to 31	Lock mode; multiply by K/3	Input clock frequency × K/3
1	X	3	K = 2 to 31	Lock mode; multiply by K/4	Input clock frequency × K/4

† X = don't care
‡ The USB clock frequency must be 48 MHz for proper operation of the USB module.

2.4.3 APLL Operation (TMS320VC5507/5509A Devices Only)

On TMS320VC5507/5509A devices, the APLL is selected if PLLSEL = 1 in USBPLLSEL. The APLL has two modes: the lock mode and the bypass mode. In the lock mode, the input clock is multiplied and/or divided, and in the bypass mode, the input clock is divided only. At device power-up, the bypass mode is selected.

To configure the APLL, use the USB analog PLL control register, USBAPLL (see Figure 7 and Table 4). This register is at address 1F00h in the I/O space of a TMS320VC5507/5509A device. The following paragraphs describe the functionality provided by the bits in USBAPLL.

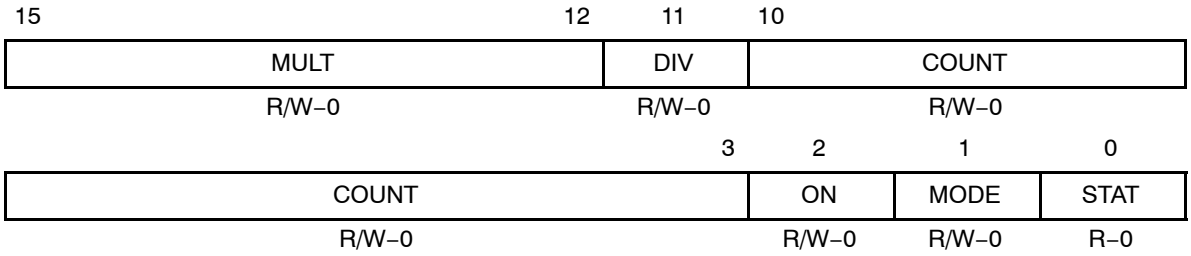
The MODE bit selects either the lock mode or the bypass mode. In the lock mode, the APLL requires the use of an internal voltage-controlled oscillator (VCO). The VCO is turned on by either the MODE bit or the ON bit, and the VCO is turned off if both of these bits are 0.

The MULT field of USBAPLL selects a multiplication factor K. In the bypass mode, K determines how the input clock (CLKIN) is divided to produce the USB module clock. In the lock mode, K is used in conjunction with the DIV bit to determine the relative frequency of the USB module clock. Table 15 summarizes all the options. Remember that the USB module requires a frequency of 48 MHz.

The COUNT field is a down-counter used to track the APLL lock time. The APLL requires approximately 350 μs to lock, and COUNT provides a means to track the lock time in terms of the input clock (CLKIN). As soon as the lock mode is turned on (that is, as soon as the MODE bit changes from 0 to 1), the phase-locking sequence begins and COUNT decremented by 1 every 16 CLKIN cycles. When COUNT reaches 0, the STAT bit is set. To make sure the countdown time matches the lock time, use the following equation when loading COUNT:

$$COUNT = \left(\frac{Lock\ time}{16 \times CLKIN\ period} \right) - 1$$

Figure 7. USB Analog PLL Control Register (USBAPLL)



Legend: R = Read; W = Write; -*n* = Value after reset
† This reserved bit must be kept 0 for proper operation of the USB clock generator.

Table 4. Bits of the USB Analog PLL Control Register (USBAPLL)

Bit	Field	Value	Description
15–12	MULT		PLL multiply value. MULT + 1 is the multiply factor K. Table 5 shows how this factor affects the operation of the APLL.
		0000b	K= 1
		0001b	K = 2
		0010b	K = 3
		0011b	K = 4
		0100b	K = 5
		0101b	K = 6
		0110b	K = 7
		0111b	K = 8
		1000b	K = 9
		1001b	K = 10
		1010b	K= 11
		1011b	K = 12
		1100b	K = 13
		1101b	K= 14
		1110b	K = 15
		1111b	K = 16

Table 4. Bits of the USB Analog PLL Control Register (USBAPLL) (Continued)

Bit	Field	Value	Description
11	DIV	0 or 1	PLL divide value. Table 5 shows how this value affects the operation of the APLL in the lock mode. This value is a don't care in the bypass mode.
10–3	COUNT	0 to 255	PLL lock counter bits. COUNT offers the ability to track the APLL lock time, which is approximately 350 μs. Before switching to the lock mode, load COUNT using the following equation: COUNT = [Lock time / (16 x CLKIN period)] –1 As soon as the lock mode is selected (that is, when the MODE bit changes from 0 to 1), COUNT is decremented by 1 every 16 CLKIN cycles. When COUNT reaches 0, the STAT bit is set.
2	ON		PLL VCO on bit. This bit can be used to enable the internal voltage-control oscillator (VCO).
		0	If the MODE bit is also 0, writing 0 to ON turns the VCO off.
		1	Writing 1 to ON turns the VCO on if it is not already running due to the MODE bit.
1	MODE		Mode selection bit
		0	Bypass mode. Writing 0 to MODE selects the bypass mode.
		1	Lock mode. Writing 1 to MODE selects the lock mode and also turns the VCO on if it is not already running due to the ON bit.
0	STAT		PLL lock status bit. If COUNT has been loaded properly, STAT = 1 indicates that enough time has passed for the APLL to achieve a phase lock.
		0	COUNT has not been decremented to 0.
		1	COUNT has been decremented to 0.

Table 5. APLL Options for the USB Module Clock Frequency

MODE	DIV	K†	APLL Option	USB Module Clock Frequency‡
0	0 or 1	1 to 15	Bypass mode; divide by 2	Input clock frequency × 1/2
0	0 or 1	16	Bypass mode; divide by 4	Input clock frequency × 1/4
1	0	1 to 15	Lock mode; multiply by K	Input clock frequency × K/1
1	0	16	Lock mode; multiply by 1	Input clock frequency × 1/1
1	1	odd	Lock mode; multiply by K/2	Input clock frequency × K/2
1	1	even	Lock mode; multiply by (K–1)/4	Input clock frequency × (K–1)/4

† K = MULT + 1
‡ The USB clock frequency must be 48 MHz for proper operation of the USB module.

2.4.4 Idle Mode Considerations

Idling the USB module does not idle the USB clock generator. Doing this simply prevents the USB module clock signal from driving the USB module.

Both the DSP clock generator and the USB clock generator are part of the CLKGEN idle domain. If the IDLE instruction deactivates the CLKGEN domain, both clock generators stop running.

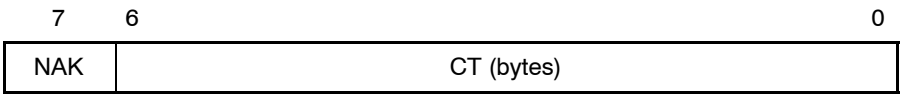
3 USB Buffer Manager (UBM)

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

Buffers For Transfers To DSP Memory	Buffers For Transfers From DSP Memory
Control endpoint buffers	
OUT0 buffer	IN0 buffer
General-purpose endpoint buffers	
OUT1 buffer (X or Y)	IN1 buffer (X or Y)
OUT2 buffer (X or Y)	IN2 buffer (X or Y)
OUT3 buffer (X or Y)	IN3 buffer (X or Y)
OUT4 buffer (X or Y)	IN4 buffer (X or Y)
OUT5 buffer (X or Y)	IN5 buffer (X or Y)
OUT6 buffer (X or Y)	IN6 buffer (X or Y)
OUT7 buffer (X or Y)	IN7 buffer (X or Y)

Each general-purpose endpoint 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:



The NAK bit corresponds to the negative acknowledgement (NAK) of the USB protocol. If the NAK bit is set (NAK = 1), the SIE sends a NAK in response to a host request to that particular endpoint. The UBM does not access the buffer until NAK is cleared (NAK = 0).

The CT (count) field indicates the number of bytes in a transfer. For an IN transfer, the CPU or the USB DMA controller must initialize the CT field and clear the NAK bit. For an OUT transfer, the UBM updates the CT field (after moving a new data packet from the SIE to the endpoint buffer) and sets the NAK bit.

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. Details can be found in section 7.3.4 (page 96).

Figure 8 summarizes the role of the NAK bit at an OUT endpoint. When an OUT packet is received by the serial interface engine (SIE), the UBM writes the incoming data to the appropriate endpoint buffer and sets the NAK bit to prevent the host from overwriting the packet before it is read by the CPU or the USB DMA controller. After the CPU or the DMA controller clears the NAK bit, the UBM can write to the buffer again.

Figure 8. Role of a NAK Bit in USB Activity at an OUT Endpoint

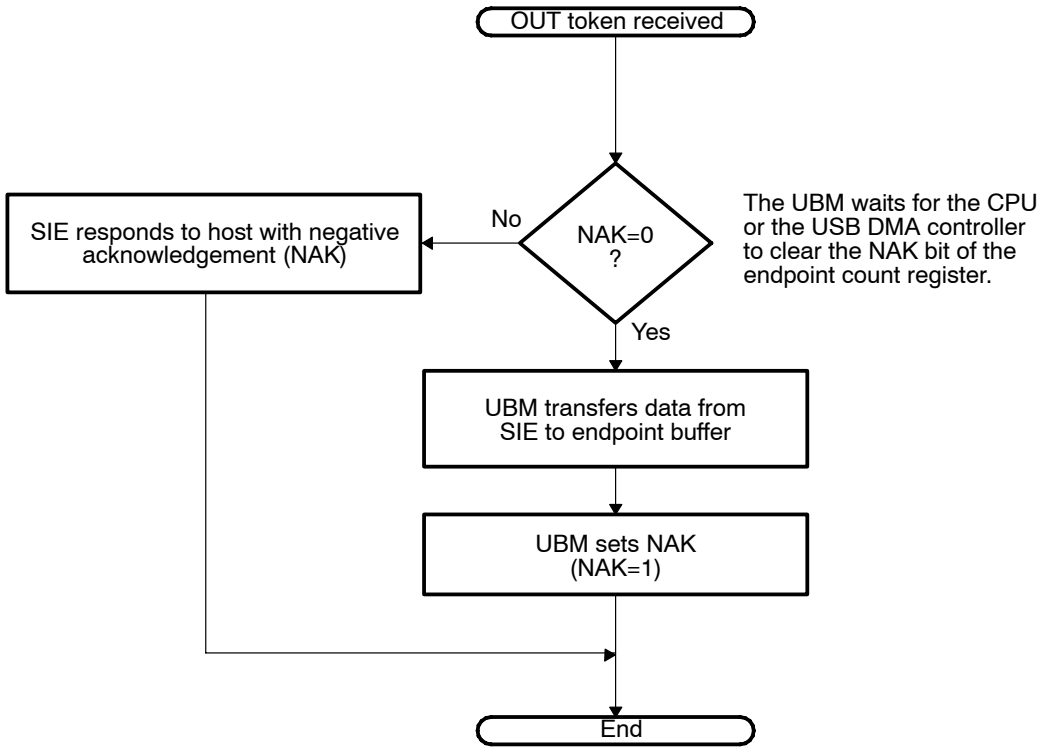
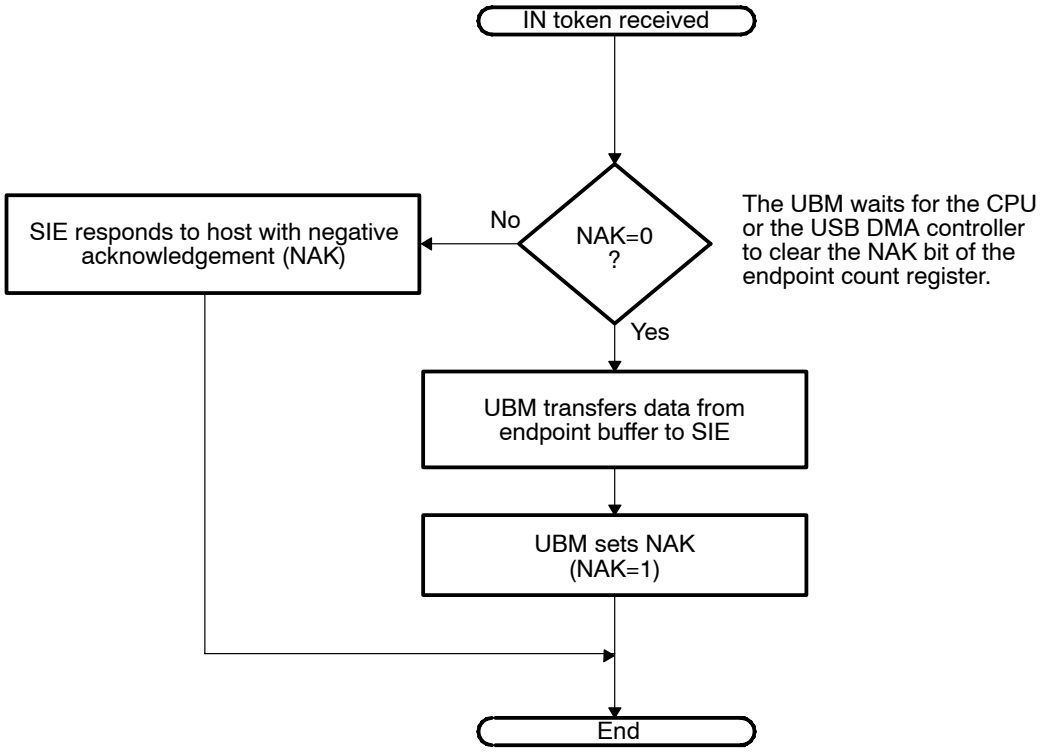


Figure 9 summarizes the role of the NAK bit at an IN endpoint. When an IN token is received, the UBM transfers data from the appropriate endpoint buffer to the SIE and sets the NAK bit so that the host will not receive the same packet again. After the CPU or the USB DMA controller loads a new packet into the endpoint buffer and clears the NAK bit, the UBM can access the buffer again.

Figure 9. Role of a NAK Bit in USB Activity at an IN Endpoint



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 (OUT1−OUT7 and IN1−IN7). The USB DMA controller cannot access the control endpoints (OUT0 and IN0).

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 sets up the DMA channel for data transfers. The CPU can continue with other tasks while the DMA controller moves the data. The DMA controller notifies the CPU of the transfer status via the GO and RLD status flags and USB DMA interrupts. For details, see section 5.3 on page 70.

4.2 Things To Consider Before Using the USB DMA Controller

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

- Each general-purpose endpoint must be in the double-buffer mode (DBUF = 1 in each of the registers USBOCNF1−USBOCNF7 and 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. 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 registers (USBODCTL1–USBODCTL7 and USBIDCTL1–USBIDCTL7). 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 performs another transfer.
- Do not enable RLD and GO interrupt requests in the registers USBODIE and USBIDIE.

4.3 Interaction Between the CPU and the USB DMA Controller

Table 6 shows how the CPU and the USB DMA controller interact. Figure 10 (a) 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 10 (b) shows how the CPU (via your code) can respond to GO and RLD interrupts from the DMA 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 4.5 on page 42): 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 6. DMA Transfers

Action of the CPU (DSP Code)	Action of the USB DMA Controller
Initialize the DMA registers. (Each general-purpose endpoint has eight DMA registers; see section 7.2 on page 78).	The USB DMA controller behaves according to the contents of the DMA registers.
Issue a GO command (set the GO bit). The GO bit is in the endpoint DMA control register (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).	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 endpoint DMA control register. Set RLD if you want the controller to begin another transfer after the current transfer is complete. Make sure the reload address and size registers are initialized first.	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.	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 using the DMA GO and RLD interrupt enable registers. 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 7.5.1 on page 106) to determine the interrupt source. Then the ISR can execute the appropriate subroutine.	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 5.3 on page 70.
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.	The USB DMA controller modifies bits in the DMA control register and in the interrupt flag registers to notify the CPU of specific actions or errors.

Figure 10. Activity for DMA Transfers

(a) USB DMA Controller Execution Flow (at an individual endpoint)

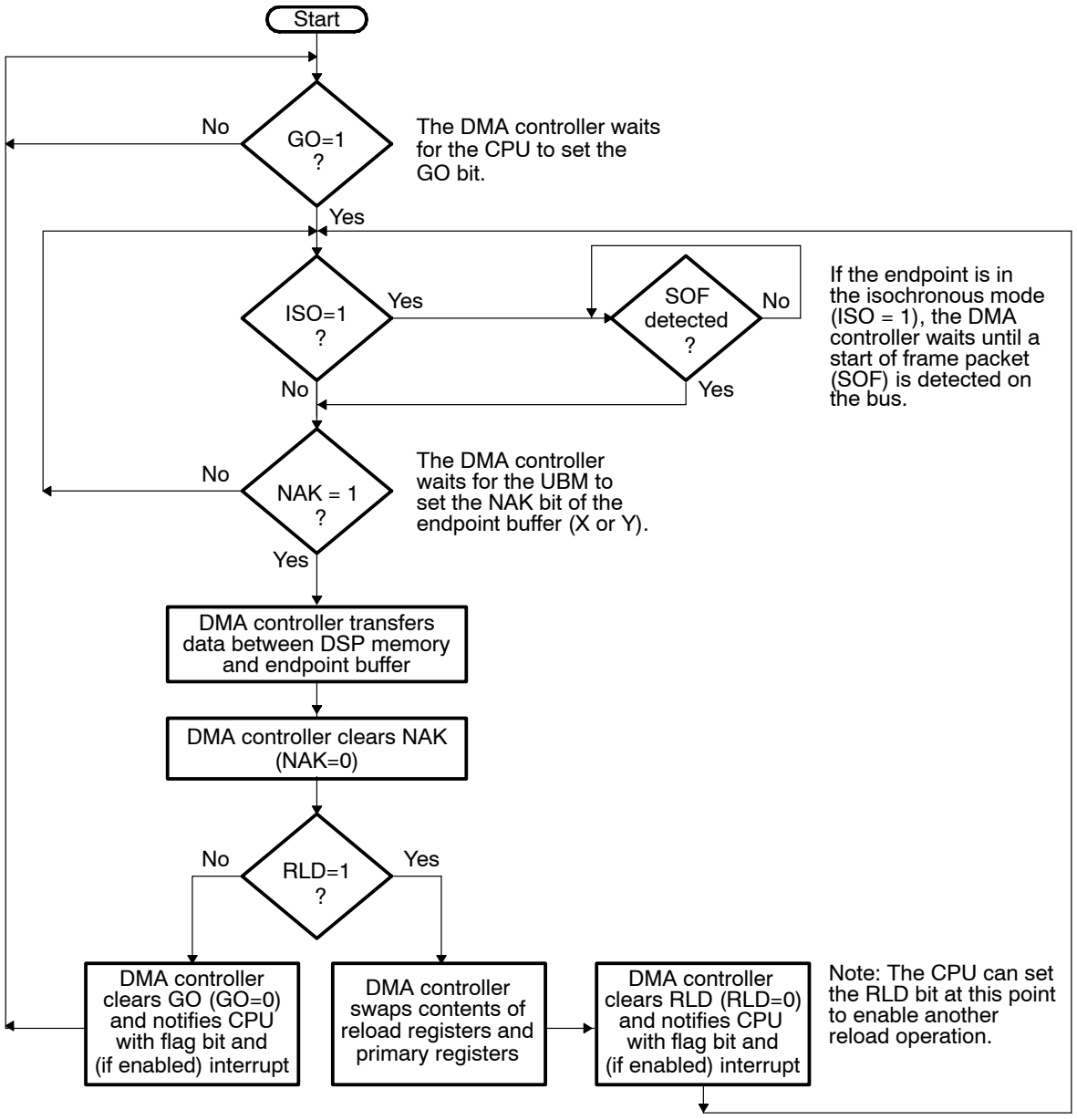
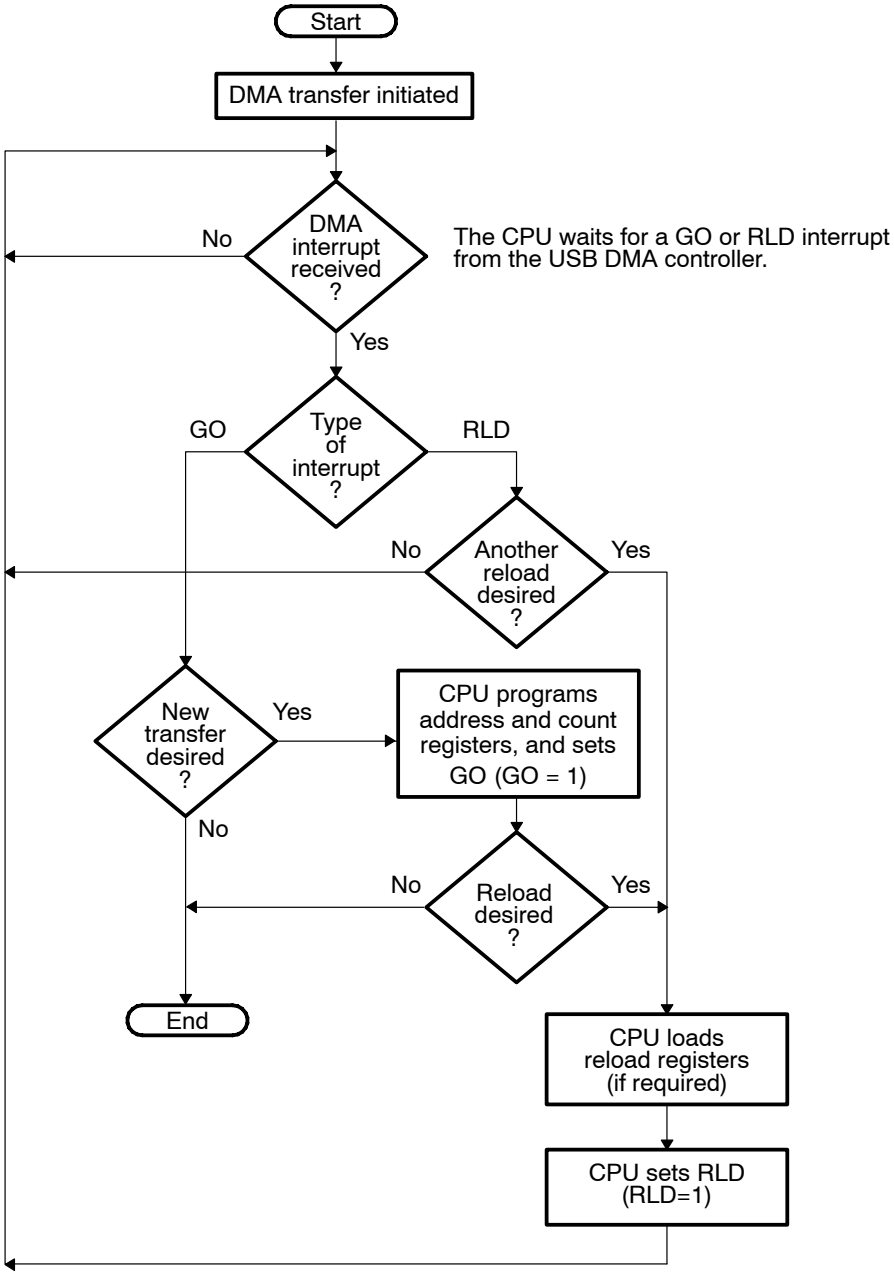


Figure 10. Activity for DMA Transfers (Continued)

(b) CPU Code Execution Flow (at an individual endpoint)



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.

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 7). 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.

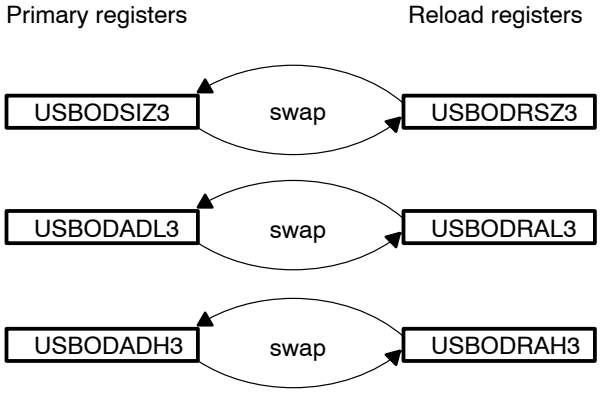
Table 7. Primary USB DMA Size and Address Registers and the Corresponding Reload Registers

Endpoint	Primary Register	Register Contains ...	Reload Register
Endpoint OUTn	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
Endpoint INn	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 6, 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 1 shows a reload operation involving the endpoint OUT3 DMA registers.

This register swapping saves CPU time if you repeatedly toggle between the same two blocks of 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 1. DMA Reload Operation for Endpoint OUT3



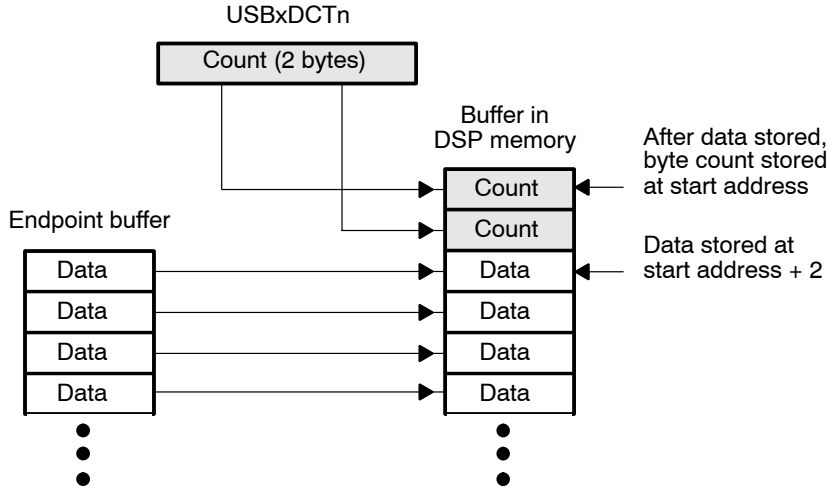
4.6 Transfer Count Saved to DSP Memory for an OUT Transfer

For each new DMA transfer, the USB DMA controller ensures that the transfer count for the endpoint starts at 0. When the user gives a GO command (GO = 1), the controller clears the endpoint count register (USBxDCTn) before moving the data. Likewise, after the controller completes a DMA reload operation (see section 4.5), it clears the count register before beginning the next DMA transfer.

At times, an OUT transfer may 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 the user can read the number of bytes in the packet. To prevent this loss of information, the controller copies the byte count to the DSP memory after every read from an endpoint buffer.

Figure 11 shows the positions of the data and the byte count in DSP memory. The start address is the address programmed in the primary address registers (USBxDADHn and USBxDADLn). 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 11. 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.

4.7 Configuring the USB DMA Controller

To configure an endpoint DMA transfer, 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 endpoint IN4.

4.7.1 Set the Transfer Size

Register(Field)	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 (see section 4.5 on page 42), the controller automatically clears USBxDCTn before beginning the next DMA transfer.

4.7.2 Set the DSP Memory Address

Register(Field)	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 general-purpose endpoint has a dedicated DMA channel, the USB DMA controller knows the location of the endpoint buffer, but you must tell the controller which address to use when accessing the DSP memory.

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 USBxDADLn is 0.

The control endpoints (OUT0 and IN0) do not have DMA channels.

4.7.3 Enable/Disable a DMA Reload Operation

Register(Field)	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)	1–65535	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 the RLD bit at the end of each DMA transfer to determine whether to stop or to begin another transfer. If you want the DMA controller to begin another transfer as soon as the current one ends, initialize the reload registers (USBxDRSZn, USBxDRAHn, and USBxDRALn) and then set the RLD bit. When the DMA controller is done with the first transfer and it finds RLD = 1, it performs a reload operation and begins the next transfer. If the DMA controller is stopped (GO = 0), setting the RLD bit has no effect.

Each time the DMA controller performs a reload operation, it clears the RLD bit 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 DMA 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).

4.7.4 Enable/Disable DMA Interrupt Requests

Register(Field)	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 endpoints OUT1–OUT7 are in USBODIE; those for endpoints IN1–IN7 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 endpoint IN6, write a 1 to USBIDIE(IE6). For endpoint OUT2, write a 1 to USBODIE(OE2).

For more details about the DMA interrupt requests, see section 5.3 on page 70.

4.7.5 Select the Endianness (Byte Orientation) of Data

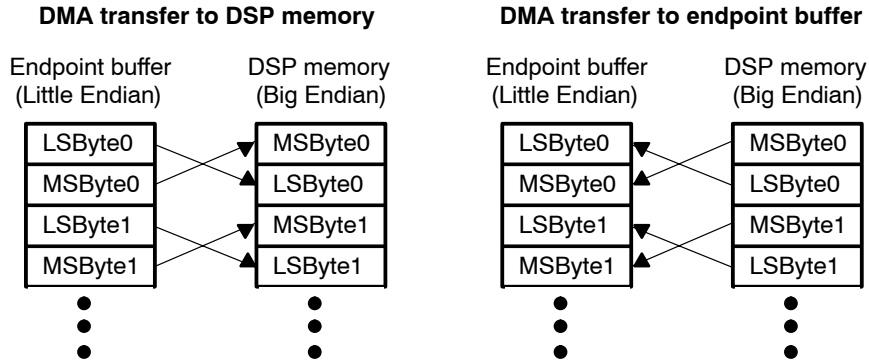
Register(Field)	Value	Description
USBxDCTLn(END)	0	Do not change the order of the bytes in the next DMA transfer.
	1	Reverse the endianness of each word moved in the next DMA transfer.

In the Big Endian orientation, 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.

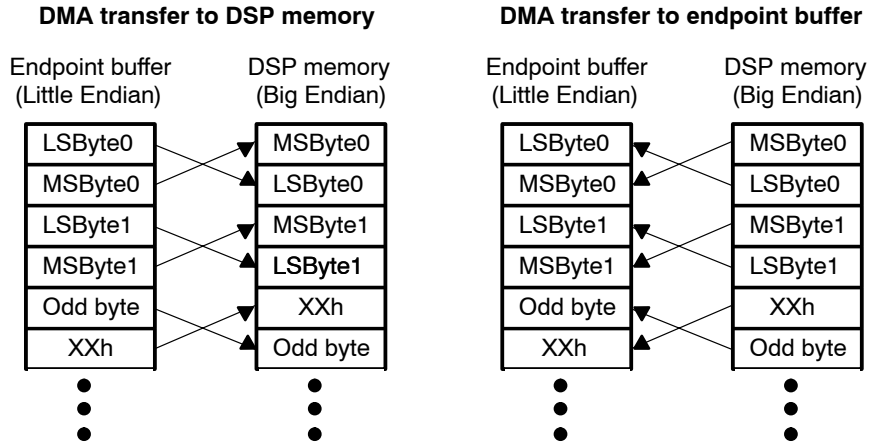
Figure 12 shows the effect of making END = 1 for an endpoint: The USB DMA controller reverses the endianness of the words it transfers between the DSP memory and the endpoint buffer. When END = 0 for an endpoint, the USB DMA controller does not change the order of the bytes.

Figure 12. The Effect of END = 1 on USB DMA Transfers

(a) Even Number of Bytes Transferred



(b) Odd Number of Bytes Transferred



4.7.6 Enable/Disable Concatenation

Register(Field)	Value	Description
USBxDCTLn(CAT)	0	Disable concatenation of DMA transfers.
	1	Enable concatenation of DMA transfers.

The USB DMA controller can combine multiple DMA transfers to service a single USB transfer. For more details, see the description for the CAT bit in section 7.2.1 (page 79).

4.7.7 Select Whether a Short Packet is Required to End a USB Transfer

Register(Field)	Value	Description
USBxDCTLn(SHT)	0	Short packet not required or expected
	1	Short packets required or expected

Typically, a USB transfer ends with a short packet, a packet that is shorter than the maximum allowable size for the endpoint. If the last packet is of the maximum packet size, 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 (or generate) a 0-byte packet when the transfer ends with a maximum-size packet. For more details, see the description for the SHT bit in section 7.2.1 (page 79).

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

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

If the USB DMA controller is handling data packets for an isochronous endpoint, this bit determines how the controller responds 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 7.2.1 (page 79).

4.8 Monitoring DMA Transfers

To monitor an endpoint DMA transfer, 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 endpoint OUT5.

4.8.1 Checking the Transfer Count

Register(Field)	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 initiated by the 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 the number of bytes that have been moved at any point during the DMA transfer.

4.8.2 Determining Whether a DMA Transfer is in Progress or is Done

Register(Field)	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 to the CPU 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 the GO bit. Instead the controller performs a DMA reload operation (see section 4.5 on page 42) 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 back-to-back transfers.

When the DMA controller completes a transfer and finds RLD = 0, it clears the GO bit. In addition, it sets the endpoint's GO flag in USBxDGIF and can generate an interrupt request. For example, if the DMA controller is done at endpoint OUT4, it sets the OE4 bit in USBODGIF. If the corresponding interrupt request is enabled by the OE4 bit in USBODIE, an interrupt request is sent to the CPU.

4.8.3 Determining Whether a DMA Reload Operation is in Progress or is Done

Register(Field)	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 to the CPU 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 DMA controller performs a reload operation. When the reload operation is done, the controller clears the RLD bit. In addition, it sets the endpoint's RLD flag in USBxDRIF and can generate an interrupt request.

For example, if the DMA controller completes a reload operation for endpoint OUT4, it sets the OE4 bit in USBODRIF. If the corresponding interrupt request is enabled by the OE4 bit in USBODIE, an interrupt request is sent to the CPU.

4.8.4 Checking for an Overflow or Underflow Condition

Register(Field)	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 7.2.1 (page 79).

4.8.5 Watching for a Missing Packet During an Isochronous Transfer

Register(Field)	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, you can program how the DMA controller should 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 7.2.1 (page 79).

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 15: Non-isochronous IN DMA Transfer (page 80)
- Table 9: Non-isochronous OUT DMA Transfer (page 54)
- Table 10: Isochronous IN DMA Transfer (page 56)
- Table 11: Isochronous OUT DMA Transfer (page 60)

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

- Figure 13: Missing Packet Response for Isochronous IN DMA Transfer (page 63)
- Figure 14: Missing Packet Response for Isochronous OUT DMA Transfer (page 64)

Table 8. 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/ 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	Yes		Yes	Yes			Idle
DMA transfer completion	Stop requested	> 0	0	1	X	X	X	Yes		Yes	Yes			Idle
Endpoint buffer full, more data remaining in DMA transfer		0	> 0	0	X	X	X					Yes	Max	Idle
DMA transfer completion, endpoint buffer full		0	0	0	0	1	X	Yes	No	Yes		Yes	Max	Idle
					1				Yes	No				
DMA transfer completion, endpoint buffer full		0	0	0	0	0	0	Yes	No	Yes		Yes	Max	Idle
					1				Yes	No				
DMA transfer completion, endpoint buffer full	Short packet requested	0	0	0	X	0	1					Yes	Max	Idle until current packet moves out, then prepare a 0-byte packet

Table 8. 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/ 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	Yes	No	Yes		Yes	Short	Idle
					1				Yes	No				
DMA transfer completion, endpoint buffer **not** full	CAT requested, next buffer is **not** ready yet	> 0	0	0	0	1	X	Yes		Yes		Yes		Pause
DMA transfer completion, endpoint buffer **not** full	CAT requested, next buffer is ready	> 0	0	0	1	1	X	Yes	Yes			Yes		In progress

Table 9. 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/ 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 complete	Stop requested	X	0	X	1	X	X	X	Yes		Yes	Yes		Yes	Yes	Idle
DMA transfer complete	Stop requested	X	> 0	0	1	X	X	X	Yes		Yes	Yes				Idle
Packet transfer complete, more data remaining in DMA transfer		Max	0	> 0	0	X	X	X						Yes	Yes	Idle
Packet transfer complete, more data remaining in DMA transfer		Short	0	> 0	0	0	X	X	Yes	No	Yes			Yes	Yes	Idle
						1				Yes	No					
DMA transfer complete, packet fits exactly		Max	0	0	0	0	X	0	Yes	No	Yes			Yes	Yes	Idle
						1				Yes	No					
DMA transfer complete, packet fits exactly		Max	0	0	0	0	1	X	Yes	No	Yes			Yes	Yes	Idle
						1				Yes	No					

Table 9. 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/ Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Clear NAK for Next Packet**	**DMA Transfer Activity**
DMA transfer complete, packet fits exactly	Short packet requested	Max	0	0	0	X	0	1						Yes	Yes	Idle, expecting a 0−byte packet
DMA transfer complete, packet fits exactly	Short packet requested	Short	0	0	0	0	X	X	Yes	No	Yes			Yes	Yes	Idle
						1				Yes	No					
DMA transfer complete, more data remaining in endpoint buffer	Overflow condition	X	> 0	0	0	X	0	X	Yes		Yes		Yes	Yes		Idle
DMA transfer complete, more data remaining in endpoint buffer	CAT requested, next buffer is not ready yet	X	> 0	0	0	0	1	X	Yes		Yes			Yes		Pause
DMA transfer complete, more data remaining in endpoint buffer	CAT requested, next buffer is ready	X	> 0	0	0	1	1	X	Yes	Yes				Yes		In progress

Table 10. State Table: Isochronous IN DMA Transfer

Description		Initial State								Final State							
DMA Transfer State	**Programmer 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 /Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Send Packet (Clear NAK)**	**DMA Transfer Activity**
DMA failed to keep up with USB		No	X	X	X	X	X	X	X	Yes		Yes		Yes			Idle
Normal transfer in progress		Yes	> 0	> 0	X	X	X	X	X								In progress
Endpoint buffer full, more data remaining in DMA transfer	Stop requested	Yes	0	> 0	X	1	X	X	X	Yes		Yes	Yes		Yes	Max	Idle
DMA transfer complete, endpoint buffer is **not** full	Stop requested	Yes	> 0	0	X	1	X	X	X	Yes		Yes	Yes				Idle
DMA transfer complete, endpoint buffer is full	Stop requested	Yes	0	0	X	1	X	X	X	Yes		Yes	Yes		Yes	Max	Idle

† Entries in this column are taken from the missing packet response state diagram of Figure 13 (page 63).

Table 10. State Table: Isochronous IN DMA Transfer (Continued)

Description		Initial State								Final State							
DMA Transfer State	**Programmer 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 /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		Yes	0	> 0	Yes	0	X	X	X	Yes		Yes			No	Max	Idle
Endpoint buffer full, more data remaining in DMA transfer		Yes	0	> 0	No	0	X	X	X						Yes	Max	Idle
DMA transfer complete, endpoint buffer is **not** full	Current buffer is the last of the transfer	Yes	> 0	0	No	0	0	0	X	Yes	No	Yes			Yes	Non-max packet	Idle
							1				Yes	No					

† Entries in this column are taken from the missing packet response state diagram of Figure 13 (page 63).

Table 10. State Table: Isochronous IN DMA Transfer (Continued)

Description		Initial State								Final State							
DMA Transfer State	**Programmer 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 /Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Send Packet (Clear NAK)**	**DMA Transfer Activity**
DMA transfer complete, endpoint buffer is **not** full	CAT requested, next buffer is not ready yet (underflow condition)	Yes	> 0	0	No	0	0	1	X	Yes		Yes		Yes	Yes	Non-max packet	Idle
DMA transfer complete, endpoint buffer is **not** full	CAT requested, next buffer is ready	Yes	> 0	0	No	0	1	1	X	Yes					Yes		Start next transfer, fill up rest of endpoint buffer
DMA transfer complete, endpoint buffer full, missing packet error seen		Yes	0	0	Yes	0	X	X	X	Yes		Yes					Idle
DMA transfer complete, endpoint buffer full		Yes	0	0	No	0	0	0	0	Yes	No	Yes			Yes	Max	Idle
							1				Yes	No					

† Entries in this column are taken from the missing packet response state diagram of Figure 13 (page 63).

Table 10. State Table: Isochronous IN DMA Transfer (Continued)

Description		Initial State								Final State							
DMA Transfer State	**Programmer 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 /Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Send Packet (Clear NAK)**	**DMA Transfer Activity**
DMA transfer complete, endpoint buffer full	Short (0-byte) packet requested	Yes	0	0	No	0	X	0	1						Yes	Short (zero)	Idle
DMA transfer complete, endpoint buffer full	CAT requested	Yes	0	0	No	0	0	1	X	Yes	No	Yes			Yes	Max	Idle
							1				Yes	No					

† Entries in this column are taken from the missing packet response state diagram of Figure 13 (page 63).

Table 11. State Table: Isochronous OUT DMA Transfer

Description		Initial State								Final State							
DMA Transfer State	**Programmer 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 /Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Receive Packet (Clear NAK)**	**DMA Transfer Activity**
DMA failed to keep up with USB		No	X	X	X	X	X	X	X	Yes		Yes		Yes			Idle
Normal transfer in progress		Yes	> 0	> 0	X	X	X	X	X								In progress
Endpoint buffer is empty	Stop requested	Yes	0	X	X	1	X	X	X	Yes		Yes	Yes		Yes	Yes	Idle
Endpoint buffer empty, more data remaining in DMA transfer		Yes	0	> 0	Normal	0	X	X	X						Yes	Yes	Idle
		Yes	0	> 0	Short	0	0	X	X	Yes	No	Yes			Yes	Yes	Idle
							1				Yes	No					
		Yes	0	> 0	Ignore	0	X	X	X							Yes	Idle
		Yes	0	> 0	Missing Error	0	X	X	X	Yes		Yes			Yes	Yes	Idle

† Entries in this column are taken from the missing packet response state diagram of Figure 14 (page 64).

Table 11. State Table: Isochronous OUT DMA Transfer (Continued)

Description		Initial State								Final State							
DMA Transfer State	**Programmer 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 /Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Receive Packet (Clear NAK)**	**DMA Transfer Activity**
DMA transfer complete, more data remaining in the endpoint buffer.	Stop requested	Yes	> 0	0	Normal	1	X	X	X	Yes		Yes	Yes			Yes	Idle
	Overflow condition	Yes	> 0	0	Normal	0	X	0	X	Yes		Yes		Yes	Yes	Yes	Idle
	CAT requested, next buffer is not ready yet (overflow condition)	Yes	> 0	0	Normal	0	0	1	X	Yes		Yes		Yes	Yes	Yes	Idle
	CAT requested, next buffer is ready	Yes	> 0	0	Normal	0	1	1	X	Yes	Yes				Yes	No	In progress

† Entries in this column are taken from the missing packet response state diagram of Figure 14 (page 64).

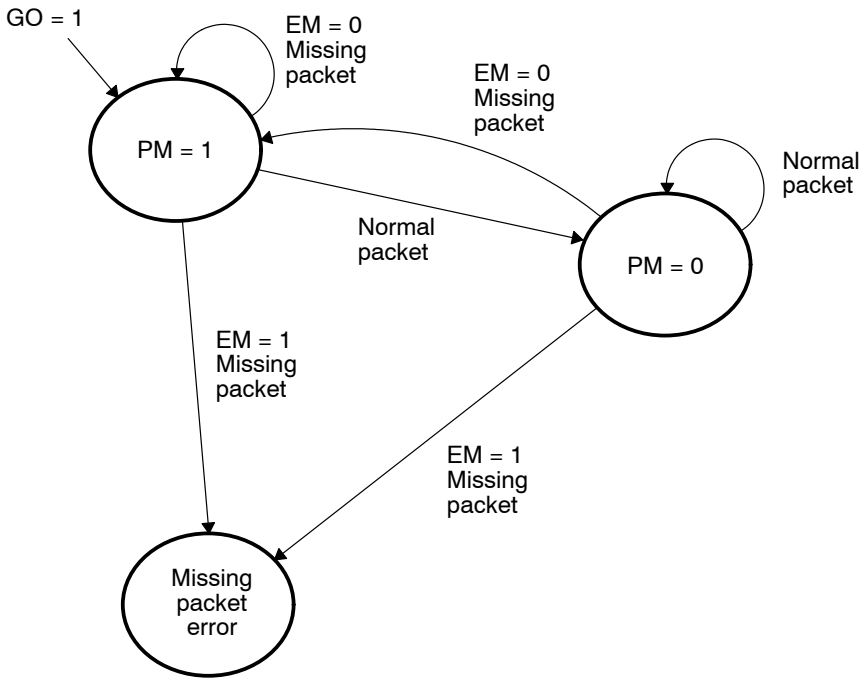
Table 11. State Table: Isochronous OUT DMA Transfer (Continued)

Description		Initial State								Final State							
DMA Transfer State	**Programmer 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 /Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Receive Packet (Clear NAK)**	**DMA Transfer Activity**
DMA transfer complete, endpoint buffer is empty	Stop requested	Yes	0	0	X	1	X	X	X	Yes		Yes	Yes		Yes	Yes	Idle
	CAT requested	Yes	0	0	Normal	0	0	1	X	Yes	No	Yes			Yes	Yes	Idle
							1				Yes	No					
		Yes	0	0	Normal	0	0	0	0	Yes	No	Yes			Yes	Yes	Idle
							1				Yes	No					
	Short (0-byte) packet expected	Yes	0	0	Normal	0	X	0	1						Yes	Yes	Expect a short packet next
DMA expecting a short (0-byte) packet to end the transfer		Yes	0	0	Missing Error	0	X	X	X	Yes		Yes			Yes	Yes	Idle
		Yes	0	0	Short	0	0	X	X	Yes	No	Yes			Yes	Yes	Idle
							1				Yes	No					
		Yes	0	0	Ignore	0	X	X	X							Yes	In progress

† Entries in this column are taken from the missing packet response state diagram of Figure 14 (page 64).

Figure 13. State Diagram: Missing Packet Response for Isochronous IN DMA Transfer

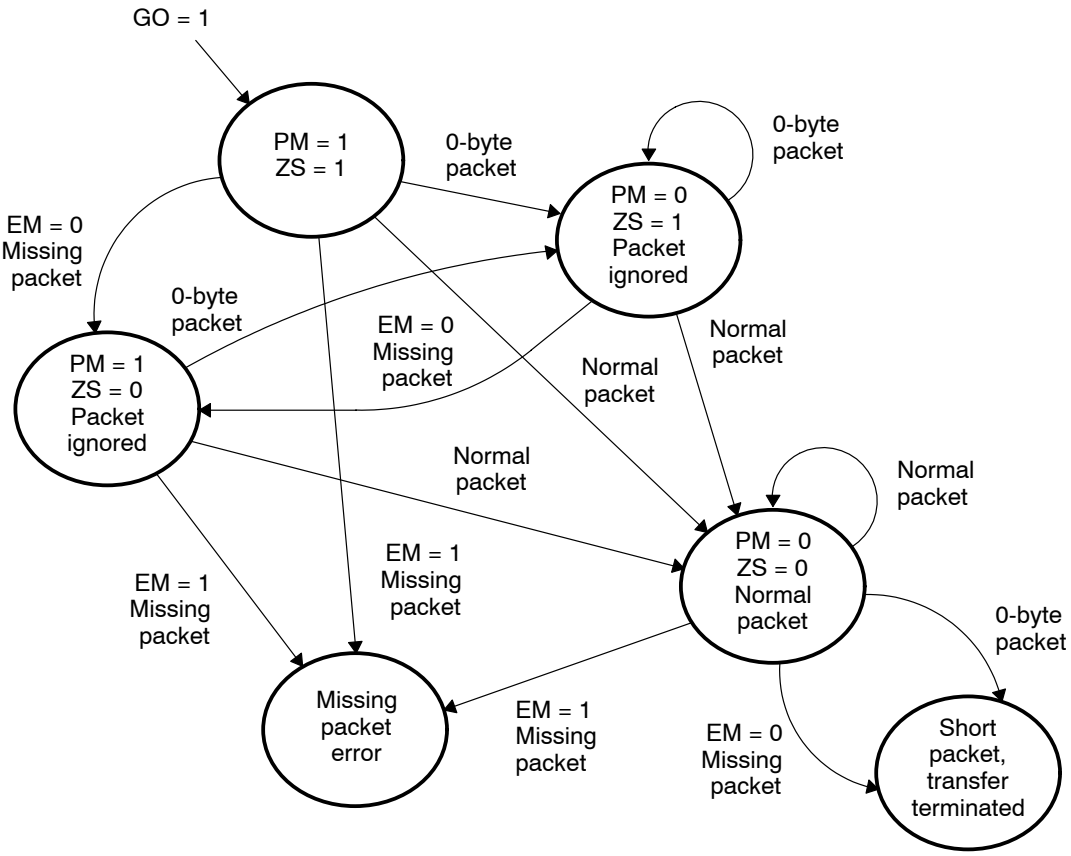
GO: Go command to USB DMA controller
PM: Previous packet missed
EM: Error if missing packet



Note: This figure supports Table 10 (page 56).

Figure 14. State Diagram: Missing Packet Response for Isochronous OUT DMA Transfer

GO: Go command to USB DMA controller
PM: Previous packet missed
EM: Error if missing packet
ZS: 0-byte packet status



Note: This figure supports Table 11 (page 60).

5 Interrupt Activity in the USB Module

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

- ❏ Bus interrupt requests (described in section 5.1)
- ❏ Endpoint interrupt requests (described in section 5.2)
- ❏ USB DMA interrupt requests (described in section 5.3)

As shown in Figure 15, all requests are multiplexed through an arbiter to a single USB interrupt request to 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. For the priority of each request, see the description of the interrupt source register (USBINTSRC) in section 7.5.1 (page 106).

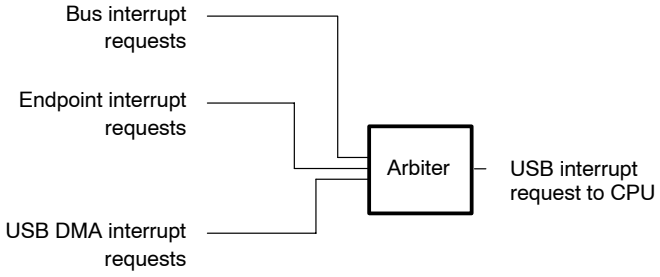
The USB interrupt is one of the maskable CPU interrupts. As with any maskable interrupt request, if it is properly enabled in the CPU, the CPU executes the corresponding interrupt service routine (ISR). The ISR that services the interrupt can determine the interrupt source by reading the interrupt source register (USBINTSRC), and can perform tasks accordingly.

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, the CPU must write a 1 to the bit.

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 15. Possible Sources of a USB Interrupt Request



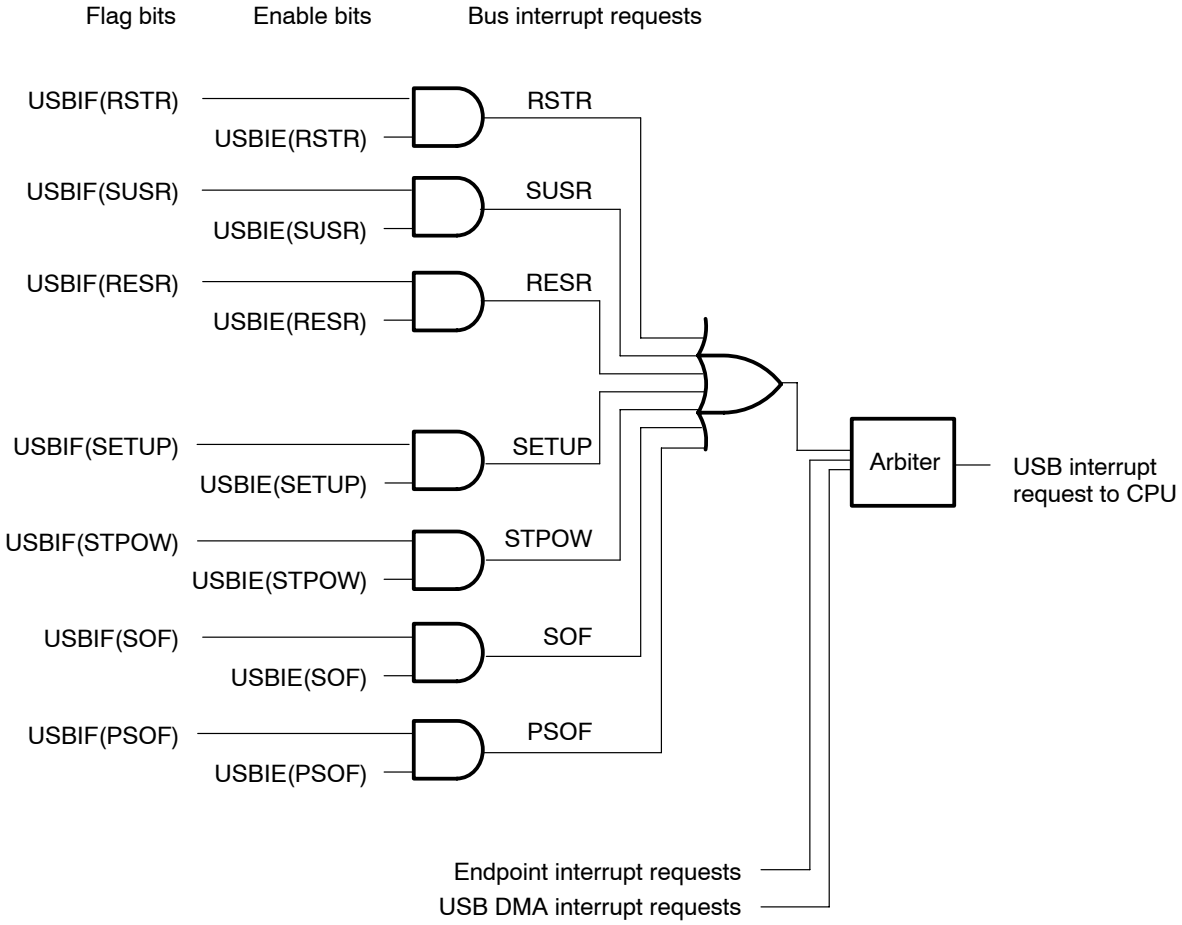
5.1 Bus Interrupt Requests

The USB module can generate a number of interrupt requests that are related to the activities on the Universal Serial Bus (see Table 12). As shown in Figure 16, 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, the corresponding 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 12. Descriptions of the Bus Interrupt Requests

Bus Interrupt Request	Interrupt Source
RSTR	A reset condition is detected on the bus.
SUSR	A suspend condition is detected on the bus.
RESR	Activity on the bus resumes, ending a suspend condition.
SETUP	A setup packet arrived. (Setup data is stored in the setup packet buffer.)
STPOW	A setup overwrite has occurred; that is, a new setup packet arrived before the previous setup packet was read from the setup packet buffer.
SOF	A start-of-frame (SOF) condition is detected on the bus.
PSOF	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 7.6.3 on page 117). The counter runs at 750 kHz (12MHz/16).

Figure 16. Enable Paths for the Bus Interrupt Requests



5.2 Endpoint Interrupt Requests

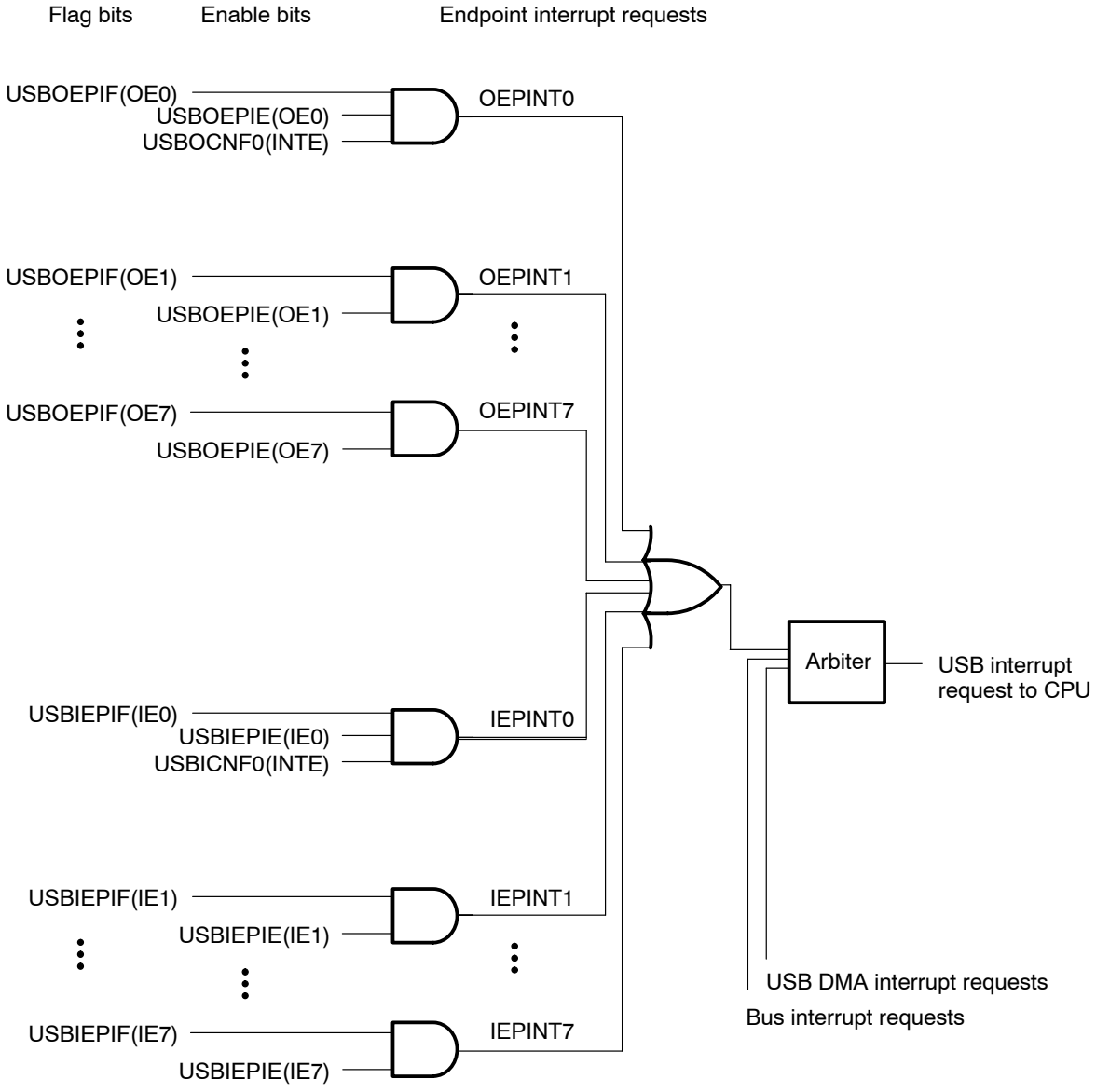
For each endpoint, the UBM can generate an interrupt request every time a packet moves in or out of the endpoint buffer. As shown in Figure 17:

- 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).
- Endpoints IN0 and OUT0 each have an additional interrupt enable bit, INTE, in the corresponding endpoint configuration register (USBICNF0 or USBOCNF0, respectively).
- For endpoints IN1–IN7 and OUT1–OUT7, when the flag bit is set and the enable bit is set, an interrupt request is passed to the CPU.
- For endpoint IN0 and OUT0, when the flag bit is set and both interrupt enable bits are set, an interrupt request is passed to the CPU.

Software must set the enable bit(s), 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. The UBM also sets the associated interrupt flag bit at this time.
- **For an IN endpoint (data to host):** When the USB module receives an IN packet, the UBM moves the data out of the appropriate IN endpoint buffer. The UBM then 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. The UBM also sets the associated interrupt flag bit at this time.

Figure 17. Enable Paths for the Endpoint Interrupt Requests



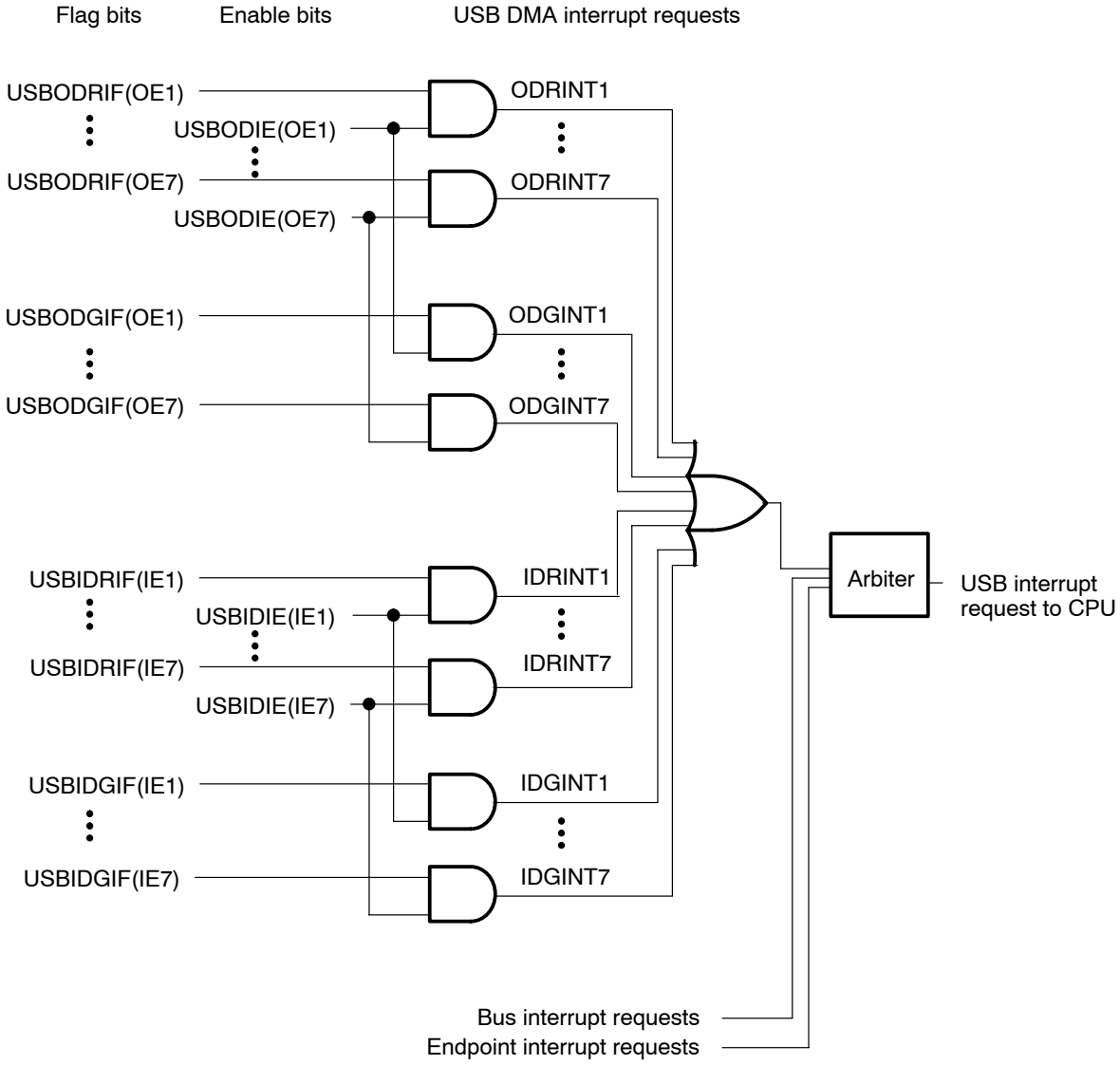
5.3 USB DMA Interrupt Requests

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 (OUT1−OUT7 and IN1−IN7). As shown in Figure 18:

- 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. Section 4.3 (page 38) explains how the GO and RLD bits control DMA activity.

Figure 18. Enable Paths for the USB DMA Interrupt Requests



6 Power, Emulation, and Reset Considerations

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

6.1 Putting the USB Module into Its Idle Mode

The USB module is one of the peripheral devices in the peripherals idle domain. 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 7.6.8 on page 123). This tells the DSP to make the USB module idle when the peripherals domain becomes idle.

2) Write a 1 to the PERI bit in ICR. This tells the DSP to make the peripherals domain idle in response to an IDLE instruction.

6.2 USB Module Indirectly Affected by Certain Idle Configurations

As mentioned in section 6.1, the USB module can be affected by any idle configuration that turns off the peripherals 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 wakes 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.

6.3 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 DMAGCR controls the emulation behavior of the DSP DMA controller. If FREE = 0 (the reset value), a breakpoint suspends DMA transfers. If FREE = 1, DMA transfers are not interrupted by a breakpoint.

6.4 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 7). One important effect is that the USB module is disconnected from the bus (CONN = 0 in USBCTL). When USBRST = 0, the CPU cannot access the USB module registers.
- 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.

7 USB Module Registers

This section covers the following topics

7.1 High-Level Summary of USB Module Registers

Table 13 lists the registers that are part of the USB module. These registers are in the I/O space of the C55x DSP. There are additional registers that also reside in I/O space but are not part of the USB module:

- USBDPLL. This register (available on TMS320VC5509 and TMS320VC5507/5509A devices) controls the operation of the digital phase-locked loop circuit (DPLL) of the USB clock generator. the dedicated USB clock generator. Details of USBDPLL are in section 2.4.2 (page 26).
- USBAPLL. This register (available on TMS320VC5507/5509A devices only) controls the operation of the analog phase-locked loop circuit (APLL) of the USB clock generator. Details of USBAPLL are in section 2.4.3 (page 30).
- USBPLLSEL. This register (available on TMS320VC5507/5509A devices only) is used to select between the DPLL and the APLL. Details of USBPLLSEL are in section 2.4.1 (page 24).
- USBIDLECTL. This register (available on TMS320VC5509 and TMS320VC5507/5509A devices) contains bits to put the USB module into its idle mode or into reset. Details about USBIDLECTL are in section 7.6.8 (page 123).

On each TMS320VC5509 or TMS320VC5507/5509A DSP, 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 13. For example, the base address on a TMS320VC5509 DSP is 5800h, and the DMA registers for endpoint OUT1 begin at I/O address 5808h.

Table 13. High-Level Summary of the USB Module Registers

I/O Address (Word Address)	Number of Registers	Width (Bits)	Description
Base address + 0000h	8	16	Reserved
Base address + 0008h	8	16	Endpoint OUT1 DMA register block
Base address + 0010h	8	16	Endpoint OUT2 DMA register block
Base address + 0018h	8	16	Endpoint OUT3 DMA register block
Base address + 0020h	8	16	Endpoint OUT4 DMA register block
Base address + 0028h	8	16	Endpoint OUT5 DMA register block
Base address + 0030h	8	16	Endpoint OUT6 DMA register block
Base address + 0038h	8	16	Endpoint OUT7 DMA register block
Base address + 0040h	8	16	Reserved
Base address + 0048h	8	16	Endpoint IN1 DMA register block
Base address + 0050h	8	16	Endpoint IN2 DMA register block
Base address + 0058h	8	16	Endpoint IN3 DMA register block
Base address + 0060h	8	16	Endpoint IN4 DMA register block
Base address + 0068h	8	16	Endpoint IN5 DMA register block
Base address + 0070h	8	16	Endpoint IN6 DMA register block
Base address + 0078h	8	16	Endpoint IN7 DMA register block

Table 13. 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	X and Y data buffers for endpoints OUT1−OUT7 and IN1−IN7
Base address + 0E80h	64	8	Endpoint OUT0 data buffer
Base address + 0EC0h	64	8	Endpoint IN0 data buffer
Base address + 0F00h	8	8	Setup packet buffer
Base address + 0F08h	8	8	Endpoint OUT1 definition register block
Base address + 0F10h	8	8	Endpoint OUT2 definition register block
Base address + 0F18h	8	8	Endpoint OUT3 definition register block
Base address + 0F20h	8	8	Endpoint OUT4 definition register block
Base address + 0F28h	8	8	Endpoint OUT5 definition register block
Base address + 0F30h	8	8	Endpoint OUT6 definition register block
Base address + 0F38h	8	8	Endpoint OUT7 definition register block
Base address + 0F40h	8	8	Reserved
Base address + 0F48h	8	8	Endpoint IN1 definition register block
Base address + 0F50h	8	8	Endpoint IN2 definition register block
Base address + 0F58h	8	8	Endpoint IN3 definition register block
Base address + 0F60h	8	8	Endpoint IN4 definition register block
Base address + 0F68h	8	8	Endpoint IN5 definition register block
Base address + 0F70h	8	8	Endpoint IN6 definition register block
Base address + 0F78h	8	8	Endpoint IN7 definition register block

Table 13. High-Level Summary of the USB Module Registers (Continued)

I/O Address (Word Address)	Number of Registers	Width (Bits)	Description
Base address + 0F80h	1	8	Endpoint IN0 configuration register
Base address + 0F81h	1	8	Endpoint IN0 byte count register
Base address + 0F82h	1	8	Endpoint OUT0 configuration register
Base address + 0F83h	1	8	Endpoint OUT0 byte count register
Base address + 0F84h	13	8	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	91	8	Reserved

Table 13. 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	1	8	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

7.2 DMA Registers

Each of the general-purpose endpoints (OUT1−OUT7 and IN1−IN7) has a dedicated DMA channel and a dedicated DMA register block for controlling and monitoring transfer activities in that channel. This section describes the function of each of the DMA registers, which are summarized in Table 14.

For each endpoint, the DMA register block starts at a different base address, but the individual registers are at the same offset from the base address. The first column of Table 14 shows the offsets.

Table 14. USB DMA Registers for Endpoint INn or OUTn (n = 1, 2, 3, 4, 5, 6, or 7)

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

7.2.1 USB DMA Control Register (USBxDCTLn) (x = I or O; 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 4.9 (page 51).

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 19. USB DMA Control Register (USBxDCTLn)

15						9	8
Reserved							PM
R–x							R–x
7	6	5	4	3	2	1	0
EM	SHT	CAT	END	OVF	RLD	STP	GO
R/W–x	R/W–x	R/W–x	R/W–x	R/W1C–0	R/W–0	R/W–0	R/W–0

Legend: R = Read; W = Write; W1C = Write 1 to clear (writing 0 has no effect); -*n* = Value after reset; -x = Value after reset is not defined

Table 15. Bits of a USB DMA Control Register (USBxDCTLn)

Bit	Field	Value	Description
15–9	Reserved	0	The read state of this field is undefined. It is recommended that you write 0s to these bit positions when writing to USBxDCTLn.
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 0-byte packets.
		1	Missing packets cause the GO bit to be cleared and the DMA to be halted. The error status is shown in the PM bit. This event occurs only when PM goes from 0 to 1.

Table 15. Bits of a USB DMA Control Register (USBxDCTLn) (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, the DMA controller does not insert an additional, 0-byte packet.
		1	If the size of the last packet in the transfer matches the maximum packet size, the DMA inserts a zero-length (0-byte) packet to terminate the transaction with a short packet. To insert a 0-byte packet, the DMA controller clears the endpoint byte count register, which forces both the byte count and the NAK bit to 0.
			For OUT transfers:
		0	If the size of the last packet in the transfer matches the maximum packet size, the DMA controller does 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, the DMA controller waits for an additional, 0-byte packet as an indication of the end of the transfer. If the next packet is not a 0-byte packet, the DMA controller ends the transfer and ignores the data in the terminating packet.
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 send out the data as a short packet.
		1	Concatenate DMA transfers. If the transfer size is not enough to fill a packet, then the DMA controller performs the next DMA transfer to fill the rest of 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, an overflow is recorded in the OVF bit.
		1	Concatenate DMA transfers. If the packet size exceeds the number of bytes remaining in the DMA transfer, an overflow is not recorded. Instead, the current position in the buffer is recorded. When the next DMA transfer starts, the DMA controllers reads the rest of the data, beginning at the recorded position.

Table 15. Bits of a USB DMA Control Register (USBxDCTLn) (Continued)

Bit	Field	Value	Description
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)
3	OVF		Overflow/underflow. For conditions that set this flag, see the state tables in section 4.9 (page 51).
			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	The DMA controller does not use the reload registers, and does not start a new DMA transfer at the end of the current transfer.
		1	The DMA controller swaps the contents of the primary and reload address and size registers, and starts a new transfer at the end of the current transfer.
1	STP		Stop DMA transfer.
		0	The DMA controller functions normally.
		1	The DMA controller stops at the end of the current DMA transfer and does not change the NAK bit. Because NAK is unchanged, the UBM does not move data into or out of the endpoint buffer. Once the DMA transfer is stopped, the GO and STP bits are cleared.

Table 15. Bits of a USB DMA Control Register (USBxDCTLn) (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 DMA controller is idling (the controller is available for a new transfer).
		1	Read 1 – The DMA controller is performing a transfer.
			Write 1 – The endpoint DMA transfer is started (STP bit must be 0).

7.2.2 USB DMA Address Registers (USBxDADHn and USBxDADLn) (x = I or O; 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 needs to supply only a start address for the data buffer in 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.

When moving data received during an OUT transfer, the DMA controller starts from address (DADH:DADL) + 2 and typically continues through address (DADH:DADL) + 2 + DSIZ, where DSIZ is the number of bytes to transfer. The DMA controller may stop sooner if 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.

For an IN transfer, the DMA controller starts at address (DADH:DADL) and continues through address (DADH:DADL) + DSIZ.

Figure 20. USB DMA Address Registers (USBxDADLn and USBxDADHn)

USBxDADLn

15	0
DADL (byte address)	
R/W−x	

USBxDADHn

15	0
DADH (byte address)	
R/W−x	

Legend: R = Read; W = Write; -x = Value after reset is not defined

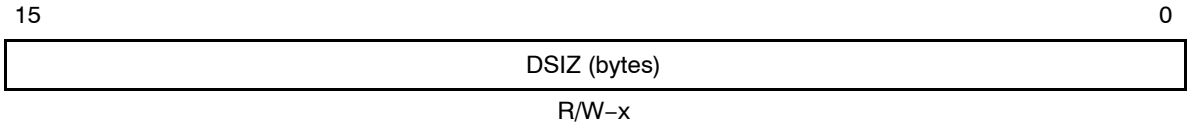
Table 16. Bits of USB DMA Address Registers (USBxDADLn and USBxDADHn)

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(15−0)	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.

7.2.3 USB DMA Size Register (USBxDSIZn) (x = I or O; 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 4.5 on page 42), the content of USBxDSIZn is swapped with the content of USBxDRSZn. USBxDRSZn is described in section 7.2.6 (page 87).

Figure 21. USB DMA Size Register (USBxDSIZn)



Legend: R = Read; W = Write; -x = Value after reset is not defined

Table 17. Bits of a USB DMA Size Register (USBxDSIZn)

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

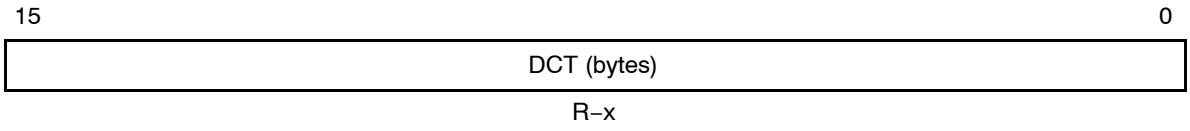
7.2.4 USB DMA Count Register (USBxDCTn) (x = I or O; n = 1, 2, 3, 4, 5, 6, or 7)

USBxDCTn counts up, to track the number of bytes that have been transferred for endpoint OUTn or INn. The USB DMA controller automatically loads this register with 0 before beginning each DMA transfer. This includes the transfer initiated by 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 4.6 on page 43).

Figure 22. USB DMA Count Register (USBxDCTn)



Legend: R = Read; -x = Value after reset is not defined

Table 18. Bits of a USB DMA Count Register (USBxDCTn)

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 resets this register to 0.

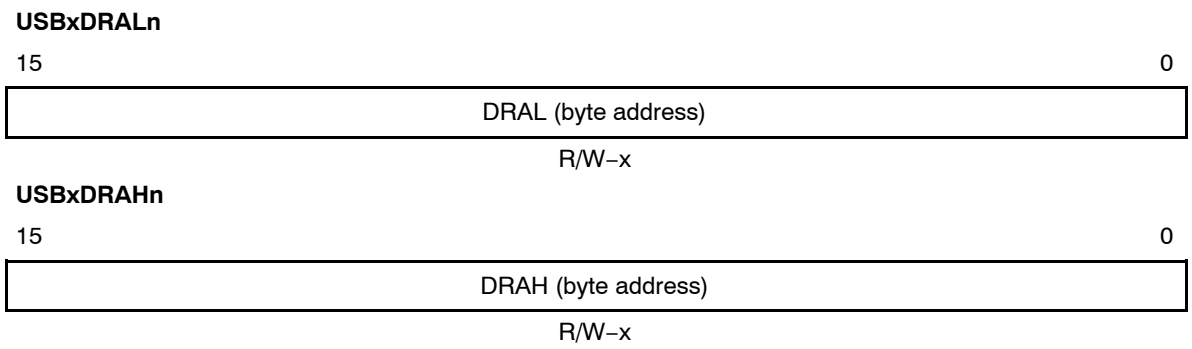
7.2.5 USB DMA Reload-Address Registers (USBxDRAHn and USBxDRALn) (x = I or O; n = 1, 2, 3, 4, 5, 6, or 7)

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, 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 4.5 (page 42).

Figure 23. USB DMA Reload-Address Registers (USBxDRALn and USBxDRAHn)



Legend: R = Read; W = Write; -x = Value after reset is not defined

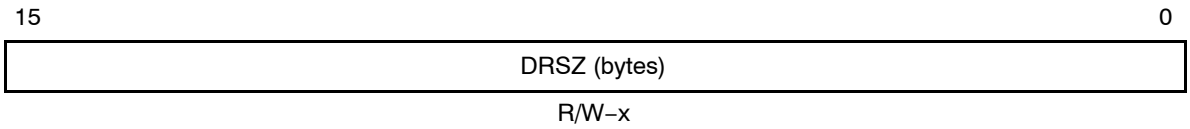
Table 19. Bits of USB DMA Reload-Address Registers (USBxDRALn and USBxDRAHn)

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.

7.2.6 USB DMA Reload-Size Register (USBxDRSZn) (x = I or O; 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 4.5 (page 42).

Figure 24. USB DMA Reload-Size Register (USBxDRSZn)



Legend: R = Read; W = Write; -x = Value after reset is not defined

Table 20. Bits of a USB DMA Reload-Size Register (USBxDRSZn)

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

7.3 Definition Registers for Endpoints IN1–IN7 and OUT1–OUT7

The general-purpose endpoints (IN1–IN7 and OUT1–OUT7) each have a block of eight definition registers to define the endpoint characteristics. Table 21 shows the definition registers available for OUT and IN endpoints. To access a definition register, find the base address of the definition block and add the offset given in the first column of Table 21. The actual addresses can be found in the device-specific data manual.

The isochronous mode (ISO) bit of each endpoint configuration register affects other fields in the endpoint configuration register and determines whether the USB module uses the extension registers (USBISIZHn, USBOCTXHn, and USBOCTYHn). Figure 25 shows the definition registers when endpoints are used in the isochronous mode (ISO = 1). Figure 26 shows the definition registers in the non-isochronous mode (ISO = 0).

Table 21. Definition Registers For Endpoint INn or OUTn (n = 1, 2, 3, 4, 5, 6, or 7)

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

Figure 25. Endpoint Definition Registers for INn and OUTn in the Isochronous Mode

Endpoint INn, Isochronous Mode (ISO = 1 in USBICNFn)

Offset	Register	7	6	5	3	2	0
0	USBICNFn	UBME	ISO = 1	CTXH		CTYH	
1	USBIBAXn	BAX					
2	USBICTXn	NAK	CTX				
3	USBISIZHn	Reserved†				SIZH	
4	USBISIZn	—†	SIZ				
5	USBIBAYn	BAY					
6	USBICTYn	NAK	CTY				
7	–	Reserved‡					

Endpoint OUTn, Isochronous mode (ISO = 1 in USBOCNFn)

Offset	Register	7	6	5	3	2	0
0	USBOCNFn	UBME	ISO = 1	Reserved†		SIZH	
1	USBOBAXn	BAX					
2	USBOCTXn	NAK	CTX				
3	USBOCTXHn	Reserved†				CTXH	
4	USBOSIZn	—†	SIZ				
5	USBOBAYn	BAY					
6	USBOCTYn	NAK	CTY				
7	USBOCTYHn	Reserved†				CTYH	

† Write 0s to these reserved bits.
‡ Do not use this register location.

Figure 26. Endpoint Definition Registers for INn and OUTn in the Non-Isochronous Mode

Endpoint INn, Non-isochronous Mode (ISO = 0 in USBICNFn)

Offset	Register	7	6	5	4	3	2	0
0	USBICNFn	UBME	ISO = 0	TOGGLE	DBUF	STALL	Reserved†	
1	USBIBAXn	BAX						
2	USBICTXn	NAK	CTX					
3	–	Reserved‡						
4	USBISIZn	—†	SIZ					
5	USBIBAYn	BAY						
6	USBICTYn	NAK	CTY					
7	–	Reserved‡						

Endpoint OUTn, Non-isochronous mode (ISO = 0 in USBOCNFn)

Offset	Register	7	6	5	4	3	2	0
0	USBOCNFn	UBME	ISO = 0	TOGGLE	DBUF	STALL	Reserved†	
1	USBOBAXn	BAX						
2	USBOCTXn	NAK	CTX					
3	–	Reserved‡						
4	USBOSIZn	—†	SIZ					
5	USBOBAYn	BAY						
6	USBOCTYn	NAK	CTY					
7	–	Reserved‡						

† Write 0s to these reserved bits.
‡ Do not use these register locations.

7.3.1 Endpoint Configuration Register for INn (USBICNFn) (n = 1, 2, 3, 4, 5, 6, or 7)

As shown in Figure 27, the function of bits 5−0 of USBICNFn depend on whether you have programmed the endpoint to operate in the non-isochronous or isochronous mode. Table 22 describes the bits of USBICNFn, taking into account the optional function of bits 5−0.

Figure 27. Endpoint Configuration Register for INn (USBICNFn)

USBICNFn in non-isochronous mode (ISO = 0)

7	6	5	4	3	2	0
UBME	ISO	TOGGLE	DBUF	STALL	Reserved†	
R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W−x	

USBICNFn in isochronous mode (ISO = 1)

7	6	5	3	2	0
UBME	ISO	CTXH		CTYH	
R/W-x	R/W-x	R/W-x		R/W-x	

Legend: R = Read; W = Write; -x = Value after reset is not defined
† Write 0s to these reserved bits.

Table 22. Bits of the Endpoint Configuration Register for INn (USBICNFn)

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 1.2 on page 17). **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 22. Bits of the Endpoint Configuration Register for INn (USBICNFn) (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 is sent to the host in response to host access requests until the STALL bit is cleared.
2–0	Reserved	0	Write 0s to these reserved bits.
			Bits 5–0 in Isochronous Mode (ISO = 1)
5–3	CTXH	000b–111b	X-buffer byte count high bits
2–0	CTYH	000b–111b	Y-buffer byte count high bits

7.3.2 Endpoint Configuration Register for OUTn (USBOCNFn) (n = 1, 2, 3, 4, 5, 6, or 7)

As shown in Figure 28, the function of bits 5–0 of USBOCNFn depend on whether you have programmed the endpoint to operate in the non-isochronous or isochronous mode. Table 23 describes the bits of USBOCNFn, taking into account the optional function of bits 5–0.

Figure 28. Endpoint Configuration Register for OUTn (USBOCNFn)

USBOCNFn in non-isochronous mode (ISO = 0)

7	6	5	4	3	2 – 0
UBME	ISO	TOGGLE	DBUF	STALL	Reserved†
R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W–x

USBOCNFn in isochronous mode (ISO = 1)

7	6	5 – 3	2 – 0
UBME	ISO	Reserved†	SIZH
R/W-x	R/W-x	R/W-x	R/W-x

Legend: R = Read; W = Write; -x = Value after reset is not defined
† Write 0s to these reserved bits.

Table 23. Bits of the Endpoint Configuration Register for OUTn (USBOCNFn)

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 1.2 on page 17). **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 23. Bits of the Endpoint Configuration Register for OUTn (USBOCNFn) (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 is sent to the host in response to host access requests until the STALL bit is cleared.
2–0	Reserved	0	Write 0s to these reserved bits.
		Bits 5–0 in Isochronous Mode (ISO = 1)	
5–3	Reserved	0	Write 0s to these reserved bits.
2–0	SIZH	000b–111b	X-/Y-buffer size high bits

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

Each general-purpose endpoint has two buffer base address registers: one for the X buffer and one for the Y buffer (see Figure 29 and Table 24). 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 2, which follows Table 24. 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 29. Endpoint Buffer Base Address Registers for INn or OUTn (USBxBAXn and USBxBAYn)

USBxBAXn

7							0
BAX (bits 11−4 of address offset)							
R/W−x							

USBxBAYn

7							0
BAY (bits 11−4 of address offset)							
R/W−x							

Legend: R = Read; W = Write; -x = Value after reset is not defined

Table 24. Bits of the Endpoint Buffer Base Address Registers for INn or OUTn (USBxBAXn and USBxBAYn)

Bit	Field	Value	Description
For endpoint INn:			
USBIBAXn(7−0)	BAX	00h−FFh	Bits 11−4 of the address offset for the X-buffer base address. Bits 3−0 are 0s. The X-buffer base address is the base address of the USB module registers plus this 12-bit offset.
USBIBAYn(7−0)	BAY	00h−FFh	Bits 11−4 of the address offset for the Y-buffer base address. Bits 3−0 are 0s. The Y-buffer base address is the base address of the USB module registers plus this 12-bit offset.
For endpoint OUTn:			
USBOBAXn(7−0)	BAX	00h−FFh	Bits 11−4 of the address offset for the X-buffer base address. Bits 3−0 are 0s. The X-buffer base address is the base address of the USB module registers plus this 12-bit offset.
USBOBAYn(7−0)	BAY	00h−FFh	Bits 11−4 of the address offset for the Y-buffer base address. Bits 3−0 are 0s. The Y-buffer base address is the base address of the USB module registers plus this 12-bit offset.

Example 2. Loading the Endpoint Buffer Base Addresses

Endpoint IN1 X buffer: Assigned to the 1st 64 bytes of the buffer RAM
Endpoint IN1 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)

7.3.4 Endpoint Buffer Count Registers for INn or OUTn (USBxCTXn, USBxCTYn) (x = I or O; n = 1, 2, 3, 4, 5, 6, or 7)

Each general-purpose endpoint has two count registers (see Figure 30 and Table 25): one for the X buffer and one for the Y buffer. The NAK bit corresponds to the negative acknowledgement (NAK) of the USB protocol. While the NAK bit is set (NAK = 1), the SIE sends a NAK in response to host data requests at the endpoint. More details about the NAK bit are given in section 3 (page 34).

Each buffer (X or Y) needs a count register to indicate how many bytes should move out of the buffer (for an IN endpoint) or how many bytes have been 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 25.

Figure 30. Endpoint Buffer Count Registers for INn or OUTn (USBxCTXn and USBxCTYn)

USBxCTXn

7	6 – 0
NAK	CTX (bytes)
R/W–x	R/W–x

USBxCTYn

7	6 – 0
NAK	CTY (bytes)
R/W–x	R/W–x

Legend: R = Read; W = Write; -x = Value after reset is not defined

Table 25. Bits of the Endpoint Buffer Count Registers for INn or OUTn (USBxCTXn and USBxCTYn)

Bit	Field	Value	Description
7	NAK		Negative acknowledgement
			For endpoint INn:
		0	Data in the endpoint buffer is ready for an IN transfer.
		1	Data in the endpoint buffer is not ready. The SIE sends a NAK in response to an IN token.
			For endpoint OUTn:
		0	The endpoint buffer is ready for an OUT transfer.
		1	Either the endpoint buffer is not ready or it contains an unread data packet from the previous transfer. The SIE sends 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 endpoint INn:** When NAK = 0, this value indicates the number of bytes to move out of buffer b in response to an IN token.
			For endpoint OUTn: This value is a running count of the bytes that have been moved to buffer b.
			Note: If CTb is programmed with a value greater than 64, the results will be unpredictable.
			In the isochronous mode:
		000 0000b–111 1111b	**For endpoint INn:** These bits are the 7 low bits of the 10-bit byte count for buffer b. As shown in Figure 31, the three high bits are taken from USBICNFn. Load these 10 bits with the number of bytes that should be moved out of buffer b in response to an IN token. When an IN token arrives and NAK = 0, this number of bytes is sent to the host.
			For endpoint OUTn: These bits are the seven low bits of the 10-bit byte counter for buffer b. As shown in Figure 32, the three high bits are taken from USBOCTXHn or USBOCTYHn. The 10-bit counter keeps a running count of the bytes that have been received in buffer b.

Figure 31. Endpoint Extended Buffer Count Values for INn in the Isochronous Mode (ISO = 1)

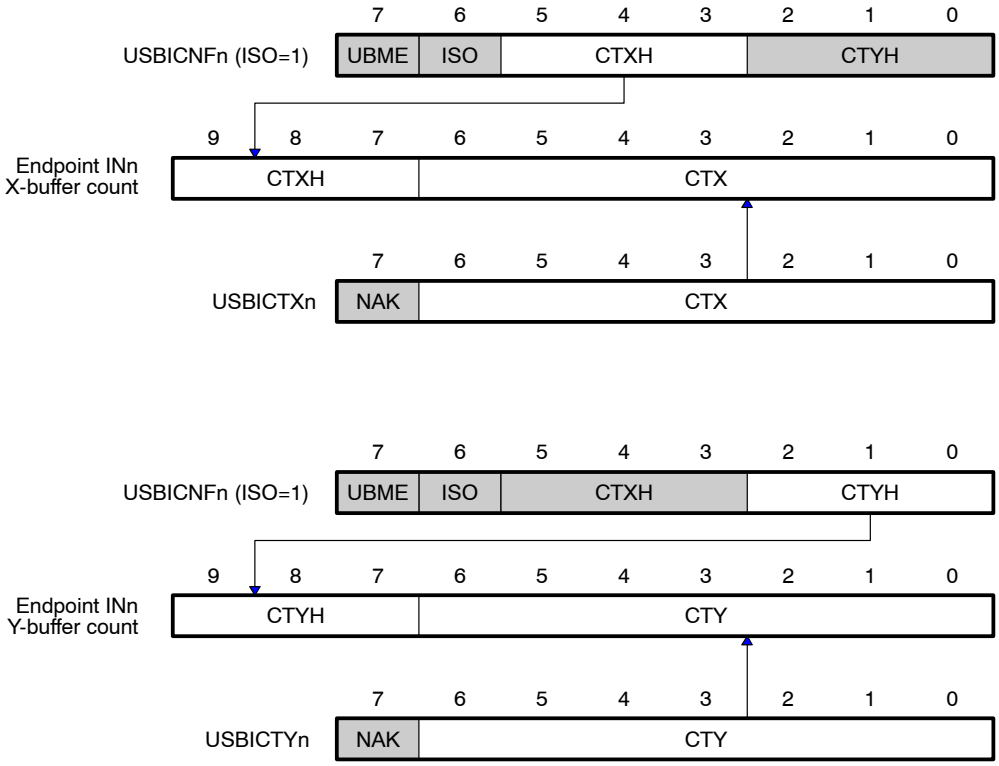
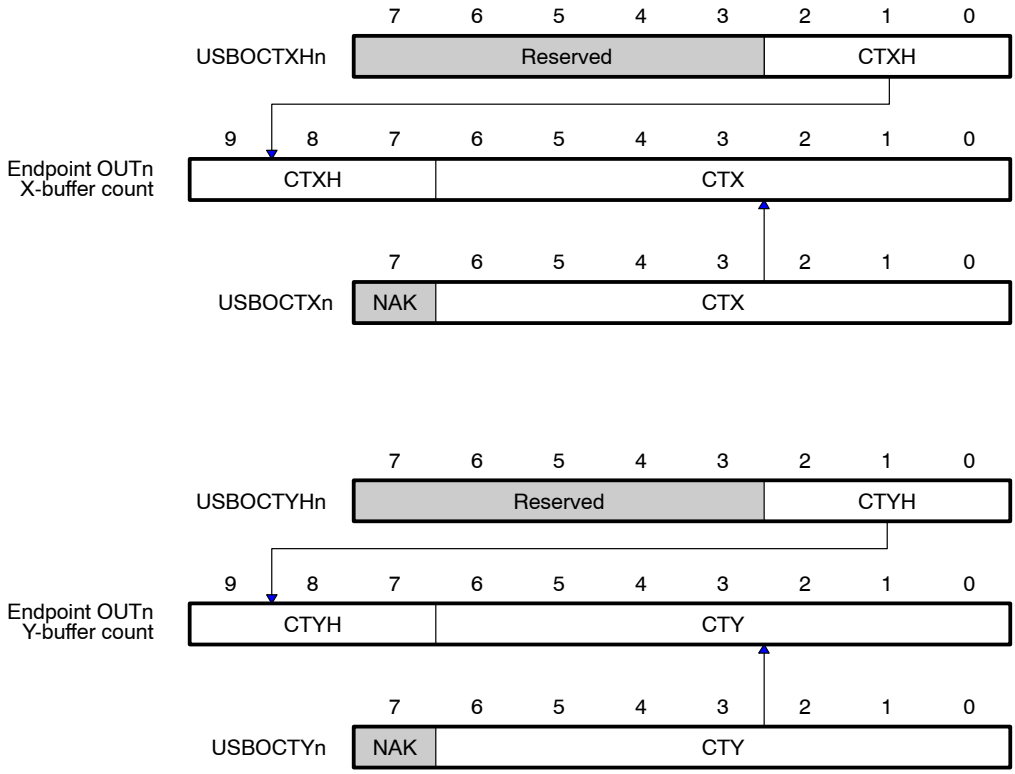


Figure 32. Endpoint Extended Buffer Count Values for OUTn in the Isochronous Mode (ISO = 1)



7.3.5 Endpoint X-/Y-Buffer Size Register for INn or OUTn (USBxSIZn) (x = I or O; 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 have the same size, which is defined by the SIZ field.

Figure 33. Endpoint X-/Y-Buffer Size Register for INn or OUTn (USBxSIZn)

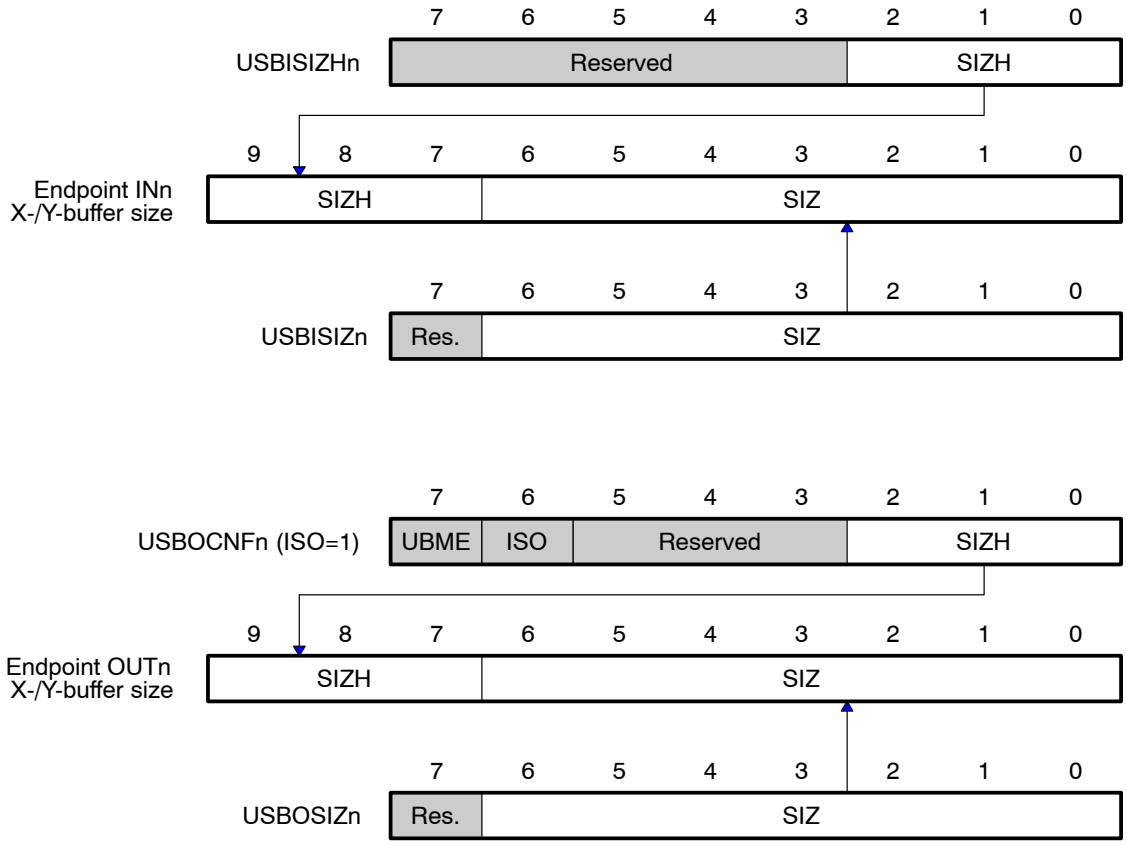
7	6 – 0
Reserved†	SIZ (bytes)
R/W–x	R/W–x

Legend: R = Read; W = Write; -x = Value after reset is not defined
† Write 0 to this reserved bit.

Table 26. Bits of the Endpoint n X-/Y-Buffer Size Register for INn or OUTn (USBxSIZn)

Bit	Field	Value	Description
7	Reserved	0	Write 0 to this reserved bit.
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 is programmed with a value greater than 64, the results will be unpredictable.
			In the isochronous mode:
		000 0001b–111 1111b	The seven low bits of the 10-bit buffer size. As shown in Figure 34, the three 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 34. Endpoint Extended Buffer Size Values for INn and OUTn in the Isochronous Mode (ISO = 1)



7.3.6 Endpoint Buffer Size and Count Extension Registers (USBISIZHn, USBOCTXHn, and 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 endpoint INn operates in 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 34 (page 101).
- If endpoint OUTn operates in 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 32 (page 99).

Figure 35. Endpoint Buffer Size and Count Extension Registers (USBISIZHn, USBOCTXHn, and USBOCTYHn)

USBISIZHn

7	3	2	0
Reserved†		SIZH	
R/W–x		R/W-x	

USBOCTXHn

7	3	2	0
Reserved†		CTXH	
R/W–x		R/W-x	

USBOCTYHn

7	3	2	0
Reserved†		CTYH	
R/W–x		R/W-x	

Legend: R = Read; W = Write; -x = Value after reset is not defined
† Write 0s to these reserved bits.

Table 27. Bits of the Endpoint Buffer Size and Count Extension Registers (USBISIZHn, USBOCTXHn, and USBOCTYHn)

Bit	Field	Value	Description
USBISIZHn(7–3)	Reserved	0	Write 0s to these reserved bits.
USBISIZHn(2–0)	SIZH	000h–111h	In the isochronous mode (ISO = 1), these are the 3 high bits of the IN endpoint X-/Y-buffer byte count. The 7 low bits are taken from the SIZ bits of USBISIZn.
USBOCTXHn(7–3)	Reserved	0	Write 0s to these reserved bits.
USBOCTXHn(2–0)	CTXH	000h–111h	In the isochronous mode (ISO = 1), these are the 3 high bits of the OUT endpoint X-buffer byte count. The 7 low bits are taken from the CTX bits of USBOCTXn.
USBOCTYHn(7–3)	Reserved	0	Write 0s to these reserved bits.
USBOCTYHn(2–0)	CTYH	000h–111h	In the isochronous mode (ISO = 1), these are the 3 high bits of the OUT endpoint Y-buffer byte count. The 7 low bits are taken from the CTY bits of USBOCTYn.

7.4 Definition Registers for Endpoints IN0 and OUT0

IN0 and OUT0 are the control endpoints. Each has a configuration register and a count register, which are described below.

7.4.1 Endpoint Configuration Register for IN0 or OUT0 (USBxCNF0) (x = I or O)

Endpoint IN0 and endpoint OUT0 each has a configuration register. The register fields are shown in Figure 36 and described in Table 28.

Figure 36. Endpoint Configuration Register for IN0 or OUT0 (USBxCNF0)

7	6	5	4	3	2	1	0
UBME	Reserved†	TOGGLE	Reserved†	STALL	INTE	Reserved†	
R/W-0	R/W–0	R-0	R/W–0	R/W-0	R/W–0	R/W–0	

Legend: R = Read; W = Write; -x = Value after reset is not defined
† Write 0s to these reserved bits.

Table 28. Bits of the Endpoint Configuration Register for IN0 or OUT0 (USBxCNF0)

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	0	Write 0 to this reserved bit.
5	TOGGLE		Endpoint data toggle. This bit reflects the data toggle sequence (see section 1.2 on page 17).
		0	The next data packet is DATA0.
		1	The next data packet is DATA1.
4	Reserved	0	Write 0 to this reserved bit.
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 is sent to the host 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	INTE	0	Endpoint interrupt enable
		0	The endpoint interrupt request is not enabled.
		1	If interrupt enable bit 0 of USBxEPIE is also 1, the endpoint interrupt request is enabled. For more details about endpoint interrupts, see section 5.2 on page 68.
1–0	Reserved	0	Write 0s to these reserved bits.

7.4.2 Endpoint Buffer Count Register for IN0 or OUT0 (USBxCT0) (x = I or O)

Endpoint IN0 and endpoint OUT0 each has one count register. As shown in Figure 37 and Figure 38, 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 29 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 SIE sends a NAK in response to host requests to the endpoint. More details about the NAK bit are given in section 3 (page 34).

The CT0 field indicates how many bytes of data should be moved out of the endpoint buffer (for IN0) or how many bytes of data have been moved into the endpoint buffer (for OUT0).

Figure 37. Endpoint Buffer Count Register for IN0 (USBICT0)

7	6	0
NAK	CT0 (bytes)	
R/W–1	R/W–0	

Legend: R = Read; W = Write; -*n* = Value after reset

Figure 38. Endpoint Buffer Count Register for OUT0 (USBOCT0)

7	6	0
NAK	CT0 (bytes)	
R/W–0	R–0	

Legend: R = Read; W = Write; -*n* = Value after reset

Table 29. Bits of the Endpoint Buffer Count Register for IN0 or OUT0 (USBxCT0)

Bit	Field	Value	Description
7	NAK		Negative acknowledgement
			For endpoint IN0:
		0	Data in the endpoint buffer is ready for an IN transfer.
		1	Data in the endpoint buffer is not ready. The SIE sends a NAK in response to an IN token.
			For endpoint OUT0:
		0	The endpoint buffer is ready for an OUT transfer.
		1	Either the endpoint buffer is not ready or it contains an unread data packet from the previous transfer. The SIE sends a NAK in response to an OUT token.
6–0	CT0		Count bits
		0–64	**For endpoint IN0:** When NAK = 0, this value indicates the number of bytes of data to be moved out of the endpoint buffer in response to an IN token.
			For endpoint OUT0: This value is a running count of the data bytes that have been received in the endpoint buffer.
			Note: If CT0 is programmed with a value greater than 64, the results will be unpredictable.

7.5 Interrupt Registers

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

7.5.1 Interrupt Source Register (USBINTSRC)

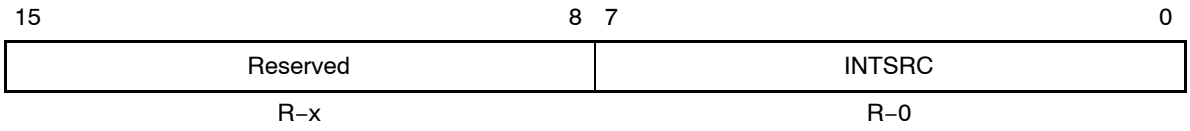
All interrupt requests generated in the USB module are multiplexed through an arbiter to a single USB interrupt request to the CPU. The interrupt service routine can determine the interrupt source by reading the interrupt source register (USBINTSRC). Then the ISR can branch to the appropriate code section.

When the CPU reads the INTSRC field (see Figure 39 and Table 30), it obtains a 7-bit interrupt source code. Table 31 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 5 on page 65.

Figure 39. Interrupt Source Register (USBINTSRC)



Legend: R = Read; -*n* = Value after reset; –x = Value after reset is not defined

Table 30. Bits of the Interrupt Source Register (USBINTSRC)

Bit	Field	Value	Description
15–8	Reserved		The read state of this field is undefined.
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 31. Interrupt Sources Matched to INTSRC Values

INTSRC Value/ Priority	Interrupt Source	INTSRC Value/ Priority	Interrupt Source
00h	(No interrupt)	36h	Endpoint OUT3 DMA reload
02h	Endpoint OUT0	37h	Endpoint OUT3 DMA GO
04h	Endpoint IN0	38h	Endpoint OUT4 DMA reload
06h	RSTR interrupt	39h	Endpoint OUT4 DMA GO
08h	SUSR interrupt	3Ah	Endpoint OUT5 DMA reload
0Ah	RESR interrupt	3Bh	Endpoint OUT5 DMA GO
0Ch	Setup packet received	3Ch	Endpoint OUT6 DMA reload
0Eh	Setup packet overwrite	3Dh	Endpoint OUT6 DMA GO
10h	SOF	3Eh	Endpoint OUT7 DMA reload
11h	PSOF	3Fh	Endpoint OUT7 DMA GO
12h	Endpoint OUT1	42h	Endpoint IN1 DMA reload
14h	Endpoint OUT2	43h	Endpoint IN1 DMA GO
16h	Endpoint OUT3	44h	Endpoint IN2 DMA reload
18h	Endpoint OUT4	45h	Endpoint IN2 DMA GO
1Ah	Endpoint OUT5	46h	Endpoint IN3 DMA reload
1Ch	Endpoint OUT6	47h	Endpoint IN3 DMA GO
1Eh	Endpoint OUT7	48h	Endpoint IN4 DMA reload
22h	Endpoint IN1	49h	Endpoint IN4 DMA GO
24h	Endpoint IN2	4Ah	Endpoint IN5 DMA reload
26h	Endpoint IN3	4Bh	Endpoint IN5 DMA GO
28h	Endpoint IN4	4Ch	Endpoint IN6 DMA reload
2Ah	Endpoint IN5	4Dh	Endpoint IN6 DMA GO
2Ch	Endpoint IN6	4Eh	Endpoint IN7 DMA reload
2Eh	Endpoint IN7	4Fh	Endpoint IN7 DMA GO
32h	Endpoint OUT1 DMA reload	Other	Reserved
33h	Endpoint OUT1 DMA GO		
34h	Endpoint OUT2 DMA reload		
35h	Endpoint OUT2 DMA GO		

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

The USB module sets an interrupt flag in USBIEPIF every time a data packet is moved out of an IN endpoint buffer and sets an interrupt flag in USBOEPIF every time a data packet is moved into an OUT endpoint. For example, if a new packet of data is moved into the OUT2 buffer, the OE2 flag is set in USBOEPIF. In addition to setting a flag, the USB module can send an endpoint interrupt request to the CPU. For more details about endpoint interrupt requests, see section 5.2 on page 68.

Figure 40. OUT Endpoint Interrupt Flag Register (USBOEPIF)

15							8
Reserved†							
R/W–x							
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

Legend: R = Read; W1C = Write 1 to clear (writing 0 has no effect); *-n* = Value after reset; –x = Value after reset is not defined
† Write 0s to these reserved bits.

Figure 41. IN Endpoint Interrupt Flag Register (USBIEPIF)

15							8
Reserved†							
R/W–x							
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

Legend: R = Read; W1C = Write 1 to clear (writing 0 has no effect); *-n* = Value after reset; –x = Value after reset is not defined
† Write 0s to these reserved bits.

Table 32. Bits of an Endpoint Interrupt Flag Register (USBxEPIF)

Bit	Field	Value	Description
15–8	Reserved	0	Write 0s to these reserved bits.
7–0	xE[7:0]		Endpoint interrupt flag bits. x = O (for OUT) or I (for IN).
			For each bit, xE7 through xE0:
		0	No interrupt pending
		1	Indicates that the corresponding endpoint generated an interrupt. For example, if OE7 = 1, endpoint OUT7 generated an interrupt. Each flag bit is set by the hardware and 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 the flag bit.

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

USBOEPIE contains interrupt enable bits for endpoints OUT0–OUT7, and USBIEPIE contains interrupt enable bits for endpoints IN0–IN7. Endpoints IN0 and OUT0 each have an additional interrupt enable bit (called INTE) in the endpoint configuration register (USBICNF0 or USBOCNF0, respectively).

For one of the endpoints IN1–IN7 or OUT1–OUT7, if the flag bit gets set in the endpoint interrupt flag register (USBxEPIF) and the corresponding bit is 1 in USBxEPIE, an endpoint interrupt request is sent to the CPU. For example, if the IE2 bit of USBIEPIF gets set and the IE2 bit of USBIEPIE is 1, an endpoint IN2 interrupt request is generated.

For endpoint IN0 or OUT0, the bit in USBxCNF0 must be 1 in additional to the flag bit in USBxEPIF and the interrupt enable bit in USBxEPIE.

More details about endpoint interrupt requests are in section 5.2 (page 68).

Figure 42. OUT Endpoint Interrupt Enable Register (USBOEPIE)

15							8
Reserved†							
R/W–x							
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

Legend: R = Read; W = Write; -*n* = Value after reset; –x = Value after reset is not defined
† Write 0s to these reserved bits.

Figure 43. IN Endpoint Interrupt Enable Register (USBIEPIE)

15							8
Reserved†							
R/W–x							
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

Legend: R = Read; W = Write; -*n* = Value after reset; –x = Value after reset is not defined
† Write 0s to these reserved bits.

Table 33. Bits of an Endpoint Interrupt Enable Register (USBxEPIE)

Bit	Field	Value	Description
15–8	Reserved	0	Write 0s to these reserved bits.
7–1	xE[7:1]		Endpoint interrupt enable bits 7–1. x = O (for OUT) or I (for IN).
			For each bit, xE7 through xE1:
		0	Endpoint interrupt requests are disabled for the corresponding endpoint.
		1	Endpoint interrupt requests are enabled for the corresponding endpoint. For example, if IE5 = 1 in USBIEPIE, endpoint interrupt requests are enabled for endpoint IN5.
0	xE0		Endpoint interrupt enable bit 0. x = O (for OUT) or I (for IN).
		0	Endpoint interrupt requests are disabled for the corresponding endpoint.
		1	If the interrupt enable bit (INTE) of USBxCNF0 is also 1, endpoint interrupt requests are enabled for the corresponding endpoint. For example, if IE0 = 1 in USBIEPIE and INTE = 1 in USBICNF0, endpoint interrupt requests are enabled for endpoint IN0.

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

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 DMA controller is idling after servicing endpoint OUT6, the OE6 bit is set in USBODGIF. In addition to setting the flag, the USB module can send a DMA GO interrupt request to the CPU. For details on DMA GO interrupt requests, see section 5.3 on page 70.

Figure 44. OUT Endpoint DMA GO Interrupt Flag Register (USBODGIF)

15							8
Reserved†							
R/W–x							
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	R/W–0

Legend: R = Read; W1C = Write 1 to clear (writing 0 has no effect); -*n* = Value after reset; –x = Value after reset is not defined
† Write 0s to these reserved bits.

Figure 45. IN Endpoint DMA GO Interrupt Flag Register (USBIDGIF)

15							8
Reserved†							
R/W–x							
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	R/W–0

Legend: R = Read; W1C = Write 1 to clear (writing 0 has no effect); -*n* = Value after reset; –x = Value after reset is not defined
† Write 0s to these reserved bits.

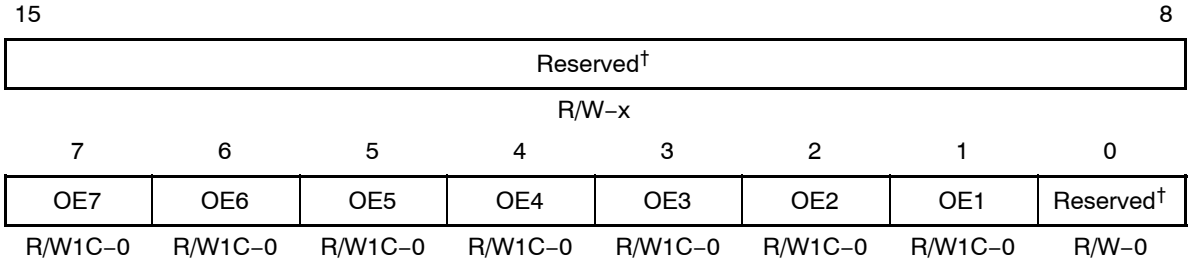
Table 34. Bits of an Endpoint DMA GO Interrupt Flag Register (USBxDGIF)

Bit	Field	Value	Description
15–8	Reserved	0	Write 0s to these reserved bits.
7–1	xE[7:1]		DMA GO interrupt flag bits. x = O (for OUT) or I (for IN). The USB DMA controller sets a DMA GO interrupt flag bit to indicate that the corresponding DMA transfer is complete. 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 the flag bit. For each bit, xE7 through xE1:
		0	GO interrupt not pending
		1	GO interrupt pending
0	Reserved	0	Write 0 to this reserved bit.

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

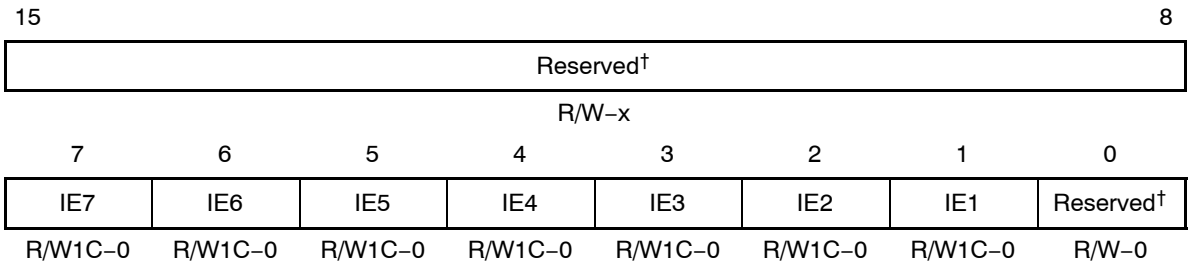
At the completion of a DMA transfer, if RLD = 1, the USB DMA controller clears the RLD bit of the endpoint, and the corresponding RLD interrupt flag is set in USBxDRIF. For example, when the DMA controller performs a reload operation for endpoint IN7, the IE7 bit is set in USBIDRIF. In addition to setting the flag, the USB module can send a DMA RLD interrupt request to the CPU. For details on DMA RLD interrupt requests, see section 5.3 on page 70.

Figure 46. OUT Endpoint DMA RLD Interrupt Flag Register (USBODRIF)



Legend: R = Read; W1C = Write 1 to clear (writing 0 has no effect); -*n* = Value after reset; –x = Value after reset is not defined
† Write 0s to these reserved bits.

Figure 47. IN Endpoint DMA RLD Interrupt Flag Register (USBIDRIF)



Legend: R = Read; W1C = Write 1 to clear (writing 0 has no effect); -*n* = Value after reset; –x = Value after reset is not defined
† Write 0s to these reserved bits.

Table 35. Bits of an Endpoint DMA RLD Interrupt Flag Register (USBxDRIF)

Bit	Field	Value	Description
15–8	Reserved	0	Write 0s to these reserved bits.
7–1	xE[7:1]		DMA RLD interrupt flag bits. x = O (for OUT) or I (for IN). The DMA RLD interrupt flag bit is set for an endpoint when the USB DMA controller completes a DMA reload operation at that endpoint (for DMA reload details, see section 4.5 on page 42). 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 the flag bit. For each bit, xE7 through xE1:
		0	RLD interrupt not pending
		1	RLD interrupt pending
0	Reserved	0	Write 0 to this reserved bit.

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

USBxDIE enables or disables both the DMA GO and DMA RLD interrupt requests. For example, if IE2 = 1 in USBIDIE, both DMA GO and DMA RLD interrupt requests are enabled for endpoint IN2. If IE2 = 0 in USBIE, both DMA GO and DMA RLD interrupt requests are disabled for endpoint IN2, regardless of the values in the corresponding interrupt flag registers (USBIDGIF and USBIDRIF). For more details about DMA interrupt requests, see section 5.3 on page 70.

Figure 48. OUT Endpoint DMA Interrupt Enable Register (USBODIE)

15							8
Reserved†							
R/W–x							
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	R/W–0

Legend: R = Read; W = Write; -*n* = Value after reset; –x = Value after reset is not defined
† Write 0s to these reserved bits.

Figure 49. IN Endpoint DMA Interrupt Enable Register (USBIDIE)

15							8
Reserved†							
R/W–x							
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	R/W–0

Legend: R = Read; W = Write; -*n* = Value after reset; –x = Value after reset is not defined
† Write 0s to these reserved bits.

Table 36. Bits of an Endpoint DMA Interrupt Enable Register (USBxDIE)

Bit	Field	Value	Description
15–8	Reserved	0	Write 0s to these reserved bits.
7–1	xE[7:1]		DMA interrupt enable bits. x = O (for OUT) or I (for IN). Each of these bits enables or disables both the DMA GO and DMA RLD interrupt requests for the endpoint. For each bit, xE7 through xE1:
		0	GO and RLD interrupt requests are disabled for the endpoint.
		1	GO and RLD interrupt requests are enabled for the endpoint.
0	Reserved	0	Write 0 to this reserved bit.

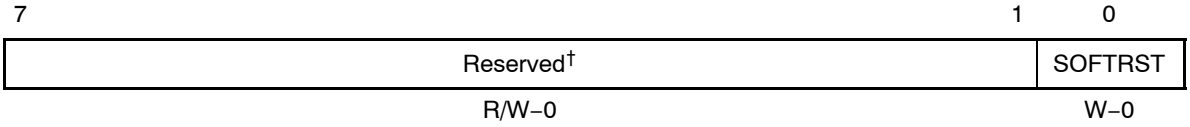
7.6 General Control and Status Registers

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 bus, and tracking USB frame numbers.

7.6.1 Global Control Register (USBGCTL)

The SOFTRST bit in USBGCTL can be used to reset the USB module without resetting the entire DSP.

Figure 50. Global Control Register (USBGCTL)



Legend: R = Read; W = Write; -*n* = Value after reset
† Write 0s to these reserved bits.

Table 37. Bits of the Global Control Register (USBGCTL)

Bit	Field	Value	Description
7−1	Reserved	0	Write 0s to these reserved bits.
0	SOFTRST		Software reset
		1	Reset the USB module. The effects are the same as a DSP hardware reset: All of the USB module registers assume their power-on default values except for USBCTL. As a result of the software reset, the USB module is inactive and is disconnected from the bus. As soon as the reset is complete, SOFTRST is cleared to 0, and the CPU is free to reprogram and run the USB module.

7.6.2 Frame Number Registers (USBFNUML and USBFNUMH)

The registers shown in Figure 51 track the current USB frame number. The 3 MSBs are in USBFNUMH and the 8 LSBs are in USBFNUML. The 11-bit value increments up to 2047 and then rolls over to 0 and starts counting up again.

Figure 51. Frame Number Registers (USBFNUML and USBFNUMH)

USBFNUMH

7	3	2	0
Reserved		FNUMH	
R–x		R–0	

USBFNUML

7	0
FNUML	
R–0	

Legend: R = Read; -*n* = Value after reset; –x = Value after reset is not defined

Table 38. Bits of the Frame Number Registers (USBFNUML and USBFNUMH)

Register(Bit)	Field	Value	Description
USBFNUMH(7–3)	Reserved		The read state of this reserved field is undefined.
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.

7.6.3 PSOF Interrupt Timer Counter (USBPSOFTMR)

USBPSOFTMR is shown in Figure 52 and described in Table 39.

A start-of-frame (SOF) token is expected on the USB every 1 ms. 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 comes from a pre-SOF (PSOF) interrupt request. If the USBPSOFTMR is programmed with a value *n* and the PSOF interrupt request is enabled, the USB module generates a PSOF interrupt request n clock cycles ahead of the expected arrival of the SOF token. The PSOF timer runs at 750 kHZ.

Figure 52. PSOF Interrupt Timer Counter (USBPSOFTMR)

7							0
PSOFTMR							
R/W−0							

Legend: R = Read; W = Write; -*n* = Value after reset

Table 39. Bits of the PSOF Interrupt Timer Counter (USBPSOFTMR)

Bit	Field	Value	Description
7−0	PSOFTMR	0−255	Indicates the number of clock cycles a PSOF interrupt should precede 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.

7.6.4 USB Control Register (USBCTL)

USBCTL enables and controls the features that are described in Table 40. This register is not affected during a reset operation that is initiated with the SOFTRST bit of USBGCTL. This register is affected by other reset operations.

Figure 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	R/W−0

Legend: R = Read; W = Write; -*n* = Value after reset
† Write 0s to these reserved bits.

Table 40. Bits of the USB Control Register (USBCTL)

Bit	Field	Value	Description
7	CONN		Connect/disconnect
		0	The upstream port is disconnected (the pull-up is disabled).
		1	The upstream port is connected (the pull-up is enabled).
6	FEN		USB module function enable
		0	The USB module is inactive.
		1	The USB module is active. If the USB module is connected to the bus (the CONN bit is 1), the USB module is ready to communicate with the host.

Table 40. Bits of the USB Control Register (USBCTL) (Continued)

Bit	Field	Value	Description
5	RWUP		Device remote wakeup request. Writing a 1 to this bit generates a remote wakeup condition 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 condition on the bus.
4	FRSTE		USB module function reset enable
		0	If a USB reset request is detected on the bus, the RSTR interrupt request is generated, but the USB module is not reset.
		1	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 (that is, the condition CONN = 1 is maintained).
3–2	Reserved	0	Write 0s to these reserved bits.
1	SETUP		Setup buffer not ready. Software can write a 1 to this bit when a setup packet is being read. A write of 0 has no effect.
		0	The setup buffer is ready for a new setup packet. The CPU has cleared this SETUP bit by writing 1 to the SETUP bit of the USB interrupt flag register (USBIF).
		1	The setup buffer is not ready for a new setup packet because the CPU has not read the previous setup packet yet. The USB module sends a STALL in response to any setup packet until the SETUP bit is 0.
0	DIR		Endpoint 0 data direction bit. The DIR bit plays a vital role during the data phase of a control transfer. 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 does 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.

7.6.5 USB Interrupt Flag Register (USBIF)

USBIF indicates the current status of the bus interrupts.

Notes:

1) The STPOW and SETUP flag bits are not automatically cleared when the CPU reads the interrupt source register (USBINTSRC). To clear these bits, write 1s to them. 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 the SETUP flag bit is cleared, the USB module sets the STPOW flag bit and an STPOW interrupt request (if enabled) is generated.

Figure 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/W−0	R/W1C−0

Legend: R = Read; W1C = Write 1 to clear (writing 0 has no effect); -*n* = Value after reset
† Write 0 to this reserved bit.

Table 41. Bits of the USB Interrupt Flag Register (USBIF)

Bit	Field	Value	Description
7	RSTR		Function reset request interrupt flag. This bit is set in response to host initiating a port reset. This bit is not affected by a USB module reset.
		0	Reset condition not detected on the bus
		1	Reset condition detected on the bus
6	SUSR		Function suspend request interrupt flag
		0	Suspend condition not detected on the bus
		1	Suspend condition detected on the bus
5	RESR		Function resume request interrupt flag
		0	Resume request not detected on the bus
		1	Resume request detected on the bus

Table 41. Bits of the USB Interrupt Flag Register (USBIF) (Continued)

Bit	Field	Value	Description
4	SOF		Start of frame (SOF) interrupt flag
		0	SOF packet not detected on the bus
		1	SOF packet detected on the bus
3	PSOF		Pre-SOF (PSOF) interrupt flag. 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	PSOF notification not received
		1	PSOF notification received from PSOF timer
2	SETUP		Setup packet interrupt flag. 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	New setup packet not received
		1	New setup packet received
1	Reserved	0	Write 0 to this reserved bit.
0	STPOW		Setup overwrite interrupt flag
		0	Setup overwrite has not occurred
		1	Setup overwrite has occurred; that is, SETUP = 1 in USBIF and another setup request has arrived.

7.6.6 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 7.6.5).

Figure 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	R/W–0

Legend: R = Read; W = Write; -*n* = Value after reset
† Write 0 to this reserved bit.

Table 42. Bits of the USB Interrupt Enable Register (USBIE)

Bit	Field	Value	Description
7	RSTR		Reset request interrupt enable. This bit is not affected by a USB reset.
		0	Reset request interrupt disabled
		1	Reset request interrupt enabled
6	SUSR		Suspend request interrupt enable
		0	Suspend request interrupt disabled
		1	Suspend request interrupt enabled
5	RESR		Resume request interrupt enable
		0	Resume request interrupt disabled
		1	Resume request 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 packet interrupt enable
		0	Setup packet interrupt disabled
		1	Setup packet interrupt enabled
1	Reserved	0	Write 0 to this reserved bit.
0	STPOW		Setup overwrite (STPOW) interrupt enable
		0	STPOW interrupt disabled
		1	STPOW interrupt enabled

7.6.7 USB Device Address Register (USBADDR)

This register contains the address that uniquely identifies the USB module on the bus.

Figure 56. USB Device Address Register (USBADDR)

7	6	0
Reserved†	ADDR	
R/W–x	R/W–0	

Legend: R = Read; W = Write; -*n* = Value after reset; –x = Value after reset is not defined.
† Write 0 to this reserved bit.

Table 43. Bits of the USB Device Address Register (USBADDR)

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

7.6.8 USB Idle Control Register (USBIDLECTL)

This register, shown in Figure 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 44.

This register resides in I/O space.

Figure 57. USB Idle Control Register (USBIDLECTL)

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

Legend: R = Read; W = Write; -*n* = Value after reset; –x = Value after reset is not defined
† Write 0s to these reserved bits.

Table 44. Bits of the USB Idle Control Register (USBIDLECTL)

Bit	Field	Value	Description
7–3	Reserved	0	Write 0s to these reserved bits.
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 is not in the idle state.
		1	The USB module is in the idle state following an IDLE instruction.
0	IDLEEN		Idle enable
		0	The USB module is not affected by an IDLE instruction.
		1	Allow the USB module to be idled by an IDLE instruction.
			If the peripherals idle domain of the DSP has been idled by an IDLE instruction (PERIS = 1 in the idle status register), the USB module also goes to the idle state.

Revision History

This document was revised to SPRU596A from SPRU596, which was released in February 2004.

The scope of this revision was limited to adding support for the TMS320VC5507 device.

The following changes were made in this revision:

Page	Additions/Modifications/Deletions
Global	Added the TMS320VC5507 device.

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

W

X

Y