TMS320VC5505/5504 DSP Universal Serial Bus 2.0 (USB) Controller

User's Guide

Texas Instruments

Literature Number: SPRUFO0
September 2009

List of Figures

List of Tables

Read This First

This document describes the universal serial bus 2.0 (USB) in the TMS320VC5505/5504 Digital Signal Processor (DSP). The controller complies with the USB 2.0 standard high-speed and full-speed functions. In addition, the four test modes for high-speed operation described in the USB 2.0 specification are supported. It also allows options that allow the USB controller to be forced into full-speed mode and high-speed mode that may be used for debug purposes.

NOTE: With regard to references of Isochronous mode in this document, note that only Isochronous Asynchronous mode is supported by C5505/04.

Notational Conventions

This document uses the following conventions.

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

Related Documentation From Texas Instruments

The following documents describe the TMS320VC5505/5504 Digital Signal Processor (DSP). Copies of these documents are available on the internet at www.ti.com.

SWPU073 — **TMS320C55x 3.0 CPU Reference Guide.** This manual describes the architecture, registers, and operation of the fixed-point TMS320C55x digital signal processor (DSP) CPU.

SPRU652 — **TMS320C55x DSP CPU Programmer’s Reference Supplement.** This document describes functional exceptions to the CPU behavior.

SPRUFO0 — **TMS320VC5505/5504 Digital Signal Processor (DSP) Universal Serial Bus 2.0 (USB) User's Guide.** This document describes the universal serial bus 2.0 (USB) in the TMS320VC5505/5504 Digital Signal Processor (DSP). The USB controller supports data throughput rates up to 480 Mbps. It provides a mechanism for data transfer between USB devices.

SPRUFO1 — **TMS320VC5505/5504 Digital Signal Processor (DSP) Inter-Integrated Circuit (I2C) Peripheral User's Guide.** This document describes the inter-integrated circuit (I2C) peripheral in the TMS320VC5505/5504 Digital Signal Processor (DSP) device. The I2C peripheral provides an interface between the device and other devices compliant with Phillips Semiconductors Inter-IC bus (I2C-bus) specification version 2.1 and connected by way of an I2C-bus. This document assumes the reader is familiar with the I2C-bus specification.

SPRUFO2 — **TMS320VC5505/5504 Digital Signal Processor (DSP) Timer/Watchdog Timer User's Guide.** This document provides an overview of the three 32-bit timers in the TMS320VC5505/5504 Digital Signal Processor (DSP) device. The 32-bit timers of the device are software programmable timers that can be configured as general-purpose (GP) timers. Timer 2 can be configured as a GP, a Watchdog (WD), or both simultaneously.

SPRUFO3 — TMS320VC5505/5504 Digital Signal Processor (DSP) Serial Peripheral Interface (SPI) User's Guide. This document describes the serial peripheral interface (SPI) in the TMS320VC5505/5504 Digital Signal Processor (DSP) device. The SPI is a high-speed synchronous serial input/output port that allows a serial bit stream of programmed length (1 to 32 bits) to be shifted into and out of the device at a programmed bit-transfer rate. The SPI supports multi-chip operation of up to four SPI slave devices. The SPI can operate as a master device only.

SPRUFO4 — TMS320VC5505/5504 Digital Signal Processor (DSP) General-Purpose Input/Output (GPIO) User's Guide. This document describes the general-purpose input/output (GPIO) on the TMS320VC5505/5504 digital signal processor (DSP). The GPIO peripheral provides dedicated general-purpose pins that can be configured as either inputs or outputs. When configured as an input, you can detect the state of an internal register. When configured as an output you can write to an internal register to control the state driven on the output pin.

SPRUFO5 — TMS320VC5505/5504 Digital Signal Processor (DSP) Universal Asynchronous Receiver/Transmitter (UART) User's Guide. This document describes the universal asynchronous receiver/transmitter (UART) peripheral in the TMS320VC5505/5504 Digital Signal Processor (DSP) device. The UART performs serial-to-parallel conversions on data received from a peripheral device and parallel-to-serial conversion on data received from the CPU.

SPRUF07 — TMS320VC5505/5504 Digital Signal Processor (DSP) Real-Time Clock (RTC) User's Guide. This document describes the operation of the Real-Time Clock (RTC) module in the TMS320VC5505/5504 Digital Signal Processor (DSP) device. The RTC also has the capability to wake-up the power management and apply power to the rest of the device through an alarm, periodic interrupt, or external WAKEUP signal.

SPRUFO8 — TMS320VC5505/5504 Digital Signal Processor (DSP) External Memory Interface (EMIF) User's Guide. This document describes the operation of the external memory interface (EMIF) in the TMS320VC5505/5504 Digital Signal Processor (DSP) device. The purpose of the EMIF is to provide a means to connect to a variety of external devices.

SPRUFO9 — TMS320VC5505/5504 Digital Signal Processor (DSP) Direct Memory Access (DMA) Controller User's Guide. This document describes the features and operation of the DMA controller that is available on the TMS320VC5505/5504 Digital Signal Processor (DSP) device. The DMA controller is used to move data among internal memory, external memory, and peripherals without intervention from the CPU and in the background of CPU operation.

SPRUFP0 — TMS320VC5505 Digital Signal Processor (DSP) System User's Guide. This document describes various aspects of the TMS320VC5505/5504 digital signal processor (DSP) including: system memory, device clocking options and operation of the DSP clock generator, power management features, interrupts, and system control.

SPRUGL6 — TMS320VC5504 Digital Signal Processor (DSP) System User's Guide. This document describes various aspects of the TMS320VC5505/5504 digital signal processor (DSP) including: system memory, device clocking options and operation of the DSP clock generator, power management features, interrupts, and system control.

SPRUFP1 — TMS320VC5505 Digital Signal Processor (DSP) Successive Approximation (SAR) Analog to Digital Converter (ADC) User's Guide. This document provides an overview of the Successive Approximation (SAR) Analog to Digital Converter (ADC) on the TMS320VC5505/5504 Digital Signal Processor (DSP). The SAR is a 10-bit ADC using a switched capacitor architecture which converts an analog input signal to a digital value.

SPRUFP3 — TMS320VC5505 Digital Signal Processor (DSP) Liquid Crystal Display Controller (LCDC) User's Guide. This document describes the liquid crystal display controller (LCDC) in the TMS320VC5505/5504 Digital Signal Processor (DSP) device. The LCD controller includes a LCD Interface Display Driver (LIDD) controller.

SPRUFP4 — TMS320VC5505/5504 Digital Signal Processor (DSP) Inter-IC Sound (I2S) Bus User's Guide. This document describes the features and operation of Inter-IC Sound (I2S) Bus in the TMS320VC5505/5504 Digital Signal Processor (DSP) device. This peripheral allows serial transfer of full duplex streaming data, usually streaming audio, between DSP and an external I2S peripheral device such as an audio codec.

User's Guide
SPRUFO0–September 2009

Universal Serial Bus (USB) Controller

1 Introduction

This document describes the universal serial bus (USB) controller. The controller complies with the USB 2.0 standard high-speed and full-speed functions. In addition, the four test modes for high-speed operation described in the USB 2.0 specification are supported. It also allows options that allow the USB controller to be forced into full-speed mode and high-speed mode that may be used for debug purposes.

1.1 Purpose of the Peripheral

The USB controller provides a low-cost connectivity solution for consumer portable devices by providing a mechanism for data transfer between USB devices up to 480 Mbps. With the USB controller, you can use the DSP to create a high-speed USB slave device that is complaint with the Universal Serial Bus Specification version 2.0.

1.2 Features

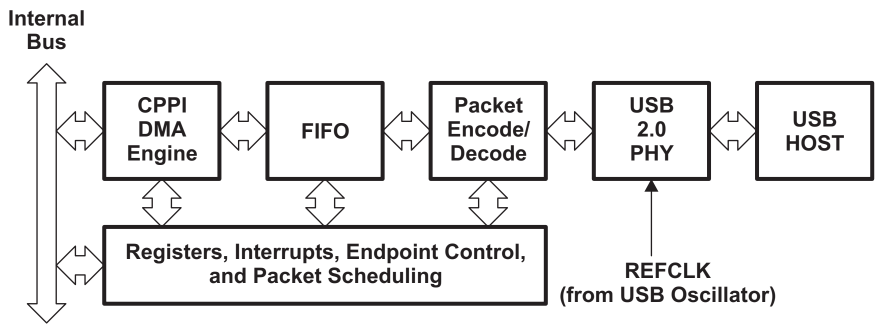
The USB has the following features:

- Operating as a peripheral, it complies with the USB 2.0 standard for high-speed (480 Mbps) and full-speed (12 Mbps) operation with a host
- Supports 4 simultaneous RX and TX endpoints, in addition to control endpoint, more devices can be supported by dynamically switching endpoints states
- Each endpoint (other than endpoint 0) can support all transfer types (control, bulk, interrupt, and isochronous)
- Includes a 4K endpoint FIFO RAM, and supports programmable FIFO sizes
- Includes a DMA controller that supports 4 TX and 4 RX DMA channels
- Includes four types of Communications Port Programming Interface (CPPI) 4.1 DMA compliant transfer modes, Transparent, Generic RNDIS, RNDIS, and Linux CDC mode of DMA for accelerating RNDIS type protocols using short packet termination over USB
- DMA supports single data transfer size up to 4Mbytes

1.3 Functional Block Diagram

The USB functional block diagram is shown in Figure 1.

Figure 1. Functional Block Diagram


Internal Bus
CPPI DMA Engine
FIFO
Packet Encode/ Decode
USB 2.0 PHY
USB HOST
Registers, Interrupts, Endpoint Control, and Packet Scheduling
REFCLK (from USB Oscillator)

1.4 Industry Standard(s) Compliance Statement

This device conforms to USB 2.0 Specification.

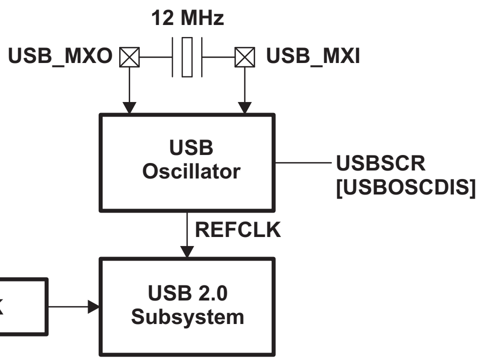
2 Architecture

2.1 Clock Control

Figure 2 shows the clock connections for the USB2.0 module. Note that there is a built-in oscillator that generates a 12 MHz reference clock for the internal PLL of the USB 2.0 subsystem. The USB2.0 subsystem peripheral bus clock is sourced from the system clock (SYSCLK).

NOTE: The device system clock (SYSCLK) must be at least 30 MHz for proper USB operation.

Figure 2. USB Clocking Diagram



2.2 Signal Descriptions

The USB controller provides the I/O signals listed in Table 1.

Table 1. USB Terminal Functions

Name	I/O [(1)]	Description
USB_DP	A I/O/Z	USB D+ (differential signal pair)
USB_DM	A I/O/Z	USB D- (differential signal pair)
USB_VBUS	A I	Five volt input that signifies that VBUS is connected.
USB_REXT	A I/O/Z	External resistor connect.
USB_MXI	I	12 MHz crystal oscillator input.
USB_MXO	O	12 MHz crystal oscillator output.
USB_LDOO	PWR	USB module LDO output. This output is regulated to 1.3V.
USB_LDOI	PWR	USB module LDO input. This input handles a voltage range of 1.8V to 3.6V.
VSS_USBOSC	PWR	3.3V USB oscillator power supply.
VDD_USBPLL	PWR	3.3V USB PLL power supply.
VDDA_USBXCVR	PWR	3.3V USB transceiver power supply.
VDDA_USB	PWR	1.3V USB analog power supply.
VDD_USB	PWR	1.3V USB PLL and oscillator digital power supply.
VSS_USBOSC	GND	USB oscillator ground.
VSS_USBPLL	GND	USB PLL ground.
VSSA_USBXCVR	GND	USB transceiver ground.
VSS_USBXCVR	GND	USB ground for reference circuits.
VSSA_USB	GND	USB analog ground.
VSS_USB	GND	USB PLL and oscillator digital ground.

(1) I = Input, O = Output, Z = High impedance, GND = Ground, A = Analog signal, PWR = Power supply pin.

2.3 Memory Map

The USB controller can access internal single-access RAM (SARAM) and external memory. It cannot access dual-access RAM (DARAM). The starting address for SARAM and external memory is different from the point-of-view of the CPU and USB controller. The memory map, as seen by the USB controller and the CPU, is shown in Table 2.

Table 2. USB Controller Memory Map

USB Start *Byte* Address	CPU Start *Word* Address	CPU Memory Map	USB Controller Memory Map
0001 0000h[(1)]	00 0000h[(1)]	DARAM	Reserved
0009 0000h	00 8000h	SARAM	SARAM
0100 0000h	02 8000h	EMIF CS0	EMIF CS0
0200 0000h	40 0000h	EMIF CS2	EMIF CS2
0300 0000h	60 0000h	EMIF CS3	EMIF CS3
0400 0000h	70 0000h	EMIF CS4	EMIF CS4
0500 0000h	78 0000h	EMIF CS5	EMIF CS5

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

2.4 USB_DP/USB_DM Polarity Inversion

The polarity of the USB data pins (USB_DP and USB_DM) can be inverted through the USBDATPOL bit of the USB system control register (USBSCR). Since USB_DP is equal to the inverse of USB_DM (they form a differential pair), inverting these pins allows you to effectively swap their function. This allows flexibility in board design by allowing different USB connector configurations. In particular, this allows for mounting the connector on either side of the board and for arranging the data pins so they do not physically cross each other.

2.5 Indexed and Non-Indexed Registers

The USB controller provides two mechanisms of accessing the endpoint control and status registers:

- Indexed Endpoint Control/Status Registers: These registers are located at I/O address 8410h to 841Fh. The endpoint is selected by programming the INDEX register of the controller.
- Non-indexed Endpoint Control/Status Registers: These registers are located at I/O address 8500h to 854Fh. Registers at address 8500h to 850Fh map to Endpoint 0; at address 8510h to 851Fh map to Endpoint 1, and so on.

For detailed information about the USB controller registers, see Section 3.

2.6 USB PHY Initialization

The general procedure for USB PHY initialization consists of enabling the USB on-chip oscillator, configuring PHY parameters, and finally resetting the PHY. The detailed USB PHY initialization sequence is as follows:

1. The bits USBOSCBIASDIS and USBOSCDIS in the USB system control register (USBSCR) should be cleared to 0 to enable the on-chip USB oscillatory if not enabled already.
2. Wait cycles for the on-chip oscillator to stabilize. Refer to the device-specific data manual for oscillator stabilization time.
3. To configure the PHY for normal operation, the bits USBPWDN, USBSESSEND, and USBPLLEN in USBSCR should be cleared to 0, the USBVBUSDET bit should be set to 1, and the USBDATPOL bit should be set according to the system requirements (set to 1 for normal operation).
4. Enable the USB clock by clearing USBCG to 0 in the peripheral clock gating configuration register 2 (PCGCR2).
5. Set the USBCLKSTPREQ bit.
6. Set COUNT = 20h in the peripheral software reset counter register (PSRCR).
7. Reset the USB controller by setting USB_RST to 1 in the peripheral reset control register (PRCR). This bit will self-clear once the reset has been completed.

For more information on the PCGCR2, CLKSTOP, PSRCR, and PRCR refer to the *TMS320VC5505 System User's Guide* (SPRUFP0).

2.6.1 USB System Control Register (USBSCR)

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

The USB system control register (USBSCR) is shown in Figure 3 and described in Table 3.

Figure 3. USB System Control Register (USBSCR) [1C32h]

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

7	6	5	4	3	2	1	0
Reserved	USBDATPOL	Reserved		USBOSCBIASDIS	USBOSCDIS	BYTEMODE	

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

Table 3. USB System Control Register (USBSCR) Field Descriptions

Bit	Field	Value	Description
15	USBPWDN		USB module power.
		0	USB module is powered.
		1	USB module is powered-down.
14	USBSESSEND		USB VBUS session end comparator enable.
		0	USB VBUS session end comparator is disabled.
		1	USB VBUS session end comparator is enabled.
13	USBVBUSDET		USB VBUS detect enable.
		0	USB VBUS detect comparator is disabled.
		1	USB VBUS detect comparator is enabled.
12	USBPLLEN		USB PLL enable.
		0	Normal USB operation.
		1	Override USB suspend end behavior and force release of PLL from suspend state.
11-7	Reserved	0	Reserved. Always write 0 to these bits.
6	USBDATPOL		USB data polarity bit.
		0	Reverse polarity on DP and DM signals.
		1	Normal polarity.
5-4	Reserved	0	Reserved.
3	USBOSCBIASDIS		USB internal oscillator bias resistor disable.
		0	Internal oscillator bias resistor enabled (normal operating mode).
		1	Internal oscillator bias resistor disabled.
2	USBOSCDIS		USB oscillator disable bit.
		0	USB internal oscillator enable.
		1	USB internal oscillator disabled.
1-0	BYTEMODE		USB byte mode select bits.
		0	Word accesses by the CPU are allowed.
		1h	Byte accesses by the CPU are allowed (high byte is selected).
		2h	Byte accesses by the CPU are allowed (low byte is selected).
		3h	Reserved.

2.7 Dynamic FIFO Sizing

The USB controller supports a total of 4K RAM to dynamically allocate FIFO to all endpoints. The allocation of FIFO space to the different endpoints requires the specification for each Tx and Rx endpoint of:

- The start address of the FIFO within the RAM block
- The maximum size of packet to be supported
- Whether double-buffering is required.

These details are specified through four registers, which are added to the indexed area of the memory map. That is, the registers for the desired endpoint are accessed after programming the INDEX register with the desired endpoint value. Section 3.48, Section 3.49, Section 3.50, and Section 3.52 provide details of these registers.

NOTE: The option of setting FIFO sizes dynamically only applies to Endpoints 1 to 4. Endpoint 0 FIFO has a fixed size (64 bytes) and a fixed location (start address 0).

It is the responsibility of the firmware to ensure that all the Tx and Rx endpoints that are active in the current USB configuration have a block of RAM assigned exclusively to that endpoint that is at least as large as the maximum packet size set for that endpoint.

2.8 USB Controller Peripheral Mode Operation

The USB controller can be used as a high-speed or a full-speed USB peripheral device attached to a conventional USB host (such as a PC).

The USB2.0 controller will transition to session when it sees power (must be greater or equal to 4.01V) on the USB0_VBUS pin, assuming that the firmware has set the SOFTCONN bit in the POWER register and has enabled the data lines and there is an external host sourcing power on the USB0_VBUS line. The USB 2.0 controller will then set the SESSION bit upon detecting the power on the USB0_VBUS line and it will connect its 1.5Kohm pull-up resistor so it signifies to the external host out it is a Full-Speed device. Note that even when operating as a High-Speed; it has to first come up as Full-Speed. The USB2.0 controller will then wait for a reset signal from the host.

2.8.1 USB Interrupts

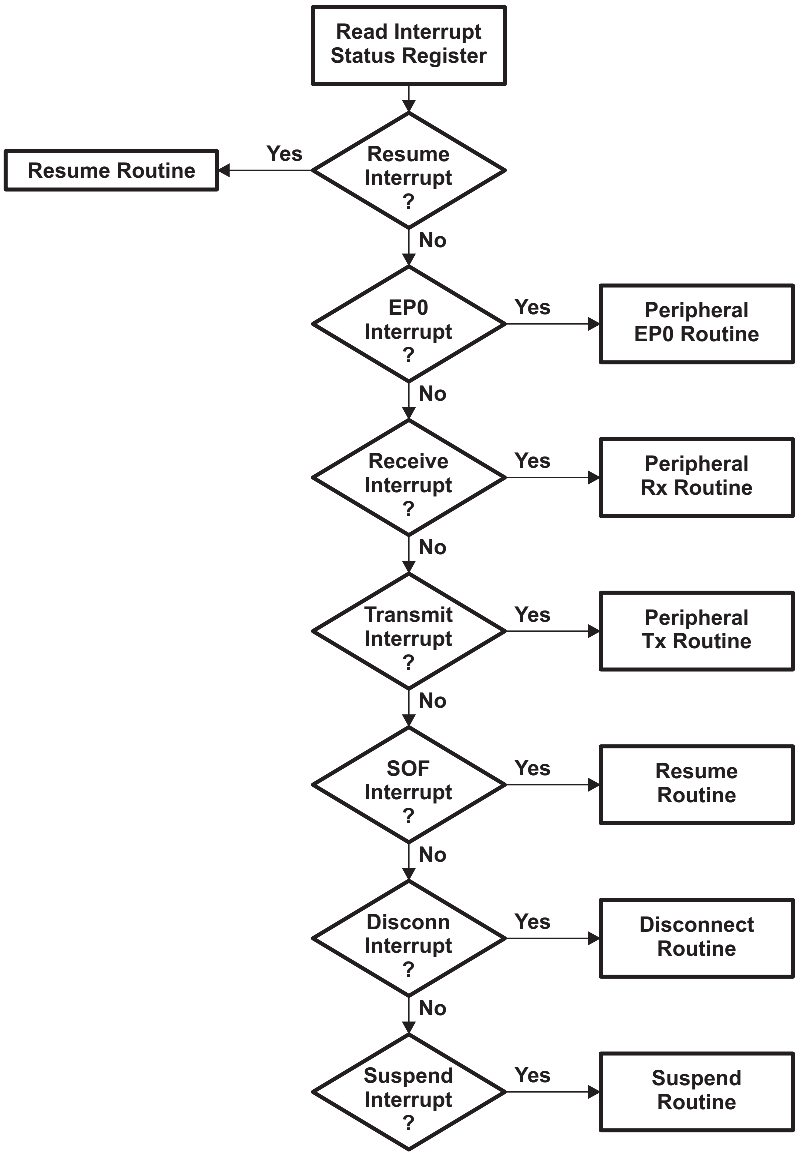
The USB controller interrupts the CPU on completion of the data transfer on any of the endpoints or on detecting reset, resume, suspend, connect, disconnect, or start-of-frame (SOF) on the bus.

When the CPU is interrupted with a USB interrupt, it needs to read the interrupt status register to determine the endpoints that have caused the interrupt and jump to the appropriate routine. If multiple endpoints have caused the interrupt, endpoint 0 should be serviced first followed by the other endpoints. The suspend interrupt should be serviced last.

The flowchart in Figure 4 describes the interrupt service routine for the USB module.

The following sections describe the programming of USB controller in peripheral mode.

Figure 4. Interrupt Service Routine Flow Chart



2.8.2 Connect, Suspend Mode, and Reset Signaling

The following sections describe the operation of the USB controller during connect, suspend mode, and USB reset.

2.8.2.1 Soft Connect

After a reset, the SOFTCONN bit in the POWER register is cleared to 0. The controller will therefore appear disconnected until the software has set the SOFTCONN bit to 1. The application software can then choose when to set the PHY into its normal mode. Systems with a lengthy initialization procedure may use this to ensure that initialization is complete and the system is ready to perform enumeration before connecting to the USB.

Once the SOFTCONN bit of the POWER register has been set, the software can also simulate a disconnect by clearing this bit to 0.

2.8.2.2 Suspend Mode

The controller monitors activity on the bus and when no activity has occurred for 3 ms, it goes into Suspend mode. If the Suspend interrupt has been enabled, an interrupt will be generated.

At this point, the controller can be left active (and hence able to detect when Resume signaling occurs on the USB), or the application may arrange to disable the controller by stopping its clock. However, the controller will not then be able to detect Resume signaling on the USB. As a result, some external hardware will be needed to detect Resume signaling (by monitoring the DM and DP signals) so that the clock to the controller can be restarted.

When Resume signaling occurs on the bus, first the clock to the controller must be restarted if necessary. Then the controller will automatically exit Suspend mode. If the Resume interrupt is enabled, an interrupt will be generated.

If the software wants to initiate a remote wake-up while the controller is in Suspend mode, it should write to the POWER register to set the RESUME bit to 1. The software should then leave this bit set for approximately 10 ms (minimum of 2 ms, a maximum of 15 ms) before resetting it to 0.

NOTE: No resume interrupt will be generated when the software initiates a remote wake-up.

2.8.2.3 Reset Signaling

If the HSENA bit in the POWER register was set, the controller also tries to negotiate for high-speed operation.

Whether high-speed operation is selected is indicated by the HSMODE bit of the POWER register.

When the application software receives a reset interrupt, it should close any open pipes and wait for bus enumeration to begin.

2.8.3 Control Transactions

Endpoint 0 is the main control endpoint of the core. The software is required to handle all the standard device requests that may be sent or received via endpoint 0. These are described in Universal Serial Bus Specification, Revision 2.0, Chapter 9. The protocol for these device requests involves different numbers and types of transactions per transfer. To accommodate this, the software needs to take a state machine approach to command decoding and handling.

The Standard Device Requests received by a USB peripheral device can be divided into three categories: Zero Data Requests (in which all the information is included in the command), Write Requests (in which the command will be followed by additional data), and Read Requests (in which the device is required to send data back to the host).

This section looks at the sequence of actions that the software must perform to process these different types of device request.

NOTE: The Setup packet associated with any standard device request should include an 8-byte command. Any setup packet containing a command field of anything other than 8 bytes will be automatically rejected by the controller.

2.8.3.1 Zero Data Requests

Zero data requests have all their information included in the 8-byte command and require no additional data to be transferred. Examples of Zero Data standard device requests are:

- SET_FEATURE
- CLEAR_FEATURE
- SET_ADDRESS
- SET_CONFIGURATION
- SET_INTERFACE

The sequence of events will begin, as with all requests, when the software receives an endpoint 0 interrupt. The RXPKTRDY bit of the PERI_CSR0 register will also have been set. The 8-byte command should then be read from the endpoint 0 FIFO, decoded, and the appropriate action taken.

For example, if the command is SET_ADDRESS, the 7-bit address value contained in the command should be written to the FADDR register. The PERI_CSR0 register should then be written to set the SERV_RXPKTRDY bit (indicating that the command has been read from the FIFO) and to set the DATAEND bit (indicating that no further data is expected for this request). The interval between setting the SERV_RXPKTRDY bit and setting the DATAEND bit should be very small to avoid getting a SetupEnd error condition.

When the host moves to the status stage of the request, a second endpoint 0 interrupt will be generated to indicate that the request has completed. No further action is required from the software. The second interrupt is just a confirmation that the request completed successfully. For SET_ADDRESS command, the address should be set in the FADDR register only after the status stage interrupt is received.

If the command is an unrecognized command, or for some other reason cannot be executed, then when it has been decoded, the PERI_CSR0 register should be written to set the SERV_RXPKTRDY bit and to set the SENDSTALL bit. When the host moves to the status stage of the request, the controller will send a STALL to tell the host that the request was not executed. A second endpoint 0 interrupt will be generated and the SENTSTALL bit in the PERI_CSR0 register will be set.

If the host sends more data after the DATAEND bit has been set, then the controller will send a STALL. An endpoint 0 interrupt will be generated and the SENTSTALL bit in the PERI_CSR0 register will be set.

NOTE: DMA is not supported for endpoint 0, so the command should be read by accessing the endpoint 0 FIFO register.

2.8.3.2 Write Requests

Write requests involve an additional packet (or packets) of data being sent from the host after the 8-byte command. An example of a Write standard device request is: SET_DESCRIPTOR.

The sequence of events will begin, as with all requests, when the software receives an endpoint 0 interrupt. The RXPKTRDY bit of PERI_CSR0 will also have been set. The 8-byte command should then be read from the Endpoint 0 FIFO and decoded.

As with a zero data request, the PERI_CSR0 register should then be written to set the SERV_RXPKTRDY bit (indicating that the command has been read from the FIFO) but in this case the DATAEND bit should not be set (indicating that more data is expected).

When a second endpoint 0 interrupt is received, the PERI_CSR0 register should be read to check the endpoint status. The RXPKTRDY bit in the PERI_CSR0 register should be set to indicate that a data packet has been received. The COUNT0 register should then be read to determine the size of this data packet. The data packet can then be read from the endpoint 0 FIFO.

If the length of the data associated with the request (indicated by the wLength field in the command) is greater than the maximum packet size for endpoint 0, further data packets will be sent. In this case, PERI_CSR0 should be written to set the SERV_RXPKTRDY bit, but the DATAEND bit should not be set.

When all the expected data packets have been received, the PERI_CSR0 register should be written to set the SERV_RXPKTRDY bit and to set the DATAEND bit (indicating that no more data is expected).

When the host moves to the status stage of the request, another endpoint 0 interrupt will be generated to indicate that the request has completed. No further action is required from the software, the interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when it has been decoded, the PERI_CSR0 register should be written to set the SERV_RXPKTRDY bit and to set the SENDSTALL bit. When the host sends more data, the controller will send a STALL to tell the host that the request was not executed. An endpoint 0 interrupt will be generated and the SENTSTALL bit in the PERI_CSR0 register will be set.

If the host sends more data after the DATAEND bit has been set, then the controller will send a STALL. An endpoint 0 interrupt will be generated and the SENTSTALL bit will be set.

2.8.3.3 Read Requests

Read requests have a packet (or packets) of data sent from the function to the host after the 8-byte command. Examples of Read Standard Device Requests are:

- GET_CONFIGURATION
- GET_INTERFACE
- GET_DESCRIPTOR
- GET_STATUS
- SYNCH_FRAME

The sequence of events will begin, as with all requests, when the software receives an endpoint 0 interrupt. The RXPKTRDY bit in the PERI_CSR0 register will also have been set. Next, the 8-byte command should be read from the endpoint 0 FIFO and decoded. The PERI_CSR0 register should then be written to set the SERV_RXPKTRDY bit (indicating that the command has read from the FIFO).

The data to be sent to the host should be written to the endpoint 0 FIFO. If the data to be sent is greater than the maximum packet size for endpoint 0, only the maximum packet size should be written to the FIFO. The PERI_CSR0 register should then be written to set the TXPKTRDY bit (indicating that there is a packet in the FIFO to be sent). When the packet has been sent to the host, another endpoint 0 interrupt will be generated and the next data packet can be written to the FIFO.

When the last data packet has been written to the FIFO, the PERI_CSR0 register should be written to set the TXPKTRDY bit and to set the DATAEND bit (indicating that there is no more data after this packet).

When the host moves to the status stage of the request, another endpoint 0 interrupt will be generated to indicate that the request has completed. No further action is required from the software: the interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when it has been decoded, the PERI_CSR0 register should be written to set the SERV_RXPKTRDY bit and to set the SENDSTALL bit. When the host requests data, the controller will send a STALL to tell the host that the request was not executed. An endpoint 0 interrupt will be generated and the SENTSTALL bit in the PERI_CSR0 register will be set.

If the host requests more data after the DATAEND bit has been set, then the controller will send a STALL. An endpoint 0 interrupt will be generated and the SENTSTALL bit will be set.

2.8.3.4 Endpoint 0 States

The endpoint 0 control needs three modes – IDLE, TX, and RX – corresponding to the different phases of the control transfer and the states endpoint 0 enters for the different phases of the transfer (described in later sections).

The default mode on power-up or reset should be IDLE. The RXPKTRDY bit in the PERI_CSR0 register becoming set when endpoint 0 is in IDLE state indicates a new device request. Once the device request is unloaded from the FIFO, the controller decodes the descriptor to find whether there is a data phase and, if so, the direction of the data phase of the control transfer (in order to set the FIFO direction). See Figure 5.

Depending on the direction of the data phase, endpoint 0 goes into either TX state or RX state. If there is no Data phase, endpoint 0 remains in IDLE state to accept the next device request.

The actions that the CPU needs to take at the different phases of the possible transfers (for example, loading the FIFO, setting TXPKTRDY) are indicated in Figure 6 .

NOTE: The controller changes the FIFO direction, depending on the direction of the data phase independently of the CPU.

Figure 5. CPU Actions at Transfer Phases

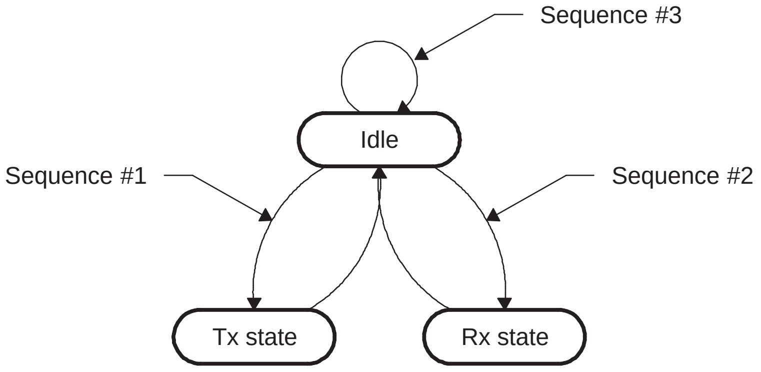
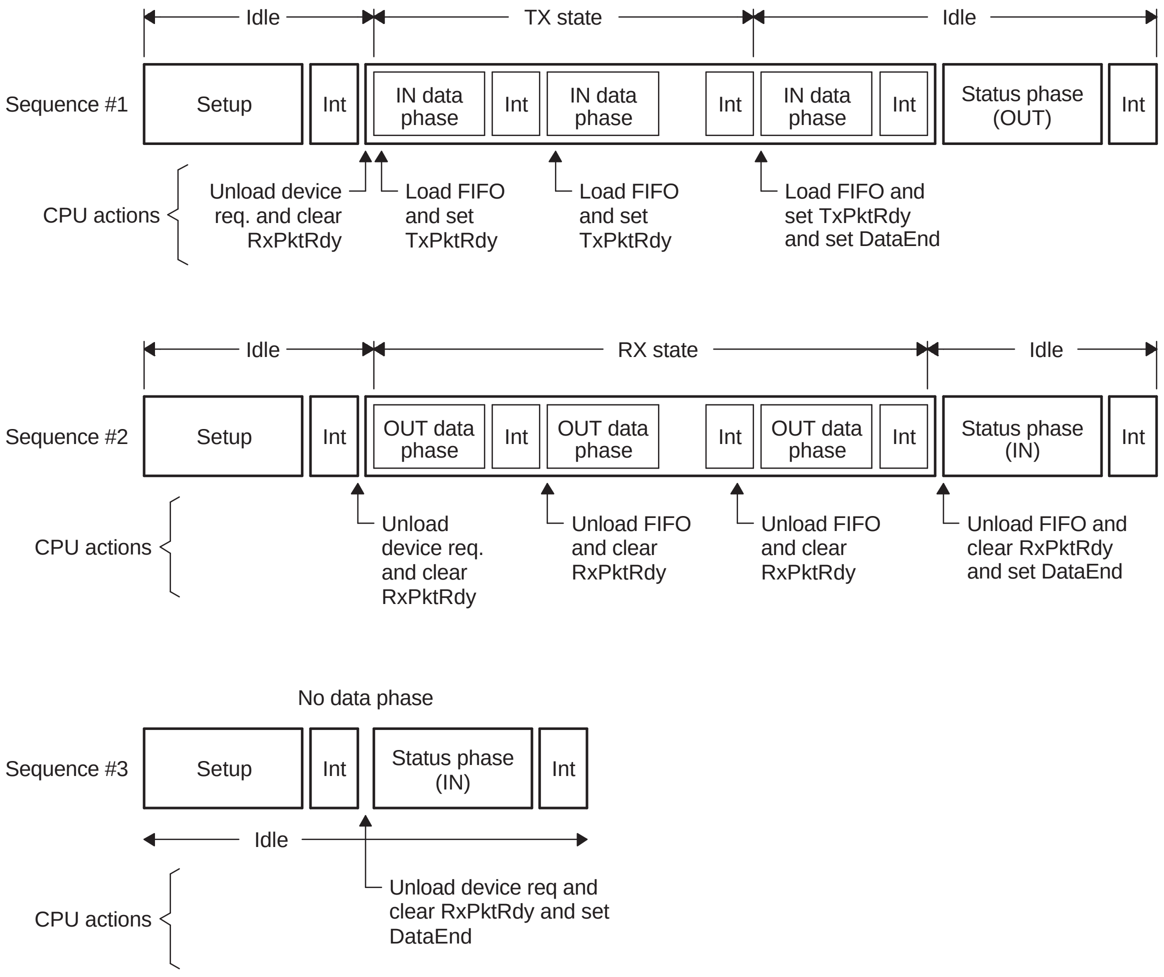


Figure 6. Sequence of Transfer


Idle
TX state
Idle
Sequence #1
Setup
Int
IN data phase
Int
IN data phase
Int
IN data phase
Int
Status phase (OUT)
Int
CPU actions
Unload device req. and clear RxPktRdy
Load FIFO and set TxPktRdy
Load FIFO and set TxPktRdy
Load FIFO and set TxPktRdy and set DataEnd
Idle
RX state
Idle
Sequence #2
Setup
Int
OUT data phase
Int
OUT data phase
Int
OUT data phase
Int
Status phase (IN)
Int
CPU actions
Unload device req. and clear RxPktRdy
Unload FIFO and clear RxPktRdy
Unload FIFO and clear RxPktRdy
Unload FIFO and clear RxPktRdy and set DataEnd
No data phase
Sequence #3
Setup
Int
Status phase (IN)
Int
Idle
CPU actions
Unload device req and clear RxPktRdy and set DataEnd

2.8.3.5 Endpoint 0 Service Routine

An Endpoint 0 interrupt is generated when:

- The controller sets the RXPKTRDY bit in the PERI_CSR0 register after a valid token has been received and data has been written to the FIFO.
- The controller clears the TXPKTRDY bit of PERI_CSR0 after the packet of data in the FIFO has been successfully transmitted to the host.
- The controller sets the SENTSTALL bit of PERI_CSR0 after a control transaction is ended due to a protocol violation.
- The controller sets the SETUPEND bit of PERI_CSR0 because a control transfer has ended before DATAEND is set.

Whenever the endpoint 0 service routine is entered, the software must first check to see if the current control transfer has been ended due to either a STALL condition or a premature end of control transfer. If the control transfer ends due to a STALL condition, the SENTSTALL bit would be set. If the control transfer ends due to a premature end of control transfer, the SETUPEND bit would be set. In either case, the software should abort processing the current control transfer and set the state to IDLE.

Once the software has determined that the interrupt was not generated by an illegal bus state, the next action taken depends on the endpoint state. Figure 7 shows the flow of this process.

If endpoint 0 is in IDLE state, the only valid reason an interrupt can be generated is as a result of the controller receiving data from the bus. The service routine must check for this by testing the RXPKTRDY bit of PERI_CSR0. If this bit is set, then the controller has received a SETUP packet. This must be unloaded from the FIFO and decoded to determine the action the controller must take. Depending on the command contained within the SETUP packet, endpoint 0 will enter one of three states:

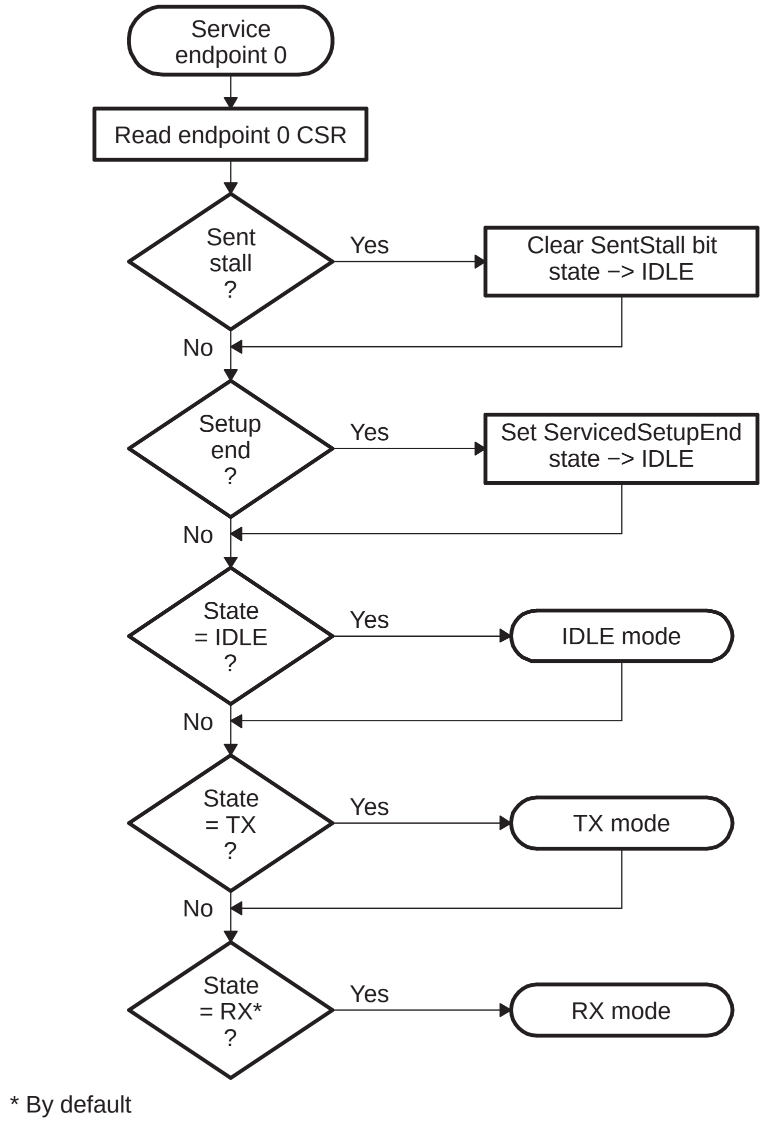
- If the command is a single packet transaction (SET_ADDRESS, SET_INTERFACE etc.) without any data phase, the endpoint will remain in IDLE state.
- If the command has an OUT data phase (SET_DESCRIPTOR etc.), the endpoint will enter RX state.
- If the command has an IN data phase (GET_DESCRIPTOR etc.), the endpoint will enter TX state.

If the endpoint 0 is in TX state, the interrupt indicates that the core has received an IN token and data from the FIFO has been sent. The software must respond to this either by placing more data in the FIFO if the host is still expecting more data or by setting the DATAEND bit to indicate that the data phase is complete. Once the data phase of the transaction has been completed, endpoint 0 should be returned to IDLE state to await the next control transaction.

NOTE: All command transactions include a field that indicates the amount of data the host expects to receive or is going to send.

If the endpoint is in RX state, the interrupt indicates that a data packet has been received. The software must respond by unloading the received data from the FIFO. The software must then determine whether it has received all of the expected data. If it has, the software should set the DATAEND bit and return endpoint 0 to IDLE state. If more data is expected, the firmware should set the SERV_RXPKTRDY bit of PERI_CSR0 to indicate that it has read the data in the FIFO and leave the endpoint in RX state.

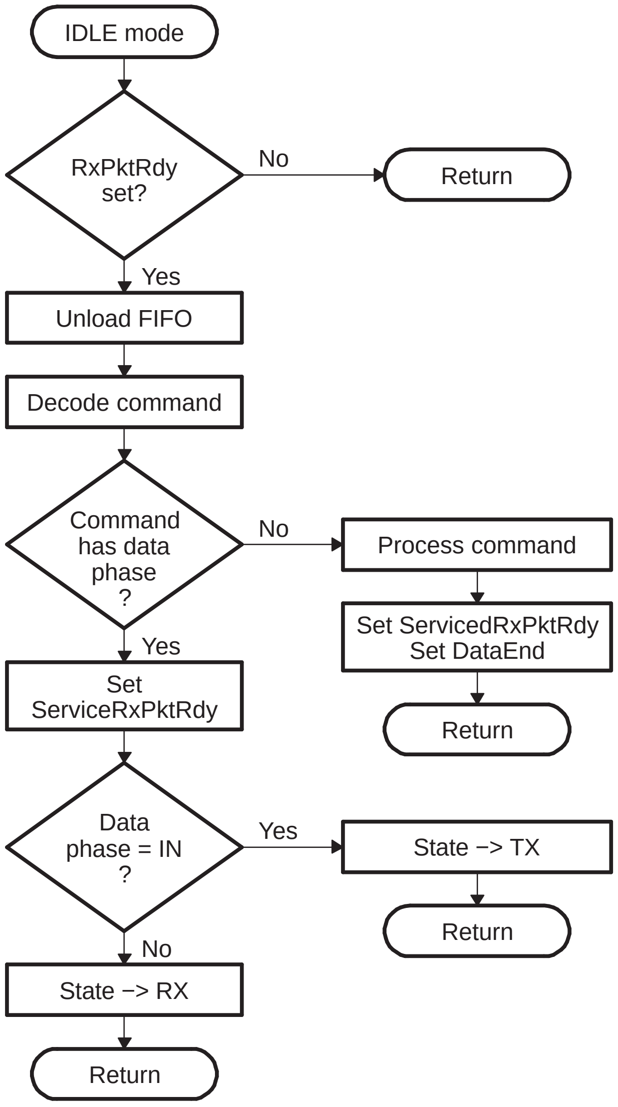
Figure 7. Service Endpoint 0 Flow Chart



2.8.3.5.1 IDLE Mode

IDLE mode is the mode the endpoint 0 control must select at power-on or reset and is the mode to which the endpoint 0 control should return when the RX and TX modes are terminated. It is also the mode in which the SETUP phase of control transfer is handled (as outlined in Figure 8).

Figure 8. IDLE Mode Flow Chart



2.8.3.5.2 TX Mode

When the endpoint is in TX state all arriving IN tokens need to be treated as part of a data phase until the required amount of data has been sent to the host. If either a SETUP or an OUT token is received while the endpoint is in the TX state, this will cause a SetupEnd condition to occur as the core expects only IN tokens. See Figure 9.

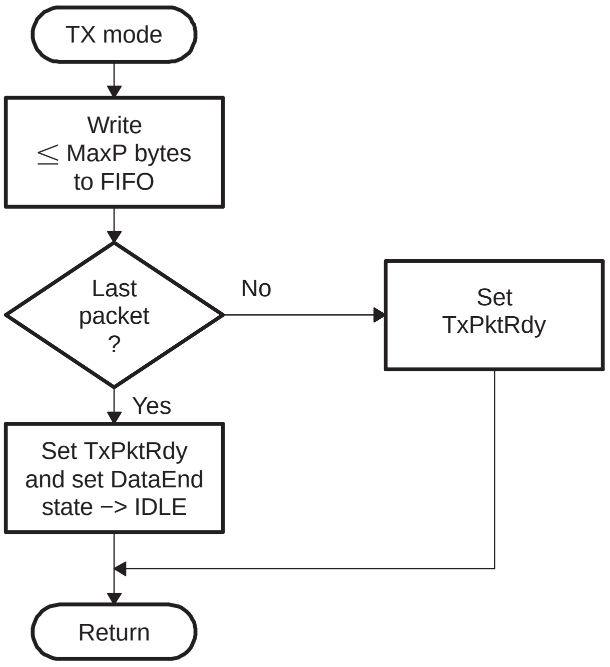
Three events can cause TX mode to be terminated before the expected amount of data has been sent:

1. The host sends an invalid token causing a SETUPEND condition (bit 4 of PERI_CSR0 set).
2. The software sends a packet containing less than the maximum packet size for endpoint 0.
3. The software sends an empty data packet.

Until the transaction is terminated, the software simply needs to load the FIFO when it receives an interrupt that indicates a packet has been sent from the FIFO. (An interrupt is generated when TXPKTRDY is cleared.)

When the software forces the termination of a transfer (by sending a short or empty data packet), it should set the DATAEND bit of PERI_CSR0 (bit 3) to indicate to the core that the data phase is complete and that the core should next receive an acknowledge packet.

Figure 9. TX Mode Flow Chart


TX mode
Write ≤ MaxP bytes to FIFO
Last packet ?
No
Set TxPktRdy
Yes
Set TxPktRdy and set DataEnd state −> IDLE
Return

2.8.3.5.3 RX Mode

In RX mode, all arriving data should be treated as part of a data phase until the expected amount of data has been received. If either a SETUP or an IN token is received while the endpoint is in RX state, a SetupEnd condition will occur as the controller expects only OUT tokens.

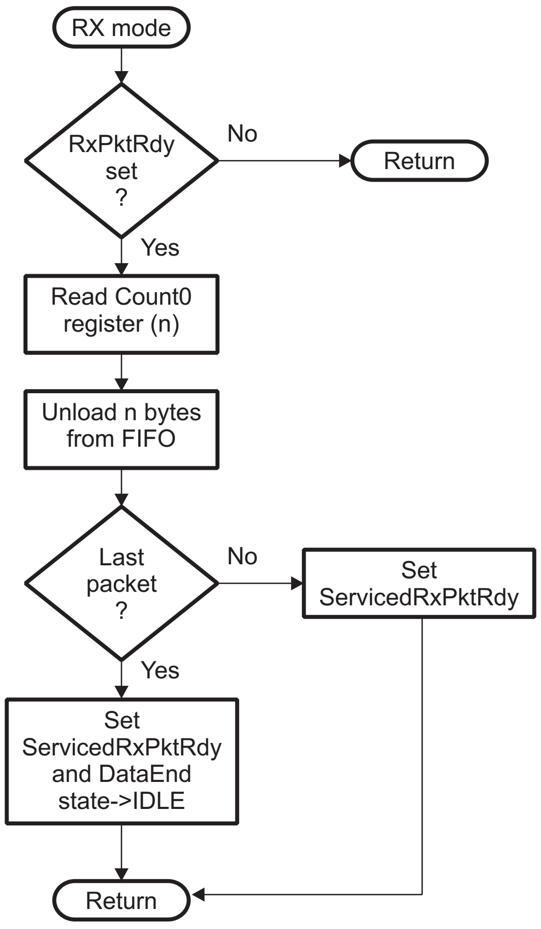
Three events can cause RX mode to be terminated before the expected amount of data has been received as shown in Figure 10:

1. The host sends an invalid token causing a SETUPEND condition (setting bit 4 of PERI_CSR0).
2. The host sends a packet which contains less than the maximum packet size for endpoint 0.
3. The host sends an empty data packet.

Until the transaction is terminated, the software unloads the FIFO when it receives an interrupt that indicates new data has arrived (setting RXPKTRDY bit of PERI_CSR0) and to clear RXPKTRDY by setting the SERV_RXPKTRDY bit of PERI_CSR0 (bit 6).

When the software detects the termination of a transfer (by receiving either the expected amount of data or an empty data packet), it should set the DATAEND bit (bit 3 of PERI_CSR0) to indicate to the controller that the data phase is complete and that the core should receive an acknowledge packet next.

Figure 10. RX Mode Flow Chart



2.8.3.5.4 Error Handling

A control transfer may be aborted due to a protocol error on the USB, the host prematurely ending the transfer, or if the software wishes to abort the transfer (for example, because it cannot process the command).

The controller automatically detects protocol errors and sends a STALL packet to the host under the following conditions:

- The host sends more data during the OUT Data phase of a write request than was specified in the command. This condition is detected when the host sends an OUT token after the DATAEND bit (bit 3 of PERI_CSR0) has been set.
- The host requests more data during the IN Data phase of a read request than was specified in the command. This condition is detected when the host sends an IN token after the DATAEND bit in the PERI_CSR0 register has been set.
- The host sends more than Max Packet Size data bytes in an OUT data packet.
- The host sends a non-zero length DATA1 packet during the STATUS phase of a read request.

When the controller has sent the STALL packet, it sets the SENTSTALL bit (bit 2 of PERI_CSR0) and generates an interrupt. When the software receives an endpoint 0 interrupt with the SENTSTALL bit set, it should abort the current transfer, clear the SENTSTALL bit, and return to the IDLE state.

If the host prematurely ends a transfer by entering the STATUS phase before all the data for the request has been transferred, or by sending a new SETUP packet before completing the current transfer, then the SETUPEND bit (bit 4 of PERI_CSR0) will be set and an endpoint 0 interrupt generated. When the software receives an endpoint 0 interrupt with the SETUPEND bit set, it should abort the current transfer, set the SERV_SETUPEND bit (bit 7 of PERI_CSR0), and return to the IDLE state. If the RXPKTRDY bit (bit 0 of PERI_CSR0) is set this indicates that the host has sent another SETUP packet and the software should then process this command.

If the software wants to abort the current transfer, because it cannot process the command or has some other internal error, then it should set the SENDSTALL bit (bit 5 of PERI_CSR0). The controller will then send a STALL packet to the host, set the SENTSTALL bit (bit 2 of PERI_CSR0) and generate an endpoint 0 interrupt.

2.8.3.5.5 Additional Conditions

The controller automatically responds to certain conditions on the USB bus or actions by the host. The details are:

- Stall Issued to Control Transfers
  - The host sends more data during an OUT Data phase of a Control transfer than was specified in the device request during the SETUP phase. This condition is detected by the controller when the host sends an OUT token (instead of an IN token) after the software has unloaded the last OUT packet and set DataEnd.
  - The host requests more data during an IN data phase of a Control transfer than was specified in the device request during the SETUP phase. This condition is detected by the controller when the host sends an IN token (instead of an OUT token) after the software has cleared TXPKTRDY and set DataEnd in response to the ACK issued by the host to what should have been the last packet.
  - The host sends more than MaxPktSize data with an OUT data token.
  - The host sends the wrong PID for the OUT Status phase of a Control transfer.
  - The host sends more than a zero length data packet for the OUT Status phase.
- Zero Length Out Data Packets In Control Transfer
  - A zero length OUT data packet is used to indicate the end of a Control transfer. In normal operation, such packets should only be received after the entire length of the device request has been transferred (i.e., after the software has set DataEnd). If, however, the host sends a zero length OUT data packet before the entire length of device request has been transferred, this signals the premature end of the transfer. In this case, the controller will automatically flush any IN token loaded by software ready for the Data phase from the FIFO and set SETUPEND bit (bit 4 of PERI_CSR0).

2.8.4 Bulk Transactions

2.8.4.1 Bulk In Transactions

A Bulk IN transaction is used to transfer non-periodic data from the USB peripheral device to the host.

The following optional features are available for use with a Tx endpoint for Bulk IN transactions:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO awaiting transmission to the host. Double packet buffering is enabled by setting the DPB bit of TXFIFOSZ register (bit 4).
- DMA: If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint is able to accept another packet in its FIFO. This feature allows the DMA controller to load packets into the FIFO without processor intervention.

  When DMA is enabled and DMAMODE bit of PERI_TXCSR is set, an endpoint interrupt is not generated for completion of the packet transfer. An endpoint interrupt is generated only in the error conditions.

2.8.4.1.1 Setup

In configuring a TX endpoint for bulk transactions, the TXMAXP register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the endpoint and the PERI_TXCSR register should be set as shown in Table 4 when using DMA:

Table 4. PERI_TXCSR Register Bit Configuration for Bulk IN Transactions

Bit Position	Bit Field Name	Configuration
Bit 14	ISO	Cleared to 0 for bulk mode operation.
Bit 13	MODE	Set to 1 to make sure the FIFO is enabled (only necessary if the FIFO is shared with an RX endpoint).
Bit 12	DMAEN	Set to 1 if DMA requests must be enabled.
Bit 11	FRCDATATOG	Cleared to 0 to allow normal data toggle operations.
Bit 10	DMAMODE	Set to 1 when DMA is enabled and EP interrupt is not needed for each packet transmission.

When the endpoint is first configured (following a SET_CONFIGURATION or SET_INTERFACE command on Endpoint 0), the lower byte of PERI_TXCSR should be written to set the CLRDATATOG bit (bit 6). This will ensure that the data toggle (which is handled automatically by the controller) starts in the correct state.

Also if there are any data packets in the FIFO, indicated by the FIFONOTEMPTY bit (bit 1 of PERI_TXCSR) being set, they should be flushed by setting the FLUSHFIFO bit (bit 3 of PERI_TXCSR).

NOTE: It may be necessary to set this bit twice in succession if double buffering is enabled.

2.8.4.1.2 Operation

When data is to be transferred over a Bulk IN pipe, a data packet needs to be loaded into the FIFO and the PERI_TXCSR register written to set the TXPKTRDY bit (bit 0). When the packet has been sent, the TXPKTRDY bit will be cleared by the USB controller and an interrupt generated so that the next packet can be loaded into the FIFO. If double packet buffering is enabled, then after the first packet has been loaded and the TXPKTRDY bit set, the TXPKTRDY bit will immediately be cleared by the USB controller and an interrupt generated so that a second packet can be loaded into the FIFO. The software should operate in the same way, loading a packet when it receives an interrupt, regardless of whether double packet buffering is enabled or not.

In the general case, the packet size must not exceed the size specified by the lower 11 bits of the TXMAXP register. This part of the register defines the payload (packet size) for transfers over the USB and is required by the USB Specification to be either 8, 16, 32, 64 (Full-Speed or High-Speed) or 512 bytes (High-Speed only).

The host may determine that all the data for a transfer has been sent by knowing the total amount of data that is expected. Alternatively it may infer that all the data has been sent when it receives a packet which is smaller than the stated payload (TXMAXP[10-0]). In the latter case, if the total size of the data block is a multiple of this payload, it will be necessary for the function to send a null packet after all the data has been sent. This is done by setting TXPKTRDY when the next interrupt is received, without loading any data into the FIFO.

If large blocks of data are being transferred, then the overhead of calling an interrupt service routine to load each packet can be avoided by using DMA.

2.8.4.1.3 Error Handling

If the software wants to shut down the Bulk IN pipe, it should set the SENDSTALL bit (bit 4 of PERI_TXCSR). When the controller receives the next IN token, it will send a STALL to the host, set the SENTSTALL bit (bit 5 of PERI_TXCSR) and generate an interrupt.

When the software receives an interrupt with the SENTSTALL bit (bit 5 of PERI_TXCSR) set, it should clear the SENTSTALL bit. It should however leave the SENDSTALL bit set until it is ready to re-enable the Bulk IN pipe.

NOTE: If the host failed to receive the STALL packet for some reason, it will send another IN token, so it is advisable to leave the SENDSTALL bit set until the software is ready to re-enable the Bulk IN pipe. When a pipe is re-enabled, the data toggle sequence should be restarted by setting the CLRDATATOG bit in the PERI_TXCSR register (bit 6).

2.8.4.2 Bulk OUT Transactions

A Bulk OUT transaction is used to transfer non-periodic data from the host to the function controller.

The following optional features are available for use with an Rx endpoint for Bulk OUT transactions:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO on reception from the host. Double packet buffering is enabled by setting the DPB bit of the RXFIFOSZ register (bit 4).
- DMA: If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint has a packet in its FIFO. This feature can be used to allow the DMA controller to unload packets from the FIFO without processor intervention.

  When DMA is enabled, endpoint interrupt will not be generated for completion of packet reception. Endpoint interrupt will be generated only in the error conditions.

2.8.4.2.1 Setup

In configuring an Rx endpoint for Bulk OUT transactions, the RXMAXP register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the endpoint. In addition, the relevant interrupt enable bit in the INTRRXE register should be set (if an interrupt is required for this endpoint) and the PERI_RXCSR register should be set as shown in Table 5.

Table 5. PERI_RXCSR Register Bit Configuration for Bulk OUT Transactions

Bit Position	Bit Field Name	Configuration
Bit 14	ISO	Cleared to 0 to enable Bulk protocol.
Bit 13	DMAEN	Set to 1 if a DMA request is required for this endpoint.
Bit 12	DISNYET	Cleared to 0 to allow normal PING flow control. This will affect only high speed transactions.
Bit 11	DMAMODE	Always clear this bit to 0.

When the endpoint is first configured (following a SET_CONFIGURATION or SET_INTERFACE command on Endpoint 0), the lower byte of PERI_RXCSR should be written to set the CLRDATATOG bit (bit 7). This will ensure that the data toggle (which is handled automatically by the USB controller) starts in the correct state.

Also if there are any data packets in the FIFO (indicated by the RXPKTRDY bit (bit 0 of PERI_RXCSR) being set), they should be flushed by setting the FLUSHFIFO bit (bit 4 of PERI_RXCSR).

NOTE: It may be necessary to set this bit twice in succession if double buffering is enabled.

2.8.4.2.2 Operation

When a data packet is received by a Bulk Rx endpoint, the RXPKTRDY bit (bit 0 of PERI_RXCSR) is set and an interrupt is generated. The software should read the RXCOUNT register for the endpoint to determine the size of the data packet. The data packet should be read from the FIFO, then the RXPKTRDY bit should be cleared.

The packets received should not exceed the size specified in the RXMAXP register (as this should be the value set in the wMaxPacketSize field of the endpoint descriptor sent to the host). When a block of data larger than wMaxPacketSize needs to be sent to the function, it will be sent as multiple packets. All the packets will be wMaxPacketSize in size, except the last packet which will contain the residue. The software may use an application specific method of determining the total size of the block and hence when the last packet has been received. Alternatively it may infer that the entire block has been received when it receives a packet which is less than wMaxPacketSize in size. (If the total size of the data block is a multiple of wMaxPacketSize, a null data packet will be sent after the data to signify that the transfer is complete.)

In the general case, the application software will need to read each packet from the FIFO individually. If large blocks of data are being transferred, the overhead of calling an interrupt service routine to unload each packet can be avoided by using DMA.

2.8.4.2.3 Error Handling

If the software wants to shut down the Bulk OUT pipe, it should set the SENDSTALL bit (bit 5 of PERI_RXCSR). When the controller receives the next packet it will send a STALL to the host, set the SENTSTALL bit (bit 6 of PERI_RXCSR) and generate an interrupt.

When the software receives an interrupt with the SENTSTALL bit (bit 6 of PERI_RXCSR) set, it should clear this bit. It should however leave the SENDSTALL bit set until it is ready to re-enable the Bulk OUT pipe.

NOTE: If the host failed to receive the STALL packet for some reason, it will send another packet, so it is advisable to leave the SENDSTALL bit set until the software is ready to re-enable the Bulk OUT pipe. When a Bulk OUT pipe is re-enabled, the data toggle sequence should be restarted by setting the CLRDATATOG bit (bit 7) in the PERI_RXCSR register.

2.8.5 Interrupt Transactions

An Interrupt IN transaction uses the same protocol as a Bulk IN transaction and can be used the same way. Similarly, an Interrupt OUT transaction uses almost the same protocol as a Bulk OUT transaction and can be used the same way.

Tx endpoints in the USB controller have one feature for Interrupt IN transactions that they do not support in Bulk IN transactions. In Interrupt IN transactions, the endpoints support continuous toggle of the data toggle bit.

This feature is enabled by setting the FRCDATATOG bit in the PERI_TXCSR register (bit 11). When this bit is set, the controller will consider the packet as having been successfully sent and toggle the data bit for the endpoint, regardless of whether an ACK was received from the host.

Another difference is that interrupt endpoints do not support PING flow control. This means that the controller should never respond with a NYET handshake, only ACK/NAK/STALL. To ensure this, the DISNYET bit in the PERI_RXCSR register (bit 12) should be set to disable the transmission of NYET handshakes in high-speed mode.

Though DMA can be used with an interrupt OUT endpoint, it generally offers little benefit as interrupt endpoints are usually expected to transfer all their data in a single packet.

2.8.6 Isochronous Transactions

2.8.6.1 Isochronous IN Transactions

An Isochronous IN transaction is used to transfer periodic data from the function controller to the host.

The following optional features are available for use with a Tx endpoint for Isochronous IN transactions:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO awaiting transmission to the host. Double packet buffering is enabled by setting the DPB bit of TXFIFOSZ register (bit 4).

NOTE: Double packet buffering is generally advisable for Isochronous transactions in order to avoid Underrun errors as described in later section.

- DMA: If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint is able to accept another packet in its FIFO. This feature allows the DMA controller to load packets into the FIFO without processor intervention.

  However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size and the PERI_TXCSR register needs to be accessed following every packet to check for Underrun errors.

  When DMA is enabled and DMAMODE bit of PERI_TXCSR is set, endpoint interrupt will not be generated for completion of packet transfer. Endpoint interrupt will be generated only in the error conditions.

2.8.6.1.1 Setup

In configuring a Tx endpoint for Isochronous IN transactions, the TXMAXP register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the endpoint. In addition, the relevant interrupt enable bit in the INTRTXE register should be set (if an interrupt is required for this endpoint) and the PERI_TXCSR register should be set as shown in Table 6.

Table 6. PERI_TXCSR Register Bit Configuration for Isochronous IN Transactions

Bit Position	Bit Field Name	Configuration
Bit 14	ISO	Set to 1 to enable Isochronous transfer protocol.
Bit 13	MODE	Set to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).
Bit 12	DMAEN	Set to 1 if DMA Requests have to be enabled.
Bit 11	FRCDATATOG	Ignored in Isochronous mode.
Bit 10	DMAMODE	Set it to 1, when DMA is enabled and EP interrupt is not needed for each packet transmission.

2.8.6.1.2 Operation

An Isochronous endpoint does not support data retries, so if data underrun is to be avoided, the data to be sent to the host must be loaded into the FIFO before the IN token is received. The host will send one IN token per frame (or microframe in High-speed mode), however the timing within the frame (or microframe) can vary. If an IN token is received near the end of one frame and then at the start of the next frame, there will be little time to reload the FIFO. For this reason, double buffering of the endpoint is usually necessary.

An interrupt is generated whenever a packet is sent to the host and the software may use this interrupt to load the next packet into the FIFO and set the TXPKTRDY bit in the PERI_TXCSR register (bit 0) in the same way as for a Bulk Tx endpoint. As the interrupt could occur almost any time within a frame(/microframe), depending on when the host has scheduled the transaction, this may result in irregular timing of FIFO load requests. If the data source for the endpoint is coming from some external hardware, it may be more convenient to wait until the end of each frame(/microframe) before loading the FIFO as this will minimize the requirement for additional buffering. This can be done by using either the SOF interrupt or the external SOF_PULSE signal from the controller to trigger the loading of the next data packet. The SOF_PULSE is generated once per frame(/microframe) when a SOF packet is received. (The controller also maintains an external frame(/microframe) counter so it can still generate a SOF_PULSE when the SOF packet has been lost.) The interrupts may still be used to set the TXPKTRDY bit in PERI_TXCSR (bit 0) and to check for data overruns/underruns.

Starting up a double-buffered Isochronous IN pipe can be a source of problems. Double buffering requires that a data packet is not transmitted until the frame(/microframe) after it is loaded. There is no problem if the function loads the first data packet at least a frame(/microframe) before the host sets up the pipe (and therefore starts sending IN tokens). But if the host has already started sending IN tokens by the time the first packet is loaded, the packet may be transmitted in the same frame(/microframe) as it is loaded, depending on whether it is loaded before, or after, the IN token is received. This potential problem can be avoided by setting the ISOUPDATE bit in the POWER register (bit 7). When this bit is set, any data packet loaded into an Isochronous Tx endpoint FIFO will not be transmitted until after the next SOF packet has been received, thereby ensuring that the data packet is not sent too early.

2.8.6.1.3 Error Handling

If the endpoint has no data in its FIFO when an IN token is received, it will send a null data packet to the host and set the UNDERRUN bit in the PERI_TXCSR register (bit 2). This is an indication that the software is not supplying data fast enough for the host. It is up to the application to determine how this error condition is handled.

If the software is loading one packet per frame(/microframe) and it finds that the TXPKTRDY bit in the PERI_TXCSR register (bit 0) is set when it wants to load the next packet, this indicates that a data packet has not been sent (perhaps because an IN token from the host was corrupted). It is up to the application how it handles this condition: it may choose to flush the unsent packet by setting the FLUSHFIFO bit in the PERI_TXCSR register (bit 3), or it may choose to skip the current packet.

2.8.6.2 Isochronous OUT Transactions

An Isochronous OUT transaction is used to transfer periodic data from the host to the function controller.

Following optional features are available for use with an Rx endpoint for Isochronous OUT transactions:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO on reception from the host. Double packet buffering is enabled by setting the DPB bit of RXFIFOSZ register (bit 4).

NOTE: Double packet buffering is generally advisable for Isochronous transactions in order to avoid Overrun errors.

- DMA: If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint has a packet in its FIFO. This feature can be used to allow the DMA controller to unload packets from the FIFO without processor intervention.

  However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size and the PERI_RXCSR register needs to be accessed following every packet to check for Overrun or CRC errors.

  When DMA is enabled, endpoint interrupt will not be generated for completion of packet reception. Endpoint interrupt will be generated only in the error conditions.

2.8.6.2.1 Setup

In configuring an Rx endpoint for Isochronous OUT transactions, the RXMAXP register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the endpoint. In addition, the relevant interrupt enable bit in the INTRRXE register should be set (if an interrupt is required for this endpoint) and the PERI_RXCSR register should be set as shown in Table 7.

Table 7. PERI_RXCSR Register Bit Configuration for Isochronous OUT Transactions

Bit Position	Bit Field Name	Configuration
Bit 14	ISO	Set to 1 to enable isochronous protocol.
Bit 13	DMAEN	Set to 1 if a DMA request is required for this endpoint.
Bit 12	DISNYET	Ignored in isochronous transfers.
Bit 11	DMAMODE	Always clear this bit to 0.

2.8.6.2.2 Operation

An Isochronous endpoint does not support data retries so, if a data overrun is to be avoided, there must be space in the FIFO to accept a packet when it is received. The host will send one packet per frame (or microframe in High-speed mode); however, the time within the frame can vary. If a packet is received near the end of one frame(/microframe) and another arrives at the start of the next frame, there will be little time to unload the FIFO. For this reason, double buffering of the endpoint is usually necessary.

An interrupt is generated whenever a packet is received from the host and the software may use this interrupt to unload the packet from the FIFO and clear the RXPKTRDY bit in the PERI_RXCSR register (bit 0) in the same way as for a Bulk Rx endpoint. As the interrupt could occur almost any time within a frame(/microframe), depending on when the host has scheduled the transaction, the timing of FIFO unload requests will probably be irregular. If the data sink for the endpoint is going to some external hardware, it may be better to minimize the requirement for additional buffering by waiting until the end of each frame(/microframe) before unloading the FIFO. This can be done by using either the SOF interrupt or the external SOF_PULSE signal from the controller to trigger the unloading of the data packet. The SOF_PULSE is generated once per frame(/microframe) when a SOF packet is received. (The controller also maintains an external frame(/microframe) counter so it can still generate a SOF_PULSE when the SOF packet has been lost.) The interrupts may still be used to clear the RXPKTRDY bit in PERI_RXCSR and to check for data overruns/underruns.

2.8.6.2.3 Error Handling

If there is no space in the FIFO to store a packet when it is received from the host, the OVERRUN bit in the PERI_RXCSR register (bit 2) will be set. This is an indication that the software is not unloading data fast enough for the host. It is up to the application to determine how this error condition is handled.

If the controller finds that a received packet has a CRC error, it will still store the packet in the FIFO and set the RXPKTRDY bit (bit 0 of PERI_RXCSR) and the DATAERROR bit (bit 3 of PERI_RXCSR). It is left up to the application how this error condition is handled.

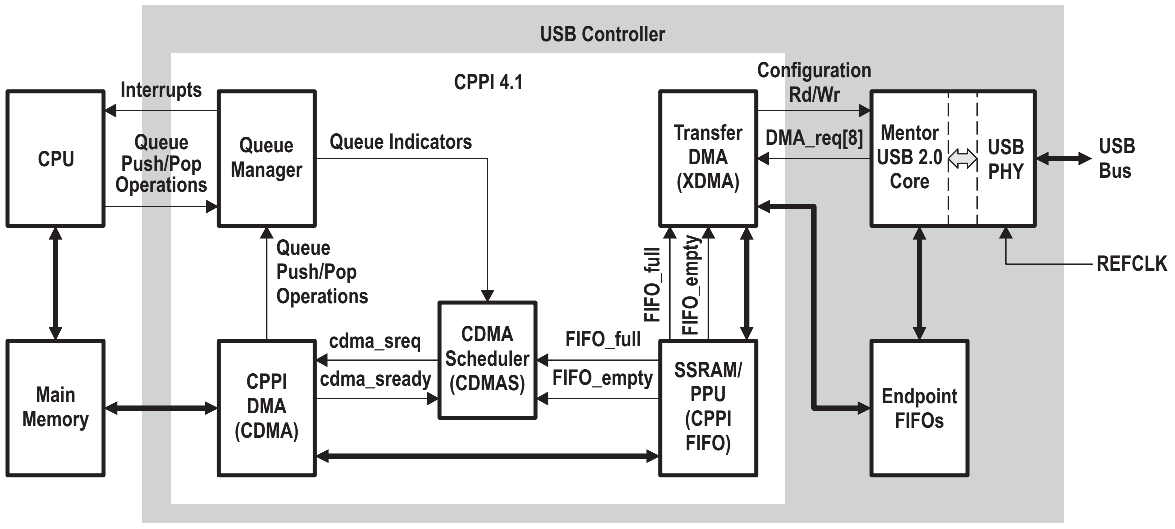
2.9 Communications Port Programming Interface (CPPI) 4.1 DMA Overview

The CPPI DMA module supports the transmission and reception of USB packets. The CPPI DMA is designed to facilitate the segmentation and reassembly of CPPI compliant packets to/from smaller data blocks that are natively compatible with the specific requirements of each networking port. Multiple Tx and Rx channels are provided within the DMA which allow multiple segmentation or reassembly operations to be effectively performed in parallel (but not actually simultaneously). The DMA controller maintains state information for each of the ports/channels which allows packet segmentation and reassembly operations to be time division multiplexed between channels in order to share the underlying DMA hardware. A DMA scheduler is used to control the ordering and rate at which this multiplexing occurs.

The CPPI (version 4.1) DMA controller sub-module is a common 4 dual-port DMA Controller. It supports 4 Tx and 4 Rx Ports and each port attaches to the associated endpoint in the controller. Port 1 maps to endpoint 1 and Port 2 maps to endpoint 2 and Port 3 maps to endpoint 3 and Port 4 maps to endpoint 4, while endpoint 0 can not utilize the DMA and the firmware is responsible to load or offload the endpoint 0 FIFO via CPU.

Figure 11 displays the USB controller block diagram.

Figure 11. USB Controller Block Diagram



Host— The host is an intelligent system resource that configures and manages each communications control module. The host is responsible for allocating memory, initializing all data structures, and responding to port interrupts.

Main Memory— The area of data storage managed by the CPU. The CPPI DMA (CDMA) reads and writes CPPI packets from and to main memory. This memory can exist internal or external from the device.

Queue Manager (QM)— The QM is responsible for accelerating management of a variety of Packet Queues and Free Descriptor / Buffer Queues. It provides status indications to the CDMA Scheduler when queues are empty or full.

CPPI DMA (CDMA)— The CDMA is responsible for transferring data between the CPPI FIFO and Main Memory. It acquires free Buffer Descriptor from the QM (Receive Submit Queue) for storage of received data, posts received packets pointers to the Receive Completion Queue, transmits packets stored on the Transmit Submit Queue (Transmit Queue) , and posts completed transmit packets to the Transmit Completion Queue.

CDMA Scheduler (CDMAS)— The CDMAS is responsible for scheduling CDMA transmit and receive operations. It uses Queue Indicators from the QM and the CDMA to determine the types of operations to schedule.

CPPI FIFO— The CPPI FIFO provides 8 FIFO interfaces (one for each of the 4 transmit and receive endpoints). Each FIFO contains two 64-byte memory storage elements (ping-pong buffer storage).

Transfer DMA (XDMA)— The XDMA receives DMA requests from the Mentor USB 2.0 Core and initiates DMAs to the CPPI FIFO.

Endpoint FIFOs— The Endpoint FIFOs are the USB packet storage elements used by the Mentor USB 2.0 Core for packet transmission or reception. The XDMA transfers data between the CPPI FIFO and the Endpoint FIFOs for transmit operations and between the Endpoint FIFOs and the CPPI FIFO for receive operations.

Mentor USB 2.0 Core— This controller is responsible for processing USB bus transfers (control, bulk, interrupt, and isochronous). It supports 4 transmit and 4 receive endpoints in addition to endpoint 0 (control).

2.9.1 CPPI Terminology

The following terms are important in the discussion of DMA CPPI.

Port— A port is the communications module (peripheral hardware) that contains the control logic for Direct Memory Access for a single transmit/receive interface or set of interfaces. Each port may have multiple communication channels that transfer data using homogenous or heterogeneous protocols. A port is usually subdivided into transmit and receive pairs which are independent of each other. Each endpoint, excluding endpoint 0, has its own dedicated port.

Channel— A channel refers to the sub-division of information (flows) that is transported across ports. Each channel has associated state information. Channels are used to segregate information flows based on the protocol used, scheduling requirements (example: CBR, VBR, ABR), or concurrency requirements (that is, blocking avoidance). All four ports have dedicated single channels, channel 0, associated for their use in a USB application.

Data Buffer— A data buffer is a single data structure that contains payload information for transmission to or reception from a port. A data buffer is a byte aligned contiguous block of memory used to store packet payload data. A data buffer may hold any portion of a packet and may be linked together (via descriptors) with other buffers to form packets. Data buffers may be allocated anywhere within the device memory map. The Buffer Length field of the packet descriptor indicates the number of valid data bytes in the buffer. There may be from 1 to 4M-1 valid data bytes in each buffer.

Host Buffer Descriptor— A buffer descriptor is a single data structure that contains information about one or more data buffers. This type of descriptor is required when more than one descriptor is needed to define an entire packet, i.e., it either defines the middle of a packet or end of a packet.

Host Packet Descriptor— A packet descriptor is another name for the first buffer descriptor within a packet. Some fields within a data buffer descriptor are only valid when it is a packet descriptor including the tags, packet length, packet type, and flags. This type of descriptor is always used to define a packet since it provides packet level information that is useful to both the ports and the Host in order to properly process the packet. It is the only descriptor used when single descriptor solely defines a packet. When multiple descriptors are needed to define a packet, the packet descriptor is the first descriptor used to define a packet.

Free Descriptor/Buffer Queue— A free descriptor/buffer queue is a hardware managed list of available descriptors with pre-linked empty buffers that are to be used by the receive ports for host type descriptors. Free Descriptor/Buffer Queues are implemented by the Queue Manager.

Teardown Descriptor— Teardown Descriptor is a special structure which is not used to describe either a packet or a buffer but is instead used to describe the completion of a channel halt and teardown event. Channel teardown is an important function because it ensures that when a connection is no longer needed that the hardware can be reliably halted and any remaining packets which had not yet been transmitted can be reclaimed by the Host without the possibility of losing buffer or descriptor references (which results in a memory leak).

Packet Queue— A packet queue is hardware managed list of valid (i.e. populated) packet descriptors that is used for forwarding a packet from one entity to another for any number of purposes.

Queue Manager— The queue manager is a hardware module that is responsible for accelerating management of the packet queues. Packets are added to a packet queue by writing the 32-bit descriptor address to a particular memory mapped location in the Queue Manager module. Packets are de-queued by reading the same location for that particular queue. A single Queue Manager is used for a USB application.

NOTE: All descriptors (regardless of type) must be allocated at addresses that are naturally aligned to the smallest power of 2 that is equal to or greater than the descriptor size.

2.9.2 Host Packet Descriptor (SOP Descriptor)

Host Packet Descriptors are designed to be used when USB like application requires support for true, unlimited fragment count scatter/gather type operations. The Host Packet Descriptor is the first descriptor on multiple descriptors setup or the only descriptor in a single descriptors setup. The Host Packet Descriptor contains the following information:

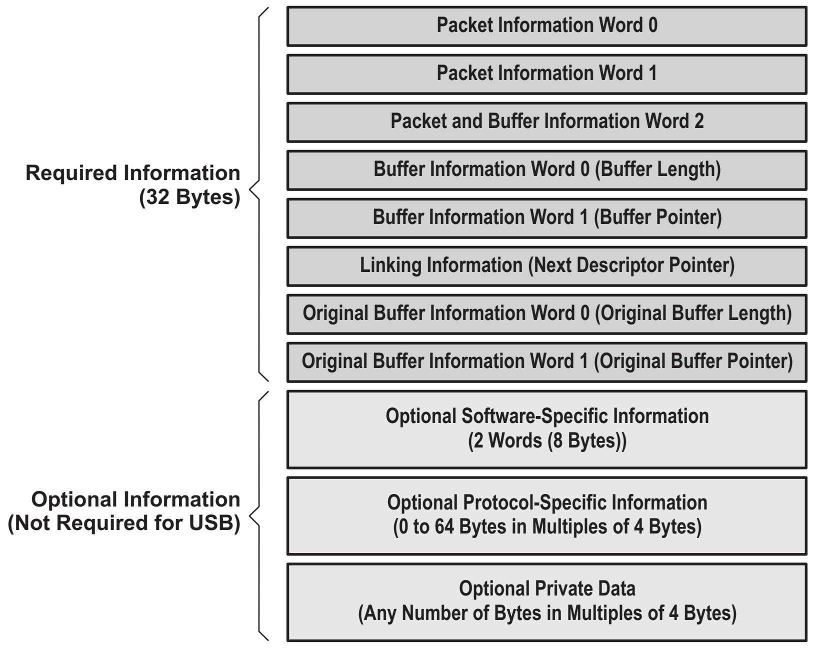
- Indicator which identifies the descriptor as a Host Packet Descriptor (always 10h)
- Source and Destination Tags (Reserved)
- Packet Type
- Packet Length
- Protocol Specific Region Size
- Protocol Specific Control/Status Bits
- Pointer to the first valid byte in the SOP data buffer
- Length of the SOP data buffer
- Pointer to the next buffer descriptor in the packet

Host Packet Descriptors can vary in size of their defined fields from 32 bytes up to 104 bytes. Within this range, Host Packet Descriptors always contain 32 bytes of required information and may also contain 8 bytes of software specific tagging information and up to 64 bytes (indicated in 4 byte increments) of protocol specific information. How much protocol specific information (and therefore the allocated size of the descriptors) is application dependent.

NOTE: Descriptors can be located anywhere within the 16MB address space of the device (except for DARAM, which is not accessible by the USB controller). However, all descriptors must be placed in a single contiguous block of up to 64KW.

The Host Packet Descriptor layout is shown in Figure 12.

Figure 12. Host Packet Descriptor Layout


Required Information
(32 Bytes)
Packet Information Word 0
Packet Information Word 1
Packet and Buffer Information Word 2
Buffer Information Word 0 (Buffer Length)
Buffer Information Word 1 (Buffer Pointer)
Linking Information (Next Descriptor Pointer)
Original Buffer Information Word 0 (Original Buffer Length)
Original Buffer Information Word 1 (Original Buffer Pointer)
Optional Information
(Not Required for USB)
Optional Software-Specific Information
(2 Words (8 Bytes))
Optional Protocol-Specific Information
(0 to 64 Bytes in Multiples of 4 Bytes)
Optional Private Data
(Any Number of Bytes in Multiples of 4 Bytes)

2.9.3 Host Buffer Descriptor (Non-SOP Descriptor)

The Host Buffer Descriptor is identical in size and organization to a Host Packet Descriptor but does not include valid information in the packet level fields and does not include a populated region for protocol specific information. The packet level fields is not needed since the SOP descriptor contain this information and additional copy of this data is not needed/necessary.

Host Buffer Descriptors are designed to be linked onto a Host Packet Descriptor or another Host Buffer Descriptor to provide support for unlimited scatter / gather type operations. Host Buffer Descriptors provide information about a single corresponding data buffer. Every Host buffer descriptor stores the following information:

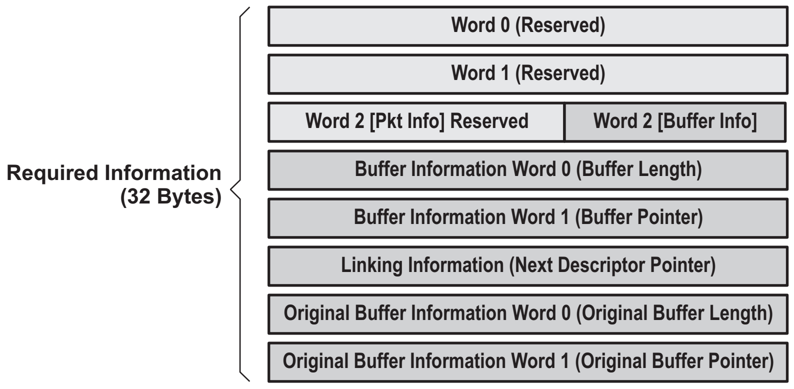
- Pointer to the first valid byte in the data buffer
- Length of the data buffer
- Pointer to the next buffer descriptor in the packet

Host Buffer Descriptors always contain 32 bytes of required information. Since it is a requirement that it is possible to convert a Host descriptor between a Buffer Descriptor and a Packet Descriptor (by filling in the appropriate fields) in practice, Host Buffer Descriptors will be allocated using the same sizes as Host Packet Descriptors. In addition, since the 5 LSBs of the Descriptor Pointers are used in CPPI 4.1 for the purpose of indicating the length of the descriptor, the minimum size of a descriptor is always 32 bytes. (For more information on Descriptor Size, see Section 3.82).

NOTE: Descriptors can be located anywhere within the 16MB address space of the device (except for DARAM, which is not accessible by the USB controller). However, all descriptors must be placed in a single contiguous block of up to 64KW.

The descriptor layout is shown in Figure 13.

Figure 13. Host Buffer Descriptor Layout


Required Information (32 Bytes)
Word 0 (Reserved)
Word 1 (Reserved)
Word 2 [Pkt Info] Reserved
Word 2 [Buffer Info]
Buffer Information Word 0 (Buffer Length)
Buffer Information Word 1 (Buffer Pointer)
Linking Information (Next Descriptor Pointer)
Original Buffer Information Word 0 (Original Buffer Length)
Original Buffer Information Word 1 (Original Buffer Pointer)

2.9.4 Teardown Descriptor

The Teardown Descriptor is not like the Host Packet or Buffer Descriptors since it is not used to describe either a packet or a buffer. The Teardown Descriptor is always 32 bytes long and is comprised of 4 bytes of actual teardown information and 28 bytes of pad (see Figure 14). Since the 5 LSBs of the Descriptor Pointers are used in CPPI 4.1 for the purpose of indicating the length of the descriptor, the minimum size of a descriptor is 32 bytes.

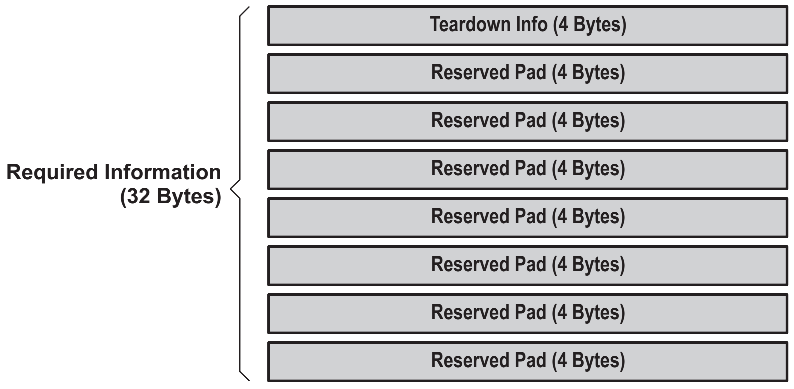
Teardown Descriptor is used to describe a channel halt and teardown event. Channel teardown ensures that when a connection is no longer needed that the hardware can be reliably halted and any remaining packets which had not yet been transmitted can be reclaimed by the Host without the possibility of losing buffer or descriptor references (which results in a memory leak).

NOTE: Descriptors can be located anywhere within the 16MB address space of the device (except for DARAM, which is not accessible by the USB controller). However, all descriptors must be placed in a single contiguous block of up to 64KW.

The Teardown Descriptor contains the following information:

- Indicator which identifies the descriptor as a Teardown Packet Descriptor
- DMA Controller Number where teardown occurred
- Channel number within DMA where teardown occurred
- Indicator of whether this teardown was for the Tx or Rx channel

Figure 14. Teardown Descriptor Layout



Teardown operation of an endpoint requires three operations. The teardown register in the CPPI DMA must be written, the corresponding endpoint bit in TEARDOWN of the USB module must be set, and the FlushFIFO bit in the Mentor USB controller TX/RXCSR register must be set. Writing to TEARDOWN in the USB2.0 module resets the CPPI FIFO occupancy value and pointers to 0. It also resets the current state of the XDMA for the endpoint selected, after any current operations have been completed. Note that due to VBUSP bridge latency, the CPPI FIFO occupancy values will not be reset immediately upon writing of TEARDOWN.

2.9.5 Queues

Several types of queues exist (a total of 64 queues) within the CPPI 4.1 DMA. Regardless of the type of queue a queue is, queues are used to hold pointers to host or buffer packet descriptors while they are being passed between the Host and / or any of the ports in the system. All queues are maintained within the Queue Manager module.

The following type of Queues exist:

- Receive Free Descriptor/Buffer Queue
- Receive Completion (Return) Queue
- Transmit Submit Queue (also referred as Transmit Queue)
- Transmit Completion (Return) Queue
- Free Descriptor Queue (Unassigned: Can be used for Completion or Application Specific purposes)

Table 8 displays the allocation (partition) of the available Queues.

Table 8. Allocation of Queues

Starting Queue Number	Number of Queues	Function
0	16	RX +Free Descriptor/Buffer (submit) queues
16	2	USB Endpoint 1 TX (submit) queues
18	2	USB Endpoint 2 TX (submit) queues
20	2	USB Endpoint 3 TX (submit) queues
22	2	USB Endpoint 4 TX (submit) queues
24	2	TX Completion (return) queues
26	2	RX Completion (return) queues
28	36	Unassigned (application-defined) queues

2.9.5.1 Queuing Packets

Prior to queuing packets, the host/firmware should construct data buffer as well host packet/buffer descriptors within the 16MB address space of the device (except for DARAM which is not accessible by the USB controller).

NOTE: Descriptors must be placed in a single contiguous block of up to 64KW anywhere in the 16MB address space of the device, except DARAM which is not accessible by the USB controller.

Queuing of packets onto a packet queue is accomplished by writing a pointer to the Packet Descriptor into a specific address within the selected queue (Register D of Queue N). Packet is always queued onto the tail of the queue. The Queue Manager provides a unique set of addresses for adding packets for each queue that it manages.

NOTE: The control register D for each queue is split up into two registers (CTRL1D and CTRL2D). To load a descriptor pointer into a queue, use a single double word write to CTRL1D.

2.9.5.2 De-Queuing Packets

De-queuing of packets from a packet queue is accomplished by reading the head packet pointer from a specific address within the selected queue (Register D of Queue N). After the head pointer has been read, the Queue Manager will invalidate the head pointer and will replace it with the next packet pointer in the queue. This functionality, which is implemented in the Queue Manager, prevents the ports from needing to traverse linked lists and allows for certain optimizations to be performed within the Queue Manager.

NOTE: The control register D for each queue is split up into two registers (CTRL1D and CTRL2D). To unload a descriptor pointer into a queue, use a single word read from CTRL1D. The return value will be the lower 16 bits of the descriptor address. Since all descriptors must be within a 64KW memory range, a read from CTRL2D is not necessary.

2.9.5.3 Type of Queues

Several types of queues exist and all are managed by the Queue Manager which is part of the CPPI 4.1 DMA. All accesses to the queues are through memory mapped registers and no external memory setup is required by the firmware.

2.9.5.3.1 Receive Free Descriptor/Buffer (Submit) Queue

Receive ports use queues referred to as "receive free descriptor / buffer queues" to forward completed receive packets to the host or another peer port entity. The entries on the Free Descriptor / Buffer Queues have pre-attached empty buffers whose size and location are described in the "original buffer information" fields in the descriptor. The host is required to allocate both the descriptor and buffer and pre-link them prior to adding (submitting) a descriptor to one of the available receive free descriptor / buffer queue. The first 16 queues (Queue 0 up to Queue 15) are reserved for all four receive ports to handle incoming packets.

2.9.5.3.2 Transmit (Submit) Queue

Transmit ports use packet queues referred to as "transmit (submit) queues" to store the packets that are waiting to be transmitted. Each port has dedicated queues (2 queues per port) that are reserved exclusively for a use by a single port. Multiple queues per port/channel are allocated to facilitate Quality of Service (QoS) for applications that require QoS. Queue 16 and 17 are allocated for port 1, Queue 18 and 19 are allocated for port 2 and Queue 20 and Queue 21 are allocated for port 3 and Queue 22 and 23 are allocated for port 4.

2.9.5.3.3 Transmit Completion Queue

Transmit ports also use packet queues referred to as "transmit completion queues" to return packets to the host after they have been transmitted. Even though, non-allocated queues can be used for this purpose, a total of two dedicated queues (Queue 24 and Queue 25), that is to be shared amongst all four transmit ports, have been reserved for returning transmit packets after end of transmit operation when the firmware desires to receive interrupt when transmission completes.

2.9.5.3.4 Receive Completion Queue

Receive ports also use packet queues referred to as "receive completion queues" to return packets to the port after they have been received. Even though, non-allocated queues can be used for this purpose, a total of two dedicated queues (Queue 26 and Queue 27), that is to be shared amongst all four transmit ports, have been reserved for returning received packets to the receive ports after end of receive operation when the firmware desires to receive interrupt when transmission completes.

2.9.5.3.5 *Unassigned (Application Defined) Queue*

Thirty-six additional queues (Queue 28 to Queue 63) exist that have not been dedicated for exclusive use. The user can use these queues as a Completion Queues or Free Descriptor/Buffer queue.

When these queues are used as Completion Queues, interrupt will not be generated. However, the queues will have the list of descriptor pointers for the packets that have completed transmission or reception. The firmware can use polling method by continually performing the de-queuing technique onto the particular unassigned queue used to identify if the reception or transmission has completed.

When unassigned queues are used as free descriptor/buffer queue, the user can use these queues to queue/store available descriptors for future receive and transmit operations by the firmware popping the respective assigned queue and retrieving and populating descriptor prior to submitting the updated descriptor.

2.9.5.3.6 *Teardown Queue*

The Teardown Queue is used by the DMA to communicate a completion of a channel teardown after a channel teardown is invoked on to a channel. The pointer to the teardown descriptor is written to the teardown queue, which is also the Completion Queue, when the channel teardown completes.

2.9.5.3.7 *Diverting Queue Packets from one Queue to Another*

The host can move the entire contents of one queue to another queue by writing the source queue number and the destination queue number to the Queue Diversion Register. When diverting packets, the host can choose whether the source queue contents should be pushed onto the tail of the destination queue.

2.9.6 Memory Regions and Linking RAM

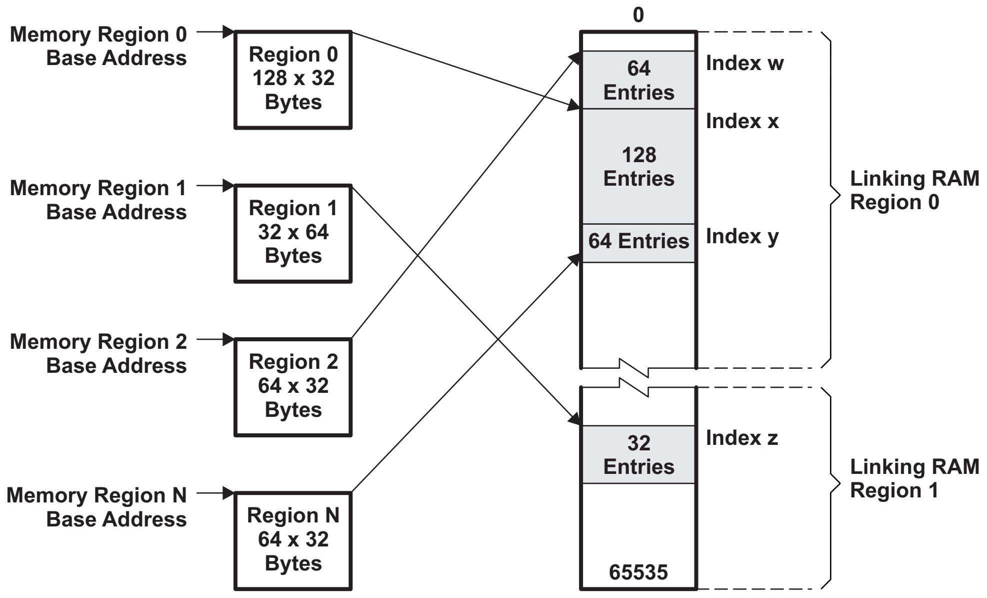
In addition to allocating memory for raw data, the host is responsible for allocating additional memory for exclusive use of the CPPI DMA as well as the Queue Manager. The Queue Manager has the capability of managing up to 16 Memory Regions. These Memory regions are used to allocate descriptors of variable sizes. The total number of descriptors that can be managed by the Queue Manager should not exceed 64K. Each memory region has descriptors of one configurable size. These 64K descriptors are referenced internally in the queue manager by a 16-bit quantity index.

The queue manager uses a linking RAM to store information (16-bit quantity index) about how the descriptors are logically connected to one another. A total of two Linking RAMs exists to be used by all Memory Regions. Each location in the linking RAM stores information for one descriptor index. The linking information for all descriptors in a given memory region is stored in a contiguous fashion in the linking RAM. The host, when it initializes the memory regions, also writes the index number corresponding to the first descriptor in a given region.

This information is used by the queue manager to compute where exactly in memory a particular descriptor is stored. The size of the linking RAM to be allotted by the host/firmware should be large enough to contain information for the amount of descriptor defined within the total Memory Regions. A total of 4 bytes of RAM is required for each descriptor. Figure 15 illustrates the relationship between memory regions and linking RAM.

NOTE: The reason for the existence of the two Linking RAMs is for the case when the user desires to allocate Linking RAMs in both internal and external memory. There is no restriction as to the placement of the Linking RAM.

Figure 15. Relationship Between Memory Regions and Linking RAM



2.9.7 Zero Length Packets

A special case is the handling of null packets with the CPPI 4.1 compliant DMA controller. Upon receiving a zero length USB packet, the XFER DMA will send a data block to the DMA controller with byte count of zero and the zero byte packet bit of INFO Word 2 set. The DMA controller will then perform normal End of Packet termination of the packet, without transferring data.

If a zero-length USB packet is received, the XDMA will send the CDMA a data block with a byte count of 0 and this bit set. The CDMA will then perform normal EOP termination of the packet without transferring data. For transmit, if a packet has this bit set, the XDMA will ignore the CPPI packet size and send a zero-length packet to the USB controller.

2.9.8 CPPI DMA Scheduler

The CPPI DMA scheduler is responsible for controlling the rate and order between the different Tx and Rx threads that are provided in the CPPI DMA controller. The scheduler table RAM exists within the scheduler.

2.9.8.1 CPPI DMA Scheduler Initialization

Before the scheduler can be used, the host is required to initialize and enable the block. This initialization is performed as follows:

1. The Host initializes entries within an internal memory array in the scheduler. This array contains up to 256 entries and each entry consists of a DMA channel number and a bit indicating if this is a Tx or Rx opportunity. These entries represent both the order and frequency that various Tx and Rx channels will be processed. A table size of 256 entries allows channel bandwidth to be allocated with a maximum precision of 1/256th of the total DMA bandwidth. The more entries that are present for a given channel, the bigger the slice of the bandwidth that channel will be given. Larger tables can be accommodated to allow for more precision. This array can only be written by the Host, it cannot be read.
2. If the application does not need to use the entire 256 entries, firmware can initialize the portion of the 256 entries and indicate the last entry used by writing to the LAST_ENTRY bits in the CDMA Scheduler Control Register 1 (DMA_SCHED_CTRL1) in the scheduler.
3. The host writes to the ENABLE bit in DMA_SCHED_CTRL1 to enable the scheduler. The scheduler is not required to be disabled in order to change the scheduler array contents.

2.9.8.2 Scheduler Operation

Once the scheduler is enabled it will begin processing the entries in the table and when appropriate passing credits to the DMA controller to perform a Tx or Rx operation. The operation of the DMA controller is as follows:

1. After the DMA scheduler is enabled it begins with the table index set to 0.
2. The scheduler reads the entry pointed to by the index and checks to see if the channel in question is currently in a state where a DMA operation can be accepted. The following must both be true:
   - The DMA channel must be enabled.
   - The CPPI FIFO that the channel talks to has free space on TX (FIFO full signal is not asserted) or a valid block on Rx (FIFO empty signal is not asserted).
3. If the DMA channel is capable of processing a credit to transfer a block, the DMA scheduler will issue that credit via the DMA scheduling interface. These are the steps:
   (a) The DMA controller may not be ready to accept the credit immediately and it may stall the scheduler until it can accept the credit. The DMA controller only accepts credits when it is in the IDLE state.
   (b) Once a credit has been accepted, the scheduler will increment the index to the next entry and will start at step 2.
4. If the channel in question is not currently capable of processing a credit, the scheduler will increment the index in the scheduler table to the next entry and will start at step 2.
5. When the scheduler attempts to increment its index to the value programmed in the table size register, the index will reset to 0.

2.9.9 CPPI DMA Transfer Interrupt Handling

The CPPI DMA 4.1 Interrupt handling mechanism does not go through the PDR Interrupt handler built into the core. The DMA interrupt line is directly routed to the Interrupt Dispatcher in a PDR compliant manner. The DMA interrupt is not maskable. The firmware needs to use queues not reserved by H/W as Completion Queues if require for DMA interrupt to be generated on a completion of a transfer.

Queues 24 and 25 are reserved by H/W for DMA transmit operations and queues 26 and 27 are reserved by H/W for DMA receive operations. If firmware uses these queues as completion queues, interrupt will be generated when the transfer completes. If need not to generate an interrupt, firmware is required to use queues that are not reserved as completion queues (queues 28 to 67).

2.9.10 DMA State Registers

The port must store and maintain state information for each transmit and receive port/channel. The state information is referred to as the Tx DMA State and Rx DMA State.

2.9.10.1 Transmit DMA State Registers

The Tx DMA State is a combination of control fields and protocol specific port scratchpad space used to manipulate data structures and transmit packets. Each transmit channel has two queues. Each queue has a one head descriptor pointer and one completion pointer. There are four Tx DMA State registers; one for each port/channel.

The following information is stored in the Tx DMA State:

- Tx Queue Head Descriptor Pointer(s)
- Tx Completion Pointer(s)
- Protocol specific control/status (port scratchpad)

2.9.10.2 Receive DMA State Registers

The Rx DMA State is a combination of control fields and protocol specific port scratchpad space used to manipulate data structures in order to receive packets. Each receive channel has only one queue. Each channel queue has one head descriptor pointer and one completion pointer. There are four Rx DMA State registers; one for each port/channel.

The following information is stored in the Rx DMA State:

- Rx Queue Head Descriptor Pointer
- Rx Queue Completion Pointer
- Rx Buffer Offset

2.9.11 USB DMA Protocols Supported

Four different type of DMA transfers are supported by the CPPI 4.1 DMA; Transparent, RNDIS, Generic RNDIS, and Linux CDC. The following sections will outline the details on these DMA transfer types.

2.9.11.1 Transparent DMA

Transparent Mode DMA operation is the default DMA mode where DMA interrupt is generated whenever a DMA packet is transferred. In the transparent mode, DMA packet size cannot be greater than USB MaxPktSize for the endpoint. This transfer type is ideal for transfer (not packet) sizes that are less than a max packet size.

Transparent DMA Transfer Setup

The following will configure all four ports/channels for Transparent DMA Transfer type.

- Make sure that RNDIS Mode is disabled globally. RNDIS bit in the control register (CTRLR) is cleared to 0.
- Configure the endpoint mode control fields in the DMA Mode Registers (MODE1 and MODE2) for Transparent Mode (RX*n*_MODE and TX*n*_MODE = 0h).

2.9.11.2 RNDIS

RNDIS mode DMA is used for large transfers (i.e., total data size to be transferred is greater than USB MaxPktSize where the MzxPktSize is a multiple of 64 bytes) that requires multiple USB packets. This is accomplished by breaking the larger packet into smaller packets, where each packet size being USB MaxPktSize except the last packet where its size is less than USB MaxPktSize, including zero bytes. This implies that multiple USB packets of MaxPktSize will be received and transferred together as a single large DMA transfer and the DMA interrupt is generated only at the end of the complete reception of DMA transfer. The protocol defines the end of the complete transfer by receiving a short USB packet (smaller than USB MaxPktSize as mentioned in USB specification 2.0). If the DMA packet size is an exact multiple of USB MaxPktSize, the DMA controller waits for a zero byte packet at the end of complete transfer to signify the completion of the transfer.

NOTE: RNDIS Mode DMA is supported only when USB MaxPktSize is an integral multiple of 64 bytes.

RNDIS DMA Transfer Setup

The following will configure all four ports/channels for RNDIS DMA Transfer type. If all endpoints are to be configured with the same RNDIS DMA transfer type, then you can enable for RNDIS mode support from the Control Register and the content of the Mode Register will be ignored.

If you need to enable RNDIS support globally.

- Enable RNDIS Mode globally. RNDIS bit in the control register (CTRLR) is set to 1.

If you need to enable RNDIS support at the port/channel (endpoint) level.

- Disable RNDIS Mode globally. RNDIS bit in the control register (CTRLR) is cleared to 0.
- Configure the endpoint mode control fields in the DMA Mode Registers (MODE1 and MODE2) for RNDIS Mode (RX*n*_MODE and TX*n*_MODE = 1h).

The above two setups yield the same result.

2.9.11.3 Generic RNDIS

Generic RNDIS DMA transfer mode is identical to the normal RNDIS mode in nearly all respects, except for the exception case where the last packet of the transfer can either be a short packet or the MaxPktSize. Generic RNDIS transfer makes use of a RNDIS EP Size register (there exists a register for each endpoint) that must be programmed with a value that is an integer multiple of the endpoint size for the DMA to know the end of the transfer when the last packet size is equal to the USB MaxPktSize. For example, it the Tx/RxMaxP is programmed with a value of 64, the Generic RNDIS EP Size register for that endpoint must be programmed with a value that is an integer multiple of 64 (for example, 64, 128, 192, 256, etc.).

In other words, when using Generic RNDIS mode and the DMA is tasked to transfer data transfer size that is less than a value programmed within the RNDIS EP Size register and this transfer will be resulting with a short packet, the DMA will terminate the transfer when encountering the short packet behaving exactly as the RNDIS DMA transfer type.

This means that Generic RNDIS mode will perform data transfer in the same manner as RNDIS mode, closing the CPPI packet when a USB packet is received that is less than the USB MaxPktSize size. Otherwise, the packet will be closed when the value in the Generic RNDIS EP Size register is reached.

Using RNDIS EP Size register, a packet of up to 64K bytes can be transferred. This is to allow the host software to program the USB module to transfer data that is an exact multiple of the USB MaxPktSize (Tx/RxMaxP programmed value) without having to send an additional short packet to terminate.

NOTE: As in RNDIS mode, the USB max packet size of any Generic RNDIS mode enabled endpoints must be a multiple of 64 bytes. Generic RNDIS acceleration should not be enabled for endpoints where the max packet size is not a multiple of 64 bytes. Only transparent mode should be used for such endpoints.

Generic RNDIS DMA Transfer Setup

The following will configure all four ports/channels for Generic RNDIS DMA Transfer type.

- Disable RNDIS Mode globally. RNDIS bit in the control register (CTRLR) is cleared to 0.
- Configure the endpoint mode control fields in the DMA Mode Registers (MODE1 and MODE2) for Generic RNDIS Mode (RX*n*_MODE and TX*n*_MODE = 3h).

2.9.11.4 Linux CDC

Linux CDC DMA transfer mode acts in the same manner as RNDIS packets, except for the case where the last data matches the max USB packet size. If the last data packet of a transfer is a short packet where the data size is greater than zero and less the USB MaxPktSize, then the behavior of the Linux CDC DMA transfer type is identical with the RNDIS DMA transfer type. The only exception is when the short packet length terminating the transfer is a Null Packet. In this case, instead of transferring the Null Packet, it will transfer a data packet of size 1 byte with the data value of 0h.

In transmit operation, if an endpoint is configured or CDC Linux mode, upon receiving a Null Packet from the CPPI DMA, the XFER DMA will then generate a packet containing 1 byte of data, whose value is 0h, indicating the end of the transfer. During receive operation, the XFER DMA will recognize the one byte zero packet as a termination of the data transfer, and sends a block of data with the EOP indicator set and a byte count of one to the CPPI DMA controller. The CPPI DMA realizing the end of the transfer termination will not update/increase the packet size count of the Host Packet Descriptor.

Linux CDC DMA Transfer Setup

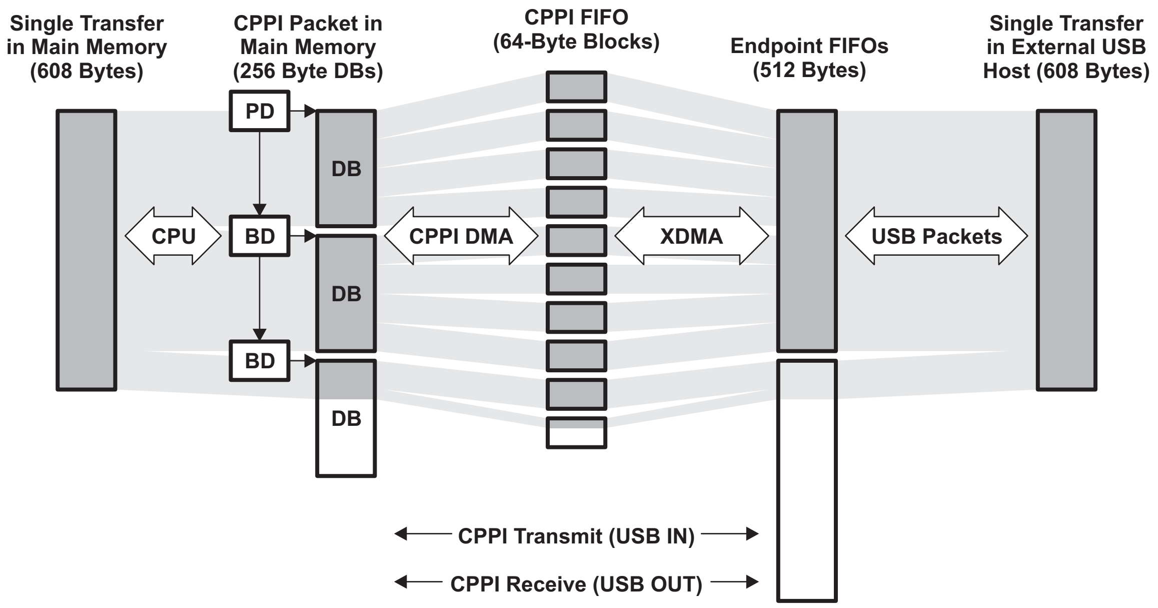
The following will configure all four ports/channels for Linux CDC DMA Transfer type.

- Disable RNDIS Mode globally. RNDIS bit in the control register (CTRLR) is cleared to 0.
- Configure the endpoint mode control fields in the DMA Mode Registers (MODE1 and MODE2) for Linux CDC Mode (RX*n*_MODE and TX*n*_MODE = 2h).

2.9.12 USB Data Flow Using DMA

The necessary steps required to perform a USB data transfer using the CPPI 4.1 DMA is expressed using an example for both transmit and receive cases. Assume a device is ready to perform a data transfer of size 608 bytes (see Figure 16).

Figure 16. High-Level Transmit and Receive Data Transfer Example


Single Transfer in Main Memory (608 Bytes)
CPPI Packet in Main Memory (256 Byte DBs)
CPPI FIFO (64-Byte Blocks)
Endpoint FIFOs (512 Bytes)
Single Transfer in External USB Host (608 Bytes)
PD
BD
BD
DB
DB
DB
CPU
CPPI DMA
XDMA
USB Packets
CPPI Transmit (USB IN)
CPPI Receive (USB OUT)

Example assumptions:

- The CPPI data buffers are 256 bytes in length.
- The USB endpoint 1 Tx and Rx endpoint 1 size are 512 bytes.
- A single transfer length is 608 bytes.
- The SOP offset is 0.

This translates to the following:

- Transmit Case:
  - 1 Host Packet Descriptor with Packet Length field of 608 bytes and a Data Buffer of size 256 Bytes linked to the 1st Host Buffer Descriptor.
  - First Host Buffer Descriptor with a Data Buffer size of 256 Bytes linked to the 2nd Buffer Descriptor.
  - Second Host Buffer Descriptor with a Data Buffer size of 96 bytes (can be greater, the Packet Descriptor contain the size of the packet) linked with its link word set to Null.
- Receive Case:
  - Two Host Buffer Descriptors with 256 bytes of Data Buffer Size
  - One Host Buffer Descriptor with 96 bytes (can be greater) of Data Buffer size

Within the rest of this section, the following nomenclature is used.

BD— Host Buffer Descriptor

DB— Data Buffer Size of 256 Bytes

PBD— Pointer to Host Buffer Descriptor

PD— Host Packet Descriptor

PPD— Pointer to Host Packet Descriptor

RXCQ— Receive Completion Queue or Receive Return Queue (for all Rx EPs, use 26 or 27)

RXSQ— Receive Free Packet/Buffer Descriptor Queue or Receive Submit Queue. (for all Rx EPs, use 0 to 15)

TXCQ— Transmit Completion Queue or Transmit Return Queue (for all Tx EPs, use 24 or 25)

TXSQ— Transmit Queue or Transmit Submit Queue (for EP1, use 16 or 17)

2.9.12.1 Transmit USB Data Flow Using DMA

The transmit descriptors and queue status configuration prior to the transfer taking place is shown in Figure 17. An example of initialization for a transmit USB data flow is shown in Figure 18.

Figure 17. Transmit Descriptors and Queue Status Configuration

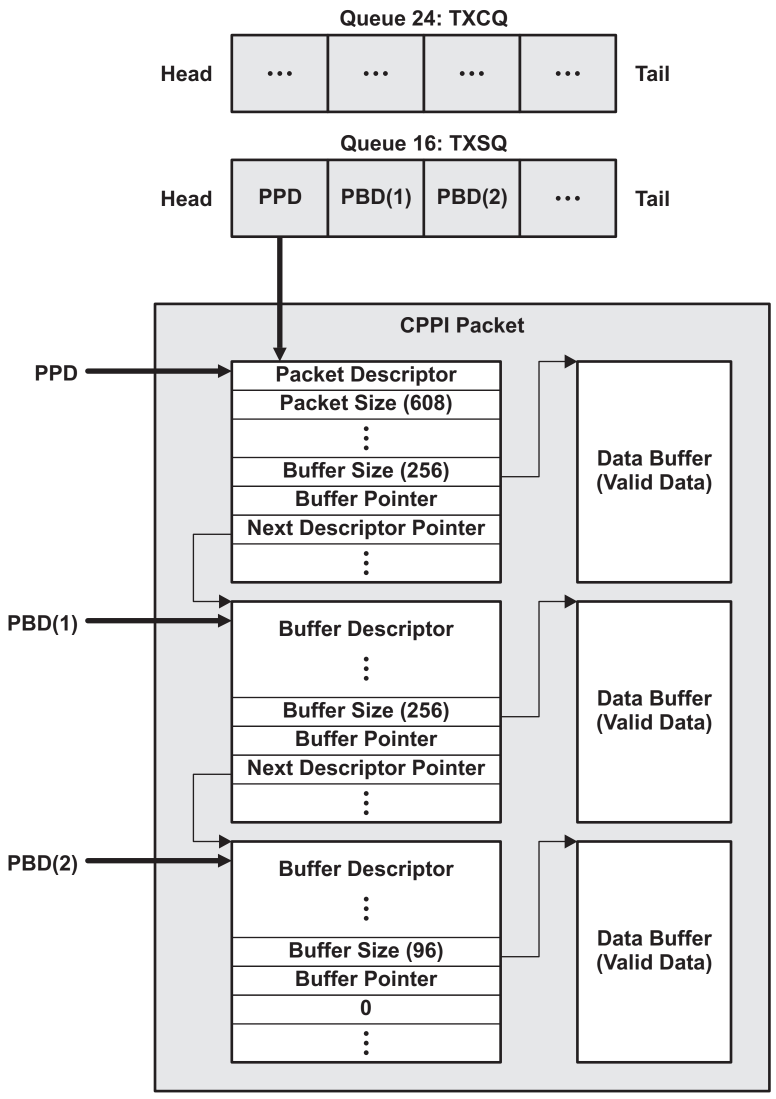
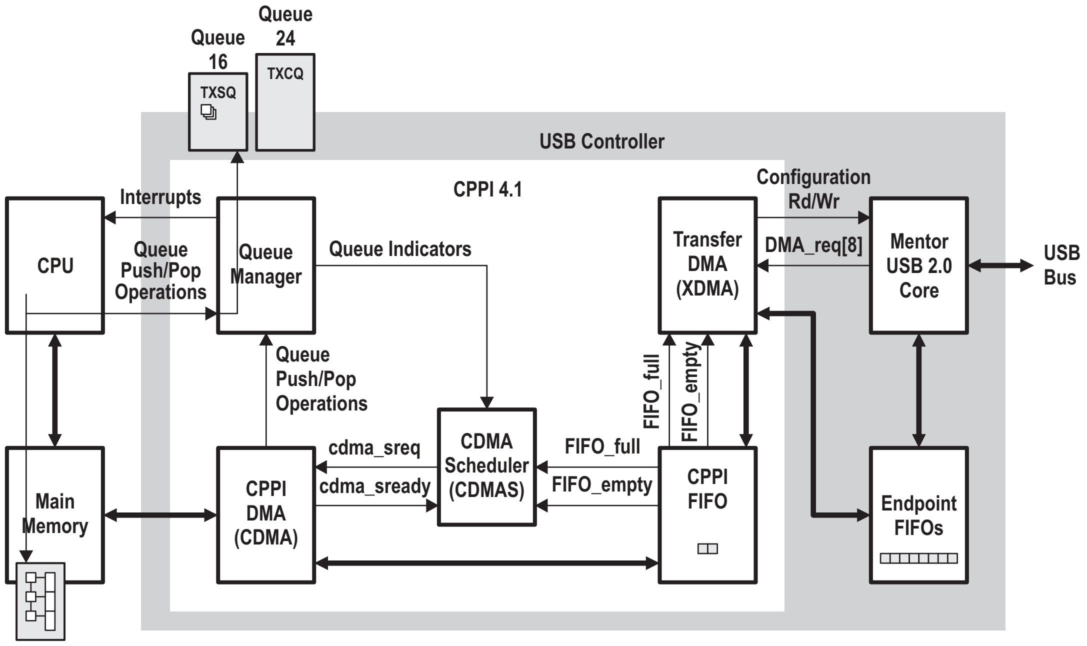


Figure 18. Transmit USB Data Flow Example (Initialization)



Step 1 (Initialization for Tx):

1. The CPU initializes Queue Manager with the Memory Region 0 base address and Memory Region 0 size, Link RAM0 Base address, Link RAM0 data size, and Link RAM1 Base address.
2. The CPU creates PD, BDs, and DBs in main memory and link as indicated in Figure 18.
3. It then initializes and configures the Queue Manager, Channel Setup, DMA Scheduler, and Mentor USB 2.0 Core.
4. It then adds (pushes) the PPD and the two PBDs to the TXSQ

NOTE: You can create more BD/DB pairs and push them on one of the unassigned queues. The firmware can pop a BD/DP pair from this chosen queue and can create its HPD or HBDs and pre link them prior to submitting the pointers to the HPD and HBD on to the TXSQ.

Step 2 (CDMA and XDMA transfers packet data into Endpoint FIFO for Tx):

1. The Queue Manager informs the CDMAS that the TXSQ is not empty.
2. CDMAS checks that the CPPI FIFO FIFO_full is not asserted, then issues a credit to the CDMA.
3. CDMA reads the packet descriptor pointer and descriptor size hint from the Queue Manager.
4. CMDA reads the packet descriptor from memory.
5. For each 64-byte block of data in the packet data payload:
   (a) The CDMA transfers a max burst of 64-byte block from the data to be transferred in main memory to the CPPI FIFO.
   (b) The XDMA sees FIFO_empty not asserted and transfers 64-byte block from CPPI FIFO to Endpoint FIFO.
   (c) The CDMA performs the above 2 steps 3 more times since the data size of the HPD is 256 bytes.
6. The CDMA reads the first buffer descriptor pointer.
7. CDMA reads the buffer descriptor from memory.

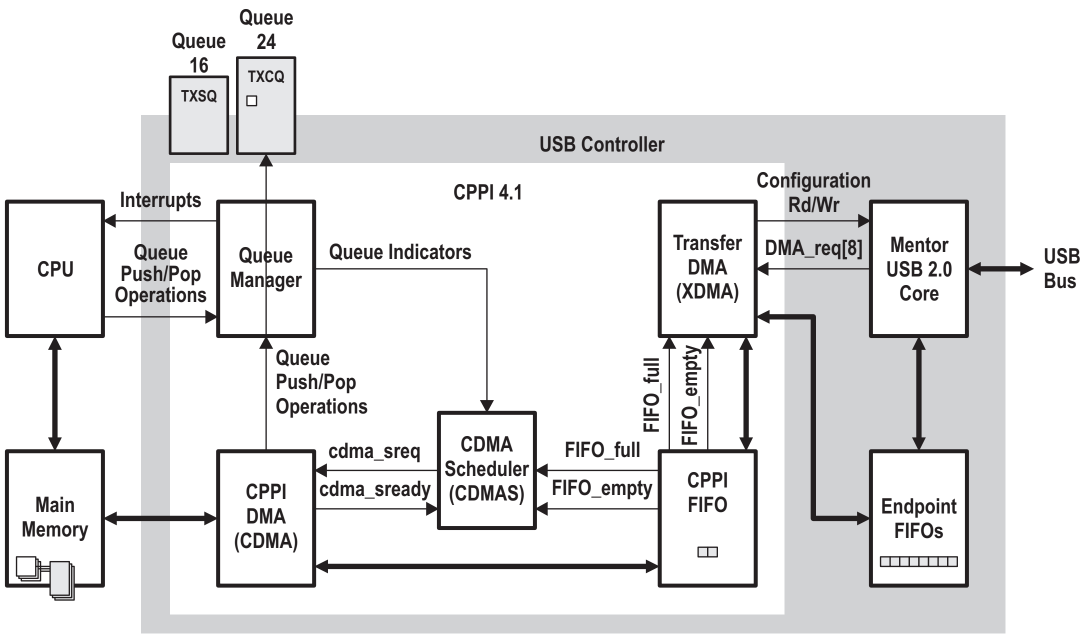
8. For each 64-byte block of data in the packet data payload:
   (a) The CDMA transfers a max burst of 64-byte block from the data to be transferred in main memory to the CPPI FIFO.
   (b) The XDMA sees FIFO_empty not asserted and transfers 64-byte block from CPPI FIFO to Endpoint FIFO.
   (c) The CDMA performs the above 2 steps 2 more times since data size of the HBD is 256 bytes.
9. The CDMA reads the second buffer descriptor pointer.
10. CDMA reads the buffer descriptor from memory.
11. For each 64-byte block of data in the packet data payload:
    (a) The CDMA transfers a max burst of 64-byte block from the data to be transferred in main memory to the CPPI FIFO.
    (b) The XDMA sees FIFO_empty not asserted and transfers 64-byte block from CPPI FIFO to Endpoint FIFO.
    (c) The CDMA transfers the last remaining 32-byte from the data to be transferred in main memory to the CPPI FIFO.
    (d) The XDMA sees FIFO_empty not asserted and transfers 32-byte block from CPPI FIFO to Endpoint FIFO.

Step 3 (Mentor USB 2.0 Core transmits USB packets for Tx):

1. Once the XDMA has transferred enough 64-byte blocks of data from the CPPI FIFO to fill the Endpoint FIFO, it signals the Mentor USB 2.0 Core that a TX packet is ready (sets the endpoint's TxPktRdy bit).
2. The Mentor USB 2.0 Core will transmit the packet from the Endpoint FIFO out on the USB BUS when it receives a corresponding IN request from the attached USB Host.
3. After the USB packet is transferred, the Mentor USB 2.0 Core issues a TX DMA_req to the XDMA.
4. This process is repeated until the entire packet has been transmitted. The XDMA will also generate the required termination packet depending on the termination mode configured for the endpoint.

An example of the completion for a transmit USB data flow is shown in Figure 19.

Figure 19. Transmit USB Data Flow Example (Completion)



Step 4 (Return packet to completion queue and interrupt CPU for Tx):

1. After all data for the packet has been transmitted (as specified by the packet size field), the CDMA will write the pointer to the packet descriptor to the TX Completion Queue specified in the return queue manager / queue number fields of the packet descriptor.

2. The Queue Manager then indicates the status of the TXSQ (empty) to the CDMAS and the TXCQ to the CPU via an interrupt.

2.9.12.2 Receive USB Data Flow Using DMA

The receive descriptors and queue status configuration prior to the transfer taking place is shown in Figure 20. An example of initialization for a receive USB data flow is shown in Figure 21.

Figure 20. Receive Descriptors and Queue Status Configuration

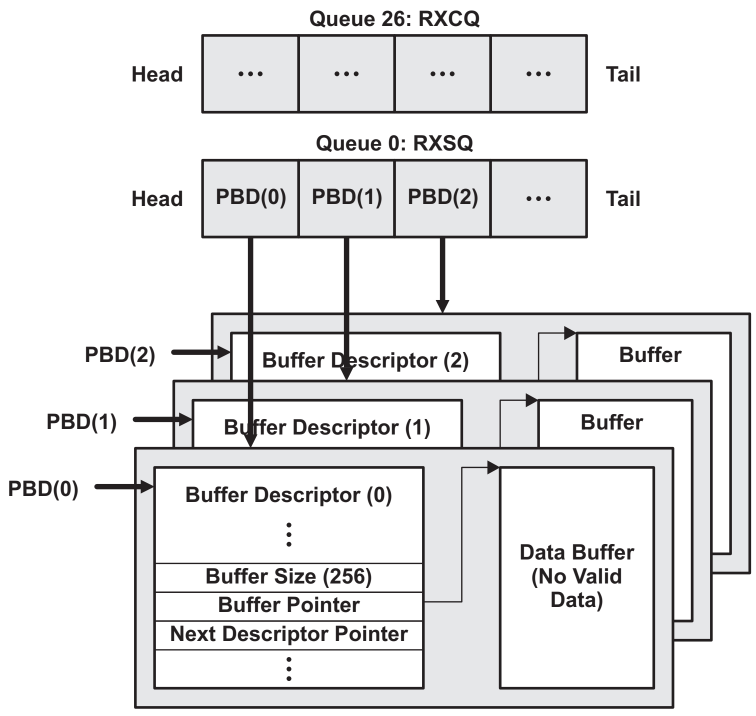
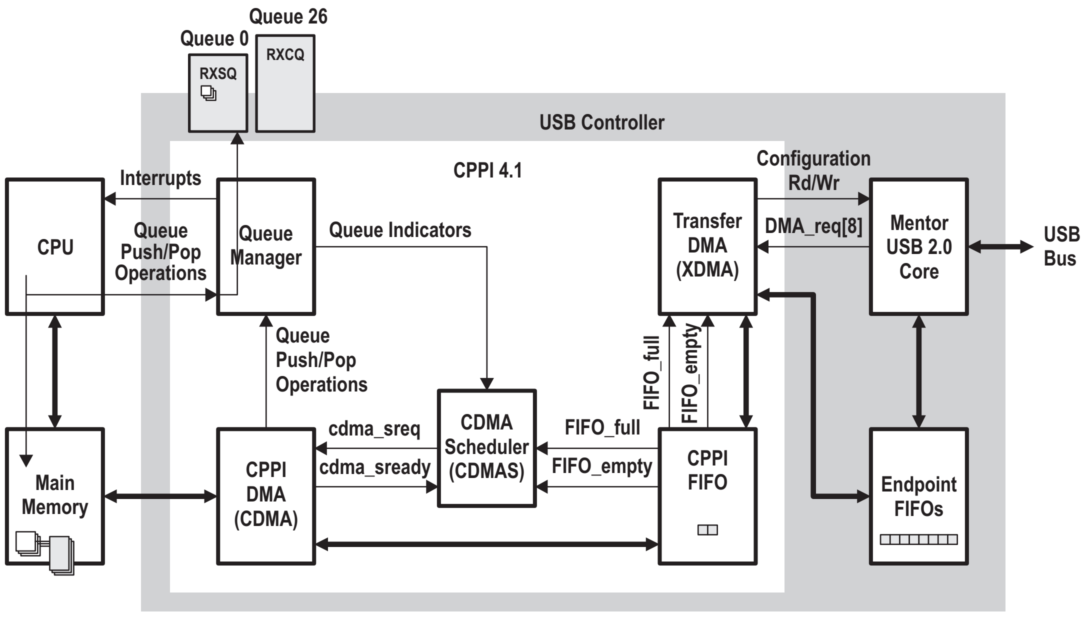


Figure 21. Receive USB Data Flow Example (Initialization)



Step 1 (Initialization for Rx):

1. The CPU initializes Queue Manager with the Memory Region 0 base address and Memory Region 0 size, Link RAM0 Base address, Link RAM0 data size, and Link RAM1 Base address.
2. The CPU creates BDs, and DBs in main memory and link them as indicated in Figure 21.
3. It then initializes the RXCQ queue and configures the Queue Manager, Channel Setup, DMA Scheduler, and Mentor USB 2.0 Core.
4. It then adds (pushes) the address of the three PHDs into the RXSQ.

Step 2 (Mentor USB 2.0 Core receives a packet, XDMA starts data transfer for Receive):

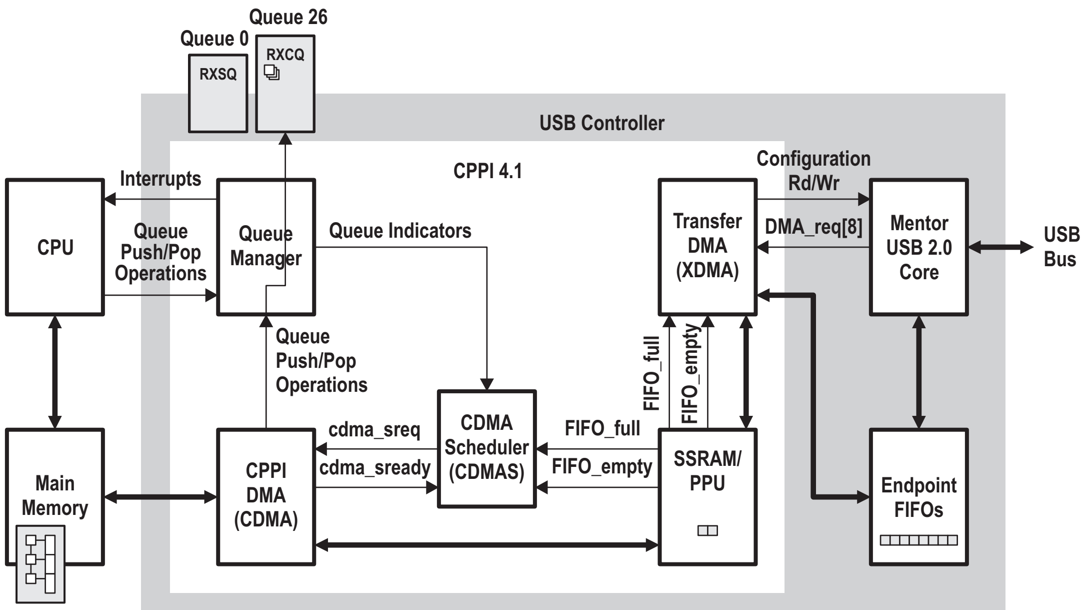
1. The Mentor USB 2.0 Core receives a USB packet from the USB Host and stores it in the Endpoint FIFO.
2. It then asserts a DMA_req to the XDMA informing it that data is available in the Endpoint FIFO.
3. The XDMA verifies the corresponding CPPI FIFO is not full via the FIFO_full signal, then starts transferring 64-byte data blocks from the Endpoint FIFO into the CPPI FIFO.

Step 3 (CDMA transfers data from SSRAM / PPU to main memory for Receive):

1. The CDMAS see FIFO_empty de-asserted (there is RX data in the FIFO) and issues a transaction credit to the CDMA.
2. The CDMA begins packet reception by fetching the first PBD from the Queue Manager using the Free Descriptor / Buffer Queue 0 (Rx Submit Queue) index that was initialized in the RX port DMA state for that channel.
3. The CDMA will then begin writing the 64-byte block of packet data into this DB.
4. The CDMA will continue filling the buffer with additional 64-byte blocks of data from the CPPI FIFO and will fetch additional PBD as needed using the Free Descriptor / Buffer Queue 1, 2, and 3 indexes for the 2nd, 3rd, and remaining buffers in the packet. After each buffer is filled, the CDMA writes the buffer descriptor to main memory.

An example of the completion for a receive USB data flow is shown in Figure 22 .

Figure 22. Receive USB Data Flow Example (Completion)



Step 4 (CDMA completes the packet transfer for Receive):

1. After the entire packet has been received, the CDMA writes the packet descriptor to main memory.
2. The CDMA then writes the packet descriptor to the RXCQ specified in the Queue Manager / Queue Number fields in the RX Global Configuration Register.
3. The Queue Manager then indicates the status of the RXCQ to the CPU via an interrupt.
4. The CPU can then process the received packet by popping the received packet information from the RXCQ and accessing the packet's data from main memory.

2.9.13 Interrupt Handling

Table 9 lists the interrupts generated by the USB controller.

Table 9. Interrupts Generated by the USB Controller

Interrupt	Description
Tx Endpoint [4-0]	Tx endpoint ready or error condition. For endpoints 4 to 0. (Rx and Tx for endpoint 0)
Rx Endpoint [4-1]	Rx endpoint ready or error condition. For endpoints 4 to 1. (Endpoint 0 has interrupt status in Tx interrupt)
USB Core[3-0]	Interrupts for 4 USB conditions
DMA Tx Completion [3-0]	Tx DMA completion interrupt for channel 3 to 0 using Queues 24 and 25
DMA Rx Completion [3-0]	Rx DMA completion interrupt for channel 3 to 0 using Queues 26 and 27

Whenever any of these interrupt conditions are generated, the host processor is interrupted. The software needs to read the different interrupt status registers (discussed in later section) to determine the source of the interrupt.

The USB interrupt conditions are listed in Table 10.

Table 10. USB Interrupt Conditions

Interrupt	Description
USB[3]	SOF started
USB[2]	Reset Signaling detected
USB[1]	Resume signaling detected
USB[0]	Suspend Signaling detected

2.9.13.1 USB Core Interrupts

Interrupt status can be determined using the INTSRCR (interrupt source) registers. These registers are non-masked. To clear the interrupt source, set the corresponding interrupt bit in INTCLRR registers. For debugging purposes, interrupt can be set manually through INTSETR registers.

The interrupt controller provides the option of masking the interrupts. A mask can be set using INTMSKSETR registers and can be cleared by setting the corresponding bit in the INTMSKCLRR registers. The mask can be read from INTMSKR registers. The masked interrupt status is determined using the INTMASKEDR registers.

The host processor software should write to the End Of Interrupt Register (EOIR) to acknowledge the completion of an interrupt.

NOTE: While EOIR is not written, the interrupt from the USB controller remains asserted.

2.10 BYTEMODE Bits of the USB System Control Register

The CPU cannot generate 8-bit accesses to its data or I/O space. This presents a problem given that some USB controller I/O registers are only 8 bits in width.

For these situations, the BYTEMODE bits of the USB system control register (USBSCR) can be used to program the DSP switched central resource (SCR) such that a CPU word access generates single byte access when reading or writing from USB controller I/O registers.

Table 11 summarizes the effect of the BYTEMODE bits for different CPU operations. For more details on USBSCR, please refer to Section 2.6.1.

Table 11. Effect of USBSCR BYTEMODE Bits on USB Access

BYTEMODE Setting	CPU Access to USB Register
BYTEMODE = 0h (16-bit word access)	Entire register contents are accessed.
BYTEMODE = 1h (8-bit access with high byte selected)	Only the upper byte of the register is accessed.
BYTEMODE = 2h (8-bit access with low byte selected)	Only the lower byte of the register is accessed.

2.11 Reset Considerations

The USB controller has two reset sources: hardware reset and the soft reset.

2.11.1 Software Reset Considerations

The USB controller can be reset by software through the RESET bit in the control register (CTRLR) or through the USB_RST bit in the peripheral reset control register (PCR).

When the RESET bit in the control register (CTRLR) is set, all the USB controller registers and DMA operations are reset. The bit is cleared automatically.

When USB_RST is set to 1, a hardware reset is forced on the USB controller. The effects of a hardware reset are described in the next section. Please note that the USB input clock must be enabled when using USB_RST (see Section 2.1).

2.11.2 Hardware Reset Considerations

A hardware reset is always initiated during a full chip reset. Alternatively, software can force an USB controller hardware reset through the USB_RST bits of the peripheral reset control register (PRCR). For more details on PRCR, please refer to the *TMS320VC5505 System User Guide* (SPRUFP0).

When a hardware reset is asserted, all the registers are set to their default values.

2.12 Interrupt Support

The USB controller is capable of interrupting the CPU. For more information on the mapping of interrupts, see the *TMS320VC5505 System User Guide* (SPRUFP0).

2.13 DMA Event Support

The USB is an internal bus master peripheral and does not utilize system DMA events. The USB has its own dedicated DMA, CPPI 4.1 DMA, that it utilizes for DMA driven data transfer.

2.14 Power Management

The USB controller can be clock gated to conserve power during periods of no activity. The clock gating the peripheral is controlled by the CPU. For detailed information on power management procedures, see the *TMS320VC5505 System User Guide* (SPRUFP0).

3 Registers

3.1 USB Controller Register Summary

The following sections summarize the registers for the universal serial bus (USB) controller. Please note that the USB controller includes an USB2.0 mentor core and a communication part programming interface (CPPI) DMA, each with its own set of registers.

3.1.1 Universal Serial Bus (USB) Controller Registers

Table 12 lists the registers of the USB controller. Refer to the sections listed for detailed information on each register.

NOTE: Some USB controller registers are 8-bits wide. However, the CPU cannot generate 8-bit accesses to its data or I/O space. When accessing these registers, program the BYTEMODE bits of the USB system control register (USBSCR) to mask the upper or lower byte of of a word access. The BYTEMODE bits should be set to 00b (16-bit access) when accessing any other register. See Section 2.10 for more details on the BYTEMODE bits.

Table 12. Universal Serial Bus (USB) Registers

CPU Word Address	Acronym	Register Description	Section
8000h	REVID1	Revision Identification Register 1	Section 3.2
8001h	REVID2	Revision Identification Register 2	Section 3.2
8004h	CTRLR	Control Register	Section 3.3
8008h	STATR	Status Register	Section 3.4
800Ch	EMUR	Emulation Register	Section 3.5
8010h	MODE1	Mode Register 1	Section 3.6
8011h	MODE2	Mode Register 2	Section 3.6
8014h	AUTOREQ	Auto Request Register	Section 3.7
8018h	SRPFIXTIME1	SRP Fix Time Register 1	Section 3.8
8019h	SRPFIXTIME2	SRP Fix Time Register 2	Section 3.8
801Ch	TEARDOWN1	Teardown Register 1	Section 3.9
801Dh	TEARDOWN2	Teardown Register 2	Section 3.9
8020h	INTSRCR1	USB Interrupt Source Register 1	Section 3.10
8021h	INTSRCR2	USB Interrupt Source Register 2	Section 3.10
8024h	INTSETR1	USB Interrupt Source Set Register 1	Section 3.11
8025h	INTSETR2	USB Interrupt Source Set Register 2	Section 3.11
8028h	INTCLRR1	USB Interrupt Source Clear Register 1	Section 3.12
8029h	INTCLRR2	USB Interrupt Source Clear Register 2	Section 3.12
802Ch	INTMSKR1	USB Interrupt Mask Register 1	Section 3.13
802Dh	INTMSKR2	USB Interrupt Mask Register 2	Section 3.13
8030h	INTMSKSETR1	USB Interrupt Mask Set Register 1	Section 3.14
8031h	INTMSKSETR2	USB Interrupt Mask Set Register 2	Section 3.14
8034h	INTMSKCLRR1	USB Interrupt Mask Clear Register 1	Section 3.15
8035h	INTMSKCLRR2	USB Interrupt Mask Clear Register 2	Section 3.15
8038h	INTMASKEDR1	USB Interrupt Source Masked Register 1	Section 3.16
8039h	INTMASKEDR2	USB Interrupt Source Masked Register 2	Section 3.16
803Ch	EOIR	USB End of Interrupt Register	Section 3.17
8040h	INTVECTR1	USB Interrupt Vector Register 1	Section 3.18
8041h	INTVECTR2	USB Interrupt Vector Register 2	Section 3.18

Table 12. Universal Serial Bus (USB) Registers (continued)

CPU Word Address	Acronym	Register Description	Section
8050h	GREP1SZR1	Generic RNDIS EP1Size Register 1	Section 3.19
8051h	GREP1SZR2	Generic RNDIS EP1Size Register 2	Section 3.19
8054h	GREP2SZR1	Generic RNDIS EP2 Size Register 1	Section 3.20
8055h	GREP2SZR2	Generic RNDIS EP2 Size Register 2	Section 3.20
8058h	GREP3SZR1	Generic RNDIS EP3 Size Register 1	Section 3.21
8059h	GREP3SZR2	Generic RNDIS EP3 Size Register 2	Section 3.21
805Ch	GREP4SZR1	Generic RNDIS EP4 Size Register 1	Section 3.22
805Dh	GREP4SZR2	Generic RNDIS EP4 Size Register 2	Section 3.22

3.1.2 Mentor USB2.0 Core Registers

This section lists the registers of the Mentor USB2.0 core integrated in the USB controller.

NOTE: Some USB controller registers are 8-bits wide. However, the CPU cannot generate 8-bit accesses to its data or I/O space. When accessing these registers, program the BYTEMODE bits of the USB system control register (USBSCR) to mask the upper or lower byte of of a word access. The BYTEMODE bits should be set to 00b (16-bit access) when accessing any other register. See Section 2.10 for more details on the BYTEMODE bits.

3.1.2.1 Common USB Registers

Table 14 lists the common USB registers. Some common USB registers are 8-bit wide and share a word address with other 8-bit registers. Table 13 describes how the common USB registers are laid out in memory.

Table 13. Common USB Register Layout

CPU Word Address	Register	
	Byte 1	Byte 0
8400h	POWER	FADDR
8401h	INTRTX	
8404h	INTRRX	
8405h	INTRTXE	
8408h	INTRRXE	
8409h	INTRUSBE	INTRUSB
840Ch	FRAME	
840Dh	TESTMODE	INDEX

Table 14. Common USB Registers

CPU Word Address	Acronym	Register Description	Section
8400h	FADDR_POWER	Function Address Register, Power Management Register	Section 3.23
8401h	INTRTX	Interrupt Register for Endpoint 0 plus Transmit Endpoints 1 to 4	Section 3.25
8404h	INTRRX	Interrupt Register for Receive Endpoints 1 to 4	Section 3.26
8405h	INTRTXE	Interrupt enable register for INTRTX	Section 3.27
8408h	INTRRXE	Interrupt Enable Register for INTRRX	Section 3.28
8409h	INTRUSB_INTRUSBE	Interrupt Register for Common USB Interrupts, Interrupt Enable Register	Section 3.29

Table 14. Common USB Registers (continued)

CPU Word Address	Acronym	Register Description	Section
840Ch	FRAME	Frame Number Register	Section 3.31
840Dh	INDEX_TESTMODE	Index Register for Selecting the Endpoint Status and Control Registers, Register to Enable the USB 2.0 Test Modes	Section 3.32

3.1.2.2 *Indexed Registers*

Table 17 lists the index registers. These registers operate on the endpoint selected by the index register. (The index register is the low-8 bits of the INDEX_TESTMODE 16 bits register). Table 15 describes how the indexed USB registers are laid out in memory when endpoint 0 is selected in the index register (INDEX = 0). Similarly, Table 16 shows the layout of the indexed registers when endpoints 1-4 are selected in the index register (INDEX = 1 or 2 or 3 or 4).

Table 15. USB Indexed Register Layout when Index Register Set to Select Endpoint 0

	Register	
CPU Word Address	Byte 1	Byte 0
8410h	Reserved	
8411h	PERI_CSR0	
8414h	Reserved	
8415h	Reserved	
8418h	COUNT0	
8419h	Reserved	
841Ch	Reserved	
841Dh	CONFIGDATA_INDX	Reserved

Table 16. USB Indexed Register Layout when Index Register Set to Select Endpoint 1-4

	Register	
CPU Word Address	Byte 1	Byte 0
8410h	TXMAXP	
8411h	PERI_TXCSR	
8414h	RXMAXP	
8415h	PERI_RXCSR	
8418h	RXCOUNY	
8419h	Reserved	
841Ch	Reserved	
841Dh	Reserved	

Table 17. USB Indexed Registers

CPU Word Address	Acronym	Register Description	Section
8410h	TXMAXP_MAP	Maximum Packet Size for Peripheral/Host Transmit Endpoint. (Index register set to select Endpoints 1-4)	Section 3.34
8411h	PERI_CSR0	Control Status Register for Peripheral Endpoint 0. (Index register set to select Endpoint 0)	Section 3.35
	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint. (Index register set to select Endpoints 1-4)	Section 3.36
8414h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint. (Index register set to select Endpoints 1-4)	Section 3.37

Table 17. USB Indexed Registers (continued)

CPU Word Address	Acronym	Register Description	Section
8415h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint. (Index register set to select Endpoints 1-4)	Section 3.38
8418h	COUNT0	Number of Received Bytes in Endpoint 0 FIFO. (Index register set to select Endpoint 0)	Section 3.39
	RXCOUNT	Number of Bytes in Host Receive Endpoint FIFO. (Index register set to select Endpoints 1- 4)	Section 3.40
8419h	-	Reserved	
841Ch	-	Reserved	
841Dh	CONFIGDATA_INDC (Upper byte of 841Dh)	Returns details of core configuration. (index register set to select Endpoint 0)	Section 3.41

3.1.2.3 FIFO Registers

Table 18 lists the FIFO registers of the USB2.0 Mentor core.

Table 18. USB FIFO Registers

CPU Word Address	Acronym	Register Description	Section
8420h	FIFO0R1	Transmit and Receive FIFO Register 1 for Endpoint 0	Section 3.42
8421h	FIFO0R2	Transmit and Receive FIFO Register 2 for Endpoint 0	Section 3.42
8424h	FIFO1R1	Transmit and Receive FIFO Register 1 for Endpoint 1	Section 3.43
8425h	FIFO1R2	Transmit and Receive FIFO Register 2 for Endpoint 1	Section 3.43
8428h	FIFO2R1	Transmit and Receive FIFO Register 1 for Endpoint 2	Section 3.44
8429h	FIFO2R2	Transmit and Receive FIFO Register 2 for Endpoint 2	Section 3.44
842Ch	FIFO3R1	Transmit and Receive FIFO Register 1 for Endpoint 3	Section 3.45
842Dh	FIFO3R2	Transmit and Receive FIFO Register 2 for Endpoint 3	Section 3.45
8430h	FIFO4R1	Transmit and Receive FIFO Register 1 for Endpoint 4	Section 3.46
8431h	FIFO4R2	Transmit and Receive FIFO Register 2 for Endpoint 4	Section 3.46

3.1.2.4 Dynamic FIFO Control Registers

Table 20 lists the dynamic FIFO control registers of the US2.0 Mentor core. Some common USB registers are 8-bit wide and share a word address with other 8-bit registers. Table 19 describes how the common USB registers are laid out in memory.

Table 19. Dynamic FIFO Control Register Layout

CPU Word Address	Register	
	Byte 1	Byte 0
8460h	Reserved	
8461h	RXFIFOSZ	TXFIFOSZ
8464h	TXFIFOADDR	
8465h	RXFIFOADDR	
846Ch	HWVERS	

Table 20. Dynamic FIFO Control Registers

CPU Word Address	Acronym	Register Description	Section
8460h	-	Reserved	
8461h	TXFIFOSZ_RXFIFOSZ	Transmit Endpoint FIFO Size, Receive Endpoint FIFO Size (Index register set to select Endpoints 1-4)	Section 3.48
8464h	TXFIFOADDR	Transmit Endpoint FIFO Address (Index register set to select Endpoints 1-4)	Section 3.50
8465h	RXFIFOADDR	Receive Endpoint FIFO Address (Index register set to select Endpoints 1-4)	Section 3.52
846Ch	HWVERS	Hardware Version Register	Section 3.51

3.1.2.5 Control and Status Registers for Endpoints 0-4

Table 21 lists the control and status registers for endpoints 0-4 of the USB2.0 Mentor Core.

Table 21. Control and Status Registers for Endpoints 0-4

CPU Word Address	Acronym	Register Description	Section
		Control and Status Register for Endpoint 1	
8510h	TXMAXP	Maximum Packet Size for Peripheral/Host Transmit Endpoint	Section 3.34
8511h	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint (peripheral mode)	Section 3.36
8514h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint	Section 3.37
8515h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint (peripheral mode)	Section 3.38
8518h	RXCOUNT	Number of Bytes in Host Receive endpoint FIFO	Section 3.40
8519h	-	Reserved	
851Ch	-	Reserved	
851Dh	-	Reserved	
		Control and Status Register for Endpoint 2	
8520h	TXMAXP	Maximum Packet Size for Peripheral/Host Transmit Endpoint	Section 3.34
8521h	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint (peripheral mode)	Section 3.36
8524h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint	Section 3.37
8525h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint (peripheral mode)	Section 3.38
8528h	RXCOUNT	Number of Bytes in Host Receive endpoint FIFO	Section 3.40
8529h	-	Reserved	
852Ch	-	Reserved	
852Dh	-	Reserved	

Table 21. Control and Status Registers for Endpoints 0-4 (continued)

CPU Word Address	Acronym	Register Description	Section
		Control and Status Register for Endpoint 3	
8530h	TXMAXP	Maximum Packet Size for Peripheral/Host Transmit Endpoint	Section 3.34
8531h	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint (peripheral mode)	Section 3.36
8534h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint	Section 3.37
8535h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint (peripheral mode)	Section 3.38
8538h	RXCOUNT	Number of Bytes in Host Receive endpoint FIFO	Section 3.40
8539h	-	Reserved	
853Ch	-	Reserved	
853Dh	-	Reserved	
		Control and Status Register for Endpoint 4	
8540h	TXMAXP	Maximum Packet Size for Peripheral/Host Transmit Endpoint	Section 3.34
8541h	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint (peripheral mode)	Section 3.36
8544h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint	Section 3.37
8545h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint (peripheral mode)	Section 3.38
8548h	RXCOUNT	Number of Bytes in Host Receive endpoint FIFO	Section 3.40
8549h	-	Reserved	
854Ch	-	Reserved	
854Dh	-	Reserved	

3.1.3 Communications Port Programming Interface (CPPI) 4.1 DMA Registers

This section lists the registers of the communications port programming interface (CPPI) DMA. Refer to the sections listed for detailed information on each register.

3.1.3.1 CPPI DMA (CMDA) Registers

Table 22 lists the register of the CPPI DMA (CMDA).

Table 22. CPPI DMA (CMDA) Registers

CPU Word Address	Acronym	Register Description	Section
9000h	DMAREVID1	CDMA Revision Identification Register 1	Section 3.53
9001h	DMAREVID2	CDMA Revision Identification Register 2	Section 3.53
9004h	TDFDQ	CDMA Teardown Free Descriptor Queue Control Register	Section 3.54
9008h	DMAEMU	CDMA Emulation Control Register	Section 3.55
9800h	TXGCR1[0]	Transmit Channel 0 Global Configuration Register 1	Section 3.56
9801h	TXGCR2[0]	Transmit Channel 0 Global Configuration Register 2	Section 3.56
9808h	RXGCR1[0]	Receive Channel 0 Global Configuration Register 1	Section 3.57
9809h	RXGCR2[0]	Receive Channel 0 Global Configuration Register 2	Section 3.57
980Ch	RXHPCR1A[0]	Receive Channel 0 Host Packet Configuration Register 1 A	Section 3.58
980Dh	RXHPCR2A[0]	Receive Channel 0 Host Packet Configuration Register 2 A	Section 3.58
9810h	RXHPCR1B[0]	Receive Channel 0 Host Packet Configuration Register 1 B	Section 3.59
9811h	RXHPCR2B[0]	Receive Channel 0 Host Packet Configuration Register 2 B	Section 3.59
9820h	TXGCR1[1]	Transmit Channel 1 Global Configuration Register 1	Section 3.56
9821h	TXGCR2[1]	Transmit Channel 1 Global Configuration Register 2	Section 3.56

Table 22. CPPI DMA (CMDA) Registers (continued)

CPU Word Address	Acronym	Register Description	Section
9828h	RXGCR1[1]	Receive Channel 1 Global Configuration Register 1	Section 3.57
9829h	RXGCR2[1]	Receive Channel 1 Global Configuration Register 2	Section 3.57
982Ch	RXHPCR1A[1]	Receive Channel 1 Host Packet Configuration Register 1 A	Section 3.58
982Dh	RXHPCR2A[1]	Receive Channel 1 Host Packet Configuration Register 2 A	Section 3.58
9830h	RXHPCR1B[1]	Receive Channel 1 Host Packet Configuration Register 1 B	Section 3.59
9831h	RXHPCR2B[1]	Receive Channel 1 Host Packet Configuration Register 2 B	Section 3.59
9840h	TXGCR1[2]	Transmit Channel 2 Global Configuration Register 1	Section 3.56
9841h	TXGCR2[2]	Transmit Channel 2 Global Configuration Register 2	Section 3.56
9848h	RXGCR1[2]	Receive Channel 2 Global Configuration Register 1	Section 3.57
9849h	RXGCR2[2]	Receive Channel 2 Global Configuration Register 2	Section 3.57
984Ch	RXHPCR1A[2]	Receive Channel 2 Host Packet Configuration Register 1 A	Section 3.58
984Dh	RXHPCR2A[2]	Receive Channel 2 Host Packet Configuration Register 2 A	Section 3.58
9850h	RXHPCR1B[2]	Receive Channel 2 Host Packet Configuration Register 1 B	Section 3.59
9851h	RXHPCR2B[2]	Receive Channel 2 Host Packet Configuration Register 2 B	Section 3.59
9860h	TXGCR1[3]	Transmit Channel 3 Global Configuration Register 1	Section 3.56
9861h	TXGCR2[3]	Transmit Channel 3 Global Configuration Register 2	Section 3.56
9868h	RXGCR1[3]	Receive Channel 3 Global Configuration Register 1	Section 3.57
9869h	RXGCR2[3]	Receive Channel 3 Global Configuration Register 2	Section 3.57
986Ch	RXHPCR1A[3]	Receive Channel 3 Host Packet Configuration Register 1 A	Section 3.58
986Dh	RXHPCR2A[3]	Receive Channel 3 Host Packet Configuration Register 2 A	Section 3.58
9870h	RXHPCR1B[3]	Receive Channel 3 Host Packet Configuration Register 1 B	Section 3.59
9871h	RXHPCR2B[3]	Receive Channel 3 Host Packet Configuration Register 2 B	Section 3.59
A000h	DMA_SCHED_CTRL1	CDMA Scheduler Control Register 1	Section 3.60
A001h	DMA_SCHED_CTRL2	CDMA Scheduler Control Register 1	Section 3.60
A800h + 4 × *N*	ENTRYLSW[*N*]	CDMA Scheduler Table Word *N* Registers LSW (*N* = 0 to 63)	Section 3.61
A801h + 4 × *N*	ENTRYMSW[*N*]	CDMA Scheduler Table Word *N* Registers MSW (*N* = 0 to 63)	Section 3.61

3.1.3.2 Queue Manager (QMGR) Registers

Table 23 lists the registers of the queue manager.

Table 23. Queue Manager (QMGR) Registers

CPU Word Address	Acronym	Register Description	Section
C000h	QMGRREVID1	Queue Manager Revision Identification Register 1	Section 3.62
C001h	QMGRREVID2	Queue Manager Revision Identification Register 2	Section 3.62
C008h	DIVERSION1	Queue Manager Queue Diversion Register 1	Section 3.63
C009h	DIVERSION2	Queue Manager Queue Diversion Register 2	Section 3.63
C020h	FDBSC0	Queue Manager Free Descriptor/Buffer Starvation Count Register 0	Section 3.64
C021h	FDBSC1	Queue Manager Free Descriptor/Buffer Starvation Count Register 1	Section 3.65
C024h	FDBSC2	Queue Manager Free Descriptor/Buffer Starvation Count Register 2	Section 3.66
C025h	FDBSC3	Queue Manager Free Descriptor/Buffer Starvation Count Register 3	Section 3.67
C028h	FDBSC4	Queue Manager Free Descriptor/Buffer Starvation Count Register 4	Section 3.68
C029h	FDBSC5	Queue Manager Free Descriptor/Buffer Starvation Count Register 5	Section 3.69
C02Ch	FDBSC6	Queue Manager Free Descriptor/Buffer Starvation Count Register 6	Section 3.70
C02Dh	FDBSC7	Queue Manager Free Descriptor/Buffer Starvation Count Register 7	Section 3.71
C080h	LRAM0BASE1	Queue Manager Linking RAM Region 0 Base Address Register 1	Section 3.72

Table 23. Queue Manager (QMGR) Registers (continued)

CPU Word Address	Acronym	Register Description	Section
C081h	LRAM0BASE2	Queue Manager Linking RAM Region 0 Base Address Register 2	Section 3.72
C084h	LRAM0SIZE	Queue Manager Linking RAM Region 0 Size Register	Section 3.73
C085h	-	Reserved	
C088h	LRAM1BASE1	Queue Manager Linking RAM Region 1 Base Address Register 1	Section 3.74
C089h	LRAM1BASE2	Queue Manager Linking RAM Region 1 Base Address Register 2	Section 3.74
C090h	PEND0	Queue Manager Queue Pending 0	Section 3.75
C091h	PEND1	Queue Manager Queue Pending 1	Section 3.76
C094h	PEND2	Queue Manager Queue Pending 2	Section 3.77
C095h	PEND3	Queue Manager Queue Pending 3	Section 3.78
C098h	PEND4	Queue Manager Queue Pending 4	Section 3.79
C099h	PEND5	Queue Manager Queue Pending 5	Section 3.80
D000h + 16 × *R*	QMEMRBASE1[*R*]	Queue Manager Memory Region *R* Base Address Register 1 (*R* = 0 to 15)	Section 3.81
D001h + 16 × *R*	QMEMRBASE2[*R*]	Queue Manager Memory Region *R* Base Address Register 2 (*R* = 0 to 15)	Section 3.81
D004h + 16 × *R*	QMEMRCTRL1[*R*]	Queue Manager Memory Region *R* Control Register (*R* = 0 to 15)	Section 3.82
D005h + 16 × *R*	QMEMRCTRL2[*R*]	Queue Manager Memory Region *R* Control Register (*R* = 0 to 15)	Section 3.82
E00Ch + 16 × *N*	CTRL1D	Queue Manager Queue *N* Control Register 1 D (*N* = 0 to 63)	Section 3.83
E00Dh + 16 × *N*	CTRL2D	Queue Manager Queue *N* Control Register 2 D (*N* = 0 to 63)	Section 3.83
E800h + 16 × *N*	QSTATA	Queue Manager Queue *N* Status Register A (*N* = 0 to 63)	Section 3.84
E804h + 16 × *N*	QSTAT1B	Queue Manager Queue *N* Status Register 1 B (*N* = 0 to 63)	Section 3.85
E805h + 16 × *N*	QSTAT2B	Queue Manager Queue *N* Status Register 2 B (*N* = 0 to 63)	Section 3.85
E808h + 16 × *N*	QSTATC	Queue Manager Queue *N* Status Register C (*N* = 0 to 63)	Section 3.86

3.2 Revision Identification Registers (REVID1 and REVID2)

The revision identification registers (REVID1 and REVID2) contain the revision for the USB 2.0 controller module. The REVID1 is shown in Figure 23 and described in Table 24. The REVID2 is shown in Figure 24 and described in Table 25.

Figure 23. Revision Identification Register (REVID1)

15 0
REVLSB
R- 0800h

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

Figure 24. Revision Identification Register (REVID2)

15 0
REVMSB
R-4EA1h

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

Table 24. Revision Identification Register (REVID1) Field Descriptions

Bit	Field	Value	Description
15-0	REVLSB	0-FFFFh	Least significant bits of the revision ID of the USB module.

Table 25. Revision Identification Register (REVID2) Field Descriptions

Bit	Field	Value	Description
15-0	REVMSB	0-FFFFh	Most significant bits of the revision ID of the USB module.

3.3 Control Register (CTRLR)

The control register (CTRLR) allows the CPU to control various aspects of the module. The CTRLR is shown in Figure 25 and described in Table 26.

Figure 25. Control Register (CTRLR)

15 5	4	3	2	1	0
Reserved	RNDIS	UINT	Reserved	CLKFACK	RESET
R-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0

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

Table 26. Control Register (CTRLR) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved
4	RNDIS		Global RNDIS mode enable for all endpoints.
		0	Global RNDIS mode is disabled.
		1	Global RNDIS mode is enabled.
3	UINT		USB non-PDR interrupt handler enable.
		0	PDR interrupt handler is enabled.
		1	PDR interrupt handler is disabled.
2	Reserved	0	Reserved
1	CLKFACK		Clock stop fast ACK enable.
		0	Clock stop fast ACK is disabled.
		1	Clock stop fast ACK is enabled.
0	RESET		Soft reset.
		0	No effect.
		1	Writing a 1 starts a module reset.

3.4 Status Register (STATR)

The status register (STATR) allows the CPU to check various aspects of the module. The STATR is shown in Figure 26 and described in Table 27.

Figure 26. Status Register (STATR)

15 1	0
Reserved	DRVVBUS
R-0	R-0

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

Table 27. Status Register (STATR) Field Descriptions

Bit	Field	Value	Description
15-1	Reserved	0	Reserved.
0	DRVVBUS		Current DRVVBUS value.
		0	DRVVBUS value is logic 0.
		1	DRVVBUS value is logic 1.

3.5 Emulation Register (EMUR)

The emulation register (EMUR) allows the CPU to configure the CBA 3.0 emulation interface. The EMUR is shown in Figure 27 and described in Table 28.

Figure 27. Emulation Register (EMUR)

15–3	2	1	0
Reserved	RT_SEL	SOFT	FREERUN
R-0	R/W-0	R/W-1	R/W-1

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

Table 28. Emulation Register (EMUR) Field Descriptions

Bit	Field	Value	Description
15-3	Reserved	0	Reserved.
2	RT_SEL		Real-time enable.
		0	Enable.
		1	No effect.
1	SOFT		Soft stop.
		0	No effect.
		1	Soft stop enable.
0	FREERUN		Free run.
		0	No effect.
		1	Free run enable.

3.6 Mode Registers (MODE1 and MODE2)

The mode registers (MODE1 and MODE2) allow the CPU to individually enable RNDIS/Generic/CDC modes for each endpoint. Using the global RNDIS bit in the control register (CTRLR) overrides this register and enables RNDIS mode for all endpoints. The MODE1 is shown in Figure 28 and described in Table 29. The MODE2 is shown in Figure 29 and described in Table 30.

Figure 28. Mode Register 1 (MODE1)

15-14	13-12	11-10	9-8	7-6	5-4	3-2	1-0
Reserved	TX4_MODE	Reserved	TX3_MODE	Reserved	TX2_MODE	Reserved	TX1_MODE
R	R/W	R	R/W	R	R/W	R	R/W

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

Figure 29. Mode Register 2 (MODE2)

15-14	13-12	11-10	9-8	7-6	5-4	3-2	1-0
Reserved	RX4_MODE	Reserved	RX3_MODE	Reserved	RX2_MODE	Reserved	RX1_MODE
R	R/W	R	R/W	R	R/W	R	R/W

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

Table 29. Mode Register 1 (MODE1) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-12	TX4_MODE	0-3h	Transmit endpoint 4 mode control.
		0	Transparent mode on Transmit endpoint 4.
		1h	RNDIS mode on Transmit endpoint 4.
		2h	CDC mode on Transmit endpoint 4.
		3h	Generic RNDIS mode on Transmit endpoint 4.
11-10	Reserved	0	Reserved.
9-8	TX3_MODE	0-3h	Transmit endpoint 3 mode control.
		0	Transparent mode on Transmit endpoint 3.
		1h	RNDIS mode on Transmit endpoint 3.
		2h	CDC mode on Transmit endpoint 3.
		3h	Generic RNDIS mode on Transmit endpoint 3.
7-6	Reserved	0	Reserved.
5-4	TX2_MODE	0-3h	Transmit endpoint 2 mode control.
		0	Transparent mode on Transmit endpoint 2.
		1h	RNDIS mode on Transmit endpoint 2.
		2h	CDC mode on Transmit endpoint 2.
		3h	Generic RNDIS mode on Transmit endpoint 2.
3-2	Reserved	0	Reserved.
1-0	TX1_MODE	0-3h	Transmit endpoint 1 mode control.
		0	Transparent mode on Transmit endpoint 1.
		1h	RNDIS mode on Transmit endpoint 1.
		2h	CDC mode on Transmit endpoint 1.
		3h	Generic RNDIS mode on Transmit endpoint 1.

Table 30. Mode Register 2 (MODE2) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-12	RX4_MODE	0-3h	Receive endpoint 4 mode control.
		0	Transparent mode on Receive endpoint 4.
		1h	RNDIS mode on Receive endpoint 4.
		2h	CDC mode on Receive endpoint 4.
		3h	Generic RNDIS mode on Receive endpoint 4.
11-10	Reserved	0	Reserved.
9-8	RX3_MODE	0-3h	Receive endpoint 3 mode control.
		0	Transparent mode on Receive endpoint 3.
		1h	RNDIS mode on Receive endpoint 3.
		2h	CDC mode on Receive endpoint 3.
		3h	Generic RNDIS mode on Receive endpoint 3.
7-6	Reserved	0	Reserved.
5-4	RX2_MODE	0-3h	Receive endpoint 2 mode control.
		0	Transparent mode on Receive endpoint 2.
		1h	RNDIS mode on Receive endpoint 2.
		2h	CDC mode on Receive endpoint 2.
		3h	Generic RNDIS mode on Receive endpoint 2.
3-2	Reserved	0	Reserved.
1-0	RX1_MODE	0-3h	Receive endpoint 1 mode control.
		0	Transparent mode on Receive endpoint 1.
		1h	RNDIS mode on Receive endpoint 1.
		2h	CDC mode on Receive endpoint 1.
		3h	Generic RNDIS mode on Receive endpoint 1.

3.7 Auto Request Register (AUTOREQ)

The auto request register (AUTOREQ) allows the CPU to enable an automatic IN token request generation for host mode RX operation per each RX endpoint. This feature has the DMA set the REQPKT bit in the control status register for host receive endpoint (HOST_RXCSR) when it clears the RXPKTRDY bit after reading out a packet. The REQPKT bit is used by the core to generate an IN token to receive data. By using this feature, the host can automatically generate an IN token after the DMA finishes receiving data and empties an endpoint buffer, thus receiving the next data packet as soon as possible from the connected device. Without this feature, the CPU will have to manually set the REQPKT bit for every USB packet.

There are two modes that auto request can function in: always or all except an EOP. The always mode sets the REQPKT bit after every USB packet the DMA receives thus generating a new IN token after each USB packet. The EOP mode sets the REQPKT bit after every USB packet that is not an EOP (end of packet) in the CPPI descriptor. For RNDIS, CDC, and Generic RNDIS modes, the auto request stops when the EOP is received (either via a short packet for RNDIS, CDC, and Generic RNDIS or the count is reached for Generic RNDIS), making it useful for starting a large RNDIS packet and having it auto generate IN tokens until the end of the RNDIS packet. For transparent mode, every USB packet is an EOP CPPI packet so the auto request never functions and acts like auto request is disabled.

The AUTOREQ is shown in Figure 30 and described in Table 31.

Figure 30. Auto Request Register (AUTOREQ)

15 8	7 6	5 4	3 2	1 0
Reserved	RX4_AUTREQ	RX3_AUTREQ	RX2_AUTREQ	RX1_AUTREQ
R-0	R/W-0	R/W-0	R/W-0	R/W-0

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

Table 31. Auto Request Register (AUTOREQ) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved.
7-6	RX4_AUTREQ	0-3h	Receive endpoint 4 auto request enable.
		0	No auto request.
		1h	Auto request on all but EOP.
		2h	Reserved.
		3h	Auto request always.
5-4	RX3_AUTREQ	0-3h	Receive endpoint 3 auto request enable.
		0	No auto request.
		1h	Auto request on all but EOP.
		2h	Reserved.
		3h	Auto request always.
3-2	RX2_AUTREQ	0-3h	Receive endpoint 2 auto request enable.
		0	No auto request.
		1h	Auto request on all but EOP.
		2h	Reserved.
		3h	Auto request always.
1-0	RX1_AUTREQ	0-3h	Receive endpoint 1 auto request enable.
		0	No auto request.
		1h	Auto request on all but EOP.
		2h	Reserved.
		3h	Auto request always.

3.8 SRP Fix Time Registers (SRPFIXTIME1 and SRPFIXTIME2)

The SRP fix time registers (SRPFIXTIME1 and SRPFIXTIME2) allow the CPU to configure the maximum amount of time the SRP fix logic blocks the Avalid from the PHY to the Mentor core. The SRPFIXTIME1 is shown in Figure 31 and described in Table 32. The SRPFIXTIME2 is shown in Figure 32 and described in Table 33.

Figure 31. SRP Fix Time Register 1 (SRPFIXTIME1)

15 … 0
SRPFIXTIMELSB
R/W-DE80h

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

Figure 32. SRP Fix Time Register 2 (SRPFIXTIME2)

15 … 0
SRPFIXTIMEMSB
R/W-0280h

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

Table 32. SRP Fix Time Register 1 (SRPFIXTIME1) Field Descriptions

Bit	Field	Value	Description
15-0	SRPFIXTIMELSB	0-FFFFh	SRP fix maximum time in 60 MHz cycles. Together, SRPFIXTIME1 and SRPFIXTIME2 specify a 32 bit value. Default is 700 ms (280 DE80h).

Table 33. SRP Fix Time Register 2 (SRPFIXTIME2) Field Descriptions

Bit	Field	Value	Description
15-0	SRPFIXTIMEMSB	0-FFFFh	SRP fix maximum time in 60 MHz cycles. Together, SRPFIXTIME1 and SRPFIXTIME2 specify a 32 bit value. Default is 700 ms (280 DE80h).

3.9 Teardown Registers (TEARDOWN1 and TEARDOWN2)

The teardown registers (TEARDOWN1 and TEARDOWN2) control the tearing down of receive and transmit FIFOs in the USB controller. When a 1 is written to a valid bit in TEARDOWN1 or TEARDOWN2, the CPPI FIFO pointers for that endpoint are cleared. TEARDOWN1 and TEARDOWN2 must be used in conjunction with the CPPI DMA teardown mechanism. The Host should also write the FLUSHFIFO bits in the TXCSR and RXCSR registers to ensure a complete teardown of the endpoint.

The TEARDOWN1 is shown in Figure 33 and described in Table 34. The TEARDOWN2 is shown in Figure 34 and described in Table 35.

Figure 33. Teardown Register 1 (TEARDOWN1)

15 – 5	4 – 1	0
Reserved	RX_TDOWN	Reserved
R-0	R/W-0	R-0

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

Figure 34. Teardown Register 2 (TEARDOWN2)

15 – 5	4 – 1	0
Reserved	TX_TDOWN	Reserved
R-0	R/W-0	R-0

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

Table 34. Teardown Register 1 (TEARDOWN1) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved.
4-1	RX_TDOWN		Receive endpoint teardown.
		0	Disable.
		1	Enable.
0	Reserved	0	Reserved.

Table 35. Teardown Register 2 (TEARDOWN2) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved.
4-1	TX_TDOWN		Transmit endpoint teardown.
		0	Disable.
		1	Enable.
0	Reserved	0	Reserved.

3.10 USB Interrupt Source Registers (INTSRCR1 and INTSRCR2)

The USB interrupt source registers (INTSRCR1 and INTSRCR2) contain the status of the interrupt sources generated by the USB core (not the DMA). The INTSRCR1 is shown in Figure 35 and described in Table 36. The INTSRCR2 is shown in Figure 36 and described in Table 37.

Figure 35. USB Interrupt Source Register 1 (INTSRCR1)

15 13	12 9	8 5	4 0
Reserved	RX	Reserved	TX
	R-0		R-0

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

Figure 36. USB Interrupt Source Register 2 (INTSRCR2)

15 9	8 0
Reserved	USB
R-0	R-0

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

Table 36. USB Interrupt Source Register 1 (INTSRCR1) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12	RX4	0/1	Receive interrupt source for EndPoint4
11	RX3	0/1	Receive interrupt source for EndPoint3
10	RX2	0/1	Receive interrupt source for EndPoint2
9	RX1	0/1	Receive interrupt source for EndPoint1
8-5	Reserved	0	
4	TX4	0/1	Transmit interrupt source for EndPoint4
3	TX3	0/1	Transmit interrupt source for EndPoint3
2	TX2	0/1	Transmit interrupt source for EndPoint2
1	TX1	0/1	Transmit interrupt source for EndPoint1
0	RX1/TX1	0/1	Both Receive and Transmit interrupt source for EndPoint0

Table 37. USB Interrupt Source Register 2 (INTSRCR2) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved.
8-0	USB	0-1FFh	USB interrupt sources. (Please see Figure 67 for the definition of each bit here.)

3.11 USB Interrupt Source Set Registers (INTSETR1 and INTSETR2)

The USB interrupt source set registers (INTSETR1 and INTSETR2) allow the USB interrupt sources to be manually triggered. A read of this register returns the USB interrupt source register value. The INTSETR1 is shown in Figure 37 and described in Table 38. The INTSETR2 is shown in Figure 38 and described in Table 39.

Figure 37. USB Interrupt Source Set Register 1 (INTSETR1)

15 13	12 9	8 5	4 0
Reserved	RX	Reserved	TX
R-0	R/W-0	R-0	R/W-0

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

Figure 38. USB Interrupt Source Set Register 2 (INTSETR2)

15 9	8 0
Reserved	USB
R-0	R/W-0

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

Table 38. USB Interrupt Source Set Register 1 (INTSETR1) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12-8	RX	0-Fh	Write a 1 to set equivalent Receive endpoint interrupt source. Allows the USB interrupt sources to be manually triggered.
7-5	Reserved	0	Reserved.
4-0	TX	0-1Fh	Write a 1 to set equivalent Transmit endpoint interrupt source. Allows the USB interrupt sources to be manually triggered.

Table 39. USB Interrupt Source Set Register 2 (INTSETR2) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved.
8-0	USB	0-1FFh	Write a 1 to set equivalent USB interrupt source. Allows the USB interrupt sources to be manually triggered.

3.12 USB Interrupt Source Clear Registers (INTCLRR1 and INTCLRR2)

The USB interrupt source clear registers (INTCLRR1 and INTCLRR2) allow the CPU to acknowledge an interrupt source and turn it off. A read of this register returns the USB interrupt source register value. The INTCLRR1 is shown in Figure 39 and described in Table 40. The INTCLRR2 is shown in Figure 40 and described in Table 41.

Figure 39. USB Interrupt Source Clear Register 1 (INTCLRR1)

15 13	12 9	8 5	4 0
Reserved	RX	Reserved	TX
R-0	R/W-0	R-0	R/W-0

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

Figure 40. USB Interrupt Source Clear Register 2 (INTCLRR2)

15 9	8 0
Reserved	USB
R-0	R/W-0

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

Table 40. USB Interrupt Source Clear Register 1 (INTCLRR1) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12-8	RX	0-Fh	Write a 1 to clear equivalent Receive endpoint interrupt source. Allows the CPU to acknowledge an interrupt source and turn it off.
7-5	Reserved	0	Reserved.
4-0	TX	0-1Fh	Write a 1 to clear equivalent Transmit endpoint interrupt source. Allows the CPU to acknowledge an interrupt source and turn it off.

Table 41. USB Interrupt Source Clear Register 2 (INTCLRR2) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved.
8-0	USB	0-1FFh	Write a 1 to clear equivalent USB interrupt source. Allows the CPU to acknowledge an interrupt source and turn it off.

3.13 USB Interrupt Mask Registers (INTMSKR1 and INTMSKR2)

The USB interrupt mask registers (INTMSKR1 and INTMSKR2) contain the masks of the interrupt sources generated by the USB core (not the DMA). These masks are used to enable or disable interrupt sources generated on the masked source interrupts (the raw source interrupts are never masked). The bit positions are maintained in the same position as the interrupt sources in the USB interrupt source register.

The INTMSKR1 is shown in Figure 41 and described in Table 42. The INTMSKR2 is shown in Figure 42 and described in Table 43.

Figure 41. USB Interrupt Mask Register 1 (INTMSKR1)

15-13	12-9	8-5	4-0
Reserved	RX	Reserved	TX
R-0	R-0	R-0	R-0

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

Figure 42. USB Interrupt Mask Register 2 (INTMSKR2)

15-9	8-0
Reserved	USB
R-0	R-0

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

Table 42. USB Interrupt Mask Register 1 (INTMSKR1) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12-8	RX	0-Fh	Receive endpoint interrupt source masks.
7-5	Reserved	0	Reserved.
4-0	TX	0-1Fh	Transmit endpoint interrupt source masks.

Table 43. USB Interrupt Mask Register 2 (INTMSKR2) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved.
8-0	USB	0-1FFh	USB interrupt source masks.

3.14 USB Interrupt Mask Set Registers (INTMSKSETR1 and INTMSKSETR2)

The USB interrupt mask set registers (INTMSKSETR1 and INTMSKSETR2) allow the USB masks to be individually enabled. A read to this register returns the USB interrupt mask register value. The INTMSKSETR1 is shown in Figure 43 and described in Table 44. The INTMSKSETR2 is shown in Figure 44 and described in Table 45.

Figure 43. USB Interrupt Mask Set Register 1 (INTMSKSETR1)

15 13	12 9	8 5	4 0
Reserved	RX	Reserved	TX
R-0	R/W-0	R-0	R/W-0

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

Figure 44. USB Interrupt Mask Set Register 2 (INTMSKSETR2)

15 9	8 0
Reserved	USB
R-0	R/W-0

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

Table 44. USB Interrupt Mask Set Register 1 (INTMSKSETR1) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12-8	RX	0-Fh	Write a 1 to set equivalent Receive endpoint interrupt mask.
7-5	Reserved	0	Reserved.
4-0	TX	0-1Fh	Write a 1 to set equivalent Transmit endpoint interrupt mask.

Table 45. USB Interrupt Mask Set Register 2 (INTMSKSETR2) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved.
8-0	USB	0-1FFh	Write a 1 to set equivalent USB interrupt mask.

3.15 USB Interrupt Mask Clear Registers (INTMSKCLRR1 and INTMSKCLRR2)

The USB interrupt mask clear registers (INTMSKCLRR1 and INTMSKCLRR2) allow the USB interrupt masks to be individually disabled. A read to this register returns the USB interrupt mask register value. The INTMSKCLRR1 is shown in Figure 45 and described in Table 46. The INTMSKCLRR2 is shown in Figure 46 and described in Table 47.

Figure 45. USB Interrupt Mask Clear Register 1 (INTMSKCLRR1)

15 13	12 9	8 5	4 0
Reserved	RX	Reserved	TX
R-0	R/W-0	R-0	R/W-0

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

Figure 46. USB Interrupt Mask Clear Register 2 (INTMSKCLRR2)

15 9	8 0
Reserved	USB
R-0	R/W-0

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

Table 46. USB Interrupt Mask Clear Register 1 (INTMSKCLRR1) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12-8	RX	0-Fh	Write a 1 to clear equivalent Receive endpoint interrupt mask.
7-5	Reserved	0	Reserved.
4-0	TX	0-1Fh	Write a 1 to clear equivalent Transmit endpoint interrupt mask.

Table 47. USB Interrupt Mask Clear Register 2 (INTMSKCLRR2) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved.
8-0	USB	0-1FFh	Write a 1 to clear equivalent USB interrupt mask.

3.16 USB Interrupt Source Masked Registers (INTMASKEDR1 and INTMASKEDR2)

The USB interrupt source masked registers (INTMASKEDR1 and INTMASKEDR2) contain the status of the interrupt sources generated by the USB core masked by the USB interrupt mask register values. The INTMASKEDR1 is shown in Figure 47 and described in Table 48. The INTMASKEDR2 is shown in Figure 48 and described in Table 49.

Figure 47. USB Interrupt Source Masked Register 1 (INTMASKEDR1)

15 - 13	12 - 9	8 - 5	4 - 0
Reserved	RX	Reserved	TX
R-0	R-0	R-0	R-0

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

Figure 48. USB Interrupt Source Masked Register 2 (INTMASKEDR2)

15 - 9	8 - 0
Reserved	USB
R-0	R-0

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

Table 48. USB Interrupt Source Masked Register 1 (INTMASKEDR1) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12-8	RX	0-Fh	Receive endpoint interrupt sources masked.
7-5	Reserved	0	Reserved.
4-0	TX	0-1Fh	Transmit endpoint interrupt sources masked.

Table 49. USB Interrupt Source Masked Register 2 (INTMASKEDR2) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved.
8-0	USB	0-1FFh	USB interrupt sources masked.

3.17 USB End of Interrupt Register (EOIR)

The USB end of interrupt register (EOIR) allows the CPU to acknowledge completion of an interrupt by writing 0 to the EOI_VECTOR bit. The EOIR is shown in Figure 49 and described in Table 50.

Figure 49. USB End of Interrupt Register (EOIR)

15 8	7 0
Reserved	EOI_VECTOR
R-0	R/W-0

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

Table 50. USB End of Interrupt Register (EOIR) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved.
7-0	EOI_VECTOR	0-FFh	EOI Vector.

3.18 USB Interrupt Vector Registers (INTVECTR1 and INTVECTR2)

The USB interrupt vector registers (INTVECTR1 and INTVECTR2) recycle the Interrupt Vector input to be read by the CPU. The INTVECTR1 is shown in Figure 50 and described in Table 51. The INTVECTR2 is shown in Figure 51 and described in Table 52.

Figure 50. USB Interrupt Vector Register 1 (INTVECTR1)

15 0
VECTORLSB
R-0

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

Figure 51. USB Interrupt Vector Register 2 (INTVECTR2)

15 0
VECTORMSB
R-0

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

Table 51. USB Interrupt Vector Register 1 (INTVECTR1) Field Descriptions

Bit	Field	Value	Description
15-0	VECTORLSB	0-FFFFh	Input Interrupt Vector. Together, INTVECTR1 and INTVECTR2 specify a 32 bit value.

Table 52. USB Interrupt Vector Register 2 (INTVECTR2) Field Descriptions

Bit	Field	Value	Description
15-0	VECTORMSB	0-FFFFh	Input Interrupt Vector. Together, INTVECTR1 and INTVECTR2 specify a 32 bit value.

3.19 Generic RNDIS EP1 Size Registers (GREP1SZR1 and GREP1SZR2)

The generic RNDIS EP1 size registers (GREP1SZR1 and GREP1SZR2) are programmed with a RNDIS packet size in bytes. When EP1 is in Generic RNDIS mode, the received USB packets are collected into a single CPPI packet that is completed when the number of bytes equal to the value of this register have been received, or a *short* packet is received. The packet size must be an integer multiple of the endpoint size. The maximum packet size that can be used is 10000h, or 65536.

The GREP1SZR1 is shown in Figure 52 and described in Table 53. The GREP1SZR2 is shown in Figure 53 and described in Table 54.

Figure 52. Generic RNDIS EP1 Size Register 1 (GREP1SZR1)

15 — 0
EP1_SIZE_LSB
R/W-0

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

Figure 53. Generic RNDIS EP1 Size Register 2 (GREP1SZR2)

15 — 1	0
Reserved	EP1_SIZE_MSB
R-0	R/W-0

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

Table 53. Generic RNDIS EP1 Size Register 1 (GREP1SZR1) Field Descriptions

Bit	Field	Value	Description
15-0	EP1_SIZE_LSB	0-FFFFh	Generic RNDIS packet size. Together, GREP1SZR1 and GREP1SZR2 specify the packet size.

Table 54. Generic RNDIS EP1 Size Register 2 (GREP1SZR2) Field Descriptions

Bit	Field	Value	Description
15-1	Reserved	0	Reserved.
0	EP1_SIZE_MSB	0-1	Generic RNDIS packet size. Together, GREP1SZR1 and GREP1SZR2 specify the packet size.

3.20 Generic RNDIS EP2 Size Registers (GREP2SZR1 and GREP2SZR2)

The generic RNDIS EP2 size registers (GREP2SZR1 and GREP2SZR2) are programmed with a RNDIS packet size in bytes. When EP2 is in Generic RNDIS mode, the received USB packets are collected into a single CPPI packet that is completed when the number of bytes equal to the value of this register have been received, or a *short* packet is received. The packet size must be an integer multiple of the endpoint size. The maximum packet size that can be used is 10000h, or 65536.

The GREP2SZR1 is shown in Figure 54 and described in Table 55. The GREP2SZR2 is shown in Figure 55 and described in Table 56.

Figure 54. Generic RNDIS EP2 Size Register 1 (GREP2SZR1)

15 — 0
EP2_SIZE_LSB
R/W-0

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

Figure 55. Generic RNDIS EP2 Size Register 2 (GREP2SZR2)

15 — 1	0
Reserved	EP2_SIZE_MSB
R-0	R/W-0

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

Table 55. Generic RNDIS EP2 Size Register 1 (GREP2SZR1) Field Descriptions

Bit	Field	Value	Description
15-0	EP2_SIZE_LSB	0-FFFFh	Generic RNDIS packet size. Together, GREP2SZR1 and GREP2SZR2 specify the packet size.

Table 56. Generic RNDIS EP2 Size Register 2 (GREP2SZR2) Field Descriptions

Bit	Field	Value	Description
15-1	Reserved	0	Reserved.
0	EP2_SIZE_MSB	0-1	Generic RNDIS packet size. Together, GREP2SZR1 and GREP2SZR2 specify the packet size.

3.21 Generic RNDIS EP3 Size Registers (GREP3SZR1 and GREP3SZR2)

The generic RNDIS EP3 size registers (GREP3SZR1 and GREP3SZR2) are programmed with a RNDIS packet size in bytes. When EP3 is in Generic RNDIS mode, the received USB packets are collected into a single CPPI packet that is completed when the number of bytes equal to the value of this register has been received, or a *short* packet is received. The packet value must be an integer multiple of the endpoint size. The maximum packet size that can be used is 10000h, or 65536.

The GREP3SZR1 is shown in Figure 56 and described in Table 57. The GREP3SZR2 is shown in Figure 57 and described in Table 58.

Figure 56. Generic RNDIS EP3 Size Register 1 (GREP3SZR1)

15   0
EP3_SIZE_LSB
R/W-0

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

Figure 57. Generic RNDIS EP3 Size Register 2 (GREP3SZR2)

15 1	0
Reserved	EP3_SIZE_MSB
R-0	R/W-0

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

Table 57. Generic RNDIS EP3 Size Register 1 (GREP3SZR1) Field Descriptions

Bit	Field	Value	Description
15-0	EP3_SIZE_LSB	0-FFFFh	Generic RNDIS packet size. Together, GREP3SZR1 and GREP3SZR2 specify the packet size.

Table 58. Generic RNDIS EP3 Size Register 2 (GREP3SZR2) Field Descriptions

Bit	Field	Value	Description
15-1	Reserved	0	Reserved.
0	EP3_SIZE_MSB	0-1	Generic RNDIS packet size. Together, GREP3SZR1 and GREP3SZR2 specify the packet size.

3.22 Generic RNDIS EP4 Size Registers (GREP4SZR1 and GREP4SZR2)

The generic RNDIS EP4 size registers (GREP4SZR1 and GREP4SZR2) are programmed with a RNDIS packet size in bytes. When EP4 is in Generic RNDIS mode, the received USB packets are collected into a single CPPI packet that is completed when the number of bytes equal to the value of this register has been received, or a *short* packet is received. The packet size must be an integer multiple of the endpoint size. The maximum packet size that can be used is 10000h, or 65536.

The GREP4SZR1 is shown in Figure 58 and described in Table 59. The GREP4SZR2 is shown in Figure 59 and described in Table 60.

Figure 58. Generic RNDIS EP4 Size Register 1 (GREP4SZR1)

15 - 0
EP4_SIZE_LSB
R/W-0

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

Figure 59. Generic RNDIS EP4 Size Register 2 (GREP4SZR2)

15 - 1	0
Reserved	EP4_SIZE_MSB
R-0	R/W-0

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

Table 59. Generic RNDIS EP4 Size Register 1 (GREP4SZR1) Field Descriptions

Bit	Field	Value	Description
15-0	EP4_SIZE_LSB	0-FFFFh	Generic RNDIS packet size. Together, GREP4SZR1 and GREP4SZR2 specify the packet size.

Table 60. Generic RNDIS EP4 Size Register 2 (GREP4SZR2) Field Descriptions

Bit	Field	Value	Description
15-1	Reserved	0	Reserved
0	EP4_SIZE_MSB	0-1	Generic RNDIS packet size. Together, GREP4SZR1 and GREP4SZR2 specify the packet size.

3.23 Function Address Register (FADDR)

The function address register (FADDR) is shown in Figure 60 and described in Table 61.

Figure 60. Function Address Register (FADDR)

7	6 ... 0
Reserved	FUNCADDR
R-0	R/W-0

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

Table 61. Function Address Register (FADDR) Field Descriptions

Bit	Field	Value	Description
7	Reserved	0	Reserved.
6-0	FUNCADDR	0-7Fh	7_bit address of the peripheral part of the transaction. This register should be written with the address received through a SET_ADDRESS command, which will then be used for decoding the function address in subsequent token packets. When used in Host mode, this register should be set to the value sent in a SET_ADDRESS command during device enumeration as the address for the peripheral device.

3.24 Power Management Register (POWER)

The power management register (POWER) is shown in Figure 61 and described in Table 62.

Figure 61. Power Management Register (POWER)

7	6	5	4	3	2	1	0
ISOUPDATE	SOFTCONN	HSEN	HSMODE	RESET	RESUME	SUSPENDM	ENSUSPM
R/W-0	R/W-0	R/W-1	R-0	R-0	R/W-0	R/W-0	R/W-0

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

Table 62. Power Management Register (POWER) Field Descriptions

Bit	Field	Value	Description
7	ISOUPDATE	0-1	When set, the USB controller will wait for an SOF token from the time TxPktRdy is set before sending the packet. If an IN token is received before an SOF token, then a zero length data packet will be sent. This bit only affects endpoints performing Isochronous transfers.
6	SOFTCONN	0-1	If Soft Connect/Disconnect feature is enabled, then the USB D+/D- lines are enabled when this bit is set and tri-stated when this bit is cleared.
5	HSEN	0-1	When set, the USB controller will negotiate for high-speed mode when the device is reset by the hub. If not set, the device will only operate in full-speed mode.
4	HSMODE	0-1	This bit is set when the USB controller has successfully negotiated for high-speed mode.
3	RESET	0-1	This bit is set when Reset signaling is present on the bus. Note: this bit is read-only.
2	RESUME	0-1	Set to generate Resume signaling when the controller is in Suspend mode. The bit should be cleared after 10 ms (a maximum of 15 ms) to end Resume signaling. In Host mode, this bit is also automatically set when Resume signaling from the target is detected while the USB controller is suspended.
1	SUSPENDM	0-1	This bit is set on entry into Suspend mode. It is cleared when the interrupt register is read, or the RESUME bit is set.
0	ENSUSPM	0-1	Set to enable the SUSPENDM output.

3.25 Interrupt Register for Endpoint 0 Plus Transmit Endpoints 1 to 4 (INTRTX)

The interrupt register for endpoint 0 plus transmit endpoints 1 to 4 (INTRTX) is shown in Figure 62 and described in Table 63.

Figure 62. Interrupt Register for Endpoint 0 Plus Tx Endpoints 1 to 4 (INTRTX)

15–8
Reserved
R-0

7–5	4	3	2	1	0
Reserved	EP4TX	EP3TX	EP2TX	EP1TX	EP0
R-0	R-0	R-0	R-0	R-0	R-0

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

Table 63. Interrupt Register for Endpoint 0 Plus Transmit Endpoints 1 to 4 (INTRTX) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved.
4	EP4TX	0-1	Transmit Endpoint 4 interrupt active.
3	EP3TX	0-1	Transmit Endpoint 3 interrupt active.
2	EP2TX	0-1	Transmit Endpoint 2 interrupt active.
1	EP1TX	0-1	Transmit Endpoint 1 interrupt active.
0	EP0	0-1	Endpoint 0 interrupt active.

3.26 Interrupt Register for Receive Endpoints 1 to 4 (INTRRX)

The interrupt register for receive endpoints 1 to 4 (INTRRX) is shown in Figure 63 and described in Table 64.

Figure 63. Interrupt Register for Receive Endpoints 1 to 4 (INTRRX)

15–8
Reserved
R-0

7–5	4	3	2	1	0
Reserved	EP4RX	EP3RX	EP2RX	EP1RX	Reserved
R-0	R-0	R-0	R-0	R-0	R-0

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

Table 64. Interrupt Register for Receive Endpoints 1 to 4 (INTRRX) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved.
4	EP4RX	0-1	Receive Endpoint 4 interrupt active.
3	EP3RX	0-1	Receive Endpoint 3 interrupt active.
2	EP2RX	0-1	Receive Endpoint 2 interrupt active.
1	EP1RX	0-1	Receive Endpoint 1 interrupt active.
0	Reserved	0	Reserved.

3.27 Interrupt Enable Register for INTRTX (INTRTXE)

The interrupt enable register for INTRTX (INTRTXE) is shown in Figure 64 and described in Table 65.

Figure 64. Interrupt Enable Register for INTRTX (INTRTXE)

15 – 8
Reserved
R-0

7 – 5	4	3	2	1	0
Reserved	EP4TX	EP3TX	EP2TX	EP1TX	EP0
R-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1

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

Table 65. Interrupt Enable Register for INTRTX (INTRTXE) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved.
4	EP4TX	0-1	Transmit Endpoint 4 interrupt active.
3	EP3TX	0-1	Transmit Endpoint 3 interrupt active.
2	EP2TX	0-1	Transmit Endpoint 2 interrupt active.
1	EP1TX	0-1	Transmit Endpoint 1 interrupt active.
0	EP0	0-1	Endpoint 0 interrupt active.

3.28 Interrupt Enable Register for INTRRX (INTRRXE)

The interrupt enable register for INTRRX (INTRRXE) is shown in Figure 65 and described in Table 66.

Figure 65. Interrupt Enable Register for INTRRX (INTRRXE)

15 – 8
Reserved
R-0

7 – 5	4	3	2	1	0
Reserved	EP4RX	EP3RX	EP2RX	EP1RX	Reserved
R-0	R/W-1	R/W-1	R/W-1	R/W-1	R-0

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

Table 66. Interrupt Enable Register for INTRRX (INTRRXE) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved.
4	EP4RX	0-1	Receive Endpoint 4 interrupt active.
3	EP3RX	0-1	Receive Endpoint 3 interrupt active.
2	EP2RX	0-1	Receive Endpoint 2 interrupt active.
1	EP1RX	0-1	Receive Endpoint 1 interrupt active.
0	Reserved	0	Reserved.

3.29 Interrupt Register for Common USB Interrupts (INTRUSB)

The interrupt register for common USB interrupts (INTRUSB) is shown in Figure 66 and described in Table 67.

NOTE: Unless the UINT bit of CTRLR is set, do not read or write this register directly. Use the INTSRCR register instead.

Figure 66. Interrupt Register for Common USB Interrupts (INTRUSB)

7	6	5	4	3	2	1	0
VBUSERR	SESSREQ	DISCON	CONN	SOF	RESET_BABBLE	RESUME	SUSPEND
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

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

Table 67. Interrupt Register for Common USB Interrupts (INTRUSB) Field Descriptions

Bit	Field	Value	Description
7	VBUSERR	0-1	Set when VBus drops below the VBus valid threshold during a session. Only valid when the USB controller is 'A' device. All active interrupts will be cleared when this register is read.
6	SESSREQ	0-1	Set when session request signaling has been detected. Only valid when USB controller is 'A' device.
5	DISCON	0-1	Set when a session ends.
4	CONN	0-1	Set when a device connection is detected. Only valid in host mode.
3	SOF	0-1	Set when a new frame starts.
2	RESET_BABBLE	0-1	Set when reset signaling is detected on the bus.
1	RESUME	0-1	Set when resume signaling is detected on the bus while the USB controller is in suspend mode.
0	SUSPEND	0-1	Set when suspend signaling is detected on the bus.

3.30 Interrupt Enable Register for INTRUSB (INTRUSBE)

The interrupt enable register for INTRUSB (INTRUSBE) is shown in Figure 67 and described in Table 68.

NOTE: Unless the UINT bit of CTRLR is set, do not read or write this register directly. Use the INTSETR/INTCLRR registers instead.

Figure 67. Interrupt Enable Register for INTRUSB (INTRUSBE)

7	6	5	4	3	2	1	0
VBUSERR	SESSREQ	DISCON	CONN	SOF	RESET_BABBLE	RESUME	SUSPEND
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-0

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

Table 68. Interrupt Enable Register for INTRUSB (INTRUSBE) Field Descriptions

Bit	Field	Value	Description
7	VBUSERR	0-1	Vbus error interrupt enable.
6	SESSREQ	0-1	Session request interrupt enable.
5	DISCON	0-1	Disconnect interrupt enable.
4	CONN	0-1	Connect interrupt enable.
3	SOF	0-1	Start of frame interrupt enable.
2	RESET_BABBLE	0-1	Reset interrupt enable.
1	RESUME	0-1	Resume interrupt enable.
0	SUSPEND	0-1	Suspend interrupt enable.

3.31 Frame Number Register (FRAME)

The frame number register (FRAME) is shown in Figure 68 and described in Table 69.

Figure 68. Frame Number Register (FRAME)

15 - 11	10 - 0
Reserved	FRAMENUMBER
R-0	R-0

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

Table 69. Frame Number Register (FRAME) Field Descriptions

Bit	Field	Value	Description
15-11	Reserved	0	Reserved.
10-0	FRAMENUMBER	0-7FFh	Last received frame number.

3.32 Index Register for Selecting the Endpoint Status and Control Registers (INDEX)

The index register for selecting the endpoint status and control registers (INDEX) is shown in Figure 69 and described in Table 70.

Figure 69. Index Register for Selecting the Endpoint Status and Control Registers (INDEX)

7-4	3-0
Reserved	EPSEL
R-0	R/W-0

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

Table 70. Index Register for Selecting the Endpoint Status and Control Registers (INDEX) Field Descriptions

Bit	Field	Value	Description
7-4	Reserved	0	Reserved.
3-0	EPSEL	0-Fh	Each transmit endpoint and each receive endpoint have their own set of control/status registers. EPSEL determines which endpoint control/status registers are accessed. Before accessing an endpoint's control/status registers, the endpoint number should be written to the Index register to ensure that the correct control/status registers appear in the memory-map.

3.33 Register to Enable the USB 2.0 Test Modes (TESTMODE)

The register to enable the USB 2.0 test modes (TESTMODE) is shown in Figure 70 and described in Table 71.

Figure 70. Register to Enable the USB 2.0 Test Modes (TESTMODE)

7	6	5	4	3	2	1	0
FORCE_HOST	FIFO_ACCESS	FORCE_FS	FORCE_HS	TEST_PACKET	TEST_K	TEST_J	TEST_SE0_NAK
R/W-0	W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

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

Table 71. Register to Enable the USB 2.0 Test Modes (TESTMODE) Field Descriptions

Bit	Field	Value	Description
7	FORCE_HOST	0-1	Set this bit to forcibly put the USB controller into Host mode when SESSION bit is set, regardless of whether it is connected to any peripheral. The controller remains in Host mode until the Session bit is cleared, even if a device is disconnected. And if the FORCE_HOST but remains set, it will re-enter Host mode next time the SESSION bit is set. The operating speed is determined using the FORCE_HS and FORCE_FS bits.
6	FIFO_ACCESS	0-1	Set this bit to transfer the packet in EP0 Tx FIFO to EP0 Receive FIFO. It is cleared automatically.
5	FORCE_FS	0-1	Set this bit to force the USB controller into full-speed mode when it receives a USB reset.
4	FORCE_HS	0-1	Set this bit to force the USB controller into high-speed mode when it receives a USB reset.
3	TEST_PACKET	0-1	Set this bit to enter the Test_Packet test mode. In this mode, the USB controller repetitively transmits a 53-byte test packet on the bus, the form of which is defined in the Universal Serial Bus Specification Revision 2.0. Note: The test packet has a fixed format and must be loaded into the Endpoint 0 FIFO before the test mode is entered.
2	TEST_K	0-1	Set this bit to enter the Test_K test mode. In this mode, the USB controller transmits a continuous K on the bus.
1	TEST_J	0-1	Set this bit to enter the Test_J test mode. In this mode, the USB controller transmits a continuous J on the bus.
0	TEST_SE0_NAK	0-1	Set this bit to enter the Test_SE0_NAK test mode. In this mode, the USB controller remains in high-speed mode, but responds to any valid IN token with a NAK.

3.34 Maximum Packet Size for Peripheral/Host Transmit Endpoint (TXMAXP)

The maximum packet size for peripheral/host transmit endpoint (TXMAXP) is shown in Figure 71 and described in Table 72.

Figure 71. Maximum Packet Size for Peripheral/Host Transmit Endpoint (TXMAXP)

15 - 11	10 - 0
Reserved	MAXPAYLOAD
R-0	R/W-0

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

Table 72. Maximum Packet Size for Peripheral/Host Transmit Endpoint (TXMAXP) Field Descriptions

Bit	Field	Value	Description
15-11	Reserved	0	Reserved.
10-0	MAXPAYLOAD	0-FFh	The maximum payload transmitted in a single transaction. The value set can be up to 1024 bytes, but is subject to the constraints placed by the USB Specification on packet sizes for Bulk, Interrupt, and Isochronous transfers in full-speed and high-speed operations. The value written to this register should match the wMaxPacketSize field of the Standard Endpoint Descriptor for the associated endpoint. A mismatch could cause unexpected results.

3.35 Control Status Register for Peripheral Endpoint 0 (PERI_CSR0)

The control status register for peripheral endpoint 0 (PERI_CSR0) is shown in Figure 72 and described in Table 73.

Figure 72. Control Status Register for Peripheral Endpoint 0 (PERI_CSR0)

15–9	8
Reserved	FLUSHFIFO
R-0	W-0

7	6	5	4	3	2	1	0
SERV_SETUPEND	SERV_RXPKTRDY	SENDSTALL	SETUPEND	DATAEND	SENTSTALL	TXPKTRDY	RXPKTRDY
W-0	W-0	W-0	R-0	W-0	R/W-0	R/W-0	R-0

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

Table 73. Control Status Register for Peripheral Endpoint 0 (PERI_CSR0) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved.
8	FLUSHFIFO	0-1	Set this bit to flush the next packet to be transmitted/read from the Endpoint 0 FIFO. The FIFO pointer is reset and the TXPKTRDY/RXPKTRDY bit is cleared. Note: FLUSHFIFO has no effect unless TXPKTRDY/RXPKTRDY is set.
7	SERV_SETUPEND	0-1	Set this bit to clear the SETUPEND bit. It is cleared automatically.
6	SERV_RXPKTRDY	0-1	Set this bit to clear the RXPKTRDY bit. It is cleared automatically.
5	SENDSTALL	0-1	Set this bit to terminate the current transaction. The STALL handshake will be transmitted and then this bit will be cleared automatically.
4	SETUPEND	0-1	This bit will be set when a control transaction ends before the DATAEND bit has been set. An interrupt will be generated, and the FIFO will be flushed at this time. The bit is cleared by the writing a 1 to the SERV_SETUPEND bit.
3	DATAEND	0-1	Set this bit to 1: a. When setting TXPKTRDY for the last data packet. b. When clearing RXPKTRDY after unloading the last data packet. c. When setting TXPKTRDY for a zero length data packet. It is cleared automatically.
2	SENTSTALL	0-1	This bit is set when a STALL handshake is transmitted. This bit should be cleared.
1	TXPKTRDY	0-1	Set this bit after loading a data packet into the FIFO. It is cleared automatically when the data packet has been transmitted. An interrupt is generated (if enabled) when the bit is cleared.
0	RXPKTRDY	0-1	This bit is set when a data packet has been received. An interrupt is generated when this bit is set. This bit is cleared by setting the SERV_RXPKTRDY bit.

3.36 Control Status Register for Peripheral Transmit Endpoint (PERI_TXCSR)

The control status register for peripheral transmit endpoint (PERI_TXCSR) is shown in Figure 73 and described in Table 74.

Figure 73. Control Status Register for Peripheral Transmit Endpoint (PERI_TXCSR)

15	14	13	12	11	10	9-7
AUTOSET	ISO	MODE	DMAEN	FRCDATATOG	DMAMODE	Reserved
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0

6	5	4	3	2	1	0
CLRDATATOG	SENTSTALL	SENDSTALL	FLUSHFIFO	UNDERRUN	FIFONOTEMPTY	TXPKTRDY
W-0	R/W-0	R/W-0	W-0	R/W-0	R/W-0	R/W-0

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

Table 74. Control Status Register for Peripheral Transmit Endpoint (PERI_TXCSR) Field Descriptions

Bit	Field	Value	Description
15	AUTOSET	0	DMA Mode: The CPU needs to set the AUTOSET bit prior to enabling the Tx DMA.
		1	CPU Mode: If the CPU sets the AUTOSET bit, the TXPKTRDY bit will be automatically set when data of the maximum packet size (value in TXMAXP) is loaded into the Tx FIFO. If a packet of less than the maximum packet size is loaded, then the TXPKTRDY bit will have to be set manually.
14	ISO	0-1	Set this bit to enable the Tx endpoint for Isochronous transfers, and clear it to enable the Tx endpoint for Bulk or Interrupt transfers.
13	MODE	0-1	Set this bit to enable the endpoint direction as Tx, and clear the bit to enable it as Rx. Note: This bit has any effect only where the same endpoint FIFO is used for both Transmit and Receive transactions.
12	DMAEN	0-1	Set this bit to enable the DMA request for the Tx endpoint.
11	FRCDATATOG	0-1	Set this bit to force the endpoint data toggle to switch and the data packet to be cleared from the FIFO, regardless of whether an ACK was received. This can be used by Interrupt Tx endpoints that are used to communicate rate feedback for Isochronous endpoints.
10	DMAMODE	0-1	Set to 1 when DMA is enabled and EP interrupt is not needed for each packet transmission.
9-7	Reserved	0	Reserved.
6	CLRDATATOG	0-1	Write a 1 to this bit to reset the endpoint data toggle to 0.
5	SENTSTALL	0-1	This bit is set automatically when a STALL handshake is transmitted. The FIFO is flushed and the TXPKTRDY bit is cleared. You should clear this bit.
4	SENDSTALL	0-1	Write a 1 to this bit to issue a STALL handshake to an IN token. Clear this bit to terminate the stall condition. Note: This bit has no effect where the endpoint is being used for Isochronous transfers.
3	FLUSHFIFO	0-1	Write a 1 to this bit to flush the next packet to be transmitted from the endpoint Tx FIFO. The FIFO pointer is reset and the TXPKTRDY bit is cleared. Note: FlushFIFO has no effect unless the TXPKTRDY bit is set. Also note that, if the FIFO is double-buffered, FlushFIFO may need to be set twice to completely clear the FIFO.
2	UNDERRUN	0-1	This bit is set automatically if an IN token is received when TXPKTRDY is not set. You should clear this bit.
1	FIFONOTEMPTY	0-1	This bit is set when there is at least 1 packet in the Tx FIFO. You should clear this bit.
0	TXPKTRDY	0-1	Set this bit after loading a data packet into the FIFO. It is cleared automatically when a data packet has been transmitted. An interrupt is generated (if enabled) when the bit is cleared.

3.37 Maximum Packet Size for Peripheral Receive Endpoint (RXMAXP)

The maximum packet size for peripheral receive endpoint (RXMAXP) is shown in Figure 74 and described in Table 75.

Figure 74. Maximum Packet Size for Peripheral Receive Endpoint (RXMAXP)

15 - 11	10 - 0
Reserved	MAXPAYLOAD
R-0	R/W-0

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

Table 75. Maximum Packet Size for Peripheral Receive Endpoint (RXMAXP) Field Descriptions

Bit	Field	Value	Description
15-11	Reserved	0	Reserved.
10-0	MAXPAYLOAD	0-FFh	Defines the maximum amount of data that can be transferred through the selected Receive endpoint in a single frame/microframe (high-speed transfers). The value set can be up to 1024 bytes, but is subject to the constraints placed by the USB Specification on packet sizes for Bulk, Interrupt, and Isochronous transfers in full-speed and high-speed operations. The value written to this register should match the wMaxPacketSize field of the Standard Endpoint Descriptor for the associated endpoint. A mismatch could cause unexpected results.

3.38 Control Status Register for Peripheral Receive Endpoint (PERI_RXCSR)

The control status register for peripheral receive endpoint (PERI_RXCSR) is shown in Figure 75 and described in Table 76.

Figure 75. Control Status Register for Peripheral Receive Endpoint (PERI_RXCSR)

15	14	13	12	11	10		8
AUTOCLEAR	ISO	DMAEN	DISNYET	DMAMODE	Reserved		
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0		

7	6	5	4	3	2	1	0
CLRDATATOG	SENTSTALL	SENDSTALL	FLUSHFIFO	DATAERROR	OVERRUN	FIFOFULL	RXPKTRDY
W-0	R/W-0	R/W-0	W-0	R-0	R/W-0	R-0	R/W-0

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

Table 76. Control Status Register for Peripheral Receive Endpoint (PERI_RXCSR) Field Descriptions

Bit	Field	Value	Description
15	AUTOCLEAR	0	DMA Mode: The CPU sets the AUTOCLEAR bit prior to enabling the Rx DMA.
		1	CPU Mode: If the CPU sets the AUTOCLEAR bit, then the RXPKTRDY bit will be automatically cleared when a packet of RXMAXP bytes has been unloaded from the Receive FIFO. When packets of less than the maximum packet size are unloaded, RXPKTRDY will have to be cleared manually.
14	ISO	0-1	Set this bit to enable the Receive endpoint for Isochronous transfers, and clear it to enable the Receive endpoint for Bulk/Interrupt transfers.
13	DMAEN	0-1	Set this bit to enable the DMA request for the Receive endpoints.
12	DISNYET	0	DISNYET: Applies only for Bulk/Interrupt Transactions: The CPU sets this bit to disable the sending of NYET handshakes. When set, all successfully received Rx packets are ACK'd including at the point at which the FIFO becomes full.
			Note: This bit only has any effect in high-speed mode, in which mode it should be set for all Interrupt endpoints.
		1	PID_ERROR: Applies only for ISO Transactions: The core sets this bit to indicate a PID error in the received packet.
11	DMAMODE	0-1	The CPU clears the DMAMODE bit prior to enabling the Rx DMA.
10-8	Reserved	0	Reserved.
7	CLRDATATOG	0-1	Write a 1 to this bit to reset the endpoint data toggle to 0.
6	SENTSTALL	0-1	This bit is set when a STALL handshake is transmitted. The FIFO is flushed and the TXPKTRDY bit is cleared. You should clear this bit.
5	SENDSTALL	0-1	Write a 1 to this bit to issue a STALL handshake. Clear this bit to terminate the stall condition.
			Note: This bit has no effect where the endpoint is being used for Isochronous transfers.
4	FLUSHFIFO	0-1	Write a 1 to this bit to flush the next packet to be read from the endpoint Receive FIFO. The FIFO pointer is reset and the RXPKTRDY bit is cleared.
			Note: FLUSHFIFO has no effect unless RXPKTRDY is set. Also note that, if the FIFO is double-buffered, FLUSHFIFO may need to be set twice to completely clear the FIFO.
3	DATAERROR	0-1	This bit is set when RXPKTRDY is set if the data packet has a CRC or bit-stuff error. It is cleared when RXPKTRDY is cleared.
			Note: This bit is only valid when the endpoint is operating in ISO mode. In Bulk mode, it always returns zero.
2	OVERRUN	0-1	This bit is set if an OUT packet cannot be loaded into the Receive FIFO. You should clear this bit.
			Note: This bit is only valid when the endpoint is operating in ISO mode. In Bulk mode, it always returns zero.
1	FIFOFULL	0-1	This bit is set when no more packets can be loaded into the Receive FIFO.
0	RXPKTRDY	0-1	This bit is set when a data packet has been received. You should clear this bit when the packet has been unloaded from the Receive FIFO. An interrupt is generated when the bit is set.

3.39 Count 0 Register (COUNT0)

The count 0 register (COUNT0) is shown in Figure 76 and described in Table 77.

Figure 76. Count 0 Register (COUNT0)

15 … 7	6 … 0
Reserved	EP0RXCOUNT
R-0	R-0

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

Table 77. Count 0 Register (COUNT0) Field Descriptions

Bit	Field	Value	Description
15-7	Reserved	0	Reserved.
6-0	EP0RXCOUNT	0-7Fh	Indicates the number of received data bytes in the Endpoint 0 FIFO. The value returned changes as the contents of the FIFO change and is only valid while RXPKTRDY of PERI_CSR0 is set.

3.40 Receive Count Register (RXCOUNT)

The receive count register (RXCOUNT) is shown in Figure 77 and described in Table 78.

Figure 77. Receive Count Register (RXCOUNT)

15 … 13	12 … 0
Reserved	EPRXCOUNT
R-0	R-0

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

Table 78. Receive Count Register (RXCOUNT) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12-0	EPRXCOUNT	0-1FFFh	Holds the number of received data bytes in the packet in the Receive FIFO. The value returned changes as the contents of the FIFO change and is only valid while RXPKTRDY of PERI_RXCSR or HOST_RXCSR is set.

3.41 Configuration Data Register (CONFIGDATA)

The configuration data register (CONFIGDATA) is shown in Figure 78 and described in Table 79.

Figure 78. Configuration Data Register (CONFIGDATA)

7	6	5	4	3	2	1	0
MPRXE	MPTXE	BIGENDIAN	HBRXE	HBTXE	DYNFIFO	SOFTCONE	UTMIDATAWIDTH
R-0	R-0	R-0	R-0	R-0	R-1	R-1	R-0

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

Table 79. Configuration Data Register (CONFIGDATA) Field Descriptions

Bit	Field	Value	Description
7	MPRXE		Indicates automatic amalgamation of bulk packets.
		0	Automatic amalgamation of bulk packets is not selected.
		1	Automatic amalgamation of bulk packets is selected.
6	MPTXE		Indicates automatic splitting of bulk packets.
		0	Automatic splitting of bulk packets is not selected.
		1	Automatic splitting of bulk packets is selected.
5	BIGENDIAN		Indicates endian ordering.
		0	Little-endian ordering is selected.
		1	Big-endian ordering is selected.
4	HBRXE		Indicates high-bandwidth Rx ISO endpoint support.
		0	High-bandwidth Rx ISO endpoint support is not selected.
		1	High-bandwidth Rx ISO endpoint support is selected.
3	HBTXE		Indicates high-bandwidth Tx ISO endpoint support.
		0	High-bandwidth Tx ISO endpoint support is not selected.
		1	High-bandwidth Tx ISO endpoint support is selected.
2	DYNFIFO		Indicates dynamic FIFO sizing.
		0	Dynamic FIFO sizing option is not selected.
		1	Dynamic FIFO sizing option is selected.
1	SOFTCONE		Indicates soft connect/disconnect.
		0	Soft connect/disconnect option is not selected.
		1	Soft connect/disconnect option is selected.
0	UTMIDATAWIDTH		Indicates selected UTMI data width.
		0	8 bits.
		1	16 bits.

3.42 Transmit and Receive FIFO Registers for Endpoint 0 (FIFO0R1 and FIFO0R2)

The transmit and receive FIFO register 1 for endpoint 0 (FIFO0R1) is shown in Figure 79 and described in Table 80. The transmit and receive FIFO register 2 for endpoint 0 (FIFO0R2) is shown in Figure 80 and described in Table 81.

Figure 79. Transmit and Receive FIFO Register 1 for Endpoint 0 (FIFO0R1)

15 … 0
DATA
R/W-0

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

Figure 80. Transmit and Receive FIFO Register 2 for Endpoint 0 (FIFO0R2)

15 … 0
DATA
R/W-0

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

Table 80. Transmit and Receive FIFO Register 1 for Endpoint 0 (FIFO0R1) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to this address loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

Table 81. Transmit and Receive FIFO Register 2 for Endpoint 0 (FIFO0R2) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to this address loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

3.43 *Transmit and Receive FIFO Registers for Endpoint 1 (FIFO1R1 and FIFO1R2)*

The transmit and receive FIFO register 1 for endpoint 1 (FIFO1R1) is shown in Figure 81 and described in Table 82. The transmit and receive FIFO register 2 for endpoint 1 (FIFO1R2) is shown in Figure 82 and described in Table 83.

Figure 81. Transmit and Receive FIFO Register 1 for Endpoint 1 (FIFO1R1)

15 – 0
DATA
R/W-0

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

Figure 82. Transmit and Receive FIFO Register 2 for Endpoint 1 (FIFO1R2)

15 – 0
DATA
R/W-0

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

Table 82. Transmit and Receive FIFO Register 1 for Endpoint 1 (FIFO1R1) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

Table 83. Transmit and Receive FIFO Register 2 for Endpoint 1 (FIFO1R2) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

3.44 Transmit and Receive FIFO Registers for Endpoint 2 (FIFO2R1 and FIFO2R2)

The transmit and receive FIFO register 1 for endpoint 2 (FIFO2R1) is shown in Figure 83 and described in Table 84. The transmit and receive FIFO register 2 for endpoint 2 (FIFO2R2) is shown in Figure 84 and described in Table 85.

Figure 83. Transmit and Receive FIFO Register 1 for Endpoint 2 (FIFO2R1)

15 — 0
DATA
R/W-0

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

Figure 84. Transmit and Receive FIFO Register 2 for Endpoint 2 (FIFO2R2)

15 — 0
DATA
R/W-0

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

Table 84. Transmit and Receive FIFO Register 1 for Endpoint 2 (FIFO2R1) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

Table 85. Transmit and Receive FIFO Register 2 for Endpoint 2 (FIFO2R2) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

3.45 Transmit and Receive FIFO Registers for Endpoint 3 (FIFO3R1 and FIFO3R2)

The transmit and receive FIFO register 1 for endpoint 3 (FIFO3R1) is shown in Figure 85 and described in Table 86. The transmit and receive FIFO register 2 for endpoint 3 (FIFO3R2) is shown in Figure 86 and described in Table 87.

Figure 85. Transmit and Receive FIFO Register 1 for Endpoint 3 (FIFO3R1)

15 … 0
DATA
R/W-0

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

Figure 86. Transmit and Receive FIFO Register 2 for Endpoint 3 (FIFO3R2)

15 … 0
DATA
R/W-0

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

Table 86. Transmit and Receive FIFO Register 1 for Endpoint 3 (FIFO3R1) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

Table 87. Transmit and Receive FIFO Register 2 for Endpoint 3 (FIFO3R2) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

3.46 Transmit and Receive FIFO Registers for Endpoint 4 (FIFO4R1 and FIFO4R2)

The transmit and receive FIFO register 1 for endpoint 4 (FIFO4R1) is shown in Figure 87 and described in Table 88. The transmit and receive FIFO register 2 for endpoint 4 (FIFO4R2) is shown in Figure 88 and described in Table 89.

Figure 87. Transmit and Receive FIFO Register 1 for Endpoint 4 (FIFO4R1)

15 … 0
DATA
R/W-0

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

Figure 88. Transmit and Receive FIFO Register 2 for Endpoint 4 (FIFO4R2)

15 … 0
DATA
R/W-0

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

Table 88. Transmit and Receive FIFO Register 1 for Endpoint 4 (FIFO4R1) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

Table 89. Transmit and Receive FIFO Register 2 for Endpoint 4 (FIFO4R2) Field Descriptions

Bit	Field	Value	Description
15-0	DATA	0-FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

3.47 Device Control Register (DEVCTL)

The device control register (DEVCTL) is shown in Figure 89 and described in Table 90.

Figure 89. Device Control Register (DEVCTL)

7	6	5	4	3	2	1	0
BDEVICE	FSDEV	LSDEV	VBUS		HOSTMODE	Reserved	SESSION
R-0	R-0	R-0	R-0		R-0	R/W-0	R/W-0

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

Table 90. Device Control Register (DEVCTL) Field Descriptions

Bit	Field	Value	Description
7	BDEVICE		This read-only bit indicates whether the USB controller is operating as the 'A' device or the 'B' device.
		0	A device.
		1	B device.
			Only valid while a session is in progress.
6	FSDEV	0-1	This read-only bit is set when a full-speed or high-speed device has been detected being connected to the port (high-speed devices are distinguished from full-speed by checking for high-speed chirps when the device is reset). Only valid in Host mode. Host mode is not supported on the C5505.
5	LSDEV	0-1	This read-only bit is set when a low-speed device has been detected being connected to the port. Only valid in Host mode. Host mode is not supported on the C5505.
	VBUS	0-3h	These read-only bits encode the current VBus level as follows:
		0	Below Session End.
		1 h	Above Session End, below AValid.
		2h	Above AValid, below VBusValid.
		3h	Above VBusValid.
2	HOSTMODE	0-1	This read-only bit is set when the USB controller is acting as a Host. Host mode is not supported on the C5505.
1	Reserved	0	Reserved.
0	SESSION	0-1	When operating as an 'A' device, you must set or clear this bit start or end a session. When operating as a 'B' device, this bit is set/cleared by the USB controller when a session starts/ends. You must also set this bit to initiate the Session Request Protocol. When the USB controller is in Suspend mode, you may clear the bit to perform a software disconnect. A special software routine is required to perform SRP. Details will be made available in a later document version.

3.48 Transmit Endpoint FIFO Size (TXFIFOSZ)

Section 2.7 describes dynamically setting endpoint FIFO sizes. The option of dynamically setting endpoint FIFO sizes only applies to Endpoints 1-4. The Endpoint 0 FIFO has a fixed size (64 bytes) and a fixed location (start address 0). It is the responsibility of the firmware to ensure that all the Tx and Rx endpoints that are active in the current USB configuration have a block of RAM assigned exclusively to that endpoint. The RAM must be at least as large as the maximum packet size set for that endpoint.

The transmit endpoint FIFO size (TXFIFOSZ) is shown in Figure 90 and described in Table 91.

Figure 90. Transmit Endpoint FIFO Size (TXFIFOSZ)

7-5	4	3-0
Reserved	DPB	SZ
R-0	R/W-0	R-0

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

Table 91. Transmit Endpoint FIFO Size (TXFIFOSZ) Field Descriptions

Bit	Field	Value	Description
7-5	Reserved	0	Reserved.
4	DPB		Double packet buffering enable.
		0	Single packet buffering is supported.
		1	Double packet buffering is enabled.
3-0	SZ	0-Fh	Maximum packet size to be allowed (before any splitting within the FIFO of Bulk packets prior to transmission). If m = SZ, the FIFO size is calculated as $2^{(m+3)}$ for single packet buffering and $2^{(m+4)}$ for dual packet buffering.

3.49 Receive Endpoint FIFO Size (RXFIFOSZ)

Section 2.7 describes dynamically setting endpoint FIFO sizes. The option of dynamically setting endpoint FIFO sizes only applies to Endpoints 1-4. The Endpoint 0 FIFO has a fixed size (64 bytes) and a fixed location (start address 0). It is the responsibility of the firmware to ensure that all the Tx and Rx endpoints that are active in the current USB configuration have a block of RAM assigned exclusively to that endpoint. The RAM must be at least as large as the maximum packet size set for that endpoint.

The receive endpoint FIFO size (RXFIFOSZ) is shown in Figure 91 and described in Table 92.

Figure 91. Receive Endpoint FIFO Size (RXFIFOSZ)

7-5	4	3-0
Reserved	DPB	SZ
R-0	R/W-0	R-0

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

Table 92. Receive Endpoint FIFO Size (RXFIFOSZ) Field Descriptions

Bit	Field	Value	Description
7-5	Reserved	0	Reserved.
4	DPB		Double packet buffering enable.
		0	Single packet buffering is supported.
		1	Double packet buffering is enabled.
3-0	SZ	0-Fh	Maximum packet size to be allowed (before any splitting within the FIFO of Bulk packets prior to transmission). If m = SZ, the FIFO size is calculated as $2^{(m+3)}$ for single packet buffering and $2^{(m+4)}$ for dual packet buffering.

3.50 Transmit Endpoint FIFO Address (TXFIFOADDR)

The transmit endpoint FIFO address (TXFIFOADDR) is shown in Figure 92 and described in Table 93.

Figure 92. Transmit Endpoint FIFO Address (TXFIFOADDR)

15 – 13	12 – 0
Reserved	ADDR
R-0	R/W-0

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

Table 93. Transmit Endpoint FIFO Address (TXFIFOADDR) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12-0	ADDR	0-1FFFh	Start Address of endpoint FIFO in units of 8 bytes. If m = ADDR, then the start address is 8 × m.

3.51 Hardware Version Register (HWVERS)

The hardware version register (HWVERS) contains the RTL major and minor version numbers for the USB 2.0 controller module. The RTL version number is REVMAJ.REVMIN. The HWVERS is shown in Figure 93 and described in Table 94.

Figure 93. Hardware Version Register (HWVERS)

15	14 – 10	9 – 0
RC	REVMAJ	REVMIN
R-0	R-0	R-0

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

Table 94. Hardware Version Register (HWVERS) Field Descriptions

Bit	Field	Value	Description
15	RC	0-1	Set to 1 if RTL is used from a Release Candidate, rather than from a full release of the core.
14-10	REVMAJ	0-1Fh	Major version of RTL. Range is 0-3.1.
9-0	REVMIN	0-3E7h	Minor version of RTL. Range is 0-999.

3.52 Receive Endpoint FIFO Address (RXFIFOADDR)

The receive endpoint FIFO address (RXFIFOADDR) is shown in Figure 94 and described in Table 95.

Figure 94. Receive Endpoint FIFO Address (RXFIFOADDR)

15 - 13	12 - 0
Reserved	ADDR
R-0	R/W-0

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

Table 95. Receive Endpoint FIFO Address (RXFIFOADDR) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved.
12-0	ADDR	0-1FFFh	Start Address of endpoint FIFO in units of 8 bytes. If m = ADDR, then the start address is 8 × m.

3.53 CDMA Revision Identification Registers (DMAREVID1 and DMAREVID2)

The CDMA revision identification registers (DMAREVID1 and DMAREVID2) contain the revision for the module. The DMAREVID1 is shown in Figure 95 and described in Table 96. The DMAREVID2 is shown in Figure 96 and described in Table 97.

Figure 95. CDMA Revision Identification Register 1 (DMAREVID1)

15 - 0
REV_LSB
R-1900h

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

Figure 96. CDMA Revision Identification Register 2 (DMAREVID2)

15 - 0
REV_MSB
R-0053h

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

Table 96. CDMA Revision Identification Register 1 (DMAREVID1) Field Descriptions

Bit	Field	Value	Description
15-0	REV_LSB	0-FFFFh	Revision ID of the CPPI DMA (CDMA) module. Least significant bits.

Table 97. CDMA Revision Identification Register 2 (DMAREVID2) Field Descriptions

Bit	Field	Value	Description
15-0	REV_MSB	0-FFFFh	Revision ID of the CPPI DMA (CDMA) module. Most significant bits.

3.54 CDMA Teardown Free Descriptor Queue Control Register (TDFDQ)

The CDMA teardown free descriptor queue control register (TDFDQ) is used to inform the DMA of the location in memory or descriptor array which is to be used for signaling of a teardown complete for each transmit and receive channel. The CDMA teardown free descriptor queue control register (TDFDQ) is shown in Figure 97 and described in Table 98.

Figure 97. CDMA Teardown Free Descriptor Queue Control Register (TDFDQ)

15 14	13 12	11 0
Reserved	TD_DESC_QMGR	TD_DESC_QNUM
R-0	R/W-0	R/W-0

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

Table 98. CDMA Teardown Free Descriptor Queue Control Register (TDFDQ) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-12	TD_DESC_QMGR	0-3h	Controls which of the four queue managers the DMA accesses to allocate a channel teardown descriptor from the teardown descriptor queue.
11-0	TD_DESC_QNUM	0-FFFh	Controls which of the 2K queues in the indicated queue manager should be read to allocate the channel teardown descriptors.

3.55 CDMA Emulation Control Register (DMAEMU)

The CDMA emulation controls the behavior of the DMA when an emulation suspend signal is asserted. The CDMA emulation control register (DMAEMU) is shown in Figure 98 and described in Table 99.

Figure 98. CDMA Emulation Control Register (DMAEMU)

15 2	1	0
Reserved	SOFT	FREE
R-0	R/W-0	R/W-0

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

Table 99. CDMA Emulation Control Register (DMAEMU) Field Descriptions

Bit	Field	Value	Description
15-2	Reserved	0	Reserved.
1	SOFT	0-1	
0	FREE	0-1	

3.56 CDMA Transmit Channel n Global Configuration Registers (TXGCR1[*n*] and TXGCR2[*n*])

The transmit channel *n* configuration registers (TXGCR2[*n*] and TXGCR1[*n*]) initialize the behavior of each of the transmit DMA channels. There are four configuration register pairs, one for each transmit DMA channel.

The transmit channel *n* configuration registers TXGCR1[*n*]) and (TXGCR2[*n*] are shown in Figure 99 and Figure 100and described in Table 100 and Table 101. .

Figure 99. CDMA Transmit Channel *n* Global Configuration Register 1 (TXGCR1[*n*])

15 14	13 12	11 0
Reserved	TX_DEFAULT_QMGR	TX_DEFAULT_QNUM
R-0	W-0	W-0

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

Figure 100. CDMA Transmit Channel *n* Global Configuration Register 2 (TXGCR2[*n*])

15	14	13 0
TX_ENABLE	TX_TEARDOWN	Reserved
R/W-0	R/W-0	R-0

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

Table 100. CDMA Transmit Channel *n* Global Configuration Register 1 (TXGCR1[*n*]) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-12	TX_DEFAULT_QMGR	0-3h	Controls the default queue manager number that is used to queue teardown descriptors back to the host.
11-0	TX_DEFAULT_QNUM	0-FFFh	Controls the default queue number within the selected queue manager onto which teardown descriptors are queued back to the host.

Table 101. CDMA Transmit Channel *n* Global Configuration Register 2 (TXGCR2[*n*]) Field Descriptions

Bit	Field	Value	Description
15	TX_ENABLE		Channel control. The TX_ENABLE field is cleared after a channel teardown is complete.
		0	Disables channel.
		1	Enables channel.
14	TX_TEARDOWN	0-1	Setting this bit requests the channel to be torn down. The TX_TEARDOWN field remains set after a channel teardown is complete.
13-0	Reserved	0	Reserved.

3.57 CDMA Receive Channel n Global Configuration Registers (RXGCR1[n] and RXGCR2[n])

The receive channel *n* global configuration registers (RXGCR1[*n*] and RXGCR2[*n*]) initialize the global (non-descriptor-type specific) behavior of each of the receive DMA channels. There are four configuration register pairs, one for each receive DMA channel. If the enable bit is being set, the receive channel *n* global configuration register should only be written after all of the other receive configuration registers have been initialized.

The receive channel *n* global configuration registers (RXGCR1[*n*] and RXGCR2[*n*]) are shown in Figure 101 and Figure 102 and are described in Table 102 and Table 103.

Figure 101. CDMA Receive Channel *n* Global Configuration Register 1 (RXGCR1[*n*])

15-14	13-12	11-0
RX_DEFAULT_DESC_TYPE	RX_DEFAULT_RQ_QMGR	RX_DEFAULT_RQ_QNUM
R-0	W-0	W-0

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

Figure 102. CDMA Receive Channel *n* Global Configuration Register 2 (RXGCR2[*n*])

15	14	13-9	8	7-0
RX_ENABLE	RX_TEARDOWN	Reserved	RX_ERROR_HANDLING	RX_SOP_OFFSET
R/W-0	R/W-0	R-0	W-0	W-0

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

Table 102. CDMA Receive Channel *n* Global Configuration Register 1 (RXGCR1[*n*]) Field Descriptions

Bit	Field	Value	Description
15-14	RX_DEFAULT_DESC_TYPE	0-3h	Indicates the default descriptor type to use. The actual descriptor type that is used for reception can be overridden by information provided in the CPPI FIFO data block.
		0	Reserved.
		1h	Host.
		2h-3h	Reserved.
13-12	RX_DEFAULT_RQ_QMGR	0-3h	Indicates the default receive queue manager that this channel should use. The actual receive queue manager index can be overridden by information provided in the CPPI FIFO data block.
11-0	RX_DEFAULT_RQ_QNUM	0-FFFh	Indicates the default receive queue that this channel should use. The actual receive queue that is used for reception can be overridden by information provided in the CPPI FIFO data block.

Table 103. CDMA Receive Channel *n* Global Configuration Register 2 (RXGCR2[*n*]) Field Descriptions

Bit	Field	Value	Description
15	RX_ENABLE		Channel control. Field is cleared after a channel teardown is complete.
		0	Disables channel.
		1	Enables channel.
14	RX_TEARDOWN	0-1	Indicates whether a receive operation is complete. Field should be cleared when a channel is initialized. Field is set after a channel teardown is complete.
13-9	Reserved	0	Reserved.

Table 103. CDMA Receive Channel *n* Global Configuration Register 2 (RXGCR2[*n*]) Field Descriptions (continued)

Bit	Field	Value	Description
8	RX_ERROR_HANDLING		Controls the error handling mode for the channel and is only used when channel errors (i.e. descriptor or buffer starvation occur):
		0	Starvation errors result in dropping packet and reclaiming any used descriptor or buffer resources back to the original queues/pools they were allocated to.
		1	Starvation errors result in subsequent retry of the descriptor allocation operation. In this mode, the DMA will return to the IDLE state without saving its internal operational state back to the internal state RAM and without issuing an advance operation on the FIFO interface. This results in the DMA re-initiating the FIFO block transfer at a later time with the intention that additional free buffers and/or descriptors will have been added.
7-0	RX_SOP_OFFSET	0–FFh	Specifies the number of bytes that are to be skipped in the SOP buffer before beginning to write the payload. This value must be less than the minimum size of a buffer in the system.

3.58 CDMA Receive Channel n Host Packet Configuration Registers A (RXHPCR1A[n] and RXHPCR2A[n])

The receive channel *n* host packet configuration registers A (RXHPCR1A[n] and RXHPCR2A[n]) initialize the behavior of each of the receive DMA channels for reception of host type packets. There are four configuration A registers, one for each receive DMA channel.

The receive channel *n* host packet configuration register 1 A (RXHPCR1A[n]) are shown in Figure 103 and described in Table 104. The receive channel *n* host packet configuration register 2 A (RXHPCR2A[n]) is shown in Figure 104 and described in Table 105.

Figure 103. Receive Channel *n* Host Packet Configuration Register 1 A (RXHPCR1A[*n*])

15 14	13 12	11 0
Reserved	RX_HOST_FDQ0_QMGR	RX_HOST_FDQ0_QNUM
R-0	W-0	W-0

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

Figure 104. Receive Channel *n* Host Packet Configuration Register 2 A (RXHPCR2A[*n*])

15 14	13 12	11 0
Reserved	RX_HOST_FDQ1_QMGR	RX_HOST_FDQ1_QNUM
R-0	W-0	W-0

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

Table 104. Receive Channel *n* Host Packet Configuration Register 1 A (RXHPCR1A[*n*]) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-12	RX_HOST_FDQ0_QMGR	0-3h	Specifies which buffer manager should be used for the first receive buffer in a host type packet.
11-0	RX_HOST_FDQ0_QNUM	0-FFFh	Specifies which free descriptor/buffer pool should be used for the first receive buffer in a host type packet.

Table 105. Receive Channel *n* Host Packet Configuration Register 2 A (RXHPCR2A[*n*]) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-12	RX_HOST_FDQ1_QMGR	0-3h	Specifies which buffer manager should be used for the second receive buffer in a host type packet.
11-0	RX_HOST_FDQ1_QNUM	0-FFFh	Specifies which free descriptor/buffer pool should be used for the second receive buffer in a host type packet.

3.59 CDMA Receive Channel n Host Packet Configuration Registers B (RXHPCR1B[n] and RXHPCR2B[n])

The receive channel *n* host packet configuration registers B (RXHPCR1B[n] and RXHPCR2B[n]) initialize the behavior of each of the receive DMA channels for reception of host type packets. There are four configuration B register pairs, one for each receive DMA channel.

The receive channel *n* host packet configuration register 1 B (RXHPCR1B[*n*]) is shown in Figure 105 and described in Table 106. The receive channel *n* host packet configuration register 2 B (RXHPCR2B[*n*]) is shown in Figure 106 and described in Table 107.

Figure 105. Receive Channel *n* Host Packet Configuration Register 1 B (RXHPCR1B[*n*])

15 14	13 12	11 0
Reserved	RX_HOST_FDQ2_QMGR	RX_HOST_FDQ2_QNUM
R-0	W-0	W-0

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

Figure 106. Receive Channel *n* Host Packet Configuration Register 2 B (RXHPCR2B[*n*])

15 14	13 12	11 0
Reserved	RX_HOST_FDQ3_QMGR	RX_HOST_FDQ3_QNUM
R-0	W-0	W-0

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

Table 106. Receive Channel *n* Host Packet Configuration Register 1 B (RXHPCR1B[*n*]) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-12	RX_HOST_FDQ2_QMGR	0-3h	Specifies which buffer manager should be used for the third receive buffer in a host type packet.
11-0	RX_HOST_FDQ2_QNUM	0-FFFh	Specifies which free descriptor/buffer pool should be used for the third receive buffer in a host type packet.

Table 107. Receive Channel *n* Host Packet Configuration Register 2 B (RXHPCR2B[*n*]) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-12	RX_HOST_FDQ3_QMGR	0-3h	Specifies which buffer manager should be used for the fourth or later receive buffer in a host type packet.
11-0	RX_HOST_FDQ3_QNUM	0-FFFh	Specifies which free descriptor/buffer pool should be used for the fourth or later receive buffer in a host type packet.

3.60 CDMA Scheduler Control Register (DMA_SCHED_CTRL1 and DMA_SCHED_CTRL2)

The CDMA scheduler control registers (DMA_SCHED_CTRL1 and DMA_SCHED_CTRL2) enable the scheduler and indicate the last entry in the scheduler table. The CDMA scheduler control register 1 (DMA_SCHED_CTRL1) is shown in Figure 107 and described in Table 108. The CDMA scheduler control register 2 (DMA_SCHED_CTRL2) is shown in Figure 108 and described in Table 109.

Figure 107. CDMA Scheduler Control Register 1 (DMA_SCHED_CTRL1)

15 8	7 0
Reserved	LAST_ENTRY
R-0	R/W-0

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

Figure 108. CDMA Scheduler Control Register 2 (DMA_SCHED_CTRL2)

15	14 0
ENABLE	Reserved
R/W-0	R-0

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

Table 108. CDMA Scheduler Control Register 1 (DMA_SCHED_CTRL1) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved	0	Reserved.
7-0	LAST_ENTRY	0-FFh	Indicates the last valid entry in the scheduler table. There are 64 words in the table and there are 4 entries in each word. The table can be programmed with any integer number of entries from 1 to 256. The corresponding encoding for this field is as follows:
		0	1 entry.
		1h	2 entries.
		2h-FFh	3 entries to 256 entries.

Table 109. CDMA Scheduler Control Register 2 (DMA_SCHED_CTRL2) Field Descriptions

Bit	Field	Value	Description
15	ENABLE		This is the enable bit for the scheduler and is encoded as follows:
		0	Scheduler is disabled and will no longer fetch entries from the scheduler table or pass credits to the DMA controller.
		1	Scheduler is enabled. This bit should only be set after the table has been initialized.
14-0	Reserved	0	Reserved.

3.61 CDMA Scheduler Table Word n Registers (ENTRYLSW[n]-ENTRYMSW[n])

The CDMA scheduler table word *n* registers (ENTRYLSW[n]-ENTRYMSW[n]) provide information about the scheduler. The CDMA scheduler table word *n* registers (ENTRYLSW[*n*]) are shown in Figure 109 and described in Table 110. The CDMA scheduler table word *n* registers (ENTRYMSW[*n*]) are shown in Figure 110 and described in Table 111.

Figure 109. CDMA Scheduler Table Word *n* Registers (ENTRYLSW[*n*])

15	14 - 12	11 - 8	7	6 - 4	3 - 0
ENTRY1_RXTX	Reserved	ENTRY1_CHANNEL	ENTRY0_RXTX	Reserved	ENTRY0_CHANNEL
W-0	R-0	W-0	W-0	R-0	W-0

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

Figure 110. CDMA Scheduler Table Word *n* Registers (ENTRYMSW[*n*])

15	14 - 12	11 - 8	7	6 - 4	3 - 0
ENTRY3_RXTX	Reserved	ENTRY3_CHANNEL	ENTRY2_RXTX	Reserved	ENTRY2_CHANNEL
W-0	R-0	W-0	W-0	R-0	W-0

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

Table 110. CDMA Scheduler Table Word *n* Registers (ENTRYLSW[*n*]) Field Descriptions

Bit	Field	Value	Description
15	ENTRY1_RXTX		This entry is for a transmit or a receive channel.
		0	Transmit channel
		1	Receive channel
14-12	Reserved	0	Reserved
11-8	ENTRY1_CHANNEL	0-Fh	Indicates the channel number that is to be given an opportunity to transfer data. If this is a transmit entry, the DMA will be presented with a scheduling credit for that exact transmit channel. If this is a receive entry, the DMA will be presented with a scheduling credit for the receive FIFO that is associated with this channel. For receive FIFOs which carry traffic for more than one receive DMA channel, the exact channel number that is given in the receive credit will actually be the channel number which is currently on the head element of that Rx FIFO, which is not necessarily the channel number given in the scheduler table entry.
7	ENTRY0_RXTX		This entry is for a transmit or a receive channel.
		0	Transmit channel
		1	Receive channel
6-4	Reserved	0	Reserved
3-0	ENTRY0_CHANNEL	0-Fh	Indicates the channel number that is to be given an opportunity to transfer data. If this is a transmit entry, the DMA will be presented with a scheduling credit for that exact transmit channel. If this is a receive entry, the DMA will be presented with a scheduling credit for the receive FIFO that is associated with this channel. For receive FIFOs which carry traffic for more than one receive DMA channel, the exact channel number that is given in the receive credit will actually be the channel number which is currently on the head element of that Rx FIFO, which is not necessarily the channel number given in the scheduler table entry.

Table 111. CDMA Scheduler Table Word *n* Registers (ENTRYMSW[*n*]) Field Descriptions

Bit	Field	Value	Description
15	ENTRY3_RXTX		This entry is for a transmit or a receive channel.
		0	Transmit channel
		1	Receive channel
14-12	Reserved	0	Reserved

Table 111. CDMA Scheduler Table Word *n* Registers (ENTRYMSW[*n*]) Field Descriptions (continued)

Bit	Field	Value	Description
11-8	ENTRY3_CHANNEL	0-Fh	Indicates the channel number that is to be given an opportunity to transfer data. If this is a transmit entry, the DMA will be presented with a scheduling credit for that exact transmit channel. If this is a receive entry, the DMA will be presented with a scheduling credit for the receive FIFO that is associated with this channel. For receive FIFOs which carry traffic for more than one receive DMA channel, the exact channel number that is given in the receive credit will actually be the channel number which is currently on the head element of that Rx FIFO, which is not necessarily the channel number given in the scheduler table entry.
7	ENTRY2_RXTX		This entry is for a transmit or a receive channel.
		0	Transmit channel
		1	Receive channel
6-4	Reserved	0	Reserved
3-0	ENTRY2_CHANNEL	0-Fh	Indicates the channel number that is to be given an opportunity to transfer data. If this is a transmit entry, the DMA will be presented with a scheduling credit for that exact transmit channel. If this is a receive entry, the DMA will be presented with a scheduling credit for the receive FIFO that is associated with this channel. For receive FIFOs which carry traffic for more than one receive DMA channel, the exact channel number that is given in the receive credit will actually be the channel number which is currently on the head element of that Rx FIFO, which is not necessarily the channel number given in the scheduler table entry.

3.62 Queue Manager Revision Identification Registers (QMGRREVID1 and QMGRREVID2)

The queue manager revision identification registers (QMGRREVID1 and QMGRREVID2) contain the major and minor revisions for the module. The QMGRREVID1 is shown in Figure 111 and described in Table 112. The QMGRREVID2 is shown in Figure 112 and described in Table 113.

Figure 111. Queue Manager Revision Identification Register 1 (QMGRREVID1)

15	0
REV_LSB	
R-1200h	

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

Figure 112. Queue Manager Revision Identification Register 2 (QMGRREVID2)

15	0
REV_MSB	
R-0052h	

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

Table 112. Queue Manager Revision Identification Register 1 (QMGRREVID1) Field Descriptions

Bit	Field	Value	Description
15-0	REV_LSB	0-FFFFh	Revision ID of the queue manager. Least-significant bits.

Table 113. Queue Manager Revision Identification Register 2 (QMGRREVID2) Field Descriptions

Bit	Field	Value	Description
15-0	REV_MSB	0-FFFFh	Revision ID of the queue manager. Most-significant bits.

3.63 Queue Manager Queue Diversion Registers (DIVERSION1 and DIVERSION2)

The queue manager queue diversion registers (DIVERSION1 and DIVERSION2) are used to transfer the contents of one queue onto another queue. It does not support byte accesses. The queue manager queue diversion register 1 (DIVERSION1) is shown in Figure 113 and described in Table 114. The queue manager queue diversion register 2 (DIVERSION2) is shown in Figure 114 and described in Table 115.

Figure 113. Queue Manager Queue Diversion Register 1 (DIVERSION1)

15–14	13–0
Reserved	SOURCE_QNUM
R-0	W-0

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

Figure 114. Queue Manager Queue Diversion Register 2 (DIVERSION2)

15	14	13–0
HEAD_TAIL	Reserved	DEST_QNUM
W-0	R-0	W-0

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

Table 114. Queue Manager Queue Diversion Register 1 (DIVERSION1) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-0	SOURCE_QNUM	0-3FFFh	Source Queue Number.

Table 115. Queue Manager Queue Diversion Register 2 (DIVERSION2 Field Descriptions

Bit	Field	Value	Description
15	HEAD_TAIL		Indicates whether queue contents should be merged on to the head or tail of the destination queue.
		0	Head.
		1	Tail.
14	Reserved	0	Reserved.
13-0	DEST_QNUM	0-3FFFh	Destination Queue Number.

3.64 Queue Manager Free Descriptor/Buffer Starvation Count Register 0 (FDBSC0)

The free descriptor/buffer queue starvation count register (FDBSC0) provides statistics about how many starvation events are occurring on the receive free descriptor/buffer queues. These registers do not support byte accesses. The free descriptor/buffer queue starvation count register (FDBSC0) is shown in Figure 115 and described in Table 116.

Figure 115. Queue Manager Free Descriptor/Buffer Starvation Count Register 0 (FDBSC0)

15 — 8	7 — 0
FDBQ1_STARVE_CNT	FDBQ0_STARVE_CNT
RC-0	RC-0

LEGEND: RC = Cleared on read; -*n* = value after reset

Table 116. Queue Manager Free Descriptor/Buffer Starvation Count Register 0 (FDBSC0) Field Descriptions

Bit	Field	Value	Description
15-8	FDBQ1_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 1 is read while it is empty. This field is cleared when read by CPU.
7-0	FDBQ0_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 0 is read while it is empty. This field is cleared when read by CPU.

3.65 Queue Manager Free Descriptor/Buffer Starvation Count Register 1 (FDBSC1)

The free descriptor/buffer queue starvation count register (FDBSC1) provides statistics about how many starvation events are occurring on the receive free descriptor/buffer queues. These registers do not support byte accesses. The free descriptor/buffer queue starvation count register (FDBSC1) is shown in Figure 116 and described in Table 117.

Figure 116. Queue Manager Free Descriptor/Buffer Starvation Count Register 1 (FDBSC1)

15 — 8	7 — 0
FDBQ3_STARVE_CNT	FDBQ2_STARVE_CNT
RC-0	RC-0

LEGEND: RC = Cleared on read; -*n* = value after reset

Table 117. Queue Manager Free Descriptor/Buffer Starvation Count Register 1 (FDBSC1) Field Descriptions

Bit	Field	Value	Description
15-8	FDBQ3_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 3 is read while it is empty. This field is cleared when readby CPU.
7-0	FDBQ2_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 2 is read while it is empty. This field is cleared when readby CPU.

3.66 Queue Manager Free Descriptor/Buffer Starvation Count Register 2 (FDBSC2)

The free descriptor/buffer queue starvation count register 2 (FDBSC2) provides statistics about how many starvation events are occurring on the receive free descriptor/buffer queues. These registers do not support byte accesses. The free descriptor/buffer queue starvation count register 2 (FDBSC2) is shown in Figure 117 and described in Table 118.

Figure 117. Queue Manager Free Descriptor/Buffer Starvation Count Register 2 (FDBSC2)

15 — 8	7 — 0
FDBQ5_STARVE_CNT	FDBQ4_STARVE_CNT
RC-0	RC-0

LEGEND: RC = Cleared on read; -*n* = value after reset

Table 118. Queue Manager Free Descriptor/Buffer Starvation Count Register 2 (FDBSC2) Field Descriptions

Bit	Field	Value	Description
15-8	FDBQ5_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 5 is read while it is empty. This field is cleared when read by CPU.
7-0	FDBQ4_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 4 is read while it is empty. This field is cleared when read by CPU.

3.67 Queue Manager Free Descriptor/Buffer Starvation Count Register 3 (FDBSC3)

The free descriptor/buffer queue starvation count register 3 (FDBSC3) provides statistics about how many starvation events are occurring on the receive free descriptor/buffer queues. These registers do not support byte accesses. The free descriptor/buffer queue starvation count register 3 (FDBSC3) is shown in Figure 118 and described in Table 119.

Figure 118. Queue Manager Free Descriptor/Buffer Starvation Count Register 3 (FDBSC3)

15 — 8	7 — 0
FDBQ7_STARVE_CNT	FDBQ6_STARVE_CNT
RC-0	RC-0

LEGEND: RC = Cleared on read; -*n* = value after reset

Table 119. Queue Manager Free Descriptor/Buffer Starvation Count Register 3 (FDBSC3) Field Descriptions

Bit	Field	Value	Description
15-8	FDBQ7_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 7 is read while it is empty. This field is cleared when read by CPU.
7-0	FDBQ6_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 6 is read while it is empty. This field is cleared when read by CPU.

3.68 Queue Manager Free Descriptor/Buffer Starvation Count Register 4 (FDBSC4)

The free descriptor/buffer queue starvation count register 4 (FDBSC4) provides statistics about how many starvation events are occurring on the receive free descriptor/buffer queues. These registers do not support byte accesses. The free descriptor/buffer queue starvation count register 4 (FDBSC4) is shown in Figure 119 and described in Table 120.

Figure 119. Queue Manager Free Descriptor/Buffer Starvation Count Register 4 (FDBSC4)

15 8	7 0
FDBQ9_STARVE_CNT	FDBQ8_STARVE_CNT
RC-0	RC-0

LEGEND: RC = Cleared on read; -*n* = value after reset

Table 120. Queue Manager Free Descriptor/Buffer Starvation Count Register 4 (FDBSC4) Field Descriptions

Bit	Field	Value	Description
15-8	FDBQ9_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 9 is read while it is empty. This field is cleared when read by CPU.
7-0	FDBQ8_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 8 is read while it is empty. This field is cleared when read by CPU.

3.69 Queue Manager Free Descriptor/Buffer Starvation Count Register 5 (FDBSC5)

The free descriptor/buffer queue starvation count register 5 (FDBSC5) provides statistics about how many starvation events are occurring on the receive free descriptor/buffer queues. These registers do not support byte accesses. The free descriptor/buffer queue starvation count register 5 (FDBSC5) is shown in Figure 120 and described in Figure 120.

Figure 120. Queue Manager Free Descriptor/Buffer Starvation Count Register 5 (FDBSC5)

15 8	7 0
FDBQ11_STARVE_CNT	FDB10_STARVE_CNT
RC-0	RC-0

LEGEND: RC = Cleared on read; -*n* = value after reset

Table 121. Queue Manager Free Descriptor/Buffer Starvation Count Register 5 (FDBSC5) Field Descriptions

Bit	Field	Value	Description
15-8	FDBQ11_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 11 is read while it is empty. This field is cleared when read by CPU.
7-0	FDBQ10_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 10 is read while it is empty. This field is cleared when read by CPU.

3.70 Queue Manager Free Descriptor/Buffer Starvation Count Register 6 (FDBSC6)

The free descriptor/buffer queue starvation count register 6 (FDBSC6) provides statistics about how many starvation events are occurring on the receive free descriptor/buffer queues. These registers do not support byte accesses. The free descriptor/buffer queue starvation count register 6 (FDBSC6) is shown in Figure 121 and described in Table 122.

Figure 121. Queue Manager Free Descriptor/Buffer Starvation Count Register 6 (FDBSC6)

15 8	7 0
FDBQ13_STARVE_CNT	FDBQ12_STARVE_CNT
RC-0	RC-0

LEGEND: RC = Cleared on read; -*n* = value after reset

Table 122. Queue Manager Free Descriptor/Buffer Starvation Count Register 6 (FDBSC6) Field Descriptions

Bit	Field	Value	Description
15-8	FDBQ13_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 13 is read while it is empty. This field is cleared when read by CPU.
7-0	FDBQ12_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 12 is read while it is empty. This field is cleared when read by CPU.

3.71 Queue Manager Free Descriptor/Buffer Starvation Count Register 7 (FDBSC7)

The free descriptor/buffer queue starvation count register 7 (FDBSC7) provides statistics about how many starvation events are occurring on the receive free descriptor/buffer queues. The registers do not support byte accesses. The free descriptor/buffer queue starvation count register 7 (FDBSC7) is shown in Figure 122 and described in Table 123.

Figure 122. Queue Manager Free Descriptor/Buffer Starvation Count Register 7 (FDBSC7)

15 8	7 0
FDBQ15_STARVE_CNT	FDB14_STARVE_CNT
RC-0	RC-0

LEGEND: RC = Cleared on read; -*n* = value after reset

Table 123. Queue Manager Free Descriptor/Buffer Starvation Count Register 7 (FDBSC7) Field Descriptions

Bit	Field	Value	Description
15-8	FDBQ15_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 15 is read while it is empty. This field is cleared when read by CPU.
7-0	FDBQ14_STARVE_CNT	0-FFh	This field increments each time the Free Descriptor/Buffer Queue 14 is read while it is empty. This field is cleared when read by CPU.

3.72 Queue Manager Linking RAM Region 0 Base Address Registers (LRAM0BASE1 and LRAM0BASE2)

The queue manager linking RAM region 0 base address registers (LRAM0BASE1 and LRAM0BASE2) set the base address for the first portion of the Linking RAM. This address must be 32-bit aligned. It is used by the Queue Manager to calculate the 32-bit linking address for a given descriptor index. These registers do not support byte accesses.

The queue manager linking RAM region 0 base address register 1 (LRAM0BASE1) is shown in Figure 123 and described in Table 124. The queue manager linking RAM region 0 base address register 2 (LRAM0BASE2) is shown in Figure 124 and described in Table 125.

Figure 123. Queue Manager Linking RAM Region 0 Base Address Register 1 (LRAM0BASE1)

15 ... 0
REGION0_BASE_LSB
R/W-0

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

Figure 124. Queue Manager Linking RAM Region 0 Base Address Register 2 (LRAM0BASE2)

15 ... 0
REGION0_BASE_MSB
R/W-0

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

Table 124. Queue Manager Linking RAM Region 0 Base Address Register 1 (LRAM0BASE1) Field Descriptions

Bit	Field	Value	Description
15-0	REGION0_BASE_LSB	0-FFFFh	This field stores the 16 least significant bits of the base address for the first region of the linking RAM. This may be anywhere in 32-bit address space but would be typically located in on-chip memory.

Table 125. Queue Manager Linking RAM Region 0 Base Address Register 2 (LRAM0BASE2) Field Descriptions

Bit	Field	Value	Description
15-0	REGION0_BASE_MSB	0-FFFFh	This field stores the 16 most significant bits of the base address for the first region of the linking RAM. This may be anywhere in 32-bit address space but would be typically located in on-chip memory.

3.73 Queue Manager Linking RAM Region 0 Size Register (LRAM0SIZE)

The queue manager linking RAM region 0 size register (LRAM0SIZE) sets the size of the array of linking pointers that are located in Region 0 of Linking RAM. The size specified the number of descriptors for which linking information is stored in this region. It does not support byte accesses. The queue manager linking RAM region 0 size register (LRAM0SIZE) is shown in Figure 125 and described in Table 126.

Figure 125. Queue Manager Linking RAM Region 0 Size Register (LRAM0SIZE)

15 14	13 0
Reserved	REGION0_SIZE
R-0	R/W-0

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

Table 126. Queue Manager Linking RAM Region 0 Size Register (LRAM0SIZE) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-0	REGION0_SIZE	0-3FFh	This field indicates the number of entries that are contained in the linking RAM region 0. A descriptor with index less than region0_size value has its linking location in region 0. A descriptor with index greater than region0_size has its linking location in region 1. The queue manager will add the index (left shifted by 2 bits) to the appropriate regionX_base_addr to get the absolute 32-bit address to the linking location for a descriptor.

3.74 Queue Manager Linking RAM Region 1 Base Address Registers (LRAM1BASE1 and LRAM1BASE2)

The queue manager linking RAM region 1 base address registers (LRAM1BASE1 and LRAM1BASE2) are used to set the base address for the first portion of the Linking RAM. This address must be 32-bit aligned. These registers are used by the Queue Manager to calculate the 32-bit linking address for a given descriptor index. These registers do not support byte accesses.

The queue manager linking RAM region 1 base address register (LRAM1BASE1) is shown in Figure 126 and described in Table 127. The queue manager linking RAM region 1 base address register (LRAM1BASE2) is shown in Figure 127 and described in Table 128.

Figure 126. Queue Manager Linking RAM Region 1 Base Address Register 1 (LRAM1BASE1)

15 … 0
REGION1_BASE_LSB
R/W-0

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

Figure 127. Queue Manager Linking RAM Region 1 Base Address Register 2 (LRAM1BASE2)

15 … 0
REGION1_BASE_MSB
R/W-0

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

Table 127. Queue Manager Linking RAM Region 1 Base Address Register 1 (LRAM1BASE1) Field Descriptions

Bit	Field	Value	Description
15-0	REGION1_BASE_LSB	0-FFFFh	This field stores the least significant bits of the base address for the second region of the linking RAM. This may be anywhere in 32-bit address space but would be typically located in off-chip memory.

Table 128. Queue Manager Linking RAM Region 1 Base Address Register (LRAM1BASE2) Field Descriptions

Bit	Field	Value	Description
15-0	REGION1_BASE_MSB	0-FFFFh	This field stores the most significant bits of the base address for the second region of the linking RAM. This may be anywhere in 32-bit address space but would be typically located in off-chip memory.

3.75 Queue Manager Queue Pending Register 0 (PEND0)

The queue pending register 0 (PEND0) can be read to find the pending status for queues 15 to 0. It does not support byte accesses. The queue pending register 0 (PEND0) is shown in Figure 128 and described in Table 129.

Figure 128. Queue Manager Queue Pending Register 0 (PEND0)

15 ... 0
QPEND0
R-0

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

Table 129. Queue Manager Queue Pending Register 0 (PEND0) Field Descriptions

Bit	Field	Value	Description
15-0	QPEND0	0-FFFFh	This field indicates the queue pending status for queues 15-0.

3.76 Queue Manager Queue Pending Register 1 (PEND1)

The queue pending register 1 (PEND1) can be read to find the pending status for queues 31 to 16. It does not support byte accesses. The queue pending register 1 (PEND1) is shown in Figure 129 and described in Table 130.

Figure 129. Queue Manager Queue Pending Register 1 (PEND1)

15 ... 0
QPEND1
R-0

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

Table 130. Queue Manager Queue Pending Register 1 (PEND1) Field Descriptions

Bit	Field	Value	Description
15-0	QPEND1	0-FFFFh	This field indicates the queue pending status for queues 31-16.

3.77 Queue Manager Queue Pending Register 2 (PEND2)

The queue pending register 2 (PEND2) can be read to find the pending status for queues 47 to 32. It does not support byte accesses. The queue pending register 2 (PEND2) is shown in Figure 130 and described in Table 131.

Figure 130. Queue Manager Queue Pending Register 2 (PEND2)

15 — 0
QPEND2
R-0

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

Table 131. Queue Manager Queue Pending Register 2 (PEND2) Field Descriptions

Bit	Field	Value	Description
15-0	QPEND2	0-FFFFh	This field indicates the queue pending status for queues 47-32.

3.78 Queue Manager Queue Pending Register 3 (PEND3)

The queue pending register 3 (PEND3) can be read to find the pending status for queues 63 to 48. It does not support byte accesses. The queue pending register 3 (PEND3) is shown in Figure 131 and described in Table 132.

Figure 131. Queue Manager Queue Pending Register 3 (PEND3)

15 — 0
QPEND3
R-0

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

Table 132. Queue Manager Queue Pending Register 3 (PEND3) Field Descriptions

Bit	Field	Value	Description
15-0	QPEND3	0-FFFFh	This field indicates the queue pending status for queues 63-48.

3.79 Queue Manager Queue Pending Register 4 (PEND4)

The queue pending register 4 (PEND4) can be read to find the pending status for queues 79 to 64. It does not support byte accesses. The queue pending register 4 (PEND4) is shown in Figure 132 and described in Table 133.

Figure 132. Queue Manager Queue Pending Register 4 (PEND4)

15 … 0
QPEND4
R-0

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

Table 133. Queue Manager Queue Pending Register 4 (PEND4) Field Descriptions

Bit	Field	Value	Description
15-0	QPEND4	0-FFFFh	This field indicates the queue pending status for queues 79-64.

3.80 Queue Manager Queue Pending Register 5 (PEND5)

The queue pending register 5 (PEND5) can be read to find the pending status for queues 95 to 80. It does not support byte accesses. The queue pending register 5 (PEND5) is shown in Figure 133 and described in Table 134.

Figure 133. Queue Manager Queue Pending Register 5 (PEND5)

15 … 0
QPEND5
R-0

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

Table 134. Queue Manager Queue Pending Register 5 (PEND5) Field Descriptions

Bit	Field	Value	Description
15-0	QPEND5	0-FFFFh	This field indicates the queue pending status for queues 95-80.

3.81 Queue Manager Memory Region *R* Base Address Registers (QMEMRBASE1[R] and QMEMRBASE2[R])

The memory region *R* base address registers (QMEMRBASE1[R] and QMEMRBASE2[R]) are written by the host to set the base address of memory region *R*, where *R* is 0-15. This memory region will store a number of descriptors of a particular size as determined by the memory region *R* control register. These registers do not support byte accesses.

The memory region *R* base address register (QMEMRBASE1[*R*]) is shown in Figure 134 and described in Table 135. The memory region *R* base address register (QMEMRBASE2[*R*]) is shown in Figure 135 and described in Table 136.

Figure 134. Queue Manager Memory Region *R* Base Address Register 1 (QMEMRBASE1[*R*])

15 … 0
REG_LSB
R/W-0

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

Figure 135. Queue Manager Memory Region *R* Base Address Register 2 (QMEMRBASE2[*R*])

15 … 0
REG_MSB
R/W-0

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

Table 135. Queue Manager Memory Region *R* Base Address Register 1 (QMEMRBASE1[*R*]) Field Descriptions

Bit	Field	Value	Description
15-0	REG_LSB	0-FFFFh	This field contains the least-significant bits of the base address of the memory region R.

Table 136. Queue Manager Memory Region *R* Base Address Register 2 (QMEMRBASE2[*R*]) Field Descriptions

Bit	Field	Value	Description
15-0	REG_MSB	0-FFFFh	This field contains the most-significant bits of the base address of the memory region R.

3.82 Queue Manager Memory Region *R* Control Registers (QMEMRCTRL1[R] and QMEMRCTRL2[R])

The memory region *R* control registers (QMEMRCTRL1[R] and QMEMRCTRL2[R]) are written by the host to configure various parameters of memory region *R*, where *R* is 0-15. These registers do not support byte accesses.

The memory region *R* control register (QMEMRCTRL1[*R*])) is shown in Figure 136 and described in Table 137. The memory region *R* control register (QMEMRCTRL2[*R*])) is shown in Figure 137 and described in Table 138.

Figure 136. Queue Manager Memory Region *R* Control Register 1 (QMEMRCTRL1[*R*])

15 - 12	11 - 8	7 - 3	2 - 0
Reserved	DESC_SIZE	Reserved	REG_SIZE
R-0	R/W-0	R-0	R/W-0

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

Figure 137. Queue Manager Memory Region *R* Control Register 2 (QMEMRCTRL2[*R*])

15 - 14	13 - 0
Reserved	START_INDEX
R-0	R/W-0

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

Table 137. Queue Manager Memory Region *R* Control Register 1 (QMEMRCTRL1[*R*]) Field Descriptions

Bit	Field	Value	Description
15-12	Reserved	0	Reserved.
11-8	DESC_SIZE	0-Fh	This field indicates the size of each descriptor in this memory region.
		0	32.
		1h	64.
		2h	128.
		3h	256.
		4h	512.
		5h	1K.
		6h	2K.
		7h	4K.
		8h	8K.
		9h-Fh	Reserved.
7-3	Reserved	0	Reserved.
2-0	REG_SIZE	0-7h	This field indicates the size of the memory region (in terms of number of descriptors).
		0	32.
		1h	64.
		2h	128.
		3h	256.
		4h	512.
		5h	1K.
		6h	2K.
		7h	4K.

Table 138. Queue Manager Memory Region *R* Control Register 2 (QMEMRCTRL2[*R*]) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-0	START_INDEX	0-3FFFh	This field indicates where in linking RAM the descriptor linking information corresponding to memory region R starts.

3.83 Queue Manager Queue N Control Register D (CTRL1D[N] and CTRL2D[N])

The queue manager queue *N* control registers D (CTRL1D[N] and CTRL2D[N]) are written to add a packet to the queue and read to pop a packets off a queue. The packet is only pushed or popped to/from the queue when the queue manager queue *N* control register D is written. These registers do not support byte accesses.

The queue manager queue *N* control register 1 D (CTRL1D[*N*]) is shown in Figure 138 and described in Table 139. The queue manager queue *N* control register 2 D (CTRL2D[*N*]) is shown in Figure 139 and described in Table 140.

Figure 138. Queue Manager Queue *N* Control Register 1 D (CTRL1D[*N*])

15 — 5	4 — 0
DESC_PTR_LSB	DESC_SIZE
R/W-0	R/W-0

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

Figure 139. Queue Manager Queue *N* Control Register 2 D (CTRL2D[*N*])

15 — 0
DESC_PTR_MSB
R/W-0

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

Table 139. Queue Manager Queue *N* Control Register 1 D (CTRL1D[*N*]) Field Descriptions

Bit	Field	Value	Description
15-5	DESC_PTR_LSB		Descriptor Pointer (Least significant bits).
		0	Queue is empty.
		1	Indicates a 32-bit aligned address that points to a descriptor.
4-0	DESC_SIZE	0-1Fh	The descriptor size is encoded in 4-byte increments. This field returns a 0 when an empty queue is read.
		0	24 bytes.
		1h	28 bytes.
		2h	32 bytes.
		3h-1Fh	36 bytes to 148 bytes.

Table 140. Queue Manager Queue *N* Control Register 2 D (CTRL2D[*N*]) Field Descriptions

Bit	Field	Value	Description
15-0	DESC_PTR_MSB		Descriptor Pointer (Most significant bits).
		0	Queue is empty.
		1	Indicates a 32-bit aligned address that points to a descriptor.

3.84 Queue Manager Queue *N* Status Register A (QSTATA[N])

The queue manager queue *N* status register A (QSTATA[*N*]) is an optional register that is only implemented for a queue if the queue supports entry/byte count feature. The entry count feature provides a count of the number of entries that are currently valid in the queue. It does not support byte accesses. The queue manager queue *N* status register A (QSTATA[*N*]) is shown in Figure 140 and described in Table 141.

Figure 140. Queue Manager Queue *N* Status Register A (QSTATA[*N*])

15 14	13 0
Reserved	QUEUE_ENTRY_COUNT
R-0	R-0

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

Table 141. Queue Manager Queue *N* Status Register A (QSTATA[*N*]) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-0	QUEUE_ENTRY_COUNT	0-3FFFh	This field indicates how many packets are currently queued on the queue.

3.85 Queue Manager Queue *N* Status Registers B (QSTAT1B[N] and QSTAT2B[N])

The queue manager queue *N* status registers B (QSTAT1B[N] and QSTAT2B[N]) are optional registers that are only implemented for a queue if the queue supports a total byte count feature. The total byte count feature provides a count of the total number of bytes in all of the packets that are currently valid in the queue. The registers do not support byte accesses.

The queue manager queue *N* status register 1 B (QSTAT1B[*N*]) is shown in Figure 141 and described in Table 142. The queue manager queue *N* status register 2 B (QSTAT2B[*N*]) is shown in Figure 142 and described in Table 143.

Figure 141. Queue Manager Queue *N* Status Register 1 B (QSTAT1B[*N*])

15 0
QUEUE_BYTE_COUNT_LSB
R-0

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

Figure 142. Queue Manager Queue *N* Status Register 2 B (QSTAT2B[*N*])

15 12	11 0
Reserved	QUEUE_BYTE_COUNT_MSB

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

Table 142. Queue Manager Queue *N* Status Register 1 B (QSTAT1B[*N*]) Field Descriptions

Bit	Field	Value	Description
15-0	QUEUE_BYTE_COUNT_LSB	0-FFFFh	Together, QUEUE_BYTE_COUNT_MSB and QUEUE_BYTE_COUNT_LSB indicate how many bytes total are contained in all of the packets which are currently queued on this queue.

Table 143. Queue Manager Queue *N* Status Register 2 B (QSTAT2B[*N*]) Field Descriptions

Bit	Field	Value	Description
15-12	Reserved	0	Reserved.
11-0	QUEUE_BYTE_COUNT_MSB	0-FFFh	Together, QUEUE_BYTE_COUNT_MSB and QUEUE_BYTE_COUNT_LSB indicate how many bytes total are contained in all of the packets which are currently queued on this queue.

3.86 Queue Manager Queue *N* Status Register C (QSTATC[N])

The queue manager queue *N* status register C (QSTATC[*N*]) specifies the packet size for the head element of a queue. It does not support byte accesses. The queue manager queue *N* status register C (QSTATC[*N*]) is shown in Figure 143 and described in Table 144.

Figure 143. Queue Manager Queue *N* Status Register C (QSTATC[*N*])

15 14	13 ... 0
Reserved	PACKET_SIZE
R-0	R-0

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

Table 144. Queue Manager Queue *N* Status Register C (QSTATC[*N*]) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved.
13-0	PACKET_SIZE	0-3FFFh	This field indicates how many packets are currently queued on the queue.