TMS320C674x/OMAP-L1x Processor Universal Serial Bus 2.0 (USB2.0) Controller

User's Guide

Literature Number: SPRUFM9H
July 2010

List of Figures

List of Tables

Read This First

About This Manual

This document describes the universal serial bus (USB) controller. The controller complies with the USB 2.0 standard high-speed and full-speed functions and low-speed, full-speed, and high-speed limited host mode operations. It also includes support for the Session Request and Host Negotiation Protocols used in point-to-point communications, details of which are given in the USB On-The-Go supplement to the USB 2.0 specification. 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, high-speed mode, or host mode that may be used for debug purposes.

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 TMS320C674x Digital Signal Processors (DSPs) and OMAP-L1x Applications Processors. Copies of these documents are available on the Internet at www.ti.com. *Tip:* Enter the literature number in the search box provided at www.ti.com.

The current documentation that describes the DSP, related peripherals, and other technical collateral, is available in the C6000 DSP product folder at: www.ti.com/c6000.

SPRUGM5 — ***TMS320C6742 DSP System Reference Guide.*** Describes the C6742 DSP subsystem, system memory, device clocking, phase-locked loop controller (PLLC), power and sleep controller (PSC), power management, and system configuration module.

SPRUGJ0 — ***TMS320C6743 DSP System Reference Guide.*** Describes the System-on-Chip (SoC) including the C6743 DSP subsystem, system memory, device clocking, phase-locked loop controller (PLLC), power and sleep controller (PSC), power management, and system configuration module.

SPRUFK4 — ***TMS320C6745/C6747 DSP System Reference Guide.*** Describes the System-on-Chip (SoC) including the C6745/C6747 DSP subsystem, system memory, device clocking, phase-locked loop controller (PLLC), power and sleep controller (PSC), power management, and system configuration module.

SPRUGM6 — ***TMS320C6746 DSP System Reference Guide.*** Describes the C6746 DSP subsystem, system memory, device clocking, phase-locked loop controller (PLLC), power and sleep controller (PSC), power management, and system configuration module.

SPRUGJ7 — ***TMS320C6748 DSP System Reference Guide.*** Describes the C6748 DSP subsystem, system memory, device clocking, phase-locked loop controller (PLLC), power and sleep controller (PSC), power management, and system configuration module.

SPRUG84 — *OMAP-L137 Applications Processor System Reference Guide.* Describes the System-on-Chip (SoC) including the ARM subsystem, DSP subsystem, system memory, device clocking, phase-locked loop controller (PLLC), power and sleep controller (PSC), power management, ARM interrupt controller (AINTC), and system configuration module.

SPRUGM7 — *OMAP-L138 Applications Processor System Reference Guide.* Describes the System-on-Chip (SoC) including the ARM subsystem, DSP subsystem, system memory, device clocking, phase-locked loop controller (PLLC), power and sleep controller (PSC), power management, ARM interrupt controller (AINTC), and system configuration module.

SPRUFK9 — *TMS320C674x/OMAP-L1x Processor Peripherals Overview Reference Guide.* Provides an overview and briefly describes the peripherals available on the TMS320C674x Digital Signal Processors (DSPs) and OMAP-L1x Applications Processors.

SPRUFK5 — *TMS320C674x DSP Megamodule Reference Guide.* Describes the TMS320C674x digital signal processor (DSP) megamodule. Included is a discussion on the internal direct memory access (IDMA) controller, the interrupt controller, the power-down controller, memory protection, bandwidth management, and the memory and cache.

SPRUFE8 — *TMS320C674x DSP CPU and Instruction Set Reference Guide.* Describes the CPU architecture, pipeline, instruction set, and interrupts for the TMS320C674x digital signal processors (DSPs). The C674x DSP is an enhancement of the C64x+ and C67x+ DSPs with added functionality and an expanded instruction set.

SPRUG82 — *TMS320C674x DSP Cache User's Guide.* Explains the fundamentals of memory caches and describes how the two-level cache-based internal memory architecture in the TMS320C674x digital signal processor (DSP) can be efficiently used in DSP applications. Shows how to maintain coherence with external memory, how to use DMA to reduce memory latencies, and how to optimize your code to improve cache efficiency. The internal memory architecture in the C674x DSP is organized in a two-level hierarchy consisting of a dedicated program cache (L1P) and a dedicated data cache (L1D) on the first level. Accesses by the CPU to the these first level caches can complete without CPU pipeline stalls. If the data requested by the CPU is not contained in cache, it is fetched from the next lower memory level, L2 or external memory.

Universal Serial Bus 2.0 (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 and low-speed, full-speed, and high-speed limited host mode operations. It also includes support for the Session Request and Host Negotiation Protocols used in point-to-point communications, details of which are given in the USB On-The-Go supplement to the USB 2.0 specification. 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, high-speed mode, or host 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. Its support for a dual-role feature allows for additional versatility supporting operation capability as a host or peripheral.

1.2 Features

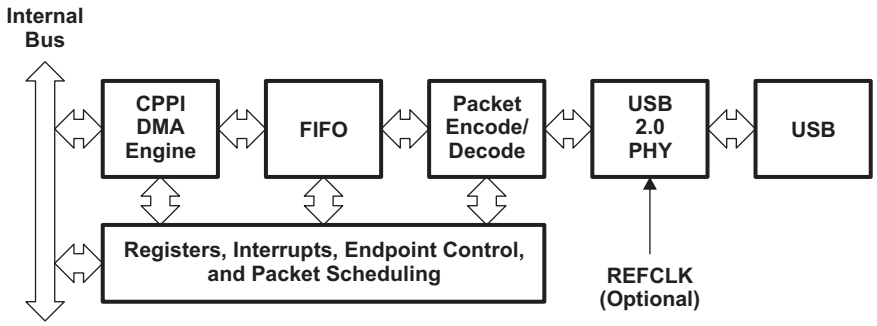
The USB has the following features:

- Operating as a host, it complies with the USB 2.0 standard for high-speed (480 Mbps), full-speed (12 Mbps), and low-speed (1.5 Mbps) operations with a peripheral
- 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 USB extensions for Session Request (SRP) and Host Negotiation (HNP) – OTG
- 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
- External 5V power supply for VBUS, when operating as host, enabled directly by the USB controller through a dedicated signal
- 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



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 no built-in oscillator. The USB2.0 subsystem requires a reference clock for its internal PLL. This reference clock can be sourced from either the USB_REFCLKIN pin or from the AUXCLK of the system PLL. The reference clock input to the USB2.0 subsystem is selected by programming the USB0PHYCLKMUX bit in the chip configuration 2 register (CFGCHIP2) of the System Configuration Module. The USB_REFCLKIN source should be selected when it is not possible (such as when specific audio rates are required) to operate the device at one of the allowed input frequencies to the USB2.0 subsystem. The USB2.0 subsystem peripheral bus clock is sourced from SYSCLK2. Table 1 determines the source origination as well as the source input frequency to the USB 2.0 PHY. Once the clock source origination (internal/external) and its frequency is determined, the firmware should program the PHY PLL with the correct input frequency via CFGCHIP2.USB0REF_FREQ (see Table 2).

NOTE: Prior to accessing any of the device configuration registers, including CFGCHIP2, in order to avoid inadvertent access, two Access Key Registers (KICK0R and KICK1R) should be written with key values. For more information on the device configuration registers, see your device-specific data manual.

Figure 2. USB Clocking Diagram

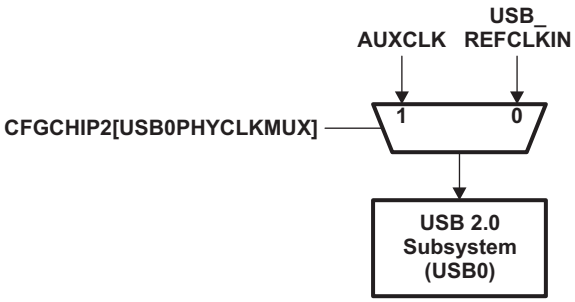


Table 1. USB2.0 and USB1.1 Clock Multiplexing Options

CFGCHIP2. USB0PHYCLKMUX bit	CFGCHIP2. USB1PHYCLKMUX bit	USB2.0 Clock Source	USB1.1 Clock Source	Additional Conditions
0	0	USB_REFCLKIN	CLK48MHZ output from USB2.0 PHY	USB_REFCLKIN must be 12, 24, 48, 19.2, 38.4, 13, 26, 20, or 40 MHz. The PLL inside the USB2.0 PHY can be configured to accept any of these input clock frequencies.
0	1	USB_REFCLKIN	USB_REFCLKIN	USB_REFCLKIN must be 48 MHz. The PLL inside the USB2.0 PHY can be configured to accept this input clock frequency.
1	0	PLL0_AUXCLK	CLK48MHZ output from USB2.0 PHY	PLL0_AUXCLK must be 12, 24, 48, 19.2, 38.4, 13, 26, 20, or 40 MHz. The PLL inside the USB2.0 PHY can be configured to accept any of these input clock frequencies.
1	1	PLL0_AUXCLK	USB_REFCLKIN	PLL0_AUXCLK must be 12, 24, 48, 19.2, 38.4, 13, 26, 20, or 40 MHz. The PLL inside the USB2.0 PHY can be configured to accept any of these input clock frequencies. USB_REFCLKIN must be 48 MHz.

Table 2. PHY PLL Clock Frequencies Supported

CFGCHIP2.USB0REF_FREQ bit	Frequency
1h	12.0 MHz
2h	24.0 MHz
3h	48.0 MHz
4h	19.2 MHz
5h	38.4 MHz
6h	13.0 MHz
7h	26.0 MHz
8h	20.0 MHz
9h	40.0 MHz

2.2 Signal Descriptions

The USB controller provides the I/O signals listed in Table 3.

Table 3. USB Terminal Functions

Name	I/O [1]	Description
USB0_DP	A I/O/Z	USB0 D+ (differential signal pair)
USB0_DM	A I/O/Z	USB0 D- (differential signal pair)
USB0_ID	A I/O	USB0 operating mode identification pin. For OTG mode or device only mode of operation, do NOT connect the USB0_ID pin, that is, leave the pin floating. For host only mode of operation, connect the USB0_ID pin to ground via a 0 ohm resistor or connect the pin directly to ground.
USB0_VBUS	A I/O/Z	5 volt input that signifies that VBUS is connected. The OTG section of the PHY can also pull-up/pull-down on this signal for HNP and SRP. For device or host only mode of operation, pull-up this pin to 5V with an external 1K ohm resistor. For host mode of operation, pull-up the USB power signal on the USB connector to 5V also. For mixed host/device mode of operation, tie this to the charge pump.
USB0_DRVVBUS	I/O/Z	Digital output to control external 5-V supply
USB0_VDDA33	I/O/Z	USB0 PHY 3.3V supply
USB0_VDDA18	I/O/Z	USB0 PHY 1.8V supply input
USB_REFCLKIN	I/O/Z	External clock input for USB PHY
USB0_VDDA12	I/O/Z	USB PHY 1.2V LD0 output for bypass CAP

[1] A = Analog signal; I = Input; O = Output; Z = High impedance

2.3 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 memory-mapped at offset 410h to 41Fh. The endpoint is selected by programming the INDEX register of the controller.
- Non-indexed Endpoint Control/Status Registers: These registers are memory-mapped at offset 500h to 54Fh. Registers at offset 500h to 50Fh map to Endpoint 0; at offset 510h to 51Fh map to Endpoint 1, and so on.

For detailed information about the USB controller registers, see Section 4.

2.4 USB PHY Initialization

Two boot configuration registers, pin multiplexing control registers, and chip configuration 2 register (CFGCHIP2) are used to configure the multiplexed pins for USB 2.0 uses. See your device-specific *System Reference Guide* for more information on the pin multiplexing control registers and CFGCHIP2.

The general procedure for USB PHY initialization starts by releasing the PHY from reset and programming the corresponding bits within the pin multiplexing control registers and CFGCHIP2, for achieving the multiplexed pins for the use of the USB2.0. Next, configuring PHY input clock related information and other PHY general configuration attributes.

When the USB2.0 controller assumes the role of a host, it is tasked to source the required 5V supply via USB0_VBUS (must be at least ≥ 4.75V) pin. The USB2.0 controller makes use of an external charge pump or logic by enabling and disabling the external power logic from the USB2.0 controller core level. It uses the USB0_DRVVBUS for controlling the enable/disable state of the external power logic. In order to achieve this task, the pin multiplexing control registers should be configured accordingly to map the USB0_DRVVBUS pin to be used for USB2.0 purposes. In addition, the source (internal or external) and frequency of the PHY clock should be identified and should be configured by the firmware. This is

achieved using CFGCHIP2. Other PHY related fields within CFGCHIP2 should be programmed as: USB0PHYPWDN and USB0OTGPWRDN should be cleared to 0 and USB0DATPOL, USB0SESNDEN, and USB0VBDTCTEN should be set to 1. This will configure the PHY for normal operation as well as also turn on the PHYs VBUS comparator logic. The final task is to turn on the PHY PLL and wait until it locks. You should wait for the PHY clock good status to be set prior to ending the PHY initialization process.

2.5 VBUS Voltage Sourcing Control

When the USB controller assumes the role of a host, it is required to supply 5V power to an attached device through its VBUS line. In order to achieve this task, the USB controller requires the use of an external logic (or charge pump) capable of sourcing 5V power. A USB_DRVVBUS is used as a control signal to enable/disable the external logic to either source or disable power on the VBUS line. This control is automatic and is handled by the controller. The USB controller drives the USB_DRVVBUS signal high when it assumes the role of a host while the controller is in Session. When assuming the role of a device, the controller drives the USB_DRVVBUS signal low disabling the external charge pump; hence, no power is driven on the VBUS line.

2.6 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 4.55, Section 4.56, Section 4.57, and Section 4.58 provide details of these registers.

NOTE: The option of dynamically setting 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.

2.7 USB Controller Host and Peripheral Modes Operation

The USB controller can be used in a range of different environments. It can be used as either a high-speed or a full-speed USB peripheral device attached to a conventional USB host (such as a PC). It can be used as either a host or a peripheral device in a point-to-point type of setup/arrangement or it can be used as a host connecting to a range of peripheral devices in a multi-point setup (that is, using a hub).

The USB2.0 controller role adaptation of a host or peripheral (device) is dependent upon the state of the USB0_ID pin on its mini-AB receptacle. If the USB0_ID pin state is not driven by the connector or is left floating, the USB2.0 controller would assume the role of a peripheral; if the USB0_ID pin state is driven low or is grounded, the USB2.0 controller would assume the role of a host. The state of the USB ID pin is controlled by the type of USB plug attached to the mini-AB connector. A mini/micro-B plug (peripheral) would leave the USB0_ID pin floating and a mini/micro-A plug (host) grounds the USB0_ID pin low. The procedure for the USB2.0 controller determining its operating modes (role of a host or a peripheral) starts when the USB 2.0 controller is in session. The USB 2.0 controller is in session when either it senses a voltage on the USB0_VBUS pin or when the firmware sets the DEVCTL[SESSION] bit.

Usually, the firmware sets the SESSION bit, when it assumes that it will be operating as a host. When the SESSION bit is set, the controller will start sensing the state of the USB0_ID pin. If the USB0_ID pin has been grounded low, then the USB2.0 controller will assume the role of a host; however, if the USB0_ID pin is left floating, then the USB2.0 controller will assume the role of a device. Upon determining its role as a host, it will drive the USB0_DRVVBUS pin high to enable the external power logic so that it start sourcing the required 5V power (must be ≥ 4.75V). The USB2.0 controller will then wait to for the voltage of the USB0_VBUS goes high. If it does not see the power on the USB0_VBUS pin greater than Vbus Valid (4.4V), it will generate an interrupt to the user indicating the existence of a problem. Assuming that the voltage level of the USB0_VBUS is found to be above Vbus Valid, then the USB 2.0 controller will wait for a device to connect, that is, for it to see one of its data lines USB0_DP/DM to be pulled high.

When assuming the role of a peripheral, assuming that the firmware has set the POWER[SOFTCONN] bit and has enabled the data lines and there is an external host sourcing power on the USB0_VBUS line, 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. The USB 2.0 controller will then set the SESSION bit upon detecting the power on the USB0_VBUS line. This will force the USB2.0 controller to sense the state of the USB0_ID pin. If the USB0_ID pin has been left floating, it will know that it has to assume the role of a device and will enable its 1.5 kohm pull-up resistor to signify to the attached external host that 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 await for a reset signal from the host.

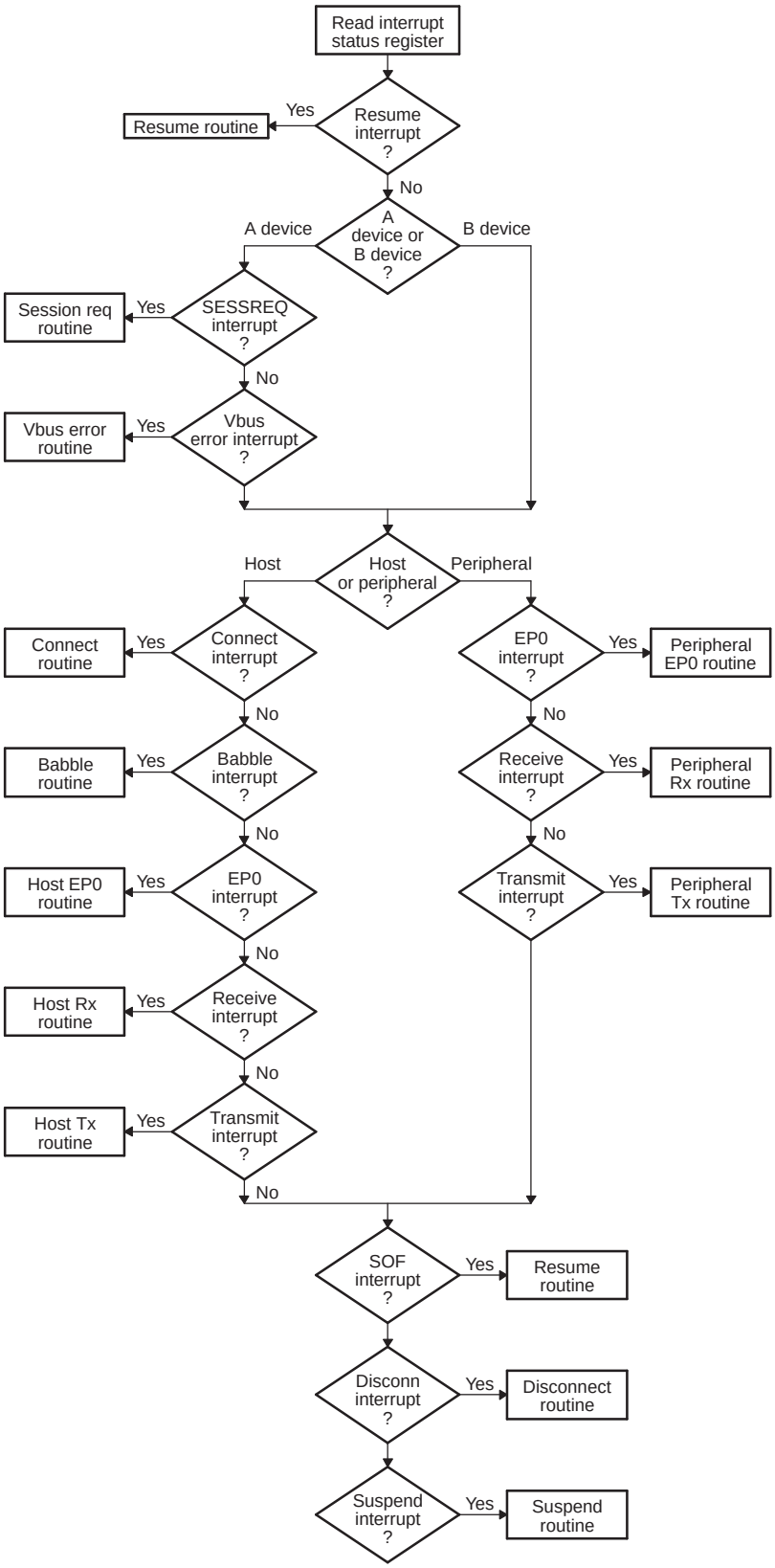
The USB controller interrupts the CPU on completion of the data transfer on any of the endpoints or on detecting reset, resume, suspend, connect, disconnect, or SOF on the bus.

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 3 describes the interrupt service routine for the USB module.

The following sections describe the programming of USB controller in Peripheral mode and Host mode. DMA operations and interrupt handler mechanisms are common to both peripheral and host mode operations and are discussed after the programming in peripheral and Host mode.

Figure 3. Interrupt Service Routine Flow Chart



2.7.1 USB Controller Peripheral Mode Operation

- *Soft connect* - After a reset, the SOFTCONN bit of POWER register (bit 6) 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 has been set, the software can also simulate a disconnect by clearing this bit to 0.
- *Entry into suspend mode* - When operating as a peripheral device, 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 time.

  At this point, the controller can then 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.
- *Resume Signaling* - 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.
- *Initiating a remote wakeup* - If the software wants to initiate a remote wakeup while the controller is in Suspend mode, it should write to the Power register to set the RESUME bit to 1. The software should leave then 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 wakeup.

- *Reset Signaling* - When reset signaling occurs on the bus, the controller performs the following actions:
  - Clears FADDR register to 0
  - Clears INDEX register to 0
  - Flushes all endpoint FIFOs
  - Clears all control/status registers
  - Generates a reset interrupt.

  If the HSENA bit in the POWER register (bit 5) was set, the controller also tries to negotiate for high-speed operation.

  Whether high-speed operation is selected is indicated by HSMODE bit of POWER register (bit 4).

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

2.7.1.1 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.

2.7.1.1.1 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 PERI_CSR0 (bit 0) 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 (bit 6) (indicating that the command has been read from the FIFO) and to set the DATAEND bit (bit 3) (indicating that no further data is expected for this request). The interval between setting SERV_RXPKTRDY bit and 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 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 (bit 6) and to set the SENDSTALL bit (bit 5). 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 (bit 2 of PERI_CSR0) 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 (bit 2 of PERI_CSR0) will be set.

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

2.7.1.1.2 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 (bit 6) (indicating that the command has been read from the FIFO) but in this case the DATAEND bit (bit 3) 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 of PERI_CSR0 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 (bit 6) and to set the SENDSTALL bit (bit 5). 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 of PERI_CSR0 (bit 2) will be set.

If the host sends more data after the DATAEND has been set, then the controller will send a STALL. An endpoint 0 interrupt will be generated and the SENTSTALL bit of PERI_CSR0 (bit 2) will be set.

2.7.1.1.3 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 of PERI_CSR0 (bit 0) will also have been set. The 8-byte command should then be read from the endpoint 0 FIFO and decoded. The PERI_CSR0 register should then be written to set the SERV_RXPKTRDY bit (bit 6) (indicating that the command has read from the FIFO).

The data to be sent to the host should then 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 (bit 1) (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 (bit 3) (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 (bit 6) and to set the SENDSTALL bit (bit 5). 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 of PERI_CSR0 (bit 2) will be set.

If the host requests more data after DATAEND (bit 3) has been set, then the controller will send a STALL. An endpoint 0 interrupt will be generated and the SENTSTALL bit of PERI_CSR0 (bit 2) will be set.

2.7.1.1.4 Endpoint 0 States

When the USB controller is operating as a peripheral device, 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. RXPKTRDY bit of PERI_CSR0 (bit 0) 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 4.

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 5 .

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

Figure 4. CPU Actions at Transfer Phases

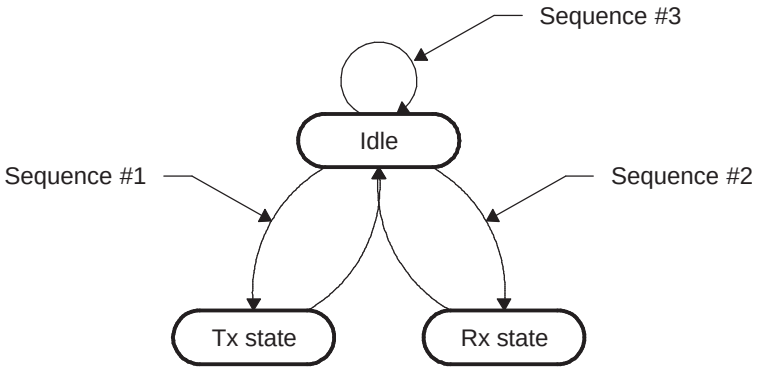
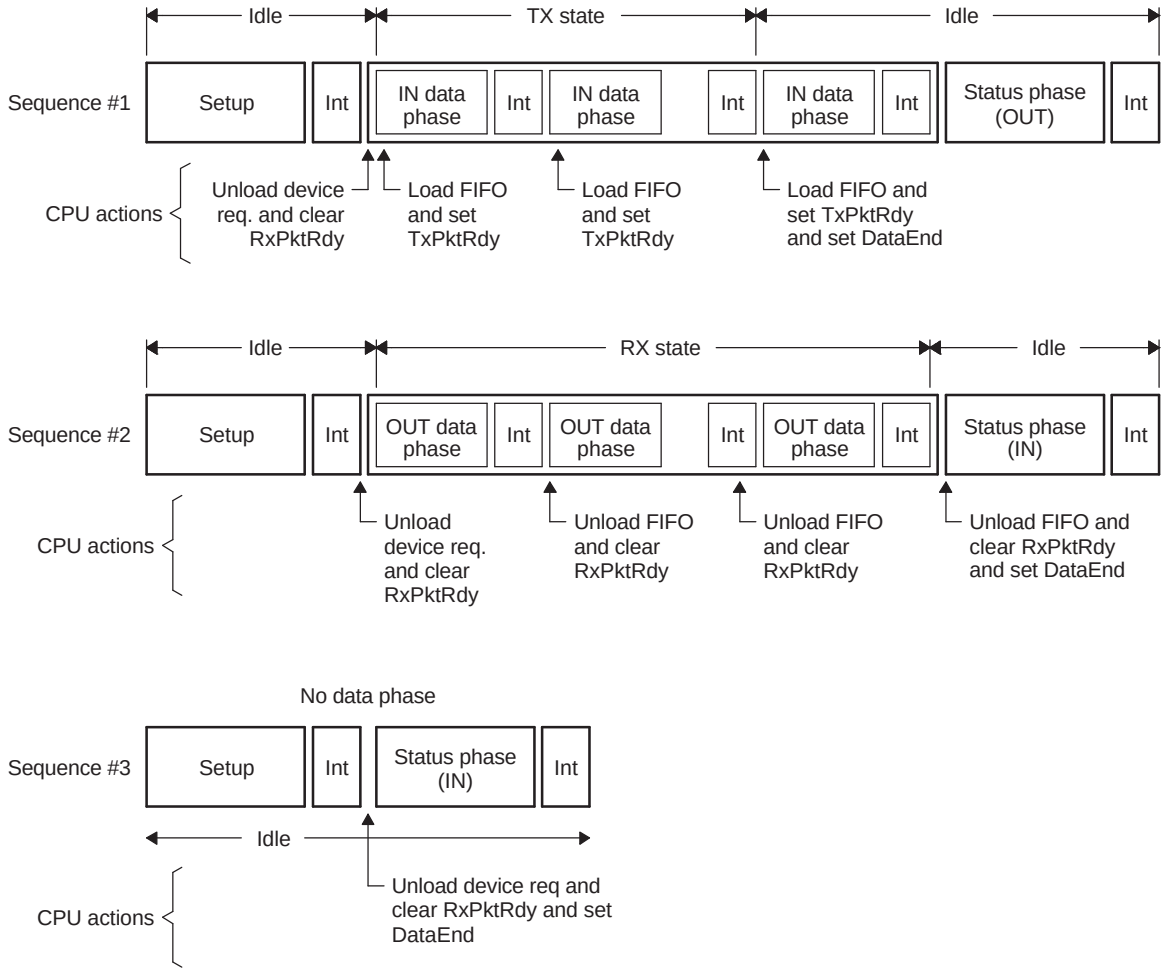


Figure 5. Sequence of Transfer



2.7.1.1.5 Endpoint 0 Service Routine

An Endpoint 0 interrupt is generated when:

- The controller sets the RXPKTRDY bit of PERI_CSR0 (bit 0) after a valid token has been received and data has been written to the FIFO.
- The controller clears the TXPKTRDY bit of PERI_CSR0 (bit 1) after the packet of data in the FIFO has been successfully transmitted to the host.
- The controller sets the SENTSTALL bit of PERI_CSR0 (bit 2) after a control transaction is ended due to a protocol violation.
- The controller sets the SETUPEND bit of PERI_CSR0 (bit 4) because a control transfer has ended before DATAEND (bit 3 of PERI_CSR0) 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 6 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 (bit 0). 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:

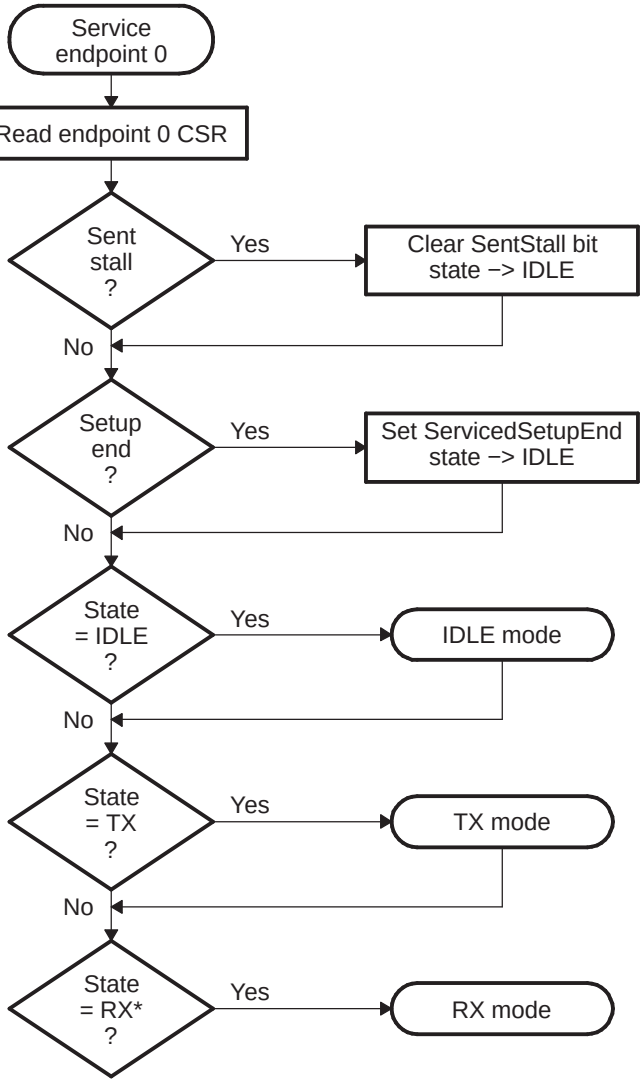
- If the command is a single packet transaction (SET_ADDRESS, SET_INTERFACE etc.) without any data phase, the endpoint will remain in IDLE state.
- If the command has an OUT data phase (SET_DESCRIPTOR etc.), the endpoint will enter RX state.
- If the command has an IN data phase (GET_DESCRIPTOR etc.), 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 (bit 6) to indicate that it has read the data in the FIFO and leave the endpoint in RX state.

Figure 6. Service Endpoint 0 Flow Chart

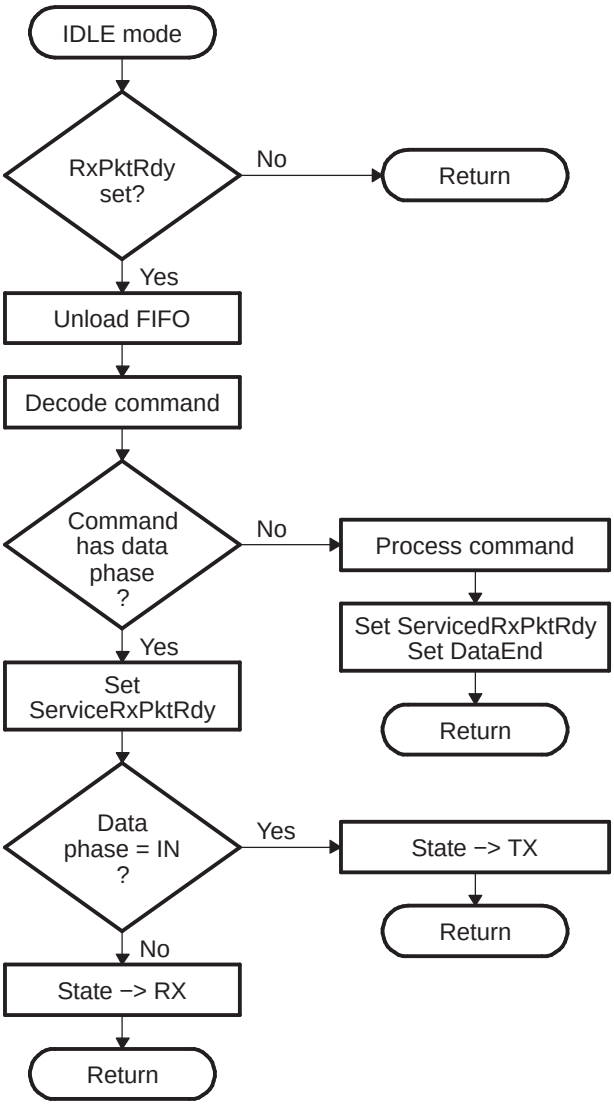


* By default

2.7.1.1.5.1 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 7).

Figure 7. IDLE Mode Flow Chart



2.7.1.1.5.2 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 8.

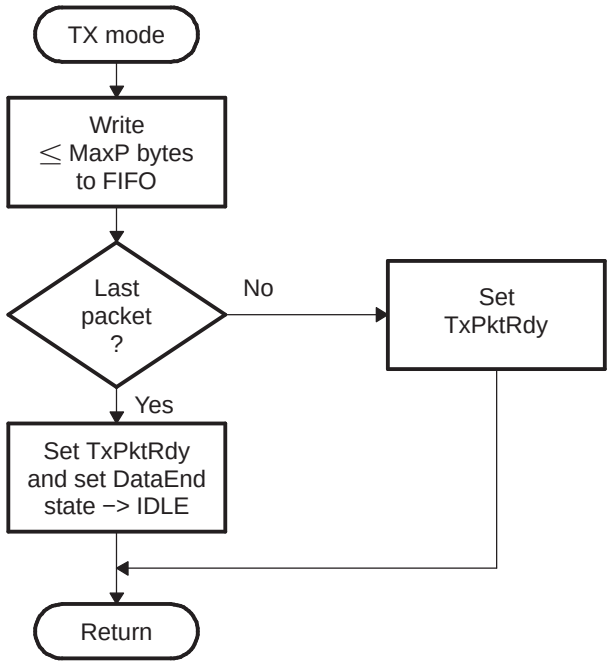
Three events can cause TX mode to be terminated before the expected amount of data has been sent:

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 8. TX Mode Flow Chart



2.7.1.1.5.3 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.

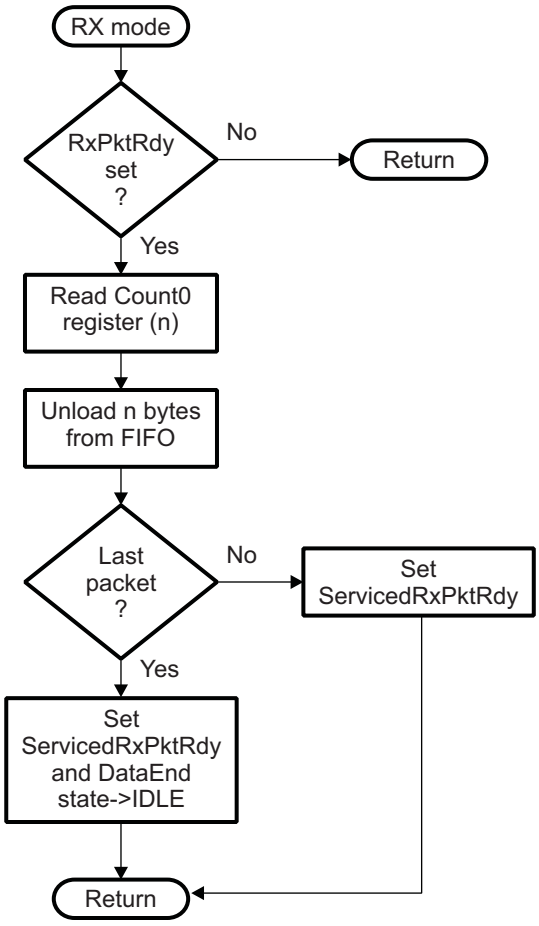
Three events can cause RX mode to be terminated before the expected amount of data has been received as shown in Figure 9:

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 9. RX Mode Flow Chart



2.7.1.1.5.4 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.

2.7.1.1.5.5 Additional Conditions

When working as a peripheral device, 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).

2.7.1.2 Bulk Transactions

2.7.1.2.1 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 used in peripheral mode 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.

2.7.1.2.1.1 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.

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.

2.7.1.2.1.2 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.

2.7.1.2.1.3 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).

2.7.1.2.2 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 used in peripheral mode 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.

2.7.1.2.2.1 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.

2.7.1.2.2.2 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.

2.7.1.2.2.3 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.

2.7.1.3 Peripheral Mode: 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.

2.7.1.4 Isochronous Transactions

2.7.1.4.1 Peripheral Mode: 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 used in Peripheral mode 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.

2.7.1.4.1.1 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 to 1 when DMA is enabled.

2.7.1.4.1.2 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.

2.7.1.4.1.3 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.

2.7.1.4.2 Peripheral Mode: 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 used in Peripheral mode 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.

2.7.1.4.2.1 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.

2.7.1.4.2.2 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.

2.7.1.4.2.3 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.

2.7.2 USB Controller Host Mode Operation

- *Entry into Suspend mode*. When operating as a host, the controller can be prompted to enter Suspend mode by setting the SUSPENDM bit in the POWER register. When this bit is set, the controller will complete the current transaction then stop the transaction scheduler and frame counter. No further transactions will be started and no SOF packets will be generated. If the ENSUSPM bit (bit 0 of POWER register) is set, PHY will go into low-power mode when the controller enters Suspend mode.
- *Sending Resume Signaling.* When the application requires the controller to leave Suspend mode, it must clear the SUSPENDM bit in the POWER register (bit 1), set the RESUME bit (bit 2) and leave it set for 20ms. While the RESUME bit is high, the controller will generate Resume signaling on the bus. After 20 ms, the application should clear the Resume bit, at which point the frame counter and transaction scheduler will be started.
- *Responding to Remote Wake-up.* If Resume signaling is detected from the target while the controller is in Suspend mode, the PHY will be brought out of low-power mode. The controller will then exit Suspend mode and automatically set the RESUME bit in the POWER register (bit 2) to take over generating the Resume signaling from the target. If the Resume interrupt is enabled, an interrupt will be generated.
- *Reset Signaling.* If the RESET bit in the POWER register (bit 3) is set while the controller is in Host mode, it will generate Reset signaling on the bus. If the HSENAB bit in the POWER register (bit 5) was set, it will also try to negotiate for high-speed operation. The software should keep the RESET bit set for at least 20 ms to ensure correct resetting of the target device. After the software has cleared the bit, the controller will start its frame counter and transaction scheduler. Whether high-speed operation is selected will be indicated by HSMODE bit of POWER register (bit 4).

2.7.2.1 Control Transactions

Host Control Transactions are conducted through Endpoint 0 and the software is required to handle all the Standard Device Requests that may be sent or received via Endpoint 0 (as described in Universal Serial Bus Specification, Revision 2.0).

As for a USB peripheral device, there are three categories of Standard Device Requests to be handled: 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).

1. Zero Data Requests consist of a SETUP command followed by an IN Status Phase
2. Write Requests consist of a SETUP command, followed by an OUT Data Phase which is in turn followed by an IN Status Phase
3. Read Requests consist of a SETUP command, followed by an IN Data Phase which is in turn followed by an OUT Status Phase

A timeout may be set to limit the length of time for which the controller will retry a transaction which is continually NAKed by the target. This limit can be between 2 and 215 frames/ microframes and is set through the HOST_NAKLIMIT0 register. The following sections describe the CPU actions required for these different types of requests by examining the steps to take in the different Control Transaction phases.

2.7.2.1.1 Setup Phase

For the SETUP Phase of a control transaction (Figure 10), the software driving the USB host device needs to:

1. Load the 8 bytes of the required Device request command into the Endpoint 0 FIFO.
2. Set SETUPPKT and TXPKTRDY (bits 3 and 1 of HOST_CSR0, respectively).

   NOTE: These bits must be set together.

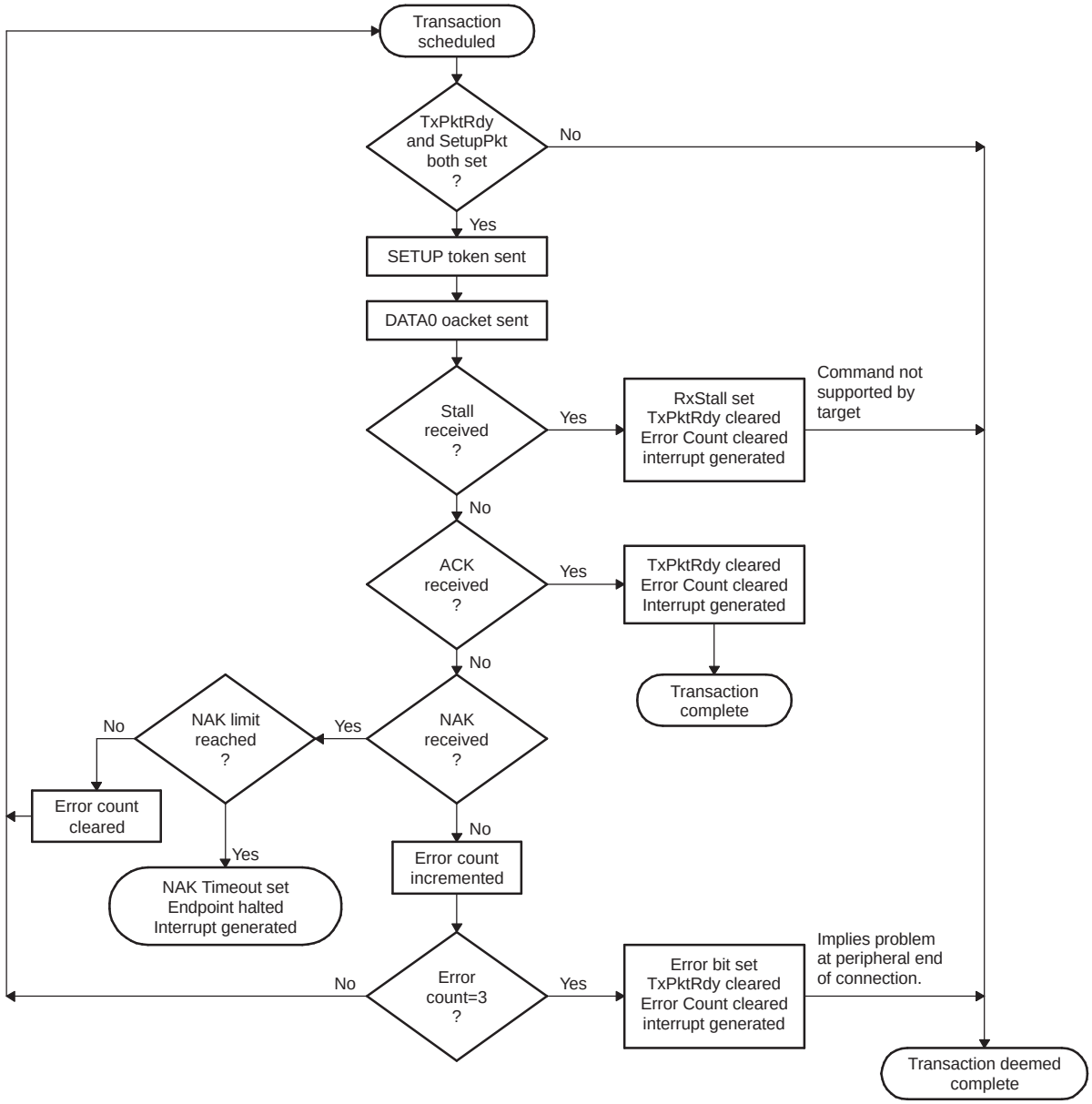
   The controller then proceeds to send a SETUP token followed by the 8-byte command to Endpoint 0 of the addressed device, retrying as necessary. (On errors, controller retries the transaction three times.)
3. At the end of the attempt to send the data, the controller will generate an Endpoint 0 interrupt. The software should then read HOST_CSR0 to establish whether the RXSTALL bit (bit 2), the ERROR bit (bit 4) or the NAK_TIMEOUT bit (bit 7) has been set.

   If RXSTALL is set, it indicates that the target did not accept the command (for example, because it is not supported by the target device) and so has issued a STALL response.

   If ERROR is set, it means that the controller has tried to send the SETUP Packet and the following data packet three times without getting any response.

   If NAK_TIMEOUT is set, it means that the controller has received a NAK response to each attempt to send the SETUP packet, for longer than the time set in HOST_NAKLIMIT0. The controller can then be directed either to continue trying this transaction (until it times out again) by clearing the NAK_TIMEOUT bit or to abort the transaction by flushing the FIFO before clearing the NAK_TIMEOUT bit.
4. If none of RXSTALL, ERROR or NAK_TIMEOUT is set, the SETUP Phase has been correctly ACKed and the software should proceed to the following IN Data Phase, OUT Data Phase or IN Status Phase specified for the particular Standard Device Request.

Figure 10. Setup Phase of a Control Transaction Flow Chart



2.7.2.1.2 IN Data Phase

For the IN Data Phase of a control transaction (Figure 11), the software driving the USB host device needs to:

1. Set REQPKT bit of HOST_CSR0 (bit 5).
2. Wait while the controller sends the IN token and receives the required data back.
3. When the controller generates the Endpoint 0 interrupt, read HOST_CSR0 to establish whether the RXSTALL bit (bit 2), the ERROR bit (bit 4), the NAK_TIMEOUT bit (bit 7) or RXPKTRDY bit (bit 0) has been set.

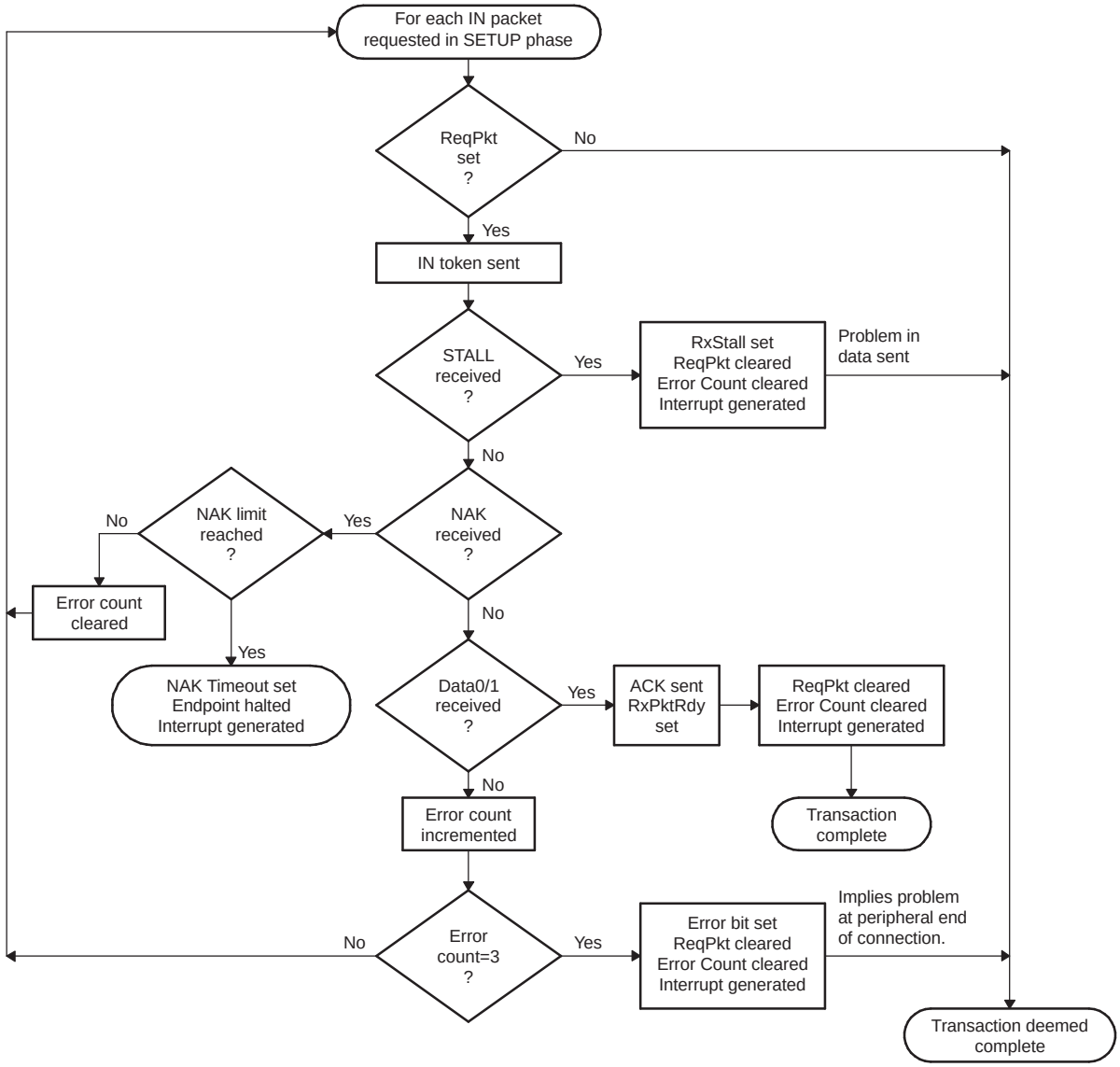
   If RXSTALL is set, it indicates that the target has issued a STALL response.

   If ERROR is set, it means that the controller has tried to send the required IN token three times without getting any response.

   If NAK_TIMEOUT bit is set, it means that the controller has received a NAK response to each attempt to send the IN token, for longer than the time set in HOST_NAKLIMIT0. The controller can then be directed either to continue trying this transaction (until it times out again) by clearing the NAK_TIMEOUT bit or to abort the transaction by clearing REQPKT before clearing the NAK_TIMEOUT bit.
4. If RXPKTRDY has been set, the software should read the data from the Endpoint 0 FIFO, then clear RXPKTRDY.
5. If further data is expected, the software should repeat Steps 1-4.

When all the data has been successfully received, the CPU should proceed to the OUT Status Phase of the Control Transaction.

Figure 11. IN Data Phase Flow Chart



2.7.2.1.3 OUT Data Phase

For the OUT Data Phase of a control transaction (Figure 12), the software driving the USB host device needs to:

1. Load the data to be sent into the endpoint 0 FIFO.
2. Set the TXPKTRDY bit of HOST_CSR0 (bit 1). The controller then proceeds to send an OUT token followed by the data from the FIFO to Endpoint 0 of the addressed device, retrying as necessary.
3. At the end of the attempt to send the data, the controller will generate an Endpoint 0 interrupt. The software should then read HOST_CSR0 to establish whether the RXSTALL bit (bit 2), the ERROR bit (bit 4) or the NAK_TIMEOUT bit (bit 7) has been set.

   If RXSTALL bit is set, it indicates that the target has issued a STALL response.

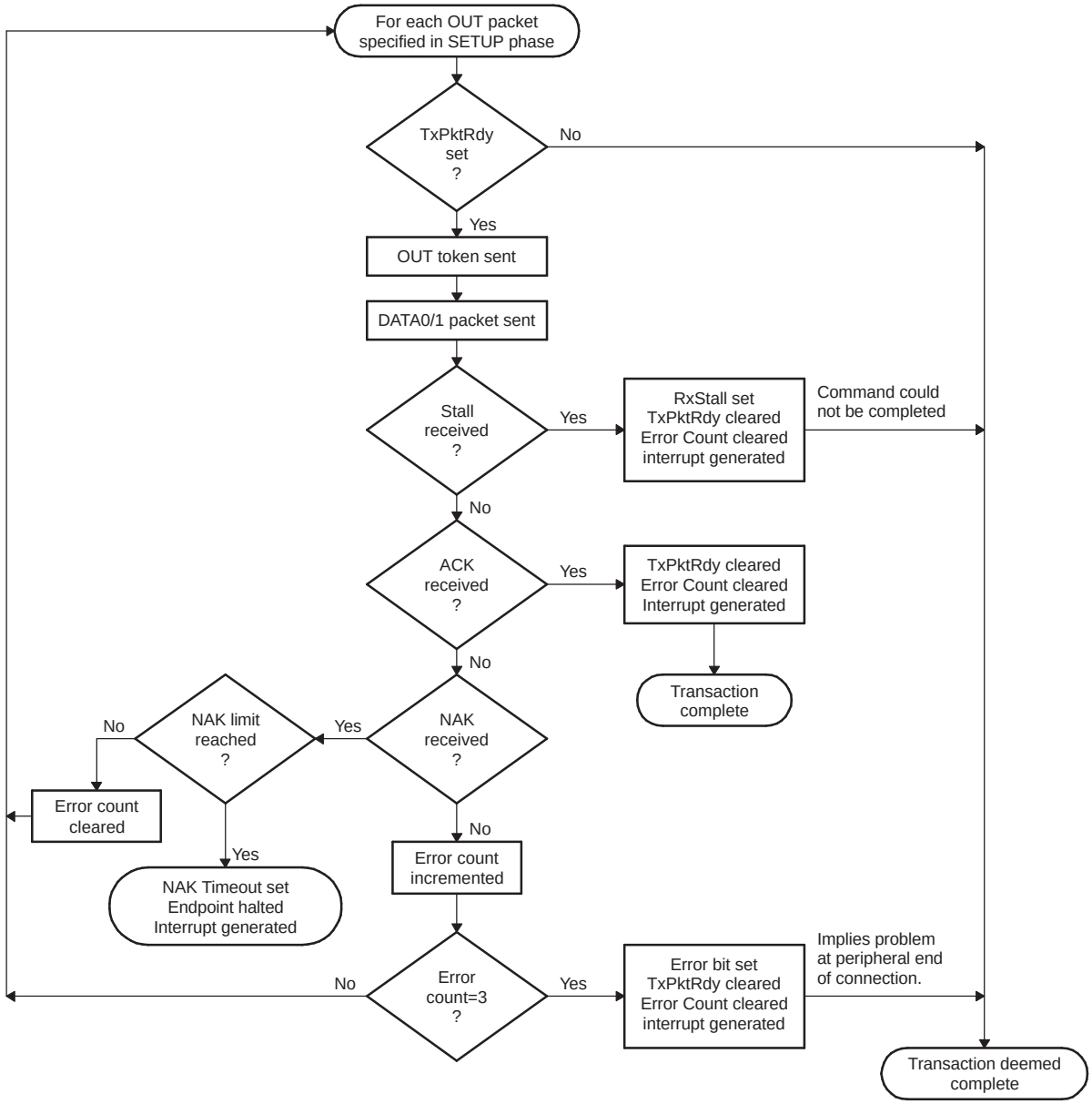
   If ERROR bit is set, it means that the controller has tried to send the OUT token and the following data packet three times without getting any response.

   If NAK_TIMEOUT is set, it means that the controller has received a NAK response to each attempt to send the OUT token, for longer than the time set in the HOST_NAKLIMIT0 register. The controller can then be directed either to continue trying this transaction (until it times out again) by clearing the NAK_TIMEOUT bit or to abort the transaction by flushing the FIFO before clearing the NAK_TIMEOUT bit.

   If none of RXSTALL, ERROR or NAKLIMIT is set, the OUT data has been correctly ACKed.
4. If further data needs to be sent, the software should repeat Steps 1-3.

   When all the data has been successfully sent, the software should proceed to the IN Status Phase of the Control Transaction.

Figure 12. OUT Data Phase Flow Chart



2.7.2.1.4 IN Status Phase (following SETUP Phase or OUT Data Phase)

For the IN Status Phase of a control transaction (Figure 13), the software driving the USB Host device needs to:

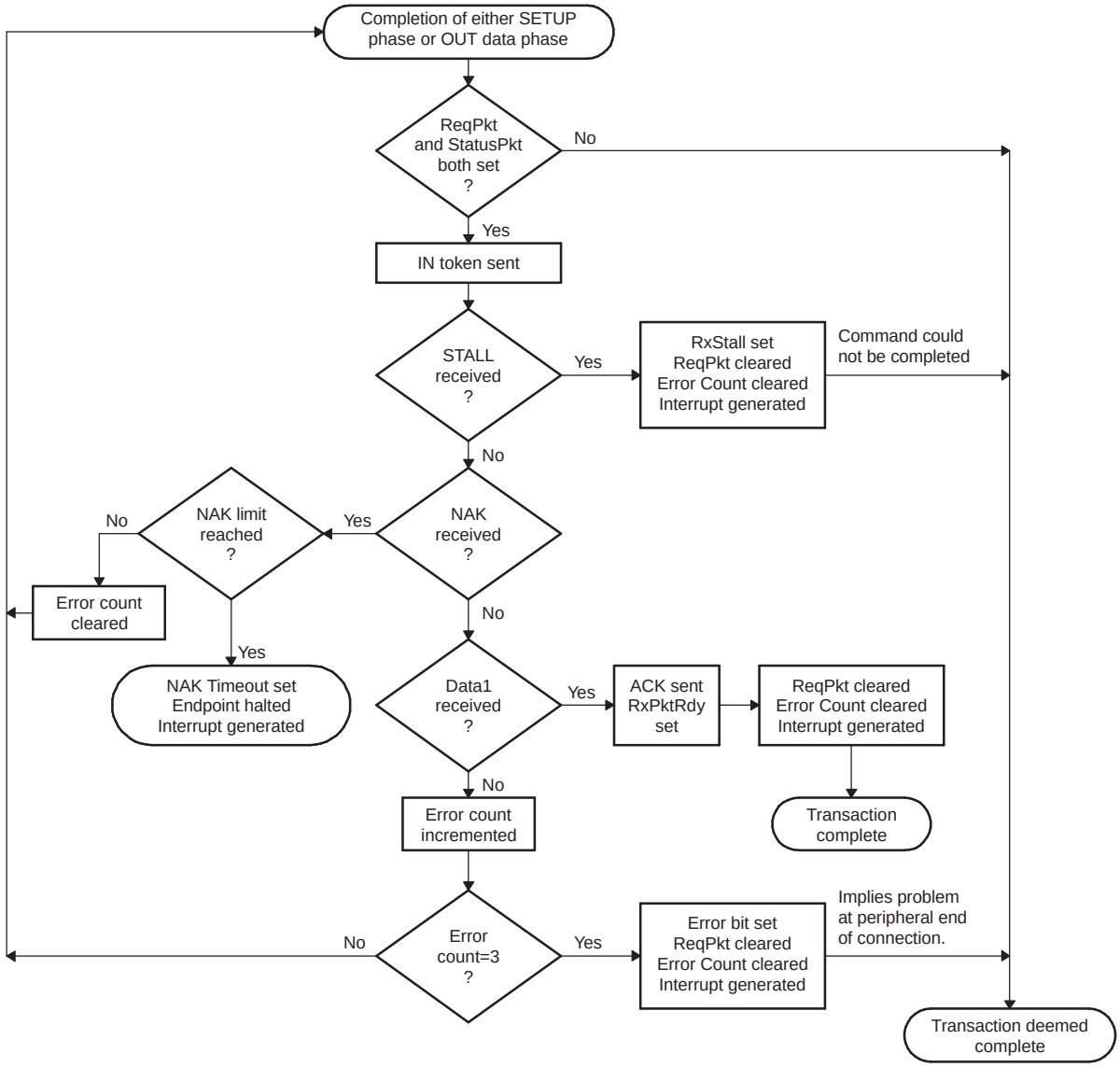
1. Set the STATUSPKT and REQPKT bits of HOST_CSR0 (bit 6 and bit 5, respectively).
2. Wait while the controller sends an IN token and receives a response from the USB peripheral device.
3. When the controller generates the Endpoint 0 interrupt, read HOST_CSR0 to establish whether the RXSTALL bit (bit 2), the ERROR bit (bit 4), the NAK_TIMEOUT bit (bit 7) or RXPKTRDY bit (bit 0) has been set.

   If RXSTALL bit is set, it indicates that the target could not complete the command and so has issued a STALL response.

   If ERROR bit is set, it means that the controller has tried to send the required IN token three times without getting any response.

   If NAK_TIMEOUT bit is set, it means that the controller has received a NAK response to each attempt to send the IN token, for longer than the time set in the HOST_NAKLIMIT0 register. The controller can then be directed either to continue trying this transaction (until it times out again) by clearing the NAK_TIMEOUT bit or to abort the transaction by clearing REQPKT bit and STATUSPKT bit before clearing the NAK_TIMEOUT bit.
4. If RxPktRdy has been set, the CPU should simply clear RxPktRdy.

Figure 13. Completion of SETUP or OUT Data Phase Flow Chart



2.7.2.1.5 OUT Status Phase (following IN Data Phase)

For the OUT Status Phase of a control transaction (Figure 14) , the CPU driving the host device needs to:

1. Set STATUSPKT and TXPKTRDY bits of HOST_CSR0 (bit 6 and bit 1, respectively).

   NOTE: These bits need to be set together.

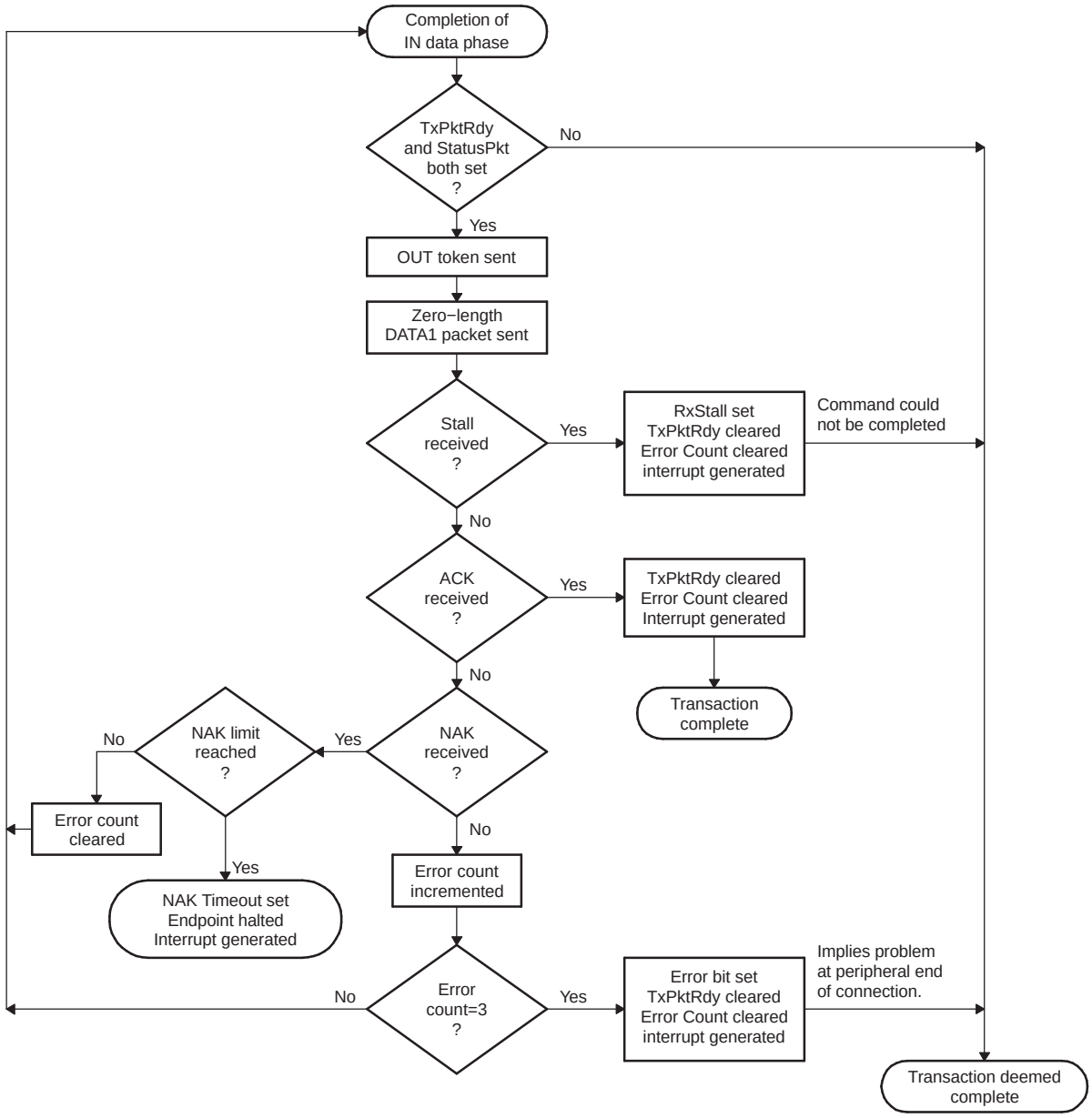
2. Wait while the controller sends the OUT token and a zero-length DATA1 packet.
3. At the end of the attempt to send the data, the controller will generate an Endpoint 0 interrupt. The software should then read HOST_CSR0 to establish whether the RXSTALL bit (bit 2), the ERROR bit (bit 4) or the NAK_TIMEOUT bit (bit 7) has been set.

   If RXSTALL bit is set, it indicates that the target could not complete the command and so has issued a STALL response.

   If ERROR bit is set, it means that the controller has tried to send the STATUS Packet and the following data packet three times without getting any response.

   If NAK_TIMEOUT bit is set, it means that the controller has received a NAK response to each attempt to send the IN token, for longer than the time set in the HOST_NAKLIMIT0 register. The controller can then be directed either to continue trying this transaction (until it times out again) by clearing the NAK_TIMEOUT bit or to abort the transaction by flushing the FIFO before clearing the NAK_TIMEOUT bit.
4. If none of RXSTALL, ERROR or NAK_TIMEOUT bits is set, the STATUS Phase has been correctly ACKed.

Figure 14. Completion of IN Data Phase Flow Chart



2.7.2.2 Bulk Transactions

2.7.2.2.1 Bulk IN Transactions

A Bulk IN transaction may be used to transfer non-periodic data from the external USB peripheral to the host.

The following optional features are available for use with an Rx endpoint used in host mode to receive the data:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO on reception from the host. This allows that one packet can be received while another is being read. Double packet buffering is enabled by setting the DPB bit of 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.
- AutoRequest: When the AutoRequest feature is enabled, the REQPKT bit of HOST_RXCSR (bit 5) will be automatically set when the RXPKTRDY bit is cleared.

  This feature is applicable only when DMA is enabled. To enable AutoRequest feature, set the AUTOREQ register for the DMA channel associated for the endpoint.

2.7.2.2.1.1 Setup

Before initiating any Bulk IN Transactions in Host mode:

- The target function address needs to be set in the RXFUNCADDR register for the selected controller endpoint. (RXFUNCADDR register is available for all endpoints from EP0 to EP4.)
- The HOST_RXTYPE register for the endpoint that is to be used needs to be programmed as:
  - Operating speed in the SPEED bit field (bits 7 and 6).
  - Set 10 (binary value) in the PROT field for bulk transfer.
  - Endpoint Number of the target device in RENDPN field. This is the endpoint number contained in the Rx endpoint descriptor returned by the target device during enumeration.
- The RXMAXP register for the controller endpoint must be written with the maximum packet size (in bytes) for the transfer. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the target endpoint.
- The HOST_RXINTERVAL register needs to be written with the required value for the NAK limit (2-215 frames/microframes), or set to zero if the NAK timeout feature is not required.
- The relevant interrupt enable bit in the INTRRXE register should be set (if an interrupt is required for this endpoint).
- The following bits of HOST_RXCSR register should be set as:
  - Set DMAEN (bit 13) to 1 if a DMA request is required for this endpoint.
  - Clear DSINYET (bit 12) to 0 to allow normal PING flow control. This will affect only High Speed transactions.
  - Always clear DMAMODE (bit 11) to 0.
- If DMA is enabled, the AUTOREQ register can be set for generating IN tokens automatically after receiving the data. Set the bit field RXn_AUTOREQ (where n is the endpoint number) with binary value 01 or 11.

When the endpoint is first configured, the endpoint data toggle should be cleared to 0 either by using the DATATOGWREN and DATATOG bits of HOST_RXCSR (bit 10 and bit 9) to toggle the current setting or by setting the CLRDATATOG bit of HOST_RXCSR (bit 7). 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 RXPKTRDY bit (bit 0 of HOST_RXCSR) being set), they should be flushed by setting the FLUSHFIFO bit of HOST_RXCSR (bit 4).

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

2.7.2.2.1.2 Operation

When Bulk data is required from the USB peripheral device, the software should set the REQPKT bit in the corresponding HOST_RXCSR register (bit 5). The controller will then send an IN token to the selected peripheral endpoint and waits for data to be returned.

If data is correctly received, RXPKTRDY bit of HOST_RXCSR (bit 0) is set. If the USB peripheral device responds with a STALL, RXSTALL bit (bit 6 of HOST_RXCSR) is set. If a NAK is received, the controller tries again and continues to try until either the transaction is successful or the POLINTVL_NAKLIMIT set in the HOST_RXINTERVAL register is reached. If no response at all is received, two further attempts are made before the controller reports an error by setting the ERROR bit of HOST_RXCSR (bit 2).

The controller then generates the appropriate endpoint interrupt, whereupon the software should read the corresponding HOST_RXCSR register to determine whether the RXPKTRDY, RXSTALL, ERROR or DATAERR_NAKTIMEOUT bit is set and act accordingly. If the DATAERR_NAKTIMEOUT bit is set, the controller can be directed either to continue trying this transaction (until it times out again) by clearing the DATAERR_NAKTIMEOUT bit or to abort the transaction by clearing REQPKT bit before clearing the DATAERR_NAKTIMEOUT bit.

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).

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.

2.7.2.2.1.3 Error Handling

If the target wants to shut down the Bulk IN pipe, it will send a STALL response to the IN token. This will result in the RXSTALL bit of HOST_RXCSR (bit 6) being set.

2.7.2.2.2 Host Mode: Bulk OUT Transactions

A Bulk OUT transaction may be used to transfer non-periodic data from the host to the USB peripheral.

Following optional features are available for use with a Tx endpoint used in Host mode to transmit this data:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO awaiting transmission to the peripheral device. 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 can be used to allow the DMA controller to load packets into the FIFO without processor intervention.

  When DMA is enabled and DMAMODE bit in HOST_TXCSR register is set, an endpoint interrupt will not be generated for completion of packet reception. An endpoint interrupt will be generated only in the error conditions.

2.7.2.2.2.1 Setup

Before initiating any bulk OUT transactions:

- The target function address needs to be set in the TXFUNCADDR register for the selected controller endpoint. (TXFUNCADDR register is available for all endpoints from EP0 to EP4.)
- The HOST_TXTYPE register for the endpoint that is to be used needs to be programmed as:
  - Operating speed in the SPEED bit field (bits 7 and 6).
  - Set 10b in the PROT field for bulk transfer.
  - Endpoint Number of the target device in TENDPN field. This is the endpoint number contained in the OUT(Tx) endpoint descriptor returned by the target device during enumeration.
- The TXMAXP register for the controller endpoint must be written with the maximum packet size (in bytes) for the transfer. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the target endpoint.

- The HOST_TXINTERVAL register needs to be written with the required value for the NAK limit (2-215 frames/microframes), or set to zero if the NAK timeout feature is not required.
- The relevant interrupt enable bit in the INTRTXE register should be set (if an interrupt is required for this endpoint).
- The following bits of HOST_TXCSR register should be set as:
  - Set the MODE bit (bit 13) to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).
  - Set the DMAEN bit (bit 12) to 1 if a DMA request is required for this endpoint.
  - Clear the FRCDATATOG bit (bit 11) to 0 to allow normal data toggle operations.
  - Set the DMAMODE bit (bit 10) to 1 when DMA is enabled.

When the endpoint is first configured, the endpoint data toggle should be cleared to 0 either by using the DATATOGWREN bit and DATATOG bit of HOST_TXCSR (bit 9 and bit 8) to toggle the current setting or by setting the CLRDATATOG bit of HOST_TXCSR (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 of HOST_TXCSR register (bit 1) being set), they should be flushed by setting the FLUSHFIFO bit (bit 3 of HOST_TXCSR).

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

2.7.2.2.2.2 Operation

When Bulk data is required to be sent to the USB peripheral device, the software should write the first packet of the data to the FIFO (or two packets if double-buffered) and set the TXPKTRDY bit in the corresponding HOST_TXCSR register (bit 0). The controller will then send an OUT token to the selected peripheral endpoint, followed by the first data packet from the FIFO.

If data is correctly received by the peripheral device, an ACK should be received whereupon the controller will clear TXPKTRDY bit of HOST_TXCSR (bit 0). If the USB peripheral device responds with a STALL, the RXSTALL bit (bit 5) of HOST_TXCSR is set. If a NAK is received, the controller tries again and continues to try until either the transaction is successful or the NAK limit set in the HOST_TXINTERVAL register is reached. If no response at all is received, two further attempts are made before the controller reports an error by setting ERROR bit in HOST_TXCSR (bit 2).

The controller then generates the appropriate endpoint interrupt, whereupon the software should read the corresponding HOST_TXCSR register to determine whether the RXSTALL (bit 5), ERROR (bit 2) or NAK_TIMEOUT (bit 7) bit is set and act accordingly. If the NAK_TIMEOUT bit is set, the controller can be directed either to continue trying this transaction (until it times out again) by clearing the NAK_TIMEOUT bit or to abort the transaction by flushing the FIFO before clearing the NAK_TIMEOUT bit.

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.

2.7.2.2.2.3 Error Handling

If the target wants to shut down the Bulk OUT pipe, it will send a STALL response. This is indicated by the RXSTALL bit of HOST_TXCSR register (bit 5) being set.

2.7.2.3 Interrupt Transactions

When the controller is operating as the host, interactions with an Interrupt endpoint on the USB peripheral device are handled in very much the same way as the equivalent Bulk transactions (described in previous sections).

The principal difference as far as operational steps are concerned is that the PROT field of HOST_RXTYPE and HOST_TXTYPE (bits 5-4) need to be set (binary value) to represent an Interrupt transaction. The required polling interval also needs to be set in the HOST_RXINTERVAL and HOST_TXINTERVAL registers.

2.7.2.4 Isochronous Transactions

2.7.2.4.1 Isochronous IN Transactions

An Isochronous IN transaction is used to transfer periodic data from the USB peripheral to the host.

The following optional features are available for use with an Rx endpoint used in Host mode to receive this data:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO on reception from the host. This allows that one packet can be received while another is being read. Double packet buffering is enabled by setting the DPB bit of 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. However, this feature is not particularly useful with isochronous endpoints because the packets transferred are often not maximum packet size.

  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.
- AutoRequest: When the AutoRequest feature is enabled, the REQPKT bit of HOST_RXCSR (bit 5) will be automatically set when the RXPKTRDY bit is cleared.

  This feature is applicable only when DMA is enabled. To enable AutoRequest feature, set the AUTOREQ register for the DMA channel associated for the endpoint.

2.7.2.4.1.1 Setup

Before initiating an Isochronous IN Transactions in Host mode:

- The target function address needs to be set in the RXFUNCADDR register for the selected controller endpoint (RXFUNCADDR register is available for all endpoints from EP0 to EP4).
- The HOST_RXTYPE register for the endpoint that is to be used needs to be programmed as:
  - Operating speed in the SPEED bit field (bits 7 and 6).
  - Set 01 (binary value) in the PROT field for isochronous transfer.
  - Endpoint Number of the target device in RENDPN field. This is the endpoint number contained in the Rx endpoint descriptor returned by the target device during enumeration.
- The RXMAXP register for the controller endpoint must be written with the maximum packet size (in bytes) for the transfer. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the target endpoint.
- The HOST_RXINTERVAL register needs to be written with the required transaction interval (usually one transaction per frame/microframe).
- The relevant interrupt enable bit in the INTRRXE register should be set (if an interrupt is required for this endpoint).
- The following bits of HOST_RXCSR register should be set as:
  - Set the DMAEN bit (bit 13) to 1 if a DMA request is required for this endpoint.
  - Clear the DISNYET it (bit 12) to 0 to allow normal PING flow control. This will only affect High Speed transactions.
  - Always clear the DMAMODE bit (bit 11) to 0.
- If DMA is enabled, AUTOREQ register can be set for generating IN tokens automatically after receiving the data. Set the bit field RX*n*_AUTOREQ (where *n* is the endpoint number) with binary value 01 or 11.

2.7.2.4.1.2 Operation

The operation starts with the software setting REQPKT bit of HOST_RXCSR (bit 5). This causes the controller to send an IN token to the target.

When a packet is received, an interrupt is generated which the software may use to unload the packet from the FIFO and clear the RXPKTRDY bit in the HOST_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), 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 before unloading the FIFO. This can be done by using the SOF_PULSE signal from the controller to trigger the unloading of the data packet. The SOF_PULSE is generated once per frame(/microframe). The interrupts may still be used to clear the RXPKTRDY bit in HOST_RXCSR.

2.7.2.4.1.3 Error Handling

If a CRC or bit-stuff error occurs during the reception of a packet, the packet will still be stored in the FIFO but the DATAERR_NAKTIMEOUT bit of HOST_RXCSR (bit 3) is set to indicate that the data may be corrupt.

2.7.2.4.2 Isochronous OUT Transactions

An Isochronous OUT transaction may be used to transfer periodic data from the host to the USB peripheral.

Following optional features are available for use with a Tx endpoint used in Host mode to transmit this data:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO awaiting transmission to the peripheral device. 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 can be used to allow 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.

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

2.7.2.4.2.1 Setup

Before initiating any Isochronous OUT transactions:

- The target function address needs to be set in the TXFUNCADDR register for the selected controller endpoint (TXFUNCADDR register is available for all endpoints from EP0 to EP4).
- The HOST_TXTYPE register for the endpoint that is to be used needs to be programmed as:
  - Operating speed in the SPEED bit field (bits 7 and 6).
  - Set 01 (binary value) in the PROT field for isochronous transfer.
  - Endpoint Number of the target device in TENDPN field. This is the endpoint number contained in the OUT(Tx) endpoint descriptor returned by the target device during enumeration.
- The TXMAXP register for the controller endpoint must be written with the maximum packet size (in bytes) for the transfer. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the target endpoint.
- The HOST_TXINTERVAL register needs to be written with the required transaction interval (usually one transaction per frame/microframe).
- The relevant interrupt enable bit in the INTRTXE register should be set (if an interrupt is required for this endpoint).
- The following bits of HOST_TXCSR register should be set as:
  - Set the MODE bit (bit 13) to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).
  - Set the DMAEN bit (bit 12) to 1 if a DMA request is required for this endpoint.
  - The FRCDATATOG bit (bit 12) is ignored for isochronous transactions.
  - Set the DMAMODE bit (bit 10) to 1 when DMA is enabled.

2.7.2.4.2.2 Operation

The operation starts when the software writes to the FIFO and sets TXPKTRDY bit of HOST_TXCSR (bit 0). This triggers the controller to send an OUT token followed by the first data packet from the FIFO.

An interrupt is generated whenever a packet is sent and the software may use this interrupt to load the next packet into the FIFO and set the TXPKTRDY bit in the HOST_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, 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 before loading the FIFO as this will minimize the requirement for additional buffering. This can be done by using the SOF_PULSE signal from the controller to trigger the loading of the next data packet. The SOF_PULSE is generated once per frame(/microframe). The interrupts may still be used to set the TXPKTRDY bit in HOST_TXCSR.

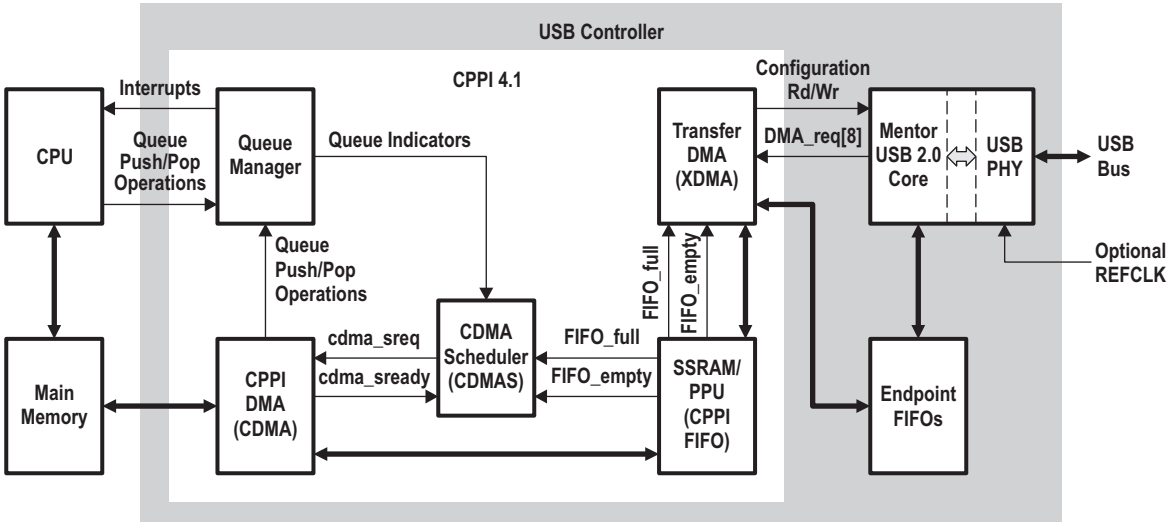
2.8 Communications Port Programming Interface (CPPI) 4.1 DMA Overview

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 15 displays the USB controller block diagram.

Figure 15. 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).

2.8.1 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 32-bit memory space. 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.

2.8.2 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.

The Host Packet Descriptor layout is shown in Figure 16 and described in Table 8 to Table 15.

Figure 16. Host Packet Descriptor Layout

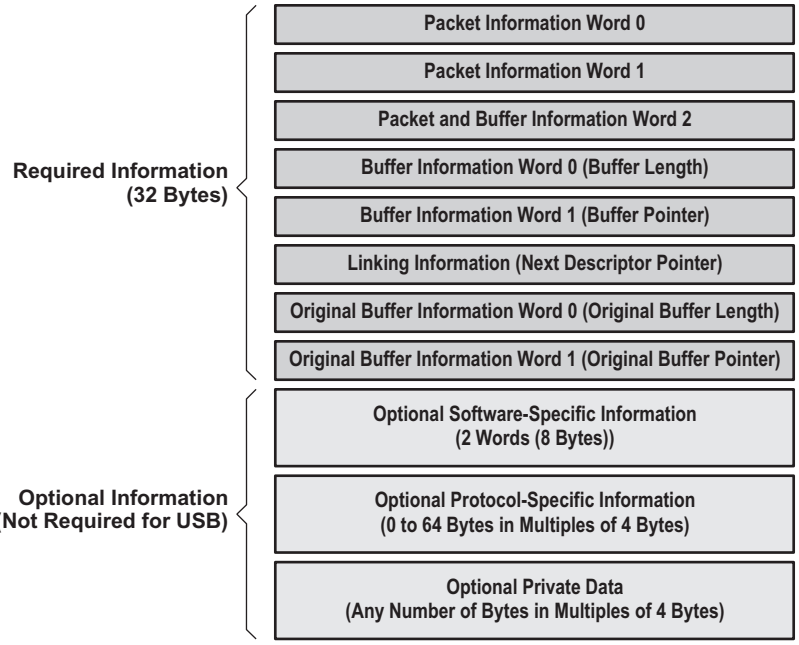


Table 8. Host Packet Descriptor Word 0 (HPD Word 0)

Bits	Name	Description
31-27	Descriptor Type	The Host Packet Descriptor Type is 16 decimal (10h). The CPU initializes this field.
26-22	Protocol Specific Valid Word Count	This field indicates the valid numberof 32-bit words in the protocol specific region. The CPU initializes this field. This is encoded in increments of 4 bytes as: 0 = 0 byte 1 = 4 bytes ... 16 = 64 bytes 17-31 = Reserved
21-0	Packet Length	The length of the packet in bytes. If the Packet Length is less than the sum of the buffer lengths, then the packet data will be truncated. A Packet Length greater than the sum of the buffers is an error. The valid range for the packet length is 0 to (4M - 1) bytes. The CPU initializes this field for transmitted packets; the DMA overwrites this field on packet reception.

Table 9. Host Packet Descriptor Word 1 (HPD Word 1)

Bits	Name	Description
31-27	Source Tag: Port #	This field indicates the port number (0-31) from which the packet originated. The DMA overwrites this field on packet reception. This is the RX Endpoint number from which the packet originated.
26-21	Source Tag: Channel #	This field indicates the channel number within the port from which the packet originated. The DMA overwrites this field on packet reception. This field is always 0-67.
20-16	Source Tag: Sub-channel #	This field indicates the sub-channel number (0-31) within the channel from which the packet originated. The DMA overwrites this field on packet reception. This field is always 0.
15-0	Destination Tag	This field is application specific. This field is always 0.

Table 10. Host Packet Descriptor Word 2 (HPD Word 2)

Bits	Name	Description
31	Packet Error	This bit indicates if an error occurred during reception of this packet (0 = No error occurred, 1 = Error occurred). The DMA overwrites this field on packet reception. Additional information about different errors may be encoded in the protocol specific fields in the descriptor.
30-26	Packet Type	This field indicates the type of this packet. The CPU initializes this field for transmitted packets; the DMA overwrites this field on packet reception. This field is encoded as: 5 = USB 8-31 = Reserved
25-20	Reserved	Reserved
19	Zero-length packet indicator	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 is 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.
18-16	Protocol Specific	This field contains protocol specific flags/information that can be assigned based on the packet type. Not used for USB.
15	Return Policy	This field indicates the return policy for this packet. The CPU initializes this field. 0 = Entire packet (still linked together) should be returned to the queue specified in bits 13-0. 1 = Each buffer should be returned to the queue specified in bits 13-0 of Word 2 in their respective descriptors. The Tx DMA will return each buffer in sequence.
14	On-chip	This field indicates whether or not this descriptor is in a region which is in on-chip memory space (1) or in external memory (0).
13-12	Packet Return Queue Mgr #	This field indicates which queue manager in the system the descriptor is to be returned to after transmission is complete. This field is not altered by the DMA during transmission or reception and is initialized by the CPU. There is only 1 Queue Manager in the USB HS/FS Device Controller, this field must always be 0.
11-0	Packet Return Queue #	This field indicates the queue number within the selected queue manager that the descriptor is to be returned to after transmission is complete. This field is not altered by the DMA during transmission or reception and is initialized by the CPU.

Table 11. Host Packet Descriptor Word 3 (HPD Word 3)

Bits	Name	Description
31-22	Reserved	Reserved
21-0	Buffer 0 Length	The Buffer Length field indicates how many valid data bytes are in the buffer. The CPU initializes this field for transmitted packets; the DMA overwrites this field on packet reception.

Table 12. Host Packet Descriptor Word 4 (HPD Word 4)

Bits	Name	Description
31-0	Buffer 0 Pointer	The Buffer Pointer is the byte aligned memory address of the buffer associated with the buffer descriptor. The CPU initializes this field for transmitted packets; the DMA overwrites this field on packet reception.

Table 13. Host Packet Descriptor Word 5 (HPD Word 5)

Bits	Name	Description
31-0	Next Descriptor Pointer	The 32-bit word aligned memory address of the next buffer descriptor in the packet. If the value of this pointer is zero, then the current buffer is the last buffer in the packet. The CPU initializes this field for transmitted packets; the DMA overwrites this field on packet reception.

Table 14. Host Packet Descriptor Word 6 (HPD Word 6)

Bits	Name	Description
31-22	Reserved	Reserved
21-0	Original Buffer 0 Length	The Buffer Length field indicates the original size of the buffer in bytes. This value is not overwritten during reception. This value is read by the Rx DMA to determine the actual buffer size as allocated by the CPU at initialization. Since the buffer length in Word 3 is overwritten by the Rx port during reception, this field is necessary to permanently store the buffer size information.

Table 15. Host Packet Descriptor Word 7 (HPD Word 7)

Bits	Name	Description
31-22	Reserved	Reserved
21-0	Original Buffer 0 Pointer	The Buffer Pointer is the byte aligned memory address of the buffer associated with the buffer descriptor. This value is not overwritten during reception. This value is read by the Rx DMA to determine the actual buffer location as allocated by the CPU at initialization. Since the buffer pointer in Word 4 is overwritten by the Rx port during reception, this field is necessary to permanently store the buffer pointer information.

2.8.3 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 4.87).

The Host Buffer Descriptor layout is shown in Figure 17 and described in Table 16 to Table 23.

Figure 17. Host Buffer Descriptor Layout

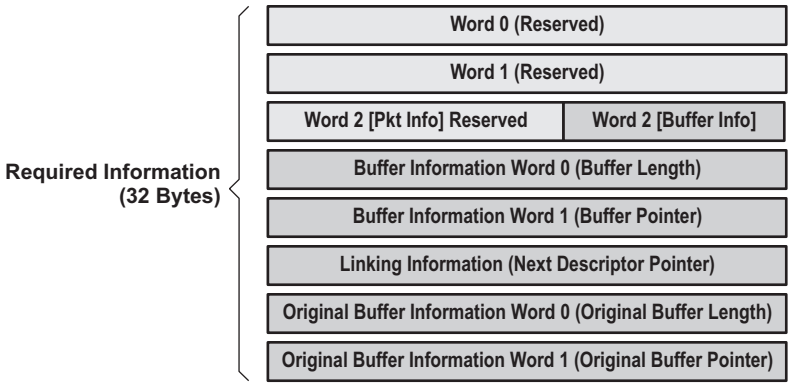


Table 16. Host Buffer Descriptor Word 0 (HBD Word 0)

Bits	Name	Description
31-0	Reserved	Reserved

Table 17. Host Buffer Descriptor Word 1 (HBD Word 1)

Bits	Name	Description
31-0	Reserved	Reserved

Table 18. Host Buffer Descriptor Word 2 (HBD Word 2)

Bits	Name	Description
31-15	Reserved	Reserved
14	On-chip	This field indicates whether or not this descriptor is in a region which is in on-chip memory space (1) or in external memory (0).
13-12	Packet Return Queue Mgr #	This field indicates which queue manager in the system the descriptor is to be returned to after transmission is complete. This field is not altered by the DMA during transmission or reception and is initialized by the CPU. There is only 1 Queue Manager in the USB HS/FS Device Controller, this field must always be 0.
11-0	Packet Return Queue #	This field indicates the queue number within the selected queue manager that the descriptor is to be returned to after transmission is complete. This field is not altered by the DMA during transmission or reception and is initialized by the CPU.

Table 19. Host Buffer Descriptor Word 3 (HBD Word 3)

Bits	Name	Description
31-22	Reserved	Reserved
21-0	Buffer 0 Length	The Buffer Length field indicates how many valid data bytes are in the buffer. The CPU initializes this field for transmitted packets; the DMA overwrites this field on packet reception.

Table 20. Host Buffer Descriptor Word 4 (HBD Word 4)

Bits	Name	Description
31-0	Buffer 0 Pointer	The Buffer Pointer is the byte aligned memory address of the buffer associated with the buffer descriptor. The CPU initializes this field for transmitted packets; the DMA overwrites this field on packet reception.

Table 21. Host Buffer Descriptor Word 5 (HBD Word 5)

Bits	Name	Description
31-0	Next Descriptor Pointer	The 32-bit word aligned memory address of the next buffer descriptor in the packet. If the value of this pointer is zero, then the current descriptor is the last descriptor in the packet. The CPU initializes this field for transmitted packets; the DMA overwrites this field on packet reception.

Table 22. Host Buffer Descriptor Word 6 (HBD Word 6)

Bits	Name	Description
31-22	Reserved	Reserved
21-0	Original Buffer 0 Length	The Buffer Length field indicates the original size of the buffer in bytes. This value is not overwritten during reception. This value is read by the Rx DMA to determine the actual buffer size as allocated by the CPU at initialization. Since the buffer length in Word 3 is overwritten by the Rx port during reception, this field is necessary to permanently store the buffer size information.

Table 23. Host Buffer Descriptor Word 7 (HBD Word 7)

Bits	Name	Description
31-0	Original Buffer 0 Pointer	The Buffer Pointer is the byte aligned memory address of the buffer associated with the buffer descriptor. This value is not overwritten during reception. This value is read by the Rx DMA to determine the actual buffer location as allocated by the CPU at initialization. Since the buffer pointer in Word 4 is overwritten by the Rx port during reception, this field is necessary to permanently store the buffer pointer information.

2.8.4 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. The Teardown Descriptor layout is shown in Figure 18 and described in Table 24 and Table 25. 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.

The 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).

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 18. Teardown Descriptor Layout

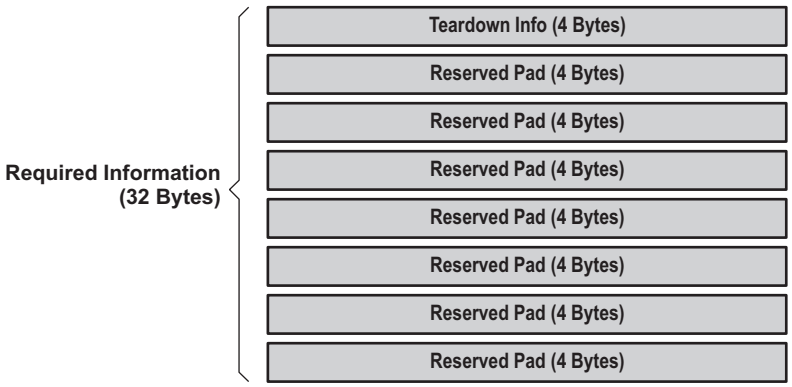


Table 24. Teardown Descriptor Word 0

Bits	Name	Description
31-27	Descriptor Type	The teardown descriptor type is 19 decimal (13h).
26-17	Reserved	Reserved
16	TX_RX	Indicates whether teardown is a TX (0) or RX (1).
15-10	DMA Number	Indicates the DMA number for this teardown.
9-6	Reserved	Reserved
5-0	Channel Number	Indicates the channel number within the DMA that was torn down.

Table 25. Teardown Descriptor Words 1-7

Bits	Name	Description
31-0	Reserved	Reserved

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.

The following is the Transmit teardown procedure highlighting the steps required to be followed.

1. Set the TX_TEARDOWN bit in the CPPI DMA TX channel *n* global configuration register (TXGCR*n*).
2. Set the appropriate TX_TDOWN bit in the USBOTG controller's USB teardown register (TEARDOWN). Write Tx Endpoint Number to teardown to TEARDOWN[TX_TDOWN] fileld.
3. Check if the teardown descriptor has been received on the teardown queue: The completion queue (Queues 24 or 25) is usually used as the Teardown queue when the Teardown descriptor has been received, the descriptor address will be loaded onto CTRLD[24/25] register:

   (a) If not, go to step 2.
   (b) If so, go to step 4.
4. Set the appropriate TX_TDOWN bit in the USBOTG controller's USB teardown register (TEARDOWN). Set the bit corresponding to the Channel Number within TEARDOWN[TX_TDOWN] field.
5. Flush the TX FIFO in the Mentor OTG core: Set PERI_TXCSR[FLUSHFIFO] for the corresponding Endpoint.
6. Re-enable the Tx DMA channel:

   (a) Clear TXGCR*n*[TX_TEARDOWN and TX_ENABLE] bit.
   (b) Set TXGCR*n*[TX_ENABLE] bit.

2.8.5 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 26 displays the allocation (partition) of the available Queues.

Table 26. 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

2.8.5.1 Queuing Packets

Prior to queuing packets, the host/firmware should construct data buffer as well host packet/buffer descriptors within memory that is external to the CPPI 4.1 DMA module.

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.

2.8.5.2 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.

2.8.5.3 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.

2.8.5.3.1 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.

2.8.5.3.2 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.

2.8.5.3.3 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.

2.8.5.3.4 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.

2.8.5.3.5 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.

2.8.5.3.6 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.

2.8.5.3.7 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.

2.8.6 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 Queue Manager to be used as a scratch PAD RAM. The Queue Manager uses this memory to manage states of Descriptors submitted within the submit queues. In other words, this memory needs not to be managed by your software and your software responsibility is only for allocation of memory. The allocated memory can be a single block of memory that is contiguous or two blocks of memory that are not contiguous. These two blocks of memory are referred as a Linking RAM Regions and should not be confused with Memory Regions that are used to store Descriptors and the use of the term Region should be used in the context of its use.

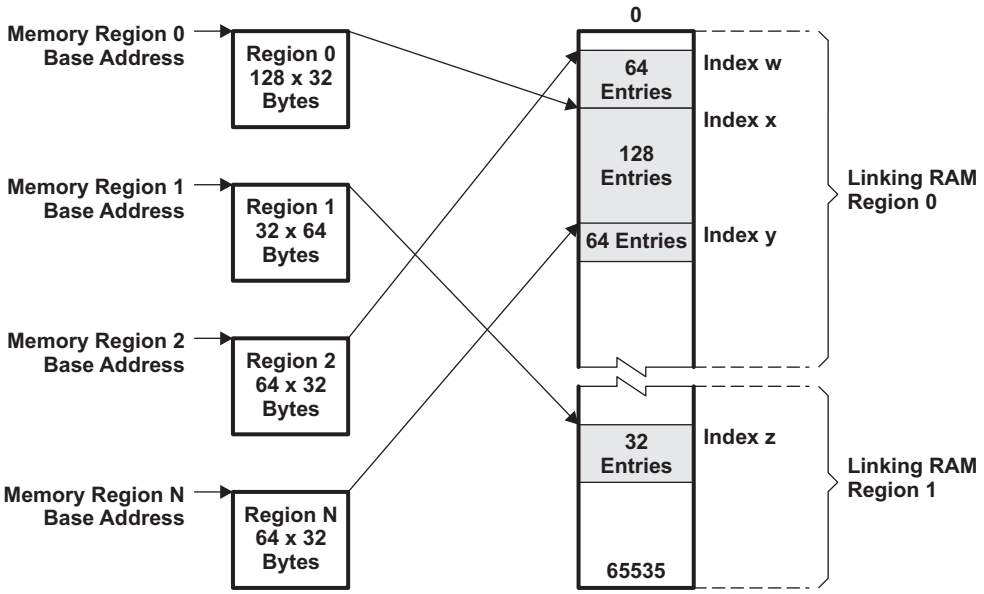
The physical size of the Linking RAM region(s) to be allocated depends on the total number of Descriptors defined within all memory regions. A minimum of four bytes of memory needs to be allocated for each Descriptor defined within all 16 Memory Regions.

The Queue Manager has the capability of managing up to 16 Memory Regions. These Memory Regions are used to store descriptors of variable sizes. The total number of Descriptors that can be managed by the Queue Manager should not exceed 16K. Each Memory Region has Descriptors of one configurable size, that is, Descriptors with different sizes cannot be housed within a single Memory Region. These 16K Descriptors are referenced internally in the Queue Manager by a 16-bit quantity index.

The information about the Linking RAM regions and the size that are allocated is communicated to the CPPI DMA via three registers dedicated for this purpose. Two of the three registers are used to store the 32-bit aligned start addresses of the Linking RAM regions. The remaining one register is used to store the size of the first Linking RAM. The size value stored here is the number of Descriptors that is to be managed by the Queue Manager within that region not the physical size of the buffer, which is four times the number of descriptors.

Note that you are not required to use both Linking RAM Regions, if the size of the Linking RAM for the first Region is large enough to accommodate all Descriptors defined. No Linking RAM size register for Linking RAM Region 2 exists. The size of the second Linking RAM, when used, is indirectly computed from the total number of Descriptors defined less the number of Descriptors managed by the first Linking RAM.

Figure 19. Relationship Between Memory Regions and Linking RAM



2.8.7 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.

2.8.8 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.

2.8.8.1 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 (4 entries per table word *n*) 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 size of the entries used by writing onto an internal register in the scheduler which sets the actual size of the array (it can be less than 256 entries).
3. The host writes an internal register bit to enable the scheduler. The scheduler is not required to be disabled in order to change the scheduler array contents.

2.8.8.2 Example of Scheduler Programming

Consider a three endpoints use on a system with the following configurations: EP1-Tx, EP2-Rx, and EP2-Tx. Two assumptions are considered:

Case 1: Assume that you would like to service each enabled endpoints (EP1-Tx, EP2-Rx, and EP2-Tx) with equal priority.

The scheduler handles the rate at which an endpoint is serviced by the number of credits programmed (entries) for that particular endpoint within the scheduler Table Words. The scheduler has up to 256 credits that it can grant and for this example the number of entries/credits could be anywhere from 3 to 256. However, the optimum and direct programming for this scenario would be programming only the first three entries of the scheduler via scheduler Table WORD[0]. Since this case expects the Scheduler to use only the first three entries, you communicate that by programming DMA_SCHED_CTRL.LAST_ENTRY with 2 (that is, 3 - 1). The Enabled Endpoint numbers and the data transfer direction is then communicated by programming the first three entries of WORD[0] (ENTRY0_CHANNEL = 1: ENTRY0_RXTX = 0; ENTRY1_CHANNEL = 2: ENTRY1_RXTX = 1;ENTRY2_CHANNEL = 2: ENTRY2_RXTX = 0). With this programming, the Scheduler will only service the first three entries in a round-robin fashion, checking each credited endpoint for transfer one after the other, and servicing the endpoint that has data to transfer.

Case 2: Enabled endpoint EP1-Tx is serviced at twice the rate as the other enabled endpoints (EP2-Rx and EP2-Tx).

The number of entries/credit that has to be awarded to EP1-Tx has to be twice as much of the others. Since only four entries/credits are required, two for EP1-Tx, one for EP2-Rx, and one for EP2-Tx, the use of scheduler Table WORD[0] would still suffice. Even though several scenarios exist to programming the order of service for this case, one scenario would be to allow EP1-Tx to be serviced back-to-back followed by the other enabled endpoints. Program DMA_SCHED_CTRL.LAST_ENTRY with 3 (that is, 4 - 1). Program WORD[0] (ENTRY0_CHANNEL = 1: ENTRY0_RXTX = 0; ENTRY1_CHANNEL = 1: ENTRY1_RXTX = 0; ENTRY2_CHANNEL = 2: ENTRY2_RXTX = 1; ENTRY3_CHANNEL = 2: ENTRY3_RXTX = 0).

2.8.8.3 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 is provided a sched_ready signal which is used to stall the scheduler until it can accept the credit. The DMA controller only asserts the sched_ready signal when it is in the IDLE state.
   (b) Once a credit has been accepted (indicated by sched_req and sched_ready both asserted), 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.

2.8.9 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 reserved by hardware as Completion Queues, if required for the DMA interrupt to be generated on a completion of a transfer.

Queues 24 and 25 are reserved by hardware for DMA transmit operations and queues 26 and 27 are reserved by hardware for DMA receive operations. If firmware uses these queues as Completion Queues, an interrupt will be generated when the transfer completes. If you need not to generate an interrupt, firmware is required to use queues that are not reserved as Completion Queues.

2.8.10 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.

2.8.10.1 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)

2.8.10.2 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

2.8.11 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.

2.8.11.1 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 DMA Mode Register (MODE) for the Endpoint field in use is programmed for Transparent Mode. MODE = 0000 0000h

2.8.11.2 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 DMA Mode Register (MODE) for the Endpoint field in use is programmed for RNDIS Mode. MODE = 1111 1111h

The above two setups yield the same result.

2.8.11.3 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 DMA Mode Register (MODE) for the Endpoint field in use is programmed for Generic RNDIS Mode. MODE = 3333 3333h

2.8.11.4 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 0x00.

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 0x00, 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

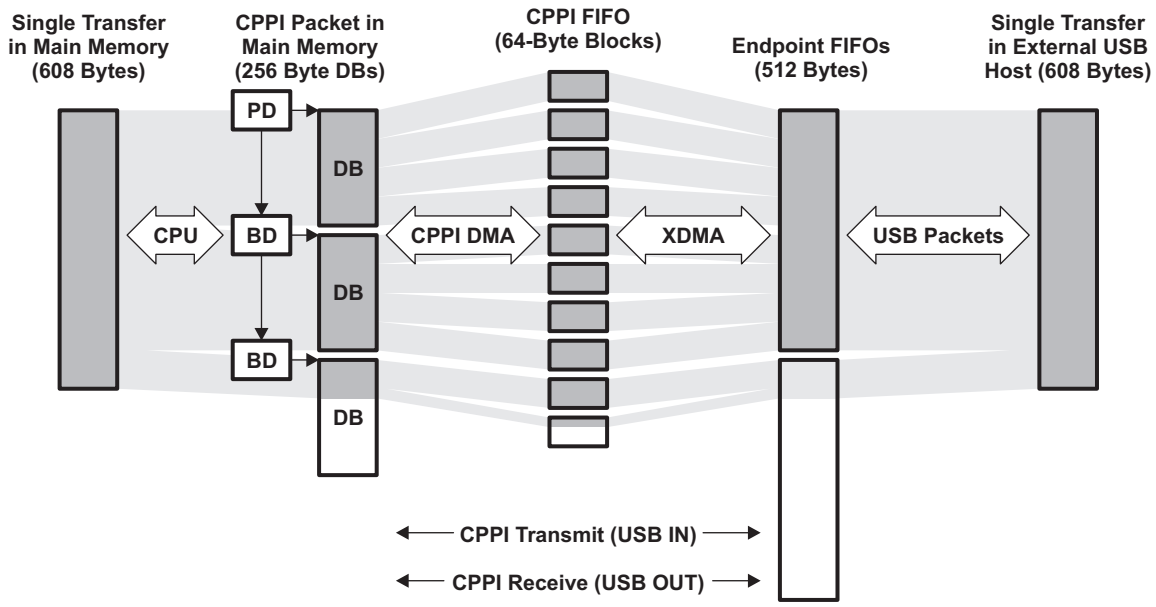
The following will configure all four ports/channels for Linux CDC DMA Transfer type.

- Disable RNDIS Mode globally. RNDIS bit in the control register (CTRLR) is cleared to 0.
- Configure the DMA Mode Register (MODE) for the Endpoint field in use is programmed for Linux CDC Mode. MODE = 2222 2222h

2.8.12 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 20).

Figure 20. High-Level Transmit and Receive Data Transfer Example



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)

2.8.12.1 Transmit USB Data Flow Using DMA

The transmit descriptors and queue status configuration prior to the transfer taking place is shown in Figure 21. An example of initialization for a transmit USB data flow is shown in Figure 22.

Figure 21. Transmit Descriptors and Queue Status Configuration

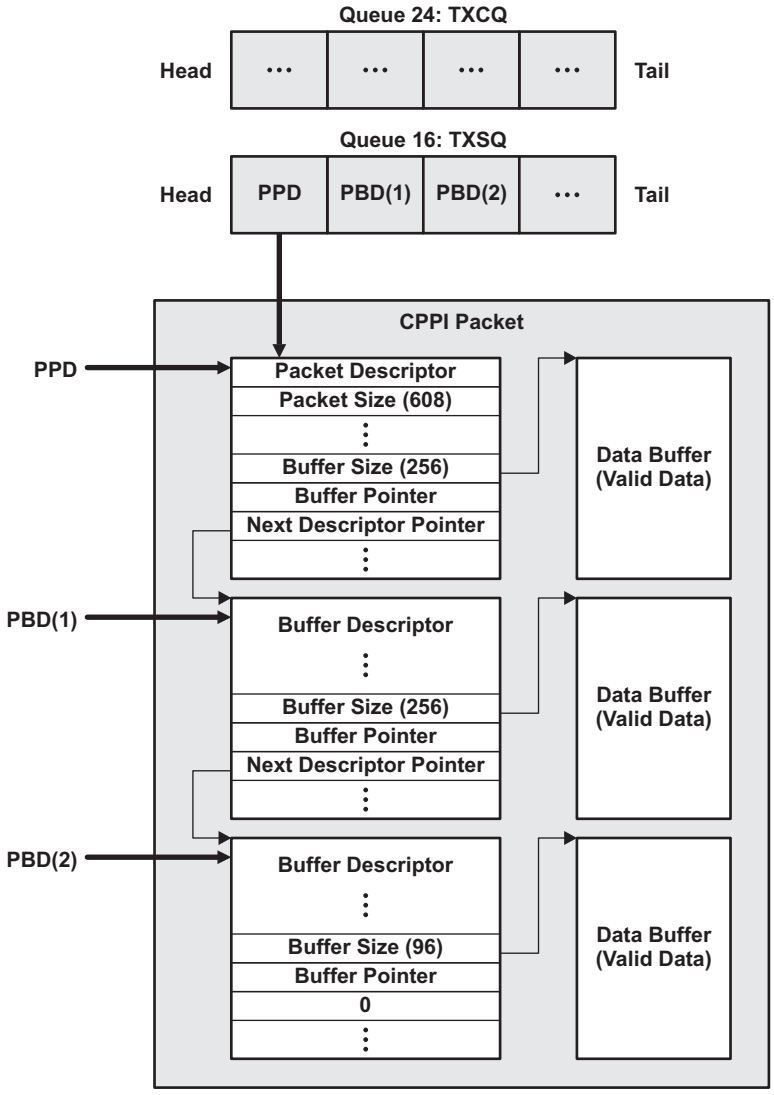
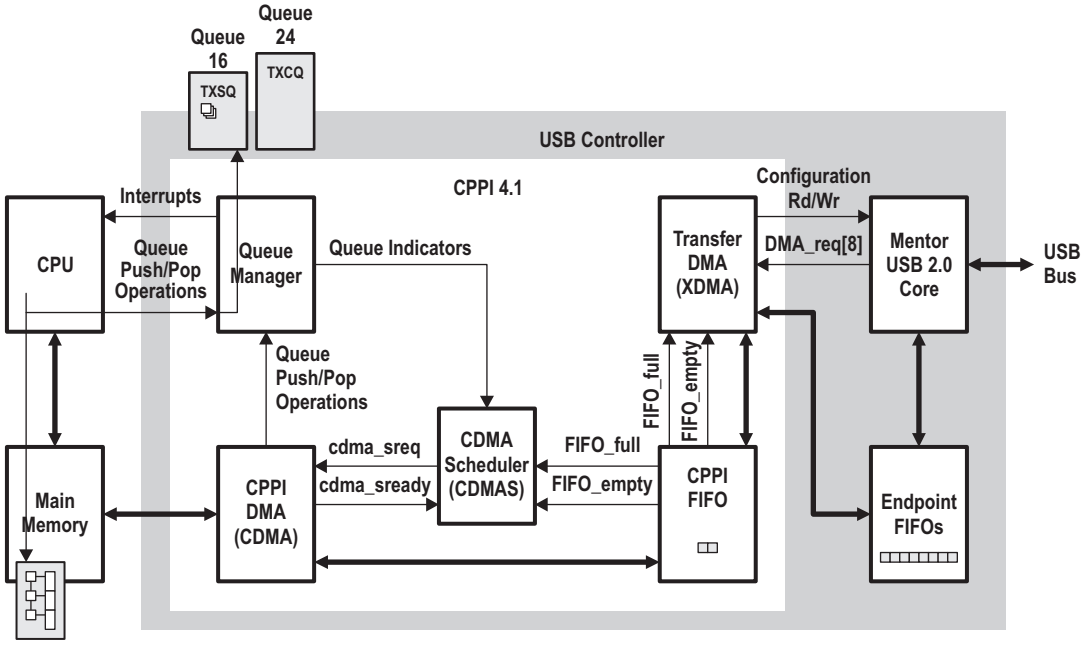


Figure 22. Transmit USB Data Flow Example (Initialization)



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 22.
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.

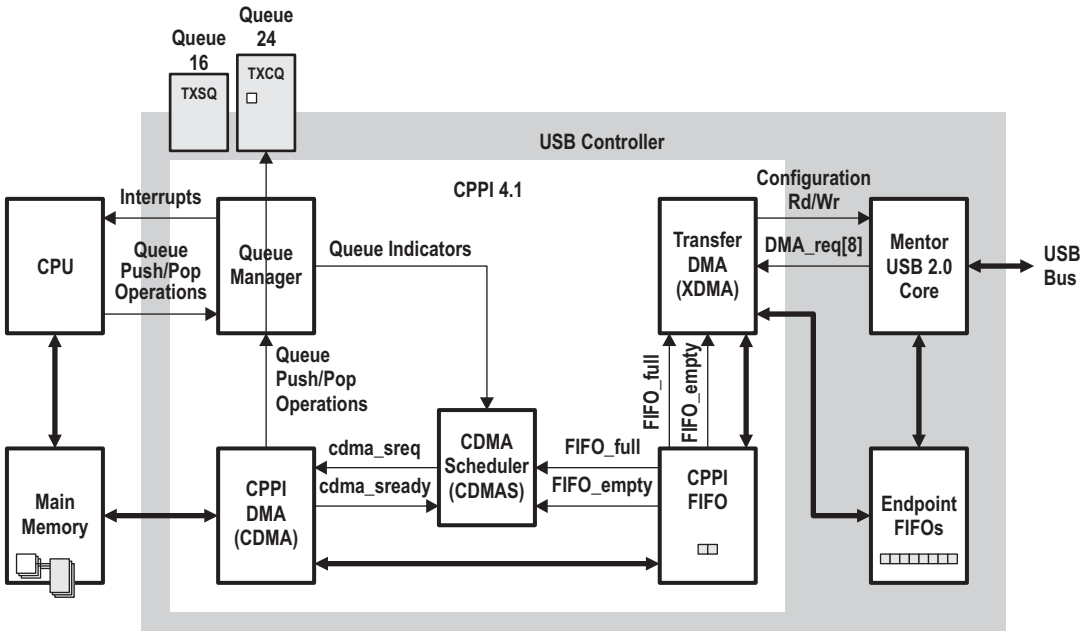
9. The CDMA reads the second buffer descriptor pointer.
10. CDMA reads the buffer descriptor from memory.
11. For each 64-byte block of data in the packet data payload:
    (a) The CDMA transfers a max burst of 64-byte block from the data to be transferred in main memory to the CPPI FIFO.
    (b) The 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 23.

Figure 23. 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.

2.8.12.2 Receive USB Data Flow Using DMA

The receive descriptors and queue status configuration prior to the transfer taking place is shown in Figure 24. An example of initialization for a receive USB data flow is shown in Figure 25.

Figure 24. Receive Descriptors and Queue Status Configuration

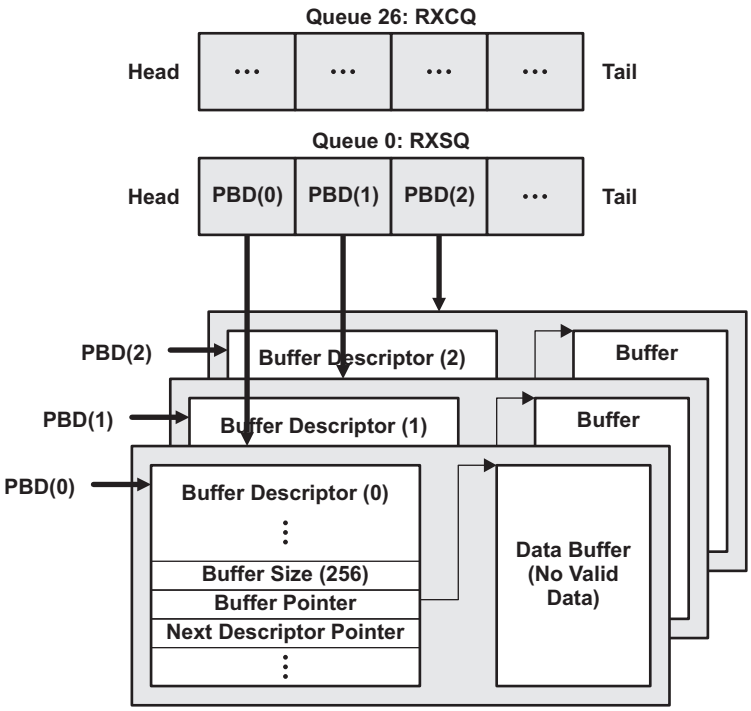
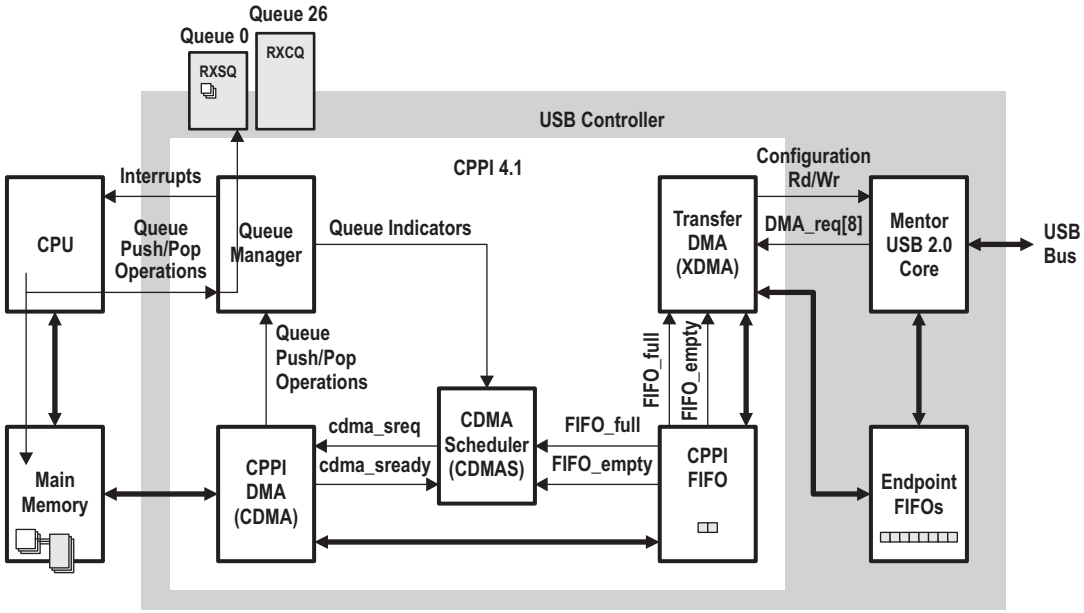


Figure 25. Receive USB Data Flow Example (Initialization)



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 25.
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):

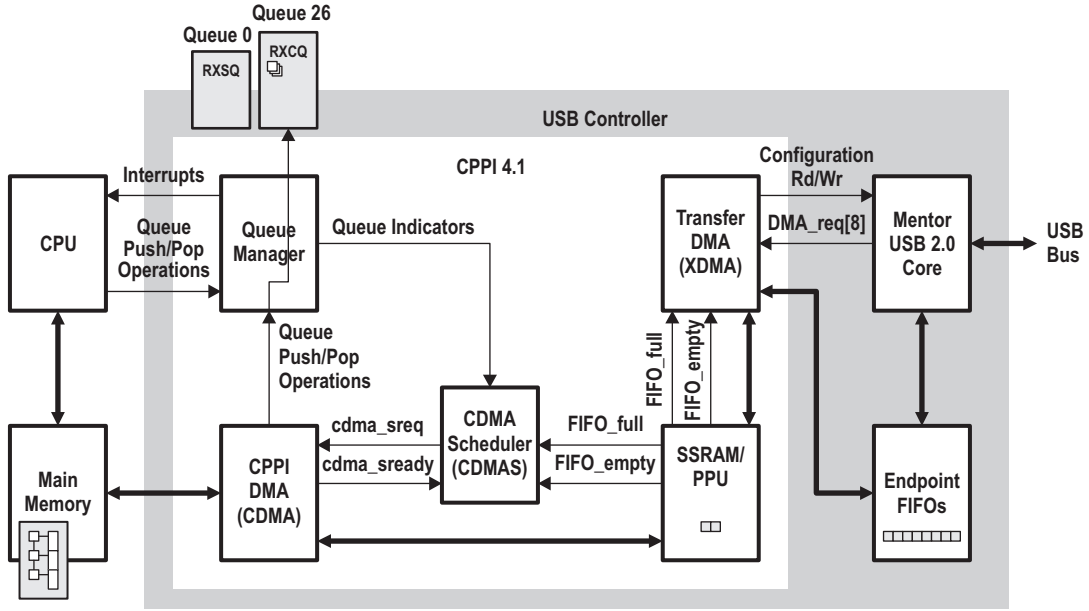
1. The Mentor USB 2.0 Core receives a USB packet from the USB Host and stores it in the Endpoint FIFO.
2. It then asserts a DMA_req to the XDMA informing it that data is available in the Endpoint FIFO.
3. The XDMA verifies the corresponding CPPI FIFO is not full via the FIFO_full signal, then starts 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 26 .

Figure 26. 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.

2.8.13 Interrupt Handling

Table 27 lists the interrupts generated by the USB controller.

Table 27. 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[8-0]	Interrupts for 9 USB conditions

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 nine USB interrupt conditions are listed in Table 28.

Table 28. USB Interrupt Conditions

Interrupt	Description
USB[8]	DRVVBUS level change
USB[7]	VBus voltage < VBus Valid Threshold (VBus error)
USB[6]	SRP detected
USB[5]	Device Disconnected (Valid in Host Mode)
USB[4]	Device Connected (Valid in Host Mode)
USB[3]	SOF started
USB[2]	Reset Signaling detected (In Peripheral Mode) Babble detected (In Host Mode)
USB[1]	Resume signaling detected
USB[0]	Suspend Signaling detected

2.8.13.1 USB Core Interrupts

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

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

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.

2.9 Test Modes

The USB2.0 controller supports the four USB 2.0 test modes defined for high-speed functions. It also supports an additional “FIFO access” test mode that can be used to test the operation of the CPU interface, the DMA controller (if configured), and the RAM block.

The test modes are entered by writing to the TESTMODE register. A test mode is usually requested by the host sending a SET_FEATURE request to Endpoint 0. When the software receives the request, it should wait until the Endpoint 0 transfer has completed (when it receives the Endpoint 0 interrupt indicating the status phase has completed) then write to the TESTMODE register.

NOTE: These test modes have no purpose in normal operation.

2.9.1 TEST_SE0_NAK

To enter the Test_SE0_NAK test mode, the software should set the TEST_SE0_NAK bit in the TESTMODE register to 1. The USB controller will then go into a mode in which it responds to any valid IN token with a NAK.

2.9.2 TEST_J

To enter the Test_J test mode, the software should set the TEST_J bit in the TESTMODE register to 1. The USB controller will then go into a mode in which it transmits a continuous J on the bus.

2.9.3 TEST_K

To enter the Test_K test mode, the software should set the TEST_K bit in the TESTMODE register to 1. The USB controller will then go into a mode in which it transmits a continuous K on the bus.

2.9.4 TEST_PACKET

To execute the Test_Packet, the software should:

1. Start a session (if the core is being used in Host mode).
2. Write the standard test packet (shown below) to the Endpoint 0 FIFO.
3. Write 8h to the TESTMODE register (TEST_PACKET = 1) to enter Test_Packet test mode.
4. Set the TxPktRdy bit in the CSR0 register (D1).

The 53 by test packet to load is as follows (all bytes in hex). The test packet only has to be loaded once; the USB controller will keep re-sending the test packet without any further intervention from the software.

00	00	00	00	00	00	00	00
00	AA	AA	AA	AA	AA	AA	AA
AA	EE	EE	EE	EE	EE	EE	EE
EE	FE	FF	FF	FF	FF	FF	FF
FF	FF	FF	FF	FF	7F	BF	DF
EF	F7	FB	FD	FC	7E	BF	DF
EF	F7	FB	FD	7E			

This data sequence is defined in Universal Serial Bus Specification Revision 2.0, Section 7.1.20. The USB controller will add the DATAA0 PID to the head of the data sequence and the CRC to the end.

2.9.5 FIFO_ACCESS

The FIFO Access test mode allows you to test the operation of CPU Interface, the DMA controller (if configured), and the RAM block by loading a packet of up to 64 bytes into the Endpoint 0 FIFO and then reading it back out again. Endpoint 0 is used because it is a bidirectional endpoint that uses the same area of RAM for its Tx and Rx FIFOs.

NOTE: The core does not need to be connected to the USB bus to run this test. If it is connected, then no session should be in progress when the test is run.

The test procedure is as follows:

1. Load a packet of up to 64 bytes into the Endpoint 0 Tx FIFO.
2. Set CSR0.TxPktRdy.
3. Write 40h to the TESTMODE register (FIFO_ACCESS = 1).
4. Unload the packet from the Endpoint Rx FIFO, again.
5. Set CSR0.ServicedRxPktRdy.

Writing 40h to the TESTMODE register causes the following sequence of events:

1. The Endpoint 0 CPU pointer (that records the number of bytes to be transmitted) is copied to the Endpoint 0 USB pointer (that records the number of bytes received).
2. The Endpoint 0 CPU pointer is reset.
3. CSR0.TxPktRdy is cleared.
4. CSR0.RxPktRdy is set.
5. An Endpoint 0 interrupt is generated (if enabled).

The effect of these steps is to make the Endpoint 0 controller act as if the packet loaded into the Tx FIFO has flushed and the same packet received over the USB. The data that was loaded in the Tx FIFO can now be read out of the Rx FIFO.

2.9.6 FORCE_HOST

The Force Host test mode enables you to instruct the core to operate in Host mode, regardless of whether it is actually connected to any peripheral; that is, the state of the CID input and the LINESTATE and HOSTDISCON signals are ignored. (While in this mode, the state of the HOSTDISCON signal can be read from the BDEVICE bit in the device control register (DEVCTL).)

This mode, which is selected by writing 80h to the TESTMODE register (FORCE_HOST = 1), allows implementation of the USB Test_Force_Enable (7.1.20). It can also be used for debugging PHY problems in hardware.

While the FORCE_HOST bit remains set, the core enters the Host mode when the SESSION bit in DEVCTL is set to 1 and remains in the Host mode until the SESSION bit is cleared to 0 even if a connected device is disconnected during the session. The operating speed while in this mode is determined by the FORCE_HS and FORCE_FS bits in the TESTMODE register.

2.10 Reset Considerations

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

2.10.1 Software Reset Considerations

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.

A software reset on the DSP or ARM CPU does not affect the register values and operation of the USB controller.

2.10.2 Hardware Reset Considerations

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

2.11 Interrupt Support

The USB peripheral provides the interrupts listed in Table 29 to the interrupt distributor module (INTD). For information on the mapping of interrupts, see your device-specific data manual.

Table 29. USB Interrupts

Event	Acronym	Source
ARM Event = 58	USB0_INT	USB 2.0 Controller
DSP Event = 19	USB0_INT	USB 2.0 Controller

2.12 DMA Event Support

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

2.13 Power Management

The USB controller can be placed in reduced power modes to conserve power during periods of low activity. The power management of the peripheral is controlled by the processor Power and Sleep Controller (PSC). The PSC acts as a master controller for power management for all of the peripherals on the device. For detailed information on power management procedures using the PSC, see your device-specific *System Reference Guide*.

3 Use Cases

The USB supports the following use cases.

3.1 User Case 1: Example of How to Initialize the USB Controller

Example 1. Initializing the USB2.0 Controller

```
void usb_init()
{
       Uint16 I;

// ************************************************************************
// Configure DRVVBUS Pin to be used for USB
// ************************************************************************
// MAKE SURE WRITE ACCESS KEY IS INITIALIZED PRIOR TO ACCESSING ANY OF THE
// BOOTCFG REGISTERS.


   BootCfg->KICK0R = KICK0KEY;      // Write Access Key 0
   BootCfg->KICK1R = KICK1KEY;      // Write Access Key 1
   /* CONFIGURE THE DRVVBUS PIN HERE.*/
   /* See your device-specific System Reference Guide for more information on how to set up the
pinmux. */


// Reset the USB controller:
        usbRegs->CTRLR |= 0x00000001;


//Wait until controller is finished with Reset. When done, it will clear the RESET bit field.
        while ((usbRegs->CTRLR & 0x1) == 1);


        // RESET: Hold PHY in Reset
        BootCfg->CFGCHIP2 |= 0x00008000;   // Hold PHY in Reset
        // Drive Reset for few clock cycles
        for (I=0; i<50; I++);
        // RESET: Release PHY from Reset
        BootCfg->CFGCHIP2 &= 0xFFFF7FFF;   // Release PHY from Reset


/* Configure PHY with the Desired Operation */
              // OTGMODE
              BootCfg->CFGCHIP2 &= 0xFFFF9FFF;   // 00=> Do Not Override PHY Values

              // PHYPWDN
              BootCfg->CFGCHIP2 &= 0xFFFFFBFF;   // 1/0 => PowerdDown/ NormalOperation
              // OTGPWRDN
              BootCfg->CFGCHIP2 &= 0xFFFFFDFF;   // 1/0 => PowerDown/ NormalOperation
              // DATAPOL
              BootCfg->CFGCHIP2 |= 0x00000100;   // 1/0 => Normal/ Reversed
              // SESNDEN
              BootCfg->CFGCHIP2 |= 0x00000020;   // 1/0 => NormalOperation/ SessionEnd
              // VBDTCTEN
              BootCfg->CFGCHIP2 |= 0x00000010;   // 1/0 => VBUS Comparator Enable/ Disable



/* Configure PHY PLL use and Select Source */
        // REF_FREQ[3:0]
        BootCfg->CFGCHIP2 |= 0x00000002;        // 0010b => 24MHz Input Source

        // USB2PHYCLKMUX: Select External Source
        BootCfg->CFGCHIP2 &= 0xFFFFF7FF;        // 1/0 => Internal/External(Pin)

        // PHY_PLLON: On Simulation PHY PLL is OFF
        BootCfg->CFGCHIP2 |= 0x00000040;        // 1/0 => On/ Off



/* Wait Until PHY Clock is Good */
```

Example 1. Initializing the USB2.0 Controller (continued)

```
        while ((BootCfg->CFGCHIP2 & 0x00020000) == 0); // Wait Until PHY Clock is Good.

        #ifndef HS_ENABLE
                // Disable high-speed
                CSL_FINS(usbRegs->POWER,USB_OTG_POWER_HSEN,0);
        #else
                // Enable high-speed
                CSL_FINS(usbRegs->POWER,USB_OTG_POWER_HSEN,1);
        #endif


// Enable Interrupts
        // Enable interrupts in OTG block
        usbRegs->CTRLR &= 0xFFFFFFF7; // Enable PDR2.0 Interrupt

        usbRegs->INTRTXE = 0x1F;              // Enable All Core Tx Endpoints Interrupts + EP0
Tx/Rx interrupt
        usbRegs->INTRRXE = 0x1E;              // Enable All Core Rx Endpoints Interrupts

        // Enable all interrupts in OTG block
        usbRegs->INTMSKSETR = 0x01FF1E1F;

        // Enable all USB interrupts in MUSBMHDRC
        usbRegs->INTRUSBE = 0xFF;

        // Enable SUSPENDM so that suspend can be seen UTMI signal
        CSL_FINS(usbRegs->POWER,USB_OTG_POWER_ENSUSPM,1);

        //Clear all pending interrupts
        usbRegs->INTCLRR = usbRegs->INTSRCR;

        #if (_USB_PERIPHERAL_)   // defined within project file if need to function as Peripheral.
                // Set softconn bit
                CSL_FINS(usbRegs->POWER,USB_OTG_POWER_SOFTCONN,1);
                while ((usbRegs->DEVCTL & 0x01) == 0);  //Stay here until controller goes in
Session.
        #else
                // Start a session
                CSL_FINS(usbRegs->DEVCTL,USB_OTG_DEVCTL_SESSION,1);
        #endif

}
```

Example 2. Initializing the CPPI 4.1 DMA Controller

```
void cppiDmaInit() {

       switch (DMAmode) {
              case RNDIS:
                     usbRegs->CTRLR |=0x00000010; // Enable RNDIS from Global Level
                     usbRegs->RNDISR=0x11111111;
                     break;

              case GENERIC_RNDIS:
                     usbRegs->CTRLR &= ~0x00000010; // Disable RNDIS from Global Level
                     usbRegs->RNDISR=0x33333333;
                     usbRegs->GENRNDISSZ[chan_num].SIZE=descPacketLength;
                     break;

              case LINUX_CDC:
                     usbRegs->CTRLR &= ~0x00000010; // Disable RNDIS from Global Level
                     usbRegs->RNDISR=0x22222222;
                     break;

              case TRANSPARENT:
                     usbRegs->CTRLR &= ~0x00000010; // Disable RNDIS from Global Level
                     usbRegs->RNDISR=0x00000000;
                     break;

              default:
                     usbRegs->CTRLR |=0x00000010; // Enable RNDIS from Global Level
                     break;
       }
       #ifndef _USB_PERIPHERAL_            // If Controller is assuming Host Role
              #ifdef TRANSPARENT
                     usbRegs->AUTOREQ=00;      // No Auto Req
              #else
                     usbRegs->AUTOREQ=01;      // Auto Req on all but EOP
             #endif
       #endif

       A Single Queue Manager (00b) exist and 16 Regions; no particular assignment exists.

       Program Link Ram0 and Link Ram1, Base & Size.
       No Link Ram1 Size Register exists. Most likely is using the same Size Register used
       //    Link Ram1 Base
       usbRegs->QMGR.LRAM0BASE = (Uint32)queueMgrLinkRam0;
       usbRegs->QMGR.LRAM0SIZE = LINKRAM0SIZE/4;
       usbRegs->QMGR.LRAM1BASE = (Uint32)queueMgrLinkRam1;

// Allocate Resource to Region 0 (can use any of the 16 available) (memory location for Host
Packet Descriptors)
// DESC_SIZE value should be [1-8]. Values of 9 - 15 are reserved.
// Since a minimum of 32 Bytes is required, only program values above 32.
// Host Packet Descriptor sizes: Min/Max = 32/(104 + Opt S/W Data)
// REG_SIZE is the total # of Descriptors the Region can accommodate. At the minimum should be
capable of handling 32 Descriptors.

// This example is allocating specific regions for each port
       usbRegs->QMEMREGION[chan_num].QMEMRBASE=((Uint32)region0DescriptorSpace);
       usbRegs->QMEMREGION[chan_num].QMEMRCTRL=(REG0START_INDEX<<16) | (DESC_SIZE<<8) | REG_SIZE;

Configure the Scheduler
// Configure the Tx/Rx Word[x=0,1] and Scheduler Configuration Register
// Priorities are handled by programming more of the channel number wanting to see serviced
// 64 Words exist for a total of 256 entries.

       // Credit can be given to both Tx and Rx Channel within the same Register.
       // Here Rx Credit is given first for the single Rx Channel defined by chan_num.
       // Here we are using the first 8 credits and is assigned to the Channel/Endpoint
       //  being serviced.
       usbRegs->DMA_SCHED.ENTRY[0]= (0x80808080 | (chan_num | (chan_num << 8) | (chan_num << 16)
| (chan_num << 24)));
       // Corresponds to WORD0 Offset 0x2800

       usbRegs->DMA_SCHED.ENTRY[1]=(0x00000000 | (chan_num | (chan_num << 8) | (chan_num << 16) |
```

Example 2. Initializing the CPPI 4.1 DMA Controller (continued)

```
(chan_num << 24)));
       // Corresponds to WORD1 Offset 0x2804, etc

       // Scheduler is Enabled and number of Credits entered (since 8 of the 256 credits are used
program 8-1=7 and enable Scheduler)
       usbRegs->DMA_SCHED.DMA_SCHED_CTRL=0x80000007; // Scheduler Control Register Offset 0x2000

// Configure Tx and Rx DMA State Registers
// The Rx Channel Global Configuration Registers are used to initialize the global
// (non descriptor type specific) behavior of each of the Rx DMA channels.  If the
// enable bit is being set, the Rx/Tx Channel Global Configuration Register SHOULD ONLY
// BE WRITTEN AFTER ALL OF THE OTHER Rx/Tx CONFIGURATION REGISTERS HAVE BEEN INITIALIZED.
// Only a Single Queue Manager exists & its value is 0.


// RxHPCRA/B requires for Queue Manager (have only one and is 00b) and Receive Submit Queues for
the
// first 4 Host Buffer Descriptors to be programmed. Since Queues 0 to 15 are alloted for Receive
// operations and there exist no dedicated queue assignments for each channel, a user can
// associate any Queue with any Channel and this association is not fixed for the Receive
Operations.
// Queue[15:0]<==>Any Rx Channel

// However, for Tx Operations, Dedicated Submit Queues have been assigned for Each Channel,
Endpoints.
// Queue[17:16]<==>TxCh[0], Queue[19:18]<==>TxCh[1], Queue[21:20]<==>TxCh[2],
Queue[23:22]<==>TxCh[3]

// CDMA Rx Chanel x Host Packet Configuration Registers A & B
// For a Single Descriptor setup, all you will be needing is RXHPCRA[13,12 & 11:00]

// Assumed that all Descriptors are going to be using the same Queue, but this is configurable.

       usbRegs->DMA_CTRL[chan_num].RXHPCRA=(rxSubmitQ | (rxSubmitQ<<16)); // Rx Channel 0 Host
Pkt Cfg Reg A Offset 0x180C
       usbRegs->DMA_CTRL[chan_num].RXHPCRB=(rxSubmitQ | (rxSubmitQ<<16)); // Rx Channel 0 Host
Pkt Cfg Reg B Offset 0x1810

       // For Loopback purposes, the same data buffer will be used for receiving and transmitting.
       //  However, different descriptors (tx) and queues will be used to process the same data
from the same buffer
       // Rx Submit Queues are assigned by H/W but they are not port specific. Queues 0-15 are
available for any channel.
       usbRegs->DMA_CTRL[chan_num].RXGCR=(0x81004000 | rxCompQ);
       // Rx Channel 0 Host Pkt Cfg Register Offset 0x1808 (Use Queue 26 for Completion/Return
Queue)

       // Tx Submit Queues are assigned by H/W and are Channel Specific. 2 Submit Queues for each
port. Queues 16, 17 for port 0, etc
       usbRegs->DMA_CTRL[chan_num].TXGCR=(0x80000000 | txCompQ); // Tx Channel 0 Host Pkt Cfg
Register Offset 0x1800
}
```

3.2 User Case 2: Example of How to Program the USB Endpoints in Peripheral Mode

Example 3. Programming the USB Endpoints in Peripheral Mode

```
// DMA channel number.  Valid values are 0, 1, 2, or 3.
int CHAN_NUM = 0;

// Fifo sizes: uncomment the desired size.
// This example uses 64-byte fifo.
// int fifosize = 0;    // 8 bytes
// int fifosize = 1;    // 16 bytes
// int fifosize = 2;    // 32 bytes
   int fifosize = 3;    // 64 bytes
// int fifosize = 4;    // 128 bytes
// int fifosize = 5;    // 256 bytes
// int fifosize = 6;    // 512 bytes
// int fifosize = 7;    // 1024 bytes
// int fifosize = 8;    // 2048 bytes
// int fifosize = 9;    // 4096 bytes

// FIFO address.  Leave 64-bytes for endpoint 0.
int fifo_start_address = 8;

// Uncomment the desired buffering.  If double-buffer is selected, actual
// FIFO space will be twice the value listed above for fifosize.
// This example uses single buffer.
   int double_buffer = 0;  // Single-buffer
// int double_buffer = 1;  // Double-buffer

// For maximum packet size this formula will usually work, but it can also be
// set to another value if needed.  If non power of 2 value is needed (such as
// 1023) set it explicitly.
#define FIFO_MAXP 8*(1<<fifosize);

// Set the following variable to the device address.
int device_address = 0;


// The following code should be run after receiving a USB reset from the host.


// Initialize the endpoint FIFO.  RX and TX will be allocated the same sizes.
usbRegs->INDEX = CHAN_NUM+1;
usbRegs->RXFIFOSZ = fifosize | ((double_buffer & 1)<<4);
usbRegs->RXFIFOADDR = fifo_start_address;
usbRegs->TXFIFOSZ =    fifosize | ((double_buffer & 1)<<4);
usbRegs->TXFIFOADDR = fifo_start_address + (1<<(fifosize+double_buffer));
usbRegs->RXMAXP = FIFO_MAXP;
usbRegs->TXMAXP = FIFO_MAXP;

// Force Data Toggle is optional for interrupt traffic.  Uncomment if needed.
// CSL_FINS(usbRegs->PERI_TXCSR,USB_PERI_TXCSR_FRCDATATOG,1);

// Uncomment below to configure the endpoint for ISO and not respond with a
// handshake packet.
// CSL_FINS(usbRegs->PERI_RXCSR,USB_PERI_RXCSR_ISO,1);
// CSL_FINS(usbRegs->PERI_TXCSR,USB_PERI_TXCSR_ISO,1);

// After receiving a successful set-address command, set the following register
// to the specified address immediately following the status stage.
usbRegs->FADDR = device_address;
```

3.3 User Case 3: Example of How to Program the USB Endpoints in Host Mode

Example 4. Programming the USB Endpoints in Host Mode

```
// DMA channel number.  Valid values are 0, 1, 2, or 3.
int CHAN_NUM = 0;

// Fifo sizes: uncomment the desired size.
// This example uses 64-byte fifo.
// int fifosize = 0;    // 8 bytes
// int fifosize = 1;    // 16 bytes
// int fifosize = 2;    // 32 bytes
   int fifosize = 3;    // 64 bytes
// int fifosize = 4;    // 128 bytes
// int fifosize = 5;    // 256 bytes
// int fifosize = 6;    // 512 bytes
// int fifosize = 7;    // 1024 bytes
// int fifosize = 8;    // 2048 bytes
// int fifosize = 9;    // 4096 bytes

// FIFO address.  Leave 64-bytes for endpoint 0.
int fifo_start_address = 8;

// Uncomment the desired buffering.  If double-buffer is selected, actual
// FIFO space will be twice the value listed above for fifosize.
// This example uses single buffer.
   int double_buffer = 0;  // Single-buffer
// int double_buffer = 1;  // Double-buffer

// Set the following variable to the device endpoint type: CONTROL ISO BULK or IN
  int device_protocol = BULK;
//int device_protocol = ISO;
//int device_protocol = INT;

// USB speeds
#define LOW_SPEED 0
#define FULL_SPEED 1
#define HIGH_SPEED 2

// TXTYPE protocol
#define CONTROL 0
#define ISO 1
#define BULK 2
#define INT 3

// For maximum packet size this formula will usually work, but it can also be
// set to another value if needed.  If non power of 2 value is needed (such as
// 1023) set it explicitly.
#define FIFO_MAXP 8*(1<<fifosize);

// Set the following variable to the device address.
int device_address = 1;

// Set the following variable to the device endpoint number.
int device_ep = 1;

// Variable used for endpoint configuration
Uint8 type = 0;

// Variable to keep track of errors
int error = 0;


// The following code should be run after resetting the attached device


// Initialize the endpoint FIFO.  RX and TX will be allocated the same sizes.
usbRegs->INDEX = CHAN_NUM+1;
usbRegs->RXFIFOSZ = fifosize | ((double_buffer & 1)<<4);
usbRegs->RXFIFOADDR = fifo_start_address;
usbRegs->TXFIFOSZ =    fifosize | ((double_buffer & 1)<<4);
```

Example 4. Programming the USB Endpoints in Host Mode (continued)

```
usbRegs->TXFIFOADDR = fifo_start_address + (1<<(fifosize+double_buffer));
usbRegs->RXMAXP = FIFO_MAXP;
usbRegs->TXMAXP = FIFO_MAXP;


//Configure the endpoint
switch (device_speed) {
    case LOW_SPEED : type = (3<<6) | ((device_protocol & 3) << 4) | (device_ep & 0xf); break;
    case FULL_SPEED: type = (2<<6) | ((device_protocol & 3) << 4) | (device_ep & 0xf); break;
    case HIGH_SPEED: type = (1<<6) | ((device_protocol & 3) << 4) | (device_ep & 0xf); break;
    default:error++;
}
usbRegs->EPCSR[CHAN_NUM+1].HOST_TYPE0 = type;  // TXTYPE
usbRegs->EPCSR[CHAN_NUM+1].HOST_RXTYPE = type;

// Set NAK limit / Polling interval (Interrupt & Iso protocols)
if ((device_protocol == INT) || (device_protocol == ISO)) {
    usbRegs->EPCSR[CHAN_NUM+1].HOST_NAKLIMIT0 = TXINTERVAL;    // TX Polling interval
    usbRegs->EPCSR[CHAN_NUM+1].HOST_RXINTERVAL = RXINTERVAL;    // RX Polling interval
} else {
    usbRegs->EPCSR[CHAN_NUM+1].HOST_NAKLIMIT0 = 2;    // Frames to timeout from NAKs
    usbRegs->EPCSR[CHAN_NUM+1].HOST_RXINTERVAL = 2;    // Frames to timeout from NAKs
}

//Set the address for transactions after SET ADDRESS successfully completed
usbRegs->EPTRG[CHAN_NUM+1].TXFUNCADDR = device_address;
usbRegs->EPTRG[CHAN_NUM+1].RXFUNCADDR = device_address;
```

3.4 User Case 4: Example of How to Program the USB DMA Controller

Example 5. Programming the USB DMA Controller

```
typedef struct {
        Uint32 PktLength:22;
        Uint32 ProtSize:5;
        Uint32 HostPktType:5; // This should be 16
}HPDWord0;

typedef struct {
        Uint32 DstTag:16;        //bits[15:0] always Zero
        Uint32 SrcSubChNum:5;    //bits[20:16] always Zero
        Uint32 SrcChNum:6;       //bits[26:21]
        Uint32 SrcPrtNum:5;      //bits[31:27]
}HPDWord1;

typedef struct {
        Uint32 PktRetQueue:12;   //bits[11:0]
        Uint32 PktRetQM:2;       //bits[13:12]
        Uint32 OnChip:1;         //bit[14]
        Uint32 RetPolicy:1;      //bit[15]
        Uint32 ProtoSpecific:4;  //bits[19:16]
        Uint32 Rsv:6;            //bits[25:20]
        Uint32 PktType:5;        //bits[30:26]
        Uint32 PktErr:1;         //bit[31]
}HPDWord2;

typedef struct hostPacketDesc {
        HPDWord0 HPDword0;
        HPDWord1 HPDword1;
        HPDWord2 HPDword2;
        Uint32 HPDword3buffLength;
        Uint32 HPDword4buffAdd;
        Uint32 HPDword5nextHBDptr;
        Uint32 HPDword6orgBuffLength;
        Uint32 HPDword7orgBuffAdd;
} HostPacketDesc;


// The following sample code uses region 0 for all of the ports. Ports/Channels/EPs are not
limited to a single region


void initSingleHPDorHBD(Uint16 descNum, dataDir dir, descType desc, Uint16 returnQueue) {

/*

*******************************************************************************************

 Initialize a Single Transmit and Receive Host Packet or Buffer Descriptor

*/

            if ((desc==PACKET_DESC) & (dir==TRANSMIT))
                        region0DescriptorSpace[descNum].HPDword0.HostPktType=16;   //This value
is always fixed. For Packet Type Decriptors=16
            else
                        region0DescriptorSpace[descNum].HPDword0.HostPktType=0;     //Word0,
Word1, and Half of Word2 are Reserved for Buffer Descriptors.


            region0DescriptorSpace[descNum].HPDword0.ProtSize=0;


            if ((dir==TRANSMIT) & (desc==PACKET_DESC)) {
                        if (DMAmode==TRANSPARENT)

region0DescriptorSpace[descNum].HPDword0.PktLength=singlePktLength;           //Packet Length
(For Transparent Mode this is <= Tx/RxMaxP Value. For RNDIS or like it is = Tx/RxMaxP Value).
                        else
```

Example 5. Programming the USB DMA Controller (continued)

```
region0DescriptorSpace[descNum].HPDword0.PktLength=descPacketLength;        //Actual Packet
Length: This is the size of the Packet noted at descriptor level to be Transmitted.



                                                                  // This is different from the USB Max
Packet Size. This is the total data length.

            }



            else      // This and other Packet related info will be updated by the PORT for
Receive and can be any value.
                        region0DescriptorSpace[descNum].HPDword0.PktLength=0;
// This is actual Packet Length. It will be populated by the Rx Port of the CPPI DMA


            region0DescriptorSpace[descNum].HPDword1.DstTag=0;              //Always programmed
to ZERO.
            region0DescriptorSpace[descNum].HPDword1.SrcSubChNum=0;  //Always programmed to ZERO.
            region0DescriptorSpace[descNum].HPDword1.SrcChNum=0;                    //Always
programmed to ZERO.


            if(desc==BUFFER_DESC)
                        region0DescriptorSpace[descNum].HPDword1.SrcPrtNum=0;        //Word1 is
Reserved for Buffer DESC.
            else
                        region0DescriptorSpace[descNum].HPDword1.SrcPrtNum=chan_num+1;
//Ports[1,2,3,4] is associated with Endpoints[1,2,3,4] respectively.



            region0DescriptorSpace[descNum].HPDword2.PktRetQueue=returnQueue; //24 and 25 for Tx -
 26 and 27 for Rx Completion
            region0DescriptorSpace[descNum].HPDword2.PktRetQM=0;
            region0DescriptorSpace[descNum].HPDword2.OnChip=1;             // Descriptor is
located On-Chip
            region0DescriptorSpace[descNum].HPDword2.RetPolicy=0;
            region0DescriptorSpace[descNum].HPDword2.ProtoSpecific=0;
            region0DescriptorSpace[descNum].HPDword2.Rsv=0;


            if(desc==BUFFER_DESC)
                        region0DescriptorSpace[descNum].HPDword2.PktType=0;
 // Half of Word 3 is Reserved for Buffer DESC.
            else
                        region0DescriptorSpace[descNum].HPDword2.PktType=5;
 // USB Packet ID is 5


            region0DescriptorSpace[descNum].HPDword2.PktErr=0;


            if(DESCsetup==SINGLE_DESC_SETUP)
                        region0DescriptorSpace[descNum].HPDword3buffLength=descPacketLength;
            else
                        region0DescriptorSpace[descNum].HPDword3buffLength=singlePktLength;


            if ((dir==TRANSMIT) & (prevDescTxNum==0)) {
                        region0DescriptorSpace[descNum].HPDword4buffAdd=(Uint32)rxBuffer;
                        prevDescTxNum++;
            } else if ((dir==TRANSMIT) & (prevDescTxNum!=0)) {

region0DescriptorSpace[descNum].HPDword4buffAdd=region0DescriptorSpace[descNum-1].HPDword4buffAdd
+ region0DescriptorSpace[descNum-1].HPDword3buffLength;
                        prevDescTxNum++;
            }
```

Example 5. Programming the USB DMA Controller (continued)

```
            if ((dir==RECEIVE) & (prevDescRxNum==0)) {
                        region0DescriptorSpace[descNum].HPDword4buffAdd=(Uint32)rxBuffer;
                        prevDescRxNum++;
            } else if ((dir==RECEIVE) & (prevDescRxNum!=0)) {

region0DescriptorSpace[descNum].HPDword4buffAdd=region0DescriptorSpace[descNum-1].HPDword4buffAdd
+ region0DescriptorSpace[descNum-1].HPDword3buffLength;
                        prevDescRxNum++;
            }


            if ((dir==TRANSMIT) & ((prevDescTxNum-1)>0))
                        region0DescriptorSpace[descNum-
1].HPDword5nextHBDptr=(Uint32)&region0DescriptorSpace[descNum];            //Modify Previous Link
Address


            region0DescriptorSpace[descNum].HPDword5nextHBDptr=0;        //Current Descriptor is
the Last Descriptor: Null Value is used as the Next Buffer Descriptor Address
region0DescriptorSpace[descNum].HPDword6orgBuffLength=region0DescriptorSpace[descNum].HPDword3buff
Length;

region0DescriptorSpace[descNum].HPDword7orgBuffAdd=region0DescriptorSpace[descNum].HPDword4buffAdd
;;
}



// usage: qDesc2SubmitQ(rx/txSubmitQ,HPD/HBD);  //Queue Number, HPDdescriptorNumber
void qDesc2SubmitQ(Uint16 queueNum, Uint16 hpdDescriptorNum) {
        usbRegs->QCTRL[queueNum].CTRLD=(Uint32)&region0DescriptorSpace[hpdDescriptorNum] | 0x2;
// bits[4:0]=dec_size=[0-31]=[24,28,32,...,148]
}

void enableCoreTxDMA(Uint16 endPoint) {
        Uint16 index_save;
        index_save=usbRegs->INDEX;
        usbRegs->INDEX=endPoint;
        usbRegs->TXCSR.PERI_TXCSR&=0x7FFF; // Clear AUTOSET
        usbRegs->TXCSR.PERI_TXCSR|=0x1400; // Set DMAReqEnab & DMAReqMode
        usbRegs->INDEX=index_save;
}

void enableCoreRxDMA(Uint16 endPoint) {
        Uint16 index_save;
        index_save=usbRegs->INDEX;
        usbRegs->INDEX=endPoint;
        usbRegs->RXCSR.PERI_RXCSR&=0x77FF;  // Clear AUTOCLEAR and DMAReqMode
        usbRegs->RXCSR.PERI_RXCSR|=0x2000;  // Set DMAReqEnab
        usbRegs->INDEX=index_save;
}

Uint32 readCompletionQueue(Uint16 queueNum) {
        Uint32 DescAddress;
        DescAddress=(Uint32)usbRegs->QCTRL[queueNum].CTRLD;
        DescAddress&=0xFFFFFFE0;
        return(DescAddress);
}

void disableCoreRxDMA(Uint16 endPoint) {
        Uint16 index_save;
        index_save=usbRegs->INDEX;
        usbRegs->INDEX=endPoint;
        usbRegs->RXCSR.PERI_RXCSR &= 0xDFFF;  // Clear DMAReqEnab
        usbRegs->INDEX=index_save;
}

void disableCoreTxDMA(Uint16 endPoint) {
```

Example 5. Programming the USB DMA Controller (continued)

```
        Uint16 index_save;
        index_save=usbRegs->INDEX;
        usbRegs->INDEX=endPoint;
        usbRegs->TXCSR.PERI_TXCSR&=0x7FFF;  // Clear AUTOSET
        usbRegs->TXCSR.PERI_TXCSR&=0xEBFF;  // Clear DMAReqEnab & DMAReqMode
        usbRegs->INDEX=index_save;
}

// sample peripheral code
Uint16 usbMain(void) {
        usb_init();
        usb_device_init();  // initialize usb core related vars: index, fifo, etc and DMA related
vars, data buff, rx/txSubmitQ, etc
        wait_for_reset();   // wait here until host performs a reset.
        cppiDmaInit();  // Init CPPI 4.1 DMA
// ****************** Initialize Receive Buffer Descriptors ***********************************
// Host is performing a transfer and the device is going to loop it back out.
// The example below (non commented part of the code) applies for a data transfer made of two 64
bytes packet.
// These two packets are treated as part of a single transfer for Non-Transparent DMA modes and
as two transfers for Transparent
//   DMA mode. This needs to be understood for the below to make sense, especially the last
Descriptor shown below.
// Initialize receive descriptors (HBDs)
        initSingleHPDorHBD(0,RECEIVE,BUFFER_DESC,rxCompQ);  //Usage:
initSingleHPDorHBD(descNum,dir,TypeOfDesc,returnQueue)
        if (DESCsetup==MULTIPLE_DESC_SETUP) {
              initSingleHPDorHBD(1,RECEIVE,BUFFER_DESC,rxCompQ);  //Usage:
initSingleHPDorHBD(descNum,dir,TypeOfDesc,returnQueue)
        //    initSingleHPDorHBD(2,RECEIVE,BUFFER_DESC,rxCompQ);  //Usage:
initSingleHPDorHBD(descNum,dir,TypeOfDesc,returnQueue)
        //    initSingleHPDorHBD(3,RECEIVE,BUFFER_DESC,rxCompQ);  //Usage:
initSingleHPDorHBD(descNum,dir,TypeOfDesc,returnQueue)

        //       :
        //       :
        }

// Last Receive Descriptor creation (this is the case where two or three Buffer Descriptors are
needed).
// Note again that this example is for a two packet data transfer.
        if (DMAmode!=TRANSPARENT) {   // Need to Create the Descriptor to be used for the Null
Packet.
            if (DESCsetup==SINGLE_DESC_SETUP)     // One additonal Rx Desc Needs to be Queued for
Handling Null Packet
                initSingleHPDorHBD(1,RECEIVE,BUFFER_DESC,rxCompQ);         //Usage:
initSingleHPDorHBD(descNum,dir,TypeOfDesc,returnQueue)
                        else
                                    initSingleHPDorHBD(2,RECEIVE,BUFFER_DESC,rxCompQ);
//Usage: initSingleHPDorHBD(descNum,dir,TypeOfDesc,returnQueue)
            }

// ****************** Initialize Transmit Packet and Buffer Descriptors************************
// See comment above to understand the reasons for the total number of descriptors.
// Initialize transmit descriptors (HPD and HBDs)
        //Initialize HPD Descriptor 16 for Transmit.
        initSingleHPDorHBD(16,TRANSMIT,PACKET_DESC,txCompQ);   //Usage:
initSingleHPDorHBD(descNum,dir,TypeOfDesc,returnQueue)
        if (DESCsetup==MULTIPLE_DESC_SETUP) {
                if(DMAmode==TRANSPARENT) {
                      initSingleHPDorHBD(17,TRANSMIT,PACKET_DESC,txCompQ);   //Usage:
initSingleHPDorHBD(descNum,dir,TypeOfDesc,returnQueue)
//                    initSingleHPDorHBD(18,TRANSMIT,PACKET_DESC,txCompQ);   //Usage:
initSingleHPDorHBD(descNum,dir,TypeOfDesc,returnQueue)
//                    initSingleHPDorHBD(19,TRANSMIT,PACKET_DESC,txCompQ);   //Usage:
initSingleHPDorHBD(descNum,dir,TypeOfDesc,returnQueue)
                }
                else {
                      initSingleHPDorHBD(17,TRANSMIT,BUFFER_DESC,txCompQ);    //Usage:
initSingleHPDorHBD(descNum,dir,TypeOfDesc,returnQueue)
```

Example 5. Programming the USB DMA Controller (continued)

```
//                    initSingleHPDorHBD(18,TRANSMIT,BUFFER_DESC,txCompQ);    //Usage:
initSingleHPDorHBD(descNum,dir,TypeOfDesc,returnQueue)
//                    initSingleHPDorHBD(19,TRANSMIT,BUFFER_DESC,txCompQ);    //Usage:
initSingleHPDorHBD(descNum,dir,TypeOfDesc,returnQueue)
                }
            }
// ****************** Submit Receive Buffer Descriptors ************************************
        //Submit Receive Descriptors
        qDesc2SubmitQ(rxSubmitQ,0);            //Queue Number, HPDdescriptorNumber
        if (DESCsetup==MULTIPLE_DESC_SETUP)
                qDesc2SubmitQ(rxSubmitQ,1);            //Queue Number, HPDdescriptorNumber

        if (DMAmode!=TRANSPARENT) {   // Need to Submit the Descriptor to be used for the Null
Packet.
                if (DESCsetup==SINGLE_DESC_SETUP)     // One additonal Rx Desc Needs to be Queued
for Handling Null Packet
                        qDesc2SubmitQ(rxSubmitQ,1);            //Queue Number, HPDdescriptorNumber
                else
                        qDesc2SubmitQ(rxSubmitQ,2);            //Queue Number, HPDdescriptorNumber
        }

        //Enable Rx DMA
        enableCoreRxDMA(endpoint);
// ****************** Submit Transmit Buffer Descriptors ************************************

// wait till all data is received here <<<<<<<


        //Submit Transmit Descriptors
        qDesc2SubmitQ(txSubmitQ,16);  //Queue Number, HPDdescriptorNumber
        if (DESCsetup==MULTIPLE_DESC_SETUP)
                qDesc2SubmitQ(txSubmitQ,17);  //Queue Number, HPDdescriptorNumber
//      qDesc2SubmitQ(txSubmitQ,18);  //Queue Number, HPDdescriptorNumber
        enableCoreTxDMA(endpoint);
// ******************************************************************************************

// wait till all data is received here <<<<<<<
```

4 Registers

Table 30 lists the memory-mapped registers for the universal serial bus OTG controller (USB0). See your device-specific data manual for the memory address of these registers. The base address is 01E0 0000h.

NOTE: In some cases, a single register address can have different names or meanings depending on the mode (host/peripheral) or the setting of the index register. The meaning of some bit fields varies with the mode.

Table 30. Universal Serial Bus OTG (USB0) Registers

VBUS Slave Address Offset	Acronym	Register Description	Section
0h	REVID	Revision Identification Register	Section 4.1
4h	CTRLR	Control Register	Section 4.2
8h	STATR	Status Register	Section 4.3
Ch	EMUR	Emulation Register	Section 4.4
10h	MODE	Mode Register	Section 4.5
14h	AUTOREQ	Autorequest Register	Section 4.6
18h	SRPFIXTIME	SRP Fix Time Register	Section 4.7
1Ch	TEARDOWN	Teardown Register	Section 4.8
20h	INTSRCR	USB Interrupt Source Register	Section 4.9
24h	INTSETR	USB Interrupt Source Set Register	Section 4.10
28h	INTCLRR	USB Interrupt Source Clear Register	Section 4.11
2Ch	INTMSKR	USB Interrupt Mask Register	Section 4.12
30h	INTMSKSETR	USB Interrupt Mask Set Register	Section 4.13
34h	INTMSKCLRR	USB Interrupt Mask Clear Register	Section 4.14
38h	INTMASKEDR	USB Interrupt Source Masked Register	Section 4.15
3Ch	EOIR	USB End of Interrupt Register	Section 4.16
50h	GENRNDISSZ1	Generic RNDIS Size EP1	Section 4.17
54h	GENRNDISSZ2	Generic RNDIS Size EP2	Section 4.18
58h	GENRNDISSZ3	Generic RNDIS Size EP3	Section 4.19
5Ch	GENRNDISSZ4	Generic RNDIS Size EP4	Section 4.20
		Common USB Registers	
400h	FADDR	Function Address Register	Section 4.21
401h	POWER	Power Management Register	Section 4.22
402h	INTRTX	Interrupt Register for Endpoint 0 plus Transmit Endpoints 1 to 4	Section 4.23
404h	INTRRX	Interrupt Register for Receive Endpoints 1 to 4	Section 4.24
406h	INTRTXE	Interrupt Enable Register for INTRTX	Section 4.25
408h	INTRRXE	Interrupt Enable Register for INTRRX	Section 4.26
40Ah	INTRUSB	Interrupt Register for Common USB Interrupts	Section 4.27
40Bh	INTRUSBE	Interrupt Enable Register for INTRUSB	Section 4.28
40Ch	FRAME	Frame Number Register	Section 4.29
40Eh	INDEX	Index Register for Selecting the Endpoint Status and Control Registers	Section 4.30
40Fh	TESTMODE	Register to Enable the USB 2.0 Test Modes	Section 4.31

Table 30. Universal Serial Bus OTG (USB0) Registers (continued)

VBUS Slave Address Offset	Acronym	Register Description	Section
		Indexed Registers These registers operate on the endpoint selected by the INDEX register	
410h	TXMAXP	Maximum Packet Size for Peripheral/Host Transmit Endpoint (Index register set to select Endpoints 1-4 only)	Section 4.32
412h	PERI_CSR0	Control Status Register for Endpoint 0 in Peripheral Mode. (Index register set to select Endpoint 0)	Section 4.33
	HOST_CSR0	Control Status Register for Endpoint 0 in Host Mode. (Index register set to select Endpoint 0)	Section 4.34
	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint. (Index register set to select Endpoints 1-4)	Section 4.35
	HOST_TXCSR	Control Status Register for Host Transmit Endpoint. (Index register set to select Endpoints 1-4)	Section 4.36
414h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint (Index register set to select Endpoints 1-4 only)	Section 4.37
416h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint. (Index register set to select Endpoints 1-4)	Section 4.38
	HOST_RXCSR	Control Status Register for Host Receive Endpoint. (Index register set to select Endpoints 1-4)	Section 4.39
418h	COUNT0	Number of Received Bytes in Endpoint 0 FIFO. (Index register set to select Endpoint 0)	Section 4.40
	RXCOUNT	Number of Bytes in Host Receive Endpoint FIFO. (Index register set to select Endpoints 1- 4)	Section 4.41
41Ah	HOST_TYPE0	Defines the speed of Endpoint 0	Section 4.42
	HOST_TXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Transmit endpoint. (Index register set to select Endpoints 1-4 only)	Section 4.43
41Bh	HOST_NAKLIMIT0	Sets the NAK response timeout on Endpoint 0. (Index register set to select Endpoint 0)	Section 4.44
	HOST_TXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Transmit endpoint. (Index register set to select Endpoints 1-4 only)	Section 4.45
41Ch	HOST_RXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Receive endpoint. (Index register set to select Endpoints 1-4 only)	Section 4.46
41Dh	HOST_RXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Receive endpoint. (Index register set to select Endpoints 1-4 only)	Section 4.47
41Fh	CONFIGDATA	Returns details of core configuration. (Index register set to select Endpoint 0)	Section 4.48
		FIFOn	
420h	FIFO0	Transmit and Receive FIFO Register for Endpoint 0	Section 4.49
424h	FIFO1	Transmit and Receive FIFO Register for Endpoint 1	Section 4.50
428h	FIFO2	Transmit and Receive FIFO Register for Endpoint 2	Section 4.51
42Ch	FIFO3	Transmit and Receive FIFO Register for Endpoint 3	Section 4.52
430h	FIFO4	Transmit and Receive FIFO Register for Endpoint 4	Section 4.53
		OTG Device Control	
460h	DEVCTL	Device Control Register	Section 4.54
		Dynamic FIFO Control	
462h	TXFIFOSZ	Transmit Endpoint FIFO Size (Index register set to select Endpoints 1-4 only)	Section 4.55
463h	RXFIFOSZ	Receive Endpoint FIFO Size (Index register set to select Endpoints 1-4 only)	Section 4.56
464h-465h	TXFIFOADDR	Transmit Endpoint FIFO Address (Index register set to select Endpoints 1-4 only)	Section 4.57

Table 30. Universal Serial Bus OTG (USB0) Registers (continued)

VBUS Slave Address Offset	Acronym	Register Description	Section
466h-467h	RXFIFOADDR	Receive Endpoint FIFO Address (Index register set to select Endpoints 1-4 only)	Section 4.58
46Ch-46Dh	HWVERS	Hardware Version Register	Section 4.59
		Target Endpoint 0 Control Registers, Valid Only in Host Mode	
480h	TXFUNCADDR	Address of the target function that has to be accessed through the associated Transmit Endpoint.	Section 4.60
482h	TXHUBADDR	Address of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.61
483h	TXHUBPORT	Port of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.62
484h	RXFUNCADDR	Address of the target function that has to be accessed through the associated Receive Endpoint.	Section 4.63
486h	RXHUBADDR	Address of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.64
487h	RXHUBPORT	Port of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.65
		Target Endpoint 1 Control Registers, Valid Only in Host Mode	
488h	TXFUNCADDR	Address of the target function that has to be accessed through the associated Transmit Endpoint.	Section 4.60
48Ah	TXHUBADDR	Address of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.61
48Bh	TXHUBPORT	Port of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.62
48Ch	RXFUNCADDR	Address of the target function that has to be accessed through the associated Receive Endpoint.	Section 4.63
48Eh	RXHUBADDR	Address of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.64
48Fh	RXHUBPORT	Port of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.65
		Target Endpoint 2 Control Registers, Valid Only in Host Mode	
490h	TXFUNCADDR	Address of the target function that has to be accessed through the associated Transmit Endpoint.	Section 4.60
492h	TXHUBADDR	Address of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.61
493h	TXHUBPORT	Port of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.62
494h	RXFUNCADDR	Address of the target function that has to be accessed through the associated Receive Endpoint.	Section 4.63
496h	RXHUBADDR	Address of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.64
497h	RXHUBPORT	Port of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.65

Table 30. Universal Serial Bus OTG (USB0) Registers (continued)

VBUS Slave Address Offset	Acronym	Register Description	Section
		Target Endpoint 3 Control Registers, Valid Only in Host Mode	
498h	TXFUNCADDR	Address of the target function that has to be accessed through the associated Transmit Endpoint.	Section 4.60
49Ah	TXHUBADDR	Address of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.61
49Bh	TXHUBPORT	Port of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.62
49Ch	RXFUNCADDR	Address of the target function that has to be accessed through the associated Receive Endpoint.	Section 4.63
49Eh	RXHUBADDR	Address of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.64
49Fh	RXHUBPORT	Port of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.65
		Target Endpoint 4 Control Registers, Valid Only in Host Mode	
4A0h	TXFUNCADDR	Address of the target function that has to be accessed through the associated Transmit Endpoint.	Section 4.60
4A2h	TXHUBADDR	Address of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.61
4A3h	TXHUBPORT	Port of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.62
4A4h	RXFUNCADDR	Address of the target function that has to be accessed through the associated Receive Endpoint.	Section 4.63
4A6h	RXHUBADDR	Address of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.64
4A7h	RXHUBPORT	Port of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.65
		Control and Status Register for Endpoint 0	
502h	PERI_CSR0	Control Status Register for Endpoint 0 in Peripheral Mode	Section 4.33
	HOST_CSR0	Control Status Register for Endpoint 0 in Host Mode	Section 4.34
508h	COUNT0	Number of Received Bytes in Endpoint 0 FIFO	Section 4.40
50Ah	HOST_TYPE0	Defines the Speed of Endpoint 0	Section 4.42
50Bh	HOST_NAKLIMIT0	Sets the NAK Response Timeout on Endpoint 0	Section 4.44
50Fh	CONFIGDATA	Returns details of core configuration	Section 4.48
		Control and Status Register for Endpoint 1	
510h	TXMAXP	Maximum Packet Size for Peripheral/Host Transmit Endpoint	Section 4.32
512h	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint (peripheral mode)	Section 4.35
	HOST_TXCSR	Control Status Register for Host Transmit Endpoint (host mode)	Section 4.36
514h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint	Section 4.37
516h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint (peripheral mode)	Section 4.38
	HOST_RXCSR	Control Status Register for Host Receive Endpoint (host mode)	Section 4.39
518h	RXCOUNT	Number of Bytes in Host Receive endpoint FIFO	Section 4.41

Table 30. Universal Serial Bus OTG (USB0) Registers (continued)

VBUS Slave Address Offset	Acronym	Register Description	Section
51Ah	HOST_TXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Transmit endpoint.	Section 4.43
51Bh	HOST_TXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Transmit endpoint.	Section 4.45
51Ch	HOST_RXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Receive endpoint.	Section 4.46
51Dh	HOST_RXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Receive endpoint.	Section 4.47
		Control and Status Register for Endpoint 2	
520h	TXMAXP	Maximum Packet Size for Peripheral/Host Transmit Endpoint	Section 4.32
522h	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint (peripheral mode)	Section 4.35
	HOST_TXCSR	Control Status Register for Host Transmit Endpoint (host mode)	Section 4.36
524h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint	Section 4.37
526h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint (peripheral mode)	Section 4.38
	HOST_RXCSR	Control Status Register for Host Receive Endpoint (host mode)	Section 4.39
528h	RXCOUNT	Number of Bytes in Host Receive endpoint FIFO	Section 4.41
52Ah	HOST_TXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Transmit endpoint.	Section 4.43
52Bh	HOST_TXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Transmit endpoint.	Section 4.45
52Ch	HOST_RXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Receive endpoint.	Section 4.46
52Dh	HOST_RXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Receive endpoint.	Section 4.47
		Control and Status Register for Endpoint 3	
530h	TXMAXP	Maximum Packet Size for Peripheral/Host Transmit Endpoint	Section 4.32
532h	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint (peripheral mode)	Section 4.35
	HOST_TXCSR	Control Status Register for Host Transmit Endpoint (host mode)	Section 4.36
534h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint	Section 4.37
536h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint (peripheral mode)	Section 4.38
	HOST_RXCSR	Control Status Register for Host Receive Endpoint (host mode)	Section 4.39
538h	RXCOUNT	Number of Bytes in Host Receive endpoint FIFO	Section 4.41
53Ah	HOST_TXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Transmit endpoint.	Section 4.43
53Bh	HOST_TXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Transmit endpoint.	Section 4.45
53Ch	HOST_RXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Receive endpoint.	Section 4.46
53Dh	HOST_RXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Receive endpoint.	Section 4.47

Table 30. Universal Serial Bus OTG (USB0) Registers (continued)

VBUS Slave Address Offset	Acronym	Register Description	Section
		Control and Status Register for Endpoint 4	
540h	TXMAXP	Maximum Packet Size for Peripheral/Host Transmit Endpoint	Section 4.32
542h	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint (peripheral mode)	Section 4.35
	HOST_TXCSR	Control Status Register for Host Transmit Endpoint (host mode)	Section 4.36
544h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint	Section 4.37
546h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint (peripheral mode)	Section 4.38
	HOST_RXCSR	Control Status Register for Host Receive Endpoint (host mode)	Section 4.39
548h	RXCOUNT	Number of Bytes in Host Receive endpoint FIFO	Section 4.41
54Ah	HOST_TXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Transmit endpoint.	Section 4.43
54Bh	HOST_TXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Transmit endpoint.	Section 4.45
54Ch	HOST_RXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Receive endpoint.	Section 4.46
54Dh	HOST_RXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Receive endpoint.	Section 4.47
		CDMA Registers	
1000h	DMAREVID	CDMA Revision Identification Register	Section 4.66
1004h	TDFDQ	CDMA Teardown Free Descriptor Queue Control Register	Section 4.67
1008h	DMAEMU	CDMA Emulation Control Register	Section 4.68
1800h	TXGCR[0]	Transmit Channel 0 Global Configuration Register	Section 4.69
1808h	RXGCR[0]	Receive Channel 0 Global Configuration Register	Section 4.70
180Ch	RXHPCRA[0]	Receive Channel 0 Host Packet Configuration Register A	Section 4.71
1810h	RXHPCRB[0]	Receive Channel 0 Host Packet Configuration Register B	Section 4.72
1820h	TXGCR[1]	Transmit Channel 1 Global Configuration Register	Section 4.69
1828h	RXGCR[1]	Receive Channel 1 Global Configuration Register	Section 4.70
182Ch	RXHPCRA[1]	Receive Channel 1 Host Packet Configuration Register A	Section 4.71
1830h	RXHPCRB[1]	Receive Channel 1 Host Packet Configuration Register B	Section 4.72
1840h	TXGCR[2]	Transmit Channel 2 Global Configuration Register	Section 4.69
1848h	RXGCR[2]	Receive Channel 2 Global Configuration Register	Section 4.70
184Ch	RXHPCRA[2]	Receive Channel 2 Host Packet Configuration Register A	Section 4.71
1850h	RXHPCRB[2]	Receive Channel 2 Host Packet Configuration Register B	Section 4.72
1860h	TXGCR[3]	Transmit Channel 3 Global Configuration Register	Section 4.69
1868h	RXGCR[3]	Receive Channel 3 Global Configuration Register	Section 4.70
186Ch	RXHPCRA[3]	Receive Channel 3 Host Packet Configuration Register A	Section 4.71
1870h	RXHPCRB[3]	Receive Channel 3 Host Packet Configuration Register B	Section 4.72
2000h	DMA_SCHED_CTRL	CDMA Scheduler Control Register	Section 4.73
2800h-28FCh	WORD[0]-WORD[63]	CDMA Scheduler Table Word 0-63 Registers	Section 4.74

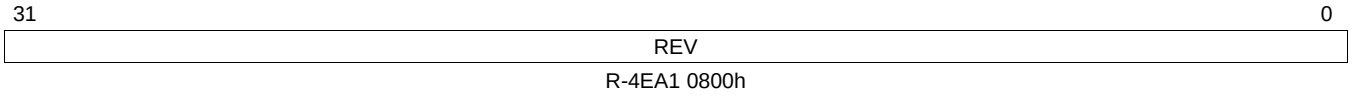
Table 30. Universal Serial Bus OTG (USB0) Registers (continued)

VBUS Slave Address Offset	Acronym	Register Description	Section
		Queue Manager (QMGR) Registers	
4000h	QMGRREVID	QMGR Revision Identification Register	Section 4.75
4008h	DIVERSION	QMGR Queue Diversion Register	Section 4.76
4020h	FDBSC0	QMGR Free Descriptor/Buffer Starvation Count Register 0	Section 4.77
4024h	FDBSC1	QMGR Free Descriptor/Buffer Starvation Count Register 1	Section 4.78
4028h	FDBSC2	QMGR Free Descriptor/Buffer Starvation Count Register 2	Section 4.79
402Ch	FDBSC3	QMGR Free Descriptor/Buffer Starvation Count Register 3	Section 4.80
4080h	LRAM0BASE	QMGR Linking RAM Region 0 Base Address Register	Section 4.81
4084h	LRAM0SIZE	QMGR Linking RAM Region 0 Size Register	Section 4.82
4088h	LRAM1BASE	QMGR Linking RAM Region 1 Base Address Register	Section 4.83
4090h	PEND0	QMGR Queue Pending Register 0	Section 4.84
4094h	PEND1	QMGR Queue Pending Register 1	Section 4.85
5000h + 16 × *R*	QMEMRBASE[*R*]	QMGR Memory Region *R* Base Address Register (*R* = 0 to 15)	Section 4.86
5004h + 16 × *R*	QMEMRCTRL[*R*]	QMGR Memory Region *R* Control Register (*R* = 0 to 15)	Section 4.87
600Ch + 16 × *N*	CTRLD[*N*]	QMGR Queue *N* Control Register D (*N* = 0 to 63)	Section 4.88
6800h + 16 × *N*	QSTATA[*N*]	QMGR Queue *N* Status Register A (*N* = 0 to 63)	Section 4.89
6804h + 16 × *N*	QSTATB[*N*]	QMGR Queue *N* Status Register B (*N* = 0 to 63)	Section 4.90
6808h + 16 × *N*	QSTATC[*N*]	QMGR Queue *N* Status Register C (*N* = 0 to 63)	Section 4.91

4.1 Revision Identification Register (REVID)

The revision identification register (REVID) contains the revision for the USB 2.0 OTG controller module. The REVID is shown in Figure 27 and described in Table 31.

Figure 27. Revision Identification Register (REVID)



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

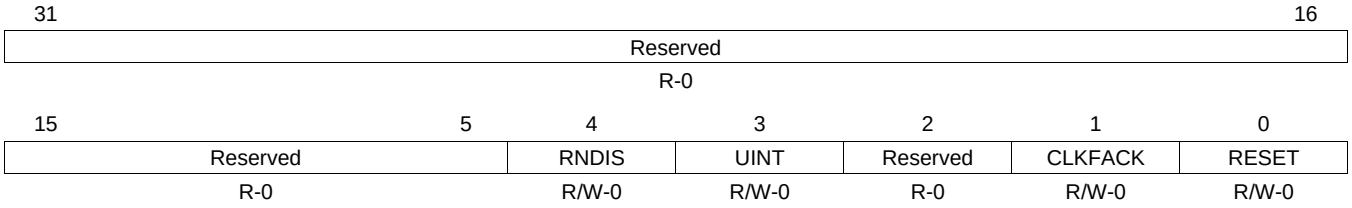
Table 31. Revision Identification Register (REVID) Field Descriptions

Bit	Field	Value	Description
31-0	REV	4EA1 0800h	Revision ID of the USB module.

4.2 Control Register (CTRLR)

The control register (CTRLR) allows the CPU to control various aspects of the module. The CTRLR is shown in Figure 28 and described in Table 32.

Figure 28. Control Register (CTRLR)



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

Table 32. Control Register (CTRLR) Field Descriptions

Bit	Field	Value	Description
31-5	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. The USB controller will clear this bit when it completes reset.

4.3 Status Register (STATR)

The status register (STATR) allows the CPU to check various aspects of the module. The STATR is shown in Figure 29 and described in Table 33.

Figure 29. Status Register (STATR)

31	16
Reserved	
R-0	

15	1	0
Reserved		DRVVBUS
R-0		R-0

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

Table 33. Status Register (STATR) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	DRVVBUS		Current DRVVBUS value.
		0	DRVVBUS value is logic 0.
		1	DRVVBUS value is logic 1.

4.4 Emulation Register (EMUR)

The emulation register (EMUR) allows the CPU to configure the CBA 3.0 emulation interface. The EMUR is shown in Figure 30 and described in Table 34.

Figure 30. Emulation Register (EMUR)

31	16
Reserved	
R-0	

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 34. Emulation Register (EMUR) Field Descriptions

Bit	Field	Value	Description
31-3	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

4.5 Mode Register (MODE)

The mode register (MODE) allows 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 MODE is shown in Figure 31 and described in Table 35.

Figure 31. Mode Register (MODE)

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16
Reserved	RX4_MODE	Reserved	RX3_MODE	Reserved	RX2_MODE	Reserved	RX1_MODE
R-0	R/W-0	R-0	R/W-0	R-0	R/W-0	R-0	R/W-0
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-0	R/W-0	R-0	R/W-0	R-0	R/W-0	R-0	R/W-0

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

Table 35. Mode Register (MODE) Field Descriptions

Bit	Field	Value	Description
31-30	Reserved	0	Reserved
29-28	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
27-26	Reserved	0	Reserved
25-24	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
23-22	Reserved	0	Reserved
21-20	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
19-18	Reserved	0	Reserved
17-16	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
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

Table 35. Mode Register (MODE) Field Descriptions (continued)

Bit	Field	Value	Description
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

4.6 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 32 and described in Table 36.

Figure 32. Auto Request Register (AUTOREQ)

31	16
Reserved	
R-0	

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 36. Auto Request Register (AUTOREQ) Field Descriptions

Bit	Field	Value	Description
31-8	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

4.7 SRP Fix Time Register (SRPFIXTIME)

The SRP fix time register (SRPFIXTIME) allows the CPU to configure the maximum amount of time the SRP fix logic blocks the Avalid from the PHY to the OTG core. The SRPFIXTIME is shown in Figure 33 and described in Table 37.

Figure 33. SRP Fix Time Register (SRPFIXTIME)

31	0
SRPFIXTIME	
R/W-280 DE80h	

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

Table 37. SRP Fix Time Register (SRPFIXTIME) Field Descriptions

Bit	Field	Value	Description
31-0	SRPFIXTIME	0-FFFF FFFFh	SRP fix maximum time in 60 MHz cycles. Default is 700 ms (280 DE80h).

4.8 Teardown Register (TEARDOWN)

The teardown register (TEARDOWN) controls the tearing down of receive and transmit FIFOs in the USB controller. When a 1 is written to a valid bit in TEARDOWN, the CPPI FIFO pointers for that endpoint are cleared. TEARDOWN 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 TEARDOWN is shown in Figure 34 and described in Table 38.

Figure 34. Teardown Register (TEARDOWN)

31	21	20	17	16
Reserved		TX_TDOWN		Rsvd
R-0		R/W-0		R-0

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

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

Table 38. Teardown Register (TEARDOWN) Field Descriptions

Bit	Field	Value	Description
31-21	Reserved	0	Reserved
20-17	TX_TDOWN		Transmit endpoint teardown. Set the bit that corresponds to the Endpoint (for EP1, set bit 17; for EP2, set bit 18; for EP3, set bit 19; for EP4, set bit 20).
		0	Disable
		1	Enable
16-5	Reserved	0	Reserved
4-1	RX_TDOWN		Receive endpoint teardown
		0	Disable
		1	Enable
0	Reserved	0	Reserved

4.9 USB Interrupt Source Register (INTSRCR)

The USB interrupt source register (INTSRCR) contains the status of the interrupt sources generated by the USB core (not by the DMA). The INTSRCR is shown in Figure 35 and described in Table 39.

NOTE: Other than the USB bit field, to make use of INTSRCR, the PDR interrupt handler must be enabled (the UINT bit in the control register (CTRLR) is cleared to 0). If the UINT bit in CTRLR is set to 1, you need to use the interrupt status/flag from the core register space.

Figure 35. USB Interrupt Source Register (INTSRCR)

31	25	24	16
Reserved		USB	
R-0		R-0	

15	13	12	9	8	5	4	1	0
Reserved		RXEP[*n*]		Reserved		TXEP[*n*]		EP0
R-0		R-0		R-0		R-0		R-0

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

Table 39. USB Interrupt Source Register (INTSRCR) Field Descriptions

Bit	Field	Value	Description
31-25	Reserved	0	Reserved
24-16	USB	0-1FFh	USB interrupt sources. Generated by the USB core (not by the DMA). Note: INTRUSB core interrupts are mapped onto bits 23-16 and bit 24 is the USBDRVVBUS interrupt status.
15-13	Reserved	0	Reserved
12-9	RXEP[*n*]		Receive endpoint *n* interrupt source.
		0	RXEP*n* interrupt is not generated by the USB core.
		1	RXEP*n* interrupt is generated by the USB core (not by the DMA).
8-5	Reserved	0	Reserved
4-1	TXEP[*n*]		Transmit endpoint *n* interrupt source.
		0	TXEP*n* interrupt is not generated by the USB core.
		1	TXEP*n* interrupt is generated by the USB core (not by the DMA).
0	EP0		Endpoint 0 interrupt source.
		0	Endpoint 0 interrupt is not generated by the USB core.
		1	Endpoint 0 interrupt is generated by the USB core (not by the DMA).

4.10 USB Interrupt Source Set Register (INTSETR)

The USB interrupt source set register (INTSETR) allows the USB interrupt sources to be manually triggered. A read of this register returns the USB interrupt source register value. The INTSETR is shown in Figure 36 and described in Table 40.

NOTE: Other than the USB bit field, to make use of INTSETR, the PDR interrupt handler must be enabled (the UINT bit in the control register (CTRLR) is cleared to 0). If the UINT bit in CTRLR is set to 1, you need to use the interrupt status/flag from the core register space.

Figure 36. USB Interrupt Source Set Register (INTSETR)

31–25	24–16
Reserved	USB
R-0	R/W-0

15–13	12–9	8–5	4–1	0
Reserved	RXEP[*n*]	Reserved	TXEP[*n*]	EP0
R-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 40. USB Interrupt Source Set Register (INTSETR) Field Descriptions

Bit	Field	Value	Description
31-25	Reserved	0	Reserved
24-16	USB	0-1FFh	Write a 1 to set equivalent USB interrupt source. Allows the USB interrupt sources to be manually triggered.
15-13	Reserved	0	Reserved
12-9	RXEP[*n*]		Set receive endpoint *n* interrupt source. Allows the receive endpoint *n* interrupt sources to be manually triggered.
		0	RXEP*n* interrupt is not set.
		1	RXEP*n* interrupt is set.
8-5	Reserved	0	Reserved
4-1	TXEP[*n*]		Set transmit endpoint *n* interrupt source. Allows the transmit endpoint *n* interrupt sources to be manually triggered.
		0	TXEP*n* interrupt is not set.
		1	TXEP*n* interrupt is set.
0	EP0		Set endpoint 0 interrupt source. Allows the endpoint 0 interrupt source to be manually triggered.
		0	Endpoint 0 interrupt is not set.
		1	Endpoint 0 interrupt is set.

4.11 USB Interrupt Source Clear Register (INTCLRR)

The USB interrupt source clear register (INTCLRR) allows the CPU to acknowledge an interrupt source and turn it off. A read of this register returns the USB interrupt source register value. The INTCLRR is shown in Figure 37 and described in Table 41.

NOTE: Other than the USB bit field, to make use of INTCLRR, the PDR interrupt handler must be enabled (the UINT bit in the control register (CTRLR) is cleared to 0). If the UINT bit in CTRLR is set to 1, you need to use the interrupt status/flag from the core register space.

Figure 37. USB Interrupt Source Clear Register (INTCLRR)

31	25	24	16
Reserved		USB	
R-0		R/W-0	

15	13	12	9	8	5	4	1	0
Reserved		RXEP[*n*]		Reserved		TXEP[*n*]		EP0
R-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 41. USB Interrupt Source Clear Register (INTCLRR) Field Descriptions

Bit	Field	Value	Description
31-25	Reserved	0	Reserved
24-16	USB	0-1FFh	Write a 1 to clear equivalent USB interrupt source. Allows the CPU to acknowledge a USB interrupt source and turn it off.
15-13	Reserved	0	Reserved
12-9	RXEP[*n*]		Clear receive endpoint *n* interrupt source. Allows the CPU to acknowledge a receive endpoint *n* interrupt source and turn it off.
		0	RXEP*n* interrupt is not cleared.
		1	RXEP*n* interrupt is cleared.
8-5	Reserved	0	Reserved
4-1	TXEP[*n*]		Clear transmit endpoint *n* interrupt source. Allows the CPU to acknowledge a transmit endpoint *n* interrupt source and turn it off.
		0	TXEP*n* interrupt is not cleared.
		1	TXEP*n* interrupt is cleared.
0	EP0		Clear endpoint 0 interrupt source. Allows the CPU to acknowledge the endpoint 0 interrupt source and turn it off.
		0	Endpoint 0 interrupt is not cleared.
		1	Endpoint 0 interrupt is cleared.

4.12 USB Interrupt Mask Register (INTMSKR)

The USB interrupt mask register (INTMSKR) contains the masks of the interrupt sources generated by the USB core (not by 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 (INTSRCR).

The INTMSKR is shown in Figure 38 and described in Table 42.

NOTE: Other than the USB bit field, to make use of INTMSKR, the PDR interrupt handler must be enabled (the UINT bit in the control register (CTRLR) is cleared to 0). If the UINT bit in CTRLR is set to 1, you need to use the interrupt status/flag from the core register space.

Figure 38. USB Interrupt Mask Register (INTMSKR)

31	25	24	16
Reserved		USB	
R-0		R-0	

15	13	12	9	8	5	4	1	0
Reserved		RXEP[*n*]		Reserved		TXEP[*n*]		EP0
R-0		R-0		R-0		R-0		R-0

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

Table 42. USB Interrupt Mask Register (INTMSKR) Field Descriptions

Bit	Field	Value	Description
31-25	Reserved	0	Reserved
24-16	USB	0-1FFh	USB interrupt source masks. Generated by the USB core (not by the DMA).
15-13	Reserved	0	Reserved
12-9	RXEP[*n*]		Receive endpoint *n* interrupt source mask.
		0	RXEP*n* interrupt mask is not generated by the USB core.
		1	RXEP*n* interrupt mask is generated by the USB core (not by the DMA).
8-5	Reserved	0	Reserved
4-1	TXEP[*n*]		Transmit endpoint *n* interrupt source mask.
		0	TXEP*n* interrupt mask is not generated by the USB core.
		1	TXEP*n* interrupt mask is generated by the USB core (not by the DMA).
0	EP0		Endpoint 0 interrupt source mask.
		0	Endpoint 0 interrupt mask is not generated by the USB core.
		1	Endpoint 0 interrupt mask is generated by the USB core (not by the DMA).

4.13 USB Interrupt Mask Set Register (INTMSKSETR)

The USB interrupt mask set register (INTMSKSETR) allows the USB masks to be individually enabled. A read to this register returns the USB interrupt mask register value. The INTMSKSETR is shown in Figure 39 and described in Table 43.

NOTE: Other than the USB bit field, to make use of INTMSKSETR, the PDR interrupt handler must be enabled (the UINT bit in the control register (CTRLR) is cleared to 0). If the UINT bit in CTRLR is set to 1, you need to use the interrupt status/flag from the core register space.

Figure 39. USB Interrupt Mask Set Register (INTMSKSETR)

31	25	24	16
Reserved		USB	
R-0		R/W-0	

15	13	12	9	8	5	4	1	0
Reserved		RXEP[*n*]		Reserved		TXEP[*n*]		EP0
R-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 43. USB Interrupt Mask Set Register (INTMSKSETR) Field Descriptions

Bit	Field	Value	Description
31-25	Reserved	0	Reserved
24-16	USB	0-1FFh	Write a 1 to set equivalent USB interrupt source mask. Allows the USB interrupt source masks to be manually enabled.
15-13	Reserved	0	Reserved
12-9	RXEP[*n*]		Set receive endpoint *n* interrupt source mask. Allows the receive endpoint *n* interrupt source masks to be manually enabled.
		0	RXEP*n* interrupt mask is not enabled.
		1	RXEP*n* interrupt mask is enabled.
8-5	Reserved	0	Reserved
4-1	TXEP[*n*]		Set transmit endpoint *n* interrupt source mask. Allows the transmit endpoint *n* interrupt source masks to be manually enabled.
		0	TXEP*n* interrupt mask is not enabled.
		1	TXEP*n* interrupt mask is enabled.
0	EP0		Set endpoint 0 interrupt source mask. Allows the endpoint 0 interrupt source mask to be manually enabled.
		0	Endpoint 0 interrupt mask is not enabled.
		1	Endpoint 0 interrupt mask is enabled.

4.14 USB Interrupt Mask Clear Register (INTMSKCLRR)

The USB interrupt mask clear register (INTMSKCLRR) allows the USB interrupt masks to be individually disabled. A read to this register returns the USB interrupt mask register value. The INTMSKCLRR is shown in Figure 40 and described in Table 44.

NOTE: Other than the USB bit field, to make use of INTMSKCLRR, the PDR interrupt handler must be enabled (the UINT bit in the control register (CTRLR) is cleared to 0). If the UINT bit in CTRLR is set to 1, you need to use the interrupt status/flag from the core register space.

Figure 40. USB Interrupt Mask Clear Register (INTMSKCLRR)

31	25	24	16
Reserved		USB	
R-0		R/W-0	

15	13	12	9	8	5	4	1	0
Reserved		RXEP[*n*]		Reserved		TXEP[*n*]		EP0
R-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 44. USB Interrupt Mask Clear Register (INTMSKCLRR) Field Descriptions

Bit	Field	Value	Description
31-25	Reserved	0	Reserved
24-16	USB	0-1FFh	Write a 1 to clear equivalent USB interrupt source mask. Allows the USB interrupt source masks to be manually disabled.
15-13	Reserved	0	Reserved
12-9	RXEP[*n*]		Clear receive endpoint *n* interrupt source mask. Allows the receive endpoint *n* interrupt source masks to be manually disabled.
		0	RXEP*n* interrupt mask is not disabled.
		1	RXEP*n* interrupt mask is disabled.
8-5	Reserved	0	Reserved
4-1	TXEP[*n*]		Clear transmit endpoint *n* interrupt source mask. Allows the transmit endpoint *n* interrupt source masks to be manually disabled.
		0	TXEP*n* interrupt mask is not disabled.
		1	TXEP*n* interrupt mask is disabled.
0	EP0		Clear endpoint 0 interrupt source mask. Allows the endpoint 0 interrupt source mask to be manually disabled.
		0	Endpoint 0 interrupt mask is not disabled.
		1	Endpoint 0 interrupt mask is disabled.

4.15 USB Interrupt Source Masked Register (INTMASKEDR)

The USB interrupt source masked register (INTMASKEDR) contains the status of the interrupt sources generated by the USB core masked by the USB interrupt mask register (INTMSKR) values. The INTMASKEDR is shown in Figure 41 and described in Table 45.

NOTE: Other than the USB bit field, to make use of INTMASKEDR, the PDR interrupt handler must be enabled (the UINT bit in the control register (CTRLR) is cleared to 0). If the UINT bit in CTRLR is set to 1, you need to use the interrupt status/flag from the core register space.

Figure 41. USB Interrupt Source Masked Register (INTMASKEDR)

31–25	24–16
Reserved	USB
R-0	R-0

15–13	12–9	8–5	4–1	0
Reserved	RXEP[*n*]	Reserved	TXEP[*n*]	EP0
R-0	R-0	R-0	R-0	R-0

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

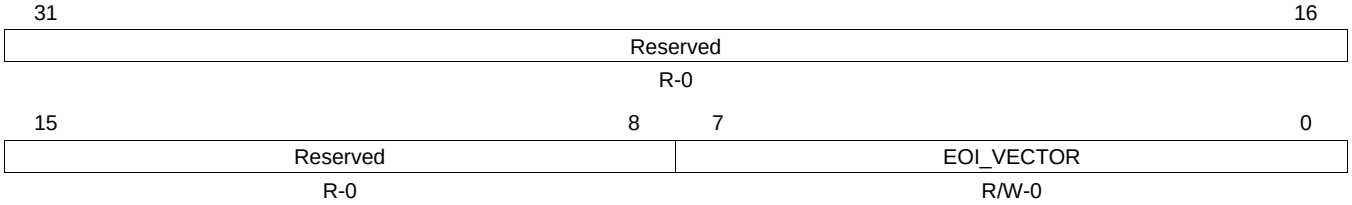
Table 45. USB Interrupt Source Masked Register (INTMASKEDR) Field Descriptions

Bit	Field	Value	Description
31-25	Reserved	0	Reserved
24-16	USB	0-1FFh	USB interrupt sources masked.
15-13	Reserved	0	Reserved
12-9	RXEP[*n*]		Receive endpoint *n* interrupt source masked.
		0	RXEP*n* interrupt source is not masked.
		1	RXEP*n* interrupt source is masked.
8-5	Reserved	0	Reserved
4-1	TXEP[*n*]		Transmit endpoint *n* interrupt source masked.
		0	TXEP*n* interrupt source is not masked.
		1	TXEP*n* interrupt source is masked.
0	EP0		Endpoint 0 interrupt source masked.
		0	Endpoint 0 interrupt source is not masked.
		1	Endpoint 0 interrupt source is masked.

4.16 USB End of Interrupt Register (EOIR)

The USB end of interrupt register (EOIR) allows the CPU to acknowledge completion of a non-DMA interrupt by writing 0 to the EOI_VECTOR field. The EOIR is shown in Figure 42 and described in Table 46.

Figure 42. USB End of Interrupt Register (EOIR)



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

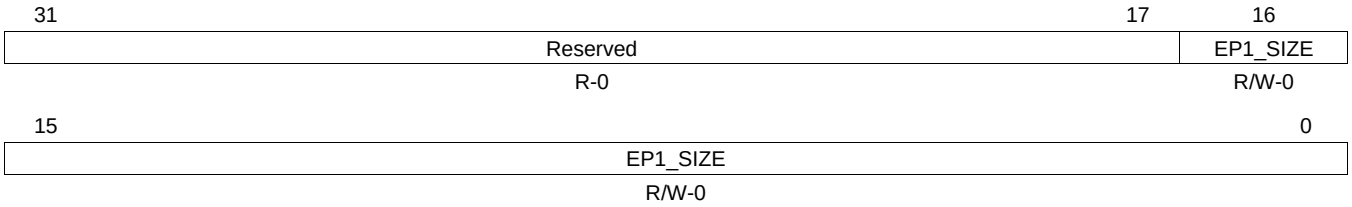
Table 46. USB End of Interrupt Register (EOIR) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7-0	EOI_VECTOR	0-FFh	End of Interrupt (EOI) Vector.

4.17 Generic RNDIS EP1 Size Register (GENRNDISSZ1)

The generic RNDIS EP1 size register (GENRNDISSZ1) is 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. This register must be programmed with a value that is an integer multiple of the endpoint size. The maximum value this register can be programmed with is 10000h, or 65536. The GENRNDISSZ1 is shown in Figure 43 and described in Table 47.

Figure 43. Generic RNDIS EP1 Size Register (GENRNDISSZ1)



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

Table 47. Generic RNDIS EP1 Size Register (GENRNDISSZ1) Field Descriptions

Bit	Field	Value	Description
31-17	Reserved	0	Reserved
16-0	EP1_SIZE	0-10000h	Generic RNDIS packet size

4.18 Generic RNDIS EP2 Size Register (GENRNDISSZ2)

The generic RNDIS EP2 size register (GENRNDISSZ2) is 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. This register must be programmed with a value that is an integer multiple of the endpoint size. The maximum value this register can be programmed with is 10000h, or 65536. The GENRNDISSZ2 is shown in Figure 44 and described in Table 48.

Figure 44. Generic RNDIS EP2 Size Register (GENRNDISSZ2)

31	17	16
Reserved		EP2_SIZE
R-0		R/W-0

15	0
EP2_SIZE	
R/W-0	

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

Table 48. Generic RNDIS EP2 Size Register (GENRNDISSZ2) Field Descriptions

Bit	Field	Value	Description
31-17	Reserved	0	Reserved
16-0	EP2_SIZE	0-10000h	Generic RNDIS packet size

4.19 Generic RNDIS EP3 Size Register (GENRNDISSZ3)

The generic RNDIS EP3 size register (GENRNDISSZ3) is 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. This register must be programmed with a value that is an integer multiple of the endpoint size. The maximum value this register can be programmed with is 10000h, or 65536. The GENRNDISSZ3 is shown in Figure 45 and described in Table 49.

Figure 45. Generic RNDIS EP3 Size Register (GENRNDISSZ3)

31	17	16
Reserved		EP3_SIZE
R-0		R/W-0

15	0
EP3_SIZE	
R/W-0	

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

Table 49. Generic RNDIS EP3 Size Register (GENRNDISSZ3) Field Descriptions

Bit	Field	Value	Description
31-17	Reserved	0	Reserved
16-0	EP3_SIZE	0-10000h	Generic RNDIS packet size

4.20 Generic RNDIS EP4 Size Register (GENRNDISSZ4)

The generic RNDIS EP4 size register (GENRNDISSZ4) is 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. This register must be programmed with a value that is an integer multiple of the endpoint size. The maximum value this register can be programmed with is 10000h, or 65536. The GENRNDISSZ4 is shown in Figure 46 and described in Table 50.

Figure 46. Generic RNDIS EP4 Size Register (GENRNDISSZ4)

31	17	16
Reserved		EP4_SIZE
R-0		R/W-0

15	0
EP4_SIZE	
R/W-0	

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

Table 50. Generic RNDIS EP4 Size Register (GENRNDISSZ4) Field Descriptions

Bit	Field	Value	Description
31-17	Reserved	0	Reserved
16-0	EP4_SIZE	0-10000h	Generic RNDIS packet size

4.21 Function Address Register (FADDR)

The function address register (FADDR) is shown in Figure 47 and described in Table 51.

Figure 47. Function Address Register (FADDR)

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

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

Table 51. 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. When used in Peripheral mode, 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.

4.22 Power Management Register (POWER)

The power management register (POWER) is shown in Figure 48 and described in Table 52.

Figure 48. 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/W-0	R/W-0	R/W-0	R/W-0

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

Table 52. 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. Note: this is only valid in Peripheral Mode. 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. Note: this is only valid in Peripheral Mode.
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/Write in Host Mode, but read-only in Peripheral Mode.
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	In Host mode, this bit should be set to enter Suspend mode. In Peripheral mode, 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.

4.23 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 49 and described in Table 53.

NOTE: Unless the UINT bit in the control register (CTRLR) is set to 1 (non-PDR interrupt mode is enabled), do not read this register directly. Performing a read clears the pending interrupt. Use INTRTX only when in the non-PDR interrupt mode, that is, when handling the interrupt directly from the controller.

Figure 49. 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 53. 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

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

The interrupt register for receive endpoints 1 to 4 (INTRRX) is shown in Figure 50 and described in Table 54.

NOTE: Unless the UINT bit in the control register (CTRLR) is set to 1 (non-PDR interrupt mode is enabled), do not read this register directly. Performing a read clears the pending interrupt. Use INTRRX only when in the non-PDR interrupt mode, that is, when handling the interrupt directly from the controller.

Figure 50. 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	Rsvd
R-0		R-0	R-0	R-0	R-0	R-0

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

Table 54. 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

4.25 Interrupt Enable Register for INTRTX (INTRTXE)

The interrupt enable register for INTRTX (INTRTXE) is shown in Figure 51 and described in Table 55.

Figure 51. 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 55. 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

4.26 Interrupt Enable Register for INTRRX (INTRRXE)

The interrupt enable register for INTRRX (INTRRXE) is shown in Figure 52 and described in Table 56.

Figure 52. 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 56. 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

4.27 Interrupt Register for Common USB Interrupts (INTRUSB)

The interrupt register for common USB interrupts (INTRUSB) is shown in Figure 53 and described in Table 57.

NOTE: Unless the UINT bit in the control register (CTRLR) is set to 1 (non-PDR interrupt mode is enabled), do not read this register directly. Performing a read clears the pending interrupt. Use INTRUSB only when in the non-PDR interrupt mode, that is, when handling the interrupt directly from the controller.

Figure 53. 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 57. 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 in host mode when a device disconnect is detected. Set in peripheral mode 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 in peripheral mode when reset signaling is detected on the bus set in host mode when babble is detected.
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 only valid in peripheral mode.

4.28 Interrupt Enable Register for INTRUSB (INTRUSBE)

The interrupt enable register for INTRUSB (INTRUSBE) is shown in Figure 54 and described in Table 58.

Figure 54. 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 58. 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

4.29 Frame Number Register (FRAME)

The frame number register (FRAME) is shown in Figure 55 and described in Table 59.

Figure 55. Frame Number Register (FRAME)

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

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

Table 59. Frame Number Register (FRAME) Field Descriptions

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

4.30 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 56 and described in Table 60.

Figure 56. 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 60. 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	0-4h	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.

4.31 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 57 and described in Table 61.

Figure 57. 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

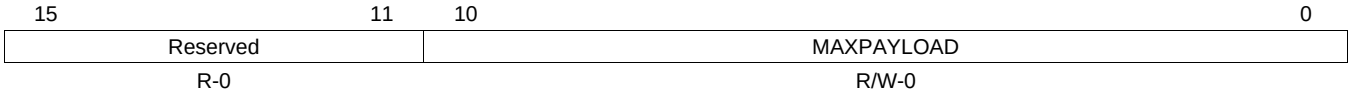
Table 61. Register to Enable the USB 2.0 Test Modes (TESTMODE) Field Descriptions

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.

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

The maximum packet size for peripheral/host transmit endpoint (TXMAXP) is shown in Figure 58 and described in Table 62.

Figure 58. Maximum Packet Size for Peripheral/Host Transmit Endpoint (TXMAXP)



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

Table 62. Maximum Packet Size for Peripheral/Host Transmit Endpoint (TXMAXP) Field Descriptions

Bit	Field	Value	Description
15-11	Reserved	0	Reserved
10-0	MAXPAYLOAD	0-400h	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.

4.33 Control Status Register for Endpoint 0 in Peripheral Mode (PERI_CSR0)

The control status register for endpoint 0 in peripheral mode (PERI_CSR0) is shown in Figure 59 and described in Table 63.

Figure 59. Control Status Register for Endpoint 0 in Peripheral Mode (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 63. Control Status Register for Endpoint 0 in Peripheral Mode (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.

4.34 Control Status Register for Endpoint 0 in Host Mode (HOST_CSR0)

The control status register for endpoint 0 in host mode (HOST_CSR0) is shown in Figure 60 and described in Table 64.

Figure 60. Control Status Register for Endpoint 0 in Host Mode (HOST_CSR0)

15			12	11	10	9	8
Reserved				DISPING	DATATOGWREN	DATATOG	FLUSHFIFO
R-0				R/W-0	W-0	R/W-0	W-0
7	6	5	4	3	2	1	0
NAK_TIMEOUT	STATUSPKT	REQPKT	ERROR	SETUPPKT	RXSTALL	TXPKTRDY	RXPKTRDY
W-0	R/W-0	R/W-0	W-0	R/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 64. Control Status Register for Endpoint 0 in Host Mode (HOST_CSR0) Field Descriptions

Bit	Field	Value	Description
15-12	Reserved	0	Reserved
11	DISPING	0-1	The CPU writes a 1 to the DSPING bit to instruct the core not to issue PING tokens in the data and status phases of a high-speed control transfer (for use with devices that do not respond to PING).
10	DATATOGWREN	0-1	Write 1 to this bit to enable the DATATOG bit to be written. This bit is automatically cleared once the new value is written to DATATOG.
9	DATATOG	0-1	When read, this bit indicates the current state of the EP0 data toggle. If DATATOGWREN is high, this bit can be written with the required setting of the data toggle. If DATATOGWREN is low, any value written to this bit is ignored.
8	FLUSHFIFO	0-1	Write 1 to 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	NAK_TIMEOUT	0-1	This bit will be set when Endpoint 0 is halted following the receipt of NAK responses for longer than the time set by the NAKLIMIT0 register. This bit should be cleared to allow the endpoint to continue.
6	STATUSPKT	0-1	Set this bit at the same time as the TXPKTRDY or REQPKT bit is set, to perform a status stage transaction. Setting this bit ensures that the data toggle is set so that a DATA1 packet is used for the Status Stage transaction.
5	REQPKT	0-1	Set this bit to request an IN transaction. It is cleared when RXPKTRDY is set.
4	ERROR	0-1	This bit will be set when three attempts have been made to perform a transaction with no response from the peripheral. You should clear this bit. An interrupt is generated when this bit is set.
3	SETUPPKT	0-1	Set this bit, at the same time as the TXPKTRDY bit is set, to send a SETUP token instead of an OUT token for the transaction.
2	RXSTALL	0-1	This bit is set when a STALL handshake is received. You should clear this bit.
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. Clear this bit by setting the SERV_RXPKTRDY bit.

4.35 Control Status Register for Peripheral Transmit Endpoint (PERI_TXCSR)

The control status register for peripheral transmit endpoint (PERI_TXCSR) is shown in Figure 61 and described in Table 65.

Figure 61. 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 65. 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	This bit should always be set to 1 when the DMA is enabled.
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.

4.36 Control Status Register for Host Transmit Endpoint (HOST_TXCSR)

The control status register for host transmit endpoint (HOST_TXCSR) is shown in Figure 62 and described in Table 66.

Figure 62. Control Status Register for Host Transmit Endpoint (HOST_TXCSR)

15	14	13	12	11	10	9	8
AUTOSET	Reserved	MODE	DMAEN	FRCDATATOG	DMAMODE	DATATOGWREN	DATATOG
R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	W-0	R/W-0
7	6	5	4	3	2	1	0
NAK_TIMEOUT	CLRDATATOG	RXSTALL	SETUPPKT	FLUSHFIFO	ERROR	FIFONOTEMPTY	TXPKTRDY
R/W-0	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 66. Control Status Register for Host Transmit Endpoint (HOST_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	Reserved	0	Reserved
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	This bit should always be set to 1 when the DMA is enabled.
9	DATATOGWREN	0-1	Write 1 to this bit to enable the DATATOG bit to be written. This bit is automatically cleared once the new value is written to DATATOG.
8	DATATOG	0-1	When read, this bit indicates the current state of the Tx EP data toggle. If DATATOGWREN is high, this bit can be written with the required setting of the data toggle. If DATATOGWREN is low, any value written to this bit is ignored.
7	NAK_TIMEOUT	0-1	This bit will be set when the Tx endpoint is halted following the receipt of NAK responses for longer than the time set as the NAKLIMIT by the TXINTERVAL register. It should be cleared to allow the endpoint to continue. Note: This is valid only for Bulk endpoints.
6	CLRDATATOG	0-1	Write a 1 to this bit to reset the endpoint data toggle to 0.
5	RXSTALL	0-1	This bit is set when a STALL handshake is received. The FIFO is flushed and the TXPKTRDY bit is cleared (see below). You should clear this bit.
4	SETUPPKT	0-1	Set this bit at the same time as TXPKTRDY is set, to send a SETUP token instead of an OUT token for the transaction. Note: Setting this bit also clears the DATATOG bit.
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 (below) 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	ERROR	0-1	The USB controller sets this bit when 3 attempts have been made to send a packet and no handshake packet has been received. You should clear this bit. An interrupt is generated when the bit is set. This is valid only when the endpoint is operating in Bulk or Interrupt mode.
1	FIFONOTEMPTY	0-1	The USB controller sets this bit when there is at least 1 packet in the Tx FIFO.
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.

4.37 Maximum Packet Size for Peripheral Host Receive Endpoint (RXMAXP)

The maximum packet size for peripheral host receive endpoint (RXMAXP) is shown in Figure 63 and described in Table 67.

Figure 63. Maximum Packet Size for Peripheral Host 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 67. Maximum Packet Size for Peripheral Host Receive Endpoint (RXMAXP) Field Descriptions

Bit	Field	Value	Description
15-11	Reserved	0	Reserved
10-0	MAXPAYLOAD	0-400h	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.

4.38 Control Status Register for Peripheral Receive Endpoint (PERI_RXCSR)

The control status register for peripheral receive endpoint (PERI_RXCSR) is shown in Figure 64 and described in Table 68.

Figure 64. 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 68. 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	Always clear this bit to 0.
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.

4.39 Control Status Register for Host Receive Endpoint (HOST_RXCSR)

The control status register for host receive endpoint (HOST_RXCSR) is shown in Figure 65 and described in Table 69.

Figure 65. Control Status Register for Host Receive Endpoint (HOST_RXCSR)

15	14	13	12	11	10	9	8
AUTOCLEAR	AUTOREQ	DMAEN	DISNYET	DMAMODE	DATATOGWREN	DATATOG	Reserved
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	W-0	R/W-0	R-0
7	6	5	4	3	2	1	0
CLRDATATOG	RXSTALL	REQPKT	FLUSHFIFO	DATAERR_ NAKTIMEOUT	ERROR	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 69. Control Status Register for Host Receive Endpoint (HOST_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	AUTOREQ	1	If the CPU sets the AUTOREQ bit, then the REQPKT bit will be automatically set when the RXPKTRDY bit is cleared. Note: This bit is automatically cleared when a short packet is received.
13	DMAEN	0-1	Set this bit to enable the DMA request for the Receive endpoints.
12	DISNYET	0-1	Set this bit to disable the sending of NYET handshakes. When set, all successfully received Receive packets are ACKED 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.
11	DMAMODE	0-1	Always clear this bit to 0.
10	DATATOGWREN	0-1	Write 1 to this bit to enable the DATATOG bit to be written. This bit is automatically cleared once the new value is written to DATATOG.
9	DATATOG	0-1	When read, this bit indicates the current state of the Receive EP data toggle. If DATATOGWREN is high, this bit can be written with the required setting of the data toggle. If DATATOGWREN is low, any value written to this bit is ignored.
8	Reserved	0	Reserved
7	CLRDATATOG	0-1	Write a 1 to this bit to reset the endpoint data toggle to 0.
6	RXSTALL	0-1	When a STALL handshake is received, this bit is set and an interrupt is generated. You should clear this bit.
5	REQPKT	0-1	Write a 1 to this bit to request an IN transaction. It is cleared when RXPKTRDY is set.
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	DATAERR_NAKTIMEOUT	0-1	When operating in ISO mode, this bit is set when RXPKTRDY is set if the data packet has a CRC or bit-stuff error and cleared when RXPKTRDY is cleared. In Bulk mode, this bit will be set when the Receive endpoint is halted following the receipt of NAK responses for longer than the time set as the NAK Limit by the RXINTERVAL register. You should clear this bit to allow the endpoint to continue.
2	ERROR	0-1	The USB controller sets this bit when 3 attempts have been made to receive a packet and no data packet has been received. You should clear this bit. An interrupt is generated when the bit is set. Note: This bit is only valid when the transmit endpoint is operating in Bulk or Interrupt mode. In ISO mode, it always returns zero.
1	FIFOFULL	0-1	This bit is set when no more packets can be loaded into the Receive FIFO.

Table 69. Control Status Register for Host Receive Endpoint (HOST_RXCSR) Field Descriptions (continued)

Bit	Field	Value	Description
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.

4.40 Count 0 Register (COUNT0)

The count 0 register (COUNT0) is shown in Figure 66 and described in Table 70.

Figure 66. Count 0 Register (COUNT0)

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

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

Table 70. 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 or HOST_CSR0 is set.

4.41 Receive Count Register (RXCOUNT)

The receive count register (RXCOUNT) is shown in Figure 67 and described in Table 71.

Figure 67. Receive Count Register (RXCOUNT)

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

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

Table 71. 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.

4.42 Type Register (Host mode only) (HOST_TYPE0)

The type register (Host mode only) (HOST_TYPE0) is shown in Figure 68 and described in Table 72.

Figure 68. Type Register (Host mode only) (HOST_TYPE0)

7	6	5	0
SPEED		Reserved	
R/W-0		R-0	

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

Table 72. Type Register (Host mode only) (HOST_TYPE0) Field Descriptions

Bit	Field	Value	Description
7-6	SPEED	0-3h	Operating Speed of Target Device
		0	Illegal
		1h	High
		2h	Full
		3h	Low
5-0	Reserved	0	Reserved

4.43 Transmit Type Register (Host mode only) (HOST_TXTYPE)

The transmit type register (Host mode only) (HOST_TXTYPE) is shown in Figure 69 and described in Table 73.

Figure 69. Transmit Type Register (Host mode only) (HOST_TXTYPE)

7	6	5	4	3	0
SPEED		PROT		TENDPN	
R/W-0		R/W-0		R/W-0	

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

Table 73. Transmit Type Register (Host mode only) (HOST_TXTYPE) Field Descriptions

Bit	Field	Value	Description
7-6	SPEED	0-3h	Operating Speed of Target Device
		0	Illegal
		1h	High
		2h	Full
		3h	Low
5-4	PROT	0-3h	Set this to select the required protocol for the transmit endpoint
		0	Control
		1h	Isochronous
		2h	Bulk
		3h	Interrupt
3-0	TENDPN	0-Fh	Set this value to the endpoint number contained in the transmit endpoint descriptor returned to the USB controller during device enumeration.

4.44 NAKLimit0 Register (Host mode only) (HOST_NAKLIMIT0)

The NAKlimit0 register (Host mode only) (HOST_NAKLIMIT0) is shown in Figure 70 and described in Table 74.

Figure 70. NAKLimit0 Register (Host mode only) (HOST_NAKLIMIT0)

7	5	4	0
Reserved		EP0NAKLIMIT	
R-0		R/W-0	

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

Table 74. NAKLimit0 Register (Host mode only) (HOST_NAKLIMIT0) Field Descriptions

Bit	Field	Value	Description
7-5	Reserved	0	Reserved
4-0	EP0NAKLIMIT	0-1Fh	Sets the number of frames/microframes (high-speed transfers) after which Endpoint 0 should time out on receiving a stream of NAK responses. The number of frames/microframes selected is $2^{(-1)}$ (where m is the value set in the register, valid values 2-16). If the host receives NAK responses from the target for more frames than the number represented by the Limit set in this register, the endpoint will be halted. Note: A value of 0 or 1 disables the NAK timeout function.

4.45 Transmit Interval Register (Host mode only) (HOST_TXINTERVAL)

The transmit interval register (Host mode only) (HOST_TXINTERVAL) is shown in Figure 71 and described in Table 75.

Figure 71. Transmit Interval Register (Host mode only) (HOST_TXINTERVAL)

7	0
POLINTVL_NAKLIMIT	
R/W-0	

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

Table 75. Transmit Interval Register (Host mode only) (HOST_TXINTERVAL) Field Descriptions

Bit	Field	Value	Description
7-0	POLINTVL_NAKLIMIT	0-FFh	For Interrupt and Isochronous transfers, defines the polling interval for the currently-selected transmit endpoint. For Bulk endpoints, sets the number of frames/microframes after which the endpoint should timeout on receiving a stream of NAK responses. There is a transmit interval register for each configured transmit endpoint (except Endpoint 0). In each case, the value that is set defines a number of frames/microframes (High-Speed transfers), as follows:

Transfer Type	Speed	Valid values (m)	Interpretation
Interrupt	Low Speed or Full Speed	1-255	Polling interval is m frames
	High Speed	1-16	Polling interval is $2^{(-1)}$ microframes
Isochronous	Full Speed or High Speed	1-16	Polling interval is $2^{(-1)}$ frames/microframes
Bulk	Full Speed or High Speed	2-16	NAK Limit is $2^{(-1)}$ frames/microframes

Note: A value of 0 or 1 disables the NAK timeout function.

4.46 Receive Type Register (Host mode only) (HOST_RXTYPE)

The receive type register (Host mode only) (HOST_RXTYPE) is shown in Figure 72 and described in Table 76.

Figure 72. Receive Type Register (Host mode only) (HOST_RXTYPE)

7	6	5	4	3	0
SPEED		PROT		RENDPN	
R/W-0		R/W-0		R/W-0	

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

Table 76. Receive Type Register (Host mode only) (HOST_RXTYPE) Field Descriptions

Bit	Field	Value	Description
7-6	SPEED	0-3h	Operating Speed of Target Device
		0	Illegal
		1h	High
		2h	Full
		3h	Low
5-4	PROT	0-3h	Set this to select the required protocol for the transmit endpoint
		0	Control
		1h	Isochronous
		2h	Bulk
		3h	Interrupt
3-0	RENDPN	0-Fh	Set this value to the endpoint number contained in the Receive endpoint descriptor returned to the USB controller during device enumeration

4.47 Receive Interval Register (Host mode only) (HOST_RXINTERVAL)

The receive interval register (Host mode only) (HOST_RXINTERVAL) is shown in Figure 73 and described in Table 77.

Figure 73. Receive Interval Register (Host mode only) (HOST_RXINTERVAL)

7	0
POLINTVL_NAKLIMIT	
R/W-0	

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

Table 77. Receive Interval Register (Host mode only) (HOST_RXINTERVAL) Field Descriptions

Bit	Field	Value	Description
7-0	POLINTVL_NAKLIMIT	0-FFh	For Interrupt and Isochronous transfers, defines the polling interval for the currently-selected transmit endpoint. For Bulk endpoints, sets the number of frames/microframes after which the endpoint should timeout on receiving a stream of NAK responses. There is a transmit interval register for each configured transmit endpoint (except Endpoint 0). In each case, the value that is set defines a number of frames/microframes (High-Speed transfers), as follows:

Transfer Type	Speed	Valid values (m)	Interpretation
Interrupt	Low Speed or Full Speed	1-255	Polling interval is m frames
	High Speed	1-16	Polling interval is $2^{(m-1)}$ microframes
Isochronous	Full Speed or High Speed	1-16	Polling interval is $2^{(m-1)}$ frames/microframes
Bulk	Full Speed or High Speed	2-16	NAK Limit is $2^{(m-1)}$ frames/microframes

Note: A value of 0 or 1 disables the NAK timeout function.

4.48 Configuration Data Register (CONFIGDATA)

The configuration data register (CONFIGDATA) is shown in Figure 74 and described in Table 78.

Figure 74. 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 78. 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

4.49 Transmit and Receive FIFO Register for Endpoint 0 (FIFO0)

The transmit and receive FIFO register for endpoint 0 (FIFO0) is shown in Figure 75 and described in Table 79.

Figure 75. Transmit and Receive FIFO Register for Endpoint 0 (FIFO0)

31	0
DATA	
R/W-0	

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

Table 79. Transmit and Receive FIFO Register for Endpoint 0 (FIFO0) Field Descriptions

Bit	Field	Value	Description
31-0	DATA	0-FFFF 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.

4.50 Transmit and Receive FIFO Register for Endpoint 1 (FIFO1)

The transmit and receive FIFO register for endpoint 1 (FIFO1) is shown in Figure 76 and described in Table 80.

Figure 76. Transmit and Receive FIFO Register for Endpoint 1 (FIFO1)

31	0
DATA	
R/W-0	

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

Table 80. Transmit and Receive FIFO Register for Endpoint 1 (FIFO1) Field Descriptions

Bit	Field	Value	Description
31-0	DATA	0-FFFF FFFF	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.

4.51 Transmit and Receive FIFO Register for Endpoint 2 (FIFO2)

The transmit and receive FIFO register for endpoint 2 (FIFO2) is shown in Figure 77 and described in Table 81.

Figure 77. Transmit and Receive FIFO Register for Endpoint 2 (FIFO2)

31	0
DATA	
R/W-0	

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

Table 81. Transmit and Receive FIFO Register for Endpoint 2 (FIFO2) Field Descriptions

Bit	Field	Value	Description
31-0	DATA	0-FFFF 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.

4.52 Transmit and Receive FIFO Register for Endpoint 3 (FIFO3)

The transmit and receive FIFO register for endpoint 3 (FIFO3) is shown in Figure 78 and described in Table 82.

Figure 78. Transmit and Receive FIFO Register for Endpoint 3 (FIFO3)

31	0
DATA	
R/W-0	

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

Table 82. Transmit and Receive FIFO Register for Endpoint 3 (FIFO3) Field Descriptions

Bit	Field	Value	Description
31-0	DATA	0-FFFF 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.

4.53 Transmit and Receive FIFO Register for Endpoint 4 (FIFO4)

The transmit and receive FIFO register for endpoint 4 (FIFO4) is shown in Figure 79 and described in Table 83.

Figure 79. Transmit and Receive FIFO Register for Endpoint 4 (FIFO4)

31	0
DATA	
R/W-0	

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

Table 83. Transmit and Receive FIFO Register for Endpoint 4 (FIFO4) Field Descriptions

Bit	Field	Value	Description
31-0	DATA	0-FFFF 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.

4.54 Device Control Register (DEVCTL)

The device control register (DEVCTL) is shown in Figure 80 and described in Table 84.

Figure 80. Device Control Register (DEVCTL)

7	6	5	4	3	2	1	0
BDEVICE	FSDEV	LSDEV	VBUS		HOSTMODE	HOSTREQ	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 84. 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.
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.
4-3	VBUS	0-3h	These read-only bits encode the current VBus level as follows:
		0	Below Session End
		1h	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.
1	HOSTREQ	0-1	When set, the USB controller will initiate the Host Negotiation when Suspend mode is entered. It is cleared when Host Negotiation is completed. ('B' device only)
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.

4.55 Transmit Endpoint FIFO Size (TXFIFOSZ)

Section 2.6 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 81 and described in Table 85.

Figure 81. 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 85. 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.

4.56 Receive Endpoint FIFO Size (RXFIFOSZ)

Section 2.6 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 82 and described in Table 86.

Figure 82. 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 86. 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.

4.57 Transmit Endpoint FIFO Address (TXFIFOADDR)

Section 2.6 describes dynamically setting endpoint FIFO sizes.

The transmit endpoint FIFO address (TXFIFOADDR) is shown in Figure 83 and described in Table 87.

Figure 83. 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 87. 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

4.58 Receive Endpoint FIFO Address (RXFIFOADDR)

Section 2.6 describes dynamically setting endpoint FIFO sizes.

The receive endpoint FIFO address (RXFIFOADDR) is shown in Figure 84 and described in Table 88.

Figure 84. 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 88. 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

4.59 Hardware Version Register (HWVERS)

The hardware version register (HWVERS) contains the RTL major and minor version numbers for the USB 2.0 OTG controller module. The RTL version number is REVMAJ.REVMIN. The HWVERS is shown in Figure 85 and described in Table 89.

Figure 85. 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 89. 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 0-31.
9-0	REVMIN	0-3E7h	Minor version of RTL. Range is 0-999.

4.60 Transmit Function Address (TXFUNCADDR)

The transmit function address (TXFUNCADDR) is shown in Figure 86 and described in Table 90.

Figure 86. Transmit Function Address (TXFUNCADDR)

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

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

Table 90. Transmit Function Address (TXFUNCADDR) Field Descriptions

Bit	Field	Value	Description
7	Reserved	0	Reserved
6-0	FUNCADDR	0-7Fh	Address of target function

4.61 Transmit Hub Address (TXHUBADDR)

The transmit hub address (TXHUBADDR) is shown in Figure 87 and described in Table 91.

Figure 87. Transmit Hub Address (TXHUBADDR)

7	6	0
MULT_TRANS	HUBADDR	
R/W-0	R/W-0	

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

Table 91. Transmit Hub Address (TXHUBADDR) Field Descriptions

Bit	Field	Value	Description
7	MULT_TRANS	0-1	Set to 1 if hub has multiple transaction translators. Cleared to 0 if only single transaction translator is available.
6-0	HUBADDR	0-7Fh	Address of hub

4.62 Transmit Hub Port (TXHUBPORT)

The transmit hub port (TXHUBPORT) is shown in Figure 88 and described in Table 92.

Figure 88. Transmit Hub Port (TXHUBPORT)

7	6	0
Reserved	HUBPORT	
R-0	R/W-0	

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

Table 92. Transmit Hub Port (TXHUBPORT) Field Descriptions

Bit	Field	Value	Description
7	Reserved	0	Reserved
6-0	HUBPORT	0-7Fh	Port number of the hub

4.63 Receive Function Address (RXFUNCADDR)

The receive function address (RXFUNCADDR) is shown in Figure 89 and described in Table 93.

Figure 89. Receive Function Address (RXFUNCADDR)

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

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

Table 93. Receive Function Address (RXFUNCADDR) Field Descriptions

Bit	Field	Value	Description
7	Reserved	0	Reserved
6-0	FUNCADDR	0-7Fh	Address of target function

4.64 Receive Hub Address (RXHUBADDR)

The receive hub address (RXHUBADDR) is shown in Figure 90 and described in Table 94.

Figure 90. Receive Hub Address (RXHUBADDR)

7	6	0
MULT_TRANS	HUBADDR	
R/W-0	R/W-0	

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

Table 94. Receive Hub Address (RXHUBADDR) Field Descriptions

Bit	Field	Value	Description
7	MULT_TRANS	0-1	Set to 1 if hub has multiple transaction translators. Cleared to 0 if only single transaction translator is available.
6-0	HUBADDR	0-7Fh	Address of hub

4.65 Receive Hub Port (RXHUBPORT)

The receive hub port (RXHUBPORT) is shown in Figure 91 and described in Table 95.

Figure 91. Receive Hub Port (RXHUBPORT)

7	6	0
Reserved	HUBPORT	
R-0	R/W-0	

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

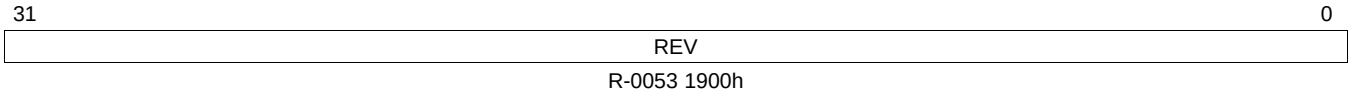
Table 95. Receive Hub Port (RXHUBPORT) Field Descriptions

Bit	Field	Value	Description
7	Reserved	0	Reserved
6-0	HUBPORT	0-7Fh	Port number of hub

4.66 CDMA Revision Identification Register (DMAREVID)

The CDMA revision identification register (DMAREVID) contains the revision for the module. The DMAREVID is shown in Figure 92 and described in Table 96.

Figure 92. CDMA Revision Identification Register (DMAREVID)



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

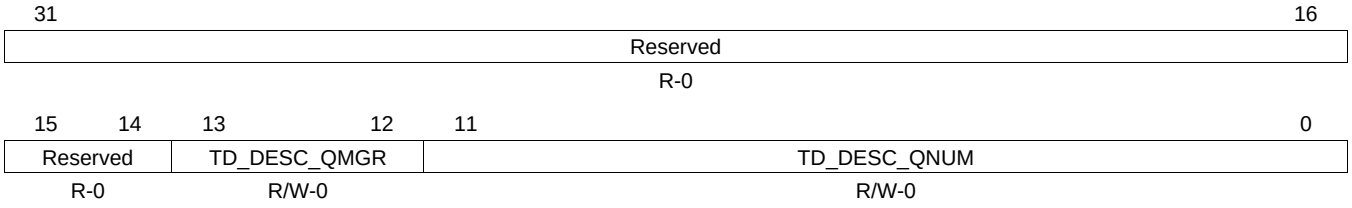
Table 96. CDMA Revision Identification Register (DMAREVID) Field Descriptions

Bit	Field	Value	Description
31-0	REV	0053 1900h	Revision ID of the CPPI DMA (CDMA) module.

4.67 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 93 and described in Table 97.

Figure 93. CDMA Teardown Free Descriptor Queue Control Register (TDFDQ)



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

Table 97. CDMA Teardown Free Descriptor Queue Control Register (TDFDQ) Field Descriptions

Bit	Field	Value	Description
31-14	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.

4.68 CDMA Emulation Control Register (DMAEMU)

The CDMA emulation controls the behavior of the DMA when the emususp input is asserted. The CDMA emulation control register (DMAEMU) is shown in Figure 94 and described in Table 98.

Figure 94. CDMA Emulation Control Register (DMAEMU)

31	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 98. CDMA Emulation Control Register (DMAEMU) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1	SOFT		Determines emulation mode functionality. When the FREE bit is cleared to 0, the SOFT bit selects the mode.
		0	Upon emulation suspend, operation is not affected.
		1	In response to an emulation suspend event, the logic halts after the current transaction is completed.
0	FREE		Free run emulation control. Determines emulation mode functionality. When the FREE bit is cleared to 0, the SOFT bit selects the mode.
		0	The SOFT bit selects the mode.
		1	Runs free regardless of the SOFT bit.

4.69 CDMA Transmit Channel n Global Configuration Registers (TXGCR[0]-TXGCR[3])

The transmit channel *n* configuration registers (TXGCR[*n*]) initialize the behavior of each of the transmit DMA channels. There are four configuration registers, one for each transmit DMA channels. The transmit channel *n* configuration registers (TXGCR[*n*]) are shown in Figure 95 and described in Table 99.

Figure 95. CDMA Transmit Channel *n* Global Configuration Registers (TXGCR[*n*])

31	30	29	16
TX_ENABLE	TX_TEARDOWN	Reserved	
R/W-0	R/W-0	R-0	

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

Table 99. CDMA Transmit Channel *n* Global Configuration Registers (TXGCR[*n*]) Field Descriptions

Bit	Field	Value	Description
31	TX_ENABLE		Channel control. The TX_ENABLE field is cleared after a channel teardown is complete.
		0	Disables channel
		1	Enables channel
30	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.
29-14	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. This is the Tx Completion Queue.

4.70 CDMA Receive Channel n Global Configuration Registers (RXGCR[0]-RXGCR[3])

The receive channel *n* global configuration registers (RXGCR[*n*]) initialize the global (non-descriptor-type specific) behavior of each of the receive DMA channels. There are four configuration registers, one for each receive DMA channels. 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 (RXGCR[*n*]) are shown in Figure 96 are described in Table 100.

Figure 96. CDMA Receive Channel *n* Global Configuration Registers (RXGCR[*n*])

31	30	29–25	24	23–16
RX_ENABLE	RX_TEARDOWN	Reserved	RX_ERROR_HANDLING	RX_SOP_OFFSET
R/W-0	R/W-0	R-0	W-0	W-0

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

Table 100. CDMA Receive Channel *n* Global Configuration Registers (RXGCR[*n*]) Field Descriptions

Bit	Field	Value	Description
31	RX_ENABLE		Channel control. Field is cleared after a channel teardown is complete.
		0	Disables channel
		1	Enables channel
30	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.
29-25	Reserved	0	Reserved
24	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.
23-16	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.
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. This is the Rx Completion Queue.

4.71 CDMA Receive Channel n Host Packet Configuration Registers A (RXHPCRA[0]-RXHPCRA[3])

The receive channel *n* host packet configuration registers A (RXHPCRA[*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 channels. The receive channel *n* host packet configuration registers A (RXHPCRA[*n*]) are shown in Figure 97 and described in Table 101.

Figure 97. Receive Channel *n* Host Packet Configuration Registers A (RXHPCRA[*n*])

31 30	29 28	27 16
Reserved	RX_HOST_FDQ1_QMGR	RX_HOST_FDQ1_QNUM
R-0	W-0	W-0

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

Table 101. Receive Channel *n* Host Packet Configuration Registers A (RXHPCRA[*n*]) Field Descriptions

Bit	Field	Value	Description
31-30	Reserved	0	Reserved
29-28	RX_HOST_FDQ1_QMGR	0-3h	Specifies which buffer manager should be used for the second receive buffer in a host type packet.
27-16	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. This is the Rx Submit Queue for the second Incoming Packet.
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. This is the Rx Submit Queue for the first Incoming Packet.

4.72 CDMA Receive Channel n Host Packet Configuration Registers B (RXHPCRB[0]-RXHPCRB[3])

The receive channel *n* host packet configuration registers B (RXHPCRB[*n*]) initialize the behavior of each of the receive DMA channels for reception of host type packets. There are four configuration B registers, one for each receive DMA channels. The receive channel *n* host packet configuration registers B (RXHPCRB[*n*]) are shown in Figure 98 and described in Table 102.

Figure 98. Receive Channel *n* Host Packet Configuration Registers B (RXHPCRB[*n*])

31	30	29	28	27	16
Reserved		RX_HOST_FDQ3_QMGR		RX_HOST_FDQ3_QNUM	
R-0		W-0		W-0	

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

Table 102. Receive Channel *n* Host Packet Configuration Registers B (RXHPCRB[*n*]) Field Descriptions

Bit	Field	Value	Description
31-30	Reserved	0	Reserved
29-28	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.
27-16	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. This is the Rx Submit Queue for the fourth and remaining Incoming Packet.
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. This is the Rx Submit Queue for the third Incoming Packet.

4.73 CDMA Scheduler Control Register (DMA_SCHED_CTRL)

The CDMA scheduler control register (DMA_SCHED_CTRL) enables the scheduler and indicates the last entry in the scheduler table. The CDMA scheduler control register (DMA_SCHED_CTRL) is shown in Figure 99 and described in Table 103.

Figure 99. CDMA Scheduler Control Register (DMA_SCHED_CTRL)

31	30	16
ENABLE	Reserved	
R/W-0	R-0	

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

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

Table 103. CDMA Scheduler Control Register (DMA_SCHED_CTRL) Field Descriptions

Bit	Field	Value	Description
31	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.
30-8	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

4.74 CDMA Scheduler Table Word n Registers (WORD[0]-WORD[63])

The CDMA scheduler table word *n* registers (WORD[*n*]) has 4 entries (ENTRY[0] to ENTRY[3]) that provide information about the scheduler. The CDMA scheduler table word *n* registers (WORD[*n*]) are shown in Figure 100 and described in Table 104.

Figure 100. CDMA Scheduler Table Word *n* Registers (WORD[*n*])

31	30	28	27	24	23	22	20	19	16
ENTRY3_RXTX	Reserved		ENTRY3_CHANNEL		ENTRY2_RXTX	Reserved		ENTRY2_CHANNEL	
W-0	R-0		W-0		W-0	R-0		W-0	

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

Table 104. CDMA Scheduler Table Word *n* Registers (WORD[*n*]) Field Descriptions

Bit	Field	Value	Description
31	ENTRY3_RXTX		This entry is for a transmit or a receive channel.
		0	Transmit channel
		1	Receive channel
30-28	Reserved	0	Reserved

Table 104. CDMA Scheduler Table Word *n* Registers (WORD[*n*]) Field Descriptions (continued)

Bit	Field	Value	Description
27-24	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.
23	ENTRY2_RXTX		This entry is for a transmit or a receive channel.
		0	Transmit channel
		1	Receive channel
22-20	Reserved	0	Reserved
19-16	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.
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.

4.75 Queue Manager Revision Identification Register (QMGRREVID)

The queue manager revision identification register (QMGRREVID) contains the major and minor revisions for the module. The QMGRREVID is shown in Figure 101 and described in Table 105.

Figure 101. Queue Manager Revision Identification Register (QMGRREVID)

31	0
REV	
R-0052 1200h	

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

Table 105. Queue Manager Revision Identification Register (QMGRREVID) Field Descriptions

Bit	Field	Value	Description
31-0	REV	0052 1200h	Revision ID of the queue manager.

4.76 Queue Manager Queue Diversion Register (DIVERSION)

The queue manager queue diversion register (DIVERSION) is used to transfer the contents of one queue onto another queue. It does not support byte accesses. The queue manager queue diversion register (DIVERSION) is shown in Figure 102and described in Table 106.

Figure 102. Queue Manager Queue Diversion Register (DIVERSION)

31	30	29	16
HEAD_TAIL	Rsvd	DEST_QNUM	
W-0	R-0	W-0	

15	14	13	0
Reserved		SOURCE_QNUM	
R-0		W-0	

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

Table 106. Queue Manager Queue Diversion Register (DIVERSION) Field Descriptions

Bit	Field	Value	Description
31	HEAD_TAIL		Indicates whether queue contents should be merged on to the head or tail of the destination queue.
		0	Head
		1	Tail
30	Reserved	0	Reserved
29-16	DEST_QNUM	0-3FFFh	Destination Queue Number
15-14	Reserved	0	Reserved
13-0	SOURCE_QNUM	0-3FFFh	Source Queue Number

4.77 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. It does not support byte accesses. The free descriptor/buffer queue starvation count register (FDBSC0) is shown in Figure 103 and described in Table 107.

Figure 103. Queue Manager Free Descriptor/Buffer Starvation Count Register 0 (FDBSC0)

31	24	23	16
FDBQ3_STARVE_CNT		FDBQ2_STARVE_CNT	
RC-0		RC-0	

15	8	7	0
FDBQ1_STARVE_CNT		FDBQ0_STARVE_CNT	
RC-0		RC-0	

LEGEND: RC = Cleared on read; *-n* = value after reset

Table 107. Queue Manager Free Descriptor/Buffer Starvation Count Register 0 (FDBSC0) Field Descriptions

Bit	Field	Value	Description
31-24	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 read.
23-16	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 read.
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.
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.

4.78 Queue Manager Free Descriptor/Buffer Starvation Count Register 1 (FDBSC1)

The free descriptor/buffer queue starvation count register 1 (FDBSC1) provides statistics about how many starvation events are occurring on the receive free descriptor/buffer queues. It does not support byte accesses. The free descriptor/buffer queue starvation count register 1 (FDBSC1) is shown in Figure 104 and described in Table 108.

Figure 104. Queue Manager Free Descriptor/Buffer Starvation Count Register 1 (FDBSC1)

31	24	23	16
FDBQ7_STARVE_CNT		FDBQ6_STARVE_CNT	
RC-0		RC-0	

15	8	7	0
FDBQ5_STARVE_CNT		FDBQ4_STARVE_CNT	
RC-0		RC-0	

LEGEND: RC = Cleared on read; *-n* = value after reset

Table 108. Queue Manager Free Descriptor/Buffer Starvation Count Register 1 (FDBSC1) Field Descriptions

Bit	Field	Value	Description
31-24	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.
23-16	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.
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.
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.

4.79 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. It does not support byte accesses. The free descriptor/buffer queue starvation count register 2 (FDBSC2) is shown in Figure 105 and described in Table 109.

Figure 105. Queue Manager Free Descriptor/Buffer Starvation Count Register 2 (FDBSC2)

31	24	23	16
FDBQ11_STARVE_CNT		FDB10_STARVE_CNT	
RC-0		RC-0	

15	8	7	0
FDBQ9_STARVE_CNT		FDBQ8_STARVE_CNT	
RC-0		RC-0	

LEGEND: RC = Cleared on read; *-n* = value after reset

Table 109. Queue Manager Free Descriptor/Buffer Starvation Count Register 2 (FDBSC2) Field Descriptions

Bit	Field	Value	Description
31-24	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.
23-16	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.
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.
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.

4.80 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. It does not support byte accesses. The free descriptor/buffer queue starvation count register 3 (FDBSC3) is shown in Figure 106 and described in Table 110.

Figure 106. Queue Manager Free Descriptor/Buffer Starvation Count Register 3 (FDBSC3)

31 – 24	23 – 16
FDBQ15_STARVE_CNT	FDB14_STARVE_CNT
RC-0	RC-0

15 – 8	7 – 0
FDBQ13_STARVE_CNT	FDBQ12_STARVE_CNT
RC-0	RC-0

LEGEND: RC = Cleared on read; *-n* = value after reset

Table 110. Queue Manager Free Descriptor/Buffer Starvation Count Register 3 (FDBSC3) Field Descriptions

Bit	Field	Value	Description
31-24	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.
23-16	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.
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.
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.

4.81 Queue Manager Linking RAM Region 0 Base Address Register (LRAM0BASE)

The queue manager linking RAM region 0 base address register (LRAM0BASE) sets 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. It does not support byte accesses. The queue manager linking RAM region 0 base address register (LRAM0BASE) is shown in Figure 107 and described in Table 111.

Figure 107. Queue Manager Linking RAM Region 0 Base Address Register (LRAM0BASE)

31 – 0
REGION0_BASE
R/W-0

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

Table 111. Queue Manager Linking RAM Region 0 Base Address Register (LRAM0BASE) Field Descriptions

Bit	Field	Value	Description
31-0	REGION0_BASE	0-FFFF FFFFh	This field stores 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.

4.82 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 108 and described in Table 112.

Figure 108. Queue Manager Linking RAM Region 0 Size Register (LRAM0SIZE)

31	16
Reserved	
R-0	

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 112. Queue Manager Linking RAM Region 0 Size Register (LRAM0SIZE) Field Descriptions

Bit	Field	Value	Description
31-14	Reserved	0	Reserved
13-0	REGION0_SIZE	0-3FFFh	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.

4.83 Queue Manager Linking RAM Region 1 Base Address Register (LRAM1BASE)

The queue manager linking RAM region 1 base address register (LRAM1BASE) is used to 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. It does not support byte accesses. The queue manager linking RAM region 1 base address register (LRAM1BASE) is shown in Figure 109 and described in Table 113.

Figure 109. Queue Manager Linking RAM Region 1 Base Address Register (LRAM1BASE)

31	0
REGION1_BASE	
R/W-0	

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

Table 113. Queue Manager Linking RAM Region 1 Base Address Register (LRAM1BASE) Field Descriptions

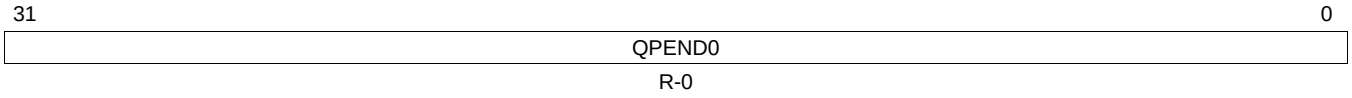
Bit	Field	Value	Description
31-0	REGION1_BASE	0-FFFF FFFFh	This field stores the base address for the second region of the linking RAM. This may be anywhere in 32-bit address space but would be typically located in off-chip memory.

4.84 Queue Manager Queue Pending Register 0 (PEND0)

The queue pending register 0 (PEND0) can be read to find the pending status for queues 31 to 0. It does not support byte accesses. The queue pending register 0 (PEND0) is shown in Figure 110 and described in Table 114.

NOTE: The pending bit gets set when a Descriptor address is loaded in a Queue. The loading action causes the corresponding bit for that Queue to get set. Similarly, the pending bit gets cleared when the Descriptor address is off-loaded from a Queue by reading it. One way to check if the receive or transmit transfer has completed is by checking the bit that corresponds to the desired Completion Queue for that particular transfer. When the Queue Manager is finished with the transfer, it will load the Descriptor address to the Completion Queue.

Figure 110. Queue Manager Queue Pending Register 0 (PEND0)



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

Table 114. Queue Manager Queue Pending Register 0 (PEND0) Field Descriptions

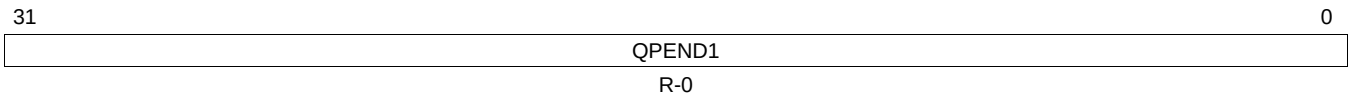
Bit	Field	Value	Description
31-0	QPEND0	0-FFFF FFFFh	This field indicates the queue pending status for queues 31-0.

4.85 Queue Manager Queue Pending Register 1 (PEND1)

The queue pending register 1 (PEND1) can be read to find the pending status for queues 63 to 32. It does not support byte accesses. The queue pending register 1 (PEND1) is shown in Figure 111 and described in Table 115.

NOTE: The pending bit gets set when a Descriptor address is loaded in a Queue. The loading action causes the corresponding bit for that Queue to get set. Similarly, the pending bit gets cleared when the Descriptor address is off-loaded from a Queue by reading it. One way to check if the receive or transmit transfer has completed is by checking the bit that corresponds to the desired Completion Queue for that particular transfer. When the Queue Manager is finished with the transfer, it will load the Descriptor address to the Completion Queue.

Figure 111. Queue Manager Queue Pending Register 1 (PEND1)



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

Table 115. Queue Manager Queue Pending Register 1 (PEND1) Field Descriptions

Bit	Field	Value	Description
31-0	QPEND1	0-FFFF FFFFh	This field indicates the queue pending status for queues 63-32.

4.86 Queue Manager Memory Region *R* Base Address Registers (QMEMRBASE[0]-QMEMRBASE[15])

The memory region *R* base address register (QMEMRBASE[*R*]) is 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. It does not support byte accesses. The memory region *R* base address register (QMEMRBASE[*R*]) is shown in Figure 112 and described in Table 116.

Figure 112. Queue Manager Memory Region *R* Base Address Registers (QMEMRBASE[*R*])

31	0
REG	
R/W-0	

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

Table 116. Queue Manager Memory Region *R* Base Address Registers (QMEMRBASE[*R*]) Field Descriptions

Bit	Field	Value	Description
31-0	REG	0-FFFF FFFFh	This field contains the base address of the memory region R.

4.87 Queue Manager Memory Region *R* Control Registers (QMEMRCTRL[0]-QMEMRCTRL[15])

The memory region *R* control register (QMEMRCTRL[*R*]) is written by the host to configure various parameters of memory region *R*, where *R* is 0-15. It does not support byte accesses. The memory region *R* control register (QMEMRCTRL[*R*])) is shown in Figure 113 and described in Table 117.

Figure 113. Queue Manager Memory Region *R* Control Registers (QMEMRCTRL[*R*])

31 30	29 16
Reserved	START_INDEX
R-0	R/W-0

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

Table 117. Queue Manager Memory Region *R* Control Registers (QMEMRCTRL[*R*]) Field Descriptions

Bit	Field	Value	Description
31-30	Reserved	0	Reserved
29-16	START_INDEX	0-3FFFh	This field indicates where in linking RAM the descriptor linking information corresponding to memory region R starts.
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

4.88 Queue Manager Queue N Control Register D (CTRLD[0]-CTRLD[63])

The queue manager queue *N* control register D (CTRLD[*N*]) is 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. It does not support byte accesses. The queue manager queue *N* control register D (CTRLD[*N*]) is shown in Figure 114 and described in Table 118.

Figure 114. Queue Manager Queue *N* Control Register D (CTRLD[*N*])

31	16
DESC_PTR	
R/W-0	

15	5	4	0
DESC_PTR		DESC_SIZE	
R/W-0		R/W-0	

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

Table 118. Queue Manager Queue *N* Control Register D (CTRLD[*N*]) Field Descriptions

Bit	Field	Value	Description
31-5	DESC_PTR		Descriptor Pointer
		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

4.89 Queue Manager Queue *N* Status Register A (QSTATA[0]-QSTATA[63])

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 115 and described in Table 119.

Figure 115. Queue Manager Queue *N* Status Register A (QSTATA[*N*])

31	16
Reserved	
R-0	

15	14	13	0
Reserved		QUEUE_ENTRY_COUNT	
R-0		R-0	

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

Table 119. Queue Manager Queue *N* Status Register A (QSTATA[*N*]) Field Descriptions

Bit	Field	Value	Description
31-14	Reserved	0	Reserved
13-0	QUEUE_ENTRY_COUNT	0-3FFFh	This field indicates how many packets are currently queued on the queue.

4.90 Queue Manager Queue *N* Status Register B (QSTATB[0]-QSTATB[63])

The queue manager queue *N* status register B (QSTATB[*N*]) is an optional register that is 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. It does not support byte accesses. The queue manager queue *N* status register B (QSTATB[*N*]) is shown in Figure 116 and described in Table 120.

Figure 116. Queue Manager Queue *N* Status Register B (QSTATB[*N*])

31	28	27	16
Reserved		QUEUE_BYTE_COUNT	
R-0		R-0	

15	0
QUEUE_BYTE_COUNT	
R-0	

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

Table 120. Queue Manager Queue *N* Status Register B (QSTATB[*N*]) Field Descriptions

Bit	Field	Value	Description
31-28	Reserved	0	Reserved
27-0	QUEUE_BYTE_COUNT	0-FFF FFFFh	Indicates how many bytes total are contained in all of the packets which are currently queued on this queue.

4.91 Queue Manager Queue *N* Status Register C (QSTATC[0]-QSTATC[63])

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 117 and described in Table 121.

Figure 117. Queue Manager Queue *N* Status Register C (QSTATC[*N*])

31	16
Reserved	
R-0	

15	14	13	0
Reserved		PACKET_SIZE	
R-0		R-0	

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

Table 121. Queue Manager Queue *N* Status Register C (QSTATC[*N*]) Field Descriptions

Bit	Field	Value	Description
31-14	Reserved	0	Reserved
13-0	PACKET_SIZE	0-3FFFh	This field indicates how many packets are currently queued on the queue.

Appendix A Revision History

Table 122 lists the changes made since the previous version of this document.

Table 122. Document Revision History

Reference	Additions/Modifications/Deletions
Table 3	Changed Description of USB0_ID pin.
Section 2.7	Changed first, second, third, and fourth paragraphs.

IMPORTANT NOTICE

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

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

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

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

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

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

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

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

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Communications and Telecom	www.ti.com/communications
DSP	dsp.ti.com	Computers and Peripherals	www.ti.com/computers
Clocks and Timers	www.ti.com/clocks	Consumer Electronics	www.ti.com/consumer-apps
Interface	interface.ti.com	Energy	www.ti.com/energy
Logic	logic.ti.com	Industrial	www.ti.com/industrial
Power Mgmt	power.ti.com	Medical	www.ti.com/medical
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Space, Avionics & Defense	www.ti.com/space-avionics-defense
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video and Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless-apps

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