MENTOR GRAPHICS

THE INTELLIGENT APPROACH TO INTELLECTUAL PROPERTY

MUSBMHDRC

USB 2.0 MULTI-POINT DUAL-ROLE CONTROLLER

Product Specification and Programming Guide

SOFT CORES

5/25/2007 PSPG-40161.03-FC

For Customer Support on this product:

- Call up the Customer Inquiry Service at **http://www.mentor.com/supportnet**

- Email **support_net@mentor.com**

- Phone **1-800-547-4303** (toll-free in US, Mexico and Canada)
  (Customers in other parts of the world should contact their local Mentor Graphics support office.)

Full details are given in the Customer Support Handbook, provided in Adobe Acrobat format as **custhb.pdf** in the **/databook** directory on Mentor Graphics Soft Cores CDs. Please note the checklists of actions to take and information to have to hand when contacting Customer Support that are given in the Customer Support Handbook.

SOFT CORES

CONFIDENTIAL

TABLE OF CONTENTS

CONFIDENTIAL

CONTENTS

CONTENTS   **CONFIDENTIAL**

SOFT CORES

SOFT CORES

CONFIDENTIAL

CONTENTS

CONFIDENTIAL

SOFT
CORES

CONFIDENTIAL

CONTENTS

CONTENTS   CONFIDENTIAL

1. INTRODUCTION

MUSBMHDRC
USB 2.0 MULTI-POINT
DUAL-ROLE CONTROLLER

♦ **Operates either as the function controller of a high- /full-speed USB peripheral or as the host/peripheral in point-to-point or multi-point communications with other USB functions**

♦ **Complies with the USB 2.0 standard for high-speed (480 Mbps) functions and with the *On-The-Go* supplement to the USB 2.0 specification**

♦ **Supports OTG communications with one or more high-, full- or low-speed device**

♦ **Supports Session Request Protocol (SRP) and Host Negotiation Protocol (HNP)**

♦ **Supports Suspend and Resume signaling**

♦ **UTMI+ Level 3 Transceiver Interface with optional ULPI Link Wrapper**

♦ **Optional USB 1.1 PHY Interface (for full-speed/low-speed operation only), with optional I^{2}C interface allowing use with I^{2}C-controlled PHYs**

♦ **Soft connect/disconnect**

♦ **Configurable for up to 15 additional Transmit endpoints and up to 15 additional Receive endpoints**

♦ **Offers dynamic allocation of endpoints, to maximize number of devices supported**

♦ **Configurable FIFOs, including the option of dynamic FIFO sizing**

♦ **Synchronous RAM interface for FIFOs**

♦ **Support for DMA access to FIFOs**

♦ **High-level AMBA™ AHB-compatible CPU interface (works with a wide range of AHB bus speeds)**

♦ **Supports multi-layer operations on the AHB bus**

♦ **Performs all transaction scheduling in hardware**

♦ **Graphical User Interface provided for core configuration**

The MUSBMHDRC is a versatile design that provides in a single core:

• the function controller of a high-/full-speed USB peripheral;

• a 'Dual-role' USB controller for point-to-point 'On-The-Go' (OTG) communications with another USB function (which can be either high-speed, full-speed or low-speed); and

• (when connecteded to a hub) the host controller for a multi-point USB system.

– in turn allowing the device in which the MUSBMHDRC core is used to switch between these different roles as required.

The core complies both with the USB 2.0 standard for high-speed and full-speed functions and with the *On-The-Go* supplement to the USB 2.0 specification. The USB *On-The-Go* specification has been introduced to provide a low-cost connectivity solution for consumer portable devices such as mobile phones, PDAs, digital still cameras and MP3 players. Devices that are solely peripherals

CONFIDENTIAL

can initiate USB traffic through a Session Request Protocol (SRP) while Dual-role devices support both SRP and Host Negotiation Protocol (HNP) and can take on the role of either Host or Peripheral as required. The MUSBMHDRC also supports split transactions, which in turn allows it to support the use of full- or low-speed devices with a USB 2.0 hub. The core also includes support for powering-down portable devices when not in use.

The MUSBMHDRC is user-configurable for up to 15 'Transmit' endpoints and/or up to 15 'Receive' endpoints in addition to Endpoint 0. (The use of these endpoints for IN transactions and OUT transactions depends on whether the MUSBMHDRC is being used as a peripheral or as a host. When used as a peripheral, IN transactions are processed through TX endpoints and OUT transactions are processed through Rx endpoints. When used as a host, IN transactions are processed through Rx endpoints and OUT transactions are processed through TX endpoints.) These additional endpoints can be individually configured in software to handle either Bulk transfers (which also allows them to handle Interrupt transfers), Isochronous transfers or Control transfers. Further, the endpoints can also be allocated to different target device functions on the fly – maximizing the number of devices that can be simultaneously supported.

Each endpoint requires a FIFO to be associated with it. The MUSBMHDRC has a RAM interface for connecting to a single block of synchronous single-port RAM which is used for all the endpoint FIFOs. (The RAM block itself needs to be added by the user.)

The FIFO for Endpoint 0 is required to be 64 bytes deep and will buffer 1 packet. The RAM interface is configurable with regard to the other endpoint FIFOs, which may be from 8 to 8192 bytes in size and can buffer either 1 or 2 packets. Separate FIFOs may be associated with each endpoint: alternatively a TX endpoint and the Rx endpoint with the same Endpoint number can be configured to use the same FIFO, for example to reduce the size of RAM block needed, provided they can never be active at the same time.

The MUSBMHDRC is offered with a 32-bit synchronous CPU interface designed for connection to an AMBA AHB bus[1]. The interface supports use with an AHB bus running at a wide range of bus speeds. Multi-layer operations on the AHB bus are also supported. The MUSBMHDRC can also be readily connected to a range of other standard buses through the addition of a suitable wrapper/bridge.

There is also support for DMA access to the Endpoint FIFOs.

The MUSBMHDRC provides a UTMI+ Level 3-compatible interface for connecting to a suitable USB high/full-speed transceiver. An optional ULPI Link Wrapper (described in the **musbhdrc_ulpi_an.pdf** document included in the **musbmhdrc/docs** directory) is included for connecting to ULPI-compatible PHYs. An alternative interface is also provided that allows use of a USB 1.1 full-speed PHY with the core but only for full-speed and low-speed transactions. (This interface is described in Section 8.1).

The MUSBMHDRC provides all the encoding, decoding, checking and re-requesting needed in sending and receiving USB packets – interrupting the CPU only when endpoint data has been successfully transferred.

When acting as the host, the MUSBMHDRC additionally maintains a frame counter and automatically schedules SOF, Isochronous, Interrupt and Bulk transfers. It also includes support for the Session Request and the 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.

The MUSBMHDRC offers a range of test modes – primarily the four test modes for High-speed operation described in the USB 2.0 specification. It also includes options that allow it to be forced into Full-speed mode, High-speed mode or Host mode. The last of these may be useful in helping to debug PHY problems in hardware.

Graphical user interface scripts are provided for configuring the core to the user's requirements. The script to use depends on the CPU interface that is selected. *Please Note:* At the time of writing, the core is only available in Verilog.

This specification should be read in conjunction with the *USB On-The-Go* specification, which also gives details of power requirements, voltage levels, connectors etc.

---

[1] Created with reference to the ARM AMBA Specification, Rev. 2.0 (Chapter 3: AMBA AHB)

CONFIDENTIAL

SOFT CORES

2. FUNCTIONAL DESCRIPTION

2.1. MODES OF OPERATION

The MUSBMHDRC has two main modes of operation – Peripheral mode and Host mode.

In Peripheral mode, the MUSBMHDRC encodes, decodes, checks and directs all USB packets sent and received. IN transactions are handled through the device's TX FIFOs, OUT transactions are handled through its Rx FIFOs. Control, Bulk, Isochronous and Interrupt transactions are supported.

In Host mode, the way in which the MUSBMHDRC behaves depends on whether it is linked up for point-to-point communications with another USB function or whether it is attached to a hub. When attached to another USB function, the MUSBMHDRC offers the range of capabilities needed in order to act as the host in point-to-point communications with this USB function. When attached to a hub, it provides the facilities required to act as the host to a number of devices, supported simultaneously.

When operating in Host mode and used for point-to-point communications with a single other USB device (which can be high-, full- or low-speed), the MUSBMHDRC can support Control, Bulk, Isochronous or Interrupt transactions. IN transactions are handled through the Rx FIFOs, OUT transactions are handled through the TX FIFOs. As well as encoding, decoding and checking the USB packets sent and received, the MUSBMHDRC will also automatically schedule Isochronous endpoints and Interrupt endpoints to perform one transaction every n frames/microframes (or up to three transactions if the high-bandwidth option is selected), where n represents the polling interval that has been programmed for the endpoint. The remaining bus bandwidth is shared equally amongst the Control and Bulk endpoints (see Section 8.5.4 Transaction Scheduling).

When attached to a hub, the MUSBMHDRC continues to offer the above facilities but it further needs to be programmed with details of:

- The function address of the target device.

- The operating speed of the target device (so that the appropriate speed conversion can be carried out).

- If the target device is a full- or low-speed device that is accessed through a high-speed hub, the endpoint additionally needs to be programmed with the function address and port number of the hub.

The device may be required to power the VBus to 5V as the 'A' device of the connection (source of power and default host) or, as the 'B' device (default peripheral), to be able to wake the 'A' device by charging the VBus to 2V. Outputs from the MUSBMHDRC indicate when these charging options are required.

Whether the MUSBMHDRC initially operates in Host mode or in Peripheral mode depends on whether it is being used in an 'A' device or a 'B' device, which in turn depends on whether the IDDIG input is low or high. When the MUSBMHDRC is operating as an 'A' device, it is initially configured to operate in Host mode. When operating as a 'B' device, the MUSBMHDRC is initially configured to operate in Peripheral mode. However, a "Host Req" bit is provided in the DevCtl register through which the CPU can request that the 'B' device becomes the Host the next time there is no activity on the USB bus.

The IDDIG input reflects the state of the ID pin of the device's mini-AB receptacle, with IDDIG being low indicating an 'A' plug i.e. operation as an 'A' device, and IDDIG being high indicating a 'B' plug and operation as a 'B' device.

Information on whether the MUSBMHDRC is acting as an 'A' device or as a 'B' device and on whether the device it is connected to is high-, full- or low-speed is also recorded in the DevCtl register, along with information about the level of the VBus relative to the high and low voltage thresholds used to signal Session Start and Session End.

The procedures for session request and for transferring host/peripheral roles between the devices at either end of the connection are described in Sections 14 and 15, respectively. The transfers that are made are all subject to the standard USB data transfer protocols.

CONFIDENTIAL

SOFT CORES

2.2. BLOCK DIAGRAM

The following block diagram shows the main functional blocks within the MUSBMHDRC. (A block diagram of any bridge that is provided for use with the core is given in the separate specification for that bridge included in the **musbmhdrc/docs** directory.)
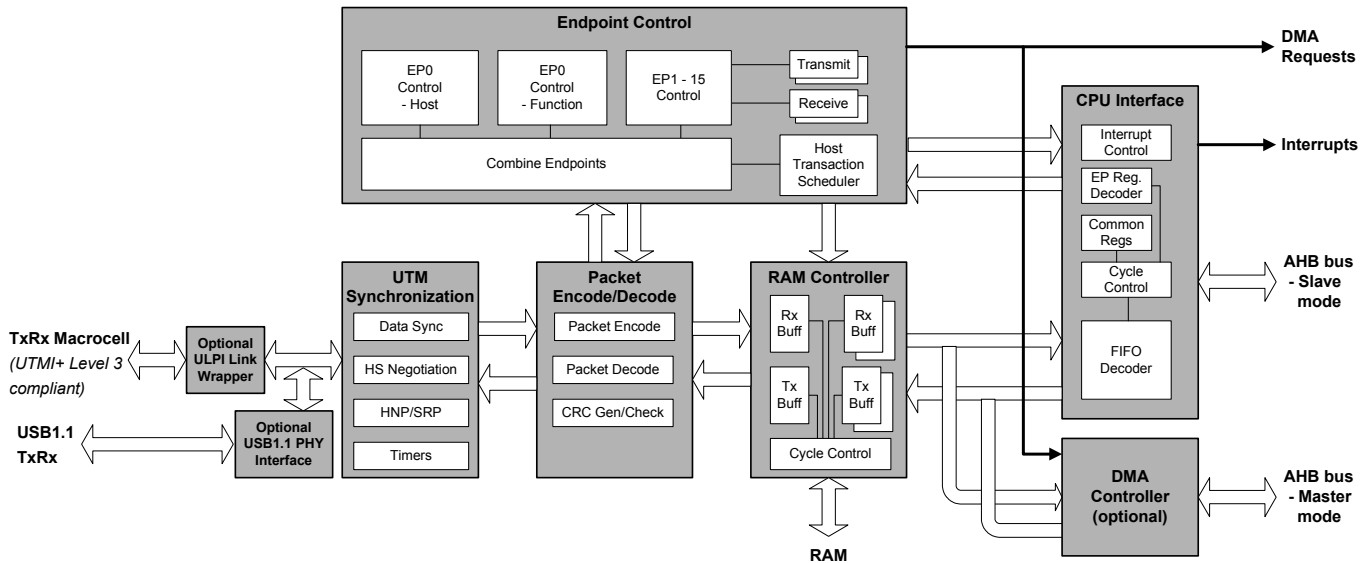
2.3. UTM SYNCHRONIZATION

The role of the UTM Synchronization block is to resynchronize between the transceiver macrocell 60MHz clock domain and the dual role controller's system clock CLK, which drives the remainder of the core up to and including the CPU interface. This allows the rest of the MUSBMHDRC to run at the CPU bus speed without requiring any further synchronization. The block also performs High-speed detection handshaking and handles HNP and SRP in point-to-point communications with another USB OTG device.

Using an eight bit interface, the block first converts the data to 16-bit – requiring the core to be driven by a system clock running at least over 30MHz. The actual the actual minimum frequency required by the system clock for data to be correctly transferred over the domain crossing is a function of technology implementation. This actual minimum frequency is defined in detail in section 4.

CONFIDENTIAL

2.4. PACKET ENCODING/DECODING

The Packet Encode/Decode block generates headers for packets to be transmitted and decodes the headers on received packets. It also generates the CRC for packets to be transmitted and checks the CRC on received packets.

2.5. ENDPOINT CONTROLLERS

Two controller state machines are used: one for control transfers over Endpoint 0 and one for Bulk/Interrupt/Isochronous transactions over Endpoints 1 to 15.

2.6. CPU INTERFACE

The CPU Interface allows access to the control/status registers and the FIFOs for each endpoint. It also generates interrupts to the CPU when packets are successfully transmitted or received, and when the core enters Suspend mode or resumes from Suspend mode.

The interface provided by the MUSBMHDRC is a 32-bit synchronous interface that follows the design specified for interfaces to an AMBA AHB bus. Interface to other bus standards may be achieved through the addition of an appropriate wrapper to the core.

2.7. RAM CONTROLLER

The RAM controller provides an interface to a single block of synchronous single-port RAM, which is used to buffer packets between the CPU and USB. It takes the FIFO pointers from the endpoint controllers, converts them to address pointers within the RAM block and generates the RAM access control signals.

2.8. DMA CONTROLLER SUPPORT

If required, the MUSBMHDRC may include a multi-channel DMA controller for efficient loading/unloading of the endpoint FIFOs. This DMA controller is configurable for up to 8 channels.

The DMA controller has its own block of control registers and its own interrupt controller. It supports two modes of operation and each channel can be independently programmed for operating mode

Alternativly, the MUSBMHDRC may be integrated with an external DMA controller for efficient loading/unloading of the endpoint FIFOs. The MUSBMHDRC outputs DMA request signals for each endpoint.

CONFIDENTIAL

2.9. TREE DIAGRAM

The following tree diagram shows the hierarchical structure of the MUSBMHDRC. (The modules associated with any bridge that is provided for use with the core are shown in the equivalent tree diagram given in the separate specification for that bridge included in the **musbmhdrc/docs** directory.) *Note:* When configured for use with the optional ULPI Link Wrapper, the core also includes a lpictl module which provides the ULPI Control registers (detailed in the ULPI Link Wrapper application note, provided as the file **musbhdrc_ulpi_an.pdf** included in the **docs** directory).
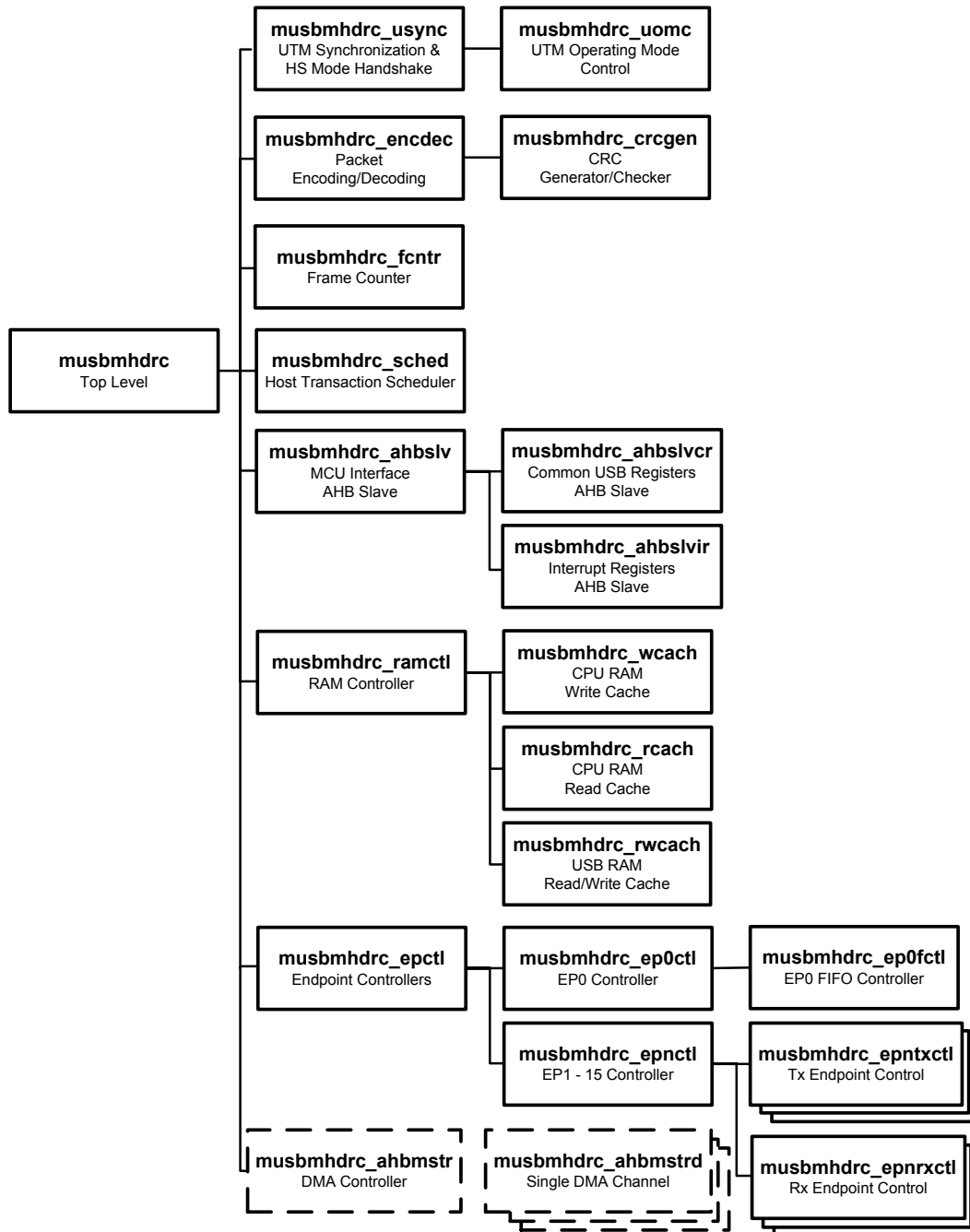
Figure 1

The MUSBMHDRC can be configured with regard to:

1. The number of TX endpoints, in addition to Endpoint 0. Namely 1, 3, 5, 7, 11, or 15 additional TX endpoints

2. The number of Rx endpoints, in addition to Endpoint 0. Namely 1, 3, 5, 7, 11, or 15 additional Rx endpoints

3. Whether any of these endpoints support high-bandwidth Isochronous transfers.

4. The size of FIFO associated with each endpoint (excluding Endpoint 0), or alternatively the total RAM size to be assigned dynamically to the different endpoints (see Section 19).

5. Which FIFOs (if any) are shared between a TX endpoint and the correspondingly numbered Rx endpoint (unless sized dynamically).

6. Whether the option of automated splitting/combining of packets is required for Bulk transfers (see Sections 8.4.1.4 and 8.4.2.4).

7. Whether a version of the core that uses the UTMI+ interface includes the UTMI+ VControl and VStatus registers, and the size of these registers if included (see Sections 3.6.1 and 3.6.2).

8. Whether the MUSBHDRC's support for multipoint (hub-support) is utilized. If utilized additional logic is enabled to allow the MUSBHDRC to manage connections through a USB hub.

9. The number of DMA Channels supported where the built-in DMA controller is used (0 if external DMA controller is used).

There can be up to 15 TX endpoints and/or up to 15 Rx endpoints in addition to Endpoint 0. Each TX endpoint is used as an IN endpoint when the MUSBMHDRC is being used in Peripheral mode, and as an OUT endpoint when the MUSBMHDRC is being used in Host mode. Similarly each Rx endpoint is used as an OUT endpoint when the MUSBMHDRC is used in Peripheral mode, and as an IN endpoint when the MUSBMHDRC is used in Host mode.

The FIFO for Endpoint 0 is required to be 64 bytes in size and will buffer one packet. The FIFOs for the other endpoints can be specified to be 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096 or 8192 bytes and, with the exception of 8-byte FIFOs, can be used to buffer either one or two packets.

However, FIFO sizes greater than 2048 bytes should only be used in conjunction with high-bandwidth Isochronous endpoints. Where required, the MUSBMHDRC will automatically split / re-combine packets of up to 3072 bytes (3k) into 2 or 3 smaller packets for transmission / reception over the bus.

(*Note:* Double-buffering is optional for Bulk or Interrupt transfers but is usually required for Isochronous transfers. You should also note the constraints placed by the USB Specification on the packet sizes for Bulk, Interrupt and Isochronous transfers in full-speed operations.)

Separate FIFOs may be associated with each endpoint: alternatively a TX endpoint and the Rx endpoint with the same endpoint number can be configured to use the same FIFO, for example to reduce the size of RAM block needed.

Further configuration options may be associated with any bridge that is provided for use with the MUSBMHDRC core (described in the separate specification for that bridge included in the **musbmhdrc/docs** directory).

CONFIDENTIAL

SOFT
CORES

Configuration is performed by running a graphical user interface script (similar to that illustrated below) prior to simulation or synthesis. The steps are explained in Section 5 of the MUSBMHDRC User Guide and in the **config.readme** file included in the simulation directory. The configuration files (*_cfg.v) should not be directly hand edited. Configuration files created by the user are required to be the result of the delivered configuration files modified only by the delivered configuration GUI's.

Please Note: Separate scripts may be provided for use where a bridge/wrapper is added to the core. For details, see either the **config.readme** file included in the simulation directory or the appropriate bridge/wrapper specification. A special script (**config_fsp.tcl**) is also provided for configuring the core where the optional USB 1.1 PHY interface is used. (Further information on this is given in Section 8.1.)

You should also note that the configuration screen display includes both an estimated gate count for the selected core configuration and a count of the amount of RAM that will be required.

The core is issued with the configuration shown when the Configuration GUI is first displayed.

CONFIDENTIAL

Please Note: This section describes the signal I/O of the MUSBMHDRC core itself. The signal I/O when the optional USB 1.1 PHY interface is used with the core is described in Section 8.1. The signal I/O when a bridge has been added is described in the appropriate bridge specification included in the **musbmhdrc/docs** directory.

The MUSBMHDRC core has a maximum of 438 external signals; 182 inputs and 256 outputs. All inputs are sampled on the positive (rising) edge of the relevant clock, and outputs change following positive clock edges.

MUSBMHDRC

Inputs	Block	Outputs
XCLK	XCVR INTERFACE (UMTI+ Level 3 compatible)	SUSPENDM
LINESTATE[1:0]		OPMODE[1:0]
XDATAIN[utm_width-1:0]		XDATAOUT[utm_width-1:0]
TXREADY		TXVALID
RXVALID		
RXACTIVE		XCVRSEL [1:0]
RXERROR		TERMSEL
VBUSVALID		DRVVBUS
AVALID		CHRGVBUS
SESSEND		DISCHRGVBUS
HOSTDISCON		DPPULLDOWN
IDDIG		DMPULLDOWN
		IDPULLUP
VSTATUS[status_bits-1:0]		*VCONTROL[cntrl_bits-1:0]*
		VCONTROLLOADM
AHB_HSEL	CPU INTERFACE (AMBA AHB Slave)	AHB_HREADYO
AHB_HREADYI		
AHB_HSIZE[1:0]		
AHB_HTRANS[1:0]		AHB_HRDATA[31:0]
AHB_HADDR[9:0]		
AHB_HWDATA[31:0]		
AHB_HWRITE		MC_NINT
	DMA INTERFACE (AMBA AHB Master) (Optional)	AHB_HBUSREQ
		AHB_HADDRM[31:0]
AHB_HGRANT		AHB_HWDATAM[31:0]
AHB_HREADYMI		AHB_HTRANSM[1:0]
AHB_HRDATAM[31:0]		AHB_HSIZEM[1:0]
		AHB_HBURSTM[2:0]
AHB_HRESPM[1:0]		AHB_HWRITEM
		DMA_NINT
	RAM INTERFACE	RAM_NCE
RAM_DATAI[31:0]		RAM_ADDR[ram_bits-1:0]
		RAM_DATAO[31:0]
		RAM_NWR
CLK	SYSTEM	USB_NRSTO
NRSTA		SOF_PULSE
NRSTO		POWERDWN
NRST		
NRSTXO		DMA_REQ[total_eps-3:0]
NRSTX		
TM1		

Implementation dependent (for VCONTROL / VCONTROLLOADM)

CONFIDENTIAL

SOFT CORES

UTMI+ INTERFACE SIGNALS (LEVEL 3)		
SIGNAL	**TYPE**	**DESCRIPTION**
XCLK	Input	Transceiver macrocell clock. 60MHz.
SUSPENDM	Output	Asynchronous Suspend mode indicator (derived from signals from both CLK and XCLK flip-flops). When enabled through bit 0 of Power register, goes low when device in Suspend mode. Otherwise high. (Intended to drive a UTMI PHY.)
LINESTATE[1:0]	Input	Shows the current state of single-ended receivers. LINESTATE[0] reflects the state of D+; LINESTATE[1] reflects state of D-. Thus: LINESTATE[1:0] 0 0 SE0 0 1 'J' State 1 0 'K' State 1 1 SE1
OPMODE[1:0]	Output	Operating mode selector OPMODE[1:0] 0 0 Normal operation 0 1 Non-Driving 1 0 Bit stuffing and NRZI encoding disabled 1 1 *Reserved*
XDATAIN[7:0]	Input	Received data.
XDATAOUT[7:0]	Output	Data to be transmitted..
TXVALID	Output	Transmit data valid. Indicates there is valid data to be transmitted.
TXREADY	Input	Transmit data ready. Indicates that the transmitter requires data.
RXVALID	Input	Receive data valid. Indicates that valid data has been received.
RXACTIVE	Input	Indicates that a valid packet is being received.
RXERROR	Input	Indicates that the packet being received is about to be aborted due to an error.
XCVRSEL [1:0]	Output	Transceiver select. XCVRSEL [1:0] 0 0 HS transceiver 0 1 FS transceiver 1 0 LS transceiver 1 1 FS transceiver, LS packet
TERMSEL	Output	Termination select. When 0, High-speed termination is enabled; when 1, Full-speed termination is enabled. *Note:* May be used to switch the pull-up resistor on D+.
VBUSVALID	Input	VBus compared to selected VBus Valid threshold (required to be between 4.4V and 4.75V). 1 = above the VBus Valid threshold, 0 = below the VBus Valid threshold.
AVALID	Input	VBus compared to Session Valid threshold for a 'B' device (required to be between 0.8V and 2V). 1 = above the Session Valid threshold, 0 = below the Session Valid threshold.
SESSEND	Input	VBus compared to Session End threshold (required to be between 0.2V and 0.8V). 0 = above the Session End threshold, 1 = below the Session End threshold.
DRVVBUS	Output	VBus power enable (used when MUSBMHDRC operating as an 'A' device).
CHRGVBUS	Output	Charge VBus (used during Session Request when MUSBMHDRC operating as 'B' device).
DISCHRGVBUS	Output	Discharge VBus (used by 'B' devices to ensure that VBus is low enough before starting Session Request Protocol (SRP)).

CONFIDENTIAL

UTMI+ INTERFACE SIGNALS (LEVEL 3)		
SIGNAL	**TYPE**	**DESCRIPTION**
HOSTDISCON	Input	*(Host mode only.)* Required to be asserted when a High-Speed disconnect occurs (in accordance with the UTMI+ Specification). *Note:* Full/Low-Speed connections are monitored via the LINESTATE signal.
DPPULLDOWN	Output	Enable for a pull-down resistor within the transceiver on the D+ line. Low when MUSBMHDRC operating as a peripheral; high when MUSBMHDRC operating as a host.
DMPULLDOWN	Output	Enable for a pull-down resistor within the transceiver on the D– line. Needs to be high when the MUSBMHDRC is being used for point-to-point communications.
IDDIG	Input	Indicates MUSBMHDRC connector type. High=>B-type, low=>A-type.
IDPULLUP	Output	Enable for IDDIG signal generation.
VSTATUS[cntrl_bits-1:0]	Input	PHY Status data (configurable up to 32 bits wide) – *if implemented*.
VCONTROL[cntrl_bits-1:0]	Output	PHY Control data (configurable up to 32 bits wide) – *if implemented*.
VCONTROLLOADM	Output	Active low signal, asserted when new Control information to be read – *if implemented*.
CPU INTERFACE SIGNALS (AMBA AHB Slave) *		
AHB_HSEL	Input	AHB select. Taken high to select MUSBMHDRC device.
AHB_HREADYI	Input	AHB ready input.
AHB_HREADYO	Output	AHB ready output.
AHB_HSIZE[1:0]	Input	AHB transfer size.
AHB_HTRANS[1:0]	Input	AHB transfer type.
AHB_HADDR[9:0]	Input	AHB address bus.
AHB_HWDATA[31:0]	Input	AHB write data bus.
AHB_HRDATA[31:0]	Output	AHB read data bus.
AHB_HWRITE	Input	AHB write not read
MC_NINT	Output	CPU interrupt. Active low.
DMA INTERFACE SIGNALS (AMBA AHB Master) – Optional		
AHB_HGRANT	Input	AHB bus master grant.
AHB_HREADYMI	Input	AHB master ready input.
AHB_HRDATAM[31:0]	Input	AHB read data bus (master mode)
AHB_HRESPM[1:0]	Input	AHB response (master mode).
AHB_HBUSREQ	Output	AHB bus master request.
AHB_HADDRM[31:0]	Output	AHB address bus (master mode).
AHB_HWDATAM[31:0]	Output	AHB write data bus (master mode).
AHB_HTRANSM[1:0]	Output	AHB transfer type (master mode).
AHB_HSIZEM[1:0]	Output	AHB transfer size (master mode).
AHB_HBURSTM[2:0]	Output	AHB burst mode (master mode).
AHB_HWRITEM	Output	AHB write not read (master mode)
DMA_NINT	Output	DMA controller interrupt. Active low.

* Where the core is used with a bridge, the device will have a different set of CPU interface signals – detailed in the separate specification for that bridge, included in the **musbmhdrc/docs** directory.

CONFIDENTIAL

SOFT CORES

RAM INTERFACE SIGNALS		
SIGNAL	TYPE	DESCRIPTION
RAM_ADDR[ram_bits–1:0]	Output	RAM address bus. Width is dependent on the number and type of endpoints configured.
RAM_DATAI[31:0]	Input	RAM data input bus.
RAM_DATAO[31:0]	Output	RAM data output bus.
RAM_NCE	Output	RAM select. Active low.
RAM_NWR	Output	RAM write enable. Active low.
SYSTEM SIGNALS		
CLK	Input	System clock (provided by AHB bus clock). This clock needs to be at least >30MHz. See section 4 for actual minimum.
NRSTA	Input	Asynchronous power up reset, Active low.
NRST	Input	Reset synchronous with CLK. Active low. Typically connected to output NRSTO.
NRSTX	Input	Reset synchronous with XCLK. Active low. Typically connected to output NRSTXO.
TM1	Input	Test Mode. Used by the supplied test benches to reduce timer length and hence the time taken for the test bench to run. For normal operation, this signal should be tied low.
NRSTO	Output	This signal is equal to the Asynchronous Input NRSTA after synchronizing to the CLK clock domain. Active low. This signal can also be asserted via register 7Fh (Soft Reset). Typically connected to input NRST.
NRSTXO	Output	This signal is equal to the Asynchronous Input NRSTA after synchronizing to the XCLK clock domain. Active low. This signal can also be asserted via register 7Fh (Soft Reset). Typically connected to input NRSTX.
USB_NRSTO	Output	USB function reset. Active low. This reset is asserted when the function controller is reset by USB signaling.
SOF_PULSE	Output	Frame Sync Pulse. Pulse length is 1 CLK period, pulse frequency is 1kHz in Full-speed/Low-speed mode, or 8kHz in High-speed mode, synchronized in Peripheral mode to received SOF/uSOF packets.
POWERDWN	Output	Asserted when CLK may be stopped to save power. (*Note:* Derived from combination of signals from CLK & XCLK flip-flops, AVALID, VBUSVALID and LINESTATE.)
DMA_REQ[total_eps–3:0]	Output	DMA endpoint requests, one for each additional Rx Endpoint and TX Endpoint. If a total of N TX Endpoints and M Rx Endpoints are defined, DMA_REQ[0] ... DMA_REQ[N–2] are associated with TX Endpoints 1 ... N–1; DMA_REQ[N–1] ... DMA_REQ[$N+M$–3] are associated with Rx Endpoints 1 ... M–1.

CONFIDENTIAL

SOFT CORES

3. REGISTER DESCRIPTION

3.1. MUSBMHDRC REGISTER MAP

The MUSBMHDRC register map is split into the following sections:

Common USB registers (00h–0Fh) – These registers provide control and status for the complete core.

Indexed Endpoint Control/Status registers (10h–1Fh) – These registers provide control and status for the currently selected endpoint. The registers mapped into this section depend on whether the core is in Peripheral mode (DevCtl.D2=0) or in Host mode (DevCtl.D2=1) and on the value of the Index register.

FIFOs (20h–5Fh) – This address range provides access to the endpoint FIFOs.

Additional Control and Configuration registers (60h–7Fh) – These registers provide additional device status and control.

Target Endpoint Control Registers (80h–FFh) – When the multipoint option is enabled in the configuration GUI, these registers provide target function and hub address details for each of the endpoints. These registers can only be accessed when the multipoint option is enabled.

Non-Indexed Endpoint Control/Status registers (100h and above) – The registers available at 10h–1Fh, accessible independently of the setting of the Index register. 100h–10Fh EP0 registers; 110h–11Fh EP1 registers; 120h–12Fh EP2; and so on.

DMA Control Registers (200h and above) – These registers only appear if the design is synthesized to include optional DMA controller (see Section 17).

RqPktCount Registers (304h – 33Ch) – These registers are used in Host mode in conjunction with AutoReq (see Section 8.5.2).

DPktBufDis Registers (340h – 343h) – These registers provide direct user control over disabling double packet buffering.

C_T_UCH Registers (344h – 345h) – These registers set the Chirp Timeout timer.

C_T_HSRTN Registers (346h – 347h) – These registers set the delay from the end of High Speed resume signaling to enabling UTM normal operating mode.

The resulting Memory Map is illustrated in the following diagram.

CONFIDENTIAL

MUSBMHDRC Memory Map
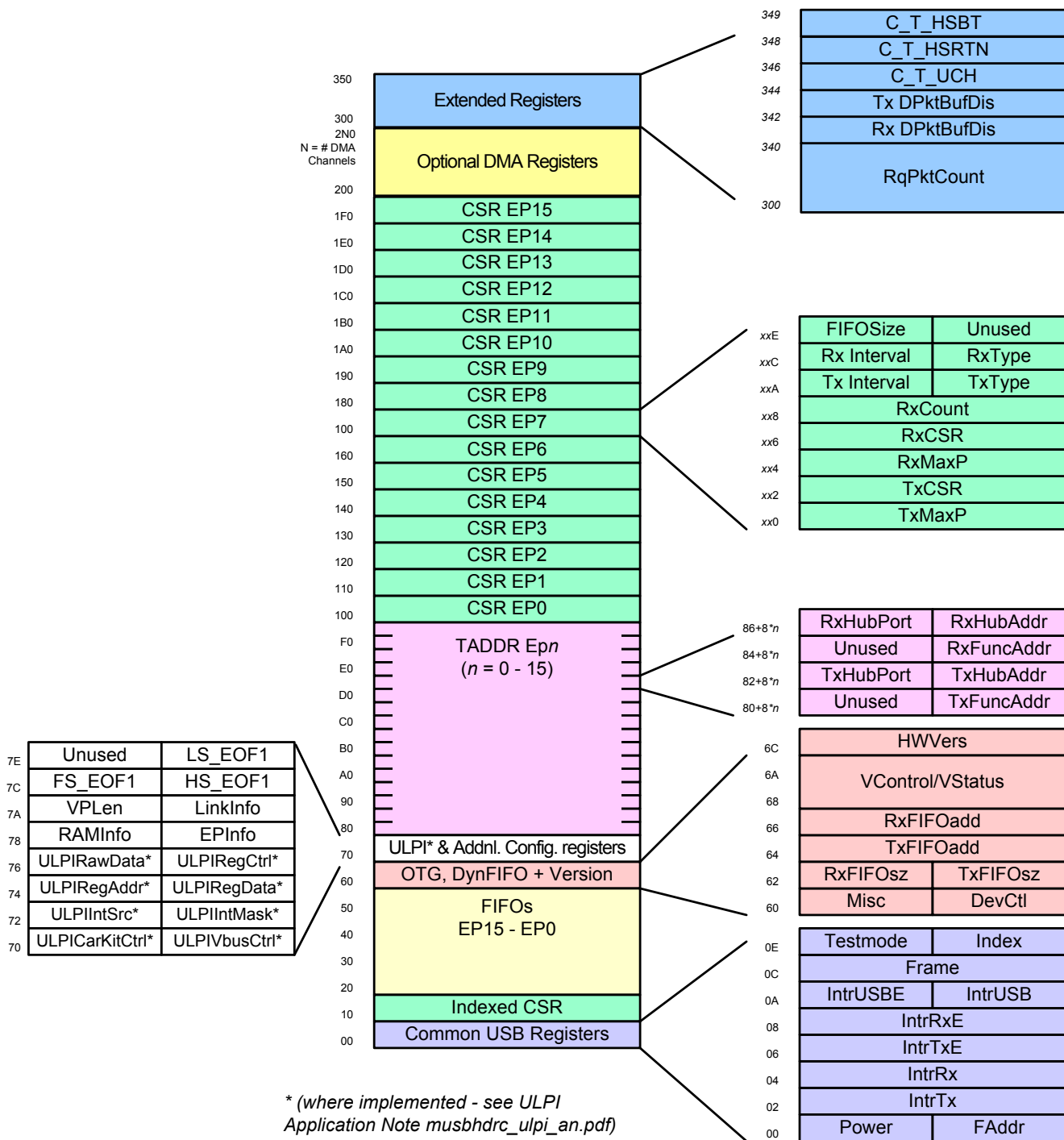
Figure 2

Note: Additional registers associated with any bridge provided for use with the MUSBMHDRC core or any changes to the following registers that result from using this bridge will be described in the separate specification for the bridge included in the **musbmhdrc/docs** directory.

CONFIDENTIAL

MUSBMHDRC REGISTER MAP: Common USB registers (00h – 0Fh)			
ADDR	**NAME**	**DESCRIPTION**	**See Section**
00	FAddr	Function address register.	3.2.1
01	Power	Power management register.	3.2.2
02,03	IntrTx	Interrupt register for Endpoint 0 plus TX Endpoints 1 to 15.	3.2.3
04,05	IntrRx	Interrupt register for Rx Endpoints 1 to 15.	3.2.4
06,07	IntrTxE	Interrupt enable register for IntrTx.	3.2.5
08,09	IntrRxE	Interrupt enable register for IntrRx.	3.2.6
0A	IntrUSB	Interrupt register for common USB interrupts.	3.2.7
0B	IntrUSBE	Interrupt enable register for IntrUSB.	3.2.8
0C,0D	Frame	Frame number.	3.2.9
0E	Index	Index register for selecting the endpoint status and control registers.	3.2.10
0F	Testmode	Enables the USB 2.0 test modes.	3.2.11

MUSBMHDRC REGISTER MAP: Indexed registers – Peripheral mode (10h – 1Fh) (Control Status registers for endpoint selected by the Index register when DevCtl.D2 = 0)			
ADDR	**NAME**	**DESCRIPTION**	**See Section**
10,11	TxMaxP	Maximum packet size for peripheral TX endpoint. (Index register set to select Endpoints 1 – 15 only)	3.3.7
12,13	CSR0L/H	Control Status register for Endpoint 0. (Index register set to select Endpoint 0)	3.3.1
	TxCSRL/H	Control Status register for peripheral TX endpoint. (Index register set to select Endpoints 1 – 15)	3.3.2
14,15	RxMaxP	Maximum packet size for peripheral Rx endpoint. (Index register set to select Endpoints 1 – 15 only)	3.3.10
16,17	RxCSRL/H	Control Status register for peripheral Rx endpoint. (Index register set to select Endpoints 1 – 15 only)	3.3.11
18,19	Count0	Number of received bytes in Endpoint 0 FIFO. (Index register set to select Endpoint 0)	3.3.3
	RxCount	Number of bytes to be read from peripheral Rx endpoint FIFO. (Index register set to select Endpoints 1 – 15)	3.3.13
1A–1B	–	*Reserved*. Value returned affected by use in Host mode (see following page).	
1C–1E	–	*Unused*, always return 0.	
1F	ConfigData	Returns details of core configuration. (Index register set to select Endpoint 0.)	3.3.5

CONFIDENTIAL

SOFT
CORES

	FIFOSize	Returns the configured size of the selected Rx FIFO and TX FIFOs (Endpoints 1 – 15 only).	3.3.18

MUSBMHDRC REGISTER MAP: Indexed registers – Host mode (10h – 1Fh) (Control Status registers for endpoint selected by the Index register when DevCtl.D2 = 1)			
ADDR	**NAME**	**DESCRIPTION**	**See Section**
10,11	TxMaxP	Maximum packet size for host TX endpoint. (Index register set to select Endpoints 1 – 15 only)	3.3.7
12,13	CSR0L/H	Control Status register for Endpoint 0. (Index register set to select Endpoint 0)	3.3.1
	TxCSRL/H	Control Status register for host TX endpoint. (Index register set to select Endpoints 1 – 15)	3.3.2
14,15	RxMaxP	Maximum packet size for host Rx endpoint. (Index register set to select Endpoints 1 – 15 only)	3.3.10
16,17	RxCSRL/H	Control Status register for host Rx endpoint. (Index register set to select Endpoints 1 – 15 only)	3.3.11
18,19	Count0	Number of received bytes in Endpoint 0 FIFO. (Index register set to select Endpoint 0)	3.3.3
	RxCount	Number of bytes to be read from host Rx endpoint FIFO. (Index register set to select Endpoints 1 – 15)	3.3.13
1A	Type0	Defines the speed of Endpoint 0. (Index register set to select Endpoint 0)	3.3.4
	TxType	Sets the transaction protocol, speed and peripheral endpoint number for the host TX endpoint. (Index register set to select Endpoints 1 – 15)	3.3.14
1B	NAKLimit0	Sets the NAK response timeout on Endpoint 0. (Index register set to select Endpoint 0)	3.3.6
	TxInterval	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host TX endpoint. (Index register set to select Endpoints 1 – 15 only)	3.3.15
1C	RxType	Sets the transaction protocol, speed and peripheral endpoint number for the host Rx endpoint. (Index register set to select Endpoints 1 – 15 only)	1.1.1
1D	RxInterval	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Rx endpoint. (Index register set to select Endpoints 1 – 15 only)	3.3.17
1E	–	*Unused*, always returns 0.	
1F	ConfigData	Returns details of core configuration. (Index register set to select Endpoint 0.)	3.3.5
	FIFOSize	Returns the configured size of the selected Rx FIFO and TX FIFOs (Endpoints 1 – 15 only).	3.3.18

MUSBMHDRC REGISTER MAP: FIFOs(20h – 5Fh)			
ADDR	**NAME**	**DESCRIPTION**	**See Section**
20 – 5F	FIFOx	FIFOs for Endpoints 0 – 15.	3.4

CONFIDENTIAL

SOFT CORES

ADDR	NAME	DESCRIPTION		See Section
MUSBMHDRC REGISTER MAP: Additional Control & Configuration Registers (60h – 7Fh)				
60	DevCtl	OTG device control register.		3.2.12
61	MISC	Miscellaneous Register		3.2.13
62	TxFIFOsz	TX Endpoint FIFO size	*Only used if Dynamic FIFO sizing option is selected. Otherwise return 0.*	3.3.18
63	RxFIFOsz	Rx Endpoint FIFO size		
64,65	TxFIFOadd	TX Endpoint FIFO address		
66,67	RxFIFOadd	Rx Endpoint FIFO address		
68–6B	VControl/ VStatus	UTMI+ PHY Vendor registers		3.6.1, 3.6.2
6C,6D	HWVers	Hardware Version Number Register		3.6.3
6E,6F	–	*Unused*		
70 – 77	–	*ULPI Registers, only implemented where ULPI Link Wrapper is used. See ULPI Application Note* **musbhdrc_ulpi_an.pdf**.		
78	EPInfo	Information about numbers of TX and Rx endpoints.		3.7.1
79	RAMInfo	Information about the width of the RAM and the number of DMA channels.		3.7.2
7A	LinkInfo	Information about delays to be applied.		1.1.1
7B	VPLen	Duration of the VBus pulsing charge.		3.7.4
7C	HS_EOF1	Time buffer available on High-Speed transactions.		3.7.5
7D	FS_EOF1	Time buffer available on Full-Speed transactions.		3.7.6
7E	LS_EOF1	Time buffer available on Low-Speed transactions.		3.7.7
7F	SOFT_RST	Soft Reset.		3.7.8

		MUSBMHDRC REGISTER MAP: Target Address Registers (80h – FFh) (These Registers are only valid if the Multipoint Option is enabled in the configuration GUI)		
80+8*n*	TxFuncAddr	Transmit Endpoint *n* Function Address (Host Mode only)	Multipoint-Only	3.5.1
81+8*n*	–	*Unused*, always returns 0.	Multipoint-Only	
82+8*n*	TxHubAddr	Transmit Endpoint *n* Hub Address (Host Mode only)	Multipoint-Only	3.5.2
83+8*n*	TxHubPort	Transmit Endpoint *n* Hub Port (Host Mode only)	Multipoint-Only	3.5.3
84+8*n*	RxFuncAddr	Receive Endpoint *n* Function Address (Host Mode only)	Multipoint-Only	3.5.1
85+8*n*	–	*Unused*, always returns 0.	Multipoint-Only	
86+8*n*	RxHubAddr	Receive Endpoint *n* Hub Address (Host Mode only)	Multipoint-Only	3.5.2
87+8*n*	RxHubPort	Receive Endpoint *n* Hub Port (Host Mode only)	Multipoint-Only	3.5.3

CONFIDENTIAL

SOFT CORES

MUSBMHDRC REGISTER MAP: Extended Registers (200h – 27Ch)				
ADDR	**NAME**	**DESCRIPTION**		**See Section**
200h	DMA_INTR	DMA Interupt register.	The DMA registers are only available if the MUSBMHDRC is configured to use at least one internal DMA channel. There is one set of registers per channel.	3.9.1
204h+ (n-1)*10h	DMA_CNTL	DMA Control Register for DMA channel n. (channel 1 thru 8).		0
208h + (n-1)*10h	DMA_ADDR	DMA Address Register for DMA channel n (channel 1 thru 8).		3.9.3
20Ch + (n-1)*10h	DMA_COUNT	DMA Count Register for DMA channel n (channel 1 thru 8).		3.9.4

MUSBMHDRC REGISTER MAP: Extended Registers (304h – 347h)			
ADDR	**NAME**	**DESCRIPTION**	**See Section**
300+4*n	RqPktCount	Number of requested packets for Receive Endpoint n (Endpoints 1 – 15 only)	3.8.1
340, 341	Rx DPktBufDis	Double Packet Buffer Disable register for Rx Endpoints 1 to 15	3.8.2.1
342, 343	TX DPktBufDis	Double Packet Buffer Disable register for TX Endpoints 1 to 15	3.8.2.2
344, 345	C_T_UCH	This register sets the Chirp Timeout Timer	3.8.3
346, 347	C_T_HSRTN	This register sets the delay from the end of High Speed resume signaling to enable UTM normal operating mode.	3.8.4
348	C_T_HSBT	This register specifies the value added to the minimum High Speed Timeout period (736 bit times) in increments of 64 High Speed bit times.	3.8.5

CONFIDENTIAL

Note: In the following bit descriptions:

'r' means that the bit is read only   'rw' means that the bit can be both read and written
'set' means that the bit can only be written to set it   'r/set' means that the bit can be read or set but it can't be cleared
'clear' means that the bit can only be written to clear it   'r/clear' means that the bit can be read or cleared but it can't be set
'self-clearing' means the bit will be cleared automatically when the associated action has been executed.

3.2. COMMON REGISTERS

– described in Address order.

3.2.1. FADDR

FAddr is an 8-bit register that should be written with the 7-bit address of the peripheral part of the transaction.

When the MUSBMHDRC is being used in Peripheral mode (DevCtl.D2=0), 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.

Notes: <u>Peripheral Mode Only!!</u>

<u>Core Configured with Multipoint support:</u> This register is only applies to operations carried out when the MUSBMHDRC is in Peripheral mode. In Host mode, this register is ignored.

<u>Core Configured with-out Multipoint support:</u> This register is applicable to operations carried out when the MUSBMHDRC is Host and Peripheral mode.

Address: 00h*; Reset value:* 8'h00

	D7	D6	D5	D4	D3	D2	D1	D0
	0	**Function Address**						
From CPU	r	rw	rw	rw	rw	rw	rw	rw
From USB	-	r	R	r	r	r	r	r

Bit	Name	Function
D7	**-**	*Unused, always returns 0.*
D6 – D0	**Func Addr**	The function address.

3.2.2. POWER

Power is an 8-bit register that is used for controlling Suspend and Resume signaling, and some basic operational aspects of the MUSBMHDRC.

Address: 01h*; Reset value:* 8'h20

		D7	D6	D5	D4	D3	D2	D1	D0
		ISO Update	**Soft Conn**	**HS Enab**	**HS Mode**	**Reset**	**Resume**	**Suspend Mode**	**Enable SuspendM**
Periphera	CPU	rw	rw	rw	r	r	rw	r	rw
mode	USB	r	r	r	rw	rw	r	rw	r

CONFIDENTIAL

SOFT CORES

Host	CPU	–	–	rw	r	rw	rw	set	rw
mode	USB	–	–	r	rw	rw	r/set	clear	r

Bit	Name	Function
D7	**ISO Update**	When set by the CPU, the MUSBMHDRC 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: Only valid in Peripheral Mode. Also, this bit only affects endpoints performing Isochronous transfers.*
D6	**Soft Conn**	If Soft Connect/Disconnect feature is enabled, then the USB D+/D- lines are enabled when this bit is set by the CPU and tri-stated when this bit is cleared by the CPU. (See Section 8.2) *Note: Only valid in Peripheral Mode.*
D5	**HS Enab**	When set by the CPU, the MUSBMHDRC 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.
D4	**HS Mode**	When set, this read-only bit indicates High-speed mode successfully negotiated during USB reset. In Peripheral Mode, becomes valid when USB reset completes (as indicated by USB reset interrupt). In Host Mode, becomes valid when Reset bit is cleared. Remains valid for the duration of the session. *Note:* Allowance is made for Tiny-J signaling in determining the transfer speed to select.
D3	**Reset**	This bit is set when Reset signaling is present on the bus. *Note: This bit is Read/Write from the CPU in Host Mode but Read-Only in Peripheral Mode.*
D2	**Resume**	Set by the CPU to generate Resume signaling when the device is in Suspend mode. In Peripheral mode, the CPU should clear this bit after 10 ms (a maximum of 15 ms), to end Resume signaling. In Host mode, the CPU should clear this bit after 20 ms.
D1	**Suspend Mode**	In Host mode, this bit is set by the CPU to enter Suspend mode. In Peripheral mode, this bit is set on entry into Suspend mode. It is cleared when the CPU reads the interrupt register, or sets the Resume bit above.
D0	**Enable SuspendM**	Set by the CPU to enable the SUSPENDM output.

3.2.3. INTRTX

IntrTx is a 16-bit read-only register that indicates which interrupts are currently active for Endpoint 0 and the TX Endpoints 1–15. *Note:* Bits relating to endpoints that have not been configured will always return 0. Note also that all active interrupts are cleared when this register is read.

Address: 02h*; Reset value:* 16'h0000

	D15	D14	D13	D12	D11	D10	D9	D8
	EP15 Tx	**EP14 Tx**	**EP13 Tx**	**EP12 Tx**	**EP11 Tx**	**EP10 Tx**	**EP9 Tx**	**EP8 Tx**
From CPU	r	r	r	r	r	r	r	r
From USB	set	set	set	set	set	set	set	set

	D7	D6	D5	D4	D3	D2	D1	D0
	EP7 Tx	**EP6 Tx**	**EP5 Tx**	**EP4 Tx**	**EP3 Tx**	**EP2 Tx**	**EP1 Tx**	**EP0**
From CPU	r	r	r	r	r	r	r	r
From USB	set	set	set	set	set	set	set	set

CONFIDENTIAL

Bit	Name	Function
D15	**EP15 TX**	TX Endpoint 15 interrupt.
D14	**EP14 TX**	TX Endpoint 14 interrupt.
D13	**EP13 TX**	TX Endpoint 13 interrupt.
D12	**EP12 TX**	TX Endpoint 12 interrupt.
D11	**EP11 TX**	TX Endpoint 11 interrupt.
D10	**EP10 TX**	TX Endpoint 10 interrupt.
D9	**EP9 TX**	TX Endpoint 9 interrupt.
D8	**EP8 TX**	TX Endpoint 8 interrupt.
D7	**EP7 TX**	TX Endpoint 7 interrupt.
D6	**EP6 TX**	TX Endpoint 6 interrupt.
D5	**EP5 TX**	TX Endpoint 5 interrupt.
D4	**EP4 TX**	TX Endpoint 4 interrupt.
D3	**EP3 TX**	TX Endpoint 3 interrupt.
D2	**EP2 TX**	TX Endpoint 2 interrupt.
D1	**EP1 TX**	TX Endpoint 1 interrupt.
D0	**EP0**	Endpoint 0 interrupt.

3.2.4. INTRRX

IntrRx is a 16-bit read-only register that indicates which of the interrupts for Rx Endpoints 1 – 15 are currently active. *Note:* Bits relating to endpoints that have not been configured will always return 0. Note also that all active interrupts are cleared when this register is read.

Address: 04h*; Reset value:* 16'h0000

	D15	D14	D13	D12	D11	D10	D9	D8
	EP15 Rx	**EP14 Rx**	**EP13 Rx**	**EP12 Rx**	**EP11 Rx**	**EP10 Rx**	**EP9 Rx**	**EP8 Rx**
From CPU	r	r	r	r	r	r	r	r
From USB	set	set	set	set	set	set	set	set

	D7	D6	D5	D4	D3	D2	D1	D0
	EP7 Rx	**EP6 Rx**	**EP5 Rx**	**EP4 Rx**	**EP3 Rx**	**EP2 Rx**	**EP1 Rx**	–
From CPU	r	r	r	r	r	r	r	r
From USB	set	set	set	set	set	set	set	r

CONFIDENTIAL

SOFT CORES

Bit	Name	Function
D15	**EP15 Rx**	Rx Endpoint 15 interrupt.
D14	**EP14 Rx**	Rx Endpoint 14 interrupt.
D13	**EP13 Rx**	Rx Endpoint 13 interrupt.
D12	**EP12 Rx**	Rx Endpoint 12 interrupt.
D11	**EP11 Rx**	Rx Endpoint 11 interrupt.
D10	**EP10 Rx**	Rx Endpoint 10 interrupt.
D9	**EP9 Rx**	Rx Endpoint 9 interrupt.
D8	**EP8 Rx**	Rx Endpoint 8 interrupt.
D7	**EP7 Rx**	Rx Endpoint 7 interrupt.
D6	**EP6 Rx**	Rx Endpoint 6 interrupt.
D5	**EP5 Rx**	Rx Endpoint 5 interrupt.
D4	**EP4 Rx**	Rx Endpoint 4 interrupt.
D3	**EP3 Rx**	Rx Endpoint 3 interrupt.
D2	**EP2 Rx**	Rx Endpoint 2 interrupt.
D1	**EP1 Rx**	Rx Endpoint 1 interrupt.
D0	–	*Unused, always returns 0*

3.2.5. INTRTXE

IntrTxE is a 16-bit register that provides interrupt enable bits for the interrupts in IntrTx. Where a bit is set to 1, MC_NINT will be asserted on the corresponding interrupt in the IntrTx register becoming set. Where a bit is set to 0, the interrupt in IntrTx is still set but MC_NINT is not asserted. On reset, the bits corresponding to Endpoint 0 and the TX endpoints included in the design are set to 1, while the remaining bits are set to 0. *Note:* Bits relating to endpoints that have not been configured will always return 0.

Address: 06h*; Reset value:* 16'hFFFF masked with the TX endpoints implemented

	D15	D14	D13	D12	D11	D10	D9	D8
	EP15 Tx	**EP14 Tx**	**EP13 Tx**	**EP12 Tx**	**EP11 Tx**	**EP10 Tx**	**EP9 Tx**	**EP8 Tx**
From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	r	r	r	r	r	r	r	r

	D7	D6	D5	D4	D3	D2	D1	D0
	EP7 Tx	**EP6 Tx**	**EP5 Tx**	**EP4 Tx**	**EP3 Tx**	**EP2 Tx**	**EP1 Tx**	**EP0**
From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	r	r	r	r	r	r	r	r

CONFIDENTIAL

3.2.6. INTRRXE

IntrRxE is a 16-bit register that provides interrupt enable bits for the interrupts in IntrRx. Where a bit is set to 1, MC_NINT will be asserted on the corresponding interrupt in the IntrRx register becoming set. Where a bit is set to 0, the interrupt in IntrRx is still set but MC_NINT is not asserted. On reset, the bits corresponding to the Rx endpoints included in the design are set to 1, while the remaining bits are set to 0. *Note:* Bits relating to endpoints that have not been configured will always return 0.

Address: 08h*; Reset value:* 16'hFFFE masked with the Rx endpoints implemented

	D15	D14	D13	D12	D11	D10	D9	D8
	EP15 Rx	**EP14 Rx**	**EP13 Rx**	**EP12 Rx**	**EP11 Rx**	**EP10 Rx**	**EP9 Rx**	**EP8 Rx**
From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	r	r	r	r	r	r	r	r

	D7	D6	D5	D4	D3	D2	D1	D0
	EP7 Rx	**EP6 Rx**	**EP5 Rx**	**EP4 Rx**	**EP3 Rx**	**EP2 Rx**	**EP1 Rx**	–
From CPU	rw	rw	rw	rw	rw	rw	rw	r
From USB	r	r	r	r	r	r	r	r

3.2.7. INTRUSB

IntrUSB is an 8-bit read-only register that indicates which USB interrupts are currently active. All active interrupts will be cleared when this register is read.

Address: 0Ah*; Reset value:* 8'h00

	D7	D6	D5	D4	D3	D2	D1	D0
	VBus Error	**Sess Req**	**Discon**	**Conn**	**SOF**	**Reset/Babble**	**Resume**	**Suspend**
From CPU	r	r	r	r	r	r	r	r
From USB	set	set	set	set	set	set	set	set

Bit	Name	Function
D7	**VBus Error**	Set when VBus drops below the VBus Valid threshold during a session. *Only valid when MUSBMHDRC is 'A' device.*
D6	**Sess Req**	Set when Session Request signaling has been detected. *Only valid when MUSBMHDRC is 'A' device.*
D5	**Discon**	Set in Host mode when a device disconnect is detected. Set in Peripheral mode when a session ends. *Valid at all transaction speeds.*
D4	**Conn**	Set when a device connection is detected. *Only valid in Host mode. Valid at all transaction speeds.*
D3	**SOF**	Set when a new frame starts.
D2	**Reset**	Set in Peripheral mode when Reset signaling is detected on the bus.
	Babble	Set in Host mode when babble is detected. *Note:* Only active after first SOF has been sent.
D1	**Resume**	Set when Resume signaling is detected on the bus while the MUSBMHDRC is in Suspend mode.
D0	**Suspend**	Set when Suspend signaling is detected on the bus. *Only valid in Peripheral mode.*

CONFIDENTIAL

SOFT CORES

3.2.8. INTRUSBE

IntrUSBE is an 8-bit register that provides interrupt enable bits for each of the interrupts in IntrUSB.

Address: 0Bh*; Reset value:* 8'h06

	D7	D6	D5	D4	D3	D2	D1	D0
	VBus Error	Sess Req	Discon	Conn	SOF	Reset/Babble	Resume	Suspend
From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	r	r	r	r	r	r	r	r

3.2.9. FRAME

Frame is a 16-bit read-only register that holds the last received frame number.

Address: 0Ch*; Reset value:* 16'h0000

	D15	...	D11	D10	...	D0
	0 0 0 0 0			(MSB)	Frame Number	(LSB)
From CPU	r	...	r	r	...	r
From USB	w	...	w	w	...	w

3.2.10. INDEX

Each TX endpoint and each Rx endpoint have their own set of control/status registers located between 100h – 1FFh. In addition one set of TX control/status and one set of Rx control/status registers appear at 10h – 19h. Index is a 4-bit register that determines which endpoint control/status registers are accessed.

Address: 0Eh; *Reset value:* 4'b0000

	D3	D2	D1	D0
	(MSB)	Selected Endpoint		(LSB)
From CPU	rw	rw	rw	rw
From USB	r	r	r	r

Before accessing an endpoint's control/status registers at 10h – 19h, the endpoint number should be written to the Index register to ensure that the correct control/status registers appear in the memory map.

3.2.11. TESTMODE

Testmode is an 8-bit register that is primarily used to put the MUSBMHDRC into one of the four test modes for High-speed operation described in the USB 2.0 specification – in response to a SET FEATURE: TESTMODE command. It is not used in normal operation.

Address: 0Fh*; Reset value:* 8'h00

	D7	D6	D5	D4	D3	D2	D1	D0
	Force_Host	FIFO_Access *(self-clearing)*	Force_FS	Force_HS	Test_Packet	Test_K	Test_J	Test_SE0_NAK
From CPU	rw	set	rw	rw	rw	rw	rw	rw
From USB	r	r	r	r	r	r	r	r

Bit	Name	Description

CONFIDENTIAL

Bit	Name	Description
D7	**Force_Host**	The CPU sets this bit to instruct the core to enter Host mode when the Session bit is set, regardless of whether it is connected to any peripheral. The state of the CID input, HostDisconnect and LineState signals are ignored. The core will then remain in Host mode until the Session bit is cleared, even if a device is disconnected, and if the Force_Host bit remains set, will re-enter Host mode the next time the Session bit is set. While in this mode, the status of the HOSTDISCON signal from the PHY may be read from bit 7 of the DevCtl register. The operating speed is determined from the Force_HS and Force_FS bits as follows: <table><tr><th>Force_HS</th><th>Force_FS</th><th>Operating Speed</th></tr><tr><td>0</td><td>0</td><td>Low Speed</td></tr><tr><td>0</td><td>1</td><td>Full Speed</td></tr><tr><td>1</td><td>0</td><td>High Speed</td></tr><tr><td>1</td><td>1</td><td>*Undefined*</td></tr></table>
D6	**FIFO_Access**	The CPU sets this bit to transfer the packet in the Endpoint 0 TX FIFO to the Endpoint 0 Rx FIFO. It is cleared automatically.
D5	**Force_FS**	The CPU sets this bit either in conjunction with bit 7 above or to force the MUSBMHDRC into Full-speed mode when it receives a USB reset.
D4	**Force_HS**	The CPU sets this bit either in conjunction with bit 7 above or to force the MUSBMHDRC into High-speed mode when it receives a USB reset.
D3	**Test_Packet**	*(High-speed mode)* The CPU sets this bit to enter the Test_Packet test mode. In this mode, the MUSBMHDRC repetitively transmits on the bus a 53-byte test packet, the form of which is defined in the *Universal Serial Bus Specification* Revision 2.0, Section 7.1.20 (and in section 28.4). *Note:* The test packet has a fixed format and must be loaded into the Endpoint 0 FIFO before the test mode is entered.
D2	**Test_K**	*(High-speed mode)* The CPU sets this bit to enter the Test_K test mode. In this mode, the MUSBMHDRC transmits a continuous K on the bus.
D1	**Test_J**	*(High-speed mode)* The CPU sets this bit to enter the Test_J test mode. In this mode, the MUSBMHDRC transmits a continuous J on the bus.
D0	**Test_SE0_NAK**	*(High-speed mode)* The CPU sets this bit to enter the Test_SE0_NAK test mode. In this mode, the MUSBMHDRC remains in High-speed mode but responds to any valid IN token with a NAK.

Note: Only one of Bits D0 – D6 should be set at any time.

3.2.12. DEVCTL

DevCtl is an 8-bit register that is used to select whether the MUSBMHDRC is operating in Peripheral mode or in Host mode, and for controlling and monitoring the USB VBus line. If the PHY is suspended no PHY clock (XCLK) is received and the VBus is not sampled.

Address: 60h; *Reset value:* 8'h80

	D7	D6	D5	D4	D3	D2	D1	D0
	B-Device	**FSDev**	**LSDev**	**VBus[1]**	**VBus[0]**	**Host Mode**	**Host Req**	**Session**
From CPU	r	r	r	r	r	r	rw	rw
From USB	rw	rw	rw	rw	rw	rw	r/clear	rw

Bit	Name	Function
D7	**B-Device**	This Read-only bit indicates whether the MUSBMHDRC is operating as the 'A' device or the 'B' device. $0 \Rightarrow$ 'A' device; $1 \Rightarrow$ 'B' device. *Only valid while a session is in progress.* *Note:* If the core is in Force_Host mode (i.e. a session has been started with Testmode.D7 = 1), this bit will indicate the state of the HOSTDISCON input signal from the PHY.
D6	**FSDev**	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.*
D5	**LSDev**	This Read-only bit is set when a low-speed device has been detected being connected to the port. *Only valid in Host mode.*
D4–D3	**VBus[1:0]**	These Read-only bits encode the current VBus level as follows: <table><tr><th>D4</th><th>D3</th><th>Meaning</th></tr><tr><td>0</td><td>0</td><td>Below SessionEnd</td></tr><tr><td>0</td><td>1</td><td>Above SessionEnd, below AValid</td></tr><tr><td>1</td><td>0</td><td>Above AValid, below VBusValid</td></tr><tr><td>1</td><td>1</td><td>Above VBusValid</td></tr></table>
D2	**Host Mode**	This Read-only bit is set when the MUSBMHDRC is acting as a Host.
D1	**Host Req**	When set, the MUSBMHDRC will initiate the Host Negotiation when Suspend mode is entered. It is cleared when Host Negotiation is completed. See Section 15. *('B' device only)*
D0	**Session**	*When operating as an 'A' device,* this bit is set or cleared by the CPU to start or end a session. *When operating as a 'B' device,* this bit is set/cleared by the MUSBMHDRC when a session starts/ends. It is also set by the CPU to initiate the Session Request Protocol. When the MUSBMHDRC is in Suspend mode, the bit may be cleared by the CPU to perform a software disconnect. *Note:* Clearing this bit when the core is not suspended will result in undefined behavior.

3.2.13.   MISC

The MISC Register is an 8-bit register that contain various common configuration bits. These bits include the Rx/TX Early DMA enable bits. The configuration bits occupy the register according to the table that follows below.

Address: 61h; *Reset value:* 8'h00

	D7	D6	D5	D4	D3	D2	D1	D0
	Unused	*Unused*	*Unused*	*Unused*	*Unused*	*Unused*	**tx_edma**	**rx_edma**
From CPU	R	r	r	r	r	r	rw	rw
From USB	r	r	r	r	r	r	rw	rw

Bit	Name	Function
D7-D2	*Unused*	These bits are reserved
D1	**tx_edma**	1'b0: DMA_REQ signal for all IN Endpoints will be de-asserted when MAXP bytes have been written to an endpoint. This is late mode. 1'b1: DMA_REQ signal for all IN Endpoints will be de-asserted when MAXP-8 bytes have been written to an endpoint. This is early mode.

CONFIDENTIAL

SOFT CORES

| D0 | **rx_edma** | 1'b0: DMA_REQ signal for all OUT Endpoints will be de-asserted when MAXP bytes have been read to an endpoint. This is late mode. |
| | | 1'b1: DMA_REQ signal for all OUT Endpoints will be de-asserted when MAXP-8 bytes have been read to an endpoint. This is early mode. |

3.3. INDEXED REGISTERS

Note: The action of the following registers when the selected endpoint has not been configured is undefined.

3.3.1. CSR0L

CSR0L is an 8-bit register that provides control and status bits for Endpoint 0. *Note:* The interpretation of the register depends on whether the MUSBMHDRC is acting as a peripheral or as a host. Users should also be aware that the value returned when the register is read reflects the status attained e.g. as a result of writing to the register.

CSR0L in Peripheral mode:

Address: 12h (with the Index register set to 0)*; Reset value:* 8'h00

	D7	D6	D5	D4	D3	D2	D1	D0
	Serviced SetupEnd *(self-clearing)*	**Serviced RxPktRdy** *(self-clearing)*	**SendStall** *(self-clearing)*	**SetupEnd**	**DataEnd** *(self-clearing)*	**SentStall**	**TxPktRdy** *(self-clearing)*	**RxPktRdy**
From CPU	set	set	set	r	set	r/clear	r/set	r
From USB	r	r	r	set	r	set	r	set

Bit	Name	Function in Peripheral mode
D7	**ServicedSetupEnd**	The CPU writes a 1 to this bit to clear the SetupEnd bit. It is cleared automatically.
D6	**ServicedRxPktRdy**	The CPU writes a 1 to this bit to clear the RxPktRdy bit. It is cleared automatically.
D5	**SendStall**	The CPU writes a 1 to this bit to terminate the current transaction. The STALL handshake will be transmitted and then this bit will be cleared automatically.
D4	**SetupEnd**	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 flushed at this time. The bit is cleared by the CPU writing a 1 to the ServicedSetupEnd bit.
D3	**DataEnd**	The CPU sets this bit: 1. When setting TxPktRdy for the last data packet. 2. When clearing RxPktRdy after unloading the last data packet. 3. When setting TxPktRdy for a zero length data packet. It is cleared automatically.
D2	**SentStall**	This bit is set when a STALL handshake is transmitted. The CPU should clear this bit.
D1	**TxPktRdy**	The CPU sets this bit after loading a data packet into the FIFO. It is cleared automatically when a data packet has been transmitted. An interrupt is also generated at this point (if enabled).
D0	**RxPktRdy**	This bit is set when a data packet has been received. An interrupt is generated when this bit is set. The CPU clears this bit by setting the ServicedRxPktRdy bit.

CONFIDENTIAL

SOFT CORES

CSR0L in Host mode: *Address:* 12h (with the Index register set to 0)*; Reset value:* 8'h00

	D7	D6	D5	D4	D3	D2	D1	D0
	NAK Timeout	**StatusPkt**	**ReqPkt**	**Error**	**SetupPkt**	**RxStall**	**TxPktRdy**	**RxPktRdy**
From CPU	r/clear	rw	rw	r/clear	r/clear	r/clear	r/set	r/clear
From USB	set	r	rw	set	r	set	clear	rw

Bit	Name	Function in Host mode
D7	**NAK Timeout**	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. The CPU should clear this bit to allow the endpoint to continue.
D6	**StatusPkt**	The CPU sets 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 to 1 so that a DATA1 packet is used for the Status Stage transaction.
D5	**ReqPkt**	The CPU sets this bit to request an IN transaction. It is cleared when RxPktRdy is set.
D4	**Error**	This bit will be set when three attempts have been made to perform a transaction with no response from the peripheral. The CPU should clear this bit. An interrupt is generated when this bit is set.
D3	**SetupPkt**	The CPU sets 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. *Note:* Setting this bit also clears the Data Toggle.
D2	**RxStall**	This bit is set when a STALL handshake is received. The CPU should clear this bit.
D1	**TxPktRdy**	The CPU sets this bit after loading a data packet into the FIFO. It is cleared automatically when a data packet has been transmitted. An interrupt is also generated at this point (if enabled).
D0	**RxPktRdy**	This bit is set when a data packet has been received. An interrupt is generated (if enabled) when this bit is set. The CPU should clear this bit when the packet has been read from the FIFO.

CONFIDENTIAL

3.3.2. CSR0H

CSR0H is an 8-bit register that provides control and status bits for Endpoint 0. *Note:* The interpretation of the register depends on whether the MUSBMHDRC is acting as a peripheral or as a host. Users should also be aware that the value returned when the register is read reflects the status attained e.g. as a result of writing to the register.

CSR0H in Peripheral mode:
Address: 13h (with the Index register set to 0)*; Reset value: 8'h00*

D7	D6	D5	D4	D3	D2	D1	D0
–	–	–	–	–	–	–	**FlushFIFO** *(self-clearing)*

From CPU	r	r	r	r	r	r	r	set
From USB	r	r	r	r	r	r	r	r

Bit	Name	Function in Peripheral mode
D7 – D1	–	*Unused. Return 0 when read.*
D0	**FlushFIFO**	The CPU writes a 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 (below) is cleared. *Note:* FlushFIFO should only be used when TxPktRdy/RxPktRdy is set. At other times, it may cause data to be corrupted.

CSR0H in Host mode:
Address: 13h (with the Index register set to 0)*; Reset value: 8'h00*

D7	D6	D5	D4	D3	D2	D1	D0
–	–	–	–	**Dis Ping**	**Data Toggle Wr. Enable** *(self-clearing)*	**Data Toggle**	**FlushFIFO** *(self-clearing)*

From CPU	r	r	r	r	rw	set	rw	set
From USB	r	r	r	r	r	r	rw	r

Bit	Name	Function in Host mode
D7 – D4	–	*Unused. Return 0 when read.*
D3	**Dis Ping**	The CPU writes a 1 to this bit to instruct the core not to issue PING tokens in data and status phases of a high-speed Control transfer (for use with devices that do not respond to PING).
D2	**Data Toggle Write Enable**	The CPU writes a 1 to this bit to enable the current state of the Endpoint 0 data toggle to be written (see Data Toggle bit, below). This bit is automatically cleared once the new value is written.
D1	**Data Toggle**	When read, this bit indicates the current state of the Endpoint 0 data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored.
D0	**FlushFIFO**	The CPU writes a 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 (below) is cleared. *Note:* FlushFIFO should only be used when TxPktRdy/RxPktRdy is set. At other times, it may cause data to be corrupted.

CONFIDENTIAL

3.3.3. COUNT0

Count0 is a 7-bit read-only register that 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 (CSR0.D0) is set.

Address: 18h (with the Index register set to 0)*; Reset value:*7'b0000000

	D6	D5	D4	D3	D2	D1	D0
	(MSB)			**Endpoint 0 Rx Count**			(LSB)
From CPU	r	r	r	r	r	r	r
From USB	w	w	w	w	w	w	w

3.3.4. TYPE0

NOTE: HOST MODE ONLY!

These bits only apply when the Multipoint option is enabled in the configuration GUI. When multipoint is enabled Type0 is an 8-bit register of which only bits 6 and 7 are implemented. These bits should be written with the operating speed of the targeted device.

Address: 1Ah*; Reset value:* 2'b00

	D7	D6
	Speed	
From CPU	rw	rw
From USB	r	r

Bit	Name	Function
D7– D6	**Speed**	Operating speed of the target device: 00: *Unused (Note: If selected, the target will be assumed to be using the same connection speed as the core.)* 01: High 10: Full 11: Low

CONFIDENTIAL

SOFT CORES

3.3.5. CONFIGDATA

ConfigData is an 8-bit Read-Only register that returns information about the selected core configuration.

Address: 1Fh (with the Index register set to 0)*; Reset value:* Configuration Dependent

D7	D6	D5	D4	D3	D2	D1	D0
MPRxE	MPTxE	BigEndian	HBRxE	HBTxE	DynFIFO Sizing	SoftConE	UTMI DataWidth
From CPU r	r	r	r	r	r	r	r
From USB r	r	r	r	r	r	r	r

Bit	Name	Function
D7	**MPRxE**	When set to '1', automatic amalgamation of bulk packets is selected (see Section 22)
D6	**MPTxE**	When set to '1', automatic splitting of bulk packets is selected (see Section 22)
D5	**BigEndian**	Always "0". Indicates Little Endian ordering.
D4	**HBRxE**	When set to '1' indicates High-bandwidth Rx ISO Endpoint Support selected.
D3	**HBTxE**	When set to '1' indicates High-bandwidth TX ISO Endpoint Support selected.
D2	**DynFIFO Sizing**	When set to '1' indicates Dynamic FIFO Sizing option selected.
D1	**SoftConE**	Always '1' . Indicates Soft Connect/Disconnect.
D0	**UTMI DataWidth**	Indicates selected UTMI+ data width. Always 0 indicating 8 bits.

3.3.6. NAKLIMIT0

NOTE: HOST MODE ONLY!

NAKLimit0 is a 5-bit register that sets the number of frames/microframes (High-Speed transfers) after which Endpoint 0 should timeout on receiving a stream of NAK responses. (Equivalent settings for other endpoints can be made through their TxInterval and RxInterval registers: see Sections 3.3.15 and 3.3.17.)

The number of frames/microframes selected is $2^{(m-1)}$ (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.

Address: 1Bh (with the Index register set to 0)*; Reset value:*5'b00000

D4	...	D0
(MSB)	**Endpoint 0 NAK Limit (m)**	(LSB)
From CPU rw	...	rw
From USB r	...	r

CONFIDENTIAL

SOFT CORES

3.3.7. TXMAXP

The TxMaxP register defines the maximum amount of data that can be transferred through the selected TX endpoint in a single operation. There is a TxMaxP register for each TX endpoint (except Endpoint 0).

Address: 10h*; Reset value:* 11/13/16'h0000

D12/15		D11	D10	...	D0
$m-1$			(MSB)	**Maximum Payload/transaction**	(LSB)

From CPU	rw	…	rw	rw	...	rw
From USB	r	…	r	r	...	r

Bits 10:0 define (in bytes) 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.

Where the option of High-bandwidth Isochronous/Interrupt endpoints or of packet splitting on Bulk endpoints has been taken when the core is configured, the register includes either 2 or 5 further bits that define a multiplier m which is equal to one more than the value recorded.

In the case of Bulk endpoints with the packet splitting option enabled, the multiplier m can be up to 32 and defines the maximum number of 'USB' packets (i.e. packets for transmission over the USB) of the specified payload into which a single data packet placed in the FIFO should be split, prior to transfer. (If the packet splitting option is not enabled, D15–D13 is not implemented and D12–D11(if included) is ignored.) **Note:** The data packet is required to be an exact multiple of the payload specified by bits 10:0, which is itself required to be either 8, 16, 32, 64 or (in the case of High Speed transfers) 512 bytes.

For Isochronous/Interrupts endpoints operating in High-Speed mode and with the High-bandwidth option enabled, m may only be either 2 or 3 (corresponding to bit 11 set or bit 12 set, respectively) and it specifies the maximum number of such transactions that can take place in a single microframe. If either bit 11 or bit 12 is non-zero, the MUSBMHDRC will automatically split any data packet written to the FIFO into up to 2 or 3 'USB' packets, each containing the specified payload (or less). The maximum payload for each transaction is 1024 bytes, so this allows up to 3072 bytes to be transmitted in each microframe. (For Isochronous transfers in Full-speed mode or if High-bandwidth is not enabled, bits 11 and 12 are ignored.)

The value written to bits 10:0 (multiplied by m in the case of high-bandwidth Isochronous transfers) must match the value given in the *wMaxPacketSize* field of the Standard Endpoint Descriptor for the associated endpoint (see *USB Specification* Revision 2.0, Chapter 9). A mismatch could cause unexpected results.

The total amount of data represented by the value written to this register (specified payload × m) must not exceed the FIFO size for the TX endpoint, and should not exceed half the FIFO size if double-buffering is required.

If this register is changed after packets have been sent from the endpoint, the TX endpoint FIFO should be completely flushed (using the FlushFIFO bit in TxCSRL) after writing the new value to this register.

 Note: TxMaxP must be set to an even number of bytes for proper interrupt generation in DMA Mode 1.

CONFIDENTIAL

3.3.8. TXCSRL

TxCSRL is an 8-bit register that provides control and status bits for transfers through the currently-selected TX endpoint. There is a TxCSRL register for each configured TX endpoint (not including Endpoint 0). *Note:* The interpretation of the register depends on whether the MUSBMHDRC is acting as a peripheral or as a host. Users should also be aware that the value returned when the register is read reflects the status attained e.g. as a result of writing to the register.

In Peripheral mode:
Address: 12h*; Reset value:* 8'h00

	D7	D6	D5	D4	D3	D2	D1	D0
	IncompTx	**ClrDataTog**	**SentStall**	**SendStall**	**FlushFIFO** *(self-clearing)*	**UnderRun**	**FIFO NotEmpty**	**TxPktRdy**
From CPU	r/clear	set	r/clear	rw	set	r/clear	r/clear	r/set
From USB	set	r/clear	set	r	r	set	set	Clear

Bit	Name	Function in Peripheral mode
D7	**IncompTx**	When the endpoint is being used for high-bandwidth Isochronous, this bit is set to indicate where a large packet has been split into 2 or 3 packets for transmission but insufficient IN tokens have been received to send all the parts. *Note: In anything other than isochronous transfers, this bit will always return 0.*
D6	**ClrDataTog**	The CPU writes a 1 to this bit to reset the endpoint data toggle to 0.
D5	**SentStall**	This bit is set when a STALL handshake is transmitted. The FIFO is flushed and the TxPktRdy bit is cleared (see below). The CPU should clear this bit.
D4	**SendStall**	The CPU writes a 1 to this bit to issue a STALL handshake to an IN token. The CPU clears this bit to terminate the stall condition. *Note: This bit has no effect where the endpoint is being used for Isochronous transfers.*
D3	**FlushFIFO**	The CPU writes a 1 to this bit to flush the latest packet from the endpoint TX FIFO. The FIFO pointer is reset, the TxPktRdy bit (below) is cleared and an interrupt is generated. May be set simultaneously with TxPktRdy to abort the packet that is currently being loaded into the FIFO. *Note:* FlushFIFO should only be used when TxPktRdy is set. At other times, it may cause data to be corrupted. Also note that, if the FIFO is double-buffered, FlushFIFO may need to be set twice to completely clear the FIFO.
D2	**UnderRun**	The USB sets this bit if an IN token is received when TxPktRdy is not set. The CPU should clear this bit.
D1	**FIFONotEmpty**	The USB sets this bit when there is at least 1 packet in the TX FIFO.
D0	**TxPktRdy**	The CPU sets this bit after loading a data packet into the FIFO. It is cleared automatically when a data packet has been transmitted. An interrupt is also generated at this point (if enabled). TxPktRdy is also automatically cleared prior to loading a second packet into a double-buffered FIFO.

CONFIDENTIAL

SOFT CORES

In Host mode:

	D7	D6	D5	D4	D3	D2	D1	D0
	NAK Timeout / IncompTx	ClrDataTog	RxStall	SetupPkt	FlushFIFO *(self-clearing)*	Error	FIFO NotEmpty	TxPktRdy
From CPU	r/clear	set	r/clear	rw	set	r/clear	r/clear	r/set
From USB	set	r/clear	set	r	r	rw	set	Clear

Bit	Name	Function in Host mode
D7	**NAK Timeout** **IncompTx**	*Bulk endpoints only:* 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 NAK Limit by the TxInterval register. The CPU should clear this bit to allow the endpoint to continue. *High-bandwidth Interrupt endpoints only:* This bit will be set if no response is received from the device to which the packet is being sent.
D6	**ClrDataTog**	The CPU writes a 1 to this bit to reset the endpoint data toggle to 0.
D5	**RxStall**	This bit is set when a STALL handshake is received. When this bit is set, any DMA request that is in progress is stopped, the FIFO is completely flushed and the TxPktRdy bit is cleared (see below). The CPU should clear this bit.
D4	**SetupPkt**	The CPU sets 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. *Note:* Setting this bit also clears the Data Toggle.
D3	**FlushFIFO**	The CPU writes a 1 to this bit to flush the latest packet from the endpoint TX FIFO. The FIFO pointer is reset, the TxPktRdy bit (below) is cleared and an interrupt is generated. May be set simultaneously with TxPktRdy to abort the packet that is currently being loaded into the FIFO. *Note:* FlushFIFO should only be used when TxPktRdy is set. At other times, it may cause data to be corrupted. Also note that, if the FIFO is double-buffered, FlushFIFO may need to be set twice to completely clear the FIFO.
D2	**Error**	The USB sets this bit when 3 attempts have been made to send a packet and no handshake packet has been received. When the bit is set, an interrupt is generated, TxPktRdy is cleared and the FIFO is completely flushed. The CPU should clear this bit. *Valid only when the endpoint is operating in Bulk or Interrupt mode.*
D1	**FIFONotEmpty**	The USB sets this bit when there is at least 1 packet in the TX FIFO.
D0	**TxPktRdy**	The CPU sets this bit after loading a data packet into the FIFO. It is cleared automatically when a data packet has been transmitted. An interrupt is also generated at this point (if enabled). TxPktRdy is also automatically cleared prior to loading a second packet into a double-buffered FIFO.

CONFIDENTIAL

3.3.9. TXCSRH

TxCSRH is an 8-bit register that provides additional control for transfers through the currently-selected TX endpoint. There is a TxCSRH register for each configured TX endpoint (not including Endpoint 0). *Note:* The interpretation of the register depends on whether the MUSBMHDRC is acting as a peripheral or as a host. Users should also be aware that the value returned when the register is read reflects the status attained e.g. as a result of writing to the register.

In Peripheral mode:

Address: 13h*; Reset value: 8'h00*

	D7	D6	D6	D4	D3	D2	D1	D0
	AutoSet	**ISO**	**Mode**	**DMAReqEnab**	**FrcDataTog**	**DMAReqMode**	**–**	**–**
From CPU	rw	rw	rw	rw	rw	rw	r	r
From USB	r	r	r	r	r	r	r	r

Bit	Name	Function in Peripheral mode
D7	**AutoSet**	If the CPU sets this bit, TxPktRdy 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 TxPktRdy will have to be set manually. *Note: Should not be set for either high-bandwidth Isochronous endpoints or high-bandwidth Interrupt endpoints.*
D6	**ISO**	The CPU sets this bit to enable the TX endpoint for Isochronous transfers, and clears it to enable the TX endpoint for Bulk or Interrupt transfers. *Note: This bit only has any effect in Peripheral mode. In Host mode, it always returns zero.*
D5	**Mode**	The CPU sets this bit to enable the endpoint direction as TX, and clears the bit to enable it as Rx. *Note: This bit only has any effect where the same endpoint FIFO is used for both TX and Rx transactions.*
D4	**DMAReqEnab**	The CPU sets this bit to enable the DMA request for the TX endpoint.
D3	**FrcDataTog**	The CPU sets 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.
D2	**DMAReqMode**	The CPU sets this bit to select DMA Request Mode 1 and clears it to select DMA Request Mode 0. *Note: This bit must not be cleared either before or in the same cycle as the above DMAReqEnab bit is cleared.*
D1 – D0	**–**	*Unused, always return 0.*

CONFIDENTIAL

In Host mode:

	D7	D6	D6	D4	D3	D2	D1	D0
	AutoSet	–	Mode	DMAReqEnab	FrcDataTog	DMAReqMode	Data Toggle Wr.Enable *(self-clearing)*	Data Toggle
From CPU	rw	r	rw	rw	rw	rw	set	rw
From USB	r	r	r	r	r	r	r	rw

Bit	Name	Function in Host mode
D7	AutoSet	If the CPU sets this bit, TxPktRdy 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 TxPktRdy will have to be set manually. *Note: Should not be set for either high-bandwidth Isochronous endpoints or high-bandwidth Interrupt endpoints.*
D6	–	*Unused, always returns zero.*
D5	Mode	The CPU sets this bit to enable the endpoint direction as TX, and clears it to enable the endpoint direction as Rx. *Note: This bit only has any effect where the same endpoint FIFO is used for both TX and Rx transactions.*
D4	DMAReqEnab	The CPU sets this bit to enable the DMA request for the TX endpoint.
D3	FrcDataTog	The CPU sets 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.
D2	DMAReqMode	The CPU sets this bit to select DMA Request Mode 1 and clears it to select DMA Request Mode 0. *Note: This bit must not be cleared either before or in the same cycle as the above DMAReqEnab bit is cleared.*
D1	Data Toggle Write Enable	The CPU writes a 1 to this bit to enable the current state of the TX Endpoint data toggle to be written (see Data Toggle bit, below). This bit is automatically cleared once the new value is written.
D0	Data Toggle	When read, this bit indicates the current state of the TX Endpoint data toggle. If D1 is high, this bit may be written with the required setting of the data toggle. If D1 is low, any value written to this bit is ignored.

CONFIDENTIAL

3.3.10. RXMAXP

The RxMaxP register defines the maximum amount of data that can be transferred through the selected Rx endpoint in a single operation. There is a RxMaxP register for each Rx endpoint (except Endpoint 0).

Address: 14h*; Reset value:* 11/13/16'h0000

D12/15		D11	D10	...	D0
m − 1			(MSB)	**Maximum Payload/transaction**	(LSB)
From CPU rw	…	rw	rw	...	rw
From USB r	…	r	r	...	r

Bits 10:0 define (in bytes) 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.

Where the option of High-bandwidth Isochronous/Interrupt endpoints or of packet splitting on Bulk endpoints has been taken when the core is configured, the register includes either 2 or 5 further bits that define a multiplier *m* which is equal to one more than the value recorded.

For Bulk endpoints with the packet splitting option enabled, the multiplier *m* can be up to 32 and defines the number of USB packets of the specified payload which are to be amalgamated into a single data packet within the FIFO. (If the packet splitting option is not enabled, D15–D13 is not implemented and D12–D11 (if included) is ignored.)

For Isochronous/Interrupt endpoints operating in High-Speed mode and with the High-bandwidth option enabled, *m* may only be either 2 or 3 (corresponding to bit 11 set or bit 12 set, respectively) and it specifies the maximum number of such transactions that can take place in a single microframe. If either bit 11 or bit 12 is non-zero, the MUSBMHDRC will automatically combine the separate USB packets received in any microframe into a single packet within the RX FIFO. The maximum payload for each transaction is 1024 bytes, so this allows up to 3072 bytes to be received in each microframe. (For Isochronous transfers in Full-speed mode or if High-bandwidth is not enabled, bits 11 and 12 are ignored.)

The value written to bits 10:0 (multiplied by *m* in the case of high-bandwidth Isochronous transfers) must match the value given in the *wMaxPacketSize* field of the Standard Endpoint Descriptor for the associated endpoint (see *USB Specification* Revision 2.0, Chapter 9). A mismatch could cause unexpected results.

The total amount of data represented by the value written to this register (specified payload × *m*) must not exceed the FIFO size for the RX endpoint, and should not exceed half the FIFO size if double-buffering is required.

Note: RxMaxP must be set to an even number of bytes for proper interrupt generation in DMA Mode 1.

CONFIDENTIAL

3.3.11. RXCSRL

RxCSRL is an 8-bit register that provides control and status bits for transfers through the currently-selected Rx endpoint. There is a RxCSRL register for each configured Rx endpoint (not including Endpoint 0). *Note:* The interpretation of the register depends on whether the MUSBMHDRC is acting as a peripheral or as a host. Users should also be aware that the value returned when the register is read reflects the status attained e.g. as a result of writing to the register.

In Peripheral mode:

Address: 16h*; Reset value: 8'h00*

	D7	D6	D5	D4	D3	D2	D1	D0
	ClrDataTog	**SentStall**	**SendStall**	**FlushFIFO** *(self-clearing)*	**DataError**	**OverRun**	**FIFOFull** *(self-clearing)*	**RxPktRdy**
From CPU	set	r/clear	rw	set	r	r/clear	r	r/clear
From USB	r/clear	set	r	r	set	set	set	set

Bit	Name	Function in Peripheral mode
D7	**ClrDataTog**	The CPU writes a 1 to this bit to reset the endpoint data toggle to 0.
D6	**SentStall**	This bit is set when a STALL handshake is transmitted. The CPU should clear this bit.
D5	**SendStall**	The CPU writes a 1 to this bit to issue a STALL handshake. The CPU clears this bit to terminate the stall condition. *Note: This bit has no effect where the endpoint is being used for Isochronous transfers.*
D4	**FlushFIFO**	The CPU writes a 1 to this bit to flush the next packet to be read from the endpoint Rx FIFO. The FIFO pointer is reset and the RxPktRdy bit (below) is cleared. *Note:* FlushFIFO should only be used when RxPktRdy is set. At other times, it may cause data to be corrupted. Also note that, if the FIFO is double-buffered, FlushFIFO may need to be set twice to completely clear the FIFO.
D3	**DataError**	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.*
D2	**OverRun**	This bit is set if an OUT packet cannot be loaded into the Rx FIFO. The CPU 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.*
D1	**FIFOFull**	This bit is set when no more packets can be loaded into the Rx FIFO.
D0	**RxPktRdy**	This bit is set when a data packet has been received. The CPU should clear this bit when the packet has been unloaded from the Rx FIFO. An interrupt is generated when the bit is set.

CONFIDENTIAL

In Host mode:

Address: 16h*; Reset value: 8'h00*

	D7	D6	D5	D4	D3	D2	D1	D0
	ClrDataTog	RxStall	ReqPkt	FlushFIFO *(self-clearing)*	DataError/ NAK Timeout	Error	FIFOFull *(self-clearing)*	RxPktRdy
From CPU	set	r/clear	rw	set	r (/clear)	r/clear	r	r/clear
From USB	r/clear	set	rw	r	set	set	set	set

Bit	Name	Function in Host mode
D7	**ClrDataTog**	The CPU writes a 1 to this bit to reset the endpoint data toggle to 0.
D6	**RxStall**	When a STALL handshake is received, this bit is set and an interrupt is generated. The CPU should clear this bit.
D5	**ReqPkt**	The CPU writes a 1 to this bit to request an IN transaction. It is cleared when RxPktRdy is set.
D4	**FlushFIFO**	The CPU writes a 1 to this bit to flush the next packet to be read from the endpoint Rx FIFO. The FIFO pointer is reset and the RxPktRdy bit (below) is cleared. *Note:* FlushFIFO should only be used when RxPktRdy is set. At other times, it may cause data to be corrupted. Also note that, if the FIFO is double-buffered, FlushFIFO may need to be set twice to completely clear the FIFO.
D3	**DataError/ NAK Timeout**	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 Rx endpoint is halted following the receipt of NAK responses for longer than the time set as the NAK Limit by the RxInterval register. The CPU should clear this bit to allow the endpoint to continue.
D2	**Error**	The USB sets this bit when 3 attempts have been made to receive a packet and no data packet has been received. The CPU should clear this bit. An interrupt is generated when the bit is set. *Note: This bit is only valid when the Rx endpoint is operating in Bulk or Interrupt mode. In ISO mode, it always returns zero.*
D1	**FIFOFull**	This bit is set when no more packets can be loaded into the Rx FIFO.
D0	**RxPktRdy**	This bit is set when a data packet has been received. The CPU should clear this bit when the packet has been unloaded from the Rx FIFO. An interrupt is generated when the bit is set.

CONFIDENTIAL

3.3.12. RXCSRH

RxCSRH is an 8-bit register that provides additional control and status bits for transfers through the currently-selected Rx endpoint. There is a RxCSRH register for each configured Rx endpoint (not including Endpoint 0). *Note:* The interpretation of the register depends on whether the MUSBMHDRC is acting as a peripheral or as a host. Users should also be aware that the value returned when the register is read reflects the status attained e.g. as a result of writing to the register.

In Peripheral mode:

Address: 17h*; Reset value:* 8'h00

	D7	D6	D5	D4	D3	D2	D1	D0
	AutoClear	**ISO**	**DMAReqEnab**	**DisNyet /PID Error**	**DMAReqMode**	–	–	**IncompRx**
From CPU	rw	rw	rw	rw/r	rw	r	r	r/clear
From USB	r	r	r	r/rw	r	r	r	set

Bit	Name	Function in Peripheral mode
D7	**AutoClear**	If the CPU sets this bit then the RxPktRdy bit will be automatically cleared when a packet of RxMaxP bytes has been unloaded from the Rx FIFO. When packets of less than the maximum packet size are unloaded, RxPktRdy will have to be cleared manually. When using a DMA to unload the RxFIFO, data is read from the RxFIFO in 4 byte chunks regardless of the RxMaxP. Therefore, the RxPktRdy bit will be cleared as follows: <table><tr><td>**Remainder (RxMaxP/4)**</td><td>**Actual Bytes Read**</td><td>**Packet Sizes that will clear RxPktRdy.**</td></tr><tr><td>0 (i.e. RXMaxP = 64 bytes)</td><td>RXMAXP</td><td>RXMAXP, RXMAXP-1, RXMAXP-2, RXMAXP-3</td></tr><tr><td>3 (i.e. RXMaxP = 63 bytes)</td><td>RXMAXP+1</td><td>RXMAXP, RXMAXP-1, RXMAXP-2</td></tr><tr><td>2 (i.e. RXMaxP = 62 bytes)</td><td>RXMAXP+2</td><td>RXMAXP, RXMAXP-1</td></tr><tr><td>1 (i.e. RXMaxP = 61 bytes)</td><td>RXMAXP+3</td><td>RXMAXP</td></tr></table> *Note: Should not be set for high-bandwidth Isochronous endpoints.*
D6	**ISO**	The CPU sets this bit to enable the Rx endpoint for Isochronous transfers, and clears it to enable the Rx endpoint for Bulk/Interrupt transfers.
D5	**DMAReqEnab**	The CPU sets this bit to enable the DMA request for the Rx endpoint.
D4	**DisNyet**	**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.*
	PID Error	**ISO Transactions:** The core sets this bit to indicate a PID error in the received packet.
D3	**DMAReqMode**	The CPU sets this bit to select DMA Request Mode 1 and clears it to select DMA Request Mode 0.
D2–D1	–	*Unused, always return 0.*
D0	**IncompRx**	This bit is set in a high-bandwidth Isochronous/Interrupt transfer if the packet in the Rx FIFO is incomplete because parts of the data were not received. It is cleared when RxPktRdy is cleared. *Note: In anything other than Isochronous transfer, this bit will always return 0.*

CONFIDENTIAL

In Host mode: *Address:* 17h*; Reset value:* 8'h00

	D7	D6	D5	D4	D3	D2	D1	D0
	AutoClear	AutoReq	DMAReqEnab	PID Error	DMAReqMode	Data Toggle Wr. Enable *(self-clearing)*	Data Toggle	IncompRx
From CPU	rw	rw	rw	r	rw	r	r	r/clear
From USB	r	r	r	rw	r	r	r	set

Bit	Name	Function in Host mode
D7	**AutoClear**	If the CPU sets this bit then the RxPktRdy bit will be automatically cleared when a packet of RxMaxP bytes has been unloaded from the Rx FIFO. When packets of less than the maximum packet size are unloaded, RxPktRdy will have to be cleared manually. When using a DMA to unload the RxFIFO, data is read from the RxFIFO in 4 byte chunks regardless of the RxMaxP. Therefore, the RxPktRdy bit will be cleared as follows: <table><tr><td>**Remainder (RxMaxP/4)**</td><td>**Actual Bytes Read**</td><td>**Packet Sizes that will clear RxPktRdy.**</td></tr><tr><td>0 (i.e. RXMaxP = 64 bytes)</td><td>RXMAXP</td><td>RXMAXP, RXMAXP-1, RXMAXP-2, RXMAXP-3</td></tr><tr><td>3 (i.e. RXMaxP = 63 bytes)</td><td>RXMAXP+1</td><td>RXMAXP, RXMAXP-1, RXMAXP-2</td></tr><tr><td>2 (i.e. RXMaxP = 62 bytes)</td><td>RXMAXP+2</td><td>RXMAXP, RXMAXP-1</td></tr><tr><td>1 (i.e. RXMaxP = 61 bytes)</td><td>RXMAXP+3</td><td>RXMAXP</td></tr></table> *Note: Should not be set for high-bandwidth Isochronous endpoints.*
D6	**AutoReq**	If the CPU sets this bit, 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.
D5	**DMAReqEnab**	The CPU sets this bit to enable the DMA request for the Rx endpoint.
D4	**PID Error**	**ISO Transactions Only:** The core sets this bit to indicate a PID error in the received packet. ***Bulk/Interrupt Transactions:*** *The setting of this bit is ignored.*
D3	**DMAReqMode**	The CPU sets this bit to select DMA Request Mode 1 and clears it to select DMA Request Mode 0.
D2	**Data Toggle Write Enable**	The CPU writes a 1 to this bit to enable the current state of the Endpoint 0 data toggle to be written (see Data Toggle bit, below). This bit is automatically cleared once the new value is written.
D1	**Data Toggle**	When read, this bit indicates the current state of the Endpoint 0 data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored.
D0	**IncompRx**	This bit will be set in a high-bandwidth Isochronous or Interrupt transfer if the packet received is incomplete. It will be cleared when RxPktRdy is cleared. *Note: If USB protocols are followed correctly, this bit should never be set. The bit becoming set indicates a failure of the associated Peripheral device to behave correctly. (In anything other than Isochronous transfer, this bit will always return 0.)*

CONFIDENTIAL

3.3.13. RXCOUNT

RxCount is a 13-bit read-only register that holds the number of data bytes in the packet currently in line to be read from the Rx FIFO. If the packet was transmitted as multiple bulk packets, the number given will be for the combined packet.
Note: The value returned changes as the FIFO is unloaded and is only valid while RxPktRdy (RxCSR.D0) is set.

Address: 18h*; Reset value:* 13'b0000000000000

D12	...	D0
(MSB)	**Endpoint Rx Count**	(LSB)

From CPU	r	...	r
From USB	w	...	w

3.3.14. TXTYPE

NOTE: HOST MODE ONLY!

TxType is an 8-bit register that should be written with the endpoint number to be targeted by the endpoint, the transaction protocol to use for the currently-selected TX endpoint, and its operating speed. There is a TxType register for each configured TX endpoint (except Endpoint 0, which has its own Type register at 1Ah). D6-D7 are only valid when the core is configured with the Multipoint option.

Address: 1Ah*; Reset value:* 8'h00

D7	D6	D5	D4	D3	D2	D1	D0
***Speed**		**Protocol**		**Target Endpoint Number**			

	D7	D6	D5	D4	D3	D2	D1	D0
From CPU	Rw	rw	rw	rw	rw	rw	rw	rw
From USB	R	r	r	r	r	r	r	r

Note: D6-D7 are only valid when the multipoint option has been enabled, otherwise these bits are not used.

Bit	Name	Function
D7– D6	**Speed**	Operating speed of the target device when the core is configured with the multipoint option: 00: *Unused (Note: If selected, the target will be assumed to be using the same connection speed as the core.)* 01: High 10: Full 11: Low When the core is not configured with the multipoint option these bits should not be accessed.
D5– D4	**Protocol**	The CPU should set this to select the required protocol for the TX endpoint: 00: Control 01: Isochronous 10: Bulk 11: Interrupt
D3 – D0	**Target Endpoint Number**	The CPU should set this value to the endpoint number contained in the TX endpoint descriptor returned to the MUSBMHDRC during device enumeration.

CONFIDENTIAL

3.3.15. TXINTERVAL

Address: 1Bh*; Reset value:* 8'b00000000

D7	...	D0

Tx Polling Interval/NAK Limit (*m*)

From CPU	rw	...	Rw
From USB	r	...	R

NOTE: HOST MODE ONLY!

TxInterval is an 8-bit register that, for Interrupt and Isochronous transfers, defines the polling interval for the currently-selected TX endpoint. For Bulk endpoints, this register sets the number of frames/microframes after which the endpoint should timeout on receiving a stream of NAK responses.

There is a TxInterval register for each configured TX 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.

3.3.16. RXTYPE

NOTE: HOST MODE ONLY!

RxType is an 8-bit register that should be written with the endpoint number to be targeted by the endpoint, the transaction protocol to use for the currently-selected Rx endpoint, and its operating speed. There is a RxType register for each configured Rx endpoint (except Endpoint 0, which has its own Type register at 1Ah).). D6-D7 are only valid when the core is configured with the Multipoint option.

Address: 1Ch*; Reset value:* 8'h00

D7	D6	D5	D4	D3	D2	D1	D0
*Speed		Protocol		Target Endpoint Number			

From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	r	r	r	r	r	r	r	r

CONFIDENTIAL

Bit	Name	Function
D7– D6	**Speed**	Operating speed of the target device when the core is configured with the multipoint option: 00: *Unused (Note: If selected, the target will be assumed to be using the same connection speed as the core.)* 01: High 10: Full 11: Low When the core is not configured with the multipoint option these bits should not be accessed.
D5– D4	**Protocol**	The CPU should set this to select the required protocol for the Rx endpoint: 00: Control 01: Isochronous 10: Bulk 11: Interrupt
D3 – D0	**Target Endpoint Number**	The CPU should set this value to the endpoint number contained in the Rx endpoint descriptor returned to the MUSBMHDRC during device enumeration.

Note: D6-D7 are only valid when the multipoint option has been enabled, otherwise these bits are not used.

3.3.17. RXINTERVAL

NOTE: HOST MODE ONLY!

RxInterval is an 8-bit register that, for Interrupt and Isochronous transfers, defines the polling interval for the currently-selected Rx endpoint. For Bulk endpoints, this register sets the number of frames/microframes after which the endpoint should timeout on receiving a stream of NAK responses. There is a RxInterval register for each configured Rx endpoint (except Endpoint 0). In each case the value that is set defines a number of frames/microframes (High Speed transfers), as follows:

Address: 1Dh*; Reset value:* 8'b00000000

D7	...	D0
Rx Polling Interval/NAK Limit (*m*)		

From CPU rw	...	rw
From USB r	...	r

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.

3.3.18. FIFOSIZE

FIFOSize is an 8-bit Read-Only register that returns the sizes of the FIFOs associated with the selected additional TX/Rx endpoints. The lower nibble encodes the size of the selected TX endpoint FIFO; the upper nibble encodes the size of the selected Rx endpoint FIFO. Values of 3 – 13 correspond to a FIFO size of 2^n bytes (8 – 8192 bytes). If an endpoint has not been configured, a value of 0 will be displayed. Where the TX and Rx endpoints share the same FIFO, the Rx FIFO size will be encoded as 0xF.

Note: The register only has this interpretation when the Index register is set to select one of Endpoints 1 – 15 and Dynamic

CONFIDENTIAL

Sizing is <u>not</u> selected. It has a special interpretation when the Index register is set to select Endpoint 0 (see Section 3.3.4), while the result returned is not valid where Dynamic FIFO sizing is used.

Address: 1Fh (Index register set to select Endpoints 1 – 15 only)*; Reset value:* Configuration Dependent

D7	...	D4	D3	…	D0
Rx FIFO Size			**Tx FIFO Size**		
From CPU r	...	r	r	…	r
From USB r	...	r	r	…	r

3.4. FIFOx (Addresses 20h – 5Fh)

This address range provides 16 addresses for CPU access to the FIFOs for each endpoint. Writing to these addresses loads data into the TxFIFO for the corresponding endpoint. Reading from these addresses unloads data from the RxFIFO for the corresponding endpoint.

The address range is 20h – 5Fh and the FIFOs are located on 32-bit double-word boundaries (Endpoint 0 at 20h, Endpoint 1 at 24h ... Endpoint 15 at 5Ch).

Note: (i) Transfers to and from FIFOs may be 8-bit, 16-bit or 32-bit as required, and any combination of access is allowed provided the data accessed is contiguous. However, all the transfers associated with one packet must be of the same width so that the data is consistently byte-, word- or double-word-aligned. The last transfer may however contain fewer bytes than the previous transfers in order to complete an odd-byte or odd-word transfer.
(ii) Depending on the size of the FIFO and the expected maximum packet size, the FIFOs support either single-packet or double-packet buffering. However, burst writing of multiple packets is not supported as flags need to be set after each packet is written.
(iii) Following a STALL response or a TX Strike Out error on Endpoint 1 – 15, the associated FIFO is completely flushed.

3.5. ADDITIONAL MULTIPOINT CONTROL/STATUS REGISTERS

The following subsections detail additional control and status registers that are only valid when the multipoint option is enabled in the configuration GUI. If the multipoint option is not enabled these reregisters should not be accessed.

3.5.1. TXFUNCADDR/RXFUNCADDR

NOTE: REQUIRED IN HOST MODE!

TxFuncAddr and RxFuncAddr are 7-bit read/write registers that record the address of the target function that is to be accessed through the associated endpoint (EP*n*). TxFuncAddr needs to be defined for each TX endpoint that is used; RxFuncAddr needs to be defined for each Rx endpoint that is used.

Note: TxFuncAddr must be defined for Endpoint 0. The RxFuncAddr register does not exist on EP0.

Address: 80+8**n*h and 84+8**n*h respectively*; Reset value* 7'h00

D6	...	D0
Address of Target Function		
From CPU rw	...	rw
From USB r	...	r

CONFIDENTIAL

3.5.2. TXHUBADDR/RXHUBADDR

NOTE: RELEVANT IN HOST MODE ONLY!

TxHubAddr and RxHubAddr are 8-bit read/write registers which, like TxHubPort and RxHubPort, only need to be written where a full- or low-speed device is connected to TX/Rx Endpoint EP*n* via a high-speed USB 2.0 hub which carries out the necessary transaction translation to convert between high-speed transmission and full-/low-speed transmission. In such circumstances:

- the lower 7 bits should record the address of this USB 2.0 hub

- the top bit should record whether the hub has multiple transaction translators (set to '0' if single transaction translator; set to '1' if multiple transaction translators).

Note: If Endpoint 0 is connected to a hub, then TxHubAddr must be defined for EP0. The RxHubAddr register does not exist on EP0.

Address: 82+8*n h and 86+8*n h respectively*; Reset value:* 8'h00

	D7	D6	…	D0
	Multiple Translators	**Hub Address**		
From CPU	rw	rw	…	rw
From USB	r	r	…	r

3.5.3. TXHUBPORT/RXHUBPORT

NOTE: RELEVANT IN HOST MODE ONLY!

TxHubPort and RxHubPort only need to be written where a full- or low-speed device is connected to TX/Rx Endpoint EP*n* via a high-speed USB 2.0 hub which carries out the necessary transaction translation. In such circumstances, these 7-bit read/write registers need to be used to record the port of that USB 2.0 hub through which the target associated with the endpoint is accessed.

Note: If Endpoint 0 is connected to a hub, then TxHubPort must be defined. The RxHubPort register does not exist on EP0.

Address: 83+8*n h and 87+8*n h respectively*; Reset value:* 7'h00

	D6	...	D0
	Hub Port		
From CPU	rw	...	rw
From USB	r	...	r

This information, together with the Hub Address details recorded above, allows the MUSBMHDRC to support split transactions.

3.6. ADDITIONAL CONTROL/STATUS REGISTERS

3.6.1. VCONTROL

NOTE: WRITE ONLY!

VControl is a UTMI+ PHY Vendor register that may optionally be included in the core when the core is configured. Its size is also configurable and may be up to 32 bits. The structure of the register is up to the system designer, though users should note that the UTMI+ specification defines a 4-bit VControl register.

The register may be accessed at address 68h.

CONFIDENTIAL

SOFT CORES

Users should also note that:

(i) If a register of more than 8 bits is specified, any write must be made to the entire register, using either a 16-bit or a 32-bit write as appropriate. Writes to individual bytes are not supported.

(ii) The latency for the write (as measured between the positive edge of CLK at the end of the AHB write cycle and the positive edge of XCLK when the UTMI+ PHY VControl register is loaded) will be between $Hc + 3Xc$ and $Hc + 4Xc$, where Hc is a cycle of CLK and Xc is a cycle of XCLK. The minimum period between successive writes to the core's VControl, register must therefore be $Hc + 4Xc$ to ensure that the value is not corrupted while it is being synchronized to the XCLK domain.

3.6.2. VSTATUS

NOTE: READ ONLY!

VStatus is a UTMI+ PHY Vendor register that may optionally be included in the core when the core is configured. Its size is also configurable and may be up to 32 bits. The structure of the register is up to the system designer, though users should note that the UTMI+ specification defines an 8-bit VStatus register.

The register may be accessed at address 68h.

Users should also note that:

(i) The VSTATUS input bus is sampled once every 6 XCLK cycles.

(ii) The latency between the VSTATUS input bus from the PHY changing and the new value being read from the VStatus register (measured to the positive edge of CLK at the end of the AHB read cycle) will be between $2Hc + Xc$ and $3Hc + 6Xc$, where Hc is a cycle of CLK and Xc is a cycle of XCLK.

3.6.3. HWVERS

HWVers register is a 16-bit read-only register that returns information about the version of the RTL from which the core hardware was generated, in particular the RTL version number (v*xx.yyy*).

Address: 6Ch*; Reset value:* Version dependent

	D15	D14	…	D10	D9	…	D0
	RC		*xx*			*yyy*	
From CPU	r	r	…	r	r	…	r
From USB	r	r	…	r	r	…	r

Bit	Name	Function
D15	**RC**	Set to '1' if RTL used from a Release Candidate rather than from a full release of the core.
D14 – D10	*xx*	Major Version Number (Range 0 – 31).
D9 – D0	*yyy*	Minor Version Number (Range 0 – 999).

CONFIDENTIAL

SOFT CORES

3.7. ADDITIONAL CONFIGURATION REGISTERS

3.7.1. EPINFO

This 8-bit read-only register allows read-back of the number of TX and Rx endpoints included in the design.

Address: 78h*; Reset value:* Implementation dependent

	D7	D6	D5	D4	D3	D2	D1	D0
	RxEndPoints				TxEndPoints			
From CPU	r	r	r	r	r	r	r	r
From USB	r	r	r	r	r	r	r	r

Bit	Name	Function
D7 – D4	**RxEndPoints**	The number of Rx endpoints implemented in the design.
D3 – D0	**TxEndPoints**	The number of TX endpoints implemented in the design.

3.7.2. RAMINFO

This 8-bit read-only register provides information about the width of the RAM.

Address: 79h*; Reset value:* Implementation dependent

	D7	D6	D5	D4	D3	D2	D1	D0
	DMAChans				RamBits			
From CPU	r	r	r	r	r	r	r	r
From USB	r	r	r	r	r	r	r	r

Bit	Name	Function
D7 – D4	**DMAChans**	The number of DMA channels implemented in the design.
D3 – D0	**RamBits**	The width of the RAM address bus.

3.7.3. LINKINFO

This 8-bit register allows some delays to be specified.

Address: 7Ah*; Reset value:* 8'h5C

	D7	D6	D5	D4	D3	D2	D1	D0
	WTCON				WTID			
From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	r	r	r	r	r	r	r	r

CONFIDENTIAL

SOFT CORES

Bit	Name	Function
D7 – D4	**WTCON**	Sets the wait to be applied to allow for the user's connect/disconnect filter in units of 533.3ns. (The default setting corresponds to 2.667µs.)
D3 – D0	**WTID**	Sets the delay to be applied from IDPULLUP being asserted to IDDIG being considered valid in units of 4.369ms. (The default setting corresponds to 52.43ms.)

3.7.4. VPLEN

This 8-bit register sets the duration of the VBus pulsing charge.

Address: 7Bh*; Reset value:* 8'h3C

	D7	D6	D5	D4	D3	D2	D1	D0
	(msb)			**VPLEN**				**(lsb)**
From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	r	r	r	r	r	r	r	r

Bit	Name	Function
D7 – D0	**VPLEN**	Sets the duration of the VBus pulsing charge in units of 546.1 µs. (The default setting corresponds to 32.77ms.)

3.7.5. HS_EOF1

This 8-bit register sets the minimum time gap that is to be allowed between the start of the last transaction and the EOF for High-speed transactions.

Address: 7Ch*; Reset value:* 8'h80

	D7	D6	D5	D4	D3	D2	D1	D0
	(msb)			**HS_EOF1**				**(lsb)**
From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	r	r	r	r	r	r	r	r

Bit	Name	Function
D7 – D0	**HS_EOF1**	Sets for High-speed transactions the time before EOF to stop beginning new transactions, in units of 133.3ns. (The default setting corresponds to 17.07µs.)

3.7.6. FS_EOF1

This 8-bit register sets the minimum time gap that is to be allowed between the start of the last transaction and the EOF for Full-speed transactions.

Address: 7Dh*; Reset value:* 8'h77

	D7	D6	D5	D4	D3	D2	D1	D0
	(msb)			**FS_EOF1**				**(lsb)**
From CPU	rw	rw	rw	rw	rw	rw	rw	rw

CONFIDENTIAL

From USB	r	r	r	r	r	r	r	r

Bit	Name	Function
D7 – D0	**FS_EOF1**	Sets for Full-speed transactions the time before EOF to stop beginning new transactions, in units of 533.3ns. (The default setting corresponds to 63.46µs.)

3.7.7. LS_EOF1

This 8-bit register sets the minimum time gap that is to be allowed between the start of the last transaction and the EOF for Low-speed transactions.

Address: 7Eh*; Reset value:* 8'h72

	D7	D6	D5	D4	D3	D2	D1	D0
	(msb)			LS_EOF1				**(lsb)**
From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	r	r	r	r	r	r	r	r

Bit	Name	Function
D7 – D0	**LS_EOF1**	Sets for Low-speed transactions the time before EOF to stop beginning new transactions, in units of 1.067µs. (The default setting corresponds to 121.6µs.)

3.7.8. SOFT_RST

This 8-bit register will assert LOW the output reset signals NRSTO and NRSTOX. This register is self clearing and will be reset by the input NRST.

Address: 7Fh*; Reset value:* 8'h00

	unused	D1	D0
	(msb) SOFT_RST		**(lsb)**
From CPU		rw	rw
From USB		r	r

Bit	Name	Function
D7 – D2	**-**	*Unused, always returns zero.*
D1	**NRSTX**	The default value of this bit is 1'b0; When a 1 is written to this bit, the output NRSTXO will be asserted (LOW) within a minimum delay of 7 cycles of the CLK input. The output NRSTXO will be asynchronously asserted and synchronously de-asserted with respect to XCLK. This register is self clearing and will be reset by the input NRST.
D0	**NRST**	The default value of this bit is 1'b0; When a 1 is written to this bit, the output NRSTO will be asserted (LOW) within a minimum delay of 7 cycles of the CLK input. The output NRSTO will be asynchronously asserted and synchronously de-asserted with respect to CLK. This register is self clearing and will be reset by the input NRST.

3.8. EXTENDED REGISTERS

The following subsections detail additional registers that control and affect the operation of the MUSBMHDRC.

CONFIDENTIAL

SOFT CORES

3.8.1. RQPKTCOUNT

NOTE: HOST MODE ONLY!

For each Rx Endpoint 1 – 15, the MUSBMHDRC provides a 16-bit RqPktCount register. This read/write register is used in Host mode to specify the number of packets that are to be transferred in a block transfer of one or more Bulk packets of length MaxP to Rx Endpoint *n*. The core uses the value recorded in this register to determine the number of requests to issue where the AutoReq option (included in the RxCSR register) has been set.

For further information, see Section 8.5.2.

Note: Multiple packets combined into a single bulk packet within the FIFO count as one packet.

Address: 300 + 4*n*; *Reset value:* 16'h0000

D15	...	D0
(msb)	RqPktCount	(lsb)

From CPU	rw	...	rw
From USB	rw	...	rw

Bit	Name	Function
D15 – D0	**RqPktCount**	Sets the number of packets of size MaxP that are to be transferred in a block transfer. *Only used in Host mode when AutoReq is set. Has no effect in Peripheral mode or when AutoReq is not set.*

3.8.2. DOUBLE PACKET BUFFER DISABLE

For each Rx and Tx Endpoint 1 – 15, the MUSBMHDRC provides a DPktBufDis bit. These bits reside in a common set of DPktBufDis registers. One set of registers is dedicated to Rx Endpoints and another set is dedicated to TX endpoints. These read/write bits are used to control the use of double packet buffering on a per endpoint basis. It is ignored when Dynamic FIFO is enabled. When asserted (DPktBufDis equals 1'b1), the bit will disable double packet buffering for the corresponding endpoint regardless of the End Point FIFO Size and the INMAXP size relationship. When de-asserted (DPktBufDis equals 1'b0), this bit does NOT necessarily enable double packet buffering but rather allows double packet buffering to be determined based upon the End Point FIFO Size and INMAXP size relationship. See Section 8.4 for details.

3.8.2.1. RX DPKTBUFDIS

Rx DPktBufDis is a 16-bit register that indicates which of the Rx endpoints have disabled the double packet buffer functionality described in section 8.4.2.2 of the MUSBMHDRC Product Specification.

Note: Bits relating to endpoints that have not been configured may be asserted by writing a '1' their respective register; however the disable bit will have no observable effect.

Address: 340h; *Reset value:* 16'h0000

D15	D14	D13	D12	D11	D10	D9	D8
EP15 RxDis	EP14 RxDis	EP13 RxDis	EP12 RxDis	EP11 RxDis	EP10 RxDis	EP9 RxDis	EP8 RxDis

From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	r	r	r	R	r	r	r	r

D7	D6	D5	D4	D3	D2	D1	D0
EP7 RxDis	EP6 RxDis	EP5 RxDis	EP4 RxDis	EP3 RxDis	EP2 RxDis	EP1 RxDis	Unused

From CPU	rw	rw	rw	rw	rw	rw	rw	r
From USB	r	r	r	r	r	r	r	r

CONFIDENTIAL

Bit	Name	Function
D15	**EP15 RxDis**	Rx Double Packet Buffer Disable for Endpoint 15.
D14	**EP14 RxDis**	Rx Double Packet Buffer Disable for Endpoint 14.
D13	**EP13 RxDis**	Rx Double Packet Buffer Disable for Endpoint 13.
D12	**EP12 RxDis**	Rx Double Packet Buffer Disable for Endpoint 12.
D11	**EP11 RxDis**	Rx Double Packet Buffer Disable for Endpoint 11.
D10	**EP10 RxDis**	Rx Double Packet Buffer Disable for Endpoint 10.
D9	**EP9 RxDis**	Rx Double Packet Buffer Disable for Endpoint 9.
D8	**EP8 RxDis**	Rx Double Packet Buffer Disable for Endpoint 8.
D7	**EP7 RxDis**	Rx Double Packet Buffer Disable for Endpoint 7.
D6	**EP6 RxDis**	Rx Double Packet Buffer Disable for Endpoint 6.
D5	**EP5 RxDis**	Rx Double Packet Buffer Disable for Endpoint 5.
D4	**EP4 RxDis**	Rx Double Packet Buffer Disable for Endpoint 4.
D3	**EP3 RxDis**	Rx Double Packet Buffer Disable for Endpoint 3.
D2	**EP2 RxDis**	Rx Double Packet Buffer Disable for Endpoint 2.
D1	**EP1 RxDis**	Rx Double Packet Buffer Disable for Endpoint 1.
D0	**Unused**	Reserved

3.8.2.2. TX DPKTBUFDIS

Tx DPktBufDis is a 16-bit register that indicates which of the TX endpoints have disabled the double packet buffer functionality described in section 8.4.1.2 of the MUSBMHDRC Product Specification.

Note: Bits relating to endpoints that have not been configured may be asserted by writing a '1' their respective register; however the disable bit will have no observable effect.

Address: 342h*; Reset value:* 16'h0000

	D15	D14	D13	D12	D11	D10	D9	D8
	EP15 TxDis	**EP14 TxDis**	**EP13 TxDis**	**EP12 TxDis**	**EP11 TxDis**	**EP10 TxDis**	**EP9 TxDis**	**EP8 TxDis**
From CPU	rw	rw	Rw	rw	Rw	rw	rw	rw
From USB	r	r	R	r	R	r	r	r

	D7	D6	D5	D4	D3	D2	D1	D0
	EP7 TxDis	**EP6 TxDis**	**EP5 TxDis**	**EP4 TxDis**	**EP3 TxDis**	**EP2 TxDis**	**EP1 TxDis**	**Unused**
From CPU	rw	rw	Rw	rw	Rw	rw	rw	r
From USB	r	r	R	r	R	r	r	r

CONFIDENTIAL

Bit	Name	Function
D15	**EP15 TxDis**	Tx Double Packet Buffer Disable for Endpoint 15.
D14	**EP14 TxDis**	Tx Double Packet Buffer Disable for Endpoint 14.
D13	**EP13 TxDis**	Tx Double Packet Buffer Disable for Endpoint 13.
D12	**EP12 TxDis**	Tx Double Packet Buffer Disable for Endpoint 12.
D11	**EP11 TxDis**	Tx Double Packet Buffer Disable for Endpoint 11.
D10	**EP10 TxDis**	Tx Double Packet Buffer Disable for Endpoint 10.
D9	**EP9 TxDis**	Tx Double Packet Buffer Disable for Endpoint 9.
D8	**EP8 TxDis**	Tx Double Packet Buffer Disable for Endpoint 8.
D7	**EP7 TxDis**	Tx Double Packet Buffer Disable for Endpoint 7.
D6	**EP6 TxDis**	Tx Double Packet Buffer Disable for Endpoint 6.
D5	**EP5 TxDis**	Tx Double Packet Buffer Disable for Endpoint 5.
D4	**EP4 TxDis**	Tx Double Packet Buffer Disable for Endpoint 4.
D3	**EP3 TxDis**	Tx Double Packet Buffer Disable for Endpoint 3.
D2	**EP2 TxDis**	Tx Double Packet Buffer Disable for Endpoint 2.
D1	**EP1 TxDis**	Tx Double Packet Buffer Disable for Endpoint 1.
D0	**Unused**	Reserved

3.8.3. C_T_UCH

This register sets the chirp timeout. This number when multiplied by 4 represents the number of XCLK cycles before the timeout occurs. That is, if XCLK is 30MHz, this number represents the number of 133ns time intervals before the timeout occurs. If XCLK is 60MHz, this number represents the number of 67ns time intervals before the timeout occurs. Although this bit is written by the host in the CLK domain, the counter that utilizes this value is in the XCLK domain. No time domain crossing is provided as the value in this register is a static. The default value is the value of the compiler directive of the same name located in the configuration file musbmhdrc_cfg.v.

Address: 344h; Reset value: Various.

	D15	D14	D13	D12	D11	D10	D9	D8
				C_T_UCH[15:8]				
From CPU	rw	rw	Rw	rw	Rw	rw	rw	rw
From USB	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	D7	D6	D5	D4	D3	D2	D1	D0
				C_T_UCH[7:0]				
From CPU	rw	rw	Rw	rw	Rw	rw	rw	rw
From USB	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

CONFIDENTIAL

Bit	Name	Function
D15-D0	**C_T_UCH**	Configurable Chirp Timeout timer; The default value is determined by compiler directive in musbhsfc_xcfg.v file. The default value is 203Ah if the host PHY data width is 16 bits (XCLK is 30MHz) and 4074h if the PHY data width is 8 bits (XCLK is 60Mhz) corresponding to a delay of 1.1ms.

3.8.4. C_T_HSRTN

This register sets the delay from the end of High Speed resume signaling to enable the UTM normal operating mode.   This number when multiplied by 4 represents the number of XCLK cycles before the timeout occurs. That is, if XCLK is 30MHz, this number represents the number of 133ns time intervals before the timeout occurs. If XCLK is 60MHz, this number represents the number of 67ns time intervals before the timeout occurs. Although this bit is written by the host in the CLK domain, the counter that utilizes this value is in the XCLK domain. No time domain crossing is provided as the value in this register is a static. The default value is the value of the compiler directive of the same name located in the configuration file musbmhdrc_cfg.v.

*Address: 346*h*; Reset value: Various.*

	D15	D14	D13	D12	D11	D10	D9	D8
				C_T_HSRTN[15:8]				
From CPU	rw	rw	Rw	rw	rw	rw	rw	rw
From USB	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

	D7	D6	D5	D4	D3	D2	D1	D0
				C_T_HSRTN[7:0]				
From CPU	rw	rw	Rw	rw	rw	rw	rw	rw
From USB	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Bit	Name	Function
D15-D0	C_T_HSRTN	The delay from the end of High Speed resume signaling to enabling UTM normal operating mode. The default value is determined by compiler directive in musbhsfc_xcfg.v file. The default value is 19h if the host PHY data width is 16 bits (XCLK is 30MHz) and 32h if the PHY data width is 8 bits (XCLK is 60Mhz) corresponding to a delay of 3us.

3.8.5. C_T_HSBT

Per USB 2.0, Section 7.1.19.2, a high-speed host or device expecting a response to a transmission must not timeout the transaction if the interpacket delay is less than 736 bit times, and it must timeout the transaction if no signaling is seen within 816 bit times. This register represents the value to be added to the mimumum high speed timeout period of 736 bit times. The timeout period can be increased in increments of 64 high speed bit times (133 ns). There are 16 possible values. By default, the adder is 0 thus setting the high speed timeout to its minimum value. Use of this register will allow the high speed timeout to be set to values that are greater the the maximum specifed in USB 2.0 making the MUSBMHDRC non-compliant.

Address: 348h; *Reset value:* 4'b0000

	D3	D2	D1	D0
	(MSB)	**HS Timeout Adder**		(LSB)
From CPU	rw	rw	rw	rw
From USB	r	r	r	r

CONFIDENTIAL

Bit	Name	Function
D3-D0	C_T_HSBT	The value added to the minimum High Speed Timeout period (736 bit times) in increments of 64 High Speed bit times. This allows the turn around timeout period to be set to 16 possible values as follows:

Register Value	HS Turnaround Timeout (HS Bit times)	HS Turnaround Timeout (us)
0	736	1.534
1	800	1.667
2	864	1.801
3	928	1.934
4	992	2.067
5	1056	2.201
6	1120	2.334
7	1184	2.467
8	1248	2.601
9	1312	2.734
10	1376	2.868
11	1440	3.001
12	1504	3.134
13	1568	3.268
14	1632	3.401
15	1696	3.534

3.9. DMA REGISTERS

The DMA registers are only available if the MUSBMHDRC is configured to use at least one internal DMA channel. There is one set of registers per channel.

3.9.1. DMA_INTR

This register provides an interrupt for each DMA channel. This interrupt register is cleared when read. When any bit of this register is set, the output DMA_NINT is asserted low. Events that cause interrupts to be set is described in section 17 (The optional DMA Controller description). Bits in this register will only be set if the DMA Interrupt Enable bit for the corresponding channel is enabled (register DMA_CNTL.D3).

Address: 200h; Reset value: 00h

	D7	D6	D5	D4	D3	D2	D1	D0
				DMA_INTR[7:0]				
From CPU	rw	rw	rw	rw	rw	r	r	R
From USB	set	set	set	set	set	set	set	Set

Bit	Name	Function
D7	CH8 DMA_INTR	Channel 8 DMA Interrupt
D6	CH7 DMA_INTR	Channel 7 DMA Interrupt
D5	CH6 DMA_INTR	Channel 6 DMA Interrupt

CONFIDENTIAL

D4	CH5 DMA_INTR	Channel 5 DMA Interrupt
D3	CH4 DMA_INTR	Channel 4 DMA Interrupt
D2	CH3 DMA_INTR	Channel 3 DMA Interrupt
D1	CH2 DMA_INTR	Channel 2 DMA Interrupt
D0	CH1 DMA_INTR	Channel 1 DMA Interrupt

3.9.2. DMA_CNTL

This register is only available if the MUSBMHDRC is configured to use at least one internal DMA channel. This register provides provdes the DMA transfer control for each channel. The enabling, transfer direction, transfer mode, the DMA burst modes are all controlled by this register.

*Address: 204h + (n-1)*10h;n=channel number 1 thru 8; Reset value: 00h*

	D10	D9	D8
	DMA_BRSTM		DMA_ERR
From CPU	rw	rw	rw
From USB	r	r	rw

	D7	D6	D5	D4	D3	D2	D1	D0
	DMAEP				DMAIE	DMAMODE	DMA_DIR	DMA_EN
From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	r	r	r	r	r	r	r	r

Bit	Name	Function
D10-D9	DMA_BRSTM	Burst Mode 00 = Burst Mode 0 : Bursts of unspecified length 01 = Burst Mode 1 : INCR4 or unspecified length 10 = Burst Mode 2 : INCR8, INCR4 or unspecified length 11 = Burst Mode 3 : INCR16, INCR8, INCR4 or unspecified length
D8	DMA_ERR	Bus Error Bit. Indicates that a bus error has been observed on the input AHB_HRESPM[1:0]. This bit is cleared by software.
D7-D4	DMAEP	The endpoint number this channel is assigned to.
D3	DMAIE	DMA Interrupt Enable.
D2	DMAMODE	This bit selects the DMA Transfer Mode. 0 = DMA Mode0 Transfer 1 = DMA Mode1 Transfer

CONFIDENTIAL

D1	DMA_DIR	This bit selects the DMA Transfer Direction. 0 = DMA Write (RX Endpoint) 1 = DMa Read (TX Endpoint)
D0	DMA_ENAB	This bit enables the DMA transfer and will cause the transfer to begin.

3.9.3. DMA_ADDR

This register identifies the current memory address of the corresponding DMA channel. The Initial memory address written to this register must have a value such that its modulo 4 value is equal to 0. That is, DMA_ADDR[1:0] must be equal to 2'b00. The lower two bits of this register are read only and cannot be set by software. As the DMA transfer progresses, the memory address will increment as bytes are tranfered.

*Address: 208h + (n-1)*10h;n=channel number 1 thru 8; Reset value: 00h*

	D31	D30	D29	D28	D27	D26	D25	D24
	DMA_ADDR[31:24]							
From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	rw	rw	rw	rw	rw	rw	rw	rw
	D23	D22	D21	D20	D19	D18	D17	D16
	DMA_ADDR[23:16]							
From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	rw	rw	rw	rw	rw	rw	rw	rw
	D15	D14	D13	D12	D11	D10	D9	D8
	DMA_ADDR[15:8]							
From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	rw	rw	rw	rw	rw	rw	rw	rw
	D7	D6	D5	D4	D3	D2	D1	D0
	DMA_ADDR[7:0]							
From CPU	rw	rw	rw	rw	rw	rw	r	r
From USB	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Name	Function
D31 – D0	DMA_ADDR	The DMA memory address. **Note that the initial memory address written to this register must have a value such that it's modulo 4 value is equal to 0. That is, DMA_ADDR[1:0] must be equal to 2'b00. The lower two bits of this register are read only and cannot be set by software.**

3.9.4. DMA_COUNT

This register identifies the current DMA count of the transfer. Software will set the initial count of the transfer which identifies the entire transfer length. As the count progresses this count is decremented as bytes are transfered.

CONFIDENTIAL

Address: 20Ch + (n-1)*10h;n=channel number 1 thru 8; Reset value: 00h

D31	D30	D29	D28	D27	D26	D25	D24
DMA_COUNT[31:24]							
From CPU rw	rw	rw	rw	rw	rw	rw	rw
From USB rw	rw	rw	rw	rw	rw	rw	rw

D23	D22	D21	D20	D19	D18	D17	D16
DMA_COUNT [23:16]							
From CPU rw	rw	rw	rw	rw	rw	rw	rw
From USB rw	rw	rw	rw	rw	rw	rw	rw

D15	D14	D13	D12	D11	D10	D9	D8
DMA_COUNT[15:8]							
From CPU rw	rw	rw	rw	rw	rw	rw	rw
From USB rw	rw	rw	rw	rw	rw	rw	rw

D7	D6	D5	D4	D3	D2	D1	D0
DMA_COUNT[7:0]							
From CPU rw	rw	rw	rw	rw	rw	r	r
From USB rw	rw	rw	rw	rw	rw	rw	rw

Bit	Name	Function
D31 – D0	DMA_COUNT	The DMA memory address for the corresponding DMA channel. **Note: If DMA is enabled with a count of 0, the bus will not be requested and a DMA interrupt will be generated.**

3.10. DYNAMIC FIFO REGISTERS

The dynamic FIFO registers are only available if the MUSBMHDRC is configured to use Dynamic FIFO Sizing. There is one set of register per End Point Excluding End Point 0. These are indexed registers therefore to access them the INDEX register at address 0Eh must be set to the appropriate end point. The limitations and use of Dynamic Fifo registers is described in section 19.

3.10.1. TXFIFOSZ

TxFIFOsz is a 5-bit register which controls the size of the selected TX endpoint FIFO.

Address: 62h; Reset value: 5'h00

	D4	D3	D2	D1	D0
	DPB	**SZ3**	**SZ2**	**SZ1**	**SZ0**
From CPU	rw	rw	rw	rw	rw
From USB	r	r	r	r	r

CONFIDENTIAL

Bit	Name	Function
D4	**DPB**	Defines whether double-packet buffering supported. When '1', double-packet buffering is supported. When '0', only single-packet buffering is supported.
D3 – D0	**SZ[3:0]**	Maximum packet size to be allowed for (*before* any splitting within the FIFO of Bulk/High-Bandwidth packets prior to transmission – see Sections 8.4.1.3, 8.4.1.4 and 8.5.3)

	SZ[3:0]			Packet Size (Bytes)
0	0	0	0	8
0	0	0	1	16
0	0	1	0	32
0	0	1	1	64
0	1	0	0	128
0	1	0	1	256
0	1	1	0	512
0	1	1	1	1024
1	0	0	0	2048
1	0	0	1	4096

If DPB = 0, the FIFO will also be this size; if DPB = 1, the FIFO will be twice this size.

3.10.2. RXFIFOSZ

RxFIFOsz is a 5-bit register which controls the size of the selected Rx endpoint FIFO.

Address: 63h*; Reset value:*5'h00

	D4	D3	D2	D1	D0
	DPB	**SZ3**	**SZ2**	**SZ1**	**SZ0**
From CPU	rw	rw	rw	rw	rw
From USB	r	r	r	r	r

Bit	Name	Function
D4	**DPB**	Defines whether double-packet buffering supported. When '1', double-packet buffering is supported. When '0', only single-packet buffering is supported.
D3 – D0	**SZ[3:0]**	Maximum packet size to be allowed for (*after* any combination within the FIFO of Bulk/High-Bandwidth packets following their reception – see Sections 8.4.2.3, 8.4.2.4 and 8.5.2)

	SZ[3:0]			Packet Size (Bytes)
0	0	0	0	8
0	0	0	1	16
0	0	1	0	32
0	0	1	1	64
0	1	0	0	128
0	1	0	1	256
0	1	1	0	512
0	1	1	1	1024
1	0	0	0	2048
1	0	0	1	4096

If DPB = 0, the FIFO will also be this size; if DPB = 1, the FIFO will be twice this size.

CONFIDENTIAL

SOFT CORES

3.10.3. TXFIFOADD

TxFIFOadd is a 14-bit register which controls the start address of the selected Tx endpoint FIFO.

Address: 64h*; Reset value*14'h0000

D13	D12	…	D0
–	**AD12**	**…**	**AD0**

From CPU — rw — rw — rw — rw
From USB — r — r — r — r

Bit	Name	Function
D13	–	*Reserved for future use.*
D12 – D0	**AD[12:0]**	Start address of the endpoint FIFO in units of 8 bytes as follows:

AD[12:0]				Start Address
0	0	0	0	0000
0	0	0	1	0008
0	0	0	2	0010
		…		…
1	F	F	F	FFF8

3.10.4. RXFIFOADD

RxFIFOadd is a 14-bit register which controls the start address of the selected Rx endpoint FIFO.

Address: 66h*; Reset value*14'h0000

D13	D12	…	D0
–	**AD12**	**…**	**AD0**

From CPU — rw — rw — rw — rw
From USB — r — r — r — r

Bit	Name	Function
D13	–	*Reserved for future use.*
D12 – D0	**AD[12:0]**	Start address of the endpoint FIFO in units of 8 bytes as follows:

AD[12:0]				Start Address
0	0	0	0	0000
0	0	0	1	0008
0	0	0	2	0010
		…		…
1	F	F	F	FFF8

4. CLOCKING AND RESET

4.1. CLOCKING

The MUSBMHDRC is designed to take its system clock CLK from the AHB bus clock. This avoids any resynchronization logic

CONFIDENTIAL

SOFT CORES

between the MUSBMHDRC and the AHB, allowing single cycle access to both the MUSBMHDRC registers and the FIFOs.

The maximum frequency at which the core can be clocked is determined by the maximum frequency for which the MUSBMHDRC can be synthesized. This is typically 80MHz in a 0.35µ technology, 100MHz in a 0.25µ technology, and 120MHz in a 0.18µ technology.

The minimum frequency at which the core (and the AHB bus) can be clocked depends on the selected UTMI width and the technology implementation. The 8-bit UTMI allows the core to be clocked at minimum speeds close to 30MHz for some technology implementations.

For higher resolution on what actual minimum frequency will be for specific technology implementations the following bounding equation is given:

$$Tclk <= (2Txclk) - (Tskew/2) - (Tsetup/2) - (Thold/2)$$

Tclk is the maximum period of signal CLK that is guaranteed to correctly transfer received data across the XCLK/CLK time domain crossing.

Txclk is the period of the signal XCLK.

Tsetup is the required setup delay of the technology.

Thold is the required hold delay of the technology.

Tskew is the worst case difference in propagation delay among the following group of signals:

       Musbhdrc.usync_1.rxbuff0[15:0]

       Musbhdrc.usync_1.rxbuff1[15:0]

       Musbhdrc.usync_1.rxbval0

       Musbhdrc.usync_1.rxbval1

Please note that the above equation assumes an ideal clock. Please add technology dependent parameters for a higher level of resolution.

4.2. RESET

The MUSBMHDRC has two clock domains; the XCLK domain which is the clock recovered from the received data by the PHY and the CLK domain used by the AHB Bus. There are two input reset signals (NRST and NRSTX) provided; one for each clock domain. NRST may be asynchronously asserted and must synchronously de-asserted (synchronous with CLK). NRSTX may be asynchronously asserted and must synchronously de-asserted (synchronous with XCLK). If these synchronous resets are not available, one could use the synchronization logic provided with the MUSBMHDRC. In this case, the asynchronous reset is input on the port NRSTA. The two output signals, NRSTO and NRSTXO are generated from NRSTA as follows:

Generation of reset synchronous with XCLK

CONFIDENTIAL
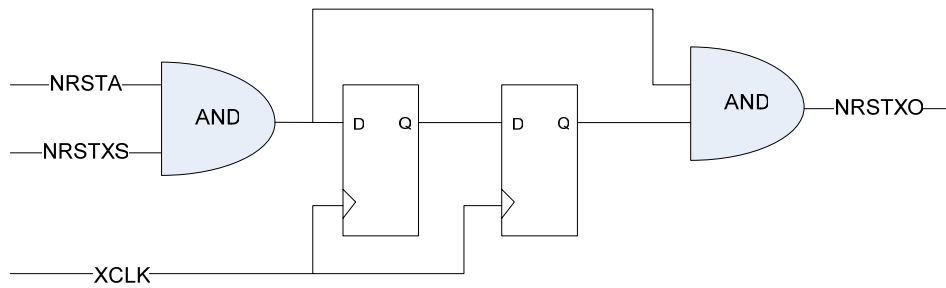
Generation of reset synchronous with CLK
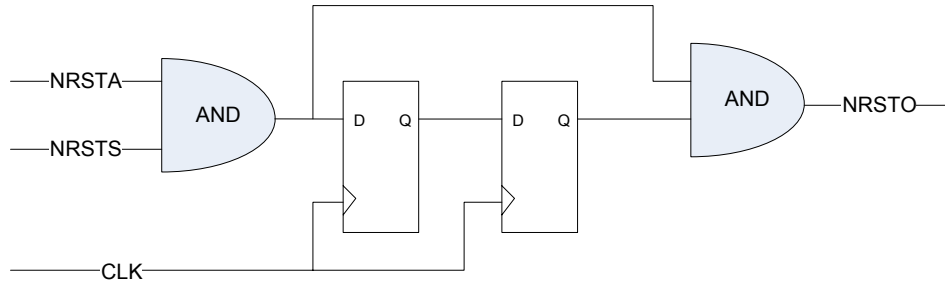
The signals NRSTXS and NRSTS are internal signals which provide software the ability reset the core. NRSTS and NRSTXS are asserted by writing to register 7F, bits 0 and 1 respectively. In order to utilize this synchronization logic, the output NRSTO must be connected to the input NRST and the output NRSTXO must be connected to the input NRSTX. If the customer does not with to use this synchronization logic, the input NRSTA should be tied high or low and the outputs NRSTO and NRSTXO should be left unconnected. NRSTA is not used for any other purpose in the MUSBMHDRC.

5. CPU INTERFACE

The basic MUSBMHDRC core offers a 32-bit synchronous CPU interface that follows the format specified for compatibility with an AMBA AHB bus.

All inputs are sampled on the positive edge of CLK, and outputs change following the positive edge of CLK.

6. DATA WIDTH

The AHB data interface is 32 bits wide.

Data can be transferred as single bytes, 16-bit half-words or 32-bit words.

In half-word and word transfers, the bytes are normally transferred lowest-order byte first as follows: (B0 represents the first byte to be transferred)

Little-endian transfers

Transfer size	Address Offset	D[31:24]	D[23:16]	D15:8]	D[7:0]
32 bits	0	B3	B2	B1	B0
16 bits	0			B1	B0
16 bits	2	B1	B0		

CONFIDENTIAL

SOFT CORES

8 bits	0				B0
8 bits	1			B0	
8 bits	2		B0		
8 bits	3	B0			

7. RAM INTERFACE

The FIFOs for all the endpoints are implemented in a single block of synchronous single-port RAM. The RAM is not included in the MUSBMHDRC, but should be connected to the MUSBMHDRC via the RAM interface.

The RAM address bus, output data bus and control signals all become valid following the positive edge of CLK. Read data is clocked into the core on the following positive edge. (See the timing diagrams in Sections 20.3 and 20.4.) A synchronous RAM block clocked on the negative edge of CLK would require an access time of less than half a clock cycle.

The RAM data bus width is 32 bits.

8. USB INTERFACE

The MUSBMHDRC is designed to be connected to a transceiver macrocell that complies with the UTMI+ Specification (Level 3). It can be configured to connect to either an 8-bit 60MHz or 16-bit 30MHz transceiver macrocell. (Alternatively, the core can be used with an optional USB 1.1 Full-Speed PHY interface as described in Section 8.1.)

For *On-The-Go* operations, the MUSBMHDRC provides:

- a DRVVBUS output, which may be used in an 'A' device configuration to drive +5V onto the USB VBus wire

- a CHRGVBUS output, which may be used in a 'B' device configuration to provide the appropriate pulse to the VBus to initiate a session (e.g. by charging the VBus to the Session Start threshold)

- a DISCHRGVBUS output, which may be used in a 'B' device configuration to discharge the VBus, down to a low enough level to start Session Request Protocol (SRP)

- DPPULLDOWN and DMPULLDOWN outputs for connecting/disconnecting the pull-down resistors on the D+ and D- lines as required when the core is being used for point-to-point communications with another USB device. (The size these resistors should be is specified in the *USB On-The-Go* Specification.)

The MUSBMHDRC also has VBUSVALID, AVALID and SESSEND inputs to identify to it the level of VBus relative to the various thresholds concerned in session control for *On-The-Go* devices. These signals should be connected to voltage comparators which respectively detect when the VBus voltage is above the VBus Valid threshold (required to be between 4.4V and 4.75V), above the Session Valid threshold for an 'A' device (required to be between 0.8V and 2V) and when it is above the Session End threshold (required to be between 0.2V and 0.8V).

(Details of the tolerances on these voltage ranges are given in the *USB On-The-Go* specification.)

The state of the DRVVBUS and CHRGVBUS signals reflect selections made in the DevCtl register. The state of the VBUSVALID, AVALID and SESSEND signals can be deduced from the VBus[1:0] bits of the DevCtl register, the values of which indicate the current level of VBus relative to the selected VBUSVALID, AVALID and SESSEND levels. (Details of the required charging currents, timings etc. are given in the UTMI+ specification.)

The USB interface also features an IDDIG signal which indicates whether the device that is plugged into the MUSBMHDRC is A-type or B-type. This input should be high when a B-type device is plugged in and low when an A-type device is plugged in. The device type is determined by sampling the incoming ID line, this sampling being enabled only when required through use of the core's IDPULLUP output to switch in an appropriate pull-up resistor on the ID line. A diagram showing an example connection

CONFIDENTIAL

MUSBMHDRC

is shown on the following page.

SOFT
CORES

5/25/2007 PSPG 40161.003-FC

Example Connection to UTM+ Macrocell

Note: The pull-up resistors required for the DP/DM signals are incorporated in the UTM+ module.
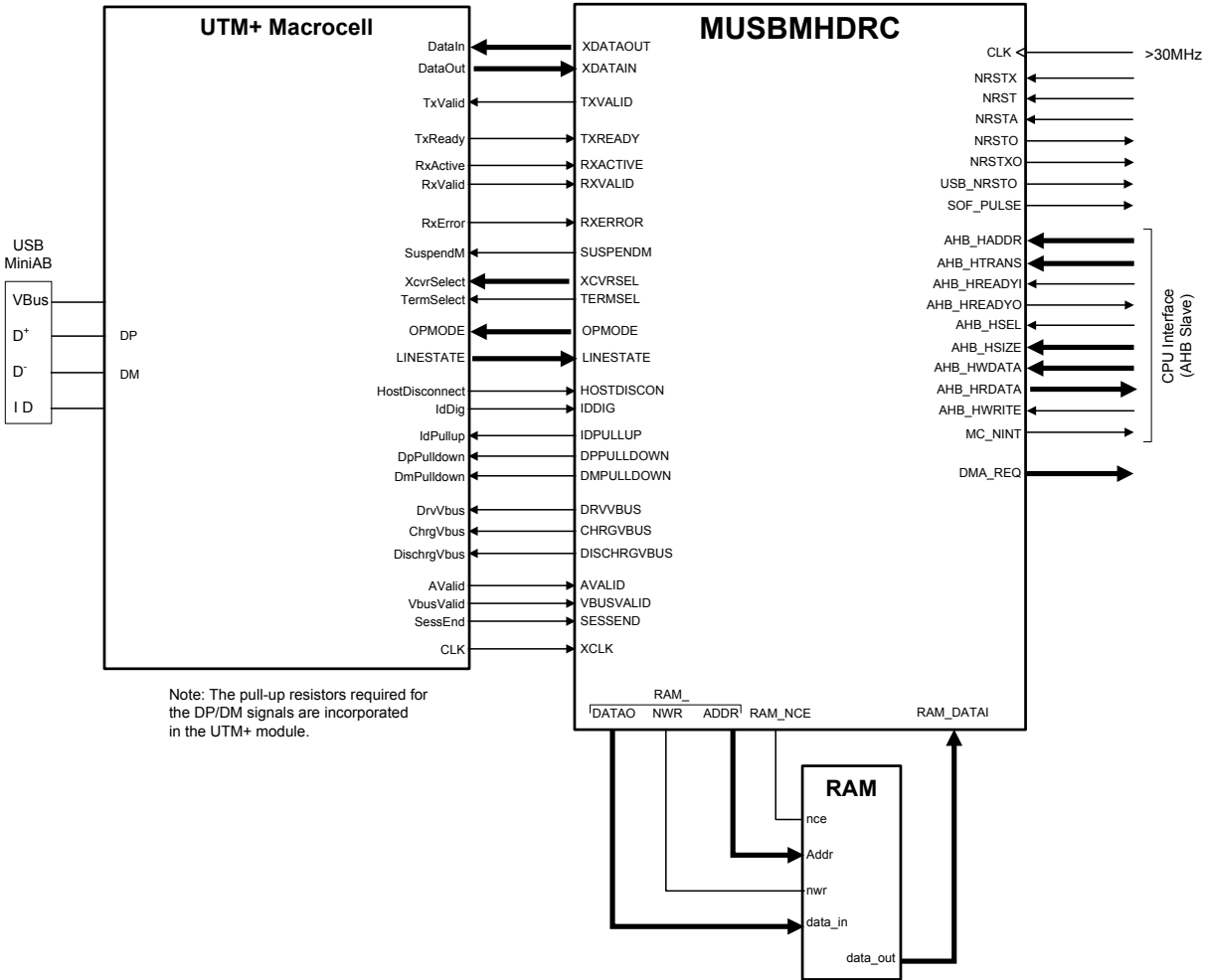
8.1. OPTIONAL USB 1.1 PHY INTERFACE

Supplied with the MUSBMHDRC core is a module (**fsi**) that can, if required, be instantiated alongside the MUSBMHDRC top-level module (**musbmhdrc**) to allow the core to operate at full-speed or low-speed with a USB 1.1 PHY instead of a UTMI+ PHY. If required, further modules (**i2c** and **i2cmstr**) can be instantiated alongside the **fsi** module to support use of the MUSBMHDRC with I²C-controlled USB 1.1 PHYs.

In Tx operations, the **fsi** module provides conversion of 8-bit parallel to serial data, automatic addition of synchronization and end-of-packet bits, NRZI encoding and automatic bit stuffing. In Rx operations, this module provides a digital phase lock loop for receive data, conversion of serial data to 8-bit parallel data, stripping of synchronization and end-of-packet bits, NRZI decoding ands stuff bit error checking.

The **fsi** module is instantiated alongside the MUSBMHDRC core in the supplied **musbmhdrc_fsp.v** wrapper. The effect on the signal I/O is shown in Section 8.1.1 below. If the I²C bus option is required (requiring the further **i2c** and **i2cmstr** modules), then the wrapper to use is **musbmhdrc_i2c.v**, and the effect on the I/O is as shown in Section 8.1.2 below. (The **i2cmstr** module is instantiated in the **i2c** module.)

An external 60MHz clock must be provided to drive both the XCLK input of the MUSBMHDRC, the **fsi** module and the **i2c** module (if used). Further, if the **musbmhdrc_i2c.v** wrapper is used to apply the optional I²C bus interface, then – in addition – the **musbmhdrc_pcfg.v** file must be edited to set at least the correct PHY address and internal register configuration (as documented within the file itself). *Note:* The **musbmhdrc_pcfg.v** file is supplied set-up for use with the Philips ISP1301 transceiver. Different defines will be needed where the core is used with a different I²C-controlled transceiver.

The core (and the configuration script) also offers the choice of output driver formats – DP/DM (i.e. D+/D-) or DAT/SE0. The DP/DM format is selected by default; the DAT/SE0 format may be selected either through the configuration script or by defining the C_DATSE0 configuration parameter.
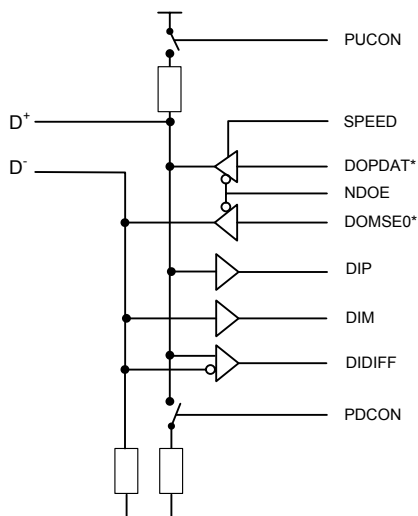
*After appropriate decoding where the
DAT/SE0 output driver format is used

As illustrated above, the DOPDAT and DOMSE0 outputs should drive the D+ and D- wires through buffers which are enabled when NDOE is low (after suitable decoding where the DAT/SE0 output driver format is used). Equally, the DIP and DIM inputs should be driven by single ended drivers connected to D+ and D-. The DIDIFF input should be driven from a differential driver connected to D+ and D-. The SPEED signal may be used to optimize the slew-rate of the drivers for full or low speed operation.

Similarly, the PUCON, PU_LO and PDCON signals are intended to be used to connect a pull-up resistor or pull-down resistor to the D+ wire. When the MUSBMHDRC operates as a host, pull-down resistors need to be connected to the USB D+ and D- wires. When it operates as a peripheral, the same pull-down resistor is needed on the D- wire but a pull-up resistor is needed on the USB

CONFIDENTIAL

D+ wire. When PUCON is high, the pull-up resistor should be connected between the D+ wire and +3.3V. When PUCON is low, this pull-up resistor should be disconnected. Similarly the pull-down resistor should be connected between D+ and ground when PDCON is high, and disconnected when PDCON is low. The PU_LO signal enables the device to select between the higher and lower value pull-up resistors allowed by Section 7.1.5 of the USB 2.0 specification by indicating when the USB bus is idle.

The VBUSLO signal is used in conjunction with VBUSVALID and AVALID for session request and for implementing host negotiation protocol between *USB On-The-Go* devices. These signals need to be connected to voltage comparators. CID is used to indicate the type of connector (mini-A or mini-B) that is plugged in and should be connected (via a pull-up) to the ID pin of a mini-AB receptacle.

Note: When the MUSBMHDRC is in Suspend mode (SUSPEND is high), the drivers may be powered-down. The POWERDWN output is also asserted. For power-saving, this signal may be used to stop CLK. However the single ended driver to DIP must remain powered so that the MUSBMHDRC can detect Resume signaling on the bus.

(If you require further information about using the MUSBMHDRC with a USB 1.1 PHY, please contact Customer Support.)

8.1.1. THE STANDARD USB 1.1 PHY INTERFACE

The following diagram shows how the **fsi** module attaches to the MUSBMHDRC core, and hence the interface presented when the **musbmhdrc_fsp.v** wrapper is used.
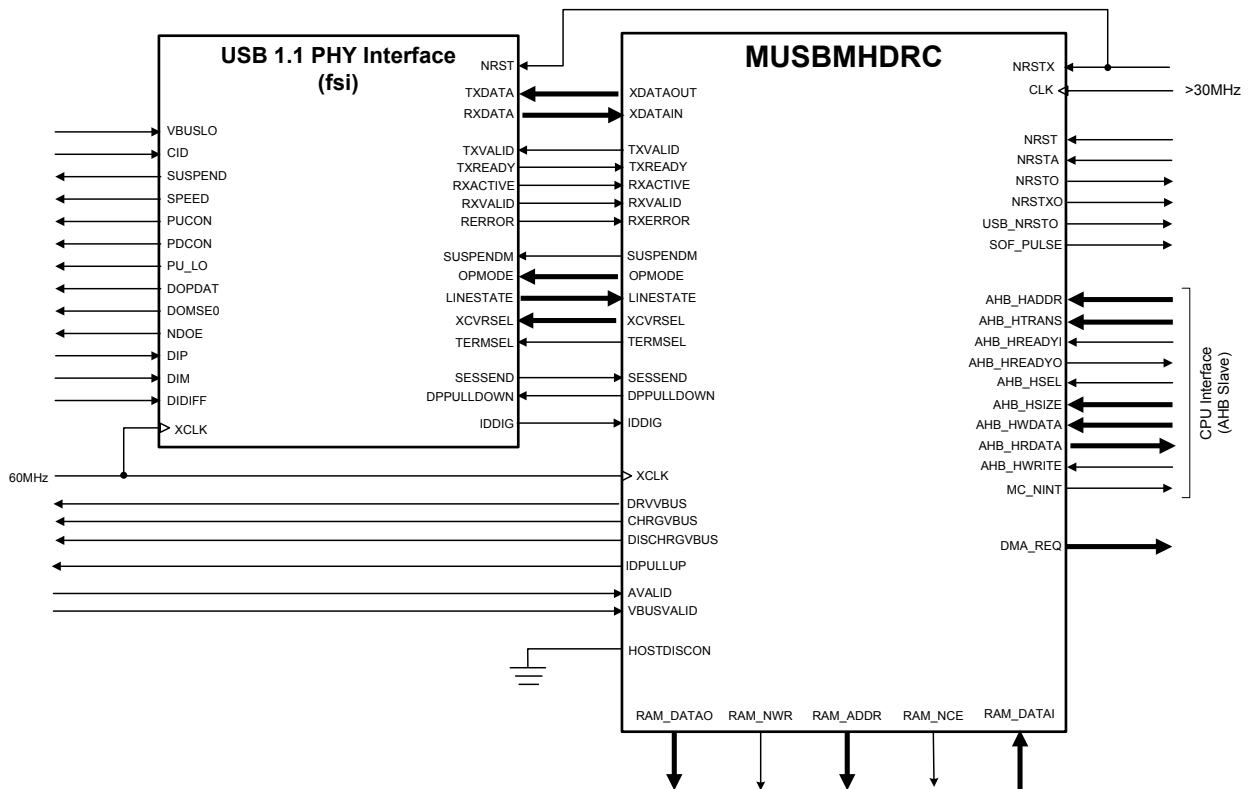
The signals that adding this module to the MUSBMHDRC core introduces into the overall pin-out are listed in the following table.

SIGNAL	TYPE	DESCRIPTION
VBUSLO	Input	VBus compared to Session End threshold (required to be between 0.2V and 0.8V). 1 = above the Session End threshold, 0 = below the Session End threshold.
CID	Input	Connector ID, deduced by sampling the device ID line. 1=B-type, 0=A-type.

CONFIDENTIAL

SIGNAL	TYPE	DESCRIPTION
SUSPEND	Output	This signal goes high when the MUSBMHDRC is in Suspend mode.
SPEED	Output	Transceiver operating speed. 1=full-speed, 0=low-speed.
PUCON	Output	When high, a pull-up resistor should be connected to D+.
PDCON	Output	When high, a pull-down resistor should be connected to D+. (The pull-down resistor required on D- should be permanently connected.)
PU_LO	Output	When high, this signal indicates that the USB is idle and so the lower value pull-up resistor may be used (if implemented). When low, it indicates that the USB is busy and so the higher value pull-up resistor should be used.
DOPDAT	Output	Provides either D+ output or DAT output depending on core configuration.
DOMSE0	Output	Provides either D- output or SE0 output depending on core configuration.
NDOE	Output	Output enable for DOP, DOM. Active low.
DIP	Input	D+ single-ended input.
DIM	Input	D- single-ended input.
DIDIFF	Input	Differential input.

8.1.2. USB 1.1 PHY INTERFACE WITH I²C-BUS CONTROL OPTION

The following diagram shows how the **i2c** module attaches to the core and hence the interface presented when the **musbmhdrc_i2c.v** wrapper is used.
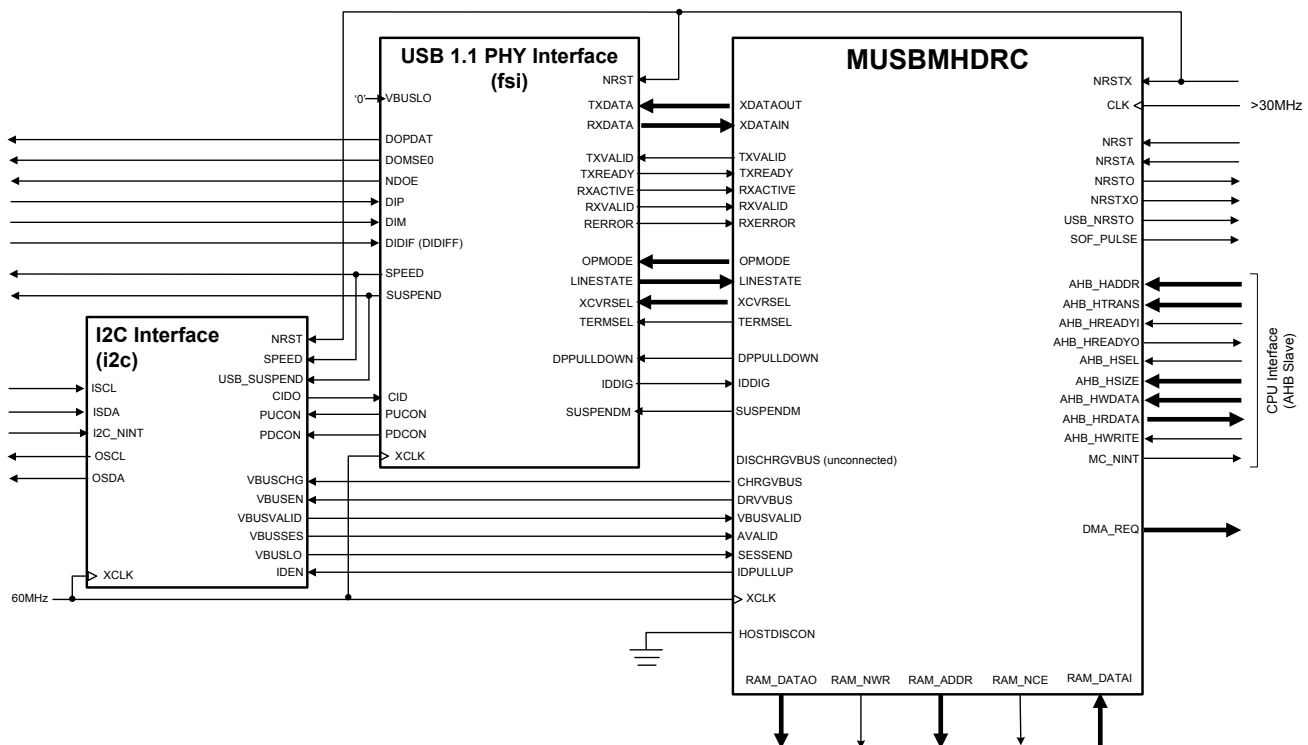
Note: At present only I²C systems with a single I²C master are supported.

CONFIDENTIAL

The signals that using the **musbmhdrc_i2c.v** wrapper introduces into the overall pin-out of the MUSBMHDRC core are listed in the following table.

SIGNAL	TYPE	DESCRIPTION
SUSPEND	Output	This signal goes high when the MUSBMHDRC is in Suspend mode.
SPEED	Output	Transceiver operating speed. 1=full-speed, 0=low-speed.
DOPDAT	Output	Provides either D+ output or DAT output depending on core configuration.
DOMSE0	Output	Provides either D- output or SE0 output depending on core configuration.
NDOE	Output	Output enable for DOP, DOM. Active low.
DIP	Input	D+ single-ended input.
DIM	Input	D- single-ended input.
DIDIF	Input	Differential input.
ISCL	Input	I²C clock input.
ISDA	Input	I²C data input.
I2C_NINT	Input	I²C interrupt (active low).
OSCL	Output	I²C clock output.
OSDA	Output	I²C data output.

Note: The I²C bus interface requires two bi-directional buffers with open collector (or open drain) outputs and Schmitt inputs. The input line of these buffers should be connected to ISDA/ISCL. The output line of these buffers should be connected to OSDA/OSCL such that when OSDA/OSCL are low, the corresponding output buffer is enabled (output low) and when OSDA/OSCL are high, the corresponding output buffer is disabled (output high impedance).

Further information is given about this interface in an Application Note supplied as the file **musbmhdrc_i2c_an.pdf**.

8.2. SOFT CONNECT/DISCONNECT

NOTE: PERIPHERAL MODE ONLY!

The MUSBMHDRC can allow its connection to the USB bus to be controlled by software.

When the MUSBMHDRC is operating in Peripheral Mode, the UTMI+-compliant PHY used alongside the MUSBMHDRC can be switched between normal mode and non-driving mode by setting/clearing bit 6 of the Power register (which is identified as the Soft Conn bit). When this Soft Conn bit is set to 1, the PHY is placed in its normal mode and the D+/D- lines of the USB bus are enabled. When this feature is enabled and the Soft Conn bit is zero, the PHY is put into non-driving mode (OPMODE[1:0] set to 01b) and D+ and D- are tri-stated. The MUSBMHDRC will then appear to have been disconnected to other devices on the USB bus.

After a hardware reset (NRST = 0), Soft Conn is cleared to 0. The MUSBMHDRC will therefore appear disconnected until the software has set Soft Conn 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.

CONFIDENTIAL

8.3. BUS TURN-AROUND TIME CONSIDERATIONS

The bus turn-around time achieved by any implementation of the MUSBMHDRC is the combined result of Rx and Tx delays within the PHY that is used and the SIE Decision Time within the MUSBMHDRC.

The SIE Decision Path for data packets within the MUSBMHDRC is illustrated below:
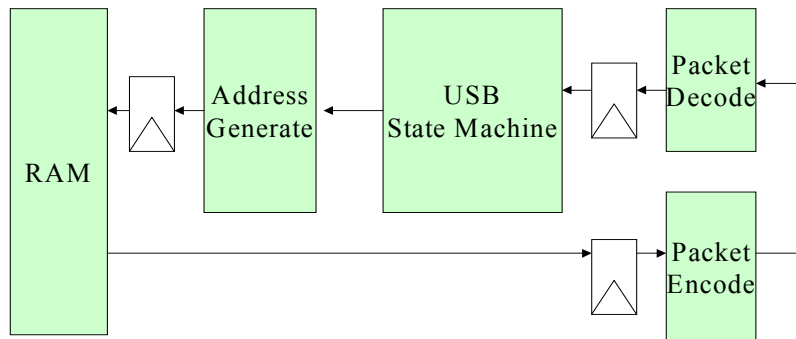
SIE Decision Path for Data Packets

The PHY operates in the XCLK domain while the CPU interface operates in the CLK domain. The data therefore has to be transferred between these two domains. The architecture of the MUSBMHDRC places this transition at the UTM interface. (Placing the transition at the CPU interface would introduce wait-states into CPU data transfers while placing it at any other point in the core would lead to timing problems between different parts of the core.) The sequence of actions contributing to the SIE Decision Time is therefore:

1. Synchronize to CLK

2. Decode packet

3. Decide response

4. Fetch data (if required)

5. Encode packet

6. Synchronize to XCLK

Should the required bus turn-around time prove difficult to achieve, we suggest increasing the CLK speed.

8.3.1.1.1.1. OPERATION AS HOST OR PERIPHERAL

The MUSBMHDRC may be used in a range of different environments. It can be used as either a high-speed or a full-speed 'peripheral' attached to a conventional USB host (such as a PC). It can be used as either host or peripheral in point-to-point data transfers with another 'peripheral' device - or, if the other device also contains a Dual-Role Controller, the two devices can switch roles as required. (This second device may be either a high-speed, full-speed or low-speed USB function.) Or the MUSBMHDRC may be used as the host to a range of such peripheral devices in a 'Multi-point' set-up.

In all cases, Control, Bulk, Isochronous or Interrupt transactions are supported between the MUSBMHDRC and the devices to which it is attached.

CONFIDENTIAL

Whether the MUSBMHDRC expects to behave as a host or as a peripheral depends on the way the devices are cabled together. Each USB cable has an 'A' end and a 'B' end. If the 'A' end of the cable is plugged into the device containing the MUSBMHDRC, the MUSBMHDRC will take the role of the Host device and go into 'Host' mode. (The Host Mode bit (DevCtl.D2) will be set to '1'.) If the 'B' end of the cable is plugged in, the MUSBMHDRC will go instead into 'Peripheral' mode and the Host Mode bit will be set to '0'.

Where the MUSBMHDRC is connected to a single device and that device contains a Dual-Role Controller, signaling may be used to switch the roles of the two devices – and without any need to switch over the cable between the devices. The conditions under which the MUSBMHDRC may switch between a Peripheral role and a Host role are explained in Section 15 Host Negotiation.

Note: The MUSBMHDRC's Multi-Point capability is associated with a range of registers recording the allocation of device functions to individual MUSBMHDRC core endpoints and device function characteristics such as endpoint number, operating speed and transaction type on an endpoint-by-endpoint basis (see Section 5.1 overleaf). Although principally associated with the use of the core as the host to a number of devices, these registers also need to be set when the core is used as the host for a single target device.

8.4. OPERATION AS A PERIPHERAL

When the Host Mode bit (DevCtl.D2) is cleared, the MUSBMHDRC operates in Peripheral mode.

This section looks at the core's actions with regard to Tx endpoints, Rx endpoints, entry into/exit from Suspend mode and recognition of Start of Frame that apply when the MUSBMHDRC is being used as a peripheral.

The conditions under which the MUSBMHDRC operates in Peripheral mode are explained in Section 15: Host Negotiation.

8.4.1. IN TRANSACTION HANDLING AS A PERIPHERAL

When the MUSBMHDRC is operating in Peripheral mode, data for IN transactions is handled through the MUSBMHDRC's Tx FIFOs.

The sizes of the Tx FIFOs for Endpoints 1 to 15 are determined by either by configuration constants in the MUSBMHDRC configuration file or, where dynamic FIFO sizing is selected, through the TxFIFO2 register. The maximum size of data packet that may be placed in a Tx endpoint's FIFO for transmission is programmable and is determined by the value written to the TxMaxP register for that endpoint (maximum payload × number of transactions/microframe (where applicable)).

Except where dynamic FIFO sizing is being used, when the maximum packet size is set to less than, or equal to, half the FIFO size, double packet buffering is enabled for IN transactions and when the maximum packet size is greater than half the FIFO size, single packet buffering is enabled. (Where dynamic FIFO sizing is selected, the use of single or double packet buffering is part of the specification for the endpoint FIFO – see Section 19) When double packet buffering is enabled, two data packets can be buffered in the FIFO: when single packet buffering is enabled, only one packet can be buffered even if the packet is less than half the FIFO size.

Note: The maximum packet size set for any endpoint must not exceed the FIFO size. You should also note that the TxMaxP register should not be written to while there is data is in the FIFO as unexpected results may occur.

8.4.1.1. SINGLE PACKET BUFFERING

If the size of the Tx endpoint FIFO is less than twice the maximum packet size for this endpoint (as set in the TxMaxP register or, where dynamic FIFO sizing is used, in the TxFIFO2 register), only one packet can be buffered in the FIFO and single packet buffering is enabled.

As each packet to be sent is loaded into the Tx FIFO, the TxPktRdy bit in TxCSR needs to be set. If the AutoSet bit in TxCSR is set, the TxPktRdy bit will be automatically set when a maximum-sized packet is loaded into the FIFO. For packet sizes less than the maximum and where AutoSet may not be used (high-bandwidth Isochronous/Interrupt transactions), TxPktRdy will always

CONFIDENTIAL

SOFT CORES

have to be set manually (i.e. by the CPU).

When the TxPktRdy bit is set, either manually or automatically, the packet is deemed ready to be sent. The FIFONotEmpty bit in TxCSR is also set.

When the packet has been successfully sent, both TxPktRdy and FIFONotEmpty will be cleared and the appropriate Tx endpoint interrupt generated (if enabled). The next packet can then be loaded into the FIFO.

8.4.1.2. DOUBLE PACKET BUFFERING

Note: Double packet buffering is disabled if an endpoint's corresponding DPktBufDis bit is asserted (equals 1'b1) in the Tx DPktBufDis register (See 3.8.2 for details). The default setting for this bit is enabled (equals 1'b0).

If the size of the Tx endpoint FIFO is at least twice the maximum packet size for this endpoint (as set in the TxMaxP register or, where dynamic FIFO sizing is used, in the TxFIFO2 register), two packets can be buffered in the FIFO and double packet buffering is enabled.

As each packet to be sent is loaded into the Tx FIFO, the TxPktRdy bit in TxCSR needs to be set. If the AutoSet bit in TxCSR is set, the TxPktRdy bit will be automatically set when a maximum-sized packet is loaded into the FIFO. For packet sizes less than the maximum and where AutoSet may not be used (high-bandwidth Isochronous/Interrupt transactions), TxPktRdy will always have to be set manually (i.e. by the CPU).

When the TxPktRdy bit is set, either manually or automatically, the packet is deemed ready to be sent. The FIFONotEmpty bit in TxCSR is also set.

After the first packet is loaded, TxPktRdy is immediately cleared and an interrupt is generated. A second packet can now be loaded into the Tx FIFO and TxPktRdy set again (either manually or automatically if the packet is the maximum size). Both packets are now ready to be sent.

After each packet has been successfully sent, TxPktRdy will be cleared and the appropriate Tx endpoint interrupt generated (if enabled) to signal that another packet can now be loaded into the Tx FIFO. The state of the FIFONotEmpty bit at this point indicates how many packets may be loaded. If the FIFONotEmpty bit is set then there is another packet in the FIFO and only one more packet can be loaded. If the FIFONotEmpty bit is clear then there are no packets in the FIFO and two more packets can be loaded.

CONFIDENTIAL

8.4.1.3. HIGH BANDWIDTH ISOCHRONOUS/INTERRUPT ENDPOINTS

In High-speed mode, Tx endpoints set up for High-Bandwidth Isochronous/Interrupt transactions can transmit up to three 'USB' packets in any microframe, with a payload of up to 1024 bytes in each packet, corresponding to a data transfer rate of up to 3072 bytes per microframe.

The MUSBMHDRC supports this by allowing the user to load data packets of up to 3072 bytes (i.e. 3 × 1024 bytes) into the associated FIFO in a single transaction. From the point of view of the software in the CPU, the operation is then exactly as described above for Single Packet Buffering or Double Packet Buffering (as appropriate) *except* that TxPktRdy will always need to be set manually (i.e. by the CPU) as AutoSet does not operate with high-bandwidth Isochronous/Interrupt transfers.

Any data packet loaded into the FIFO that is larger than the maximum payload is automatically split into 'USB' packets of the maximum payload, or smaller, for transmission over the USB. The number of USB packets transmitted per microframe and the maximum payload in each packet is defined through the TxMaxP register. Bits 10–0 of the TxMaxP register determine the maximum payload in any USB packet while bits 12,11 determine the maximum number of such packets that can be sent in one microframe (2 or 3). Together, these set the maximum size of packet that can be loaded into the FIFO.

At least one USB packet will always be sent: the number of further USB packets sent in the same microframe will depend on the amount of data loaded into the FIFO. The TxPktRdy bit will be cleared and an interrupt generated only when all the packets have been sent.

Each USB packet is sent in response to an IN token. If, at the end of a microframe, the MUSBMHDRC has not received enough IN tokens to send all the USB packets (e.g. because one of the IN tokens received was corrupted), the remaining data will be flushed from the FIFO. The TxPktRdy bit will then be cleared and the IncompTx bit in the TxCSR register set to indicate that not all of the data loaded into the FIFO was sent.

CONFIDENTIAL

USB

System

FIFO

Block of data for transfer

Single packet of
up to InMaxP
(max. 3072 bytes)
written to FIFO

Single packet of
up to InMaxP
(max. 3072 bytes)
written to FIFO

Remainder

'USB' packets of
Payload bytes
(up to 1024 bytes)
sent over USB

Remainder

'USB' packets of
Payload bytes
(up to 1024 bytes)
sent over USB

Microframe

Microframe

3

2

1

3

2

1

8.4.1.4. OPTIONAL SPECIAL HANDLING

The packets transferred in Bulk operations are defined by the USB Specification to be either 8, 16, 32, 64 or 512 bytes in size, with the 512 byte option only applying to High Speed transfers. For some system designs, however, it may be more convenient for the application software to write larger amounts of data to an endpoint in a single operation than can be transferred in a single USB operation. A particular case in point is where the same endpoint is used for high-speed transfers of 512 bytes under certain circumstances but for full-speed transfers under other circumstances. When operating at full-speed, the maximum amount of data transferred in a single operation is then just 64 bytes.

CONFIDENTIAL

To cater for such circumstances, the MUSBMHDRC includes a 'Tx bulk packet splitting' configuration option which, if selected, allows larger data packets to be written to Bulk Tx endpoints which are then split into packets of an appropriate (specified) size for transfer across the USB bus. (A similar option exists for reading from Bulk Rx endpoints in larger volumes than individual USB packets – see Section 8.4.2.4.)

Under this option, the TxMaxP register for the endpoint is increased to 16 bits and the bottom 11 bits of the register define the *payload* for each individual transfer, while the top 5 bits define a *multiplier*. The application software can then write data packets of size *multiplier × payload* to the FIFO which the MUSBMHDRC will then split into individual packets of the stated payload for transmission over the USB. From the application software's point of view, the resulting operation will not differ from the transmission of a single USB packet except in the size of the packet written.

This facility is offered as a configuration option rather than as a standard feature because it significantly increases the gate count.

Note: This feature is only for use with Bulk endpoints and, in accordance with the USB Specification, the payload must be either 8, 16, 32, 64 or 512 bytes with the 512-byte option only applicable for High-Speed transfers. The payload recorded in the TxMaxP register must also match the *wMaxPacketSize* field of the Standard Endpoint Descriptor for the endpoint. The associated FIFO must also be large enough to accommodate the data packet prior to being split.

8.4.2. OUT TRANSACTION HANDLING AS A PERIPHERAL

When the MUSBMHDRC is operating in Peripheral mode, data for OUT transactions is handled through the MUSBMHDRC's Rx FIFOs.

The sizes of the Rx FIFOs for Endpoints 1 to 15 are determined either by configuration constants in the MUSBMHDRC configuration file or, where dynamic FIFO sizing is selected, through the RxFIFO2 register. The maximum amount of data received by an Rx endpoint in any frame or microframe (in High-speed mode) is programmable and is determined by the value written to the RxMaxP register for that endpoint. (maximum payload × number of transactions/microframe (where applicable)).

Except where dynamic FIFO sizing is being used, when the maximum packet size is set to less than, or equal to, half the FIFO size, double packet buffering is enabled for OUT transactions and when the maximum packet size is greater than half the FIFO size, single packet buffering is enabled. (Where dynamic FIFO sizing is selected, the use of single or double packet buffering is part of the specification for the endpoint FIFO – see Section 19.) When double packet buffering is enabled, two data packets can be buffered in the FIFO: when single packet buffering is enabled, only one packet can be buffered even if the packet is less than half the FIFO size.

Note: The maximum packet size must not exceed the FIFO size.

8.4.2.1. SINGLE PACKET BUFFERING

If the size of the Rx endpoint FIFO is less than twice the maximum packet size for this endpoint (as set in the RxMaxP register or, where dynamic FIFO sizing is used, in the RxFIFO2 register), only one data packet can be buffered in the FIFO and single packet buffering is enabled.

When a packet is received and placed in the Rx FIFO, the RxPktRdy bit (D0) and the FIFOFull bit (D1) in RxCSR are set and the appropriate Rx endpoint is generated (if enabled) to signal that a packet can now be unloaded from the FIFO.

After the packet has been unloaded, the RxPktRdy bit needs to be cleared in order to allow further packets to be received. If the AutoClear bit in RxCSR (D15) is set and a maximum-sized packet is unloaded from the FIFO, the RxPktRdy bit is cleared automatically. The FIFOFull bit is also cleared. For packet sizes less than the maximum, RxPktRdy will always have to be cleared manually (i.e. by the CPU) (with exceptions, see register description).

8.4.2.2. DOUBLE PACKET BUFFERING

Note: Double packet buffering is disabled if an endpoint's corresponding DPktBufDis bit is asserted (equals 1'b1) in the Rx

CONFIDENTIAL

DPktBufDis register (See 3.8.2 for details). The default setting for this bit is enabled (equals 1'b0).

If the size of the Rx endpoint FIFO is at least twice the maximum packet size for the endpoint (as set in the RxMaxP register or, where dynamic FIFO sizing is used, in the RxFIFO2 register), two data packets can be buffered and double packet buffering is enabled.

When the first packet to be received is loaded into the Rx FIFO, the RxPktRdy bit in RxCSR is set and the appropriate Rx endpoint interrupt is generated (if enabled) to signal that a packet can now be unloaded from the FIFO. *Note:* The FIFOFull bit in RxCSR is not set at this point: it is only set if a second packet is received and loaded into the Rx FIFO.

After each packet has been unloaded, RxPktRdy needs to be cleared in order to allow further packets to be received. If the AutoClr bit in RxCSR is set and a maximum-sized packet is unloaded from the FIFO, the RxPktRdy bit will be cleared automatically. For packet sizes less than the maximum, RxPktRdy will always have to be cleared manually (i.e. by the CPU).

If the FIFOFull bit was set to 1 when RxPktRdy is cleared, the MUSBMHDRC will first clear the FIFOFull bit. It will then set RxPktRdy again to indicate that there is another packet waiting in the FIFO to be unloaded.

8.4.2.3. HIGH BANDWIDTH ISOCHRONOUS/INTERRUPT ENDPOINTS

In High-speed mode, Rx endpoints set up for High-Bandwidth Isochronous transactions can receive up to three 'USB' packets in any microframe, with a payload of up to 1024 bytes in each packet, corresponding to a data transfer rate of up to 3072 bytes per microframe. High-Bandwidth Interrupt transactions can similarly be received in Host mode, but note there is no support for high-bandwidth Interrupt transactions in Peripheral mode.

The MUSBMHDRC supports this by automatically combining all the USB packets received during a microframe into a single packet of up to 3072 bytes (i.e. 3 × 1024 bytes) within the Rx FIFO. From the point of view of the software in the CPU, the operation is then exactly as described above for Single Packet Buffering or Double Packet Buffering (as appropriate) *except* that RxPktRdy will always need to be cleared manually (i.e. by the CPU) as AutoClear does not operate with high-bandwidth Isochronous transfers.

The maximum number of USB packets that may be received in any microframe and the maximum payload of these packets are defined through the RxMaxP register. Bits 10–0 of the RxMaxP register determine the maximum payload in any USB packet while bits 12,11 determine the maximum number of these packets that may be received in a microframe (2 or 3).

The number of USB packets sent in any microframe will depend on the amount of data to be transferred, and is indicated through the PIDs used for the individual packets. If the indicated number of packets have not been received by the end of a microframe, the IncompRx bit in the RxCSR register will be set to indicate that the data in the FIFO is incomplete. Equally, if a packet of the wrong data type is received, then the PID Error bit is the RxCSR register will be set. In each case, an interrupt will, however, still be generated to allow the data that has been received to be read from the FIFO.

Note: The circumstances in which a PID Error or IncompRx is reported depends on the precise sequence of packets received. When the core is operating in Peripheral mode, the details are as follows. (A separate set of details applies when the core is operating in Host mode: see Section 8.5.2.)

CONFIDENTIAL

No. pkts expected	Data pkt(s) received	Response	No. pkts expected	Data pkt(s) received	Response
1	DATA0 ('D0')	OK	3	D0	OK
	DATA1 ('D1')	PID Error set		D1	IncompRx set
	DATA2 ('D2')	PID Error set		D2	IncompRx set
	MDATA ('DM')	PID Error set		DM	IncompRx set
2	D0	OK		DM D0	PID Error set
	D1	IncompRx set		DM D1	OK
	D2	IncompRx set + PID Error set		DM D2	IncompRx set
	DM	IncompRx set		DM DM	IncompRx set
	DM D0	PID Error set		DM DM D0	PID Error set
	DM D1	OK		DM DM D1	PID Error set
	DM D2	PID Error set		DM DM D2	OK
	DM DM	PID Error set		DM DM DM	PID Error set

CONFIDENTIAL

USB

FIFO

System

3 | Remainder

2

Two 'USB'
packets of
Payload bytes
(up to 1024 bytes)

1

Microframe

3 | Remainder

Single packet of
up to OutMaxP
(max. 3072 bytes)
read from FIFO

Microframe

2

Two 'USB'
packets of
Payload bytes
(up to 1024 bytes)

1

Single packet of
up to OutMaxP
(max. 3072 bytes)
read from FIFO

8.4.2.4. OPTIONAL SPECIAL HANDLING

The packets transferred in Bulk operations are defined by the USB Specification to be either 8, 16, 32, 64 or 512 bytes in size, with the 512 byte option only applying to High Speed transfers. For some system designs, however, it may be more convenient for the application software to read larger amounts of data from an endpoint in a single operation than can be transferred in a single USB operation. A particular case in point is where the same endpoint is used for high-speed transfers of 512 bytes under certain circumstances but for full-speed transfers under other circumstances. When operating at full-speed, the maximum amount of data transferred in a single operation is then just 64 bytes.

CONFIDENTIAL

SOFT
CORES

To cater for such circumstances, the MUSBMHDRC includes an 'Rx bulk packet combining' configuration option which, if selected, causes the MUSBMHDRC to amalgamate the packets received across the USB bus into larger data packets prior to being read by the application software. (A similar option exists for writing to Bulk Tx endpoints in larger volumes than individual USB packets – see Section 8.4.1.4.)

Under this option, the RxMaxP register for the endpoint is increased to 16 bits and the bottom 11 bits of the register define the *payload* for each individual transfer, while the top 5 bits define a *multiplier*. The MUSBMHDRC will then amalgamate the appropriate number of the USB packets it receives into a single data packet of size *multiplier × payload* within the FIFO before asserting RxPktRdy to alert the application software to the presence of a packet to read in the FIFO. The size of the resulting packet is reported in RxCount. From the application software's point of view, the resulting operation will not differ from the receipt of a single USB packet except in the size of the packet read.

This facility is offered as a configuration option rather than as a standard feature because it increases the gate count.

Note: This feature is only for use with Bulk endpoints and, in accordance with the USB Specification, the payload must be either 8, 16, 32, 64 or 512 bytes with the 512-byte option only applicable for High-Speed transfers. The payload recorded in the RxMaxP register must also match the *wMaxPacketSize* field of the Standard Endpoint Descriptor for the endpoint. The associated FIFO must also be large enough to accommodate the amalgamated data packet.

Note also that the RxPktRdy is only set when either the specified number of packets have been received or a "short" USB packet is received (i.e. a packet of less than the specified payload for the endpoint). If a protocol is being used whereby the endpoint receives bulk transfers that are a multiple of the recorded payload size with no short packet to terminate it, the RxMaxP register should not be programmed to expect more packets than there are in the transfer (otherwise the software will not be interrupted at the end of the transfer).

8.4.3. ADDITIONAL ACTIONS

The MUSBMHDRC core responds automatically to certain conditions on the USB bus or actions by the host. The details are given below:

STALL ISSUED TO CONTROL TRANSFER

The MUSBMHDRC core will automatically issue a STALL handshake to a Control transfer under the following conditions:

1. 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 MUSBMHDRC when the host sends an OUT token (instead of an IN token) after the CPU has unloaded the last OUT packet and set DataEnd.

2. 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 MUSBMHDRC when the host sends an IN token (instead of an OUT token) after the CPU has cleared TxPktRdy and set DataEnd in response to the ACK issued by the host to what should have been the last packet.

3. The host sends more than MaxP data with an OUT data token.

4. The host sends more than a zero length data packet for the OUT Status phase.

ZERO LENGTH OUT DATA PACKETS IN CONTROL TRANSFERS

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 CPU 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 MUSBMHDRC will automatically flush any IN token loaded by CPU ready for the Data phase from the FIFO and set SetupEnd.

CONFIDENTIAL

8.4.4. PERIPHERAL MODE SUSPEND

When no activity has occurred on the USB for 3 ms, the MUSBMHDRC will enter Suspend mode. If the Suspend interrupt has been enabled, an interrupt will be generated at this time. When in Suspend mode, the SUSPENDM output will go low (if enabled): this can be used to put the PHY into suspend mode. In addition the POWERDWN output is asserted: this signal may be used to stop CLK and so save power while in this state. POWERDWN then remains asserted until either power is removed from the bus (indicating that the device has been disconnected) or Resume signaling or Reset signaling is detected on the bus.

When Resume signaling is detected, the MUSBMHDRC will exit Suspend mode and take SUSPENDM high. In response, the PHY should be taken out of suspend. If the Resume interrupt is enabled, an interrupt will be generated. The CPU can also force the MUSBMHDRC to exit Suspend mode by setting the Resume bit in the Power register. When this bit is set, the MUSBMHDRC will exit Suspend mode and drive Resume signaling onto the bus. The CPU should clear this bit after 10 ms (a maximum of 15 ms) to end Resume signaling.

No Resume interrupt is generated when Suspend mode is exited by the CPU.

8.4.5. START-OF-FRAME

When the MUSBMHDRC is operating in Peripheral mode, it should receive a Start-Of-Frame packet from the host once every millisecond when in Full-speed mode, or every 125 microseconds when in High-speed mode.

When the SOF packet is received, the 11-bit frame number contained in the packet is written into the Frame register and an output pulse, lasting one CLK bit period, is generated on SOF_PULSE. A SOF interrupt is also generated (if enabled in the IntrUSBE register).

Once the MUSBMHDRC has started to receive SOF packets, it expects one every millisecond (or every 125 microseconds). If no SOF packet is received after 1.00358 ms (or 125.125 μs), it is assumed that the packet has been lost and an SOF_PULSE (together with a SOF interrupt, if enabled) is still generated though the Frame register is not updated. The MUSBMHDRC will continue to generate an SOF_PULSE every millisecond (or 125 microseconds) and will resynchronize these pulses to the received SOF packets when these packets are successfully received again.

8.5. OPERATION AS A HOST

When the Host Mode bit (DevCtl.D2) is set to '1', the MUSBMHDRC operates as a host either for point-to-point communications with another USB device or, when attached to a hub, for communication with a whole range of devices in a multi-point set-up. High-speed, full-speed and low-speed USB functions are supported, both for point-to-point communication and for operation through a hub. (Where necessary, the core automatically carries out the necessary transaction translation needed to allow a low-speed or full-speed device to be used with a USB 2.0 hub.)

Control, Bulk, Isochronous or Interrupt transactions are supported.

This section looks at the core's actions with regard to Tx endpoints, Rx endpoints, transaction scheduling, entry into/exit from Suspend mode and reset that apply when the MUSBMHDRC is being used as a host. Host mode is automatically selected where the core is connected to a hub. The conditions under which the MUSBMHDRC operates in Host mode for point-to-point operations are explained in Section 15: Host Negotiation.

8.5.1. DEVICE SET-UP FOR MULTIPOINT CONFIGURATION

The following setup requirements apply to the core only if the Multipoint configuration is enabled in the configuration GUI. When the multipoint option is not enabled the following setup should not be executed.

Prior to accessing any device as a host – whether for point-to-point communications or for multi-point communications via a hub – the relevant RxFuncAddr or TxFuncAddr registers need to be set for each used Rx or Tx endpoint to record the function address of the device being accessed. Where a full- or low-speed device is connected to the MUSBMHDRC via a High-speed USB 2.0 hub, details of the hub address and the hub port also need to be recorded in the corresponding RxHubAddr /TxHubAddr and RxHubPort/TxHubPort registers. This allows the core to support split transactions.

CONFIDENTIAL

SOFT CORES

In addition the speed at which the device operates (high, full or low) needs to be recorded in the Type0 (Endpoint 0), TxType or RxType registers for each endpoint that is accessed by the device.

RxFuncAddr, TxFuncAddr, RxHubAddr, TxHubAddr, RxHubPort, TxHubPort are all 7-bit read/write registers. Further information about these registers is given in Section 3.5.2. Information about the Type0, TxType and RxType registers is given in Sections 3.3.4, 3.3.14 and 3.3.16 respectively.

For multi-point communications, it should be noted that the settings in these registers record the *current* allocation of the core's endpoints to the functions associated with the attached devices. To maximize the number of devices supported, the MUSBMHDRC allows this allocation to be changed dynamically – simply by updating the address and speed information recorded in these registers.

Any changes in the allocation of endpoints to device functions need to be made following the completion of any on-going transactions on the endpoints affected.

Further information on allocating endpoints to device functions and switching between different allocations is given in section 11.1.

8.5.2. IN TRANSACTION HANDLING AS A HOST

When the MUSBMHDRC is operating as a host, IN transactions are handled in a similar manner to the way in which OUT transactions are handled when the MUSBMHDRC is operating as a peripheral except that the transaction needs first to be initiated by setting the ReqPkt bit in RxCSR. This indicates to the transaction scheduler that there is an active transaction on this endpoint. The transaction scheduler then sends an IN token to the target function.

When the packet is received and placed in the Rx FIFO, the RxPktRdy bit in RxCSR is set and the appropriate Rx endpoint interrupt is generated (if enabled) to signal that a packet can now be unloaded from the FIFO.

When the packet has been unloaded, RxPktRdy should be cleared. The AutoClear bit in the RxCSR register can be used to have RxPktRdy automatically cleared when a maximum sized packet) (with exceptions, see register description) has been unloaded from the FIFO.

There is also an AutoReq bit in RxCSR which causes the ReqPkt bit to be automatically set when the RxPktRdy bit is cleared. The AutoClear and AutoReq bits can be used with an DMA controller to perform complete Bulk transfers without CPU intervention. Where a known number of MaxP packets is to be transferred, this number should be set in the RqPktCount register associated with the endpoint (see Section 3.8.1). The core decrements the value in the RqPktCount register following each request. When the value decrements from 1 to 0, the AutoReq bit is cleared to prevent any further transactions being attempted. For cases where the size of the transfer is unknown, RqPktCount should be left set to zero. AutoReq will then remain set until cleared by the reception of a short packet (i.e. less than MaxP) such as may occur at the end of a bulk transfer.

If the target function responds to a Bulk/Interrupt IN token with a NAK, the MUSBMHDRC will keep retrying the transaction until any NAK Limit that has been set has been reached. If the target function responds with a STALL, however, the MUSBMHDRC will not retry the transaction but will interrupt the CPU with the RxStall bit in the RxCSR register set. If the target function does not respond to the IN token within the required time (or there was a CRC or bit-stuff error in the packet), the MUSBMHDRC will retry the transaction. If after three attempts the target function has still not responded, the MUSBMHDRC will clear the ReqPkt bit and interrupt the CPU with the Error bit in RxCSR set. *Note:* In the case of high-bandwidth Interrupt transactions, the host will attempt 2 or 3 transactions during a single microframe and generate an interrupt when all packets have been received. If any of these transactions is not ACKed by the target, no further transactions will be attempted during the same microframe.

Note: The number of USB packets sent in any microframe will depend on the amount of data to be transferred, and is indicated through the PIDs used for the individual packets. If the indicated number of packets has not been received by the end of a microframe, the IncompRx bit in the RxCSR register will be set to indicate that the data in the FIFO is incomplete. Equally, if a packet of the wrong data type is received, then the PID Error bit is the RxCSR register will be set. In each case, an interrupt will, however, still be generated to allow the data that has been received to be read from the FIFO.

The circumstances in which a PID Error or IncompRx is reported depends on the precise sequence of packets received. When the core is operating in Peripheral mode, the details are as follows. (A separate set of details applies when the core is operating in

CONFIDENTIAL

SOFT CORES

Peripheral mode: see Section 8.4.2.3.)

No. pkts expected	Data pkt(s) received	Response	No. pkts expected	Data pkt(s) received	Response
1	DATA0 ('D0')	OK	2	D1 DM	PID Error set
	DATA1 ('D1')	PID Error set		D1 NR	IncompRx set
	DATA2 ('D2')	PID Error set		D2 D0	PID Error set
	MDATA ('DM')	PID Error set		D2 D1	PID Error set
	No response ('NR')	IncompRx set*		D2 D2	PID Error set
2	D0	OK		D2 DM	PID Error set
	D1 D0	OK		D2 NR	IncompRx set + PID Error set
	D1 D1	PID Error set		DM	PID Error set
	D1 D2	PID Error set		NR	IncompRx set*
No. pkts expected	Data pkt(s) received	Response	No. pkts expected	Data pkt(s) received	Response
3	D0	OK	3	D2 D2 D1	PID Error set
	D1 D0	OK		D2 D2 D2	PID Error set
	D1 D1	PID Error set		D2 D2 DM	PID Error set
	D1 D2	PID Error set		D2 D2 NR	IncompRx set + PID Error set
	D1 DM	PID Error set			
	D1 NR	IncompRx set		D2 DM D0	PID Error set
	D2 D0	PID Error set		D2 DM D1	PID Error set
	D2 D1 D0	OK		D2 DM D2	PID Error set
	D2 D1 D1	PID Error set		D2 DM DM	PID Error set
	D2 D1 D2	PID Error set		D2 DM NR	IncompRx set + PID Error set
	D2 D1 DM	PID Error set		D2 NR	IncompRx set
	D2 D1 NR	IncompRx set		DM	PID Error set
	D2 D2 D0	PID Error set		NR	IncompRx set*

* In these cases, the interrupt will still be generated but RxPktRdy will not be set as no data will have been placed in the FIFO.

8.5.3. OUT TRANSACTION HANDLING AS A HOST

When the MUSBMHDRC is operating as a host, OUT transactions are handled in a similar manner to the way IN transactions are handled when the MUSBMHDRC is operating as a peripheral.

The TxPktRdy bit in the TxCSR register needs to be set as each packet is loaded into the TxFIFO. Again, setting the AutoSet bit in TxCSR (where applicable) will cause TxPktRdy to be automatically set when a maximum sized packet has been loaded into the FIFO. Furthermore, AutoSet can be used with an DMA controller to perform complete Bulk transfers without CPU intervention.

If the target function responds to the OUT token with a NAK, the MUSBMHDRC will keep retrying the transaction until any NAK Limit that has been set has been reached. If the target function responds with a STALL, however, the MUSBMHDRC will not retry the transaction but will interrupt the CPU with the RxStall bit in the TxCSR register set. If the target function does not respond to the OUT token within the required time (or there was a CRC or bit-stuff error in the packet), the MUSBMHDRC will retry the transaction. If after three attempts the target function has still not responded, the MUSBMHDRC will flush the FIFO

CONFIDENTIAL

and interrupt the CPU with the Error bit in TxCSR set.

8.5.4. TRANSACTION SCHEDULING

When operating as a host, the MUSBMHDRC maintains a frame/microframe counter. If the target function is a full-speed or high-speed device, the MUSBMHDRC will automatically send an SOF/uSOF packet at the start of each frame/microframe. If the target function is a low-speed device, a 'K' state will be transmitted on the bus to act as a "keep-alive" to stop the low-speed device going into Suspend mode.

After the SOF/uSOF packet has been transmitted, the MUSBMHDRC will cycle through all the configured endpoints looking for active transactions. An active transaction is defined as an Rx endpoint for which the ReqPkt bit is set or a Tx endpoint for which the TxPktRdy bit and/or the FIFONotEmpty bit is set.

An active Isochronous or Interrupt transaction will only be started if found on the first transaction scheduler cycle of a frame/microframe and if the interval counter for that endpoint has counted down to zero. This ensures that only one Interrupt/Isochronous transaction will occur per endpoint per n frames/microframes (or up to three if high-bandwidth support is selected) where n is the interval set via the TxInterval/RxInterval register for that endpoint – see Sections 3.3.15 and 3.3.17.

An active Bulk transaction will be started immediately, provided there is sufficient time left in the frame/microframe to complete the transaction before the next SOF/uSOF packet is due. If the transaction needs to be retried (e.g. because a NAK was received or the target function did not respond) then the transaction will not be retried until the transaction scheduler has checked all the other endpoints for active transactions first. This ensures that an endpoint that is sending a lot of NAKs does not block other transactions on the bus. The core also allows the user to specify a limit to the length of time for NAKs may be received from a particular target before the endpoint is timed out (see Sections 3.3.4, 3.3.15 and 3.3.17).

8.5.5. BABBLE

The MUSBMHDRC will not start a transaction until the bus has been inactive for at least the minimum interpacket delay. It will also not start a transaction unless it can be finished before the end of the frame. If the bus is still active at the end of a frame then the MUSBMHDRC will assume that the function it is connected to has malfunctioned and will suspend all transactions and generate a babble interrupt.

8.5.6. HOST MODE SUSPEND

If the SuspendMode bit in the Power register is set, the MUSBMHDRC 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.

To exit Suspend mode, the CPU should set the Resume bit and clear the Suspend bit in the Power register. While the Resume bit is high, the MUSBMHDRC will generate Resume signaling on the bus. After 20 ms, the CPU should clear the Resume bit, at which point the frame counter and transaction scheduler will be started.

While in Suspend mode, the SUSPENDM output will also go low, if enabled: this may be used to power-down the USB drivers. In addition the POWERDWN output is asserted: this signal may be used to stop CLK and so save power while in this state. However, if remote wake-up is to be supported, power to the PHY must be maintained so that the MUSBMHDRC can detect Resume signaling on the bus.

CONFIDENTIAL

9. USB RESET

9.1. IN PERIPHERAL MODE

When the MUSBMHDRC is acting as a peripheral and a reset condition is detected on the USB, the device will perform the following actions:

- Sets FAddr to 0.
- Sets Index to 0.
- Flushes all endpoint FIFOs.
- Clears all control/status registers.
- Enables all endpoint interrupts.
- Generates a Reset interrupt.

If the HS Enab bit in the Power register (D5) was set, the MUSBMHDRC also tries to negotiate for high-speed operation. Whether high-speed operation is selected is indicated by HS Mode bit (Power.D4).

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

9.2. IN HOST MODE

If the Reset bit in the Power register is set while the MUSBMHDRC is in Host mode, the MUSBMHDRC will generate Reset signaling on the bus. If the HS Enab bit in the Power register (D5) was set, it will also try to negotiate for high-speed operation.

The CPU should keep the Reset bit set for at least 20 ms to ensure correct resetting of the target device.

After the CPU has cleared the bit, the MUSBMHDRC will start its frame counter and transaction scheduler. Whether high-speed operation is selected will be indicated by HS Mode bit (Power.D4).

10. SUSPEND/RESUME

How the MUSBMHDRC enters and leaves Suspend mode depends on whether it is currently operating as a host or as a peripheral.

10.1. WHEN THE MUSBMHDRC IS OPERATING AS A PERIPHERAL

(i) Entry into Suspend mode. When operating as a peripheral, the MUSBMHDRC monitors activity on the USB 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. The SUSPENDM output will also go low (if enabled).

At this point, the POWERDWN signal is also asserted to indicate that the application may save power by stopping CLK. POWERDWN then remains asserted until either power is removed from the bus (indicating that the device has been disconnected) or Resume signaling or Reset signaling is detected on the bus.

(ii) When Resume signaling occurs on the bus, first CLK must be restarted if necessary. The MUSBMHDRC will then automatically exit Suspend mode. If the Resume interrupt is enabled, an interrupt will be generated.

(iii) Initiating a Remote Wakeup. If the software wants to initiate a remote wakeup while the MUSBMHDRC is in Suspend mode, it should write to the Power register to set the Resume bit (D2) to '1'. (Note: If CLK has been stopped, it will need to be restarted before this write can occur.) The software should leave then this bit set for approximately 10 ms (minimum of 2 ms, a maximum

CONFIDENTIAL

of 15 ms) before resetting it to 0. By this time the hub should have taken over driving Resume signaling on the USB.

Note: No Resume interrupt will be generated when the software initiates a remote wakeup.

10.2. WHEN THE MUSBMHDRC IS OPERATING AS A HOST

(i) Entry into Suspend mode. When operating as a host, the MUSBMHDRC can be prompted to go into Suspend mode by setting the SuspendMode bit in the Power register. When this bit is set, the MUSBMHDRC 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 Enable SuspendM bit (Power.D0) is set, the UTMI+ PHY will go into low-power mode when the MUSBMHDRC goes into Suspend mode and stop XCLK.

(ii) Sending Resume Signaling. When the application requires the MUSBMHDRC to leave Suspend mode, it needs to clear the Suspend bit in the Power register, set the Resume bit and leave it set for 20ms. While the Resume bit is high, the MUSBMHDRC will generate Resume signaling on the bus. After 20 ms, the CPU should clear the Resume bit, at which point the frame counter and transaction scheduler will be started.

(iii) Responding to Remote Wake-up. If Resume signaling is detected from the target while the MUSBMHDRC is in Suspend mode, the UTMI+ PHY will be brought out of low-power mode and restart XCLK. The MUSBMHDRC will then exit Suspend mode and automatically set the Resume bit in the Power register (D2) to '1' to take over generating the Resume signaling from the target. If the Resume interrupt is enabled, an interrupt will be generated.

11. SUPPORT FOR MULTIPLE DEVICES

This section only applies if MUSBMHDRC is has the multipoint option enabled in the configuration GUI. The MUSBMHDRC has the facility, when operating in Host mode, to act as the host to a range of USB peripheral devices – high-speed, full-speed or low-speed – where these devices are connected to the MUSBMHDRC via a USB hub.

The key feature of the core's support for multiple devices is its facility to allow the functions of the target devices to be individually allocated to the different Rx and Tx endpoints implemented in the MUSBMHDRC core. Furthermore, this allocation can be made dynamically, allowing the devices from the targeted peripheral list to be used in different combinations. The combinations of peripheral devices that may be used together are however limited by the numbers of Tx and Rx endpoints implemented in the core. Further devices can only be added where the endpoints they require remain available.

11.1. ALLOCATING DEVICES TO ENDPOINTS

The separate functions of the connected devices are allocated to the endpoints within the MUSBMHDRC core through a group of three registers, which are associated with each Rx or Tx endpoint implemented in the core (including Endpoint 0).

The registers concerned are Tx/RxFuncAddr, Tx/RxHubAddr and Tx/RxHubPort. (The location of these registers depends on which of the MUSBMHDRC's endpoints is being addressed.)

The information that needs to be recorded in the Tx/RxFuncAddr register is the address of the target function that is to be accessed through the selected endpoint. This information needs to be recorded separately for each Tx and Rx endpoint that is used. In particular, both TxFuncAddr and RxFuncAddr need to be set for Endpoint 0.

The Tx/RxHubAddr and Tx/RxHubPort registers are provided for the case where a full- or low-speed device is connected to the MUSBMHDRC via a high-speed USB 2.0 hub, which carries out the required transaction translation between high-speed transmission and low-/full-speed transmission. Where this is the case, the Tx/RxHubAddr and Tx/RxHubPort registers need to record the address of the hub that carries out the transaction translation and the port of that hub through which the associated Tx/Rx endpoint needs to access the device. *Note:* If Endpoint 0 is connected to a hub, then both the Tx and the Rx versions of

CONFIDENTIAL

these registers need to be set for this endpoint.

The Tx/RxHubAddr register is further used to record whether the hub offers multiple transaction translators or just a single transaction translator. This has a significant effect on the overall bandwidth that can be achieved.

Details of these registers are given in Sections 3.5.2 and 3.5.3.

In addition to recording the address of the target function through these three registers, the endpoint number and operating speed of the target device and the type of transaction that will be executed need to be recorded.

For a Tx endpoint, this information needs to be set in the TxType register located at 1Ah when the Index register is set to select the required endpoint. For an Rx endpoint, this information needs to be set in the RxType register located at 1Ch when the Index register is set to select the required endpoint. In both cases, the endpoint number is recorded in bits D3 – D0, the transaction type is selected through bits D5 – D4, and the operating speed is selected through bits D7 – D6.

In the case of Endpoint 0, just the speed needs to be set (this endpoint only having the facilities to handle Control transactions and therefore always being associated with a device Endpoint 0). This speed setting is made through bits D7 – D6 of the Type 0 register, which is located at 1Ah when the Index register is set to 0.

Details of the Type0, TxType and RxType registers are given in Sections 3.3.4, 3.3.14 and 3.3.16, respectively.

11.2. OPERATION

Once the allocation of functions to endpoints has been made and the operating speed of the target device recorded as described in Section 11.1, most operations in a Multi-point set-up are no different from those for the equivalent actions where the core is attached to a single other device. The details are given elsewhere in this Guide.

However, additional steps are required:

- where the option of dynamically switching the allocation of functions to endpoints is taken (e.g. to allow a wider range of devices to be supported)

- where the control packets normally associated with Endpoint 0 are handled through a different endpoint.

If dynamic allocation is used, it becomes essential for the user to keep track of the current data toggle state associated with the endpoint and with each of the devices that are allocated to that endpoint. Knowledge of this state is necessary to allow the *user* to select the correct data toggle state when the switch is made between one device and other. (This action is the user's responsibility because the core cannot determine what data toggle state is expected when a function is being switched in and out of use.)

The data toggle state can be switched from its current state by writing to the appropriate TxCSRL/RxCSRL register to set the Data Toggle Write Enable and Data Toggle bits that are included in these registers when the core is in Host mode. (See Sections 3.3.9, and 3.3.11)

(*Note:* Data Toggle Write Enable and Data Toggle bits are also included in CSR0 when the core is operating in Host mode. However, Control operations carried out through the core's Endpoint 0 should normally always leave the data toggle in the expected state.)

Where control packets are handled through an endpoint other than Endpoint 0, the user has additionally to prompt for each Setup token to be sent. This involves setting the SetupPkt bit that is included in the TxCSR when the core is operating in Host mode, alongside the TkPktRdy bit. If the SetupPkt bit is not set, an OUT token will be sent.

Overall, the recommendation is to use the core's Endpoint 0 to handle Control packets for all of the devices attached to the core, and to switch the allocation of this endpoint as appropriate. The issue of sending the correct token is then taken care of, as is the issue of ensuring that the data toggle is correctly set for this endpoint.

Using a different endpoint for this function is possible, as described above, but the further points to note are:

CONFIDENTIAL

(i)   the control function must be allocated to an Rx/Tx endpoint pair (i.e. with the same endpoint number).

(ii)   the chosen endpoints must each be associated with FIFOs that can accommodate the packet size associated with EP0 transactions at the chosen operating speed (i.e. a minimum of 8 bytes for Low- or Full-speed transactions but 64 bytes for High-speed transactions

11.3. BANDWIDTH ISSUES

The ability of a multi-point system to cope with isochronous transactions (in particular) is determined by the available bandwidth.

Once an endpoint has been set up, all scheduling is handled in hardware. However, as with PC-based EHCI/OHCI/UHCI hosts, before opening a periodic pipe (for use by isochronous or interrupt traffic), *software* must determine that there is sufficient bandwidth available. Further, if the periodic pipe is opened to a full-speed device through a high-speed hub, software must confirm that sufficient bandwidth is available both on the local high-speed bus and the full-speed bus generated by the transaction translator in the hub.

The bandwidth required for different transactions can be determined using similar algorithms to those used in connection with PC-based hosts (detailed in Section 5.11.3 of the USB 2.0 Specification).

As would be expected, the bandwidth available will be greater where the hub used supports multiple transaction translators.

12. CONNECT/DISCONNECT

The particular behavior related to connecting and disconnecting the MUSBMHDRC concerns its use either in Host mode or Peripheral mode in peer-to-peer communications.

12.1. IN HOST MODE

Where the MUSBMHDRC is operating in Host mode, the CPU starts the session by setting the Session bit (DevCtl.D0). Power is then applied to VBus and the core waits for a device to be connected.

When a device is detected, a Connect interrupt is generated (i.e. IntrUSB.D4 goes high). The speed of the device that has been connected can be determined by reading the DevCtl register where the FSDev bit (D6) will be high for a high-speed/full-speed device and the LSDev bit (D5) will be high for a low-speed device. The CPU should then reset the device. If both FSDev and HS Enab (Power.D5) are set, the MUSBMHDRC will try to negotiate for high-speed operation. Whether this is successful will be indicated by the HS Mode bit (Power.D4).

The CPU should keep the Reset bit set for 20ms to ensure that the target is reset. It can then begin device enumeration.

If the device is disconnected while a session is in progress, a Disconnect interrupt will be generated (i.e. IntrUSB.D5 goes high).

12.2. IN PERIPHERAL MODE

Where the MUSBMHDRC is operating in Peripheral Mode, no interrupt is generated when the device is connected to the host. However a Disconnect interrupt (IntrUSB.D5) is generated when the host terminates a session.

13. PROGRAMMING SCHEME

This and the following sections look at the actions that the device controlling the MUSBMHDRC core will need to perform and at the aspects of the operation of the core that affect this.

CONFIDENTIAL

Throughout this discussion, the controlling device is assumed to be a microcontroller running some firmware but it could be a customized hard-wired logic block.

13.1. SOFT CONNECT/DISCONNECT

If required, the MUSBMHDRC will allow its connection to the USB bus to be controlled by software.

When the Soft Connect/Disconnect is selected, then when the MUSBHHDRC is operating in Peripheral Mode, the UTMI+-compliant PHY used alongside the MUSBMHDRC can be switched between normal mode and non-driving mode by setting/clearing bit 6 of the Power register (which is identified as the Soft Conn bit). When this Soft Conn bit is set to 1, the PHY is placed in its normal mode and the D+/D- lines of the USB bus are enabled. At the same time, the MUSBMHDRC is placed in 'Powered' state, in which it will not respond to any USB signaling except a USB reset.

When this feature is enabled and the Soft Conn bit is zero, the PHY is put into non-driving mode, D+ and D- are tri-stated and the MUSBMHDRC appears to other devices on the USB bus as if it has been disconnected.

After a hardware reset (NRST = 0), Soft Conn is cleared to 0. The MUSBMHDRC will therefore appear disconnected until the software has set Soft Conn 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 Soft Conn bit has been set to 1, the software can also simulate a disconnect by clearing this bit to 0.

13.2. USB INTERRUPT HANDLING

When the CPU is interrupted with a USB interrupt, it needs to read the interrupt status register to determine which endpoint(s) 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.

A flowchart for the USB Interrupt Service Routine is given in the following flowchart.

```
                                    ┌──────────────┐
                                    │   USB ISR    │
                                    └──────┬───────┘
                                           │
                                    ┌──────────────┐
                                    │Read Interrupt│
                                    │  Registers   │
                                    └──────┬───────┘
                                           │
                                       ╱Resume╲      Yes    ┌──────────────┐
                                      ╱interrupt?╲─────────→│RESUME routine│
                                       ╲        ╱           └──────────────┘
                                           │
              ┌──────────────┐         ╱Session Req╲  Yes
              │ SESSION REQ. │←──Yes──╱ interrupt?  ╲
              │   routine    │         ╲           ╱
              └──────────────┘             │
              ┌──────────────┐         ╱Vbus Error╲  Yes
              │ VBUS ERROR   │←──Yes──╱ interrupt? ╲
              │   routine    │         ╲          ╱
              └──────────────┘             │
                                       ╱Suspend ╲    Yes    ┌───────────────┐
                                      ╱interrupt? ╲─────────→│SUSPEND routine│
                                       ╲         ╱          └───────────────┘
                                           │
              ┌──────────────┐         ╱Connect ╲  Yes
              │CONNECT routine│←──Yes──╱interrupt?╲
              └──────────────┘         ╲        ╱
                                           │
                                       ╱Disconnect╲  Yes   ┌──────────────┐
                                      ╱ interrupt? ╲───────→│  DISCONNECT  │
                                       ╲          ╱         │   routine    │
                                           │                └──────────────┘
   ┌──────────────┐   ╱Host ╲          ╱Reset/Babble╲  Yes
   │BABBLE routine│←Yes╱Mode?╲←───Yes─╱  interrupt?  ╲
   └──────────────┘    ╲    ╱          ╲            ╱
                         │
                  ┌──────────────┐  ╱SOF ╲          Yes    ┌───────────┐
                  │Reset routine │ ╱interrupt?╲────────────→│SOF routine│
                  └──────────────┘ ╲        ╱              └───────────┘
                                       │
         ┌────────────────┐        ╱Endpoint 0╲  Yes
         │Endpoint 0 routine│←──Yes╱ interrupt? ╲
         └────────────────┘        ╲          ╱
                                       │
         ┌────────────────┐        ╱Tx Endpoint╲  Yes
         │Tx Endpoint routine│←─Yes╱ interrupt? ╲
         └────────────────┘        ╲           ╱
                                       │
         ┌────────────────┐        ╱Rx Endpoint╲  Yes
         │Rx Endpoint routine│←─Yes╱ interrupt? ╲
         └────────────────┘        ╲           ╱
```

USB Interrupt Service Routine

CONFIDENTIAL

SOFT CORES

14. OTG SESSION REQUEST

In order to conserve power, the *USB On-The-Go* supplement allows VBus to only be powered up when required and to be turned off when the bus is not in use. The MUSBMHDRC further promotes additional power saving through allowing CLK to be stopped when no session is in progress. CLK may be stopped when POWERDWN is asserted.

VBus is always supplied by the 'A' device on the bus. The MUSBMHDRC determines whether it is the 'A' device or the 'B' device by sampling the IDDIG input from the UTMI+ PHY. This signal is pulled low when an 'A-type' plug is sensed (signifying that the MUSBMHDRC is the 'A' device) but taken high when a 'B-type' plug is sensed (signifying that the MUSBMHDRC is the 'B' device).

14.1. STARTING A SESSION

When the device containing the MUSBMHDRC requires starting a session, the CPU needs to set the Session bit in the DevCtl register (D0). The MUSBMHDRC will then enable ID pin sensing. This results in the IDDIG input either being taken low if an A-type connection is detected or high if a B-type connection is detected. The B-Device bit in the DevCtl register (D7) is also set to indicate whether the MUSBMHDRC has adopted the role of the 'A' device or the 'B' device.

If the MUSBMHDRC is the 'A' device: The MUSBMHDRC will then enter Host mode (the 'A' device is always the default host), turn on VBus and wait for VBus to go above the VBus Valid threshold, as indicated by the VBUSVALID input going high. (This event also causes the Vbus[1:0] bits in the DevCtl register (D4 – D3) to go to 11b.)

The MUSBMHDRC will then wait for a peripheral to be connected. When a peripheral is detected, a Connect interrupt (IntrUSB.D4) will be generated (if enabled) and either the FSDev or LSDev bit in the DevCtl register (D6/D5 respectively) will be set depending on whether a high-speed/full-speed peripheral or a low-speed peripheral was detected.

The CPU should then reset this peripheral. If both FSDev and HSEnab (Power.D5) are set, the MUSBMHDRC will monitor LINESTATE during the reset to see if a high-speed chirp is received from the peripheral. If a chirp is received, the MUSBMHDRC will respond with high-speed chirps and enter High-Speed mode.

To end the session, the CPU should clear the Session bit (DevCtl.D0). The MUSBMHDRC will also automatically end the session if babble is detected.

If the MUSBMHDRC is the 'B' device: The MUSBMHDRC will request a session using the Session Request Protocol defined in the *USB On-The-Go* supplement, i.e. it will first assert DISCHRGVBUS to discharge VBus. Then when VBus has gone below the Session End threshold (as indicated by the SESSEND input going high and the VBus[1:0] bits in the DevCtl register (D4 – D3) going to 00b) – and the line state has been SE0 for > 2 ms – the MUSBMHDRC will first pulse the data line, then pulse VBus (by taking CHRGVBUS high).

At the end of the session, the Session bit is cleared – usually by the MUSBMHDRC but it can also be cleared by the CPU if the application software wishes to perform a software disconnect (see the description of DevCtl in Section 3.2.12). The MUSBMHDRC will then switch TERMSEL, which will cause the PHY to switch out the pull-up resistor on D+. This signals the 'A' device to end the session.

14.2. DETECTING ACTIVITY

When the other device of the OTG set-up wishes to a session to start, it will either raise VBus above the Session Valid threshold if it is the 'A' device (as indicated by the AVALID input going high and the VBus[1:0] bits in the DevCtl register (D4 – D3) going to 10b) or, if it is the 'B' device, it will first pulse the data line, then pulse VBus. Depending on which of these actions happens, the MUSBMHDRC can determine whether it is the 'A' device or the 'B' device in the current set-up and act accordingly as follows:

If VBus is raised above the Session Valid threshold: Then the MUSBMHDRC is the 'B' device. The MUSBMHDRC will set

SOFT CORES

the Session bit in the DevCtl register (D0). When Reset signaling is detected on the bus, a Reset interrupt (IntrUSB.D2) will be generated (if enabled) which the CPU should interpret as the start of a session.

The MUSBMHDRC will be in Peripheral mode at this point as the 'B' device is the default peripheral.

At the end of the session, the 'A' device will turn off the power to VBus. When VBus drops below the Session Valid threshold (as indicated by the AVALID input going low and the VBus[1:0] bits in the DevCtl register (D4 – D3) going to 01b), the MUSBMHDRC will detect this and clear the Session bit to indicate that the session has ended. A Disconnect interrupt (IntrUSB.D5) will also be generated (if enabled)

If data line/VBus pulsing is detected: Then the MUSBMHDRC is the 'A' device. It will generate a Session Request interrupt (IntrUSB.D6 – if enabled) to indicate that the 'B' device is requesting a session. The CPU should then start a session by setting the Session bit (DevCtl.D0).

15. HOST NEGOTIATION

When the MUSBMHDRC is the 'A' device (IDDIG low, B-Device (DevCtl.D7) = 0), it will automatically enter Host mode when a session starts.

When the MUSBMHDRC is the 'B' device (IDDIG high, B-Device (DevCtl.D7) = 1), it will automatically enter Peripheral mode when a session starts. The CPU can however request that the MUSBMHDRC becomes the Host by setting the Host Req bit in the DevCtl register (D1). This bit can be set either at the same time as requesting a Session Start by setting the Session bit (DevCtl.D0) or at any time after a session has started. When the MUSBMHDRC next enters Suspend mode (no activity on the bus for 3 ms), then assuming the Host Req bit remains set, it will enter Host mode and begin host negotiation (as specified in the *USB On-The-Go* supplement) by switching TERMSEL, causing the PHY to disconnect the pull-up resistor on the D+ line. This should cause the 'A' device to switch to Peripheral mode and connect its own pull-up resistor. When the MUSBMHDRC detects this, it will generate a Connect interrupt (IntrUSB.D4) if this is enabled. It will also set the Reset bit in the Power register (D3) to begin resetting the 'A' device. (The MUSBMHDRC begins this reset sequence automatically to ensure that reset is started as required within 1 ms of the 'A' device connecting its pull-up resistor). The CPU should wait at least 20 ms, then clear the Reset bit and enumerate the 'A' device.

When the MUSBMHDRC-based 'B' device has finished using the bus, the CPU should put it into Suspend mode by setting the Suspend Mode bit in the Power register (D1). The 'A' device should detect this and either terminate the session or revert to Host mode. If the 'A' device is MUSBMHDRC-based, it will generate a Disconnect interrupt (IntrUSB.D5) if this is enabled.

16. FUNDAMENTAL DMA SUPPORT

The MUSBMHDRC supports DMA access to the FIFOs for Tx Endpoints 1 – 15 and Rx Endpoints 1 – 15 either by the built-in DMA controller (if implemented) or by an external DMA controller.

Underlying the support for either the built-in DMA controller or an external DMA controller is a separate DMA request line for each Tx endpoint and each Rx endpoint. If a total of N Tx Endpoints and M Rx Endpoints are defined (in addition to Endpoint 0), DMA_REQ[0] ... DMA_REQ[N–1] are associated with Tx Endpoints 1 ... N; DMA_REQ[N] ... DMA_REQ[$N+M$–1] are associated with Rx Endpoints 1 ... M. These request lines are handled internally where the built-in DMA controller is used but are also made available at the top-level of the core to allow their use by an external DMA controller.

The DMA request lines are individually enabled through the DMAReqEnab bit in the appropriate TxCSR control register (D12) or RxCSR control register (D13) and operate in two modes, referred to as DMA Request Mode 0 and DMA Request Mode 1. The choice of operating mode is made through the DMAReqMode bit (TxCSR.D10 for Tx endpoints; RxCSR.D11 for Rx endpoints). The required Request Mode needs to be selected both where using an external DMA controller and where using the built-in DMA controller.

For Rx endpoints operating in Request Mode 0, the DMA request line goes high when a data packet is available in the endpoint

CONFIDENTIAL

FIFO and normally goes low at the end of the cycle in which the 8th from last byte starts to be processed (which happens two transfers minus one CLK cycle in advance of the transfer containing this byte). If however an external DMA controller is being used and the Early DMA Assert option is not taken (see Section 3.2.13), the request line will go low at the end of the cycle in which the end of the transfer is identified (which happens one CLK cycle after the penultimate transfer). The request line will also go low if the CPU clears the RxPktRdy bit (RxCSRL.D0).

The behavior of the DMA request lines for Rx endpoints in Request Mode 1 is similar except the request line will only go high when the packet received is of the maximum packet size (as set in the RxMaxP register). If the packet received is of some other size, the DMA request line will stay low. Note, however, that if the Request Mode is switched from Request Mode 1 to Request Mode 0, the request line will be asserted if there is a packet in the FIFO in order to allow this 'pre-received' packet to be downloaded.

For Tx endpoints operating in either Request Mode 0 or Request Mode 1, the DMA request line will go high when the endpoint FIFO is able to accept a data packet. It will normally go low one CLK cycle after the 8th from last of TxMaxP bytes have been loaded into the FIFO, but if an external DMA controller is being used and the Early DMA Assert option is not taken (see Section 3.2.13), the request line will instead go low one CLK cycle after all TxMaxP bytes have been loaded. The request line will also go low if the CPU sets the TxPktRdy bit (TxCSRL.D0).

Note: When operating in Host mode, if either the RxStall bit (TxCSRL.D5) or the Error bit (TxCSRL.D2) becomes set following three failed attempts to transmit a packet, the DMA request line will be disabled until the RxStall/Error bit has been cleared.

The mode selected also affects the generation of Endpoint interrupts (if enabled). In DMA Request Mode 0, no interrupt is generated when packets are received but the appropriate Endpoint interrupt is generated to prompt the loading of all packets. In DMA Request Mode 1, the Endpoint interrupt is suppressed except following the receipt of a short packet (i.e. one of less than RxMaxP bytes).

The conditions under which Tx and Rx Endpoint interrupts are generated are summarized in the following tables.

EPInterrupt associated with RxPktRdy being set		
DMAReqEnab	*DMAReqMode*	*Interrupt generated?*
0	X	Yes
1	0	No
1	1	Only if short packet

EP Interrupt associated with TxPktRdy being cleared		
DMAReqEnab	*DMAReqMode*	*Interrupt generated?*
0	X	Yes
1	0	Yes
1	1	No

DMA Request Mode 0 can be used equally well for Bulk, Interrupt or Isochronous transfers. Indeed, if the endpoint is configured for Isochronous transfers, DMA Request Mode 0 should always be selected where DMA is used.

DMA Request Mode 1 is chiefly valuable where large blocks of data are transferred to a Bulk endpoint. The USB protocol requires such packets to be split into a series of packets of the maximum packet size for the endpoint (512 bytes for high speed, 64 bytes for full speed). Note that Tx/RxMaxP must be set to an even number of bytes for proper interrupt generation. DMA Request Mode 1 can be used, with a suitably programmed DMA controller, to avoid the overhead of having to interrupt the processor after each individual packet: instead the processor is only interrupted after the transfer has completed. In some cases, the block of data transferred will comprise a pre-defined number of these packets, that the controlling software 'counts' through the transfer process. In other cases, the last packet in the series may be less than the maximum packet size and the receiver may use this 'short' packet to signal the end of the transfer. (If the total size of the transfer is an exact multiple of the maximum packet size, the transmitting software should send a null packet for the receiver to detect.)

Note: Tx/RxMaxP must be set to an even number of bytes for proper interrupt generation in Mode 1.

Further information on using DMA for Bulk transfers is given in Section 22.3.

DMA transfers may be 8-bit, 16-bit, or 32-bit as required. However, all the transfers associated with one packet (with the exception of the last) must be of the same width so that the data is consistently byte-, word- or double-word-aligned. The last transfer may contain fewer bytes than the previous transfers in order to complete an odd-byte or odd-word transfer.

Note: DMA Requests should be disabled before the DMA Request Mode is changed. In particular, the DMAReqMode bit in the TxCSRH register should not be set to zero either before or in the same cycle as the corresponding DMAReqEnab bit is cleared to

CONFIDENTIAL

zero.

17. OPTIONAL DMA CONTROLLER

The MUSBMHDRC may optionally include a multi-channel DMA controller, configurable for up to 8 channels.

This DMA controller supports two DMA modes, referred to as DMA Modes 0 and 1 and it can handle packet sizes up to 8k (for use in conjunction with the Tx bulk packet splitting, Rx bulk packet combining options described in Sections 8.4.1.4 and 8.4.2.4) In addition, the controller can be programmed to conduct transfers using INCR4, INCR8 and INCR16 4/8/16-beat incrementing bursts rather than bursts of unspecified length.

When operating in DMA Mode 0, the DMA controller can be only programmed to load/unload one packet, so processor intervention is required for each packet transferred over the USB. This mode can be used with any endpoint, whether it uses Control, Bulk, Isochronous, or Interrupt transactions (i.e. including Endpoint 0).

When operating in DMA Mode 1, the DMA controller can be programmed to load/unload a complete bulk transfer (which can be many packets). Once set up, the DMA controller will load/unload all packets of the transfer, interrupting the processor only when the transfer has completed. DMA Mode 1 can only be used with endpoints that use Bulk transactions.

Each channel can be independently programmed for the selected operating mode.

17.1. DMA REGISTERS

The DMA controller has one interrupt register which indicates which channels have a pending interrupt, and a set of three control registers for each configured channel. Full descriptions of the DMA registers is provided in section Register Description section 3.8.5.

Address*	Register	Description
200h	DMA_INTR	DMA Interupt register.
204h+ (n-1)*10h	DMA_CNTL	DMA Control Register.
208h + (n-1)*10h	DMA_ADDR	DMA Address Register.
20Ch + (n-1)*10h	DMA_COUNT	DMA Count Register.

*n = channel number 1 thru 8

17.2. DMA BUS CYCLES

The DMA controller uses incrementing bursts on the AHB. It starts a new burst when it is first granted bus mastership (whether at the start of a USB packet or when regaining the bus after being thrown off part way through a packet), and when the AHB address starts a new 1K byte block. *Note:* The requirement to start a new burst at 1K boundaries is handled automatically by the DMA controller. The user does not need to take these boundaries into account in programming the DMA controller.

These bursts may be either 4-beat, 8-beat, 16-beat or of unspecified length, according to how the DMA channel is programmed, the size of packet being transferred and the location relative to the next 1K boundary. Bits D10–9 of the CNTL register select Burst Mode 0 – 3 which in turn define which burst types may be used (see Section 17.1 above). For example, selection of Burst Mode 2 allows use of 8-beat (INCR8), 4-beat (INCR4) bursts and bursts of unspecified length but not 16-beat (INCR16) bursts.

CONFIDENTIAL

There is no restriction on the Burst Mode selected for transfers in either DMA Mode 0 or DMA Mode 1.

Each transfer of a packet is generally carried out using word transfers (32 bits) but there may be additional byte or half-word transfers at the end of the packet transfer. Since the the start address (written to the DMA ADDR (*n*) register) must be word aligned, the packet transfer will start with a word transfer, but half-word and/or byte transfers may be added at the end to handle any residue.   AHB Split transactions and retries are supported.

17.3. BUS ERRORS

If a bus error occurs while the DMA controller is accessing memory on the AHB, the DMA controller will immediately terminate the DMA transfer and interrupt the processor with the Bus Error (D8) bit of the CNTL register set.

Note: The generation of this interrupt is _not_ affected by the setting of the Interrupt Enable bit in the DMA CNTL register (bit 3). The interrupt will still be generated when CNTL.D3 = 0.

17.4. TRANSFERRING PACKETS

Use of the built-in DMA controller to access the MUSBMHDRC FIFOs requires both the DMA controller and the MUSBMHDRC endpoint to be appropriately programmed. Many variations are possible. The following sections detail the standard set-ups used for the basic actions of transferring individual packets and multiple packets.

17.4.1. INDIVIDUAL PACKET: RX ENDPOINT

The transfer of individual packets will normally be carried out using DMA Mode 0.

For this, the MUSBMHDRC Rx endpoint should be programmed as follows:

The relevant interrupt enable bit in the IntrRxE register set to 1.

The DMAReqEnab bit (D13) of the appropriate RxCSR register set to 0. (**Note:** There is no need to set the MUSBMHDRC to support DMA for this operation.)

When a packet has been received by the MUSBMHDRC, it will generate the appropriate Endpoint interrupt. The processor should then program selected channel of the DMA controller as follows:

ADDR :   Memory address to store packet

COUNT :   Size of packet (determined by reading the MUSBMHDRC RxCount register)

CNTL :   DMA Enable (D0) =1; Direction (D1) =0; DMA Mode (D2) =0; Interrupt Enable (D3) =1;
Required Burst Mode (D10–9).

The DMA controller will then request bus mastership and transfer the packet to memory. When it has completed the transfer, it will generate a DMA interrupt (DMA_NINT taken low). The processor should then clear the RxPktRdy bit in the MUSBMHDRC RxCSR register.

17.4.2. INDIVIDUAL PACKET: TX ENDPOINT

To carry out this operation using DMA Mode 0, an MUSBMHDRC Tx endpoint should be programmed as follows:

The relevant interrupt enable bit in the IntrTxE register set to 1.

The DMAReqEnab bit (D12) of the appropriate TxCSR register set to 0. (**Note:** There is no need to set the MUSBMHDRC to support DMA for this operation.)

When the FIFO in the MUSBMHDRC becomes available, the MUSBMHDRC will interrupt the processor with the appropriate Tx Endpoint interrupt. The processor should then program the DMA controller as follows:

CONFIDENTIAL

ADDR :   Memory address of packet to send

COUNT :   Size of packet to be sent

CNTL :   DMA Enable (D0) =1; Direction (D1) =1; DMA Mode (D2) =0; Interrupt Enable (D3) =1; Required Burst Mode (D10–9).

The DMA controller will then request bus mastership and transfer the packet to the MUSBMHDRC FIFO. When it has completed the transfer, it will generate a DMA interrupt. The processor should then set the TxPktRdy bit in the MUSBMHDRC TxCSR register.

17.4.3. MULTIPLE PACKETS: RX ENDPOINT

The transfer of multiple packets will normally be carried out using DMA Mode 1.

Where multiple packets are to be received using DMA Mode 1, the DMA Controller should be programmed as follows:

ADDR :   Memory address of the buffer in which to store transfer

COUNT :   Maximum size of data buffer

CNTL :   DMA Enable (D0) =1; Direction (D1) =0; DMA Mode (D2) =1; Interrupt Enable (D3) =1; Required Burst Mode (D10–9).

and the MUSBMHDRC Rx endpoint should be programmed as follows:

The relevant interrupt enable bit in the IntrRxE register should be set to 1.

The AutoClear (D15), DMAReqEnab (D13) and DMAReqMode (D11) bits of the appropriate RxCSR register should be set to 1. In Host mode, the AutoReq (D14) bit should also be set to 1 and the RqPktCount register should be programmed with the number of packets in the transfer.

As each packet is received by the MUSBMHDRC, the DMA controller will request bus mastership and transfer the packet to memory. With AutoClear set, the MUSBMHDRC will automatically clear the RxPktRdy bit.

In Peripheral mode or where RqPktCount is zero, this process will continue automatically until the MUSBMHDRC receives a 'short packet' (one of less than the maximum packet size for the endpoint) signifying the end of the transfer. This 'short packet' will not be transferred by the DMA controller: instead the MUSBMHDRC will interrupt the processor by generating the appropriate Endpoint interrupt. The processor can then read the MUSBMHDRC RxCount register to see the size of the 'short packet' and either unload it manually or reprogram the DMA controller in Mode 0 to unload the packet.

In Host mode with AutoReq set and RqPktCount non-zero, the core will decrement the value in the RqPktCount register following each request. When the value decrements from 1 to 0, the AutoReq bit is cleared to prevent any further transactions being attempted.

The DMA controller ADDR register will have been incremented as the packets were unloaded so the processor can determine the size of the transfer by comparing the current value of ADDR against the start address of the memory buffer.

Note: If the size of the transfer exceeds the data buffer size, the DMA controller will stop unloading the FIFO and interrupt the processor via the DMA_NINT line.

17.4.4. MULTIPLE PACKETS: TX ENDPOINT

To carry out this operation using DMA Mode 1, the DMA controller should be programmed as follows:

ADDR :   Memory address of data block to send

COUNT :   Size of data block

CNTL :   DMA Enable (D0) =1; Direction (D1) =1; DMA Mode (D2) =1; Interrupt Enable (D3) =1; Required Burst Mode (D10–9).

CONFIDENTIAL

and the MUSBMHDRC Tx endpoint should be programmed as follows:

The relevant interrupt enable bit in the IntrTxE register should be set to 1 (simply so that errors can be detected).

The AutoSet (D15), DMAReqEnab (D12) and the DMAReqMode (D10) bits of the appropriate TxCSR register should be set to 1.

When the FIFO in the MUSBMHDRC becomes available, the DMA controller will request bus mastership and transfer a packet to the FIFO. With AutoSet set, the MUSBMHDRC will automatically set the TxPktRdy bit. This process will continue until the entire data block has been transferred to the MUSBMHDRC. The DMA controller will then interrupt the processor by taking DMA_NINT low. If the last packet to be loaded was less than the maximum packet size for the endpoint, the TxPktRdy bit will not have been set for this packet: the processor should therefore respond to the DMA interrupt by setting the TxPktRdy bit to allow the last 'short packet' to be sent. If the last packet to be loaded was of the maximum packet size, then the action to take depends on whether the transfer is under the control of an application such as the mass storage software on a Windows system that keeps count of the individual packets sent. If the transfer isn't under such control, the processor should still respond to the DMA interrupt by setting the TxPktRdy bit. This has the effect of sending a null packet for the receiving software to interpret as indicating the end of the transfer.

18. VBUS EVENTS

The USB On-The-Go specification defines a series of thresholds to which the devices involved in point-to-point communications are required to respond:

- VBus Valid (required to be greater than 4.4 and 4.75VV)

- Session Valid for 'A' device (required to be between 0.8V and 2.1V)

- Session End (required to be between 0.2V and 0.8V)

 (The actual thresholds used in a particular device are set through a series of comparators, external to the MUSBMHDRC core, which take the corresponding VBUSVALID, AVALID and SESSEND inputs high or low depending on the level of VBus.)

Which of these thresholds are critical and the way in which the CPU controlling the MUSBMHDRC needs to respond depends on whether the device is the 'A' device or the 'B' device and the circumstances under which the event happens. The required actions are summarized below.

18.1.1.1. ACTIONS AS AN 'A' DEVICE

VBus > VBus Valid with session initiated by MUSBMHDRC (i.e. Vbus[1:0] (DevCtl.[D4:D3]) = 11b, Session bit (DevCtl.D0) set). When VBus becomes greater than VBus Valid, the MUSBMHDRC selects Host Mode and waits for a device to be connected. It then generates a Connect interrupt (IntrUSB.D4). The CPU should reset and enumerate the connected 'B' device.

VBus > Session Valid with session initiated by 'B' device (i.e. Vbus[1:0] (DevCtl.[D4:D3]) = 10b, Session bit (DevCtl.D0) clear). When VBus becomes greater than Session Valid, the MUSBMHDRC will generate a Session Request interrupt (IntrUSB.D6). The CPU should set the Session bit. The MUSBMHDRC will then either stay in Host mode or change to Peripheral mode depending on the state of the pull-up resistor on the 'B' device. The selected mode will be indicated by the state of the Host Mode bit (DevCtl.D2).

VBus below VBus Valid while the Session bit remains set (i.e. Vbus[1:0] (DevCtl.[D4:D3]) ≠ 11b, Session bit (DevCtl.D0) set)**.** This indicates a problem with the VBus power level. For example, the battery power may have dropped too low to sustain VBus Valid. Alternatively, the 'B' device may be drawing more current than the 'A' device can provide. In either case, the MUSBMHDRC will automatically terminate the session and generate a VBus Error interrupt (IntrUSB.D7).

18.1.1.2. ACTIONS AS AN 'B' DEVICE

VBus > Session Valid (i.e. Vbus[1:0] (DevCtl.[D4:D3]) = 10b, Session bit (DevCtl.D0) clear). This indicates activity from the 'A' device. The MUSBMHDRC will set the Session bit and take the DPPULLDOWN output low in order to disconnect the pull-down resistor on the D+ line.

VBus < Session Valid while the Session bit remains set (i.e. Vbus[1:0] (DevCtl.[D4:D3]) = 01b, Session bit (DevCtl.D0) set). This indicates that the 'A' device has lost power (or become disconnected). The MUSBMHDRC will clear the Session bit (DevCtl.D0) and generate a Disconnect interrupt (IntrUSB.D5). The CPU should end the session.

VBus < Session End (i.e. Vbus[1:0] (DevCtl.[D4:D3]) = 00b). This is the condition under which a 'B' device can initiate a session request. If the Session bit (DevCtl.D0) is set, then after 2ms of SE0 on the bus, the MUSBMHDRC will start SRP by first pulsing the data line, then pulsing VBus (by taking CHRGVBUS high).

19. DYNAMIC FIFO SIZING

If needed, the MUSBMHDRC can be configured to have a single overall FIFO size of 128, 256, 512, 1K … 64K bytes, areas of which may then be allocated to the different endpoints when the MUSBMHDRC is initialized. (See Section 3.3.18)

Note: You are strongly advised to only use this feature where the MUSBMHDRC is used in a device that requires different FIFO sizes in different contexts. If the FIFO sizes don't need to change, it is better to set these sizes through the standard configuration options as the option of dynamic FIFO sizes significantly increases the size of the core and requires more complex firmware to handle it.

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 last two together define the amount of space that needs to be allocated to the FIFO.)

These details may be specified through the following four registers, which are added to the Indexed area of the MUSBMHDRC register map when the option of Dynamic FIFO sizing is selected:

MUSBMHDRC REGISTER MAP		
ADDR	**NAME**	**DESCRIPTION**
62	TxFIFOsz	Tx Endpoint FIFO size
63	RxFIFOsz	Rx Endpoint FIFO size
64,65	TxFIFOadd	Tx Endpoint FIFO address
66,67	RxFIFOadd	Rx Endpoint FIFO address

Details of these registers are given below are provided in the register description section 3.10.

Note: (i) The option of setting FIFO sizes dynamically only applies to Endpoints 1…15. The Endpoint 0 FIFO has a fixed size (64 bytes) and a fixed location (start address 0).
(ii) It is the responsibility of the firmware (and the system designer) 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 them that is at least as large as the maximum packet size set for the endpoint.

CONFIDENTIAL

20. TIMING WAVEFORMS

The following timing diagrams are given as a guide to when inputs to the MUSBMHDRC must be valid, and to when outputs are valid. The actual set-up and delay times will also depend on the technology and layout used for the MUSBMHDRC. **Note:** Timings related to any alternative bridge that is used are to be found in the associated bridge specification included in the **musbmhdrc/docs** directory.

20.1. CPU READ

READ TIMING (AHB-compatible interface)

20.2. CPU WRITE

WRITE TIMING (AHB-compatible interface)

CONFIDENTIAL

20.3. RAM WRITE

RAM WRITE

| CLK |
| RAM_NCE |
| RAM_NWR |
| RAM_ADDR | A_1 | A_2 | A_3 |
| RAM_DATAO | D_1 | D_2 | D_3 |

CONFIDENTIAL

SOFT CORES

RAM READ

Signal	Waveform
CLK	
RAM_NCE	
RAM_NWR	
RAM_ADDR	A_1 A_2 A_3
RAM_DATAI	D_1 D_2 D_3

CONFIDENTIAL
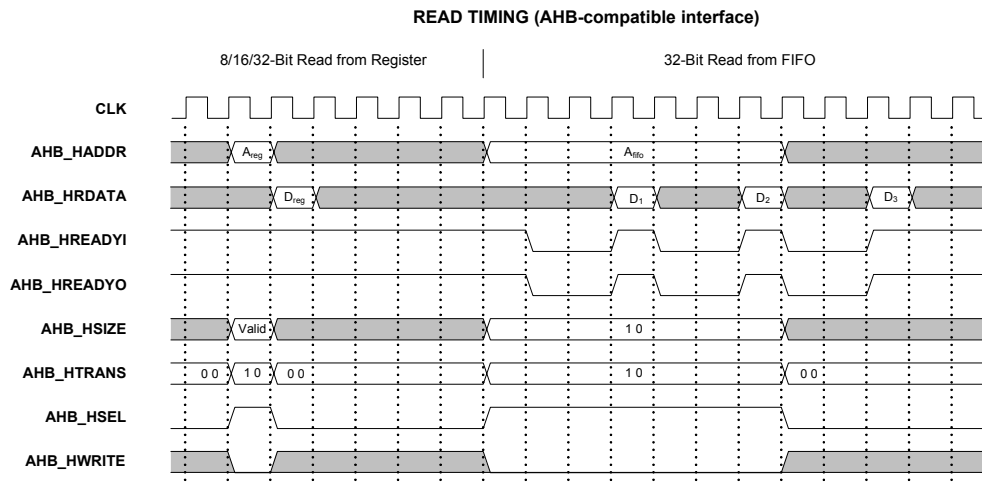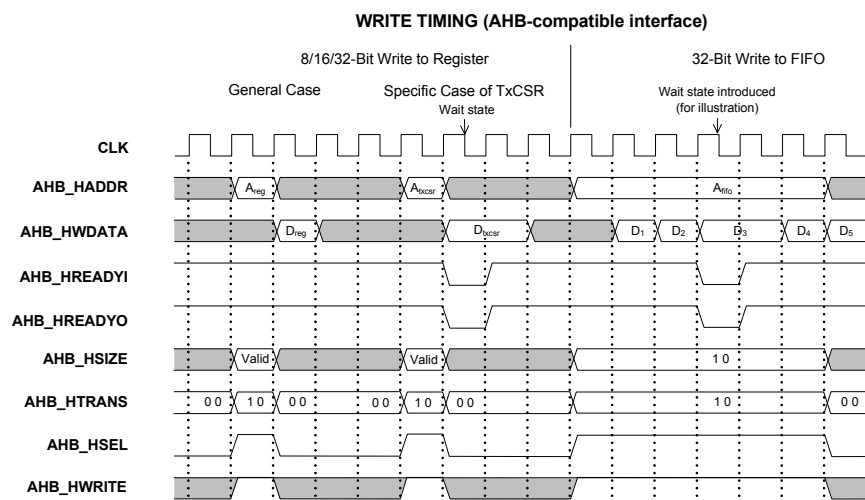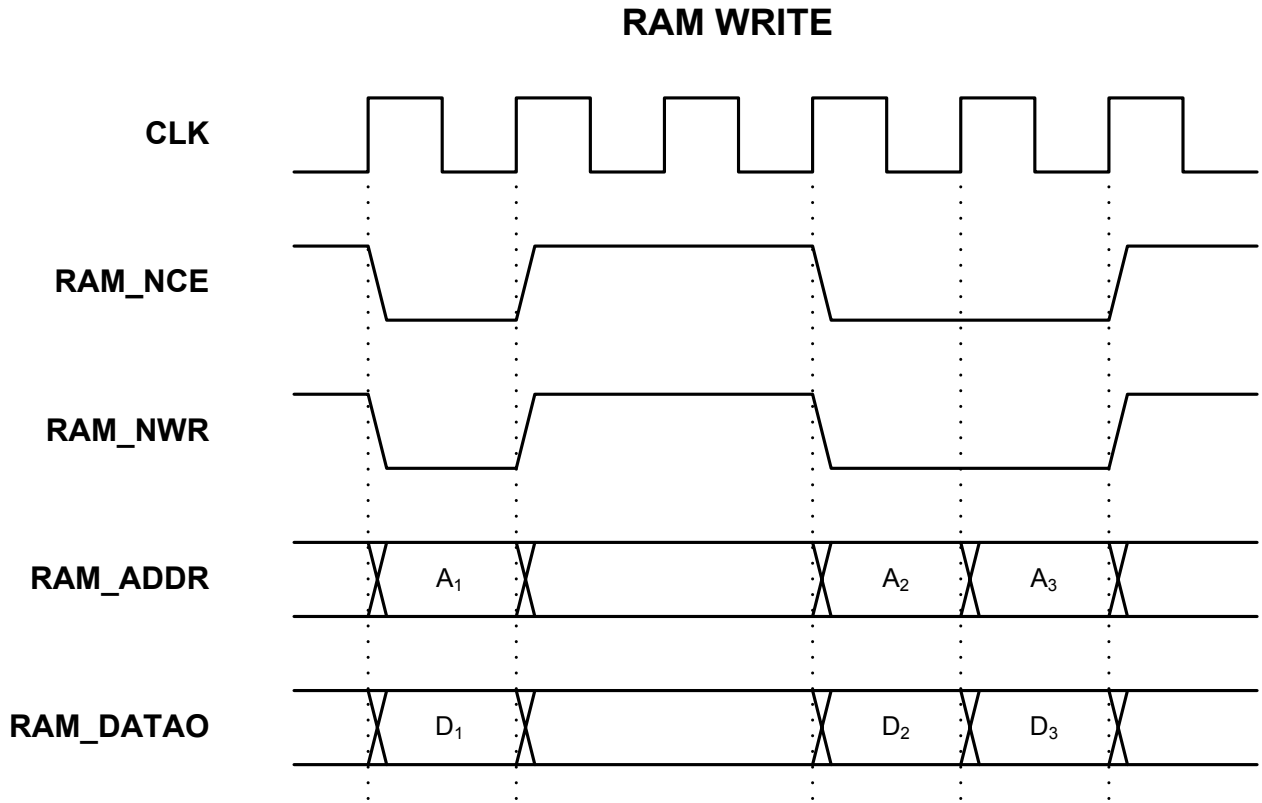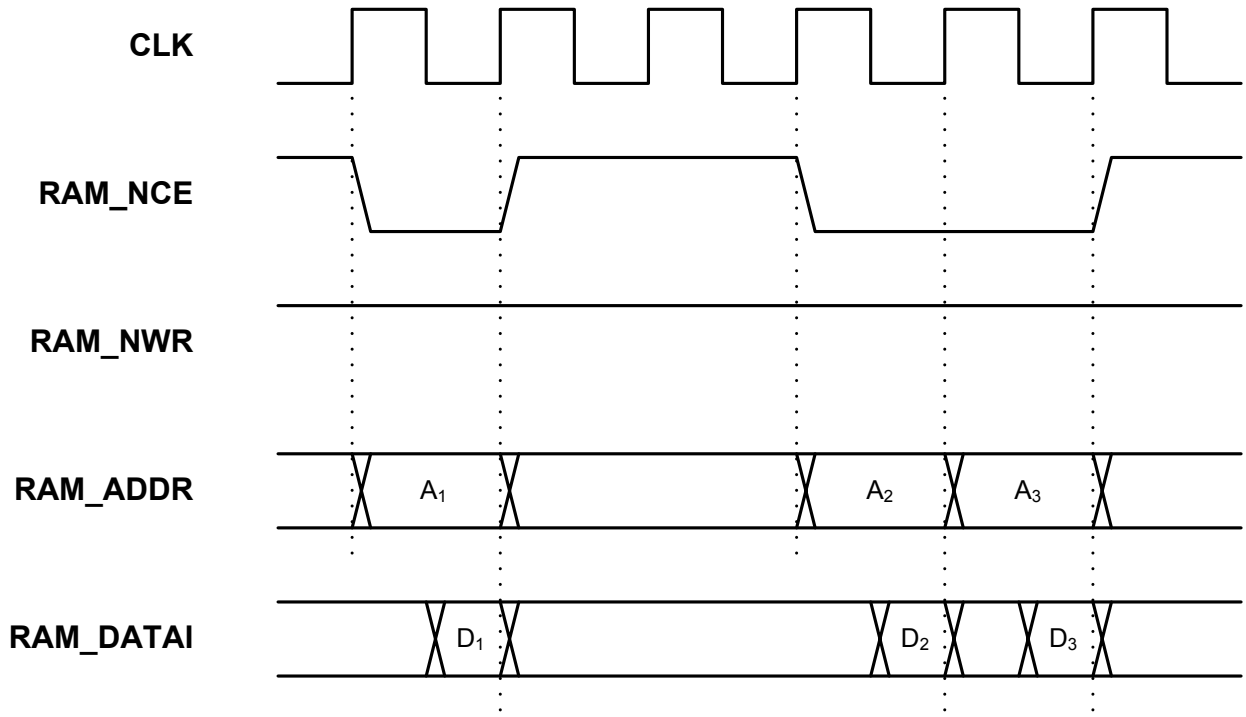
20.5. DMA TIMINGS

20.5.1. BUILT-IN DMA CONTROLLER

DMA LOAD TO Tx ENDPOINT (32 bytes)

CLK (=HCLK)										
AHB_HBUSREQ										
AHB_HGRANT										
AHB_HTRANSM	00	10	11	11	11	11	11	11	11	00
AHB_HADDRM		A_0	A_4	A_8	A_{12}	A_{16}	A_{20}	A_{24}	A_{28}	
AHB_HWRITEM										
AHB_HSIZEM		10								
AHB_HBURSTM		001								
AHB_HRDATAM		D_0	D_4	D_8	D_{12}	D_{16}	D_{20}	D_{24}	D_{28}	
AHB_HREADYMI										

DMA UNLOAD FROM Rx ENDPOINT (32 bytes)

CLK (=HCLK)										
AHB_HBUSREQ										
AHB_HGRANT										
AHB_HTRANSM	00	10	11	11	11	11	11	11	11	00
AHB_HADDRM		A_0	A_4	A_8	A_{12}	A_{16}	A_{20}	A_{24}	A_{28}	
AHB_HWRITEM										
AHB_HSIZEM		10								
AHB_HBURSTM		001								
AHB_HWDATAM		D_0	D_4	D_8	D_{12}	D_{16}	D_{20}	D_{24}	D_{28}	
AHB_HREADYMI										

CONFIDENTIAL
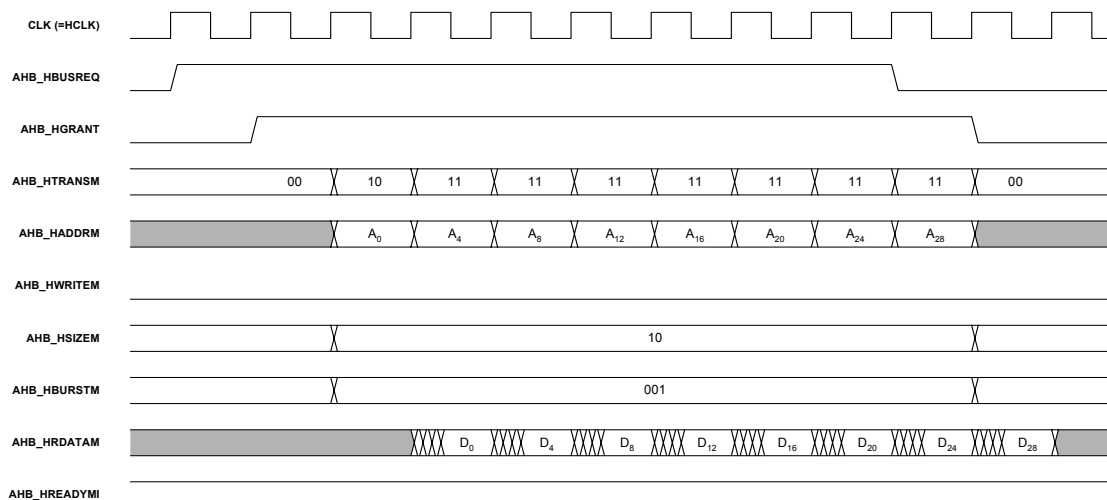
20.5.2. EXTERNAL DMA CONTROLLER INTERFACE

DMA LOAD TO Tx ENDPOINT (32 bytes)

DMA UNLOAD FROM Rx ENDPOINT (32 bytes)

CONFIDENTIAL
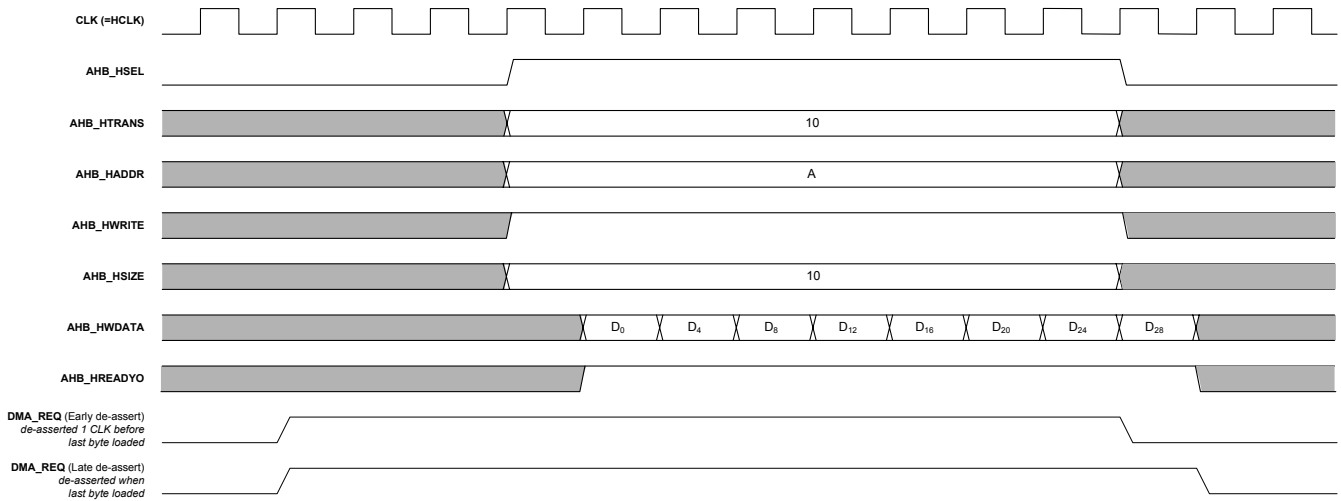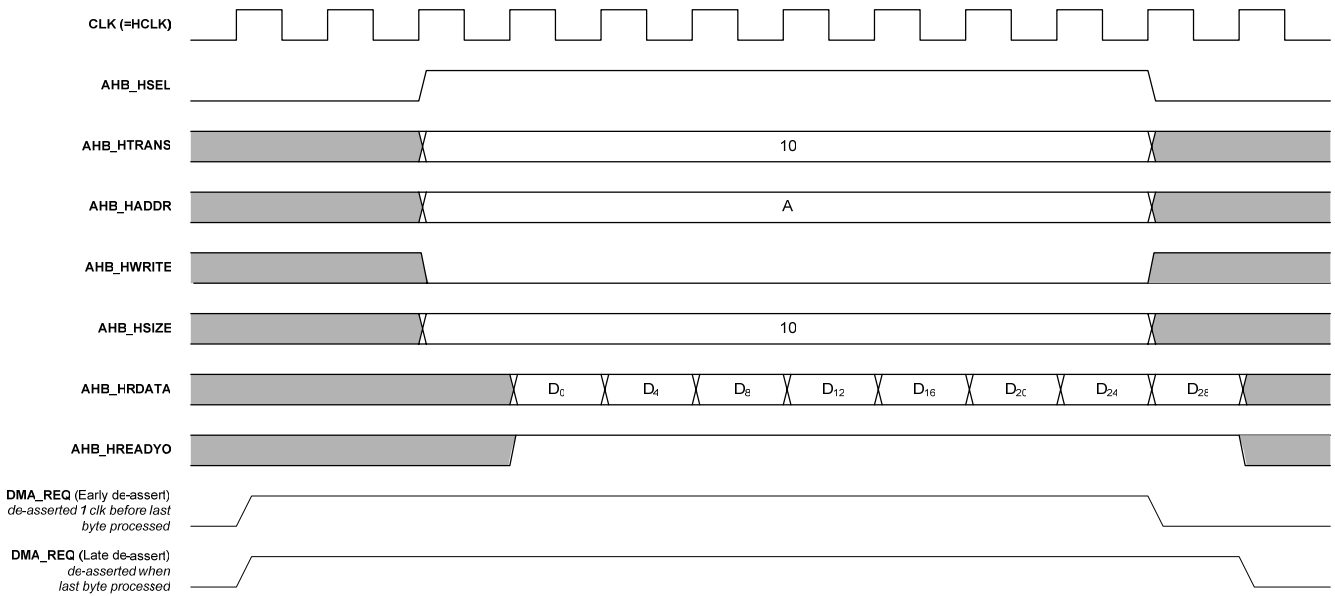
20.6. SESSION CONTROL
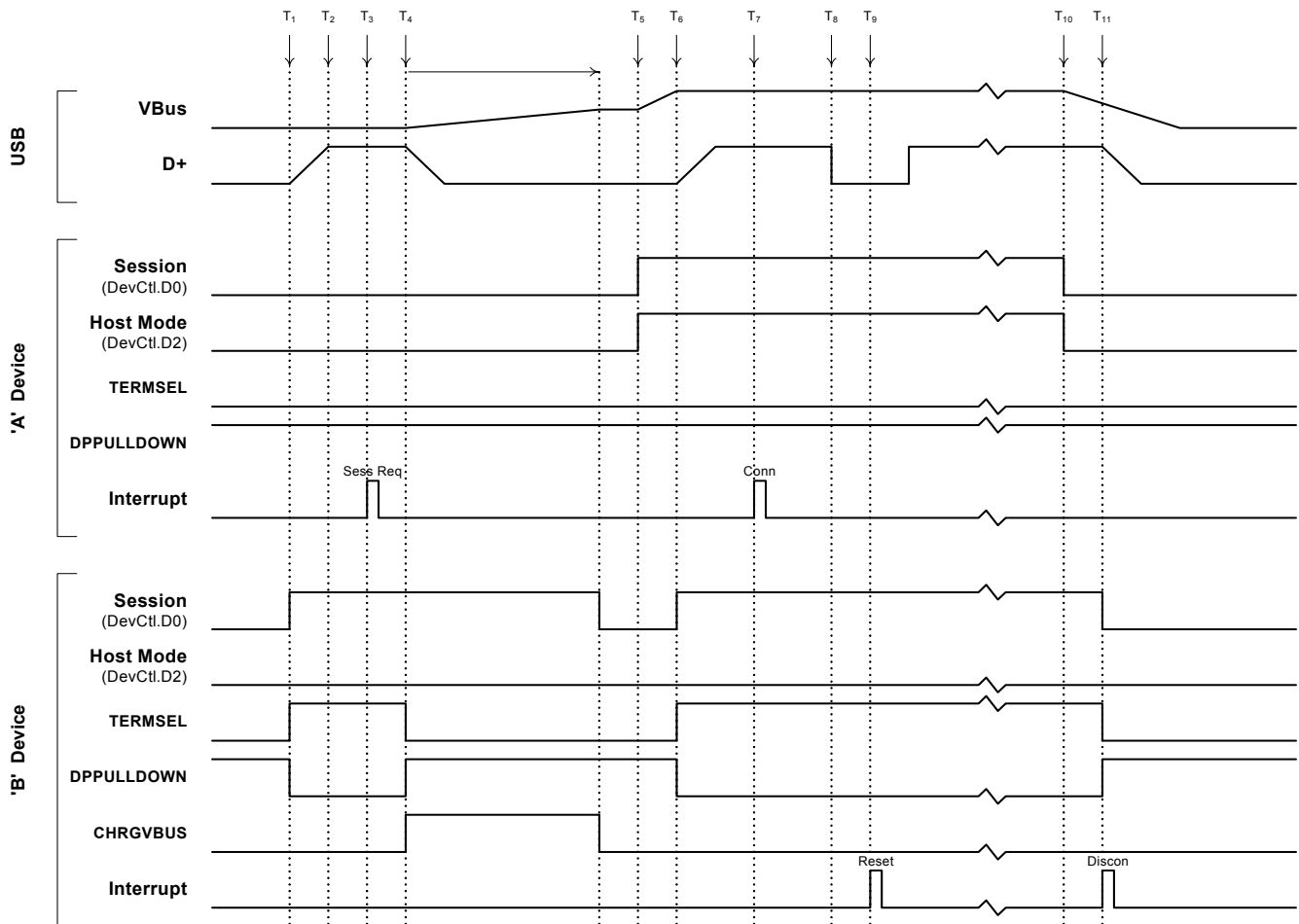
T$_1$ 'B' Device firmware requests Session Start by setting the Session bit (DevCtl.D0).

T$_2$ 'B' Device pulses the data line.

T$_3$ 'A' Device detects the data line pulsing and generates a Session Request interrupt (IntrUSB.D6).

T$_4$ 'B' Device pulses VBus (by taking CHRGVBUS high).

T$_5$ 'A' Device firmware starts the session by setting the Session bit (DevCtl.D0).

T$_6$ 'B' Device detects the Session Start and takes TERMSEL high (may be used to apply a pull-up resistor to D+).

T$_7$ 'A' Device detects the addition of the pull-up resistor and generates a Connect interrupt (IntrUSB.D4).

T$_8$ 'A' Device firmware resets the 'B' device, and then starts transactions.

T$_9$ 'B' Device detects reset and generates a Reset interrupt (IntrUSB.D2).

T$_{10}$ 'A' Device firmware ends the session by clearing the Session bit.

T$_{11}$ 'B' Device detects the Session End and generates a Disconnect interrupt (IntrUSB.D5).

CONFIDENTIAL

20.7. HOST NEGOTIATION
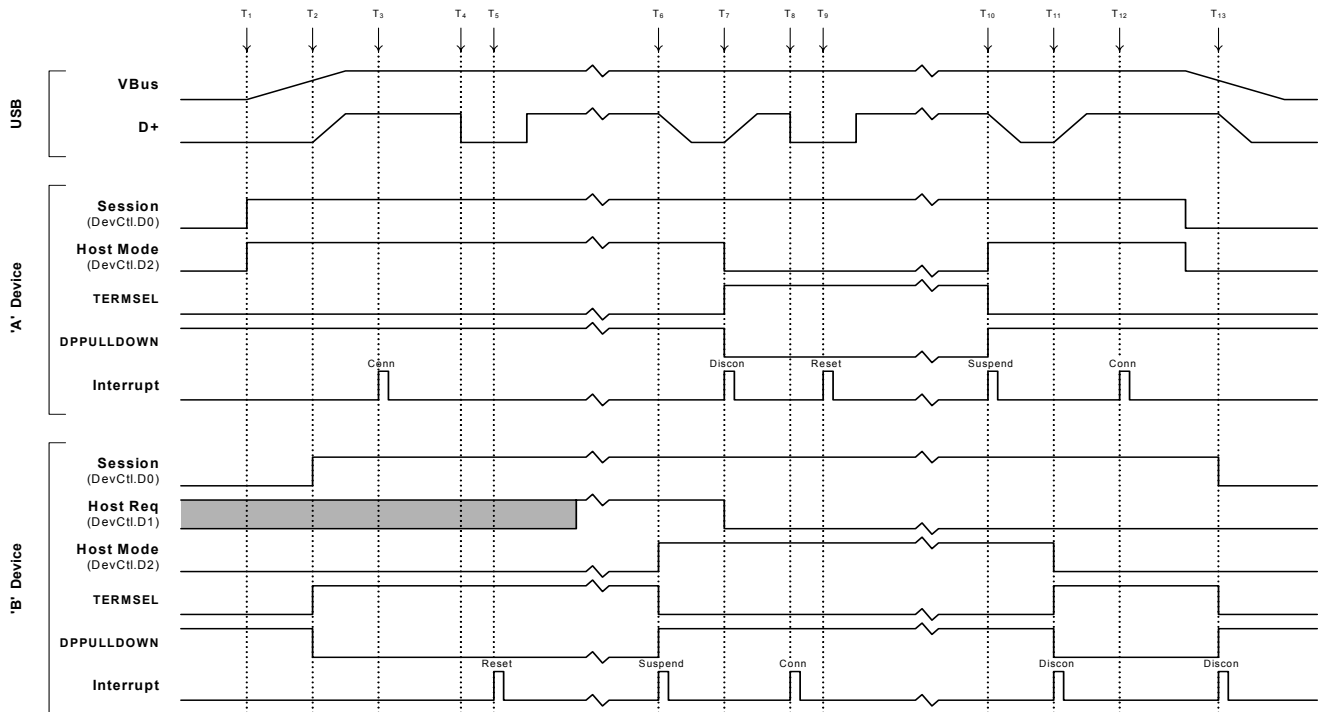
T$_1$ 'A' Device firmware starts the session by setting the Session bit (DevCtl.D0).

T$_2$ 'B' Device detects the Session Start and takes TERMSEL high (may be used to apply the pull-up resistor to D+).

T$_3$ 'A' Device detects the presence of the 'B' device and generates a Connect interrupt (IntrUSB.D4).

T$_4$ 'A' Device firmware resets the 'B' device, and then starts transactions.

T$_5$ 'B' Device detects reset and generates a Reset interrupt (IntrUSB.D2).

T$_6$ 'B' device detects Suspend mode on bus while Host Req (DevCtl.D1) is set, generates Suspend interrupt (IntrUSB.D0) and takes TERMSEL low, removing the pull-up on D+.

T$_7$ 'A' device detects the disconnect, generates a Disconnect interrupt (IntrUSB.D5) and takes its TERMSEL high (which again may be used to apply a pull-up to D+).

T$_8$ 'B' device detects the 'A' device, generates a Connect interrupt (IntrUSB.D4) and initiates a reset of the 'A' device. (The 'B' device firmware may clear the Reset bit (Power.D3) after 20 ms and begin transactions.)

T$_9$ 'A' device detects reset and generates a Reset interrupt (IntrUSB.D2).

T$_{10}$ 'A' Device detects Suspend mode on bus, generates Suspend interrupt (IntrUSB.D0) and takes TERMSEL low, removing the pull-up.

T$_{11}$ 'B' device detects the disconnect, generates a Disconnect interrupt (IntrUSB.D5) and takes its TERMSEL high, applying its pull-up to D+.

T$_{12}$ 'A' Device detects 'B' device, then ends the session by clearing the Session bit.

T$_{13}$ 'B' Device detects the Session End and generates a Disconnect interrupt (IntrUSB.D5).

CONFIDENTIAL

21. CONTROL TRANSACTIONS (VIA ENDPOINT 0)

21.1. CONTROL TRANSACTIONS AS A PERIPHERAL

Endpoint 0 is the main control endpoint of the core. As such, the routines required to service Endpoint 0 are more complicated than those required to service other endpoints.

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 transaction per transfer. To accommodate this, the CPU needs to take a state machine approach to command decoding and handling.

The Standard Device Requests received by a USB peripheral can be divided into three categories: *Zero Data Requests* (in which all the information is included in the command), W*rite 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 MUSBMHDRC core.

21.1.1. 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 (CSR0L.D0) 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 CSR0 register should then be written to set the ServicedRxPktRdy bit (D6) (indicating that the command has been read from the FIFO) and to set the DataEnd bit (D3) (indicating that no further data is expected for this request).

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.

If the command is an unrecognized command, or for some other reason cannot be executed, then when it has been decoded, the CSR0 register should be written to set the ServicedRxPktRdy bit (D6) and to set the SendStall bit (D5). When the host moves to the status stage of the request, the MUSBMHDRC 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 (CSR0L.D2) will be set.

If the host sends more data after the DataEnd bit has been set, then the MUSBMHDRC will send a STALL. An Endpoint 0 interrupt will be generated and the SentStall bit (CSR0L.D2) will be set.

CONFIDENTIAL

**SOFT
CORES**

21.1.2. 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 (CSR0L.D0) 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 CSR0 register should then be written to set the ServicedRxPktRdy bit (D6) (indicating that the command has been read from the FIFO) but in this case the DataEnd bit (D3) should not be set (indicating that more data is expected).

When a second Endpoint 0 interrupt is received, the CSR0 register should be read to check the endpoint status. The RxPktRdy bit (CSR0L.D0) 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, CSR0 should be written to set the ServicedRxPktRdy bit, but the DataEnd bit should not be set.

When all the expected data packets have been received, the CSR0 register should be written to set the ServicedRxPktRdy 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 CSR0 register should be written to set the ServicedRxPktRdy bit (D6) and to set the SendStall bit (D5). When the host sends more data, the MUSBMHDRC 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 (CSR0L.D2) will be set.

If the host sends more data after the DataEnd has been set, then the MUSBMHDRC will send a STALL. An Endpoint 0 interrupt will be generated and the SentStall bit (CSR0L.D2) will be set.

21.1.3. 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 (CSR0L.D0) will also have been set. The 8-byte command should then be read from the Endpoint 0 FIFO and decoded. The CSR0 register should then be written to set the ServicedRxPktRdy bit (D6) (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 CSR0 register should then be written to set the TxPktRdy bit (D1) (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.

CONFIDENTIAL

When the last data packet has been written to the FIFO, the CSR0 register should be written to set the TxPktRdy bit and to set the DataEnd bit (D3) (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 CSR0 register should be written to set the ServicedRxPktRdy bit (D6) and to set the SendStall bit (D5). When the host requests data, the MUSBMHDRC 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 (CSR0L.D2) will be set.

If the host requests more data after DataEnd (D3) has been set, then the MUSBMHDRC will send a STALL. An Endpoint 0 interrupt will be generated and the SentStall bit (CSR0L.D2) will be set.

21.1.4. ENDPOINT 0 STATES

When the MUSBMHDRC is operating as a peripheral, 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.

The default mode on power-up or reset should be IDLE.

RxPktRdy (CSR0L.D0) becoming set when Endpoint 0 is in IDLE state indicates a new device request. Once the device request is unloaded from the FIFO, the MUSBMHDRC 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).

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 (e.g. loading the FIFO, Setting TxPktRdy) are indicated in the diagram on the following page.

Note that the MUSBMHDRC changes the FIFO direction depending on the direction of the Data phase *independently* of the CPU.

Figure 21-1 Endpoint 0 States for Peripheral
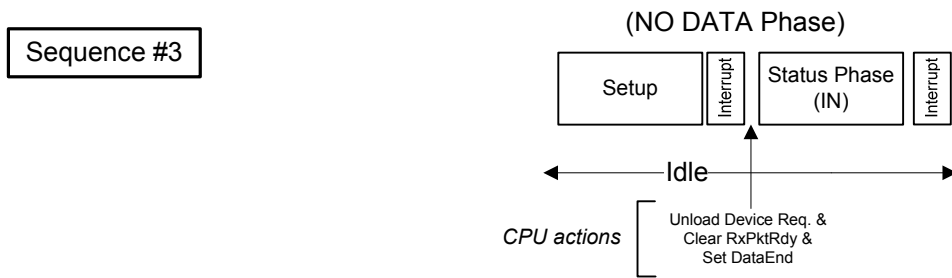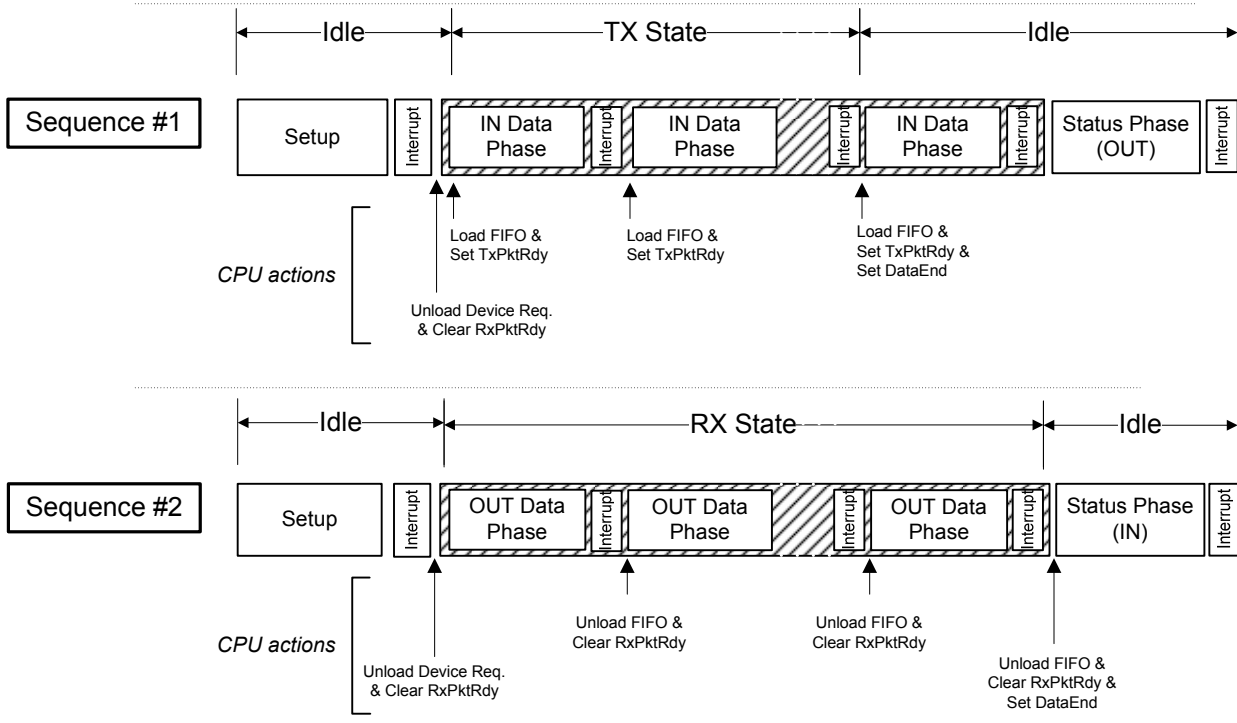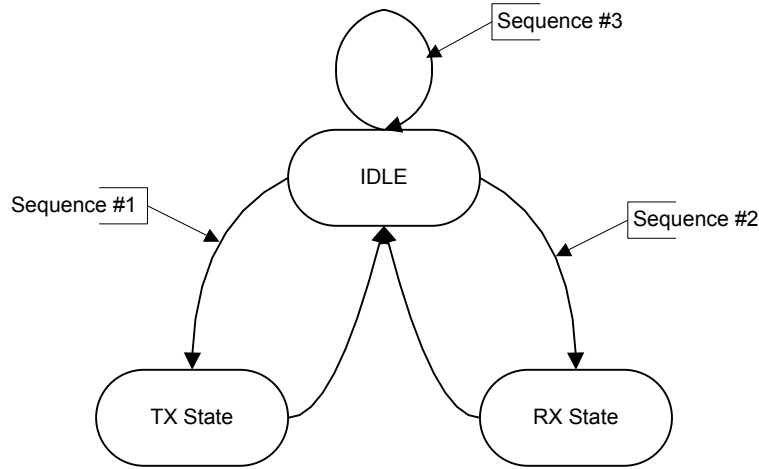
21.1.5.   ENDPOINT 0 SERVICE ROUTINE AS PERIPHERAL

An Endpoint 0 interrupt is generated:

- When the core sets the RxPktRdy bit (CSR0L.D0) after a valid token has been received and data has been written to the FIFO.

- When the core clears the TxPktRdy bit (CSR0L.D1) after the packet of data in the FIFO has been successfully transmitted to the host.

- When the core sets the SentStall bit (CSR0L.D2) after a control transaction is ended due to a protocol violation.

- When the core sets the SetupEnd bit (CSR0L.D4) because a control transfer has ended before DataEnd (CSR0L.D3) is set.

Whenever the Endpoint 0 service routine is entered, the firmware 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 firmware should abort processing the current control transfer and set the state to IDLE.

Once the firmware has determined that the interrupt was not generated by an illegal bus state, the next action taken depends on the Endpoint state.

If Endpoint 0 is in IDLE state, the only valid reason an interrupt can be generated is as a result of the core receiving data from the USB bus. The service routine must check for this by testing the RxPktRdy (CSR0L.D0) bit. If this bit is set, then the core has received a SETUP packet. This must be unloaded from the FIFO and decoded to determine the action the core 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 is in TX state, the interrupt indicates that the core has received an IN token and data from the FIFO has been sent. The firmware must respond to this either by placing more data in the FIFO if the host is still expecting more data[2] 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.

If the endpoint is in RX state, the interrupt indicates that a data packet has been received. The firmware must respond by unloading the received data from the FIFO. The firmware must then determine whether it has received all of the expected data[2]. If it has, the firmware should set the DataEnd bit and return Endpoint 0 to IDLE state. If more data is expected, the firmware should set the ServicedRxPktRdy bit (CSR0L.D6) to indicate that it has read the data in the FIFO and leave the endpoint in RX state.

---

[2] Command transactions all include a field that indicates the amount of data the host expects to receive or is going to send.

CONFIDENTIAL

SOFT CORES

Figure 21-2 Flow for Endpoint 0 Service Routine when MUSBMHDRC operating as a Peripheral

```
      ┌─────────────┐
      │   Service   │
      │ Endpoint 0  │
      └──────┬──────┘
             │
   ┌─────────▼─────────┐
   │ Read Endpoint 0 CSR│
   └─────────┬─────────┘
             │
          ◇─────── Yes ──────►  ┌──────────────────────┐
       Sent Stall ?              │  Clear SentStall bit │
          ◇                      │   State → IDLE       │
          │ No                   └──────────────────────┘
          │
          ▼
       Setup  ── Yes ──►  ┌──────────────────────┐
       End ?              │ Set ServicedSetupEnd │
          │ No            │   State → IDLE       │
          ▼               └──────────────────────┘
       State
         =   ── Yes ──►  ( IDLE Mode )
       IDLE ?
          │ No
          ▼
       State
         =   ── Yes ──►  ( TX Mode )
        TX ?
          │ No
          ▼
       State
         =   ── Yes ──►  ( RX Mode )
        RX*
```
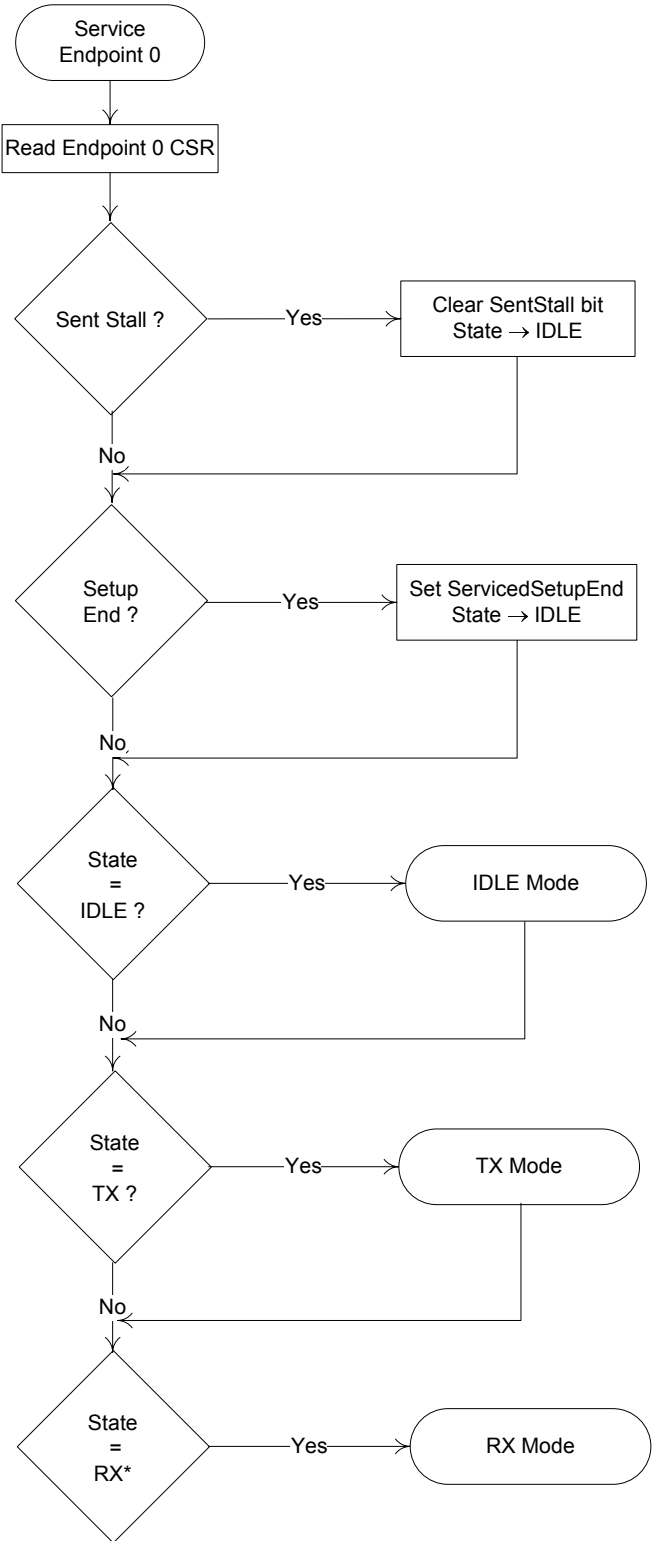
* By default

21.1.5.1. IDLE MODE

IDLE mode is the mode the Endpoint 0 control needs to 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 the figure below).

Figure 21-3 Flow for SETUP phase of Control Transfer when MUSBMHDRC operating as a Peripheral

```
                          ┌──────────────┐
                          │  IDLE Mode   │
                          └──────┬───────┘
                                 │
                            ◇ RxPktRdy
                              set ?      ──No──┐
                                 │             │
                                Yes      ┌───────────┐
                                 │       │  Return   │
                          ┌──────────────┐└───────────┘
                          │ Unload FIFO  │
                          └──────┬───────┘
                          ┌──────────────┐
                          │Decode command│
                          └──────┬───────┘
                                 │
                          ◇ Command
                            has Data  ──No──→ ┌──────────────────┐
                            Phase ?           │ Process command  │
                                 │            └────────┬─────────┘
                                Yes           ┌──────────────────┐
                          ┌──────────────┐    │Set ServicedRxPktRdy│
                          │     Set      │    │   Set DataEnd    │
                          │ServicedRxPktRdy│  └────────┬─────────┘
                          └──────┬───────┘    ┌──────────────────┐
                                 │            │     Return       │
                          ◇ Data Phase        └──────────────────┘
                             = 
                            IN ?     ──Yes──→ ┌──────────────────┐
                                 │            │  State → TX      │
                                No            └────────┬─────────┘
                          ┌──────────────┐    ┌──────────────────┐
                          │  State → RX  │    │     Return       │
                          └──────┬───────┘    └──────────────────┘
                          ┌──────────────┐
                          │    Return    │
                          └──────────────┘
```
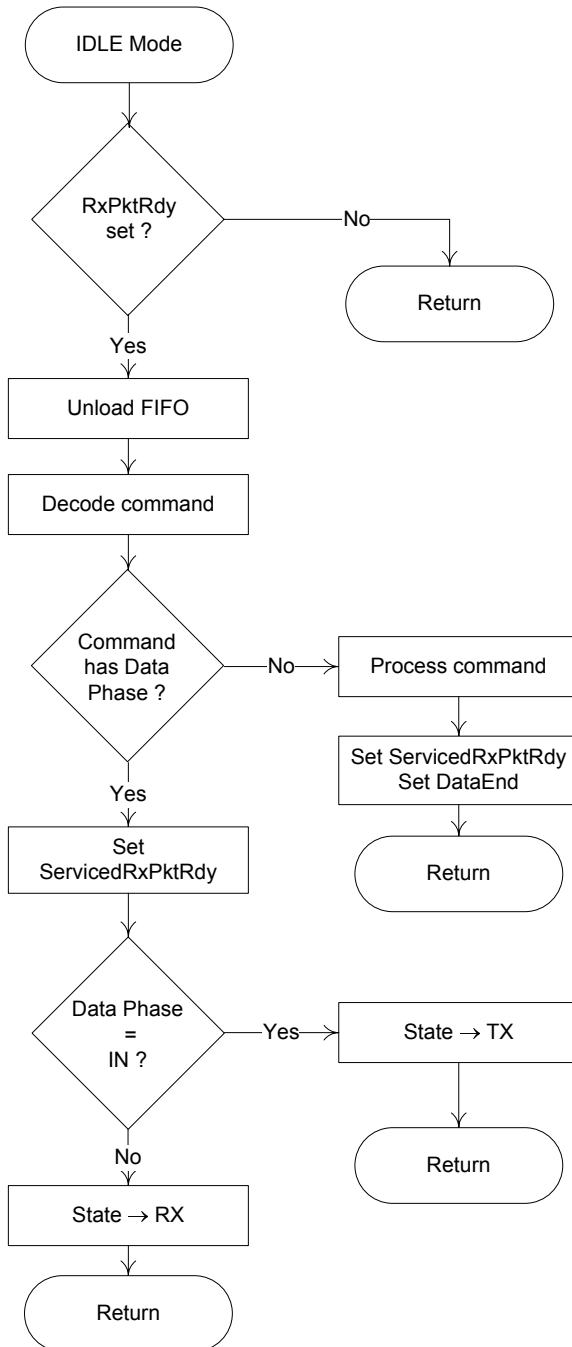
21.1.5.2. TX MODE

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 whilst the endpoint is in the TX state, this will cause a SetupEnd condition to occur as the core expects only IN tokens.

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

- The host sends an invalid token causing a SetupEnd condition (CSR0L.D4 set)

- The firmware sends a packet containing less than the maximum packet size for Endpoint 0 (MaxP)

- The firmware sends an empty data packet

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

When the firmware forces the termination of a transfer (by sending a short or empty data packet), it should set the DataEnd bit (CSR0L.D3) to indicate to the core that the Data phase is complete and that the core should next receive an acknowledge packet.

Figure 21-4 Flow for IN Data phase of Control Transfer when MUSBMHDRC operating as a Peripheral
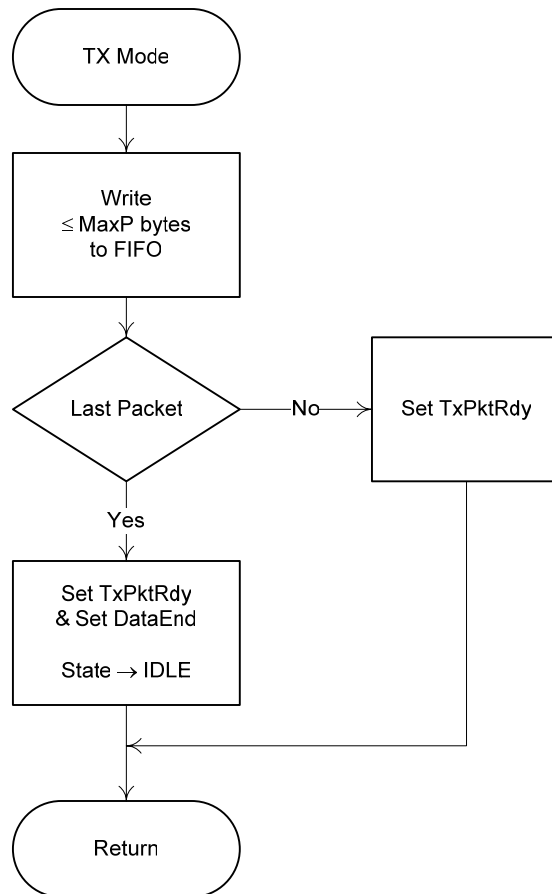
21.1.5.3. RX MODE

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, this will cause a SetupEnd condition to occur as the core expects only OUT tokens.

CONFIDENTIAL

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

- The host sends an invalid token causing a SetupEnd condition (CSR0L.D4 set)

- The host sends a packet which contains less than the maximum packet size for Endpoint 0

- The host sends an empty data packet

Until the transaction is terminated, the firmware simply needs to unload the FIFO when it receives an interrupt which indicates that new data has arrived (RxPktRdy (CSR0L.D0) set) and to clear RxPktRdy by setting the ServicedRxPktRdy bit (CSR0L.D6).

When the firmware 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 (CSR0L.D3) to indicate to the core that the Data phase is complete and that the core should receive an acknowledge packet next.

Figure 21-5 Flow for OUT Data phase of Control Transfer when MUSBMHDRC operating as a Peripheral
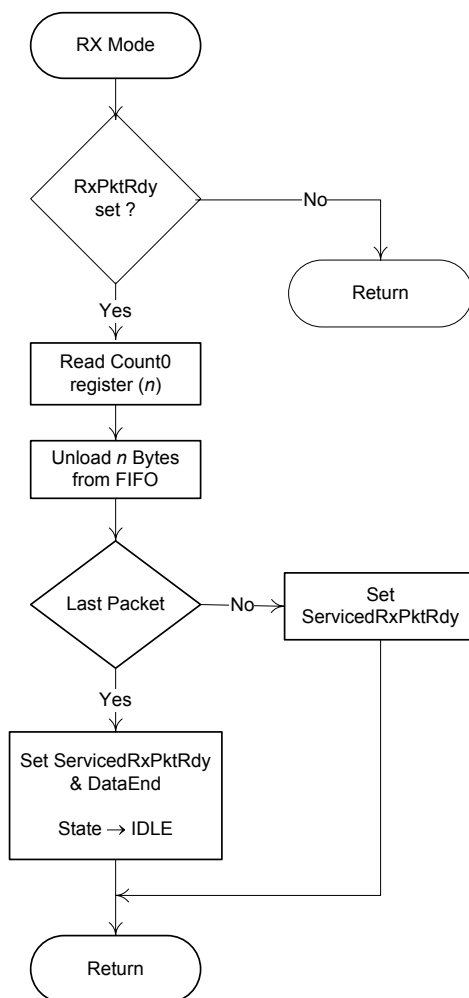
21.1.6. ERROR HANDLING AS A PERIPHERAL

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

The MUSBMHDRC will automatically detect protocol errors and send a STALL packet to the host under the following

CONFIDENTIAL

conditions:

1. 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 (CSR0L.D3) has been set.

2. The host request 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 CSR0 register has been set.

3. The host sends more than MaxP data bytes in an OUT data packet.

4. The host sends a non-zero length DATA1 packet during the STATUS phase of a read request.

When the MUSBMHDRC has sent the STALL packet, it sets the SentStall bit (CSR0L.D2) 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 (CSR0L.D4) 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 ServicedSetupEnd bit (CSR0L.D7), and return to the IDLE state. If the RxPktRdy bit (CSR0L.D0) 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 (CSR0L.D5). The MUSBMHDRC will then send a STALL packet to the host, set the SentStall bit (CSR0L.D2) and generate an Endpoint 0 interrupt.

21.1.7. ADDITIONAL ACTIONS

When working as a peripheral, the MUSBMHDRC core automatically responds to certain conditions on the USB bus or actions by the host. The details are given below:

STALL ISSUED TO CONTROL TRANSFER

The MUSBMHDRC core will automatically issue a STALL handshake to a Control transfer under the following conditions:

1. 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 MUSBMHDRC when the host sends an OUT token (instead of an IN token) after the CPU has unloaded the last OUT packet and set DataEnd.

2. 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 MUSBMHDRC when the host sends an IN token (instead of an OUT token) after the CPU has cleared TxPktRdy and set DataEnd in response to the ACK issued by the host to what should have been the last packet.

3. The host sends more than MaxP data with an OUT data token.

4. The host sends more than a zero length data packet for the OUT Status phase.

ZERO-LENGTH OUT DATA PACKETS IN CONTROL TRANSFERS

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 CPU 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 MUSBMHDRC will automatically flush any IN token loaded by CPU ready for the Data phase from the FIFO and set SetupEnd (CSR0L.D4).

CONFIDENTIAL

21.2. CONTROL TRANSACTIONS AS A HOST

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, Chapter 9).

As for a USB peripheral, there are three categories of Standard Device Requests to be handled: *Zero Data Requests* (in which all the information is included in the command); W*rite 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).

- Zero Data Requests comprise a SETUP command followed by an IN Status Phase

- Write Requests comprise a SETUP command, followed by an OUT Data Phase which is in turn followed by an IN Status Phase.

- Read Requests comprise 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 MUSBMHDRC will retry a transaction which is continually NAKed by the target. This limit can be between 2 and 2^{15} frames/microframes and is set through the NAKLimit0 register (see Section 3.3.6).

The following sections describe the actions that the CPU needs to take in issuing these different types of request through looking at the steps to take in the different phases of a Control Transaction.

Note: Before initiating any transactions as a Host, the FAddr register needs to be set to address the peripheral device. When the device is first connected, FAddr should be set to zero. After a SET_ADDRESS command is issued, FAddr should be set the target's new address.

21.2.1. SETUP PHASE AS A HOST

For the SETUP Phase of a Control Transaction, the CPU driving the Host device needs to:

1. Load the 8 bytes of the required Device request command into the Endpoint 0 FIFO

2. Then set SetupPkt and TxPtRdy (bits CSR0L.D3 and CSR0L.D1, respectively). *Note:* These bits need to be set together.

   The MUSBMHDRC then proceeds to send a SETUP token followed by the 8-byte command to Endpoint 0 of the addressed device, retrying as necessary. (The details of this operation are shown in Section 19.1.)

3. At the end of the attempt to send the data, the MUSBMHDRC will generate an Endpoint 0 interrupt (i.e. set IntrTx.D0). The CPU should then read CSR0 to establish whether the RxStall bit (D2), the Error bit (D4) or the NAK Timeout bit (D7) has been set.

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

   If Error is set, it means that the MUSBMHDRC 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 MUSBMHDRC has received a NAK response to each attempt to send the SETUP packet, for longer than the time set in the NAKLimit0 register. The MUSBMHDRC 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 CPU should proceed to the following IN Data Phase, OUT Data Phase or IN Status Phase specified for the particular Standard Device Request.

21.2.2. IN DATA PHASE AS A HOST

For the IN Data Phase of a Control Transaction, the CPU driving the Host device needs to:

1. Set ReqPkt (CSR0L.D5).

2. Wait while the MUSBMHDRC both sends the IN token and receives the required data back. (The details of this operation

CONFIDENTIAL

SOFT CORES

are shown in Section 19.1.)

3. When the MUSBMHDRC generates the Endpoint 0 interrupt (i.e. sets IntrTx.D0), read CSR0 to establish whether the RxStall bit (D2), the Error bit (D4), the NAK Timeout bit (D7) or RxPktRdy (D0) 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 MUSBMHDRC has tried to send the required IN token three times without getting any response.

    If NAK Timeout is set, it means that the MUSBMHDRC has received a NAK response to each attempt to send the IN token, for longer than the time set in the NAKLimit0 register. The MUSBMHDRC 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 CPU should read the data from the Endpoint 0 FIFO, then clear RxPktRdy.

5. If further data is expected, the CPU 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.

21.2.3. OUT DATA PHASE AS A HOST

For the OUT Data Phase of a Control Transaction, the CPU driving the Host device needs to:

1. Load the data to be sent into the Endpoint 0 FIFO.

2. Then set the TxPtRdy bit (CSR0L.D1).

    The MUSBMHDRC then proceeds to send an OUT token followed by the data from the FIFO to Endpoint 0 of the addressed device, retrying as necessary. (The details of this operation are shown in Section 19.1.)

3. At the end of the attempt to send the data, the MUSBMHDRC will generate an Endpoint 0 interrupt (i.e. set IntrTx.D0). The CPU should then read CSR0 to establish whether the RxStall bit (D2), the Error bit (D4) or the NAK Timeout bit (D7) 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 MUSBMHDRC 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 MUSBMHDRC has received a NAK response to each attempt to send the OUT token, for longer than the time set in the NAKLimit0 register. The MUSBMHDRC 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 CPU should repeat Steps 1 – 3.

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

21.2.4. IN STATUS PHASE AS A HOST

(FOLLOWING SETUP PHASE OR OUT DATA PHASE)

For the IN Status Phase of a Control Transaction, the CPU driving the Host device needs to:

1. Set StatusPkt and ReqPkt (bits CSR0L.D6 and CSR0L.D5, respectively). *Note:* These bits need to be set together i.e. in the same write operation.

2. Wait while the MUSBMHDRC both sends an IN token and receives a response from the USB peripheral. (The details of this operation are shown in Section 19.1.)

3. When the MUSBMHDRC generates the Endpoint 0 interrupt (i.e. sets IntrTx.D0), read CSR0 to establish whether the

CONFIDENTIAL

SOFT CORES

RxStall bit (D2), the Error bit (D4), the NAK Timeout bit (D7) or RxPktRdy (D0) has been set.

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

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

If NAK Timeout is set, it means that the MUSBMHDRC has received a NAK response to each attempt to send the IN token, for longer than the time set in the NAKLimit0 register. The MUSBMHDRC 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 and StatusPkt before clearing the NAK Timeout bit.

4.   The CPU should clear the StatusPkt bit, together with (i.e. in the same write operation as) RxPktRdy if this has been set.

21.2.5. OUT STATUS PHASE AS A HOST

(FOLLOWING IN DATA PHASE)

For the OUT Status Phase of a Control Transaction, the CPU driving the Host device needs to:

1.   Set StatusPkt and TxPktRdy (bits CSR0L.D6 and CSR0L.D1, respectively). *Note:* These bits need to be set together.

2.   Wait while the MUSBMHDRC both sends the OUT token and a zero-length DATA1 packet. (The details of this operation are shown in Section 19.1.)

3.   At the end of the attempt to send the data, the MUSBMHDRC will generate an Endpoint 0 interrupt (i.e. set IntrTx.D0). The CPU should then read CSR0 to establish whether the RxStall bit (D2), the Error bit (D4) or the NAK Timeout bit (D7) has been set.

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

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

If NAK Timeout is set, it means that the MUSBMHDRC has received a NAK response to each attempt to send the IN token, for longer than the time set in the NAKLimit0 register. The MUSBMHDRC 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 STATUS Phase has been correctly ACKed.

22. BULK TRANSACTIONS

22.1. HANDLING BULK TRANSACTIONS AS A PERIPHERAL

22.1.1. BULK IN TRANSACTIONS

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

Four optional features are available for use with a Tx endpoint used in Peripheral mode for Bulk IN transactions:

- **Double packet buffering**

Except where dynamic FIFO sizing is being used, when the value written to the TxMaxP register is less than, or equal to, half the size of the FIFO allocated to the endpoint, double packet buffering will be automatically enabled. (Where dynamic FIFO sizing is selected, the use of single or double packet buffering is part of the specification for the endpoint FIFO – see Section 3.3.18) When enabled, up to two packets can be stored in the FIFO awaiting transmission to the host.

- **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 a DMA controller (such as the one optionally included in the MUSBMHDRC design)

CONFIDENTIAL

SOFT CORES

to load packets into the FIFO without processor intervention. . If DMA mode 1 is used the TxMaxP[D10:0] must be set to an even number for proper interrupt generation.

AutoSet

When the AutoSet feature is enabled, the TxPktRdy bit (TxCSRL.D0) will be automatically set when a packet of TxMaxP bytes has been loaded into the FIFO. This is particularly useful when DMA is used to load the FIFO as it avoids the need for any processor intervention when loading individual packets during a large Bulk transfer.

- **Automatic Packet Splitting**

For some system designs, it may be convenient for the application software to write larger amounts of data to an endpoint in a single operation than can be transferred in a single USB operation. A particular case in point is where the same endpoint is used for high-speed transfers of 512 bytes under certain circumstances but for full-speed transfers under other circumstances. When operating at full-speed, the maximum amount of data transferred in a single operation is then just 64 bytes. To cater for such circumstances, the MUSBMHDRC includes a configuration option which, if selected, allows larger data packets to be written to Bulk endpoints which are then split into packets of an appropriate (specified) size for transfer across the USB bus. Whether this option is selected can be determined from the setting of the MPTxE bit (D6) of the ConfigData register (see Section 3.3.5). The necessary packet size information is set via the TxMaxP register (see Section 3.3.7).

22.1.1.1. 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. In addition, the relevant interrupt enable bit in the IntrTxE register should be set to '1' (if an interrupt is required for this endpoint) and the high byte of the TxCSR register should be set as shown below (Bits D9 – D8 are unused):

D15	**AutoSet**	0/1	Set to 1 if the AutoSet feature is required.
D14	**ISO**	0	Set to 0 to enable Bulk protocol.
D13	**Mode**	1	Set to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).
D12	**DMAReqEnab**	0/1	Set to 1 if DMA Requests are required. *Note:* If set to 1, will also need to select the chosen DMAReqMode (TxCSRH.D2).
D11	**FrcDataTog**	0	Set to 0 to allow normal data toggle operation.

When the endpoint is first configured (following a SET_CONFIGURATION or SET_INTERFACE command on Endpoint 0), the lower byte of TxCSR should be written to set the ClrDataTog bit (D6). This will ensure that the data toggle (which is handled automatically by the MUSBMHDRC) starts in the correct state. Also if there are any data packets in the FIFO (indicated by the FIFONotEmpty bit (TxCSRL.D1) being set), they should be flushed by setting the FlushFIFO bit (TxCSRL.D3). *Note:* It may be necessary to set this bit twice in succession if double buffering is enabled.

22.1.1.2. OPERATION

When data is to be transferred over a Bulk IN pipe, a data packet needs to be loaded into the FIFO and the TxCSR register written to set the TxPktRdy bit (D0). When the packet has been sent, the TxPktRdy bit will be cleared by the MUSBMHDRC 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 MUSBMHDRC 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 bottom 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). If more than this amount of data is to be transferred, this needs to be sent as multiple USB packets which should all be TxMaxP[D10:D0] in size, except for the last packet which holds the residue.

The exception to this rule applies where the automatic Bulk packet splitting option has been selected when the core was configured.

CONFIDENTIAL

SOFT CORES

(This can be determined from the setting of the MPTxE bit (D6) of the ConfigData register.) Where this option has been selected, packets up to 32 times the size specified by TxMaxP[D10:D0] can be written to the FIFO (assuming that the FIFO is big enough to accept these larger packets) which are then split by the core into packets of the appropriate size for transfer over the USB. The size of the packets written to the FIFO is given by $m \times USB\text{-}payload$ where TxMaxP[D15:D11] $= m - 1$. All the application software needs to do to take advantage of this feature is set the appropriate values in the TxMaxP register (and ensure that the value written to bits 10:0 matches the value given in the *wMaxPacketSize* field of the Standard Endpoint Descriptor for the associated endpoint). As far as the application software is concerned, the process of transferring these larger packets is no different from that used to transfer a standard-sized Bulk packet.

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 have been sent when it receives a packet which is smaller than the stated payload (TxMaxP[D10:D0]). 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 a DMAError Handling

If the software wants to shut down the Bulk IN pipe, it should set the SendStall bit (TxCSRL.D4). When the MUSBMHDRC receives the next IN token, it will send a STALL to the host, set the SentStall bit (TxCSRL.D5) and generate an interrupt.

When the software receives an interrupt with the SentStall bit (TxCSRL.D5) set, it should clear the SentStall bit. It should however leave the SendStall bit (TxCSRL.D4) 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 TxCSR register (D6).

22.1.2. BULK OUT TRANSACTIONS AS A PERIPHERAL

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

Four optional features are available for use with an Rx endpoint used in Peripheral mode for Bulk OUT transactions:

- **Double packet buffering**

Except where dynamic FIFO sizing is being used, when the value written to the RxMaxP register is less than, or equal to, half the size of the FIFO allocated to the endpoint, double packet buffering will be automatically enabled. (Where dynamic FIFO sizing is selected, the use of single or double packet buffering is part of the specification for the endpoint FIFO – see Section 3.3.18) When enabled, up to two packets can be stored in the FIFO.

- **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 a DMA without processor intervention. . If DMA mode 1 is used the RxMaxP[D10:0] must be set to an even number for proper interrupt generation.

- **AutoClear**

When the AutoClear feature is enabled, the RxPktRdy bit (RxCSRL.D0) will be automatically cleared when a packet of RxMaxP bytes has been unloaded from the FIFO (with exceptions, see register description). This is particularly useful when DMA is used to unload the FIFO as it avoids the need for any processor intervention when unloading individual packets during a large Bulk transfer.

- **Automatic Packet Combining**

For some system designs, it may be convenient for the application software to read larger amounts of data from an endpoint in a single operation than can be transferred in a single USB operation. A particular case in point is where the same endpoint is used for high-speed transfers of 512 bytes under certain circumstances but for full-speed transfers under other circumstances. When operating at full-speed, the maximum amount of data transferred in a single operation is then just 64 bytes. To cater for such circumstances, the MUSBMHDRC includes a configuration option which, if selected, causes the MUSBMHDRC to combine the

CONFIDENTIAL

packets received across the USB bus into larger data packets prior to being read by the application software. Whether this option is selected can be determined from the setting of the MPRxE bit (D7) of the ConfigData register (see Section 3.3.5). The necessary packet size information is set via the RxMaxP register (see Section 3.3.10), while the size of the amalgamated packet currently in line to be read is given in the RxCount register (see Section 3.3.13).

22.1.2.1. 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 to '1' (if an interrupt is required for this endpoint) and the high byte of the RxCSR register should be set as shown below (Bits D10 – D8 are unused/Read-Only):

D15	**AutoClear**	0/1	Set to 1 if the AutoClear feature is required.
D14	**ISO**	0	Set to 0 to enable Bulk protocol.
D13	**DMAReqEnab**	0/1	Set to 1 if a DMA request is required for this endpoint. *Note:* If set to 1, will also need to select the chosen DMAReqMode (RxCSRH.D3).
D12	**DisNyet**	0	Set to 0 to allow normal PING flow control.

When the endpoint is first configured (following a SET_CONFIGURATION or SET_INTERFACE command on Endpoint 0), the lower byte of RxCSR should be written to set the ClrDataTog bit (D7). This will ensure that the data toggle (which is handled automatically by the MUSBMHDRC) starts in the correct state. Also if there are any data packets in the FIFO (indicated by the RxPktRdy bit (RxCSRL.D0) being set), they should be flushed by setting the FlushFIFO bit (RxCSRL.D4). *Note:* It may be necessary to set this bit twice in succession if double buffering is enabled.

22.1.2.2. OPERATION

When a data packet is received by a Bulk Rx endpoint, the RxPktRdy bit (RxCSRL.D0) 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 RxPktRdy should be cleared. If the FIFOFull bit was set to 1 when RxPktRdy is cleared, the MUSBMHDRC will first clear the FIFOFull bit. It will then set RxPktRdy again to indicate that there is another packet waiting in the FIFO to be unloaded.

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. The exception to this rule applies where the option for automatic combining of Bulk packets has been selected when the core was configured. (This can be determined from the setting of the MPRxE bit (D7) of the ConfigData register.) Where this option has been selected, the core can receive up to 32 packets at a time and combine them into a single packet within the FIFO (assuming that the FIFO is big enough to accept these larger packets). The size of the packets written to the FIFO is given by $m \times wMaxPacketSize$ where RxMaxP[D15:D11] = $m - 1$. All the application software needs to do to take advantage of this feature is set the appropriate values in the RxMaxP register (and ensure that the value written to bits 10:0 matches the value given in the *wMaxPacketSize* field of the endpoint descriptor). As far as the application software is concerned, the process of transferring these larger packets is no different from that used to transfer a standard-sized Bulk packet.

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.

CONFIDENTIAL

22.1.2.3. ERROR HANDLING

If the software wants to shut down the Bulk OUT pipe, it should set the SendStall bit (RxCSRL.D5). When the MUSBMHDRC receives the next packet it will send a STALL to the host, set the SentStall bit (RxCSRL.D6) and generate an interrupt.

When the software receives an interrupt with the SentStall bit (RxCSRL.D6) set, it should clear this bit. It should however leave the SendStall bit (RxCSRL.D5) 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 in the RxCSR register (D7).

22.2. HANDLING BULK TRANSACTIONS AS A HOST

22.2.1. BULK IN TRANSACTION AS A HOST

A Bulk IN transaction may be used to transfer non-periodic data from the function controller to the host.

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

- **Double packet buffering**

When double packet buffering is enabled, one packet can be received while another is being read. Except where dynamic FIFO sizing is used, if the value written to the RxMaxP register is less than, or equal to, half the size of the FIFO allocated to the endpoint, double packet buffering will be automatically enabled. (Where dynamic FIFO sizing is selected, the use of single or double packet buffering is part of the specification for the endpoint FIFO – see Section 8.4.2.2.)

- **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 DMA to unload packets from the FIFO without processor intervention.

- **AutoReq(uest)**

When the AutoReq(uest) feature is enabled, the ReqPkt bit (RxCSRL.D5) will be automatically set when the RxPktRdy bit is cleared. This feature may be used in conjunction with the RqPktCount register to request the required number of maximum-size packets.

- **AutoClear**

When the AutoClear feature is enabled, the RxPktRdy bit (RxCSRL.D0) will be automatically cleared when a packet of RxMaxP bytes has been unloaded from the FIFO (with exceptions, see register description). This is particularly useful together with AutoRequest when DMA is used to unload the FIFO as it avoids the need for any processor intervention when unloading individual packets during a large Bulk transfer.

- **Automatic Packet Combining**

For some system designs, it may be convenient for the application software to read larger amounts of data from an endpoint in a single operation than can be transferred in a single USB operation. A particular case in point is where the same endpoint is used for high-speed transfers of 512 bytes under certain circumstances but for full-speed transfers under other circumstances. When operating at full-speed, the maximum amount of data transferred in a single operation is then just 64 bytes. To cater for such circumstances, the MUSBMHDRC includes a configuration option which, if selected, causes the MUSBMHDRC to combine the packets received across the USB bus into larger data packets prior to being read by the application software. Whether this option is selected can be determined from the setting of the MPRxE bit (D7) of the ConfigData register (see Section 3.3.5).
The necessary packet size information is set via the RxMaxP register (see Section 3.3.10), while the size of the amalgamated packet currently in line to be read is given in the RxCount register (see Section 3.3.13).

CONFIDENTIAL

SOFT
CORES

Before initiating any Bulk IN Transactions in Host mode:

- The target function address needs to be set as described in Section 8.5.1.

- The RxType register for the MUSBMHDRC endpoint that is to be used needs to be written with bits D7, D6 set to select the operating speed, bits D5, D4 = 10 (to select a Bulk transfer) and bits D3 – D0 set to the value of the endpoint number contained in the IN endpoint descriptor returned to the MUSBMHDRC during device enumeration (see Section 3.3.16).

- The RxMaxP register for the MUSBMHDRC endpoint must be written with the maximum packet size (in bytes) for the transmission. This value should be the same as the *wMaxPacketSize* field of the Standard Endpoint Descriptor for the target endpoint.

- The RxInterval register needs to be written with the required value for the NAK Limit ($2 - 2^{15}$ 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 to '1' (if an interrupt is required for this endpoint)

- The following bits of RxCSR register should be set as shown below:

D15	**AutoClear**	0/1	Set to 1 if the AutoClear feature is required.
D14	**AutoReq**	0/1	Set to 1 if the AutoRequest feature is required.
D13	**DMAReqEnab**	0/1	Set to 1 if a DMA request is required for this endpoint. *Note:* If set to 1, will also need to select the chosen DMAReqMode (RxCSRH.D3).
D12	**DisNyet**	0	Set to 0 to allow normal PING flow control.

When the endpoint is first configured, the endpoint data toggle should be set to 0 either by using the Data Toggle Write Enable and Data Toggle bits (RxCSRH.D2 and D9) to toggle the current setting or by writing the lower byte of RxCSR to set the ClrDataTog bit (D7). This will ensure that the data toggle (which is handled automatically by the MUSBMHDRC) starts in the correct state. Also if there are any data packets in the FIFO (indicated by the RxPktRdy bit (RxCSRL.D0) being set), they should be flushed by setting the FlushFIFO bit (RxCSRL.D4). *Note:* It may be necessary to set this bit twice in succession if double buffering is enabled.

22.2.1.2. OPERATION

When Bulk data is required from the USB peripheral, the CPU should set the ReqPkt bit in the corresponding RxCSR register (D5). The MUSBMHDRC will then send an IN token to the selected Peripheral endpoint and waits for data to be returned.

If data is correctly received, RxPktRdy (RxCSRL.D0) is set. If the USB peripheral responds with a STALL, RxStall (RxCSRL.D6) is set. If a NAK is received, the MUSBMHDRC tries again – and continues to try until either the transaction is successful or the NAKLimit set in the RxInterval register is reached. If no response at all is received, two further attempts are made before the MUSBMHDRC reports an error (RxCSRL.D2 set).

The MUSBMHDRC then generates the appropriate endpoint interrupt, whereupon the CPU should read the corresponding RxCSR register to determine whether the RxPktRdy, RxStall, Error or NAK Timeout bit is set and act accordingly. (If the NAK Timeout bit is set, the MUSBMHDRC 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 clearing ReqPkt before clearing the NAK Timeout bit.)

The packets received should not exceed the size specified by RxMaxP[D10:D0] (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, it will be sent as multiple packets. All the packets will be *wMaxPacketSize* in size, except the last packet which will contain the residue. If the size of the block to be transferred is known, the number of packets of *wMaxPacketSize* to be transferred may be written to the RqPktCount register and the AutoReq option set. As each packet is requested, the value in the RqPktCount register will be decremented. At the point that RqPktCount is decremented from 1 to 0, AutoReq is cleared to stop any further requests being made. Alternatively it may be inferred that the

CONFIDENTIAL

entire block has been received when a packet which is less than *wMaxPacketSize* in size is received. (If the total size of the data block is a multiple of *wMaxPacketSize*, the last packet may still be less than *wMaxPacketSize* in size as a null data packet is often sent after the data to signify that the transfer is complete.) In this case, RqPktCount should be left set to zero. Then if AutoReq is set, it will be automatically cleared when the short packet is received.

The above describes the general case in which the application software read each packet from the FIFO individually. The behavior differs slightly where the option for automatic combining of Bulk packets was selected when the core was configured. (This can be determined from the setting of the MPRxE bit (D7) of the ConfigData register.) Where this option is selected, the core can receive up to 32 packets at a time and combine them into a single packet within the FIFO (assuming that the FIFO is big enough to accept these larger packets). The size of the packets written to the FIFO is given by $m \times wMaxPacketSize$ where RxMaxP[D15:D11] = $m - 1$. All the application software needs to do to take advantage of this feature is to set the appropriate values in the RxMaxP register (and ensure that the value written to bits 10:0 matches the value given in the *wMaxPacketSize* field of the endpoint descriptor). Values such as the number to set in the RqPktCount register are then calculated on the basis of packets of $m \times wMaxPacketSize$, making the process of transferring these larger packets is no different from that used to transfer a standard-sized Bulk packet as far as the application software is concerned.

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.

22.2.1.3. 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 (RxCSRL.D6) being set.

22.2.2. BULK OUT TRANSACTION AS A HOST

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

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

- **Double packet buffering**

Except where dynamic FIFO sizing is being used, when the value written to the TxMaxP register is less than, or equal to, half the size of the FIFO allocated to the endpoint, double packet buffering will be automatically enabled. (Where dynamic FIFO sizing is selected, the use of single or double packet buffering is part of the specification for the endpoint FIFO – see Section 8.4.1.2) When enabled, up to two packets can be stored in the FIFO awaiting transmission to the peripheral.

- **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 DMA to load packets into the FIFO without processor intervention.

- **AutoSet**

When the AutoSet feature is enabled, the TxPktRdy bit (TxCSRL.D0) will be automatically set when a packet of TxMaxP bytes has been loaded into the FIFO. This is particularly useful when DMA is used to load the FIFO as it avoids the need for any processor intervention when loading individual packets during a large Bulk transfer.

- **Automatic Packet Splitting**

For some system designs, it may be convenient for the application software to write larger amounts of data to an endpoint in a single operation than can be transferred in a single USB operation. A particular case in point is where the same endpoint is used for high-speed transfers of 512 bytes under certain circumstances but for full-speed transfers under other circumstances. When operating at full-speed, the maximum amount of data transferred in a single operation is then just 64 bytes. To cater for such circumstances, the MUSBMHDRC includes a configuration option which, if selected, allows larger data packets to be written to Bulk endpoints which are then split into packets of an appropriate (specified) size for transfer across the USB bus. Whether this option is selected can be determined from the setting of the MPTxE bit (D6) of the ConfigData register (see Section 3.3.5).

CONFIDENTIAL

The necessary packet size information is set via the TxMaxP register (see Section 3.3.7).

22.2.2.1. SETUP

Before initiating any Bulk OUT transactions:

- The target function address needs to be set as described in Section 8.5.1.

- The TxType register for the MUSBMHDRC endpoint that is to be used needs to be written with bits D7, D6 set to select the operating speed, bits D5, D4 = 10 (to select a Bulk transfer) and bits D3 – D0 set to the value of the endpoint number contained in the OUT endpoint descriptor returned to the MUSBMHDRC during device enumeration (see Section 3.3.14).

- The TxMaxP register for the MUSBMHDRC endpoint must be written with the maximum packet size (in bytes) for the transmission. This value should be the same as the *wMaxPacketSize* field of the Standard Endpoint Descriptor for the target endpoint.

- The TxInterval register needs to be written with the required value for the NAK Limit ($2 - 2^{15}$ 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 to '1' (if an interrupt is required for this endpoint)

- The following bits of the TxCSR register should be set as shown below:

D15	**AutoSet**	0/1	Set to 1 if the AutoSet feature is required.
D13	**Mode**	1	Set to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).
D12	**DMAReqEnab**	0/1	Set to 1 if a DMA request is required for this endpoint. *Note:* If set to 1, will also need to select the chosen DMAReqMode (TxCSRH.D2).
D11	**FrcDataTog**	0	Set to 0 to allow normal data toggle operation.

When the endpoint is first configured, the endpoint data toggle should be set to 0 either by using the Data Toggle Write Enable and Data Toggle bits (TxCSRH.D1 and D0) to toggle the current setting or by writing the lower byte of TxCSR to set the ClrDataTog bit (D6). This will ensure that the data toggle (which is handled automatically by the MUSBMHDRC) starts in the correct state. Also, if there are any data packets in the FIFO (indicated by the FIFONotEmpty bit (TxCSRL.D1) being set), they should be flushed by setting the FlushFIFO bit (TxCSRL.D3). *Note:* It may be necessary to set this bit twice in succession if double buffering is enabled.

22.2.2.2. OPERATION

When Bulk data is required to be sent to the USB peripheral, the CPU 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 TxCSR register (D0). The MUSBMHDRC 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, an ACK should be received whereupon the MUSBMHDRC will clear TxPktRdy (TxCSRL.D0). If the USB peripheral responds with a STALL, RxStall (TxCSRL.D5) is set. If a NAK is received, the MUSBMHDRC tries again – and continues to try until either the transaction is successful or the NAKLimit set in the TxInterval register is reached. If no response at all is received, two further attempts are made before the MUSBMHDRC reports an error (TxCSRL.D2 set).

The MUSBMHDRC then generates the appropriate endpoint interrupt, whereupon the CPU should read the corresponding TxCSR register to determine whether the RxStall (D5), Error (D2) or NAK Timeout (D7) bit is set and act accordingly. (If the NAK Timeout bit is set, the MUSBMHDRC 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.)

In the general case, packet sizes should not exceed the size specified by the bottom 11 bits of the TxMaxP register (which should have been set to match the value set in the *wMaxPacketSize* field of the appropriate endpoint descriptor). When a block of data larger than TxMaxP needs to be sent, it will need to be sent as multiple packets – each sent as described above. These packets should all be TxMaxP[D10:D0] in size, except the last packet which holds the residue.

CONFIDENTIAL

The exception to this rule applies where the automatic Bulk packet splitting option has been selected when the core was configured. (This can be determined from the setting of the MPTxE bit (D6) of the ConfigData register.) Where this option has been selected, packets up to 32 times the size specified by TxMaxP[D10:D0] can be written to the FIFO (assuming that the FIFO is big enough to accept these larger packets) which are then split by the core into packets of the appropriate size for transfer over the USB. The size of the packets written to the FIFO is given by $m \times$ *USB-payload* where TxMaxP[D15:D11] $= m - 1$. All the application software needs to do to take advantage of this feature is set the appropriate values in the TxMaxP register. As far as the application software is concerned, the process of transferring these larger packets is no different from that used to transfer a standard-sized Bulk packet.

If the total size of the data block is a multiple of TxMaxP, the host may need to send a null packet after all the data has been sent. This can be done by setting TxPktRdy after the last 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.

22.2.2.3. 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 (TxCSRL.D5) being set.

22.3. EMPLOYING DMA

The advantage of employing DMA is that it improves bus and processor utilization when loading or unloading the FIFOs.

DMA may be used in connection with any type of transfer but it is particularly useful when large blocks of data are to be transferred through a Bulk endpoint. The USB protocol requires that large data blocks are transferred by sending a series of packets of the maximum packet size for the endpoint (512 bytes for high speed, 64 bytes for full speed). The last packet in the series may be less than the maximum packet size. Indeed, the receiver may use the reception of this 'short' packet to signal the end of the transfer (a null packet may be sent at the end of the series if the size of the data block is an exact multiple of the maximum packet size).

The DMA facilities of the MUSBMHDRC may be used in either Peripheral mode or Host mode to avoid the overhead of having to interrupt the processor after each individual packet, instead only interrupting the processor after the transfer has completed.

The following sections outline the basic actions that are involved in using DMA alongside some standard types of Bulk Tx and Bulk Rx transfer. These actions may be carried out either using the built-in DMA controller (where this is implemented in the core) or using an external DMA controller.

22.3.1. USING DMA WITH BULK TX ENDPOINTS

For Tx endpoints, the DMA request line goes high when the endpoint FIFO is able to accept a data packet, and goes low when TxMaxP bytes have been loaded into the FIFO. Alternatively, the request line will go low when the TxPktRdy bit in TxCSR is set.

To use DMA to send a large block of data to the USB host over a Bulk Tx endpoint, we recommend setting up the DMA controller and the MUSBMHDRC as follows.

The DMA controller should be programmed to perform a burst DMA read of the maximum size of packet for the endpoint (512 bytes for high speed, 64 bytes for full speed) when the DMA request line for the endpoint transitions from low to high. Details of the settings to make in the case of the built-in DMA controller are given in Section 17 of the MUSBMHDRC Product Specification. The controller should keep performing these burst reads on each DMA request until the entire data block has been transferred. (The last burst may however be of less than the maximum packet size). It should then interrupt the CPU.

The MUSBMHDRC should be programmed to enable AutoSet and DMA Request Mode 1 by setting the AutoSet, DMAReqEnab and DMAReqMode bits in the TxCSR register (bits D15, D12 and D10 respectively).

Programmed like this, the MUSBMHDRC will take the DMA request line high whenever there is space in its FIFO to accept a packet. Further, the TxPktRdy bit will be automatically set after the DMA controller has loaded the FIFO with a packet of the maximum packet size. The packet is then ready to be sent to the host. When the last packet has been loaded by the DMA controller,

CONFIDENTIAL

the controller should interrupt the processor. (The built-in controller does this by asserting DMA_NINT.) If the last packet loaded was less than the maximum packet size, the TxPktRdy bit will not have been set and will therefore need to be set manually (i.e. by the CPU) to allow the last packet to be sent. The TxPktRdy bit will also need to be set manually if the last packet was of the maximum packet size and a null packet is to be sent to indicate the end of the transfer.

Note: If, when operating in Host mode, the core fails to successfully transmit a packet three times, the Error bit in the TxCSR register (TxCSR.D2) will become set and the DMA request line will be disabled until this Error bit is cleared again. It should also be noted that the DMAReqMode bit in the TxCSR register must not be cleared either before or in the same cycle as the corresponding DMAReqEnab bit is cleared.

22.3.2. USING DMA WITH BULK RX ENDPOINTS

The behavior of the DMA request line for an Rx Endpoint depends on the DMA Request Mode selected through the RxCSR register (D11). In DMA Request Mode 0, the Rx DMA request line goes high when a data packet is available in the endpoint FIFO and goes low either when the last byte of the data packet has been read – or when the RxPktRdy bit in RxCSR is cleared. In DMA Request Mode 1, the DMA request line only goes high when the packet received is of the maximum packet size (as set in the RxMaxP register). If the packet received is of some other size, the DMA request line stays low with the result that the packet remains in the FIFO with RxPktRdy set. This causes an Rx Endpoint interrupt to be generated (if enabled).

The DMA Request Modes are primarily designed to be used where large packets of data are transferred to a Bulk endpoint. The USB protocol requires such packets to be split into a series of packets of maximum packet size (512 bytes for high speed, 64 bytes for full speed). The last packet in the series may be less than the maximum packet size (or a null packet if the total size of the transfer is an exact multiple of the maximum packet size) and the receiver may interpret this 'short' packet as signaling the end of the transfer. DMA Request Mode 1 can be used, with a suitably programmed DMA controller, to avoid the overhead of having to interrupt the processor after each individual packet – instead just interrupting the processor after the transfer has completed. *Note:* If the Request Mode is switched from Request Mode 1 to Request Mode 0, the request line will be asserted if there is a packet in the FIFO in order to allow this 'pre-received' packet to be downloaded.

22.3.3. EXAMPLES

The following sections describe set-ups we recommend using when receiving a large block of data – firstly in the case where the size of the block of data is known in advance, then in the case where the size of this block isn't known in advance. *Note:* One case uses the MUSBMHDRC core's DMA Request Mode 0 while the other uses DMA Request Mode 1 but *both* operations are carried out using the built-in DMA controller's <u>DMA Mode 1</u>.

22.3.3.1.CASE 1: SIZE OF EXPECTED DATA BLOCK KNOWN

If the size of a large block of data to be received from the USB host is known before it is sent over a Bulk Rx endpoint, we recommend setting up the DMA controller and MUSBMHDRC as follows:

The DMA controller should be programmed with the size of the block to be transferred and to perform a burst DMA write of the maximum size of packet for the endpoint (512 bytes for high speed, 64 bytes for full speed) when the DMA request line for the endpoint transitions from low to high. *Details of the settings to make in the case of the built-in DMA controller are given in Section 17.4.3 of the MUSBMHDRC Product Specification.*

The MUSBMHDRC should be programmed for DMA Request Mode 0 by setting the DMAReqEnab bit in the RxCSR register (D13) and ensuring that the DMAReqMode bit (D11) is clear. As DMA Request Mode 0 handles each packet individually, it is also advisable in this instance to select AutoClear (by setting RxCSR.D15).

Programmed like this, the MUSBMHDRC will set the DMA request line high whenever it receives a packet from the host, whereupon the DMA controller should perform a burst write of the data to memory. When the DMA controller has read a packet of the maximum packet size from the FIFO, the RxPktRdy bit is automatically cleared. The DMA controller should keep performing these burst writes on each DMA request until the entire data block has been transferred (the last burst may be less than the maximum packet size).

When the DMA controller has read the last packet, it should interrupt the processor. (The built-in DMA controller does this by asserting DMA_NINT.) The CPU's response to this interrupt will depend on whether the last packet read was of maximum

CONFIDENTIAL

SOFT CORES

packet size or not. If it was less than the maximum packet size, the RxPktRdy bit will not have been cleared and will need to be cleared manually (i.e. by the CPU).

22.3.3.2. CASE 2: SIZE OF EXPECTED DATA BLOCK NOT KNOWN

Where the size of the large data block to be received from the USB host is unknown, the conventional method of identifying the end of the block is by spotting the reception of a packet of less than the maximum packet size. For this operation, we recommend setting up the DMA controller and MUSBMHDRC as follows:

The DMA controller should be programmed to perform a burst write to memory of the maximum size of packet for the endpoint (512 bytes for high speed, 64 bytes for full speed) when the DMA request line for the endpoint transitions from low to high. *Details of the settings to make in the case of the built-in DMA controller are given in Section 17.4.3 of the MUSBMHDRC Product Specification.*

The MUSBMHDRC should be programmed to enable AutoClear and DMA Request Mode 1 by setting the AutoClear, DMAReqEnab and DMAReqMode bits in the RxCSR register (Bits D15, D13 and D11 respectively), and the interrupt for the endpoint should be enabled. If the MUSBMHDRC is operating in Host mode, AutoReq (RxCSR.D14) should also be selected.

Programmed in this way, the MUSBMHDRC will set the DMA request line high (but not generate any interrupt) whenever it receives a packet of the maximum packet size from the host. When the DMA controller has read the packet from the FIFO, the RxPktRdy bit will be automatically cleared. When a packet less than the maximum packet size is received, no DMA request will be generated and so the packet will remain in the FIFO with RxPktRdy set. This will cause the MUSBMHDRC to generate the corresponding Endpoint interrupt. On receiving this interrupt, the CPU should read the RxCount register for the endpoint to determine the size of the packet and then either read this short packet manually or reprogram the DMA controller to read this packet. RxPktRdy will need to be cleared manually (i.e. by the CPU).

23. FULL-SPEED/LOW-BANDWIDTH INTERRUPT TRANSACTIONS

23.1. INTERRUPT TRANSACTIONS AS A PERIPHERAL

An Interrupt IN transaction uses the same protocol as a Bulk IN transaction (described in Section 22.1.1) and can be used the same way. Similarly, an Interrupt OUT transaction uses almost the same protocol as a Bulk OUT transaction (described in Section 22.2.2) and can be used the same way.

You should however note that Tx endpoints on a MUSBMHDRC that is used as a peripheral support one feature for Interrupt IN transactions that they do not support in Bulk IN transactions, in that they support continuous toggle of the data toggle bit. This feature is enabled by setting the FrcDataTog bit in the TxCSR register (D11). When this bit is set to '1', the MUSBMHDRC 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 MUSBMHDRC should never respond with a NYET handshake, only ACK/NAK/STALL. To ensure this, the DisNyet bit in the RxCSR register (D12) should be set to '1' 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.

23.2. INTERRUPT TRANSACTIONS AS A HOST

When the MUSBMHDRC is operating as the host, interactions with an Interrupt endpoint on the USB peripheral are handled in very much the same way as the equivalent Bulk transactions (described in Sections 22.2.1 and 22.2.2, respectively) – except that high-bandwidth Interrupt transactions are supported.

The principal difference as far as operational steps are concerned is that RxType[5:4] and TxType[5:4] need to be set to 11 (to

CONFIDENTIAL

represent an Interrupt transaction) rather than to 10.

The required polling interval also needs to be set in the RxInterval/TxInterval registers (see Sections 3.3.17 and 3.3.15).

24. FULL-SPEED/LOW-BANDWIDTH ISOCHRONOUS TRANSACTIONS

24.1. HANDLING ISOCHRONOUS TRANSACTIONS AS A PERIPHERAL

24.1.1. ISOCHRONOUS IN TRANSACTIONS

An Isochronous IN transaction is used to transfer periodic data from the function controller to the host. This section describes the use in Peripheral mode of full-speed Isochronous Tx endpoints and low bandwidth (1 packet per microframe) high-speed Isochronous Tx endpoints. High bandwidth high-speed (> 8 Mbps) endpoints are described in a later section.

Three optional features are available for use with a Tx endpoint used in Peripheral mode for Isochronous IN transactions:

- **Double packet buffering**

Except where dynamic FIFO sizing is being used, double packet buffering is automatically enabled when the value written to the TxMaxP register is less than or equal to half the size of the FIFO allocated to the endpoint. (Where dynamic FIFO sizing is selected, the use of single or double packet buffering is part of the specification for the endpoint FIFO – see Section 8.4.1.2) When enabled, up to two packets can be stored in the FIFO awaiting transmission to the host. *Note:* Double packet buffering is generally advisable for Isochronous transactions in order to avoid Under run errors (see 'Operation' below).

- **DMA**

If DMA is enabled for the endpoint, DMA request will be generated whenever the endpoint is able to accept another packet in its FIFO. This feature can be used to allow a 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 TxCSR register needs to be accessed following every packet to check for Under run errors.

- **AutoSet**

When the AutoSet feature is enabled for a low-bandwidth Isochronous endpoint, the TxPktRdy bit (TxCSRL.D0) will be automatically set when a packet of TxMaxP bytes has been loaded into the FIFO. However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size and the TxCSR register needs to be accessed following every packet to check for Under run errors.

24.1.1.1. 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 to 1 (if an interrupt is required for this endpoint) and the high byte of the TxCSR register should be set as shown below (Bits D9 – D8 are unused):

D15	**AutoSet**	0/1	Set to 1 if the AutoSet feature is required.
D14	**ISO**	1	Set to 1 to enable Isochronous protocol.
D13	**Mode**	1	Set to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).
D12	**DMAReqEnab**	0/1	Set to 1 if a DMA request is required for this endpoint. *Note:* If set to 1, will also need to select the chosen DMAReqMode (TxCSRH.D2).
D11	**FrcDataTog**	0	Ignored in Isochronous mode.

CONFIDENTIAL

SOFT CORES

24.1.1.2. OPERATION

An Isochronous endpoint does not support data retries, so if data under run 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.

The AutoSet feature can be used with a low-bandwidth Isochronous Tx endpoint, in the same way as with a Bulk Tx endpoint. However, unless the data arrives from the source at an absolutely consistent rate, synchronized to the host's frame clock, the size of the packets sent to the host will have to increase or decrease from frame to frame (or from microframe to microframe) to match the source data rate. This means that the actual packet sizes will not always be TxMaxP in size, rendering the AutoSet feature useless.

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 TxCSR register (D0) 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 (IntrUSB.D3) or the external SOF_PULSE signal from the MUSBMHDRC 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 MUSBMHDRC also maintains an external frames(/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 TxCSR (D0) and to check for data overruns/under runs (see 'Error Handling' below).

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 ISO Update bit in the Power register (D7). When this bit is set to 1, 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.

24.1.1.3. 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 TxCSR register (D2). 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 TxCSR register (D0) 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 TxCSR register (D3), or it may choose to skip the current packet.

24.1.2. ISOCHRONOUS OUT TRANSACTIONS

An Isochronous OUT transaction is used to transfer periodic data from the host to the function controller. This section describes the use in Peripheral mode of full-speed Isochronous Rx endpoints and low bandwidth (1 packet per microframe) high-speed Isochronous Rx endpoints. High bandwidth high-speed (> 8 Mbps) endpoints are described in a later section

Three optional features are available for use with an Rx endpoint used in Peripheral mode for Isochronous OUT transactions:

- **Double packet buffering**

Except where dynamic FIFO sizing is being used, double packet buffering is automatically enabled when the value written to the RxMaxP register is less than or equal to half the size of the FIFO allocated to the endpoint. (Where dynamic FIFO sizing is selected, the use of single or double packet buffering is part of the specification for the endpoint FIFO – see Section 8.4.2.2) When enabled, up to two packets can be stored in the FIFO awaiting transmission to the host. *Note:* Double packet buffering is

CONFIDENTIAL

SOFT CORES

generally advisable for Isochronous transactions in order to avoid Overrun errors (see 'Operation' below).

- **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 a 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 RxCSR register needs to be accessed following every packet to check for Overrun or CRC errors.

- **AutoClear**

When the AutoClear feature is enabled, the RxPktRdy bit (RxCSRL.D0) will be automatically cleared when a packet of RxMaxP bytes has been unloaded from the FIFO (with exceptions, see register description). However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size and the RxCSR register needs to be accessed following every packet to check for Overrun or CRC errors.

24.1.2.1. 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 to 1 (if an interrupt is required for this endpoint) and the high byte of the RxCSR register should be set as shown below (Bits D10 – D8 are unused/Read-only):

D15	**AutoClear**	0/1	Set to 1 if the AutoClear feature is required.
D14	**ISO**	1	Set to 1 to enable Isochronous protocol.
D13	**DMAReqEnab**	0/1	Set to 1 if a DMA request is required for this endpoint. *Note:* If set to 1, will also need to select the chosen DMAReqMode (RxCSRH.D3).
D12	**DisNyet**	0	Ignored in Isochronous mode.

24.1.2.2. 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.

The AutoClear feature can be used with an Isochronous Rx endpoint, in the same way as for a Bulk Rx endpoint. However, unless the data sink receives data at an absolutely consistent rate and is synchronized to the host's frame clock, the size of the packets sent from the host will have to increase or decrease from frame to frame (or from microframe to microframe) to match the required data rate. This means that the actual packet sizes will not always be RxMaxP in size, rendering the AutoClear feature useless.

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 RxCSR register (D0) 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 (IntrUSB.D3) or the external SOF_PULSE signal from the MUSBMHDRC to trigger the unloading of the data packet. The SOF_PULSE is generated once per frame(/microframe) when a SOF packet is received. (The MUSBMHDRC also maintains an external frames(/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 RxCSR and to check for data overruns/under runs (see 'Error Handling' below).

24.1.2.3. 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 RxCSR register (D2) 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

CONFIDENTIAL

SOFT CORES

determine how this error condition is handled.

If the MUSBMHDRC finds that a received packet has a CRC error, it will still store the packet in the FIFO and set the RxPktRdy bit (RxCSRL.D0) and the DataError bit (RxCSRL.D3). It is left up to the application how this error condition is handled.

24.2. HANDLING ISOCHRONOUS TRANSACTIONS AS A HOST

24.2.1. ISOCHRONOUS IN TRANSACTIONS

An Isochronous IN transaction is used to transfer periodic data from the function controller to the host. This section describes the use in Host mode of full-speed Isochronous Rx endpoints and low bandwidth (1 packet per microframe) high-speed Isochronous Rx endpoints. High bandwidth high-speed (> 8 Mbps) endpoints are described in a later section.

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

- **Double packet buffering**

When double packet buffering is enabled, one packet can be received while another is being read. Except where dynamic FIFO sizing is being used, double packet buffering will be automatically enabled when the value written to the RxMaxP register is less than or equal to half the size of the FIFO allocated to the endpoint. (Where dynamic FIFO sizing is selected, the use of single or double packet buffering is part of the specification for the endpoint FIFO – see Section 8.4.1.2). *Note:* Double packet buffering is generally advisable for Isochronous transactions in order to avoid data overrun (see 'Operation' below).

- **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 a 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.

- **AutoClear**

When the AutoClear feature is enabled, the RxPktRdy bit (RxCSRL.D0) will be automatically cleared when a packet of RxMaxP bytes has been unloaded from the FIFO (with exceptions, see register description). However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size and the RxCSR register needs to be accessed following every packet to check for Overrun or CRC errors.

- **AutoReq(uest)**

When the AutoReq(uest) feature is enabled, the ReqPkt bit (RxCSRL.D5) will be automatically set when the RxPktRdy bit is cleared.

24.2.1.1. SETUP

Before initiating an Isochronous IN Transaction:

- The target function address needs to be set as described in Section 8.5.1.

- The RxType register for the MUSBMHDRC endpoint that is to be used needs to be written with bits D7, D6 set to select the operating speed, bits D5, D4 = 01 (to select an Isochronous transfer) and bits D3 – D0 set to the value of the endpoint number contained in the IN endpoint descriptor returned to the MUSBMHDRC during device enumeration (see Section 3.3.16).

- The RxInterval register for the MUSBMHDRC endpoint needs to be written with the required transaction interval (usually one transaction every frame/microframe) – see Section 3.3.17.

- The RxMaxP register for the MUSBMHDRC endpoint must be written with the maximum packet size (in bytes) for the transmission. This value should be the same as the *wMaxPacketSize* field of the Standard Endpoint Descriptor for the target endpoint.

- The relevant interrupt enable bit in the IntrRxE register should be set to '1' (if an interrupt is required for this endpoint)

- The following bits of the RxCSR register should be set as shown below:

CONFIDENTIAL

D15	**AutoClear**	0/1	Set to 1 if the AutoClear feature is required.
D14	**AutoReq**	0/1	Set to 1 if the AutoRequest feature is required.
D13	**DMAReqEnab**	0/1	Set to 1 if a DMA request is required for this endpoint. *Note:* If set to 1, will also need to select the chosen DMAReqMode (RxCSRH.D3).
D12	**DisNyet**	0	Ignored in Isochronous mode.

24.2.1.2. OPERATION

The operation starts with the CPU setting ReqPkt (RxCSRL.D5). This causes the MUSBMHDRC 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 RxCSR register (D0) 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 MUSBMHDRC 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 RxCSR.

The AutoClear feature can be used with an Isochronous Rx endpoint, in the same way as for a Bulk Rx endpoint. However, unless the data sink receives data at an absolutely consistent rate and is synchronized to the MUSBMHDRC's frame clock, the size of the packets will increase or decrease from frame to frame (or microframe to microframe) to match the required data rate. This means that the actual packet sizes will not always be RxMaxP in size, rendering the AutoClear feature useless.

24.2.1.3. 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 DataError bit (RxCSRL.D3) is set to indicate that the data may be corrupt.

24.2.2. ISOCHRONOUS OUT TRANSACTIONS

An Isochronous OUT transaction is used to transfer periodic data from the host to the function controller. This section describes the use of full-speed Isochronous Tx endpoints and low bandwidth (1 packet per microframe) high-speed Isochronous Tx endpoints. High bandwidth high-speed (> 8 Mbps) endpoints are described in a later section.

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

- **Double packet buffering**

Except where dynamic FIFO sizing is being used, double packet buffering is automatically enabled when the value written to the TxMaxP register is less than or equal to half the size of the FIFO allocated to the endpoint. (Where dynamic FIFO sizing is selected, the use of single or double packet buffering is part of the specification for the endpoint FIFO – see Section 8.4.2.2.). When enabled, up to two packets can be stored in the FIFO awaiting transmission to the peripheral.

- **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. However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size.

- **AutoSet**

When the AutoSet feature is enabled with a low-bandwidth Isochronous endpoint, the TxPktRdy bit (TxCSRL.D0) will be automatically set when a packet of TxMaxP bytes has been loaded into the FIFO. However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size.

CONFIDENTIAL

SOFT CORES

24.2.2.1. SETUP

Before initiating an Isochronous OUT Transaction:

- The target function address needs to be set as described in Section 8.5.1.

- The TxType register for the MUSBMHDRC endpoint that is to be used needs to be written with bits D7, D6 set to select the operating speed, bits D5, D4 = 01 (to select an Isochronous transfer) and bits D3 – D0 set to the value of the endpoint number contained in the OUT endpoint descriptor returned to the MUSBMHDRC during device enumeration (see Section 3.3.14).

- The TxInterval register for the MUSBMHDRC endpoint needs to be written with the required transaction interval (usually one transaction every frame/microframe) – see Section 3.3.15.

- The TxMaxP register for the MUSBMHDRC endpoint must be written with the maximum packet size (in bytes) for the transmission. This value should be the same as the *wMaxPacketSize* field of the Standard Endpoint Descriptor for the target endpoint.

- The relevant interrupt enable bit in the IntrTxE register should be set to '1' (if an interrupt is required for this endpoint)

- The following bits of the TxCSR register should be set as shown below:

D15	**AutoSet**	0/1	Set to 1 if the AutoSet feature is required.
D13	**Mode**	1	Set to 1 to ensure FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).
D12	**DMAReqEnab**	0/1	Set to 1 if a DMA request is required for this endpoint. *Note:* If set to 1, will also need to select the chosen DMAReqMode (TxCSRH.D2).
D11	**FrcDataTog**	0	Ignored in Isochronous mode.

24.2.2.2. OPERATION

The operation starts when the CPU writes to the FIFO then sets TxPktRdy (TxCSRL.D0). This triggers the MUSBMHDRC 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 TxCSR register (D0) 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 MUSBMHDRC 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 TxCSR.

The AutoSet feature can be used with a low-bandwidth Isochronous Tx endpoint, in the same way as with a Bulk Tx endpoint. However, unless the data arrives from the source at an absolutely consistent rate, synchronized to the MUSBMHDRC's frame clock, the size of the packets will increase or decrease from frame to frame (or from microframe to microframe) to match the source data rate. This means that the actual packet sizes will not always be TxMaxP in size, rendering the AutoSet feature useless.

25. HIGH-BANDWIDTH ISOCHRONOUS/INTERRUPT TRANSACTIONS

High-Bandwidth Isochronous/Interrupt transactions use much the same protocol as other Isochronous/Interrupt transactions. There are, however, some special features to conducting High-Bandwidth transactions.

1. High-Bandwidth Isochronous/Interrupt transactions can only conducted if the MUSBMHDRC core has been configured to support these transactions (see Section 3 of the MUSBMHDRC User Guide).

The core also needs to have been configured such that the endpoints used for High-Bandwidth transactions have FIFOs of sufficient size to hold the data for at least one High-Bandwidth packet (over 1Kbyte and up to 3Kbytes).

CONFIDENTIAL

2. When setting the Maximum Packet size handled by the endpoint in the TxMaxP/RxMaxP register, the maximum number of transactions per microframe also needs to be set via bits D11 and D12 of that register.

This maximum number of transactions (2 or 3) also represents the maximum number of sections in which any single 'High-Bandwidth' packet can be transferred, which in turn sets the maximum size of packet to 2 or 3 times the maximum payload specified for the endpoint in the same register.

Note: The maximum payload that can be sent in any transaction is 1Kbyte.

3. When sending packets, TxPktRdy needs to be set by the application software. Similarly, when unloading packets from the Rx endpoint FIFO, RxPktRdy needs to be cleared by the application software.

The AutoSet and AutoClear functions cannot be used to set and clear these bits in High-Bandwidth transactions.

4. The transmission of packets as a number of sections introduces a further type of error – the transmission of Incomplete packets.

For Tx endpoints, the issue principally applies when the MUSBMHDRC is in Peripheral mode and occurs when the MUSBMHDRC fails to receive enough IN tokens from the host to send all the parts of the data packet. It can also apply to High-Bandwidth Interrupt transactions in Host mode where the core does not receive any response from the device to which the packet is being sent. In both cases, the MUSBMHDRC will set the IncompTx bit in the TxCSR register (D7).

For Rx endpoints, the issue occurs when the PIDs of the received parts of the data packet show that one or more parts of the data packet has not been received. When this happens, the MUSBMHDRC sets the IncompRx bit in the RxCSR register (D8). In the main, this bit will only be set when the core is operating in Peripheral mode but it can also be set in Host mode if (and only if) the device with which the MUSBMHDRC is communicating fails to respond in accordance with the USB protocol.

CONFIDENTIAL

SOFT CORES

26. TRANSACTION FLOWS AS A PERIPHERAL

Note: Host actions are shown against a white background. MUSBMHDRC actions are shown shaded.

26.1. CONTROL TRANSACTIONS

26.1.1. SETUP PHASE

```
                        ┌──────────────┐ ◄───────────────┐
                        │  IDLE State  │                  │
                        └──────┬───────┘                  │
                               │                          │
                 ┌─────────────▼────────────┐             │
                 │    Token sent by Host     │            │
                 │  (SETUP Token expected)   │            │
                 └─────────────┬────────────┘             │
                               │                          │
     2 - 16                   ◆                           │
     full-speed         Valid Setup Token?   ──No──►──────┤
     bit periods              ◆                           │
     OR                       │                           │
     8 - 736                 Yes                           │
     high-speed               │                           │
     bit periods   ┌──────────▼──────────┐                │
                   │    DATA0 packet      │               │
                   │    sent by Host      │               │
                   └──────────┬──────────┘                │
                              │                           │
     2 - 6.5                 ◆                            │
     full-speed      Valid Data0 Packet?                  │
     bit periods         Sent within      ──No──►─────────┘
     OR              required time?
     8 - 192             ◆
     high-speed          │
     bit periods        Yes
                         │
          ┌──────────────▼──────────────────┐
          │      Data loaded into FIFO       │
          │         RxPktRdy set             │
          │     ACK sent by MUSBMHDRC        │
          │  EP0 interrupt generated (if enabled) │
          └──────────────┬──────────────────┘
                         │
            CPU should unload FIFO, clear RxPktRdy -
            then reload FIFO and set InPktRdy if IN Data Phase
            expected or set DataEnd if no Data Phase expected
                         │
                  ┌──────▼───────┐
                  │ Appropriate  │
                  │ Data Phase/  │
                  │OUT Transaction│
                  │ Status Phase │
                  └──────────────┘
```

CONFIDENTIAL

26.1.2. IN DATA PHASE

```
                              ┌──────────────────┐
                              │    TX State      │◄──────────────────────────┐
                              └────────┬─────────┘                            │
                                       │                                      │
                              ┌────────▼─────────┐                            │
                              │ Token sent by Host│                           │
                              │ (IN Token expected)│                          │
                              └────────┬─────────┘                            │
                                       │                                      │
                          ┌────────────▼─────────┐      ┌──────────────┐      │
   2 - 6.5                │   Valid IN Token?    │──No─►│Valid OUT Token?│─No─┤
   full-speed             └────────────┬─────────┘      └──────┬───────┘      │
   bit periods                        Yes                     Yes             │
   OR                          ┌────────▼────────┐    ┌──────────────────────┐│
   8 - 192                     │  SendStall Set? │    │SetupEnd set; TxPktRdy │ │
   high-speed                  └────────┬────────┘    │cleared                │ │
   bit periods          Yes            No             │EP0 Interrupt generated*│ │
                                        │              └──────────┬───────────┘ │
                                ┌───────▼────────┐               │             │
                                │  TxPktRdy Set? │──No─►     ┌────▼─────────┐   │
                                └───────┬────────┘          │Early Status Phase│ │
                                       Yes                  │(see next page)│   │
                                        │                   └──────────────┘   │
   ┌────────────┐   ┌──────────────┐  ┌─▼──────────────┐  ┌──────────┐         │
   │STALL sent  │   │Data1/0 packet│  │NAK sent        │  │          │         │
   │SentStall   │   │sent by       │  │                │──┼──────────┤         │
   │bit set     │   │MUSBMHDRC     │  └────────────────┘  │          │         │
   │EP0 interrupt│  └──────┬───────┘                      │          │         │
   │generated * │          │                              │          │         │
   └────────────┘   ┌──────▼───────┐                      │          │         │
   2 - 16           │ ACK          │                      │          │         │
   full-speed       │ sent by Host │                      │          │         │
   bit periods      └──────┬───────┘                      │          │         │
   OR                      │                              │          │         │
   8 - 736          ┌──────▼───────┐                      │          │         │
   high-speed       │   Valid      │──No──────────────────►          │         │
   bit periods      │ ACK received?│                                           │
                    └──────┬───────┘                                           │
                          Yes                                                  │
                    ┌──────▼────────┐                                          │
                    │TxPktRdy cleared│                                         │
                    │FIFO flushed    │                                         │
                    │EP0 interrupt   │                                         │
                    │generated *     │                                         │
                    └──────┬─────────┘                                         │
                    ┌──────▼───────┐                                           │
                    │ DataEnd set? │──No──────────────────────────────────────┤
                    └──────┬───────┘                                           │
                          Yes                                                  │
                    ┌──────▼───────┐    CPU should reload FIFO & set TxPktRdy, │
                    │Status Phase  │    plus set DataEnd if appropriate        │
                    │(see next page)│                                          │
                    └──────────────┘                                           │
   * If enabled                        CPU should clear SentStall bit ─────────┘
```

SOFT CORES

2 6 . 1 . 3 .   F O L L O W I N G   T H E   S T A T U S   P H A S E

© 2003-2006 Mentor Graphics Corporation.
5/25/2007 PSPG 40161.003-FC

26.1.4. OUT DATA PHASE

2 - 16
full-speed
bit periods
OR
8 - 736
high-speed
bit periods

2 - 6.5
full-speed
bit periods
OR
8 - 192
high-speed
bit periods

2 - 6.5
full-speed
bit periods
OR
8 - 192
high-speed
bit periods

RX State

Token sent by Host
(OUT Token expected)

Valid OUT Token? —No→ Valid IN Token? —No→ Valid PING Token? —No→

Yes

DATA1/0 packet
sent by Host

Set SetupEnd
EP0 Interrupt generated*

Early Status Phase
(see next page)

High-Speed Mode? —No→

Yes

SendStall Set?

Yes / No

RxPktRdy set? —Yes→

No

SendStall Set? —Yes→

No

DataEnd set? —Yes→

No

Valid Data Packet?
Sent within
required time? —No→

Yes

RxPktRdy set? —Yes→

No

NAK sent

Data loaded into FIFO
EP0 Interrupt generated*
RxPktRdy set & ACK sent
- or NYET if appropriate
(High-Speed mode only)

STALL sent
SentStall bit set
EP0 Interrupt
generated*

IDLE - waiting for
Status Phase

STALL sent
SentStall bit set
EP0 Interrupt
generated*

CPU should clear SendStall bit

STALL sent
SentStall bit set
EP0 Interrupt
generated*

ACK sent

NAK sent

Last packet? —No→

Yes

Status Phase
(see next page)

*CPU should unload FIFO ,
clear RxPktRdy
& set DataEnd as appropriate*

* If enabled

SOFT CORES

26.1.5. FOLLOWING THE STATUS PHASE

IDLE State, following Setup / receipt of OUT packets

Token sent by Host (IN Token expected)

Valid IN Token? — No → **Valid OUT Token?** — No →

2 - 6.5 full-speed bit periods OR 8 - 192 high-speed bit periods

Early Status Phase →

Yes ↓

SendStall Set? — Yes →

Yes ← *Late Status Stage (Protocol Stall)*

No ↓

Zero-byte DATA1 packet sent

**STALL sent
SentStall bit set
EP0 Interrupt generated***

**STALL sent
SentStall bit set
EP0 Interrupt generated ***

2 - 16 full-speed bit periods OR 8 - 736 high-speed bit periods

**IDLE -
waiting for new
Setup Phase**

ACK sent by Host

**Valid ACK?
Sent within required time?** — No →

Yes ↓

EP0 interrupt generated *

* If enabled

**IDLE - waiting for
new Setup Phase**

SOFT CORES

© 2003-2006 Mentor Graphics Corporation.

26.2. BULK/LOW-BANDWIDTH INTERRUPT TRANSACTIONS

26.2.1. IN TRANSACTION

2 - 6.5
full-speed
bit periods
OR
8 - 192
high-speed
bit periods

2 - 16
full-speed
bit periods
OR
8 - 736
high-speed
bit periods

*CPU should clear
SendStall bit*

*CPU needs to reload FIFO ,
and set TxPktRdy*

* If enabled

IDLE State

IN Token
sent by Host

Valid IN Token? — No

SendStall Set? — Yes

TxPktRdy Set? — No

STALL sent, SentStall set
FIFO flushed
TxPktRdy cleared
EP Interrupt generated *

Data0/1 packet
sent by MUSBMHDRC

NAK sent

ACK
sent by Host

Valid
ACK received? — No

TxPktRdy cleared
EP interrupt generated *

IDLE

26.2.2. OUT TRANSACTION

26.3. FULL-SPEED/LOW-BANDWIDTH ISOCHRONOUS TRANSACTIONS

26.3.1. IN TRANSACTION

26.3.2. OUT TRANSACTION

2 - 16
full-speed
bit periods
or
8 - 736
high-speed
bit periods

2 - 6.5
full-speed
bit periods
or
8 - 192
high-speed
bit periods

IDLE State

OUT Token
sent by Host

Valid OUT Token?   No

DATAx packet
sent by Host

Valid Data Packet?
Sent within
required time?   No

Yes

FIFOFull set?   Yes

No

RxPktRdy set
EP Interrupt generated *

Overrun set

*CPU needs to unload FIFO ,
and clear RxPktRdy*

IDLE

* If enabled

26.4. HIGH-BANDWIDTH TRANSACTIONS (ISOCHRONOUS/INTERRUPT)

26.4.1. IN TRANSACTION

26.4.2. OUT TRANSACTION

27. TRANSACTION FLOWS AS A HOST

27.1. CONTROL TRANSACTIONS

27.1.1. SETUP PHASE

SOFT CORES

27.1.2. IN DATA PHASE...

```
             ┌──────────────────────┐
             │  For each IN packet  │
             │ requested in SETUP   │
             │       phase          │
             └──────────────────────┘
                        │
                   ◇ ReqPkt set? ──────── No ──────────────┐
                        │                                   │
                       Yes                                  │
             ┌──────────────────────┐                       │
             │   IN token sent      │                       │
             └──────────────────────┘                       │
                        │                                   │
                   ◇ STALL          Yes    ┌──────────────┐ │
                     received? ──────────→ │ RxStall set  │ Problem in data sent
                        │                   │ ReqPkt cleared│
                       No                   │ Error Count cleared│
                                            │ Interrupt generated│
                                            └──────────────┘
```

- **For each IN packet requested in SETUP phase**
- **ReqPkt set?** — No →
- **IN token sent**
- **STALL received?** — Yes → **RxStall set / ReqPkt cleared / Error Count cleared / Interrupt generated** — *Problem in data sent*
- **NAK Limit reached?** — Yes (from NAK received?)
  - No → **Error Count cleared**
  - Yes → **NAK Timeout set / Endpoint halted / Interrupt generated**
- **NAK received?** — Yes → NAK Limit reached?
  - No →
- **DATA0/1 received?** — Yes → **ACK sent / RxPktRdy set** → **ReqPkt cleared / Error Count cleared / Interrupt generated** → **Transaction complete**
  - No → **Error Count incremented**
- **Error Count = 3?** — Yes → **Error bit set / ReqPkt cleared / Error Count cleared / Interrupt generated** — *Implies problem at peripheral end of connection.*
  - No →
- **Transaction deemed completed**

27.1.3. FOLLOWING THE STATUS PHASE

```
          ┌─────────────────────┐
          │  Completion of either│
          │    SETUP Phase or    │
          │    OUT Data Phase    │
          └─────────────────────┘
                     │
                     ▼
              ◇ ReqPkt
               and StatusPkt ──── No ──────────────────────────────►
                both set?
                     │
                    Yes
                     ▼
          ┌─────────────────────┐
          │    IN token sent     │
          └─────────────────────┘
                     │
                     ▼                    ┌──────────────────────┐
              ◇ STALL                     │    RxStall set       │   Command could not
                received? ──── Yes ─────► │   ReqPkt cleared     │   be completed
                     │                    │  Error Count cleared │────────────────────►
                    No                    │  Interrupt generated │
                     ▼                    └──────────────────────┘
```

◇ NAK Limit reached? ── No → Error Count cleared

◇ NAK received? ── Yes → NAK Limit reached?

NAK Limit reached? ── Yes → NAK Timeout set / Endpoint halted / Interrupt generated

◇ NAK received? ── No ↓

◇ DATA1 received? ── Yes → ACK sent / RxPktRdy set → ReqPkt cleared / Error Count cleared / Interrupt generated → Transaction complete

◇ DATA1 received? ── No ↓

Error Count incremented

◇ Error Count = 3? ── Yes → Error bit set / ReqPkt cleared / Error Count cleared / Interrupt generated

Implies problem at peripheral end of connection.

◇ Error Count = 3? ── No

Transaction deemed completed

CONFIDENTIAL

27.1.4. OUT DATA PHASE...

```
                        ┌─────────────────────┐
                        │  For each OUT packet │
                        │ specified in SETUP   │
                        │       phase          │
                        └─────────────────────┘
                                 │
                            ◇ TxPktRdy ◇──── No ──────────────────────┐
                            ◇  set?    ◇                              │
                                 │                                    │
                               Yes                                    │
                        ┌─────────────────┐                          │
                        │  OUT token sent │                          │
                        └─────────────────┘                          │
                        ┌─────────────────┐                          │
                        │ DATA0/1 Packet  │                          │
                        │      sent       │                          │
                        └─────────────────┘                          │
                                 │        ┌──────────────────┐       │
                            ◇  STALL   ◇  │   RxStall set    │  Command could not
                            ◇ received? ◇─Yes│ TxPktRdy cleared │  be completed ──┐
                                 │        │ Error Count      │                  │
                                No        │   cleared        │                  │
                                 │        │ Interrupt        │                  │
                                          │   generated      │                  │
                                          └──────────────────┘                  │
                                          ┌──────────────────┐                  │
                            ◇   ACK    ◇  │ TxPktRdy cleared │                  │
                            ◇ received? ◇─Yes│ Error Count      │                  │
                                 │        │   cleared        │                  │
                                No        │ Interrupt        │                  │
                                          │   generated      │                  │
                                          └──────────────────┘                  │
                                               │                                │
                                          ┌──────────┐                          │
                                          │Transaction│                         │
                                          │ complete │                          │
                                          └──────────┘                          │
          ◇ NAK Limit ◇       ◇   NAK    ◇                                      │
    No ──◇ reached?  ◇◄─Yes─◇ received? ◇                                      │
          │       │                 │                                          │
  ┌───────────┐  Yes               No                                          │
  │Error Count│   │          ┌──────────────┐                                  │
  │  cleared  │   │          │ Error Count  │                                  │
  └───────────┘   │          │ incremented  │                                  │
              ┌──────────────┐└──────────────┘                                 │
              │NAK Timeout set│      │                                         │
              │Endpoint halted│      │        ┌──────────────────┐   Implies   │
              │  Interrupt    │ ◇ Error Count◇│  Error bit set   │  problem at │
              │   generated   │ ◇   = 3?    ◇─Yes│ TxPktRdy cleared │ peripheral ─┤
              └──────────────┘      │        │ Error Count      │  end of     │
                     No             │        │   cleared        │  connection.│
                                    │        │ Interrupt        │             │
                                             │   generated      │             │
                                             └──────────────────┘             │
                                                              ┌──────────────┐│
                                                              │ Transaction  ││
                                                              │   deemed     │◄┘
                                                              │  completed   │
                                                              └──────────────┘
```

27.1.5. FOLLOWING THE STATUS PHASE

```
            Completion of either
              SETUP Phase or
               OUT Data Phase

                    │
                    ▼
                ReqPkt
              and StatusPkt ──────── No ─────────────────────────────►
               both set?

                  Yes
                    │
                    ▼
              IN token sent

                    │
                    ▼
                                                  RxStall set
                STALL                            ReqPkt cleared        Command could not
              received? ──── Yes ──►           Error Count cleared ───── be completed ─────►
                                               Interrupt generated

                    No
                    │
   NAK Limit              NAK
   reached? ◄── Yes ──  received?
  No                      No
   │                       │
   ▼                       │
 Error Count               │
  cleared                  │
   ▲   Yes                 ▼
   │    │              DATA1                    ACK sent            ReqPkt cleared
   │    ▼            received? ──── Yes ──►     RxPktRdy set ───► Error Count cleared
   │  NAK Timeout set                                            Interrupt generated
   │  Endpoint halted                                                    │
   │  Interrupt generated                                                ▼
   │                      No                                        Transaction
   │                       │                                         complete
   │                       ▼
   │                 Error Count
   │                 incremented

   │                       │
   │                       ▼                                     Implies problem at
   │                                            Error bit set    peripheral end of
   No ─────────────────  Error Count          ReqPkt cleared     connection.
                         = 3? ──── Yes ──►    Error Count cleared ─────►
                                              Interrupt generated

                                                                  Transaction deemed
                                                                     completed
```

SOFT CORES

27.2. BULK/LOW-BANDWIDTH INTERRUPT TRANSACTIONS

27.2.1. IN TRANSACTION

27.2.2. OUT TRANSACTION

```
                        ┌─────────────────────┐
                        │ Transaction scheduled│
                        └─────────────────────┘
                                  │
                                  ▼
                            ╱ TxPktRdy ╲        No
                           ╱   set?     ╲──────────────────────────────┐
                            ╲           ╱                              │
                             ╲_____╱                               │
                                  │ Yes                                │
                                  ▼                                    │
                        ┌─────────────────┐                           │
                        │  OUT token sent │                           │
                        └─────────────────┘                           │
                                  │                                    │
                                  ▼                                    │
                        ┌─────────────────┐                           │
                        │DATA0/1 Packet sent│                          │
                        └─────────────────┘                           │
                                  │                                    │
                                  ▼                     ┌──────────────────────┐
                            ╱  STALL  ╲    Yes          │ RxStall set          │   Target has
                           ╱ received? ╲───────────────▶│ TxPktRdy cleared     │   shut down pipe
                            ╲          ╱                 │ Error Count cleared  │────────────────┐
                             ╲_____╱                  │ Interrupt generated  │                │
                                  │ No                   └──────────────────────┘                │
                                  ▼                     ┌──────────────────────┐                 │
                            ╱  ACK   ╲     Yes           │ TxPktRdy cleared     │                 │
                           ╱ received?╲────────────────▶│ Error Count cleared  │                 │
                            ╲         ╱                  │ Interrupt generated  │                 │
                             ╲_____╱                  └──────────────────────┘                 │
                                  │ No                              │                             │
                                  ▼                                 ▼                             │
    ╱ NAK Limit ╲   Yes      ╱  NAK   ╲                 ┌──────────────────────┐                 │
   ╱  reached?  ╲◀──────────╱ received?╲                │ Transaction complete │                 │
    ╲          ╱            ╲         ╱                  └──────────────────────┘                 │
     ╲_____╱   No          ╲_____╱                                                           │
    No │   │                      │ No                                                            │
       │   ▼                      ▼                                                               │
┌──────────────┐         ┌─────────────────┐                                                     │
│ Error Count  │         │ Error Count     │                                                     │
│ cleared      │         │ incremented     │                                                     │
└──────────────┘         └─────────────────┘                                                     │
       ▲   │ Yes                  │                                                               │
       │   ▼                      ▼                                                               │
       │ ┌──────────────────┐  ╱ Error Count ╲   Yes    ┌──────────────────────┐                 │
       │ │ NAK Timeout set  │ ╱   = 3?        ╲────────▶│ Error bit set        │  Implies problem at
       │ │ Endpoint halted  │  ╲              ╱          │ TxPktRdy cleared     │  peripheral end of
       │ │ Interrupt gen.   │   ╲_____╱           │ Error Count cleared  │  connection.
       │ └──────────────────┘        │ No                │ Interrupt generated  │──────────────┐
       │                             │                   └──────────────────────┘              │
       └─────────────────────────────┘                                                         │
                                                                              ┌────────────────────┐
                                                                              │ Transaction deemed │
                                                                              │ completed          │
                                                                              └────────────────────┘
```

CONFIDENTIAL

27.3. FULL-SPEED / LOW-BANDWIDTH ISOCHRONOUS TRANSACTIONS

27.3.1. IN TRANSACTION

27.3.2. OUT TRANSACTION

```
        ┌─────────────────────────┐
        │   Transaction scheduled │
        └─────────────────────────┘
                    │
                    ▼
              ◇ TxPktRdy set? ◇ ──── No ────┐
                    │                        │
                   Yes                       │
                    ▼                        │
        ┌─────────────────────────┐          │
        │      OUT token sent     │          │
        └─────────────────────────┘          │
                    │                        │
                    ▼                        │
        ┌─────────────────────────┐          │
        │       DATA0 sent        │          │
        └─────────────────────────┘          │
                    │                        │
                    ▼                        │
        ┌─────────────────────────┐          │
        │    TxPktRdy cleared      │          │
        │   Interrupt generated    │          │
        └─────────────────────────┘          │
                    │◄───────────────────────┘
                    ▼
        ┌─────────────────────────┐
        │      Transaction         │
        │       complete           │
        └─────────────────────────┘
```

CONFIDENTIAL

SOFT CORES

27.4. HIGH-BANDWIDTH TRANSACTIONS (ISOCHRONOUS/INTERRUPT)

27.4.1. IN TRANSACTION

27.4.2.   OUT TRANSACTION

```
        ┌─────────────────────────┐
        │  Transaction scheduled  │
        └─────────────────────────┘
                    │
                    ▼
              ╱ TxPktRdy ╲ ──────No──────────────────┐
              ╲  set?   ╱                             │
                 │                                    │
                Yes                                   │
                 │                                    │
        ┌─────────────────┐                           │
        │  OUT token sent │                           │
        └─────────────────┘                           │
                 │                                     │
                 ▼                                     │
          ╱ Response  ╲                ┌─────────────┐ │
         ╱ received from ╲ ──No──────► │ IncompTx set│─┤
         ╲ device?       ╱             └─────────────┘ │
          ╲(Interrupt Tx)╱                             │
                 │                                     │
                Yes                                    │
                 │                                     │
         ╱ 3 subpackets ╲ ──Yes─────────────┐         │
         ╲ waiting to be ╱                   │         │
          ╲   sent?     ╱                    │         │
                 │                           │         │
                No                           │         │
                 │                           │         │
         ╱ 2 subpackets ╲ ──Yes───┐          │         │
         ╲ waiting to be ╱         │          │         │
          ╲   sent?     ╱          │          │         │
                 │                 │          │         │
                No                 │          │         │
                 │                 │          │         │
   ┌──────────┐  ▼   ┌──────────┐  ▼  ┌──────────┐     │
   │  Data2   │      │  Data1   │     │  Data0   │     │
   │  packet  │─────►│  packet  │────►│  packet  │     │
   │  sent    │      │  sent    │     │  sent    │     │
   └──────────┘      └──────────┘     └──────────┘     │
                                           │           │
                                           ▼           │
                              ┌──────────────────┐     │
                              │ TxPktRdy cleared  │     │
                              │ Interrupt generated│    │
                              └──────────────────┘     │
                                           │           │
                                           ▼           │
                                  ┌─────────────┐      │
                                  │ Transaction │◄─────┘
                                  │ complete    │
                                  └─────────────┘
```

27.5. DMA OPERATIONS (WITH BUILT IN DMA CONTROLLER)

27.5.1. SINGLE PACKET TX

```
                        ┌────────────────────┐
                        │     IDLE State     │
                        └────────────────────┘
                                  │
                        ┌────────────────────────────┐
                        │   Set IntrTxE.Dn = 1        │
                        │   Set TxCSR.D12 (DMAReqEnab) = 0 │
                        └────────────────────────────┘
                                  │
                        ┌────────────────────────────────┐
                        │ Set DMA registers as follows:  │
                        │ Set ADDR = Address of pkt to send │
                        │ Set COUNT = Size of pkt to be sent │
                        │ Set CNTL.D0, CNTL.D1, CNTL.D3 = 1 │
                        │ Set CNTL.D2 = 0; CNTL[D10,9] as required │
                        └────────────────────────────────┘
                                  │
                        ┌────────────────────────────┐
                        │  DMA Controller requests bus │
                        └────────────────────────────┘
                                  │
                              ◇ Is AHB_HGRANT high? ◇ ──No──┐
                                  │ Yes                      │
                        ┌────────────────────────────┐       │
                        │ DMA Controller reads from ADDR │    │
                        │   and writes to FIFO        │       │
                        └────────────────────────────┘       │
                                  │                           │
                              ◇ On DMA_NINT low,          ──No──┘
                                 is INTR.channel = 1 ◇
                                  │ Yes
                        ┌────────────────────┐
                        │    Set TxPktRdy    │
                        └────────────────────┘
                                  │
                        ┌────────────────────┐
                        │   Continue as for  │
                        │  Bulk IN Transaction │
                        └────────────────────┘
```

Actions carried out by built-in DMA Controller

27.5.2. SINGLE PACKET RX

IDLE State

Set IntrRxE.D*n* = 1
Set RxCSR.D13 (DMAReqEnab) = 0

*Wait for packet to be received
as per BULK OUT TRANSACTION*

On MC_NINT=0,
is IntrRx.D*n* = 1? — No

Yes

Set DMA registers as follows:
Set ADDR = Address to store pkt
Set COUNT = Size of pkt (from RxCount)
Set CNTL.D0, CNTL.D3 = 1
Set CNTL.D1, CNTL.D2 = 0
Set CNTL[D10,9] as required

DMA Controller requests bus

*Actions carried out by
built-in DMA Controller*

Is AHB_HGRANT high? — No

Yes

DMA Controller reads from FIFO
and writes to ADDR

On DMA_NINT low,
is INTR.*channel* = 1 — No

Yes

Clear RxPktRdy

IDLE State

CONFIDENTIAL

27.5.3. MULTIPLE PACKET TX

```
                    ┌──────────────────────┐
                    │      IDLE State      │
                    └──────────────────────┘
                               │
                    ┌──────────────────────┐
                    │  Set IntrTxE.Dn = 1   │
                    │ Set TxCSR.D15 (AutoSet) = 1│
                    │ Set TxCSR.D12 (DMAReqEnab) = 1│
                    │ Set TxCSR.D10 (DMAReqMode) = 1│
                    └──────────────────────┘
                               │
                    ┌──────────────────────┐
                    │ Set DMA registers as follows:│
                    │ Set ADDR = Address of data to send│
                    │ Set COUNT = Amount of data to be sent│
                    │ Set CNTL.D0, CNTL.D1 = 1│
                    │ Set CNTL.D2, CNTL.D3 = 1│
                    │ Set CNTL[D10,9] as required│
                    └──────────────────────┘
```

Is DMA_REQ[*n-1*] high? — No

Yes

DMA Controller requests bus

Is AHB_HGRANT high? — No

Yes

Actions carried out by built-in DMA Controller

DMA Controller reads from ADDR, writes to FIFO and decrements COUNT

Is COUNT = 0? — Yes → DMA Controller sets DMA_NINT

No

TxPktRdy set and packet processed as for BULK IN Transaction

Set TxPktRdy
(in general case)

Last packet sent
(in general case)

IDLE State

CONFIDENTIAL

SOFT CORES

27.5.4. MULTIPLE PACKET RX

If Size of Data Block Known:

```
                    ┌─────────────────┐
                    │   IDLE State    │
                    └─────────────────┘
                             │
          ┌──────────────────────────────────────┐
          │ Set IntrRxE.Dn = 0                    │
          │ Set RxCSR.D15 (AutoClear) = 1         │
          │ Set RxCSR.D13 (DMAReqEnab) = 1        │
          │ Set RxCSR.D11 (DMAReqMode) = 0        │
          │ (If Host: Also set RxCSR.D14 (AutoReq) = 1) │
          └──────────────────────────────────────┘
                             │
          ┌──────────────────────────────────────┐
          │ Set DMA registers as follows:         │
          │ Set ADDR = Address to store data      │
          │ Set COUNT = Amount of data            │
          │ Set CNTL.D0, CNTL.D2, CNTL.D3 = 1     │
          │ Set CNTL.D1 = 0                       │
          │ Set CNTL[D10,9] as required           │
          └──────────────────────────────────────┘
```

Actions carried out by built-in DMA Controller

- Is COUNT = 0? — Yes → DMA Controller asserts DMA_NINT → If necessary, clear RxPktRdy → IDLE State
- No → Is DMAReq[m] high? — No (loop) / Yes → DMA Controller requests bus
- Is AHB_HGRANT high? — No (loop) / Yes → DMA Controller reads from FIFO, writes to ADDR and decrements COUNT
- RxPktRdy cleared (unless packet less than RxMaxP)

CONFIDENTIAL

If Size of Data Block Not Known:

```
                            ┌──────────────────┐
                            │    IDLE State     │
                            └──────────────────┘
                                     │
        ┌────────────────────────────────────────────────────┐
        │ Set IntrRxE.Dn = 1                                  │
        │ Set RxCSR.D15 (AutoClear) = 1                       │
        │ Set RxCSR.D13 (DMAReqEnab) = 1                      │
        │ Set RxCSR.D11 (DMAReqMode) = 1                      │
        │ (If Host: Also set RxCSR.D14 (AutoReq) = 1)         │
        └────────────────────────────────────────────────────┘
                                     │
        ┌────────────────────────────────────────────────────┐
        │ Set DMA registers as follows:                       │
        │ Set ADDR = Address to store data                    │
        │ Set COUNT = Size of buffer                          │
        │ Set CNTL.D0, CNTL.D2, CNTL.D3 = 1                   │
        │ Set CNTL.D1 = 0                                     │
        │ Set CNTL[D10,9] in line with MaxP                   │
        └────────────────────────────────────────────────────┘
```

Is Packet Size = RxMaxP? — No → MUSBMHDRC asserts Rx Endpoint Interrupt

Yes

Is DMAReq[m] high? — No

Yes

DMA Controller requests bus

Is AHB_HGRANT high? — No

Yes

DMA Controller reads from FIFO, writes to ADDR and decrements COUNT

RxPktRdy cleared

Actions carried out by built-in DMA Controller

On MC_NINT = 1, Is IntrRx[n] = 1? — No

Yes

Read packet from FIFO
Clear RxPkyRdy

IDLE State

CONFIDENTIAL

28. TEST MODES

The MUSBMHDRC 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 and the RAM block.

The test modes are entered by writing to the TestMode register (address 0Fh). 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 that the status phase has completed) then write to the TestMode register.

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

28.1. TEST_SE0_NAK

To enter the Test_SE0_NAK test mode, the software should set the Test_SE0_NAK bit by writing 8'h01 to the TestMode register. The MUSBMHDRC will then go into a mode in which it responds to any valid IN token with a NAK.

28.2. TEST_J

To enter the Test_J test mode, the software should set the Test_J bit by writing 8'h02 to the TestMode register. The MUSBMHDRC will then go into a mode in which it transmits a continuous J on the bus.

28.3. TEST_K

To enter the Test_K test mode, the software should set the Test_K bit by writing 8'h04 to the TestMode register. The MUSBMHDRC will then go into a mode in which it transmits a continuous K on the bus.

28.4. TEST_PACKET

To execute the Test_Packet test, the software should:

(i) Start a session (if the core is being used in Host mode).

(ii) Write the standard test packet (shown below) to the Endpoint 0 FIFO.

(iii) Write 7'h08 to the TestMode register to enter Test_Packet test mode.

(iv) Set the TxPktRdy bit in the CSR0 register (D1).

The 53 byte test packet to load is as follows (all bytes in hex). The test packet only has to be loaded once; the MUSBMHDRC 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 MUSBMHDRC will add the DATA0 PID to the head of the data sequence and the CRC to the end.

CONFIDENTIAL

28.5. FIFO_ACCESS

The FIFO Access test mode allows the user to test the operation of CPU Interfaceand 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 bi-directional 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 using either CPU accesses.

2. Set TxPktRdy (CSR0L.D1).

3. Write 8'h40 to the Testmode register.

4. Unload the packet from the Endpoint Rx FIFO, again using either CPU accesses.

5. Set ServicedRxPktRdy (CSR0L.D6).

Writing 0x40 to the Testmode register causes the following sequence of events:

1. The Endpoint 0 CPU pointer (which records the number of bytes to be transmitted) is copied to the Endpoint 0 USB pointer (which records the number of bytes received).

2. The Endpoint 0 CPU pointer is reset.

3. TxPktRdy (CSR0L.D1) is cleared.

4. RxPktRdy (CSR0L.D0) 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 been flushed and the same packet received over the USB. The data that was loaded into the Tx FIFO can now be read out of the Rx FIFO.

28.6. FORCE_HOST

The Force Host test mode enables the user to instruct the core to operate in Host mode, regardless of whether it is actually connected to any peripheral i.e. 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 bit 7 of the DevCtl register.)

This mode, which is selected by setting bit 7 within the Testmode register, 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 will enter Host mode when the Session bit is set and remain in Host mode until the Session bit is cleared even if a connected device is disconnected during the session. The operating speed while in this mode is determined from the settings of the Force_HS and Force_FS bits of the Testmode register, as detailed in Section 3.2.11.

29. HARDWARE READBACK

The MUSBMHDRC includes facilities for reading back information about the core configuration and the version of the core RTL from which the core was created.

29.1. HARDWARE CONFIGURATION READBACK

Various details about how the MUSBMHDRC core has been configured is available from the core's ConfigData register

CONFIDENTIAL

(described in Section 3.3.5), while such information as the number of Tx and Rx endpoints have been configured and the width of the RAM address bus can be obtained from the EPInfo and RAMInfo registers (see Sections 3.7.1 and 3.7.2). In addition, details of the size of FIFO associated with each endpoint and whether the FIFO is shared between a Tx endpoint and an Rx endpoint may be obtained from the FIFOSize register (see Section 3.3.18).

The ConfigData and FIFOSize registers are both included among the core's set of Indexed registers and are both located at offset 0x1F. When the register at this location is read for Endpoint 0 (i.e. with Index register set to 0), it returns the following ConfigData information:

Bit	Value	Interpretation
D7	0/1	When set to '1' indicates that automatic combining of Bulk packets has been selected.
D6	0/1	When set to '1' indicates that automatic splitting of Bulk packets has been selected.
D5	0/1	Always "0" for Little Endian.
D4	0/1	When set to '1' indicates High-bandwidth Rx ISO Endpoint Support selected.
D3	0/1	When set to '1' indicates High-bandwidth Tx I SO Endpoint Support selected.
D2	0/1	When set to '1' indicates Dynamic FIFO Sizing option selected.
D1	0/1	Always '1' for Soft Connect/Disconnect.
D0	0/1	Always "0" for UTMI+ data width of 8 bits.

When read for endpoint numbers 1 – 15, the register returns details of the FIFOs for which the corresponding Tx and Rx endpoints have been configured – as follows: (For Endpoint 0, there is a single 64-byte FIFO used for both Rx and Tx transactions.)

Bits	Value	Interpretation
0 – 3	0	No Tx endpoint with this endpoint number
	3 – 11	TxFIFO size = 2^n (8 – 2048 bytes)
4 – 7	0	No Rx endpoint with this endpoint number
	3 – 11	RxFIFO size = 2^n (8 – 2048 bytes)
	15	Tx and Rx endpoints share the same FIFO (size given by bits 0 – 3)

Note: The FIFOSize register does not return valid information where the Dynamic FIFO Sizing option is used.

29.2. RTL VERSION READBACK

Details concerning the version of the RTL from which a particular implementation of the MUSBMHDRC core was created can be read back from the HWVers register.

This register, which is located at 6Ch, chiefly records the version number of the RTL from which the core was created. These version numbers take the form *xx.yyy* where *xx* is the major revision number and *yyy* is the minor revision number. The major revision number is recorded in bits 14 – 10 of the HWVers register, while the minor revision number is recorded in bits 9 – 0 of this register.

The top bit of the register is used by the core to indicate where a core has been created from a 'Release Candidate' rather than a full release of the core. Release Candidates should not be used for anything other than testing purposes. They should not be used to create finished silicon.

CONFIDENTIAL

30. REVISION HISTORY

30.1. ISSUE 1

1-Aug-06 – Combined contents of the MUSBMHDRC Programmer's Guide into the MUSBMHDRC Product Specification.

30.2. ISSUE 2

17 October 17, 2006. Added content for internal DMA. Moved all register description to section 3.

30.3. ISSUE 3

25 May 2007 – Release 1.900; Modified for the following defects.

- dts0100383116 - Make the compiler directive C_T_HSBT programmable

- dts0100398841 - clean up Internal DMA documentation that implies that the start address can be non word boundaries.

- dts0100402572 - The DMA Load/Unload Timing Diagrams in section 20.5.2 of musbmhdrc_pspg.pdf are incorrect.

- dts0100402896 - The clock synchronization diagram in section 4.2 is incorrect.

- dts0100386440 - ULPI Link AN incorrectly references MUSBMHDRC_ulpictl not MUSBMHDRC_lpictl

CONFIDENTIAL

CONFIDENTIAL

5/25/2007 PSPG 40161.003-FC