BUS INTERFACE

MENTOR GRAPHICS

THE INTELLIGENT APPROACH TO INTELLECTUAL PROPERTY

MUSBFDRC

USB FULL-SPEED DUAL-ROLE CONTROLLER

Programmer's Guide

CONFIDENTIAL

SOFT CORES

11/07 PG-40134.005-FC

Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070.

For Customer Support on this product:

- Call up the Mentor Graphics Customer Inquiry Service at **http://www.mentor.com/supportnet**
- Email **support_net@mentor.com**
- Phone **1-877-763-8470** (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 the Mentor Graphics Soft Cores CD. Please note the checklists of actions to take and information to have to hand when contacting Mentor Graphics Customer Support that are given in the Customer Support Handbook.

TABLE OF CONTENTS

1. INTRODUCTION 6

2. REGISTER DESCRIPTION 7

2.1. MUSBFDRC Register Map 7

2.2. Common Registers 10

2.2.1. FAddr 10

2.2.2. Power 10

2.2.3. IntrTx1 12

2.2.4. IntrTx2 12

2.2.5. IntrRx1 13

2.2.6. IntrRx2 13

2.2.7. IntrUSB 14

2.2.8. IntrTx1E and IntrTx2E 15

2.2.9. IntrRx1E and IntrRx2E 15

2.2.10. IntrUSBE 15

2.2.11. Frame1 16

2.2.12. Frame2 16

2.2.13. Index 16

2.2.14. DevCtl 17

2.3. Indexed Registers 18

2.3.1. CSR0 18

2.3.2. CSR02 19

2.3.3. Count0 20

2.3.4. NAKLimit0 (Host mode only) 20

2.3.5. TxMaxP 20

2.3.6. TxCSR1 21

2.3.7. TxCSR2 22

2.3.8. RxMaxP 23

2.3.9. RxCSR1 23

2.3.10. RxCSR2 25

2.3.11. RxCount1 25

2.3.12. RxCount2 26

2.3.13. TxType (Host mode only) 26

2.3.14. TxInterval (Host mode only) 26

2.3.15. RxType (Host mode only) 27

2.3.16. RxInterval (Host mode only) 27
2.3.17. FIFOSize 28
2.4. FIFOx (Address 20h – 2Fh) 28
3. DYNAMIC FIFO SIZING 29
4. USB INTERRUPT HANDLING 31
5. SUSPEND/RESUME 32
5.1. MUSBFDRC Active During Suspend 32
5.2. MUSBFDRC Inactive During Suspend 32
5.3. Remote Wakeup 32
6. USB RESET (Peripheral mode only) 32
7. CONTROL TRANSACTIONS (via ENDPOINT 0) 33
7.1. Control Transactions As a Peripheral 33
7.1.1. Zero Data Requests 33
7.1.2. Write Requests 33
7.1.3. Read Requests 34
7.1.4. Endpoint 0 States 35
7.1.5. Endpoint 0 Service Routine as Peripheral 38
7.1.6. Error Handling as a Peripheral 43
7.2. Control Transactions as a Host 43
7.2.1. SETUP Phase as a Host 44
7.2.2. IN Data Phase as a Host 44
7.2.3. OUT Data Phase as a Host 45
7.2.4. IN Status Phase as a Host (following either the SETUP Phase or an OUT Data Phase) 45
7.2.5. OUT Status Phase as a Host (following IN Data Phase) 45
8. BULK TRANSACTIONS 46
8.1. Handling Bulk Transactions As a Peripheral 46
8.1.1. Bulk IN Transactions as a Peripheral 46
8.1.2. Bulk OUT Transactions as a Peripheral 47
8.2. Handling Bulk Transactions As a Host 49
8.2.1. Bulk IN Transaction as a Host 49
8.2.2. Bulk OUT Transaction 51
9. INTERRUPT TRANSACTIONS 53
9.1. Interrupt transactions as a Peripheral 53
9.2. Interrupt transactions as a Host 53
10. ISOCHRONOUS TRANSACTIONS 53

10.1. Handling Isochronous Transactions As a Peripheral 53
10.1.1. Isochronous IN Transactions 53
10.1.2. Isochronous OUT Transactions 55
10.2. Handling Isochronous Transactions As a Host 57
10.2.1. Isochronous IN Transactions 57
10.2.2. Isochronous OUT Transactions 58
11. CONNECT/DISCONNECT 60
11.1. In Host Mode 60
11.2. In Peripheral Mode 60
12. OTG SESSION REQUEST 60
12.1. When MUSBFDRC is the 'A' Device 60
12.2. When MUSBFDRC is the 'B' Device 61
13. HOST NEGOTIATION 61
14. VBUS EVENTS 62
15. TRANSACTION FLOWS AS A PERIPHERAL 63
15.1. Control Transactions 63
15.2. Bulk/Interrupt Transactions 68
15.3. Isochronous Transactions 70
16. TRANSACTION FLOWS AS A HOST 72
16.1. Control Transactions 72
16.2. Bulk/Interrupt Transactions 77
16.3. Isochronous Transactions 79
17. HARDWARE CONFIGURATION READBACK 80
18. REVISION HISTORY 81
18.1. Issue 1 81
18.2. Issue 2 81
18.3. Issue 3 81
18.4. Issue 4 81
18.5. Issue 5 81
18.6. Issue 6 81

1. INTRODUCTION

The Mentor Graphics MUSBFDRC soft core provides a 'Dual-role' USB controller for use both as a full-speed (12 Mbps) function controller in a USB peripheral and as either the host or the peripheral in point-to-point communications with another USB function (either full- or low-speed).

It complies both with the USB standard for 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 portable devices such as mobile phones, PDAs, digital still cameras and MP3 players with a low-cost connectivity solution. The specification defines the limited host capability which a device requires to act as the host for point-to-point sessions with another similar device. It also defines protocols for sessions to be requested by either of the two devices and for 'Dual-role' devices to switch between host and peripheral roles in such sessions.

The MUSBFDRC is user-configurable for up to 15 'Transmit' endpoints and/or up to 15 'Receive' endpoints in addition to Endpoint 0. (Whether these endpoints are used for IN transactions or for OUT transactions at any time will depend on whether the device containing the MUSBFDRC is currently being used as a USB peripheral or as the host for point-to-point communications with another USB peripheral.) These additional endpoints can further be individually programmed for Bulk/Interrupt or Isochronous transfers.

Each endpoint requires a FIFO to be associated with it. The MUSBFDRC 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 2048 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.

The MUSBFDRC is offered with a choice of high-level CPU interfaces. In the base model, access to the FIFOs and the internal control/status registers is via an 8-bit PVCI[1]-compatible synchronous CPU interface. However, the MUSBFDRC can readily be connected to a range of standard buses through bridges such as the MUSBFDRC – AHB bridge provided with the core (and described in the separate MUSBFSFC/MUSBFDRC – AHB Bridge specification provided in the **musbfdrc/docs** directory as **musbfsfc_ahb_an.pdf**).

There is also support for DMA access to the Endpoint FIFOs (this is described in Section 9 of the MUSBFDRC User Guide, provided in the **musbfdrc/docs** directory as **musbfdrc_pu.pdf**). The optional bridges may also offer DMA support in which case this is described in the appropriate Bridge Specification.

The MUSBFDRC provides all the encoding, decoding and checking needed in sending and receiving USB packets – interrupting the CPU only when endpoint data has been successfully transferred. When acting as the host for point-to-point communications, the MUSBFDRC additionally maintains a frame counter and automatically schedules SOF, Isochronous, Interrupt and Bulk transfers. It also includes support for the Session Request Protocol (SRP) and Host Negotiation Protocol (HNP) required for point-to-point communications, details of which are given in the *USB On-The-Go* supplement to the USB 2.0 specification.

A graphical user interface is provided for configuring the core to the user's requirements. This is described in the MUSBFDRC User Guide.

This document describes the software interface to the MUSBFDRC and gives examples of how a typical application should drive the core to produce a USB-compliant function.

[1] Peripheral Virtual Component Interface, as defined by the VSI Alliance™ Virtual Component Interface Standard (OCB 2 1.0).

2. REGISTER DESCRIPTION

2.1. MUSBFDRC REGISTER MAP

The MUSBFDRC register map is split into three sections:

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

Endpoint Control/Status registers (10h–1Fh) – These registers provide control and status for the endpoints. 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–2Fh) – This address range provides access to the endpoint FIFOs.

Note: Any additional registers associated with any bridge provided for use with the MUSBFDRC core or any changes to the following registers that result from using this bridge are described in the separate specification for the bridge included in the **musbfdrc/docs** directory.

MUSBFDRC REGISTER MAP: Common USB registers			
ADDR	**NAME**	**DESCRIPTION**	**See Section**
00	FAddr	Function address register.	2.2.1
01	Power	Power management register.	2.2.2
02	IntrTx1	Interrupt register for Endpoint 0 plus Tx Endpoints 1 to 7.	2.2.3
03	IntrTx2	Interrupt register for Tx Endpoints 8 to 15.	2.2.4
04	IntrRx1	Interrupt register for Rx Endpoints 1 to 7.	2.2.5
05	IntrRx2	Interrupt register for Rx Endpoints 8 to 15.	2.2.6
06	IntrUSB	Interrupt register for common USB interrupts.	2.2.7
07	IntrTx1E	Interrupt enable register for IntrTx1.	2.2.8
08	IntrTx2E	Interrupt enable register for IntrTx2.	2.2.8
09	IntrRx1E	Interrupt enable register for IntrRx1.	2.2.9
0A	IntrRx2E	Interrupt enable register for IntrRx2.	2.2.9
0B	IntrUSBE	Interrupt enable register for IntrUSB.	2.2.10
0C	Frame1	Frame number bits 0 to 7.	2.2.11
0D	Frame2	Frame number bits 8 to 10.	2.2.12
0E	Index	Index register for selecting the endpoint status and control registers.	2.2.13
0F	DevCtl	USB device control register.	2.2.14

MUSBFDRC REGISTER MAP: Indexed registers – Peripheral mode (Control Status registers for endpoint selected by the Index register when DevCtl.D2 = 0)			
ADDR	**NAME**	**DESCRIPTION**	**See Section**
10	TxMaxP	Maximum packet size for peripheral IN endpoint. (Index register set to select Endpoints 1 – 15 only)	2.3.4
11	CSR0	Control Status register for Endpoint 0. (Index register set to select Endpoint 0)	2.3.1
	TxCSR1	Control Status register 1 for peripheral IN endpoint. (Index register set to select Endpoints 1 – 15)	2.3.6
12	CSR02	Subsidiary Control Status register for Endpoint 0, containing FlushFIFO bit. (Index register set to select Endpoint 0)	2.3.2
	TxCSR2	Control Status register 2 for peripheral IN endpoint. (Index register set to select Endpoints 1 – 15 only)	2.3.7
13	RxMaxP	Maximum packet size for peripheral OUT endpoint. (Index register set to select Endpoints 1 – 15 only)	2.3.8
14	RxCSR1	Control Status register 1 for peripheral OUT endpoint. (Index register set to select Endpoints 1 – 15 only)	2.3.9
15	RxCSR2	Control Status register 2 for peripheral OUT endpoint. (Index register set to select Endpoints 1 – 15 only)	2.3.10
16	Count0	Number of received bytes in Endpoint 0 FIFO. (Index register set to select Endpoint 0)	2.3.2
	RxCount1	Number of bytes in peripheral OUT endpoint FIFO (lower byte). (Index register set to select Endpoints 1 – 15)	2.3.11
17	RxCount2	Number of bytes in peripheral OUT endpoint FIFO (upper byte). (Index register set to select Endpoints 1 – 15 only)	2.3.12
18–1B	–	*Reserved.* Value returned affected by use in Host mode (see overleaf).	
1C–1E	–	*Only used if Dynamic FIFO sizing option is selected. Otherwise always return zero.*	*3*
1F	FIFOSize	Returns the configured size of the selected Rx FIFO and Tx FIFOs. (Index register set to select Endpoints 1 – 15 only) *Note: Not valid when Dynamic FIFO sizing is selected (when register has different interpretation.)*	2.3.17

MUSBFDRC REGISTER MAP: Indexed registers – Host mode (Control Status registers for endpoint selected by the Index register when DevCtl.D2 = 1)			
ADDR	**NAME**	**DESCRIPTION**	**See Section**
10	TxMaxP	Maximum packet size for host OUT endpoint. (Index register set to select Endpoints 1 – 15 only)	2.3.4
11	CSR0	Control Status register for Endpoint 0. (Index register set to select Endpoint 0)	2.3.1
	TxCSR1	Control Status register 1 for host OUT endpoint. (Index register set to select Endpoints 1 – 15)	2.3.6
12	CSR02	Subsidiary Control Status register for Endpoint 0, containing FlushFIFO bit. (Index register set to select Endpoint 0)	2.3.2
	TxCSR2	Control Status register 2 for host OUT endpoint. (Index register set to select Endpoints 1 – 15 only)	2.3.7
13	RxMaxP	Maximum packet size for host IN endpoint. (Index register set to select Endpoints 1 – 15 only)	2.3.8
14	RxCSR1	Control Status register 1 for host IN endpoint. (Index register set to select Endpoints 1 – 15 only)	2.3.9
15	RxCSR2	Control Status register 2 for host IN endpoint. (Index register set to select Endpoints 1 – 15 only)	2.3.10
16	Count0	Number of received bytes in Endpoint 0 FIFO. (Index register set to select Endpoint 0)	2.3.2
	RxCount1	Number of bytes in host IN endpoint FIFO (lower byte). (Index register set to select Endpoints 1 – 15)	2.3.11
17	RxCount2	Number of bytes in host IN endpoint FIFO (upper byte). (Index register set to select Endpoints 1 – 15 only)	2.3.12
18	TxType	Sets the transaction protocol and peripheral endpoint number for the host OUT endpoint. (Index register set to select Endpoints 1 – 15 only)	2.3.13
19	NAKLimit0	Sets the NAK response timeout on Endpoint 0. (Index register set to select Endpoint 0)	2.3.4
	TxInterval	Sets the polling interval for an OUT Interrupt endpoint, in ms. (Index register set to select Endpoints 1 – 15 only)	2.3.14
1A	RxType	Sets the transaction protocol and peripheral endpoint number for the host IN endpoint. (Index register set to select Endpoints 1 – 15 only)	2.3.15
1B	RxInterval	Sets the polling interval for an IN Interrupt endpoint, in ms. (Index register set to select Endpoints 1 – 15 only)	2.3.16
1C–1E	–	*Only used if Dynamic FIFO sizing option is selected. Otherwise always return zero.*	*3*
1F	FIFOSize	Returns the configured size of the selected Rx FIFO and Tx FIFOs. (Index register set to select Endpoints 1 – 15 only) *Note: Not valid when Dynamic FIFO sizing is selected (when register has different interpretation.)*	2.3.17

MUSBFDRC REGISTER MAP: FIFOs			
ADDR	**NAME**	**DESCRIPTION**	**See Section**
20– 2F	FIFOx	FIFOs for Endpoints 0 – 15.	2.4

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.

2.2. COMMON REGISTERS

– described in Address order.

2.2.1. 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 MUSBFDRC is being used in Host mode (DevCtl.D2=1), this register should be set to the value sent in a SET_ADDRESS command during device enumeration as the address for the peripheral device.

When the MUSBFDRC 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.

Note: The new address takes immediate effect, so software should not update the FAddr register until the current transfer has completed.

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.

2.2.2. POWER

Power is an 8-bit register that is used for controlling Suspend and Resume signaling, IN packet timing for Isochronous endpoints and, for point-to-point communications, to indicate the type of device connected.

Address: 01h; *Reset value:* 8'h00

		D7	D6	D5	D4	D3	D2	D1	D0
		ISO Update	**VBus Val**	**VBus Sess**	**VBus Lo**	**Reset**	**Resume**	**Suspend Mode**	**Enable Suspend**
Peripheral mode	*CPU*	rw	r	r	r	r	rw	r	rw
	USB	r	rw	rw	rw	rw	r	set	r
Host mode	*CPU*	r	r	r	r	rw	rw	set	r
	USB	r	rw	rw	rw	rw	r	clear	r

Bit	Name	Function
D7	**ISO Update**	Set by the CPU to cause the MUSBFDRC to wait for a SOF token after TxPktRdy has been set before sending the packet. If an IN token is received before a SOF token, a zero length data packet will be sent. *This bit is only valid when the MUSBFDRC is in Peripheral mode and then only affects endpoints performing Isochronous transfers.*
D6	**VBus Val**	This bit indicates the state of the VBUSVAL input signal to the MUSBFDRC. 1 ⇒ VBus above the VBus Valid threshold, 0 ⇒ VBus below this threshold.
D5	**VBus Sess**	This bit indicates the state of the VBUSSES input signal to the MUSBFDRC. 1 ⇒ VBus above the Session Valid threshold for a 'B' device, 0 ⇒ VBus below this threshold.
D4	**VBus Lo**	This bit indicates the state of the VBUSLO input signal to the MUSBFDRC. 1 ⇒ VBus above the Session End threshold, 0 ⇒ VBus below this threshold.
D3	**Reset**	In Peripheral mode, this read-only bit is set while Reset signaling is present on the bus. In Host mode, this bit is set by the CPU for 20 ms, to generate Reset signaling on the bus. It is automatically set when Host mode is entered following Host Negotiation (see Section 13): the CPU should then clear it after 20 ms.
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 Peripheral mode, this bit is set by the USB when Suspend mode is entered. In Host mode, this bit is set by the CPU when Suspend mode is to be entered. The bit is cleared when the CPU reads the Interrupt register or sets the Resume bit (above) or leaves Host mode.
D0	**Enable Suspend**	Set by the CPU to enable entry into Suspend mode when Suspend signaling is received on the bus. *Note: This bit is only valid when the MUSBFDRC is in Peripheral mode.*

The **ISO Update** bit affects all IN Isochronous endpoints in the MUSBFDRC. It is normally used as a method of ensuring a "clean" start-up of an IN Isochronous pipe. See the section on IN Isochronous Endpoints (Section 10) for more details on starting up IN Isochronous pipes.

The **Vbus Val**, **VBus Sess** and **VBus Lo** bits indicate the state of the VBUSVAL, VBUSSES and VBUSLO signals respectively. External comparators switch these signals between low and high as the level of VBus goes above or below levels corresponding to the VBus Valid, Session Valid and Session End thresholds for the design. The state of these bits is used in particular to spot the start or end of point-to-point communications initiated by the companion device.

The **Reset** bit can be used to determine when Reset signaling is present on the USB. It will go high when Reset signaling is detected and remain high until the bus reverts to an idle state.

The **Resume** bit is used to force the MUSBFDRC to generate Resume signaling on the USB to perform remote wake-up from Suspend mode. Once set high, it should be left high for approximately 10 ms (at least 1 ms and no more than 15 ms), then cleared.

The **Suspend Mode** bit is set by the MUSBFDRC when Suspend mode is entered. It will be cleared when the IntrUSB register is read (as a result of receiving a Suspend interrupt). It will also be cleared if Suspend mode is left by setting the Resume bit to initiate a remote wake-up.

The **Enable Suspend** bit is set to enable entry into Suspend mode when Suspend signaling is received on the bus.

2.2.3. INTRTX1

IntrTx1 is an 8-bit read-only register that indicates which interrupts are currently active for Endpoint 0 and Tx Endpoints 1–7. *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:* 8'h00

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

Bit	Name	Function
D7	**EP7**	Tx Endpoint 7 interrupt.
D6	**EP6**	Tx Endpoint 6 interrupt.
D5	**EP5**	Tx Endpoint 5 interrupt.
D4	**EP4**	Tx Endpoint 4 interrupt.
D3	**EP3**	Tx Endpoint 3 interrupt.
D2	**EP2**	Tx Endpoint 2 interrupt.
D1	**EP1**	Tx Endpoint 1 interrupt.
D0	**EP0**	Endpoint 0 interrupt.

2.2.4. INTRTX2

IntrTx2 is an 8-bit read-only register that indicates which of the interrupts for Tx Endpoints 8 – 15 are currently active. *Note:* This register will only be present if the MUSBFDRC is configured for more than 7 Tx endpoints and 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: 03h*; Reset value:* 8'h00

	D7	D6	D5	D4	D3	D2	D1	D0
	EP15	**EP14**	**EP13**	**EP12**	**EP11**	**EP10**	**EP9**	**EP8**
From CPU	r	r	r	r	r	r	r	r
From USB	set	set	set	set	set	set	set	set

Bit	Name	Function
D7	**EP15**	Tx Endpoint 15 interrupt.
D6	**EP14**	Tx Endpoint 14 interrupt.
D5	**EP13**	Tx Endpoint 13 interrupt.
D4	**EP12**	Tx Endpoint 12 interrupt.

Bit	Name	Function
D3	**EP11**	Tx Endpoint 11 interrupt.
D2	**EP10**	Tx Endpoint 10 interrupt.
D1	**EP9**	Tx Endpoint 9 interrupt.
D0	**EP8**	Tx Endpoint 8 interrupt.

2.2.5. INTRRX1

IntrRx1 is an 8-bit read-only register that indicates which of the interrupts for Rx Endpoints 1 – 7 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:* 8'h00

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

Bit	Name	Function
D7	**EP7**	Rx Endpoint 7 interrupt.
D6	**EP6**	Rx Endpoint 6 interrupt.
D5	**EP5**	Rx Endpoint 5 interrupt.
D4	**EP4**	Rx Endpoint 4 interrupt.
D3	**EP3**	Rx Endpoint 3 interrupt.
D2	**EP2**	Rx Endpoint 2 interrupt.
D1	**EP1**	Rx Endpoint 1 interrupt.
D0	**–**	*Unused*, always returns 0

2.2.6. INTRRX2

IntrRx2 is an 8-bit read-only register that indicates which of the interrupts for Rx Endpoints 8 – 15 are currently active. *Note:* This register will only be present if the MUSBFDRC is configured for more than 7 Tx endpoints and 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: 05h*; Reset value:* 8'h00

	D7	D6	D5	D4	D3	D2	D1	D0
	EP15	**EP14**	**EP13**	**EP12**	**EP11**	**EP10**	**EP9**	**EP8**
From CPU	r	r	r	r	r	r	r	r
From USB	set	set	set	set	set	set	set	set

Bit	Name	Function
D7	**EP15**	Rx Endpoint 15 interrupt.
D6	**EP14**	Rx Endpoint 14 interrupt.
D5	**EP13**	Rx Endpoint 13 interrupt.
D4	**EP12**	Rx Endpoint 12 interrupt.
D3	**EP11**	Rx Endpoint 11 interrupt.
D2	**EP10**	Rx Endpoint 10 interrupt.
D1	**EP9**	Rx Endpoint 9 interrupt.
D0	**EP8**	Rx Endpoint 8 interrupt.

2.2.7. 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: 06h*; 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 MUSBFDRC is 'A' device.*
D6	**Sess Req**	Set when Session Request signaling has been detected. *Only valid when MUSBFDRC is 'A' device.*
D5	**Discon**	Set in Host mode when a device disconnect is detected. Set in Peripheral mode when a session ends.
D4	**Conn**	Set when a device connection is detected. *Only valid in Host mode.*
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.
D1	**Resume**	Set when Resume signaling is detected on the bus while the MUSBFDRC is in Suspend mode.
D0	**Suspend**	Set when Suspend signaling is detected on the bus. *Only valid in Peripheral mode.*

2.2.8. INTRTX1E AND INTRTX2E

IntrTx1E and IntrTx2E are 8-bit registers that provide interrupt enable bits for the interrupts in IntrTx1 and IntrTx2 respectively. 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.

IntrTx1E *Address:* 07h*; Reset value:*8'hFF masked with the Tx endpoints implemented

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

IntrTx2E *Address:* 08h*; Reset value:*8'hFF masked with the Tx endpoints implemented

	D7	D6	D5	D4	D3	D2	D1	D0
	EP15	**EP14**	**EP13**	**EP12**	**EP11**	**EP10**	**EP9**	**EP8**
From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	r	r	r	r	r	r	r	r

2.2.9. INTRRX1E AND INTRRX2E

IntrRx1E and IntrRx2E are 8-bit registers that provide interrupt enable bits for the interrupts in IntrRx1 and IntrRx2 respectively. 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.

IntrRx1E *Address:* 09h*; Reset value:* 8'hFE masked with the Rx endpoints implemented

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

IntrRx2E *Address:* 0Ah*; Reset value:* 8'hFF masked with the Rx endpoints implemented

	D7	D6	D5	D4	D3	D2	D1	D0
	EP15	**EP14**	**EP13**	**EP12**	**EP11**	**EP10**	**EP9**	**EP8**
From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	r	r	r	r	r	r	r	r

2.2.10. 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

2.2.11. FRAME1

Frame1 is an 8-bit read-only register that in Peripheral mode holds the lower 8 bits of the last received frame number and in Host mode holds the lower 8 bits of the current frame number.

*Address:*0Ch*; Reset value:* 8'b00000000

	D7	...	D0
	Lower 8 bits of Frame Number		
From CPU	r	...	r
From USB	rw	...	rw

2.2.12. FRAME2

Frame2 is a 3-bit read-only register that in Peripheral mode holds the upper 3 bits of the last received frame number and in Host mode holds the upper 3 bits of the current frame number.

*Address:*0Dh*; Reset value:*3'b000

	D2	...	D0
	Upper 3 bits of Frame Number		
From CPU	r	...	r
From USB	rw	...	rw

2.2.13. INDEX

Index is a 4-bit register that determines which endpoint control/status registers are accessed at addresses 10h to 19h.

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

	D3	D2	D1	D0
	Selected Endpoint			
From CPU	rw	rw	rw	rw
From USB	r	r	r	r

Each Tx endpoint and each Rx endpoint have their own set of control/status registers. Only one set of Tx control/status and one set of Rx control/status registers appear in the memory map at any one time. Before accessing an endpoint's control/status registers, the endpoint number should be written to the Index register to ensure that the correct control/status registers appear in the memory map.

2.2.14. DEVCTL

DevCtl is an 8-bit register that is used to select whether the MUSBFDRC is operating in Peripheral mode or in Host mode, and for controlling and monitoring the USB VBus line.

Address: 0Fh; *Reset value:* 8'h88

	D7	D6	D5	D4	D3	D2	D1	D0
	CID	**FSDev**	**LSDev**	**PUCON**	**PDCON**	**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	**CID**	This bit indicates the state of the CID input signal to the MUSBFDRC. It should be used to indicate the ID pin of the mini-A/B connector. 1 ⇒ B-type, 0 ⇒ A-type.
D6	**FSDev**	This bit is set when a full-speed device has been detected being connected to the port. *Only valid in Host mode.*
D5	**LSDev**	This bit is set when a low-speed device has been detected being connected to the port. *Only valid in Host mode.*
D4	**PUCON**	This Read-only bit becomes set prior to operation as a peripheral to indicate that a pull-up resistor is required on the USB D+ line. The selection of this resistor will normally be handled automatically by external circuitry using the PUCON output (which will also have gone high).
D3	**PDCON**	This Read-only bit becomes set prior to operation as a host to indicate that a pull-down resistor is required on the USB D+ line (alongside a similar pull-down resistor on the USB D- line. The selection of these resistors will normally be handled automatically by external circuitry using the PDCON output (which will also have gone high).
D2	**Host Mode**	This Read-only bit is set when the MUSBFDRC is acting as a Host.
D1	**Host Req**	When set, the MUSBFDRC will initiate the Host Negotiation when Suspend mode is entered. It is cleared when Host Negotiation is completed. See Section 13. *('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 MUSBFDRC when a session starts/ends. It can also be set by the CPU to initiate the Session Request Protocol, and cleared to execute a soft disconnect from the host (which then ends the session). *Note:* When the Session bit is cleared, CLKOUT is stopped so that the PHY is put into Suspend mode and power is saved while no session is in progress.

2.3. INDEXED REGISTERS

2.3.1. CSR0

CSR0 is an 8-bit register that provides control and status bits for Endpoint 0.

CSR0 is used for all control/status of Endpoint 0. For details of how to service device requests to Endpoint 0, see Section 7: 'Endpoint 0 Handling'.

The interpretation of the register depends on whether the MUSBFDRC is acting as a peripheral or as a host.

CSR0 in Peripheral mode: *Address:* 11h (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 the data packet has been transmitted. An interrupt is generated when the bit is cleared.
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.

CSR0 in Host mode:

Address: 11h (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	rw	r/clear	r/set	r/clear
From USB	set	r	rw	set	rw	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.
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 the data packet has been transmitted. An interrupt is generated when the bit is cleared.
D0	**RxPktRdy**	This bit is set when a data packet has been received. An interrupt is generated when this bit is set. The CPU should clear this bit when the packet has been read from the FIFO.

2.3.2. CSR02

The CSR02 register comprises a single self-clearing bit which may be used to flush the Endpoint 0 FIFO.

Address: 12h (with the Index register set to 0)*; Reset value:* 1'b0

Bit	Name	Function
D0 *Set from CPU; Read only from USB*	**FlushFIFO** *(Self clearing)*	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 (above) is cleared. *Note:* FlushFIFO has no effect unless TxPktRdy/RxPktRdy is set.

2.3.3. 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 is valid while RxPktRdy (CSR0.D0) is set.

Address: 16h (with the Index register set to 0)*; Reset value:7*'b0000000

	D6	D5	D4	D3	D2	D1	D0
	Endpoint 0 Count						
From CPU	r	r	r	r	r	r	r
From USB	w	w	w	w	w	w	w

2.3.4. NAKLIMIT0 (HOST MODE ONLY)

NAKLimit0 is an 8-bit register that sets the number of frames 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 2.3.14 and 2.3.16.)

The number of frames selected may be between 2 and 255. 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: 19h (with the Index register set to 0)*; Reset value:*8'b00000000

	D7	...	D0
	Endpoint 0 NAK Limit		
From CPU	rw	...	rw
From USB	r	...	r

2.3.5. TXMAXP

TxMaxP is an 8-bit register that holds the maximum packet size for transactions through the currently-selected Tx endpoint – in units of 8 bytes, except that a value of 128 sets the maximum packet size to 1023 (the maximum size for an Isochronous packet transferred in a full-speed transaction) rather than 1024. In setting this value, you should note the constraints placed by the USB Specification on packet sizes for Bulk, Interrupt and Isochronous transactions in full-speed operations.

There is a TxMaxP register for each configured Tx endpoint (except Endpoint 0).

Address: 10h*; Reset value:* 8'h00

	D7	...	D0
	Maximum Packet Size/transaction		
From CPU	rw	...	rw
From USB	r	...	r

The value written to this register should match the *wMaxPacketSize* field of the Standard Endpoint Descriptor for the associated endpoint (see *Universal Serial Bus Specification* Revision 2.0, Chapter 9). A mismatch could cause unexpected results.

The value written to this register should not exceed the Tx FIFO size or, where dynamic FIFO sizing is used, the maximum packet size specified in the TxFIFO2 register. If the value written to this register is less than, or equal to, half the Tx FIFO size,

two packets can be buffered (except where dynamic FIFO sizing has been selected: see Section 3).

The register is reset to 0. If this register is changed after packets have been sent from the endpoint, then the endpoint FIFO should be completely flushed (using the FlushFIFO bit (D3) in TxCSR1) after writing the new value to the TxMaxP register.

2.3.6. TXCSR1

TxCSR1 is an 8-bit register that provides control and status bits for transfers through the currently-selected Tx endpoint. There is a TxCSR1 register for each configured Tx endpoint (not including Endpoint 0).

The interpretation of the register depends on whether the MUSBFDRC is acting as a peripheral or as a host.

TxCSR1 in Peripheral mode:

Address: 11h*; Reset value:* 8'h00

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

Bit	Name	Function in Peripheral mode
D7	**–**	*Unused.* Returns zero when read.
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 next packet to be transmitted from the endpoint Tx FIFO. The FIFO pointer is reset and the TxPktRdy bit (below) is cleared. *Note:* FlushFIFO has no effect unless TxPktRdy is set. 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 the TxPktRdy bit 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 generated (if enabled) when the bit is cleared.

TxCSR1 in Host mode:

Address: 11h*; Reset value:* 8'h00

	D7	D6	D5	D4	D3	D2	D1	D0
	NAK Timeout	**ClrDataTog**	**RxStall**	**-**	**FlushFIFO** *(self-clearing)*	**Error**	**FIFO NotEmpty**	**TxPktRdy**
From CPU	r/clear	set	r/clear	r	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**	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. *Note: Valid only for Bulk endpoints.*
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. The FIFO is flushed and the TxPktRdy bit is cleared (see below). The CPU should clear this bit.
D4	**-**	*Unused.* Returns zero when read.
D3	**FlushFIFO**	The CPU writes a 1 to this bit to flush the next packet to be transmitted from the endpoint Tx FIFO. The FIFO pointer is reset and the TxPktRdy bit (below) is cleared. *Note:* FlushFIFO has no effect unless TxPktRdy is set. 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. The CPU should clear this bit. An interrupt is generated when the bit is set. *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 generated (if enabled) when the bit is cleared.

2.3.7. TXCSR2

TxCSR2 is an 8-bit register that provides further control bits for transfers through the currently-selected Tx endpoint. There is an TxCSR2 register for each configured Tx endpoint (not including Endpoint 0).

Address: 12h*; Reset value:*8'h20

	D7	D6	D5	D4	D3	D2	D1	D0
	AutoSet	**ISO**	**Mode**	**DMAEnab**	**FrcDataTog**	**DMAMode**	–	–
From CPU	rw	rw	rw	rw	rw	rw	r	r
From USB	r	r	r	r	r	r	r	r

Bit	Name	Function
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.
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 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	**DMAEnab**	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 endpoints that are used to communicate rate feedback for Isochronous endpoints.

Bit	Name	Function
D2	**DMAMode**	Two modes of DMA operation are supported. The CPU sets this bit to select DMA Mode 1 (in which a DMA request (but no interrupt) is automatically generated for packets of size TxMaxP bytes) and clears this bit to select DMA Mode 0 (in which a DMA request and an interrupt are generated for all packets).
D1 – D0	–	*Unused*, always returns 0.

2.3.8. RXMAXP

RxMaxP is an 8-bit register that holds the maximum packet size for transactions through the currently-selected Rx endpoint – in units of 8 bytes, except that a value of 128 sets the maximum packet size to 1023 (the maximum size for an Isochronous packet transferred in a full-speed transaction) rather than 1024. In setting this value, you should note the constraints placed by the USB Specification on packet sizes for Bulk, Interrupt and Isochronous transactions in full-speed operations.

There is an RxMaxP register for each configured Rx endpoint (except Endpoint 0).

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

	D7	...	D0
	Maximum Packet Size/transaction		
From CPU	rw	...	rw
From USB	r	...	r

The value written to this register should match the *wMaxPacketSize* field of the Standard Endpoint Descriptor for the associated endpoint (see *Universal Serial Bus Specification* Revision 2.0, Chapter 9). A mismatch could cause unexpected results.

The value written to this register must not exceed the Rx FIFO size or, where dynamic FIFO sizing is used, the maximum packet size specified in the RxFIFO2 register. If the value written to this register is less than, or equal to, half the Rx FIFO size, two packets can be buffered (except where dynamic FIFO sizing has been selected: see Section 3).

2.3.9. RXCSR1

RxCSR1 is an 8-bit register that provides control and status bits for transfers through the currently-selected Rx endpoint. There is an RxCSR1 register for each configured Rx endpoint (not including Endpoint 0). The interpretation of the register depends on whether the MUSBFDRC is acting as a peripheral or as a host.

RxCSR1 in Peripheral mode:

*Address:*14h*; 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.*

Bit	Name	Function in Peripheral mode
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 has no effect unless RxPktRdy is set. Also note that, if the FIFO is double-buffered, FlushFIFO may need to be set twice to completely clear the FIFO.
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.

RxCSR1 in Host mode:

*Address:*14h*; 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. For double packet buffering the ReqPkt is cleared immediately after being set if there is no request pending. If there is a request pending then the ReqPkt is cleared when the RxPktRdy for the pending request is cleared. Writing a 0 to the ReqPkt will clear all outstanding requests.
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 has no effect unless RxPktRdy is set. Also note that, if the FIFO is double-buffered, FlushFIFO may need to be set twice to completely clear the FIFO.
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 IN 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.

2.3.10. RXCSR2

RxCSR2 is an 8-bit register that provides further control bits for transfers through the currently-selected Rx endpoint. There is an RxCSR2 register for each configured Rx endpoint (not including Endpoint 0).

*Address:*15h*; Reset value:*8'h00

	D7	D6	D5	D4	D3	D2	D1	D0
	AutoClear	**ISO/AutoReq**	**DMAEnab**	**DMAMode**	–	–	–	–
From CPU	rw	rw	rw	rw	r	r	r	r
From USB	r	r	r	r	r	r	r	r

Bit	Name	Function
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.
D6	**ISO** **AutoReq**	In Peripheral mode, the CPU sets this bit to enable the Rx endpoint for Isochronous transfers, and clears it to enable the Rx endpoint for Bulk or Interrupt transfers. If the CPU sets this bit in Host mode, the ReqPkt bit will be automatically set when the RxPktRdy bit is cleared.
D5	**DMAEnab**	The CPU sets this bit to enable the DMA request for the Rx endpoint.
D4	**DMAMode**	Two modes of DMA operation are supported: DMA Mode 0 in which a DMA request is generated for all received packets, together with an interrupt (if enabled); and DMA Mode 1 in which a DMA request (but no interrupt) is generated for Rx packets of size RxMaxP bytes and an interrupt (but no DMA request) is generated for Rx packets of any other size. The CPU sets this bit to select DMA Mode 1 and clears this bit to select DMA Mode 0.
D3 – D0	–	*Unused*, always return 0.

2.3.11. RXCOUNT1

RxCount1 is a 8-bit read-only register that holds the lower 8 bits of the number of received data bytes in the packet in the FIFO associated with the currently-selected Rx endpoint. The value returned is valid while RxPktRdy (RxCSR1.D0) is set.

Address: 16h*; Reset value:* 8'b00000000

	D7	...	D0
	Endpoint Rx Count – lower 8 bits		
From CPU	r	...	r
From USB	w	...	w

2.3.12. RXCOUNT2

RxCount2 is a 3-bit read-only register that holds the upper 3 bits of the number of received data bytes in the packet in the FIFO associated with the currently-selected Rx endpoint. The value returned is valid while RxPktRdy (RxCSR1.D0) is set.

Address: 17h*; Reset value:* 3'b000

	D2	...	D0
	Endpoint Rx Count – upper 3 bits		
From CPU	r	...	r
From USB	w	...	w

2.3.13. TXTYPE (HOST MODE ONLY)

TxType is a 6-bit register that should be written with the endpoint number to be targeted by the endpoint in the lower 4 bits, and the transaction protocol to use for the currently-selected Tx endpoint in the upper 2 bits.

There is a TxType register for each configured Tx endpoint (except Endpoint 0).

*Address:*18h*; Reset value:*8'h00

	D5	D4	D3	D2	D1	D0
	Protocol		**Target Endpoint Number**			
From CPU	rw	rw	rw	rw	rw	rw
From USB	r	r	r	r	r	r

Bit	Name	Function
D5-4	**Protocol**	The CPU should set this to select the required protocol for the Tx endpoint: 00 : Illegal 01: Isochronous 10: Bulk 11: Interrupt
D3 – D0	**Target Endpoint Number**	The CPU should set this value to the endpoint number contained in the OUT endpoint descriptor returned to the MUSBFDRC during device enumeration.

2.3.14. TXINTERVAL (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 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).

Address: 19h*; Reset value:* 8'b00000000

	D7	...	D0
	Tx Polling Interval/NAK Limit (*n*)		
From CPU	rw	...	rw
From USB	r	...	r

In each case the value that is set defines a number of frames, as follows:

Transfer Type	Valid values (*n*)	Interpretation
Interrupt or Isochronous	1 – 255	Polling interval is *n* frames.
Bulk	2 – 255	NAK Limit is *n* frames. *Note:* A value of 0 or 1 disables the NAK timeout function.

Note: The USB Specification allows polling intervals greater than 255ms for an Isochronous endpoints. Where required, these must be implemented in software.

2.3.15. RXTYPE (HOST MODE ONLY)

RxType is a 6-bit register that should be written with the endpoint number to be targeted by the endpoint in the lower 4 bits, and the transaction protocol to use for the currently-selected Rx endpoint in the upper 2 bits.

There is an RxType register for each configured Rx endpoint (except Endpoint 0).

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

	D5	D4	D3	D2	D1	D0
	Protocol		**Target Endpoint Number**			
From CPU	rw	rw	rw	rw	rw	rw
From USB	r	r	r	r	r	r

Bit	Name	Function
D5-4	**Protocol**	The CPU should set this to select the required protocol for the Rx endpoint: 00 : Illegal 01: Isochronous 10: Bulk 11: Interrupt
D3 – D0	**Target Endpoint Number**	The CPU should set this value to the endpoint number contained in the IN endpoint descriptor returned to the MUSBFDRC during device enumeration.

2.3.16. RXINTERVAL (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 after which the endpoint should timeout on receiving a stream of NAK responses. There is an RxInterval register for each configured Rx endpoint (except Endpoint 0).

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

	D7	...	D0
	Rx Polling Interval/NAK Timeout (*n*)		
From CPU	rw	...	rw
From USB	r	...	r

In each case the value that is set defines a number of frames, as follows:

Transfer Type	Valid values (*n*)	Interpretation
Interrupt or Isochronous	1 – 255	Polling interval is *n* frames.
Bulk	2 – 255	NAK Limit is *n* frames. *Note:* A value of 0 or 1 disables the NAK timeout function.

Note: The USB Specification allows polling intervals greater than 255ms for an Isochronous endpoints. Where required, these must be implemented in software.

2.3.17. FIFOSIZE

FIFOSize is an 8-bit Read-Only register that returns the size of the FIFO associated with the selected endpoint (numbers 1 – 15) where set as part of the core configuration. *Note:* The register value is not valid for Endpoint 0 (always 64 bytes) or where Dynamic FIFO Sizing is used.

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 – 11 correspond to a FIFO size of 2^n bytes (8 – 2048 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.

Address: 1Fh*; 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

2.4. FIFOx (Address 20h – 2Fh)

These 16 addresses provide CPU access to the FIFOs for each endpoint. Writing to these addresses loads data into the Tx FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Rx FIFO for the corresponding endpoint.

The FIFOs are located on byte boundaries (Endpoint 0 at 20h, Endpoint 1 at 21h ... Endpoint 15 at 2Fh).

Note: (i) Using the MUSBFDRC – AHB bridge requires a different address mapping to be used for these FIFOs. Details are given in the separate specification for that bridge included in the **musbfdrc/docs** directory.
(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.

3. DYNAMIC FIFO SIZING

If needed, the MUSBFDRC can be configured to have a single overall FIFO size of 128, 256, 512, 1K … 32K bytes, areas of which may then be allocated to the different endpoints when the MUSBFDRC is initialized – though users are strongly advised to only use this feature where the MUSBFDRC is used in a device that actually requires different FIFO sizes in different contexts.

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 MUSBFDRC register map when the option of Dynamic FIFO sizing is selected:

MUSBFDRC REGISTER MAP: Indexed area (Endpoint selected by the Index register)		
ADDR	**NAME**	**DESCRIPTION**
1C	TxFIFO1	8-bit registers which together control the start address and the size of the selected Tx endpoint FIFO
1D	TxFIFO2	
1E	RxFIFO1	8-bit registers which together control the start address and the size of the selected Rx endpoint FIFO
1F	RxFIFO2	

Details of these registers are given below.

TxFIFO1 *Address:* 1Ch*; Reset value:*8'h00

	D7	D6	D5	D4	D3	D2	D1	D0
	AD7	**AD6**	**AD5**	**AD4**	**AD3**	**AD2**	**AD1**	**AD0**
From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	r	r	r	r	r	r	r	r

TxFIFO2 *Address:* 1Dh*; Reset value:*8'h00

	D7	D6	D5	D4	D3	D2	D1	D0
	SZ2	**SZ1**	**SZ0**	**DPB**	**AD11**	**AD10**	**AD9**	**AD8**
From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	r	r	r	r	r	r	r	r

RxFIFO1 *Address:* 1Eh*; Reset value:*8'h00

	D7	D6	D5	D4	D3	D2	D1	D0
	AD7	**AD6**	**AD5**	**AD4**	**AD3**	**AD2**	**AD1**	**AD0**
From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	r	r	r	r	r	r	r	r

RxFIFO2 *Address:* 1Fh*; Reset value:*8'h00

	D7	D6	D5	D4	D3	D2	D1	D0
	SZ2	**SZ1**	**SZ0**	**DPB**	**AD11**	**AD10**	**AD9**	**AD8**
From CPU	rw	rw	rw	rw	rw	rw	rw	rw
From USB	r	r	r	r	r	r	r	r

<table>
<tr><th>Bit</th><th>Name</th><th>Function</th></tr>
<tr><td colspan="2">TxFIFO1/RxFIFO1</td><td rowspan="4">Start address of the endpoint FIFO in units of 8 bytes as follows:
<table>
<tr><th colspan="3">AD[11:0]</th><th>Start Address</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0000</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0008</td></tr>
<tr><td>0</td><td>0</td><td>2</td><td>0010</td></tr>
<tr><td></td><td>…</td><td></td><td>…</td></tr>
<tr><td>F</td><td>F</td><td>F</td><td>7FF8</td></tr>
</table></td></tr>
<tr><td>D7 – D0</td><td>AD[7:0]</td></tr>
<tr><td colspan="2">TxFIFO2/RxFIFO2</td></tr>
<tr><td>D3 – D0</td><td>AD[11:8]</td></tr>
<tr><td>D4</td><td>DPB</td><td>Defines whether double-packet buffering supported. When '1', double-packet buffering is supported. When '0', only single-packet buffering is supported.</td></tr>
<tr><td>D7 – D5</td><td>SZ[2:0]</td><td>Maximum packet size to be allowed for
<table>
<tr><th colspan="3">SZ[2:0]</th><th>Packet Size (Bytes)</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>8</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>16</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>32</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>64</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>128</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>256</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>512</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1024</td></tr>
</table>
If DPB = 0, the FIFO will also be this size; if DPB = 1, the FIFO will be twice this size.</td></tr>
</table>

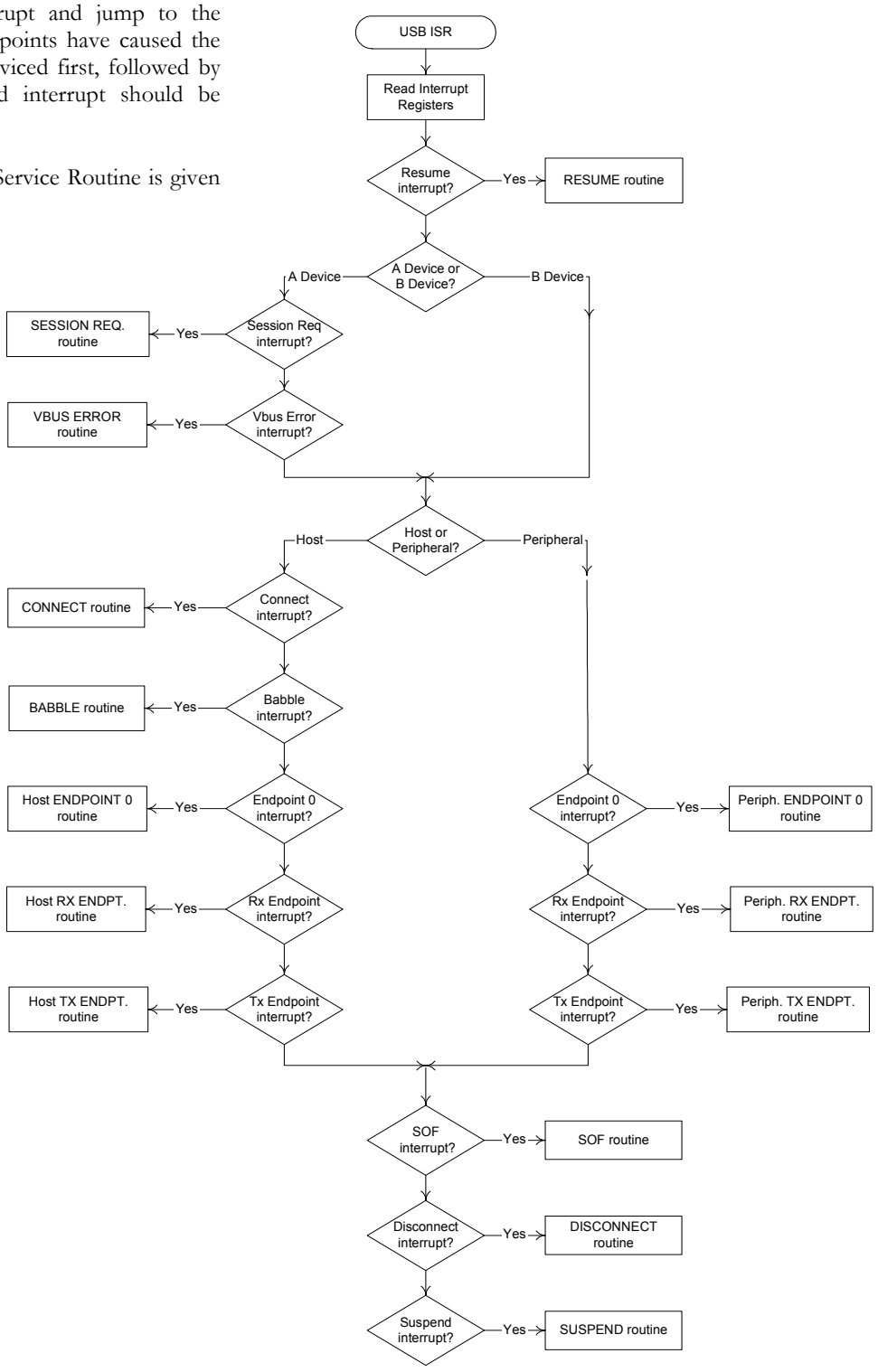
Note: It is the responsibility of the firmware to ensure that all the Tx and Rx endpoints that are active in the current USB configuration have a block of RAM assigned exclusively to them that is at least as large as the maximum packet size set for the endpoint.

4. 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. The Suspend interrupt should be serviced last.

A flowchart for the USB Interrupt Service Routine is given in Figure 4-1.

Figure 4-1 USB Interrupt Service Routine



5. SUSPEND/RESUME

When the MUSBFDRC sees no activity on the USB for 3 ms, it will generate a Suspend interrupt (if enabled).

It is up to the software to decide what, if anything, to disable when the USB is in Suspend mode.

5.1. MUSBFDRC ACTIVE DURING SUSPEND

The USB may exit Suspend mode by sending Resume signaling on the bus.

If the MUSBFDRC is left active while the USB is in Suspend mode, the MUSBFDRC will monitor the USB for Resume signaling. When Resume signaling appears on the bus, the MUSBFDRC will generate a Resume interrupt.

5.2. MUSBFDRC INACTIVE DURING SUSPEND

When the Suspend interrupt described above is received, the software may disable the MUSBFDRC by stopping its clock (this must be done by some external means). However, because the MUSBFDRC is disabled, it will not then be able to detect Resume signaling on the USB. As a result, some external hardware will be needed to detect Resume signaling (by monitoring the DIM and DIP signals), so that the clock to the MUSBFDRC can be restarted.

5.3. REMOTE WAKEUP

If the MUSBFDRC is in Suspend mode and the software wants to initiate a remote wakeup, it should write to the Power register to set the Resume bit (D2) to 1. (If the clock to the MUSBFDRC has been stopped, it will need to be restarted before this write can occur.)

The software should leave this bit set for approximately 10 ms (minimum of 2 ms, a maximum of 15 ms) then reset 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.

6. USB RESET (Peripheral mode only)

When a reset condition is detected on the USB, the MUSBFDRC performs the following actions:

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

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

7. CONTROL TRANSACTIONS (via ENDPOINT 0)

All Control transactions are carried out through Endpoint 0, both where the MUSBFDRC is used in a peripheral and when it is being used as a host.. As a result, the routines required to service Endpoint 0 are more complicated than those required to service other endpoints.

This section first describes the actions that need to be taken where the MUSBFDRC is used in a USB peripheral, then the actions that need to be taken where the device incorporating the MUSBFDRC is acting as the host for point-to-point communications.

7.1. CONTROL TRANSACTIONS AS A PERIPHERAL

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 MUSBFDRC core.

7.1.1. 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 (CSR0.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 MUSBFDRC 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 (CSR0.D2) will be set.

If the host sends more data after the DataEnd bit has been set, then the MUSBFDRC will send a STALL. An Endpoint 0 interrupt will be generated and the SentStall bit (CSR0.D2) will be set.

7.1.2. 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 (CSR0.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 (CSR0.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 MUSBFDRC 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 (CSR0.D2) will be set.

If the host sends more data after the DataEnd has been set, then the MUSBFDRC will send a STALL. An Endpoint 0 interrupt will be generated and the SentStall bit (CSR0.D2) will be set.

7.1.3. 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 (CSR0.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.

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 MUSBFDRC 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 (CSR0.D2) will be set.

If the host requests more data after DataEnd (D3) has been set, then the MUSBFDRC will send a STALL. An Endpoint 0 interrupt will be generated and the SentStall bit (CSR0.D2) will be set.

7.1.4. ENDPOINT 0 STATES

When the MUSBFDRC 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 (see Figure 7-1 below).

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

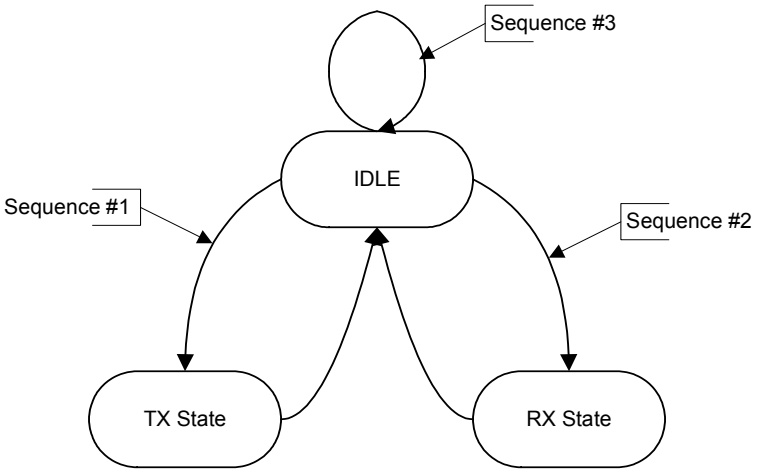
RxPktRdy (CSR0.D0) becoming set when Endpoint 0 is in IDLE state indicates a new device request. Once the device request is unloaded from the FIFO, the MUSBFDRC decodes the descriptor to find whether there is a Data phase and, if so, the direction of the Data phase for the control transfer (in order to set the FIFO direction).

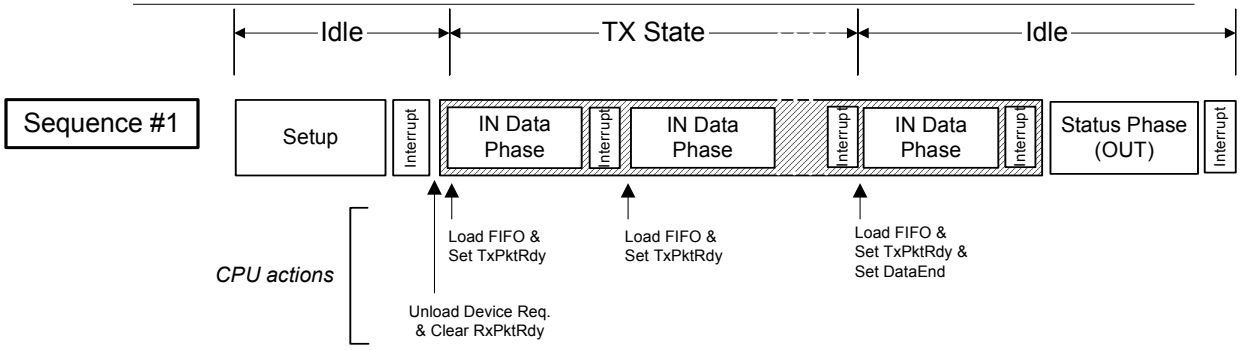
Depending on the direction of the Data phase, Endpoint 0 goes into either TX state or RX state. If there is no Data phase, Endpoint 0 remains in IDLE state to accept the next device request.

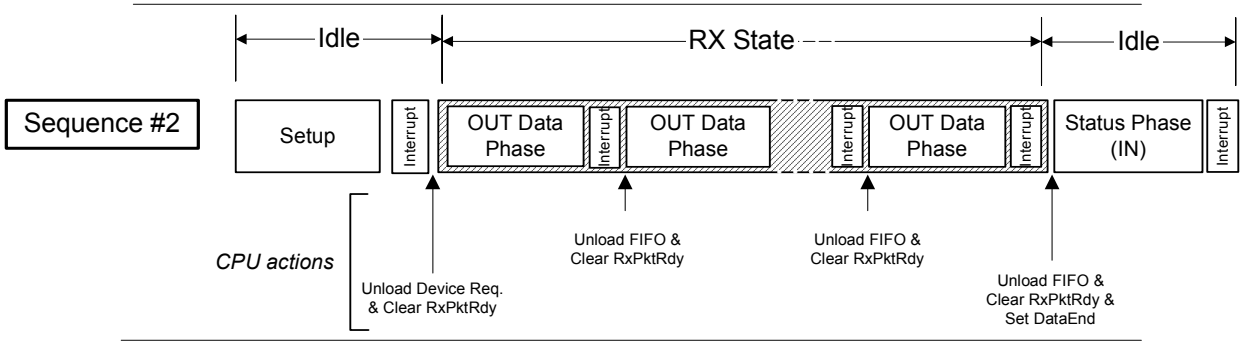
The actions that the CPU needs to take at the different phases of the possible transfers (e.g. Loading the FIFO, Setting TxPktRdy) are indicated in the diagram on the following page.

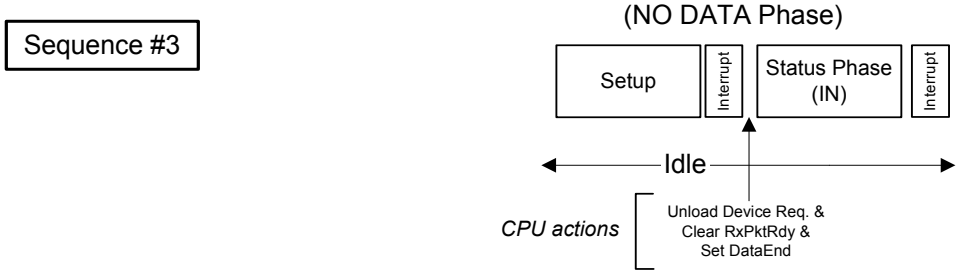
Note that the MUSBFDRC changes the FIFO direction depending on the direction of the Data phase *independently* of the CPU.

Figure 7-1 Endpoint 0 States for Peripheral









7.1.5. ENDPOINT 0 SERVICE ROUTINE AS PERIPHERAL

An Endpoint 0 interrupt is generated:

- When the core sets the RxPktRdy bit (CSR0.D0) after a valid token has been received and data has been written to the FIFO.
- When the core clears the TxPktRdy bit (CSR0.D1) after the packet of data in the FIFO has been successfully transmitted to the host.
- When the core sets the SentStall bit (CSR0.D2) after a control transaction is ended due to a protocol violation.
- When the core sets the SetupEnd bit (CSR0.D4) because a control transfer has ended before DataEnd (CSR0.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 (CSR0.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:

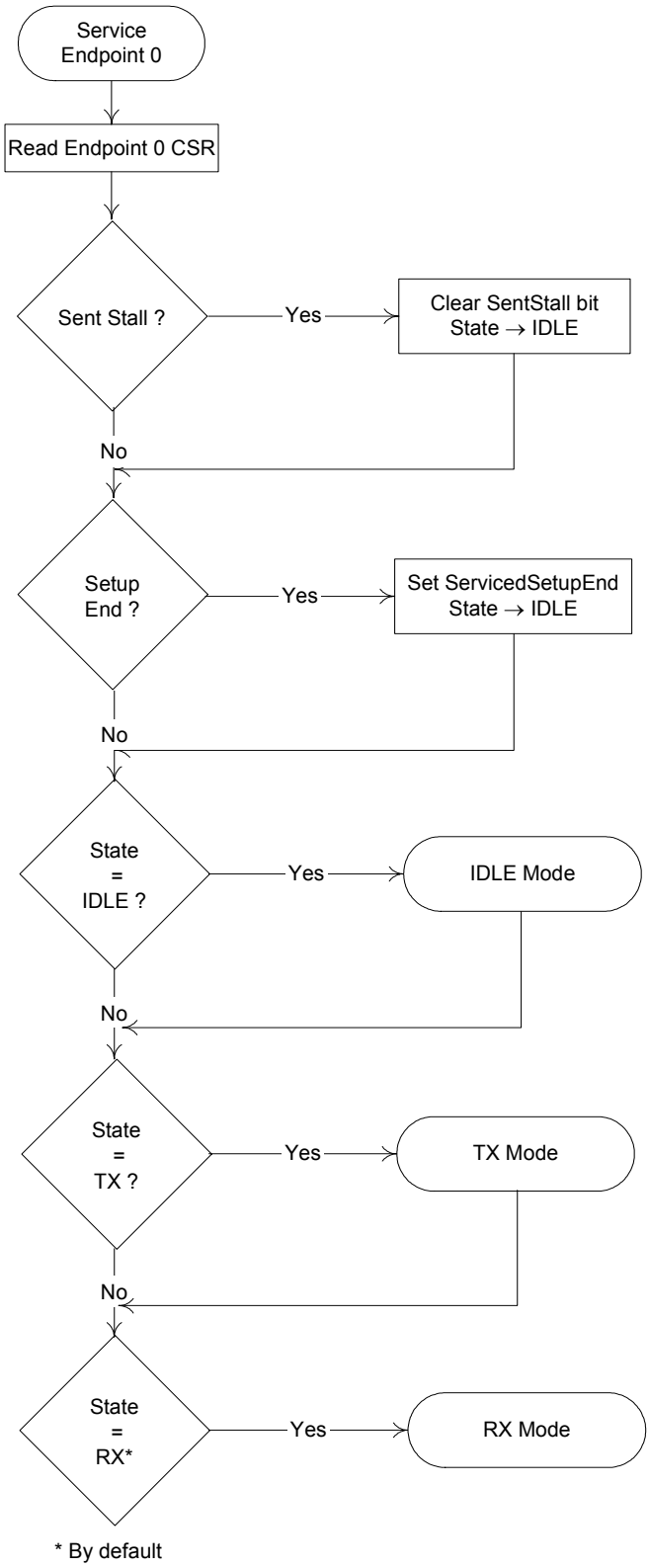
- If the command is a single packet transaction (SET_ADDRESS, SET_INTERFACE etc.) without any data phase, the endpoint will remain in IDLE state.
- If the command has an OUT data phase (SET_DESCRIPTOR etc.), the endpoint will enter RX state.
- If the command has an IN data phase (GET_DESCRIPTOR etc.), 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 (CSR0.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.

Figure 7-2 Flow for Endpoint 0 Service Routine when MUSBFDRC operating as a Peripheral

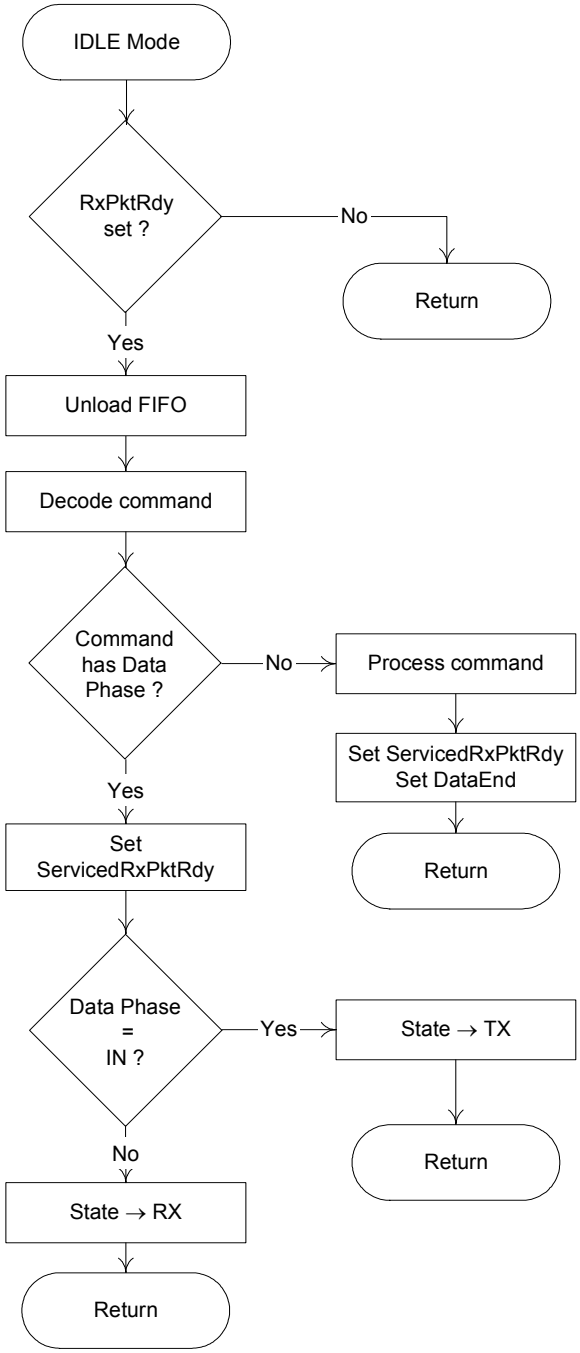


IDLE MODE

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 Figure 7-3 below).

Figure 7-3 Flow for SETUP phase of Control Transfer when MUSBFDRC operating as a Peripheral



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.

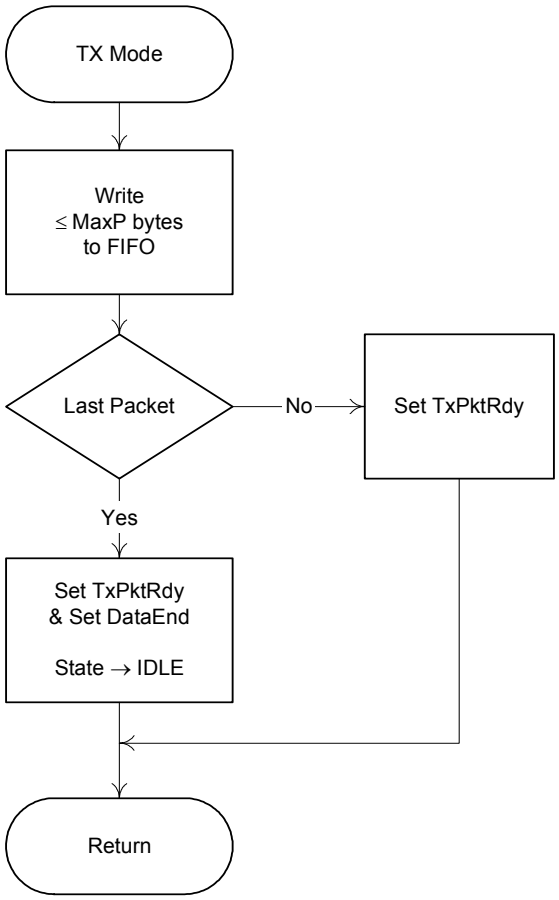
Three events can cause TX mode to be terminated before the expected amount of data has been sent:

- The host sends an invalid token causing a SetupEnd condition (CSR0.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 (CSR0.D3) to indicate to the core that the Data phase is complete and that the core should next receive an acknowledge packet.

Figure 7-4 Flow for IN Data phase of Control Transfer when MUSBFDRC operating as a Peripheral



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.

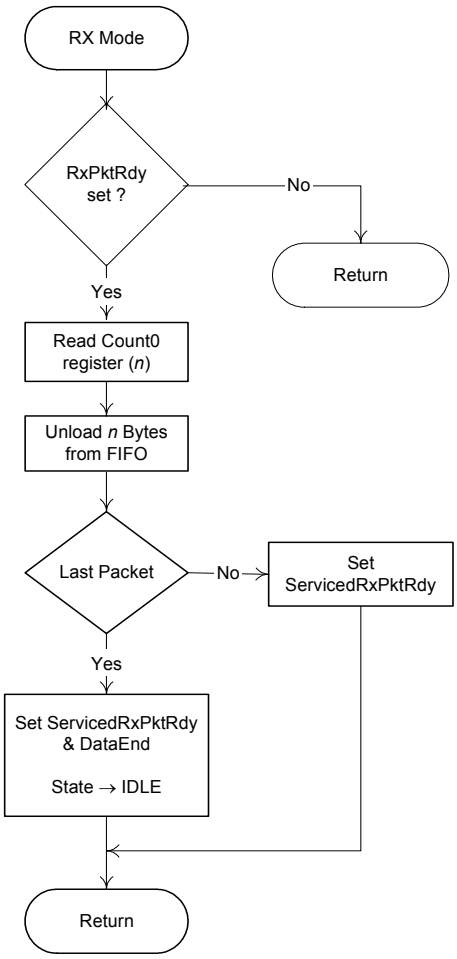
Three events can cause RX mode to be terminated before the expected amount of data has been received:

- The host sends an invalid token causing a SetupEnd condition (CSR0.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 (CSR0.D0) set) and to clear RxPktRdy by setting the ServicedRxPktRdy bit (CSR0.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 (CSR0.D3) to indicate to the core that the Data phase is complete and that the core should receive an acknowledge packet next.

Figure 7-5 Flow for OUT Data phase of Control Transfer when MUSBFDRC operating as a Peripheral



7.1.6. 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 MUSBFDRC will automatically detect protocol errors and send a STALL packet to the host under the following 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 (CSR0.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 MUSBFDRC has sent the STALL packet, it sets the SentStall bit (CSR0.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 (CSR0.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 (CSR0.D7), and return to the IDLE state. If the RxPktRdy bit (CSR0.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 (CSR0.D5). The MUSBFDRC will then send a STALL packet to the host, set the SentStall bit (CSR0.D2) and generate an Endpoint 0 interrupt.

7.2. 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 MUSBFDRC will retry a transaction which is continually NAKed by the target. This limit can be between 2 and 255 ms and is set through the NAKLimit0 register (see Section 2.3.4).

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.

7.2.1. 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 CSR0.D3 and CSR0.D1, respectively). *Note:* These bits need to be set together.

   The MUSBFDRC 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 16.1.)
3. At the end of the attempt to send the data, the MUSBFDRC will generate an Endpoint 0 interrupt (i.e. set IntrTx1.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 MUSBFDRC 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 MUSBFDRC has received a NAK response to each attempt to send the SETUP packet, for longer than the time set in the NAKLimit0 register. The MUSBFDRC 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.

7.2.2. 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 (CSR0.D5).
2. Wait while the MUSBFDRC both sends the IN token and receives the required data back. (The details of this operation are shown in Section 16.1.)
3. When the MUSBFDRC generates the Endpoint 0 interrupt (i.e. sets IntrTx1.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 MUSBFDRC has tried to send the required IN token three times without getting any response.

   If NAK Timeout is set, it means that the MUSBFDRC has received a NAK response to each attempt to send the IN token, for longer than the time set in the NAKLimit0 register. The MUSBFDRC 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.

7.2.3. 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 (CSR0.D1).

   The MUSBFDRC then proceeds to send a 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 16.1.)
3. At the end of the attempt to send the data, the MUSBFDRC will generate an Endpoint 0 interrupt (i.e. set IntrTx1.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 MUSBFDRC 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 MUSBFDRC has received a NAK response to each attempt to send the OUT token, for longer than the time set in the NAKLimit0 register. The MUSBFDRC 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.

7.2.4. IN STATUS PHASE AS A HOST (FOLLOWING EITHER THE SETUP PHASE OR AN 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 CSR0.D6 and CSR0.D5, respectively). *Note:* These bits need to be set together.
2. Wait while the MUSBFDRC both sends an IN token and receives a response from the USB peripheral. (The details of this operation are shown in Section 16.1.)
3. When the MUSBFDRC generates the Endpoint 0 interrupt (i.e. sets IntrTx1.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 could not complete the command and so has issued a STALL response.

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

   If NAK Timeout is set, it means that the MUSBFDRC has received a NAK response to each attempt to send the IN token, for longer than the time set in the NAKLimit0 register. The MUSBFDRC 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. If RxPktRdy has been set, the CPU should simply clear RxPktRdy.

7.2.5. 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 CSR0.D6 and CSR0.D1, respectively). *Note:* These bits need to be set together.
2. Wait while the MUSBFDRC both sends the OUT token and a zero-length DATA1 packet. (The details of this operation are

shown in Section 16.1.)

3. At the end of the attempt to send the data, the MUSBFDRC will generate an Endpoint 0 interrupt (i.e. set IntrTx1.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 MUSBFDRC 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 MUSBFDRC has received a NAK response to each attempt to send the IN token, for longer than the time set in the NAKLimit0 register. The MUSBFDRC 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.

8. BULK TRANSACTIONS

8.1. HANDLING BULK TRANSACTIONS AS A PERIPHERAL

8.1.1. BULK IN TRANSACTIONS AS A PERIPHERAL

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

Three 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.) 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 an external DMA controller to load packets into the FIFO without processor intervention.

- **AutoSet**

When the AutoSet feature is enabled, the TxPktRdy bit (TxCSR1.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.

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 IntrTx1E/IntrTx2E register should be set to 1 (if an interrupt is required for this endpoint) and the TxCSR2 register should be set as shown below (Bits D2 – D0 of TxCSR2 are unused):

D7	**AutoSet**	0/1	Set to 1 if the AutoSet feature is required.
D6	**ISO**	0	Set to 0 to enable Bulk protocol.
D5	**Mode**	1	Set to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an OUT endpoint).
D4	**DMAEnab**	0/1	Set to 1 if a DMA request is required for this endpoint.
D3	**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 TxCSR1 register should be written to set the ClrDataTog bit (D6). This will ensure that the data toggle (which is handled automatically by the MUSBFDRC) starts in the correct state. Also, if there are any data packets in the FIFO (indicated by the FIFONotEmpty bit (TxCSR1.D1) being set), they should be flushed by setting the FlushFIFO bit (TxCSR1.D3). *Note:* It may be necessary to set this bit twice in succession if double buffering is enabled.

OPERATION

When data is to be transferred over a Bulk IN pipe, a data packet needs to be loaded into the FIFO and the TxCSR1 register written to set the TxPktRdy bit (D0). When the packet has been sent, the TxPktRdy bit will be cleared by the MUSBFDRC 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 MUSBFDRC 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.

The packet size must not exceed the size specified in the TxMaxP register. When a block of data larger than TxMaxP is to be transferred, it must be sent as multiple packets. These packets should be TxMaxP in size, except the last packet which holds the residue. 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 less than TxMaxP in size. In the latter case, if the total size of the data block is a multiple of TxMaxP, it will be necessary for the function to send a null packet after all the data has been sent. This is done by setting TxPktRdy when the next interrupt is received, without loading any data into the FIFO.

If large blocks of data are being transferred, then the overhead of calling an interrupt service routine to load each packet can be avoided by using DMA. See Section 9.1 of the MUSBFDRC User Guide.

ERROR HANDLING

If the software wants to shut down the Bulk IN pipe, it should set the SendStall bit (TxCSR1.D4). When the MUSBFDRC receives the next IN token, it will send a STALL to the host, set the SentStall bit (TxCSR1.D5) and generate an interrupt.

When the software receives an interrupt with the SentStall bit (TxCSR1.D5) set, it should clear the SentStall bit. It should leave the SendStall bit (TxCSR1.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 TxCSR1 register (D6).

8.1.2. BULK OUT TRANSACTIONS AS A PERIPHERAL

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

Three 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.) 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 an external DMA controller to unload packets from the FIFO without processor intervention.

- **AutoClear**

When the AutoClear feature is enabled, the RxPktRdy bit (RxCSR1.D0) will be automatically cleared when a packet of RxMaxP bytes has been unloaded from the FIFO. 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.

SETUP

In configuring an Rx endpoint for Bulk 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 IntrRx1E/IntrRx2E register should be set to 1 (if an interrupt is required for this endpoint) and the RxCSR2 register should be set as shown below (Bits D4 – D0 of RxCSR2 are unused.):

D7	**AutoClear**	0/1	Set to 1 if the AutoClear feature is required.
D6	**ISO**	0	Set to 0 to enable Bulk protocol.
D5	**DMAEnab**	0/1	Set to 1 if a DMA request is required for this endpoint.

When the endpoint is first configured, following a SET_CONFIGURATION or SET_INTERFACE command on Endpoint 0, RxCSR1 should be written to set the ClrDataTog bit (D7). This will ensure that the data toggle (which is handled automatically by the MUSBFDRC) starts in the correct state. Also, if there are any data packets in the FIFO (indicated by the RxPktRdy bit (RxCSR1.D0) being set), they should be flushed by setting the FlushFIFO bit (RxCSR1.D4). *Note:* It may be necessary to set this bit twice in succession if double buffering is enabled.

OPERATION

When a data packet is received by a Bulk Rx endpoint, the RxPktRdy bit (RxCSR1.D0) is set and an interrupt is generated. The software should read the two RxCount registers for the endpoint to determine the size of the data packet. The data packet should be read from the FIFO, then the RxPktRdy bit should be cleared.

The packet sizes 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 RxMaxP is to be sent to the function, it will be sent as multiple packets. All the packets will be RxMaxP 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 RxMaxP in size. (If the total size of the data block is a multiple of RxMaxP, a null data packet will be sent after the data to signify that the transfer is complete.)

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. See Section 9.2 of the MUSBFDRC User Guide.

ERROR HANDLING

If the software wants to shut down the Bulk OUT pipe, it should set the SendStall bit (RxCSR1.D5). When the MUSBFDRC receives the next packet it will send a STALL to the host, set the SentStall bit (RxCSR1.D6) and generate an interrupt.

When the software receives an interrupt with the SentStall bit (RxCSR1.D6) set, it should clear the SentStall bit. It should leave the SendStall bit (RxCSR1.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 RxCSR1 register (D7).

8.2. HANDLING BULK TRANSACTIONS AS A HOST

8.2.1. BULK IN TRANSACTION AS A HOST

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 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 3.)

- **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 an external DMA controller to unload packets from the FIFO without processor intervention.

- **AutoReq(uest)**

When the Autoreq(uest) feature is enabled, the ReqPkt bit (RxCSR1.D5) will be automatically set when the RxPktRdy bit is cleared.

- **AutoClear**

When the AutoClear feature is enabled, the RxPktRdy bit (RxCSR1.D0) will be automatically cleared when a packet of RxMaxP bytes has been unloaded from the FIFO. This is particularly useful used 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.

SETUP

Before initiating any Bulk IN Transactions:

- The RxType register for the MUSBFDRC endpoint that is to be used needs to be written with 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 MUSBFDRC during device enumeration.
- The RxMaxP register for the MUSBFDRC 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 – 255 ms), or set to zero if the

NAK Timeout feature is not required.

- The relevant interrupt enable bit in the IntrRx1E/IntrRx2E register should be set to 1 (if an interrupt is required for this endpoint)
- The following bits of RxCSR2 register should be set as shown below:

D7	**AutoClear**	0/1	Set to 1 if the AutoClear feature is required.
D6	**AutoReq**	0	Set to 1 to enable automatic setting of the ReqPkt bit whenever RxPktRdy is cleared.
D5	**DMAEnab**	0/1	Set to 1 if a DMA request is required for this endpoint.

When the endpoint is first configured, following a SET_CONFIGURATION or SET_INTERFACE command on Endpoint 0, RxCSR1 should be written to set the ClrDataTog bit (D7). This will ensure that the data toggle (which is handled automatically by the MUSBFDRC) starts in the correct state. Also, if there are any data packets in the FIFO (indicated by the RxPktRdy bit (RxCSR1.D0) being set), they should be flushed by setting the FlushFIFO bit (RxCSR1.D4). *Note:* It may be necessary to set this bit twice in succession if double buffering is enabled.

OPERATION

When Bulk data is required from the USB peripheral, the CPU should set the ReqPkt bit in the corresponding RxCSR1 register (D5). The MUSBFDRC will then send an IN token to the selected Peripheral endpoint and waits for data to be returned.

If data is correctly received, RxPktRdy (RxCSR1.D0) is set. If the USB peripheral responds with a STALL, RxStall (RxCSR1.D6) is set. If a NAK is received, the MUSBFDRC 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, a further two attempts are made before the MUSBFDRC reports an error (RxCSR1.D2 set).

The MUSBFDRC then generates the appropriate endpoint interrupt, whereupon the CPU should read the corresponding RxCSR1 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 MUSBFDRC 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 packet sizes should not exceed the size specified in the RxMaxP register (as this should be the value set in the *wMaxPacketSize* field of the appropriate endpoint descriptor). When a block of data larger than RxMaxP is required, it will need to be sent as multiple packets – each requiring a separate IN token. The CPU will therefore have to repeat the above steps for each packet that is to be sent.

Assuming RxMaxP has been set correctly, all the packets will be RxMaxP 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 RxMaxP in size. (If the total size of the data block is a multiple of RxMaxP, the last packet will still be less than RxMaxP in size as a null data packet will be sent after the data to signify that the transfer is complete.)

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. See Section 9.2 of the MUSBFDRC User Guide.

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 (RxCSR1.D6) being set.

8.2.2. BULK OUT TRANSACTION

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

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, 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.) 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 an external DMA controller to load packets into the FIFO without processor intervention.

- **AutoSet**

When the AutoSet feature is enabled, the TxPktRdy bit (TxCSR1.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.

SETUP

Before initiating any Bulk OUT transactions:

- The TxType register for the MUSBFDRC endpoint that is to be used needs to be written with 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 MUSBFDRC during device enumeration.
- The TxMaxP register for the MUSBFDRC 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 – 255 ms), or set to zero if the NAK Timeout feature is not required.
- The relevant interrupt enable bit in the IntrTx1E/IntrTx2E register should be set to 1 (if an interrupt is required for this endpoint)
- The following bits of TxCSR2 register should be set as shown below:

D7	**AutoSet**	0/1	Set to 1 if the AutoSet feature is required.
D5	**Mode**	1	Set to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an OUT endpoint).
D4	**DMAEnab**	0/1	Set to 1 if a DMA request is required for this endpoint.
D3	**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 TxCSR1 register should be written to set the ClrDataTog bit (D6). This will ensure that the data toggle (which is handled automatically by the MUSBFDRC) starts in the correct state. Also, if there are any data packets in the FIFO (indicated by the

FIFONotEmpty bit (TxCSR1.D1) being set), they should be flushed by setting the FlushFIFO bit (TxCSR1.D3). *Note:* It may be necessary to set this bit twice in succession if double buffering is enabled.

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 TxCSR1 register (D0). The MUSBFDRC 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 MUSBFDRC will clear TxPktRdy (TxCSR1.D0). If the USB peripheral responds with a STALL, RxStall (TxCSR1.D5) is set. If a NAK is received, the MUSBFDRC 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, a further two attempts are made before the MUSBFDRC reports an error (RxCSR1.D2 set).

The MUSBFDRC then generates the appropriate endpoint interrupt, whereupon the CPU should read the corresponding TxCSR1 register to determine whether the RxStall, Error or NAK Timeout bit is set and act accordingly. (If the NAK Timeout bit is set, the MUSBFDRC 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.)

The packet sizes should not exceed the size specified in the TxMaxP register (as this should be 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 be TxMaxP in size, except the last packet which holds the residue. If the total size of the data block is a multiple of TxMaxP, the host will need to send a null packet after all the data has been sent. This is 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. See Section 9.1 of the MUSBFDRC User Guide.

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 (TxCSR1.D5) being set.

9. INTERRUPT TRANSACTIONS

Interrupt transactions are handled in a similar way to Bulk transactions, both when the MUSBFDRC is acting as a peripheral and when it is acting as a host. There are, however, a few special points to note. In particular, though DMA can be used, it offers little benefit as Interrupt endpoints are usually expected to transfer all their data in a single packet. This applies in all cases.

9.1. INTERRUPT TRANSACTIONS AS A PERIPHERAL

An Interrupt IN transaction uses the same protocol as a Bulk IN transaction (described in Section 8.1.1) 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 8.1.2) and can be used the same way.

You should however note that Tx endpoints on a MUSBFDRC used in Peripheral mode for Interrupt IN transactions support one feature 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 TxCSR2 register (D3). When this bit is set to 1, the MUSBFDRC 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.

9.2. INTERRUPT TRANSACTIONS AS A HOST

When operating as the host, transactions with an Interrupt endpoint on the USB peripheral are handled in very much the same way as the equivalent Bulk transactions (described in Sections 8.2.1 and 8.2.2, respectively). 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 represent an Interrupt transaction) rather than to 10. The required polling interval also needs to be set in the RxInterval/TxInterval registers.

10. ISOCHRONOUS TRANSACTIONS

10.1. HANDLING ISOCHRONOUS TRANSACTIONS AS A PERIPHERAL

10.1.1. ISOCHRONOUS IN TRANSACTIONS

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

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 3.) 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 Underrun errors (see 'Operation' below).

- **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 an external 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 TxCSR1 register needs to be accessed following every packet to check for Underrun errors.

- **AutoSet**

When the AutoSet feature is enabled, the TxPktRdy bit (TxCSR1.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 TxCSR1 register needs to be accessed following every packet to check for Underrun errors.

SETUP

In configuring a Tx endpoint for Isochronous 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 IntrTx1E/IntrTx2E register should be set to 1 (if an interrupt is required for this endpoint) and the TxCSR2 register should be set as shown below (Bits D2 – D0 of TxCSR2 are unused.).

D7	**AutoSet**	0/1	Set to 1 if the AutoSet feature is required.
D6	**ISO**	1	Set to 1 to enable Isochronous protocol.
D5	**Mode**	1	Set to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).
D4	**DMAEnab**	0/1	Set to 1 if a DMA request is required for this endpoint.
D3	**FrcDataTog**	0	Ignored in Isochronous mode.

OPERATION

An Isochronous endpoint does not support data retries, so if data underrun is to be avoided, the data to be sent to the host must be loaded into the FIFO before the IN token is received. The host will send one IN token per frame, however the time within the frame 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 an 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 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 TxCSR1 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 either the SOF interrupt (IntrUSB.D3) or the external SOF_PULSE signal from the MUSBFDRC to trigger the loading of the next data packet. The SOF_PULSE is generated once per frame when a SOF packet is received. (The MUSBFDRC also maintains an external frame 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 TxCSR1 (D0) and to check for data overruns/underruns (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 after it is loaded. There is no problem if the function loads the first data packet at least a frame 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 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.

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 TxCSR1 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 and it finds that the TxPktRdy bit in the TxCSR1 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 TxCSR1 register (D3), or it may choose to skip the current packet.

10.1.2. ISOCHRONOUS OUT TRANSACTIONS

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

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 3.) 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 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 an external 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 RxCSR1 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 (RxCSR1.D0) will be automatically cleared when a packet of RxMaxP bytes has been unloaded from the FIFO. However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size and the RxCSR1 register needs to be accessed following every packet to check for Overrun or CRC errors.

SETUP

In configuring an Rx endpoint for Isochronous 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 IntrRx1E/IntrRx2E register should be set to 1 (if an interrupt is required for this endpoint) and the following bits of the RxCSR2 register should be set as shown:

D7	**AutoClear**	0/1	Set to 1 if the AutoClear feature is required.
D6	**ISO**	1	Set to 1 to enable Isochronous protocol.
D5	**DMAEnab**	0/1	Set to 1 if a DMA request is required for this endpoint.

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, however the time within the frame can vary. If a packet is received near the end of one frame 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 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 RxCSR1 register (D0) in the same way as for a Bulk Rx endpoint. As the interrupt could occur almost any time within a frame, 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 before unloading the FIFO. This can be done by using either the SOF interrupt (IntrUSB.D3) or the external SOF_PULSE signal from the MUSBFDRC to trigger the unloading of the data packet. The SOF_PULSE is generated once per frame when a SOF packet is received. (The MUSBFDRC also maintains an external frame 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 RxCSR1 and to check for data overruns/underruns (see 'Error Handling' below).

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 RxCSR1 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 determine how this error condition is handled.

If the MUSBFDRC finds that a received packet has a CRC error, it will still store the packet in the FIFO and set the RxPktRdy bit (OutCSR1.D0) and the DataError bit (RxCSR1.D3). It is left up to the application how this error condition is handled.

10.2. HANDLING ISOCHRONOUS TRANSACTIONS AS A HOST

10.2.1. ISOCHRONOUS IN TRANSACTIONS

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

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 3.).

- **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 an external 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 (RxCSR1.D0) will be automatically cleared when a packet of RxMaxP bytes has been unloaded from the FIFO. However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size.

- **AutoReq(uest)**

When the Autoreq(uest) feature is enabled, the ReqPkt bit (RxCSR1.D5) will be automatically set when the RxPktRdy bit is cleared.

SETUP

Before initiating an Isochronous IN Transaction:

- The RxType register for the MUSBFDRC endpoint that is to be used needs to be written with 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 MUSBFDRC during device enumeration.
- The RxInterval register for the MUSBFDRC endpoint needs to be written with the required transaction interval (usually 1 for one transaction every 1ms).
- The RxMaxP register for the MUSBFDRC 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 IntrRx1E/IntrRx2E register should be set to 1 (if an interrupt is required for this endpoint)

- The following bits of RxCSR2 register should be set as shown below:

D7	**AutoClear**	0/1	Set to 1 if the AutoClear feature is required.
D6	**AutoReq**	0	Set to 1 to enable automatic setting of the ReqPkt bit whenever RxPktRdy is cleared.
D5	**DMAEnab**	0/1	Set to 1 if a DMA request is required for this endpoint.

OPERATION

The operation starts with the CPU setting ReqPkt (RxCSR1.D5). This causes the MUSBFDRC 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 RxCSR1 register (D0) in the same way as for a Bulk Rx endpoint. As the interrupt could occur almost any time within a frame, 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.

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 will increase or decrease from frame to frame 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.

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 (RxCSR1.D3) is set to indicate that the data may be corrupt.

10.2.2. ISOCHRONOUS OUT TRANSACTIONS

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

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 3.). 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, the TxPktRdy bit (TxCSR1.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.

SETUP

Before initiating an Isochronous OUT Transaction:

- The TxType register for the MUSBFDRC endpoint that is to be used needs to be written with 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 MUSBFDRC during device enumeration.
- The TxInterval register for the MUSBFDRC endpoint needs to be written with the required transaction interval (usually 1 for one transaction every 1ms).
- The TxMaxP register for the MUSBFDRC 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 IntrTx1E/IntrTx2E register should be set to 1 (if an interrupt is required for this endpoint)
- The following bits of TxCSR2 register should be set as shown below:

D7	**AutoSet**	0/1	Set to 1 if the AutoSet feature is required.
D5	**Mode**	1	Set to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an OUT endpoint).
D4	**DMAEnab**	0/1	Set to 1 if a DMA request is required for this endpoint.
D3	**FrcDataTog**	0	Set to 0 to allow normal data toggle operation.

OPERATION

The operation starts when the CPU writes to the FIFO then sets TxPktRdy (TxCSR1.D0). This triggers the MUSBFDRC 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 TxCSR1 register (D0) in the same way as for a Bulk Tx endpoint. However, as the interrupt could occur almost any time within a frame, 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.

The AutoSet feature can be used with an Isochronous Tx endpoint, in the same way it can be used with a Bulk Tx endpoint. However, unless the data arrives from the source at an absolutely consistent rate, synchronized to the MUSBFDRC's frame clock, the size of the packets will have to increase or decrease from frame to frame 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.

11. CONNECT/DISCONNECT

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

11.1. IN HOST MODE

Where the MUSBFDRC 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). Whether a full-speed or a low-speed device has been connected can be determined by reading the DevCtl register where the FSDev bit (D6) will be high for a full-speed device and the LSDev bit (D5) will be high for a low-speed device.

The CPU should then reset the device and 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).

11.2. IN PERIPHERAL MODE

Where the MUSBFDRC 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.

While a session is in progress, the MUSBFDRC can also be made to execute a 'soft' disconnect by clearing the Session bit (DevCtl.D0). The MUSBFDRC just disconnects its pull-up resistor but, as far as the host device is concerned, it is as if the MUSBFDRC has been detached from the bus. The host then ends the session.

12. 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.

VBus is always supplied by the 'A' device on the bus. The MUSBFDRC determines whether it is the 'A' device by monitoring the CID input. This pin should be connected to the ID pin on the mini-AB receptacle. CID will be pulled low when an 'A' plug is inserted, signifying that the MUSBFDRC is the 'A' device.

Whether the MUSBHDRC is the 'A' device or the 'B' device can be determined from the setting of the CID bit (DevCtl.D7) which is '0' when the MUSBHDRC is the 'A' device and '1' when it is the 'B' device.

12.1. WHEN MUSBFDRC IS THE 'A' DEVICE

Starting a Session: When the MUSBFDRC is the 'A' device on the bus, the CPU can start a session by setting the Session bit in the DevCtl register (D0). The MUSBFDRC will then turn on VBus and wait for VBus to go above the VBus Valid threshold, as indicated by the VBUSVAL input going high. (This event also causes the VBus Val bit in the Power register (D6) to go high.)

The MUSBFDRC will then enter Host mode (the 'A' device is always the default host) and wait for a peripheral to be connected. A Connect interrupt (IntrUSB.D4) will be generated (if enabled) when a peripheral is detected and either the FSDev or LSDev bit in the DevCtl register (D6 or D5) will be set depending on whether a full-speed or low-speed peripheral was detected. The CPU may then reset and enumerate the device.

To end the session, the CPU should clear the Session bit (DevCtl.D0). When the Session bit is cleared, CLKOUT will be stopped and the PHY will go into Suspend mode.

Detecting activity: If the MUSBFDRC detects data line pulsing on the bus when a session is not in progress, 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.

12.2. WHEN MUSBFDRC IS THE 'B' DEVICE

Detecting activity: When the MUSBFDRC is the 'B' device on the bus and it detects that VBus goes above the Session Valid threshold (as indicated by the VBUSSES input going high and reflected by the VBus Sess bit in the Power register (D5)), it will set 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 MUSBFDRC will be in Peripheral mode at this point as the 'B' device is the default peripheral.

When VBus drops below the Session Valid threshold, the Session bit will be cleared and a Disconnect interrupt (IntrUSB.D5) will be generated (if enabled) to indicate that the session has ended.

Starting a Session: If the CPU wants to start a session, it should set the Session bit (DevCtl.D0). This will cause the MUSBFDRC to request a session using the Session Request Protocol defined in the *USB On-The-Go* supplement.

The MUSBFDRC will wait for the initial conditions to be met (VBus below the Session End threshold and SE0 for > 2 ms) then it will try pulsing the data line, then – if this fails – it will try pulsing VBus (by taking VBUSCHG high).

Once started, the session will normally be ended by the host device. However, with the MUSBFDRC, it is also possible for the peripheral device to force the session to end by clearing the Session bit. This causes the MUSBFDRC to execute a 'soft' disconnect from the USB bus, disconnect its pull-up resistor and stop CLKOUT (as a consequence of which, the PHY will go into Suspend mode). The host device will then end the current session.

Note: It is possible that the CPU may receive a spurious Disconnect interrupt after requesting a Session Start. This can occur if VBus pulsing takes VBus above the Session Valid threshold, but VBus then falls below the Session Valid threshold before the 'A' device starts the session.

13. HOST NEGOTIATION

When the MUSBFDRC is the 'A' device (CID = 0), it will automatically enter Host mode when a session starts.

When the MUSBFDRC is the 'B' device (CID = 1), it will automatically enter Peripheral mode when a session starts. The CPU can however request that the MUSBFDRC becomes the Host by setting the Host Req bit in the DevCtl register (D1). This bit can be set either when the CPU requests a Session Start by setting the Session bit (DevCtl.D0) or at any time after a session has started. When the MUSBFDRC 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 disconnect the pull-up resistor to begin host negotiation (as specified in the *USB On-The-Go* supplement). This should cause the 'A' device to switch to Peripheral mode and connect its own pull-up resistor. When the MUSBFDRC detects this, it will generate a Connect interrupt (IntrUSB.D4) if enabled. It will also set the Reset bit in the Power register (D3) to begin resetting the 'A' device. (The MUSBFDRC 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 MUSBFDRC-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 MUSBFDRC-based, it will generate a Disconnect interrupt (IntrUSB.D5) if enabled.

14. VBUS EVENTS

The USB On-The-Go specification defines a series of thresholds:

- VBus Valid (required to be greater than 4.4V)
- Session Valid (required to be between 0.8V and 4V)
- Session End (required to be between 0.2V and 0.8V)

to which the devices involved in point-to-point communications are required to respond

(The actual thresholds used in a particular device are set through a series of comparators, external to the MUSBFDRC core, which take the corresponding VBUSVAL, VBUSSES and VBUSLO 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 MUSBFDRC 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.

ACTIONS AS AN 'A' DEVICE

VBus > VBus Valid with session initiated by MUSBFDRC (i.e. Session bit (DevCtl.D0) set). When VBus becomes greater than VBus Valid, the MUSBFDRC 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. When VBus becomes greater than Session Valid, the MUSBFDRC will generate a Session Request interrupt (IntrUSB.D6). The CPU should set the Session bit (DevCtl.D0). The MUSBFDRC 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. 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 MUSBFDRC will automatically terminate the session and generate a VBus Error interrupt (IntrUSB.D7).

ACTIONS AS AN 'B' DEVICE

VBus > Session Valid. This indicates activity from the 'A' device. The MUSBFDRC will set the Session bit (DevCtl.D0) and take PUCON high in order to connect the pull-up resistor to the D+ line. It will also set the PUCON bit in the DevCtl register (D4).

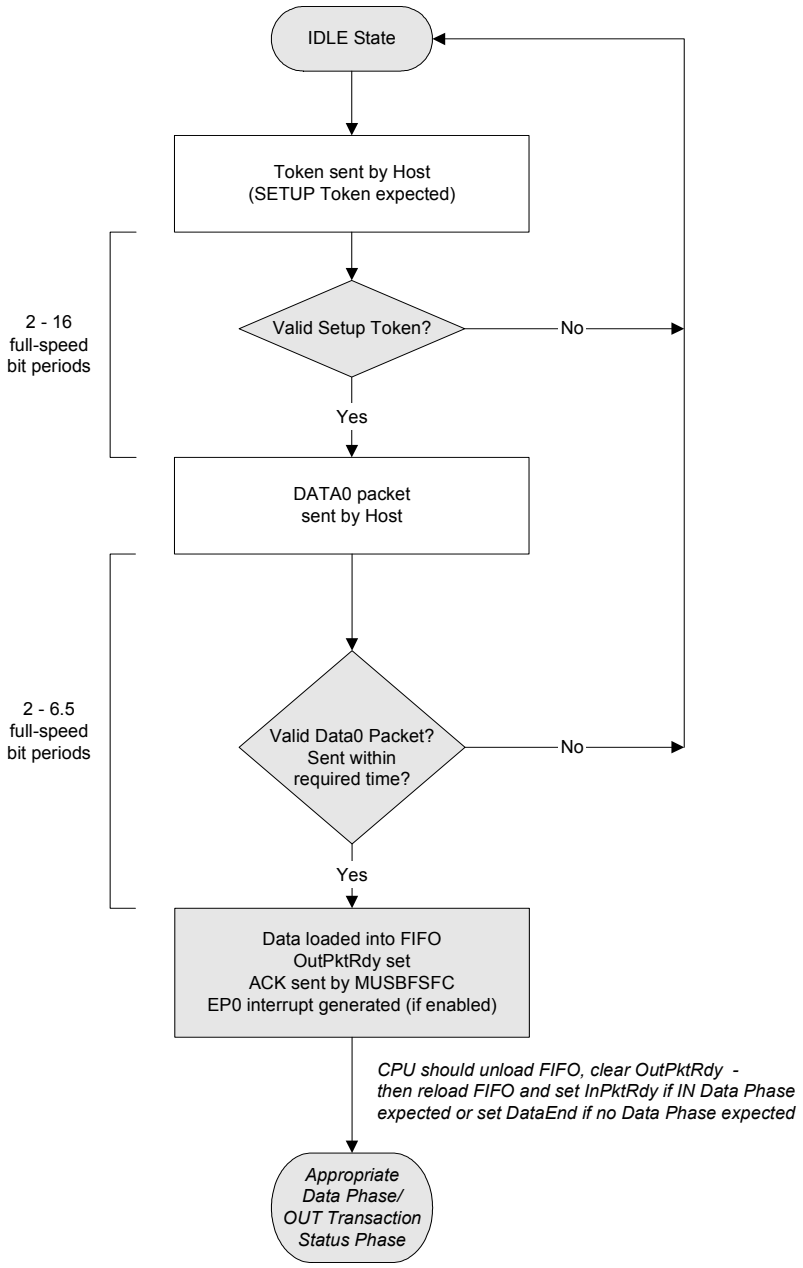
VBus < Session Valid while the Session bit remains set. This indicates that the 'A' device has lost power (or become disconnected). The MUSBFDRC will clear the Session bit (DevCtl.D0) and generate a Disconnect interrupt (IntrUSB.D5). The CPU should end the session.

VBus < Session End with the Host Request bit (DevCtl.D1) set). 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).

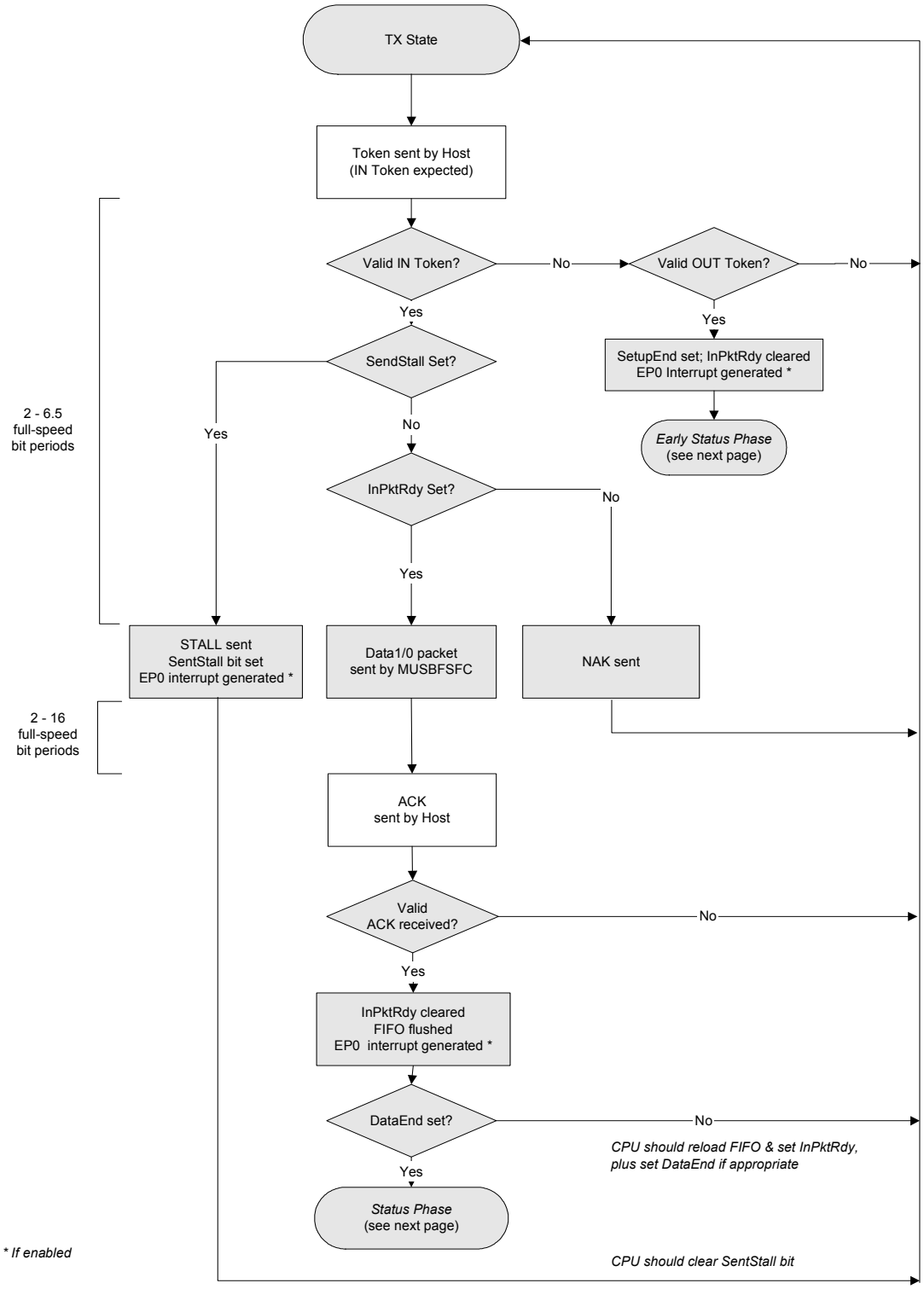
15. TRANSACTION FLOWS AS A PERIPHERAL

15.1. CONTROL TRANSACTIONS

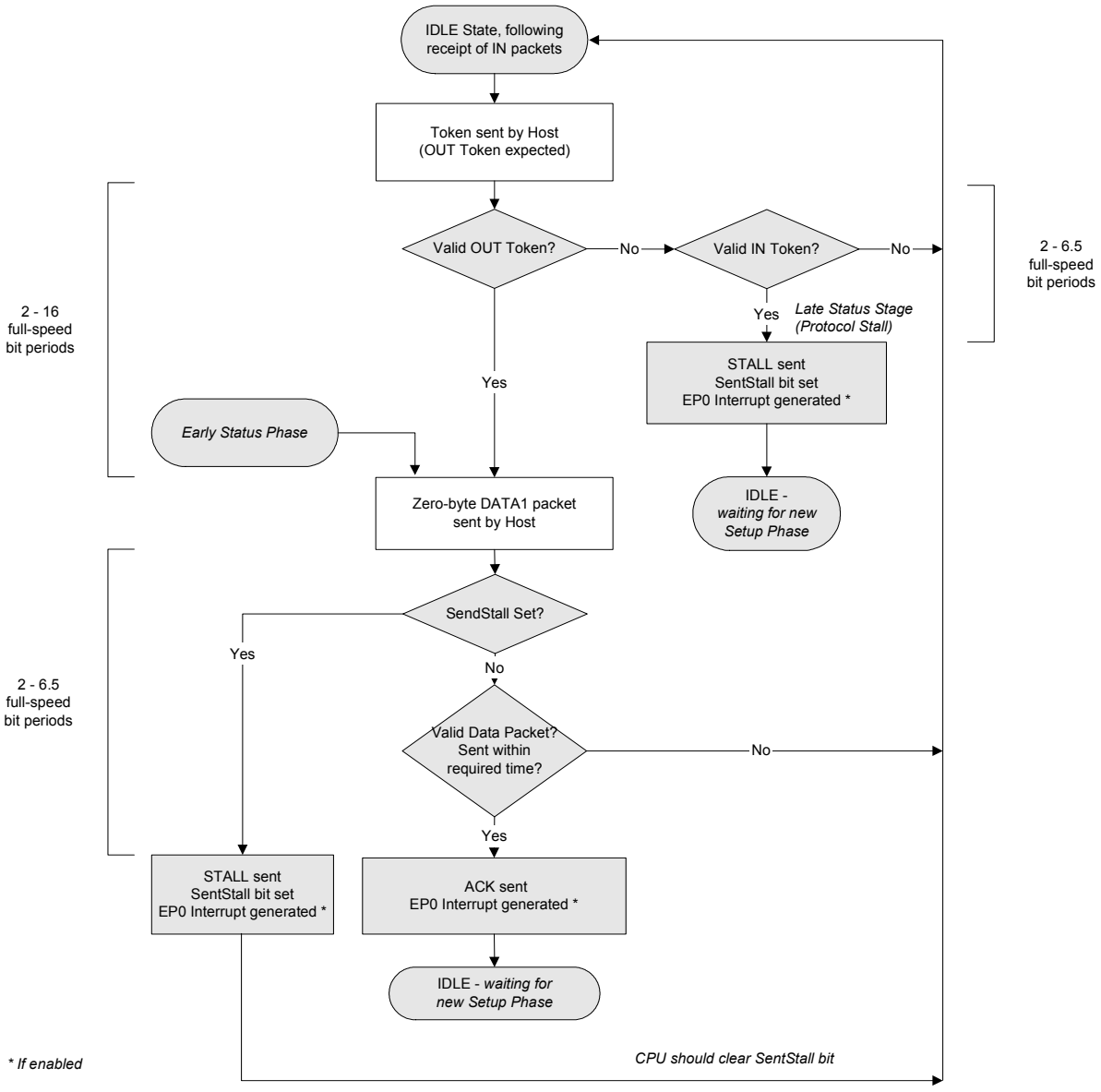
SETUP PHASE



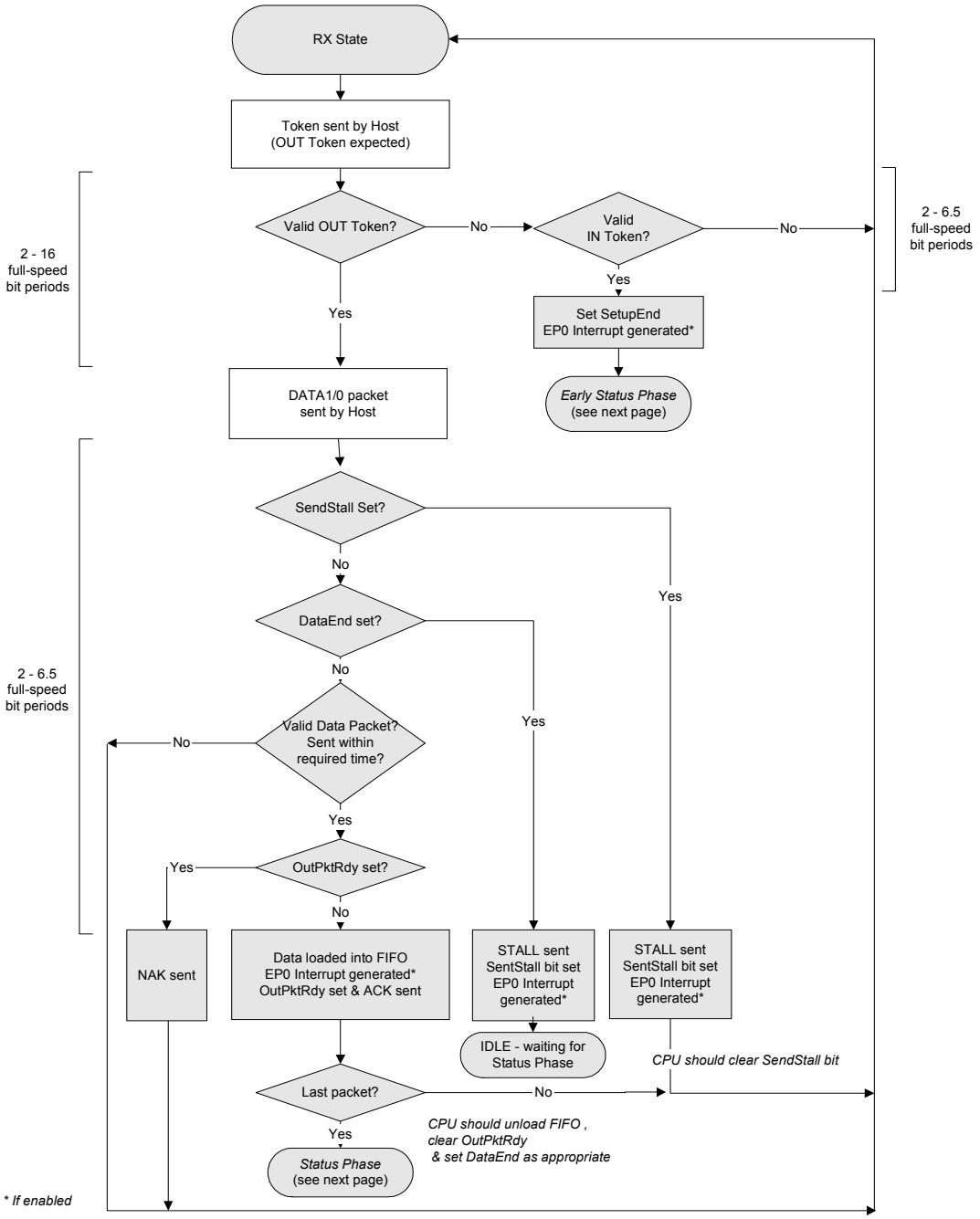
IN DATA PHASE ...



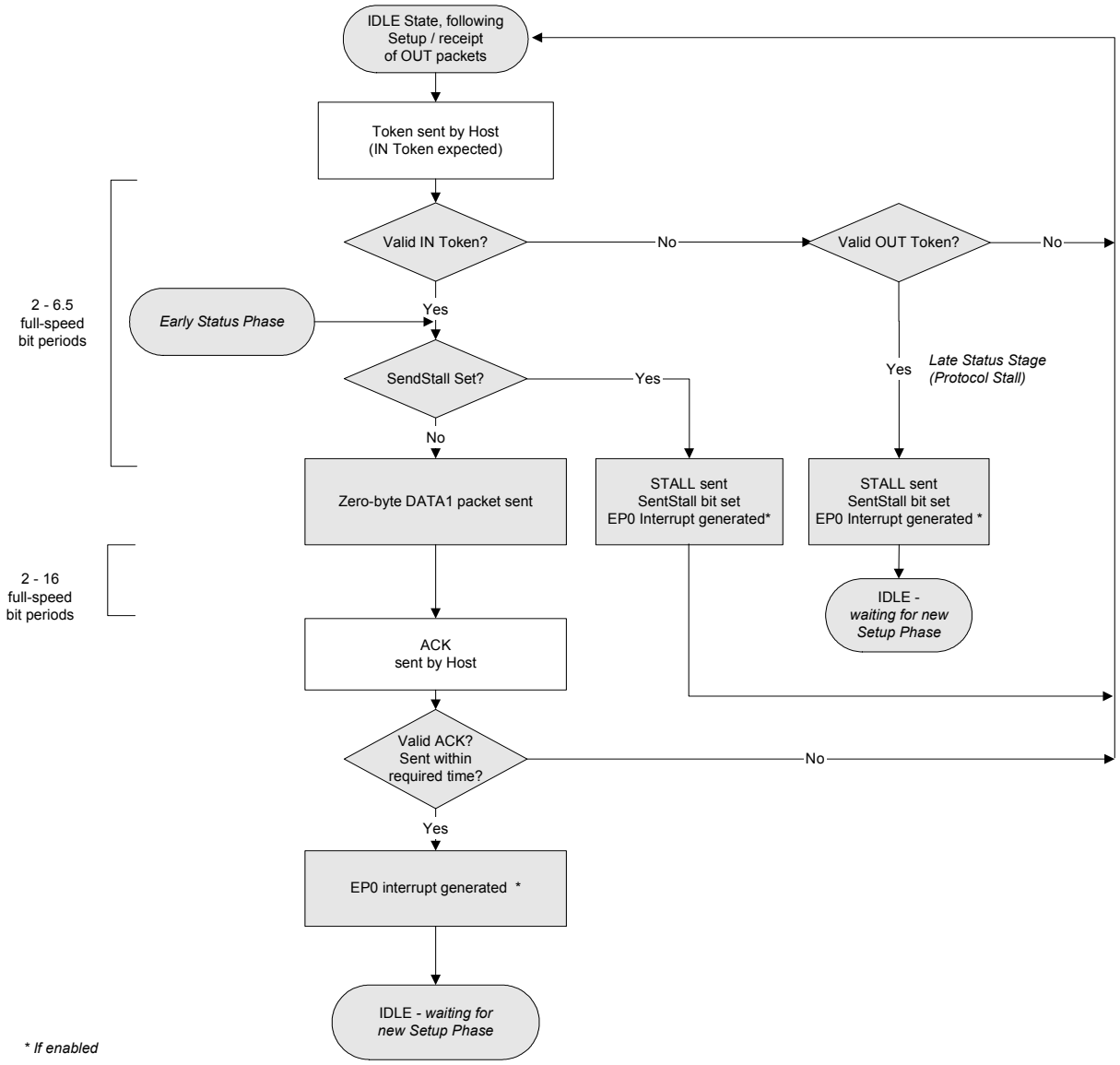
... AND FOLLOWING STATUS PHASE



OUT DATA PHASE...

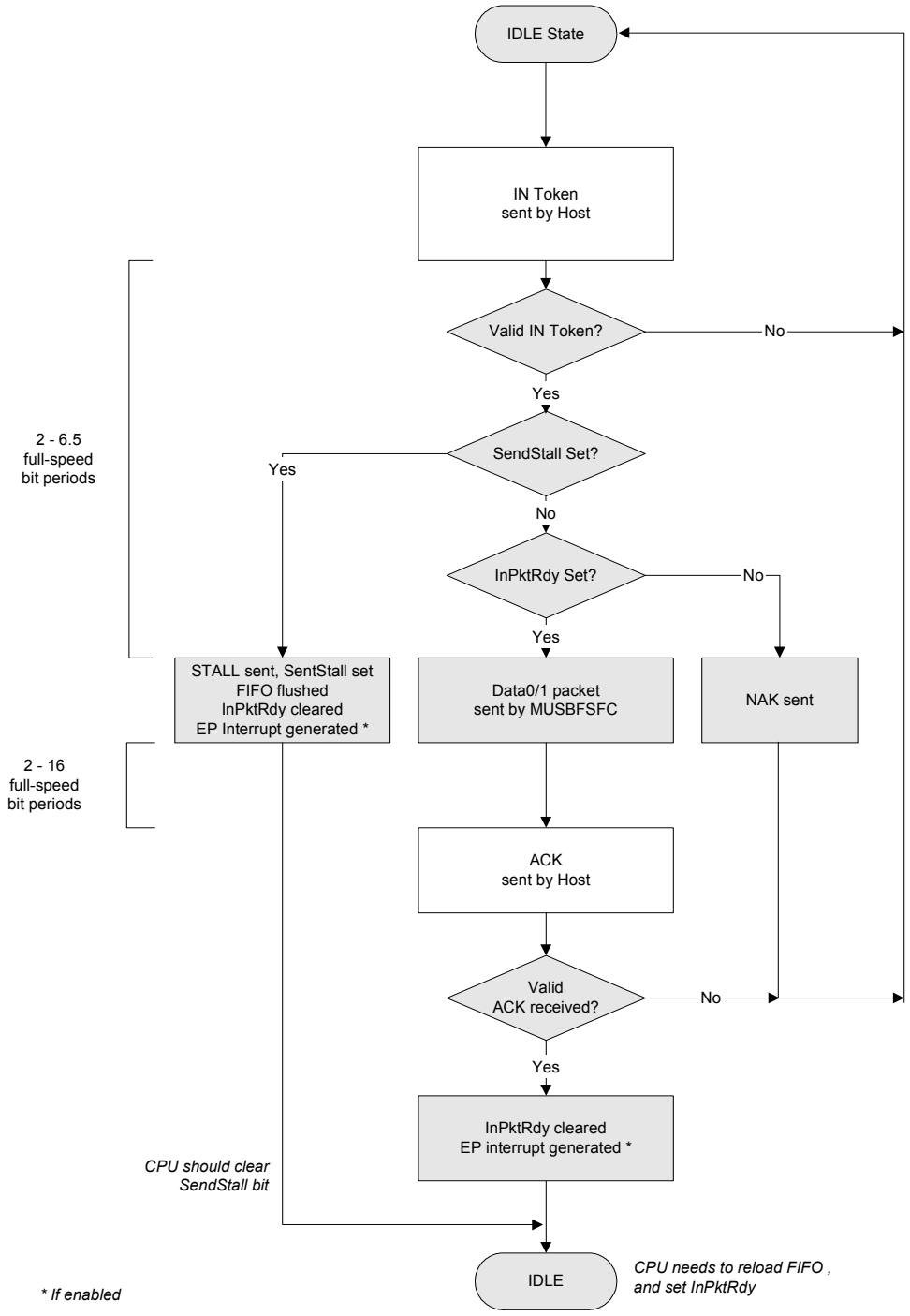


... AND FOLLOWING STATUS PHASE

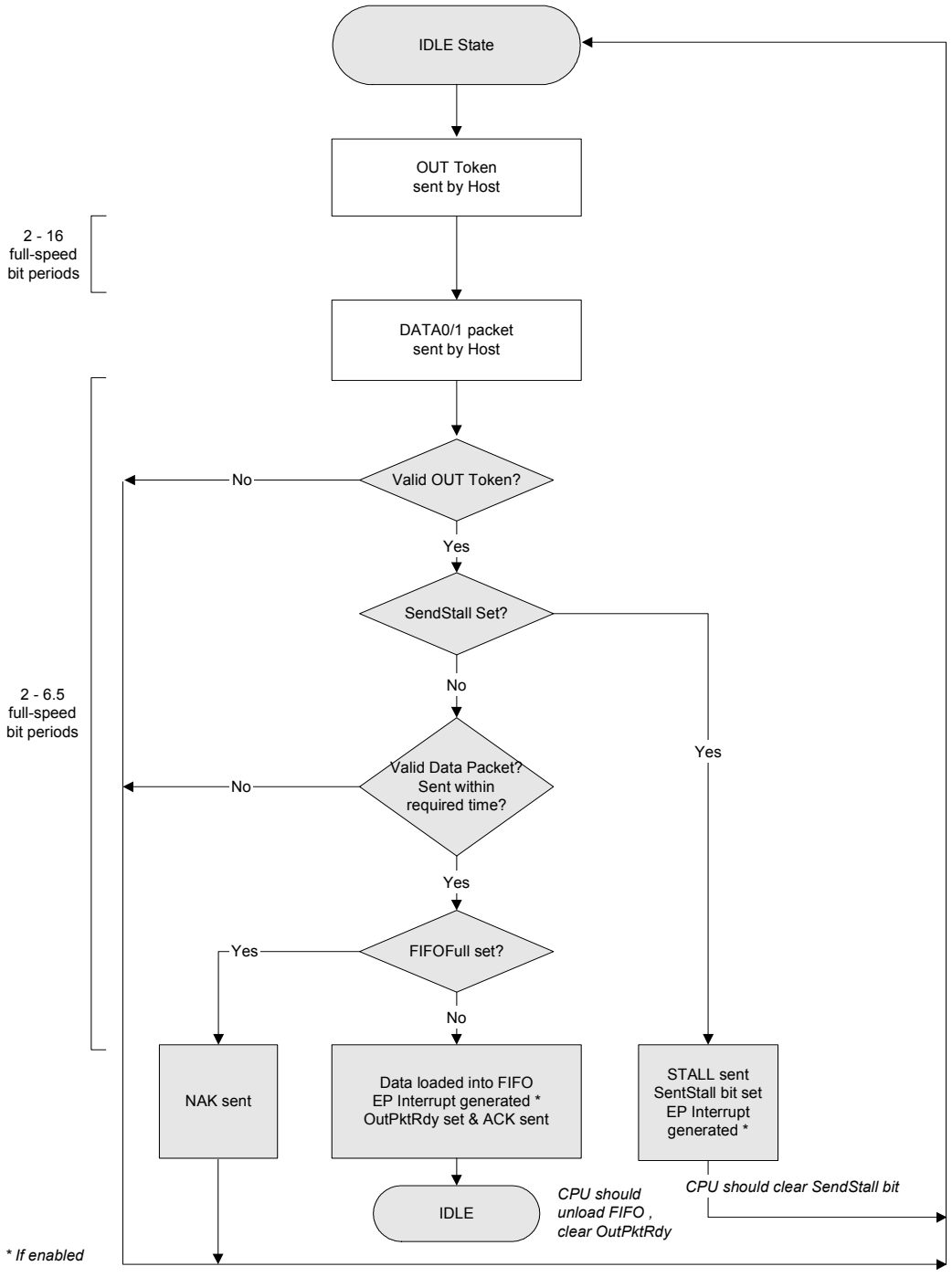


15.2. BULK/INTERRUPT TRANSACTIONS

IN TRANSACTION

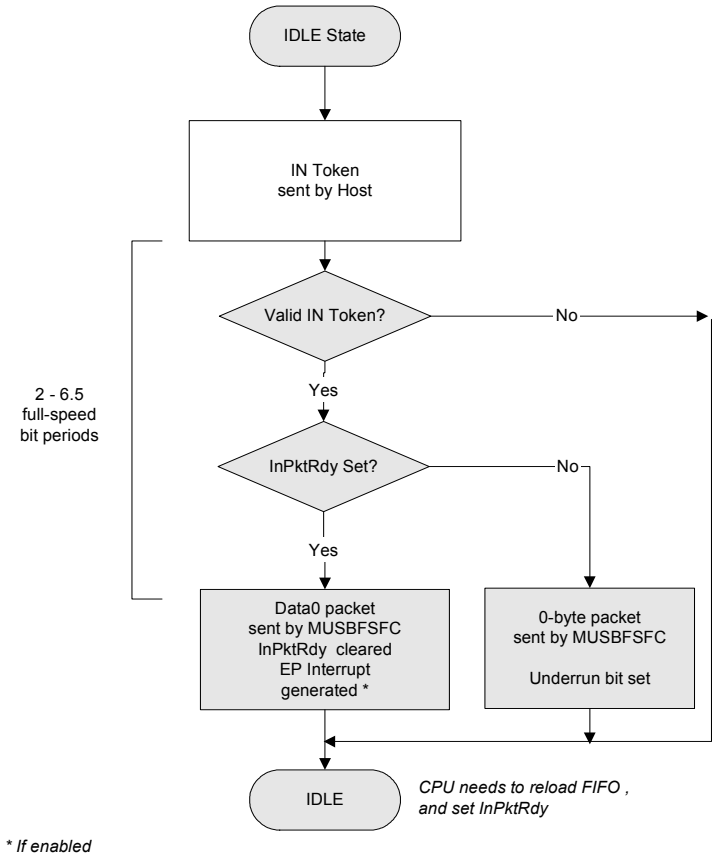


OUT TRANSACTION

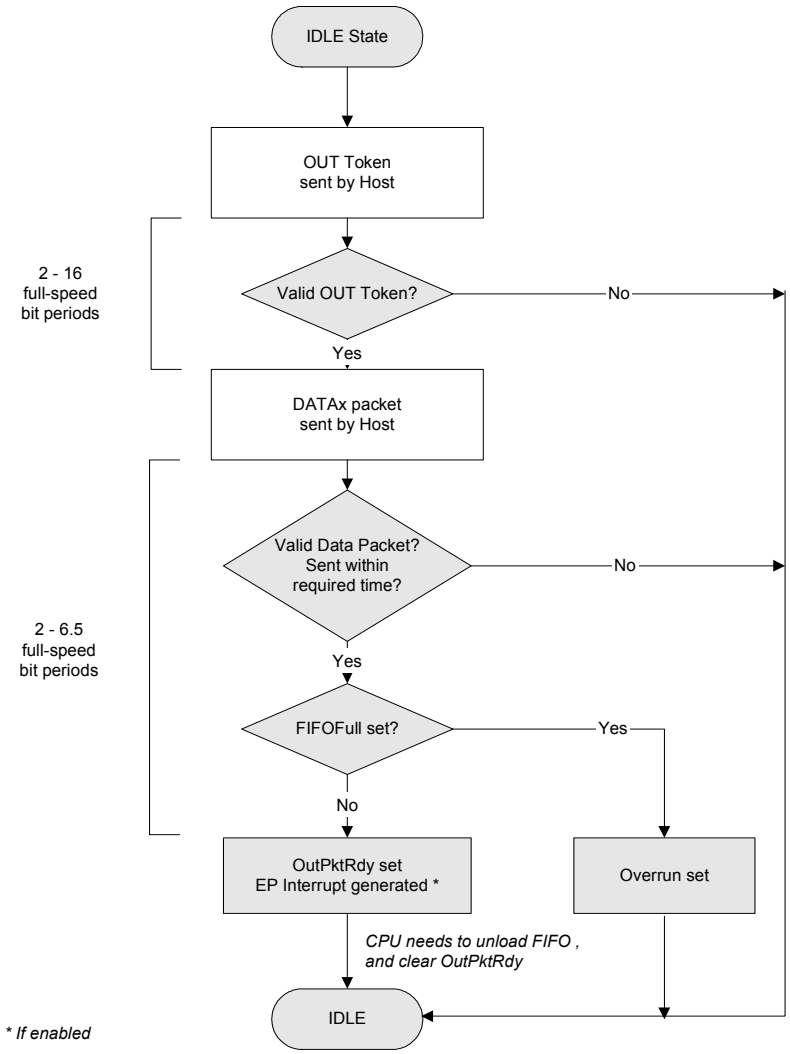


15.3. ISOCHRONOUS TRANSACTIONS

IN TRANSACTION



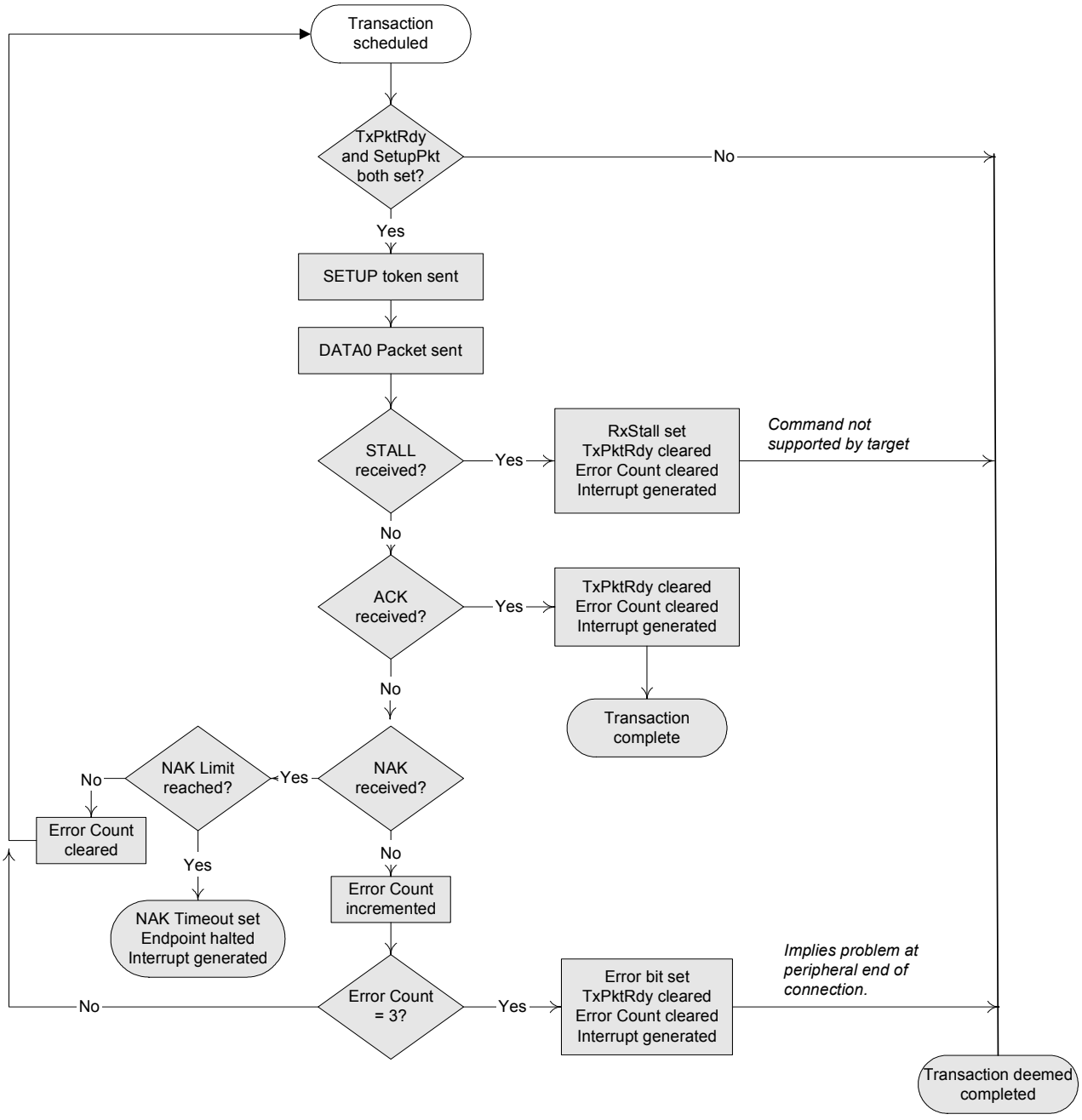
OUT TRANSACTION



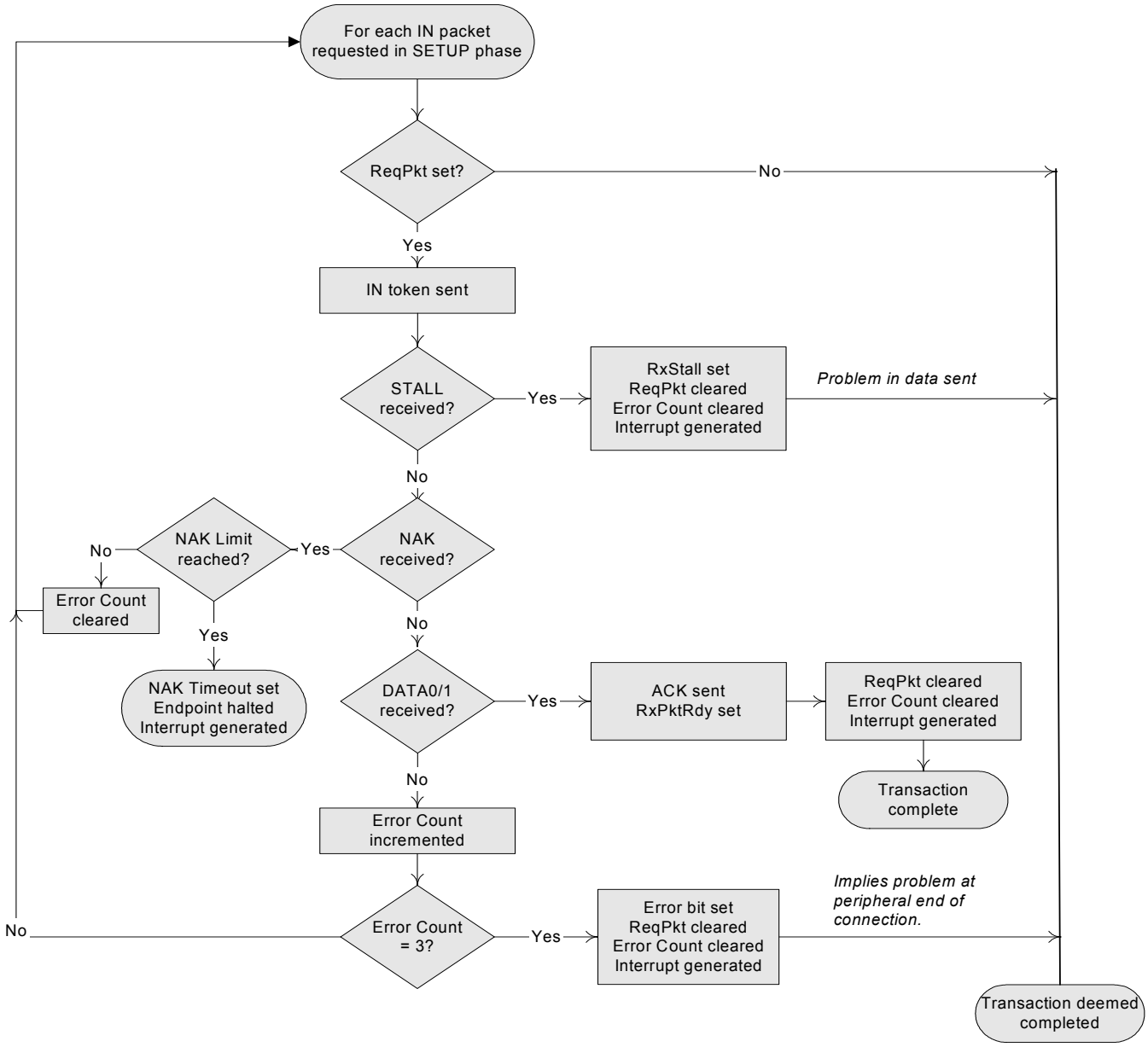
16. TRANSACTION FLOWS AS A HOST

16.1. CONTROL TRANSACTIONS

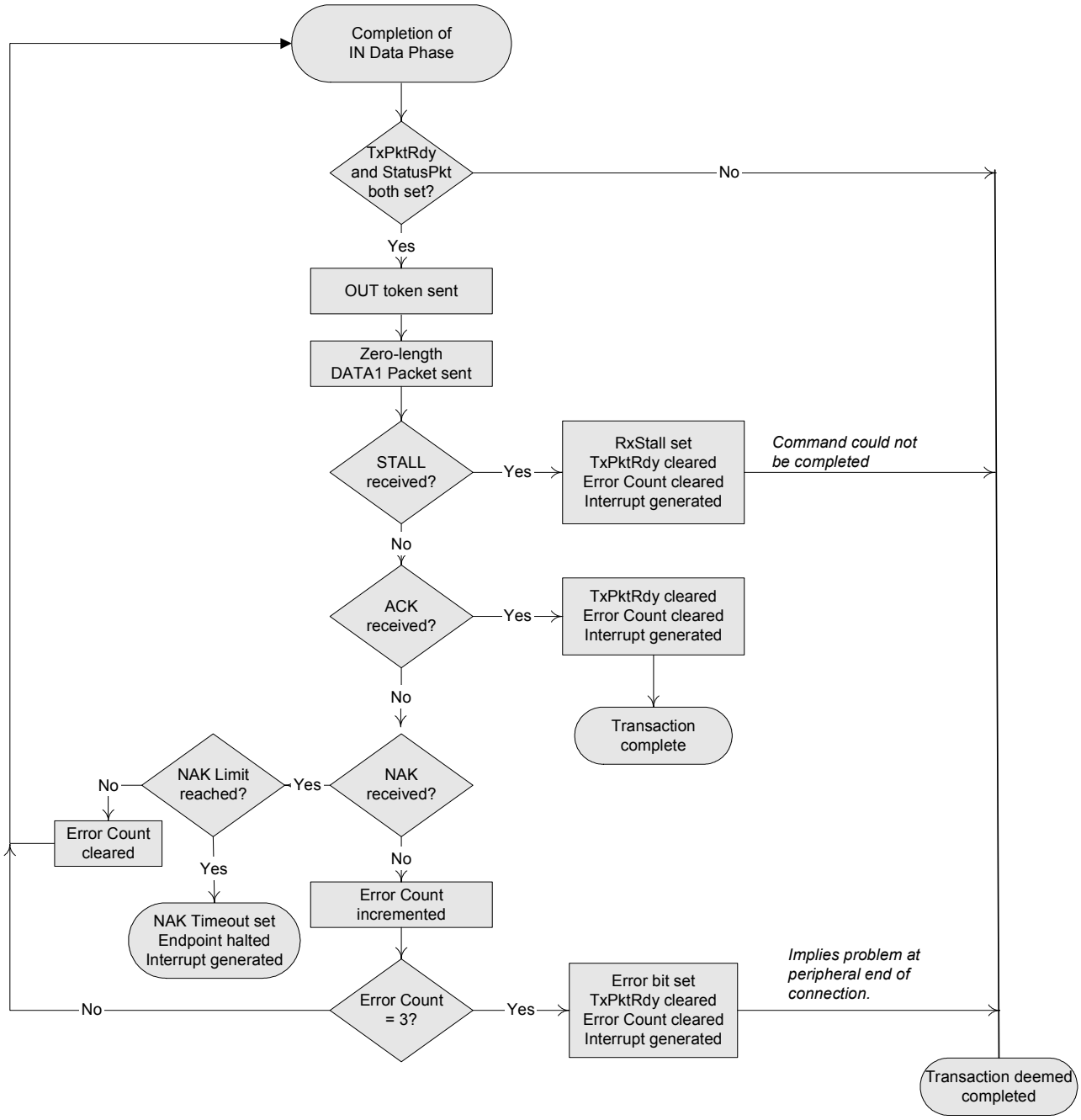
SETUP PHASE



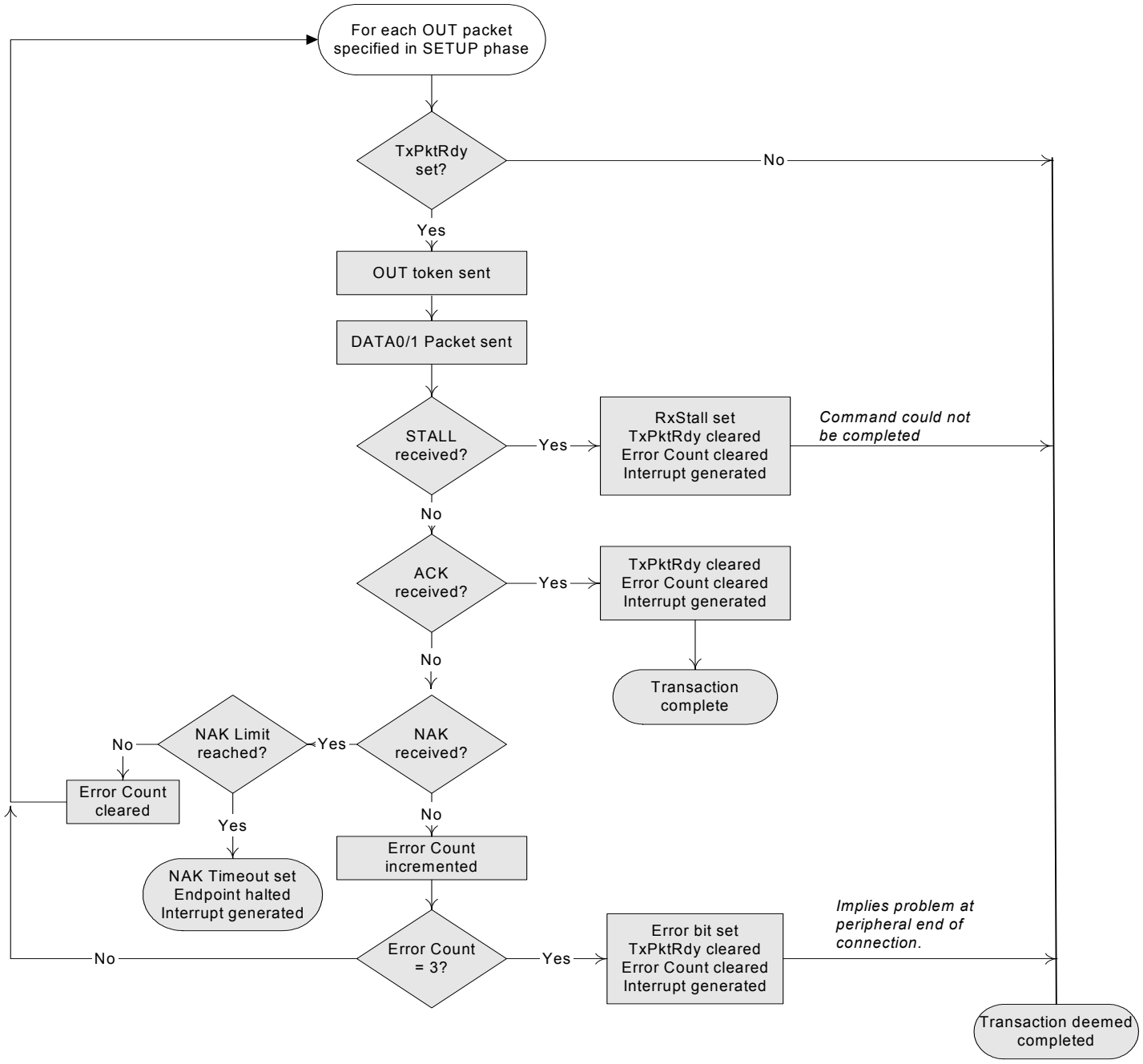
IN DATA PHASE...



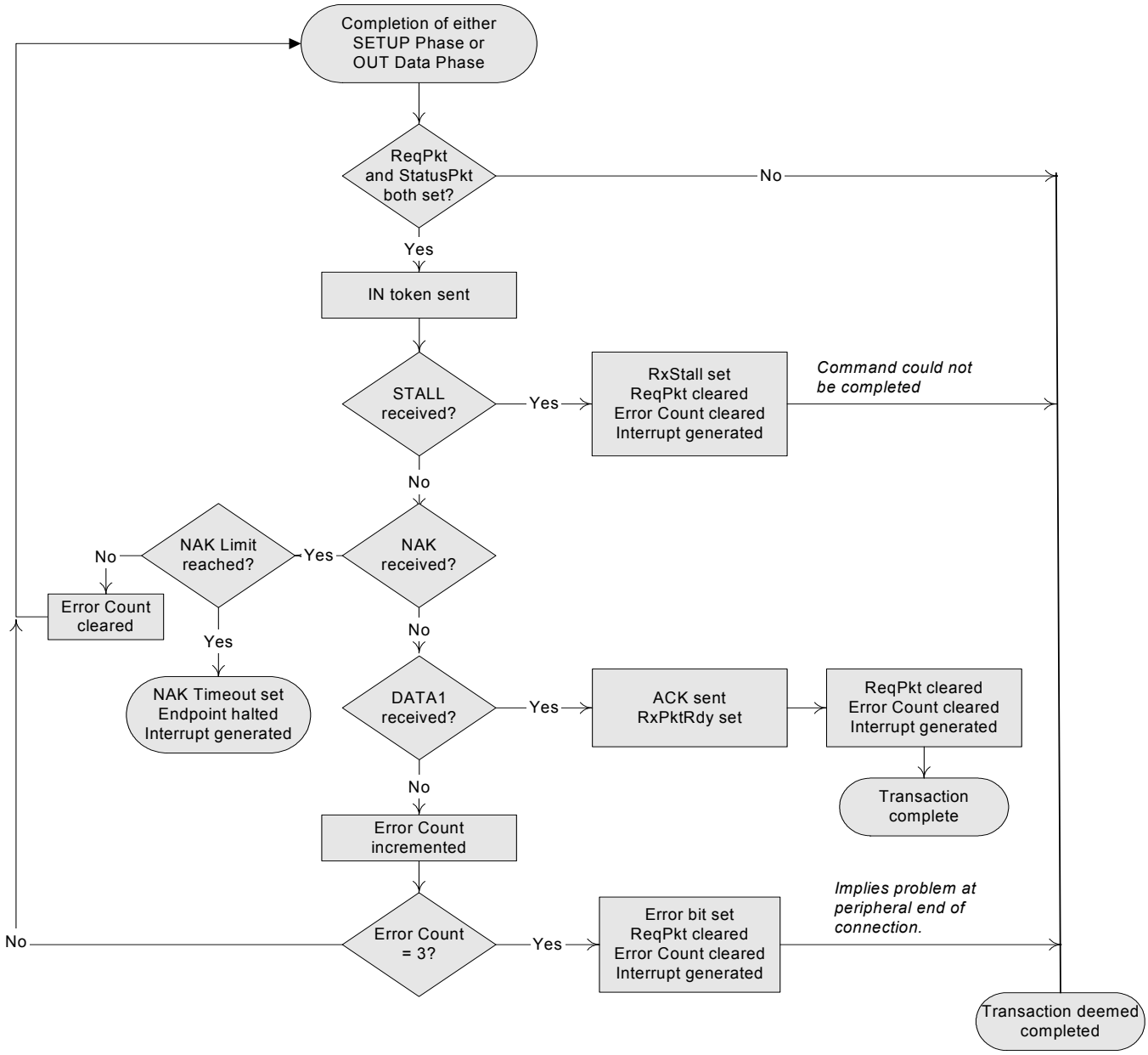
... AND FOLLOWING STATUS PHASE



OUT DATA PHASE...

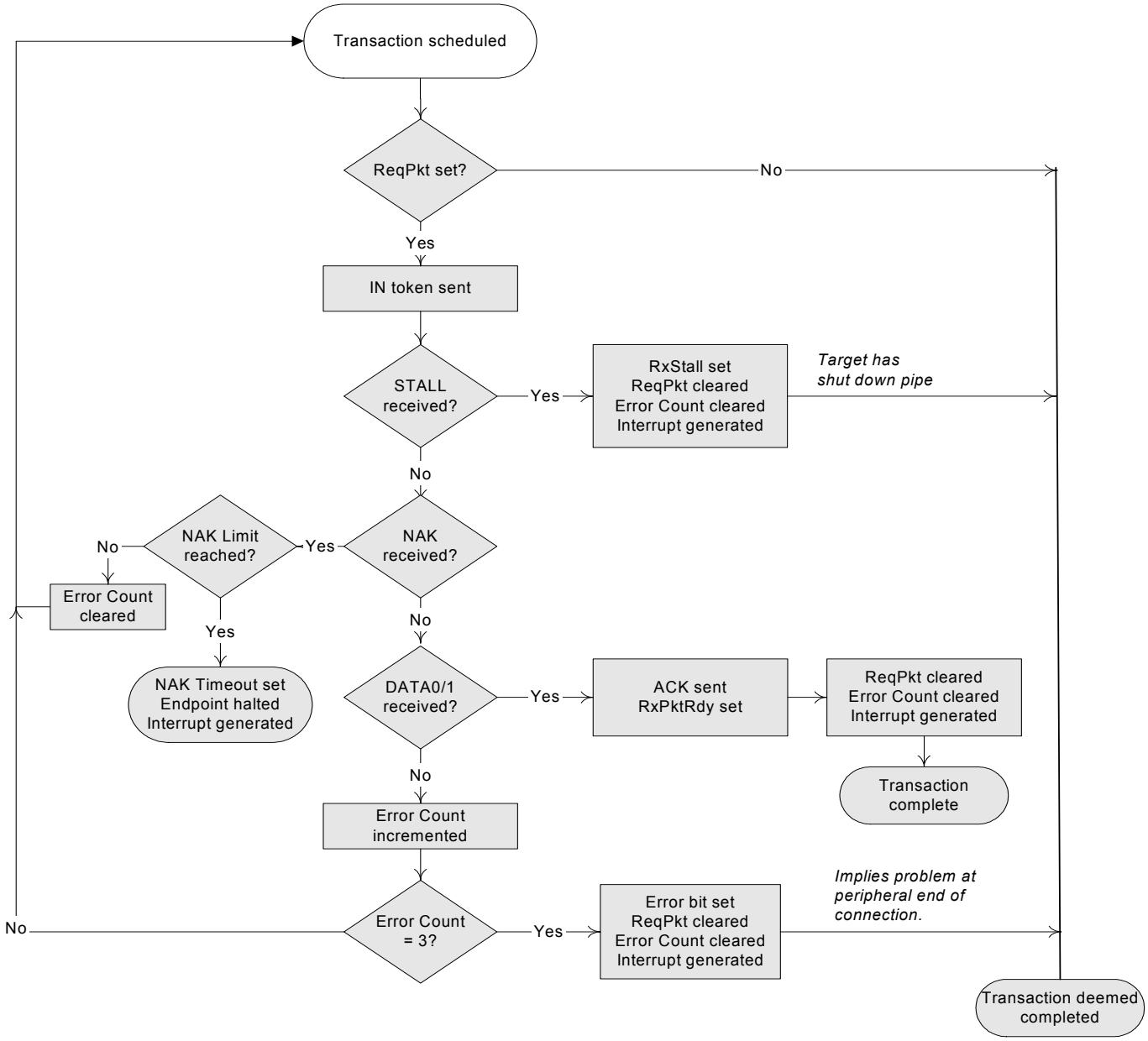


... AND FOLLOWING STATUS PHASE

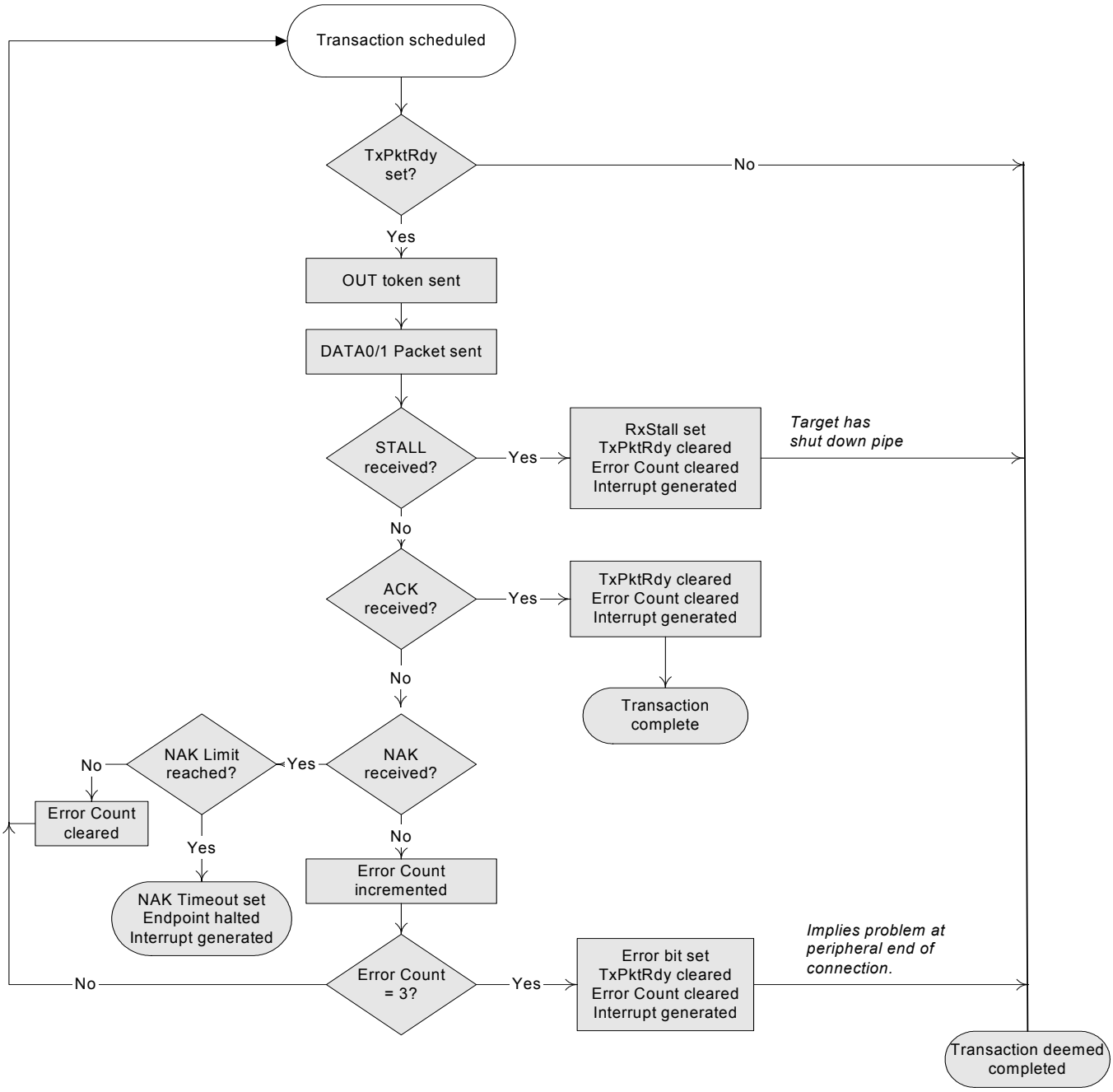


16.2. BULK/INTERRUPT TRANSACTIONS

IN TRANSACTION

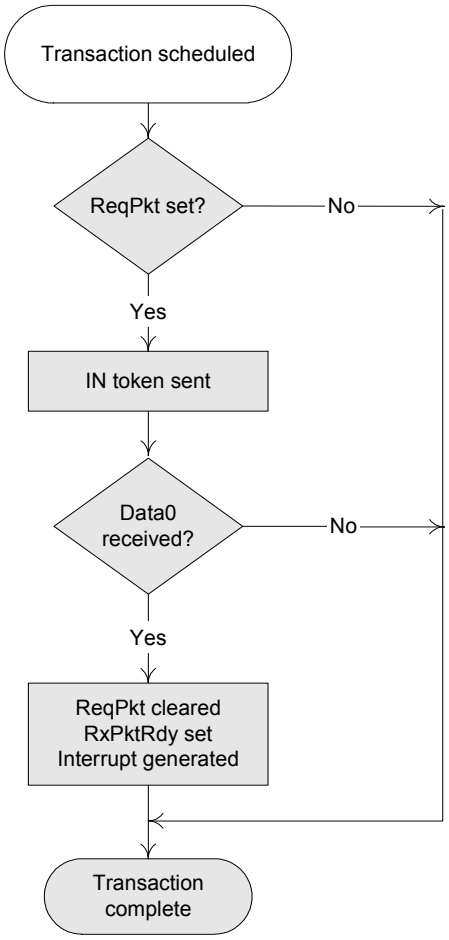


OUT TRANSACTION

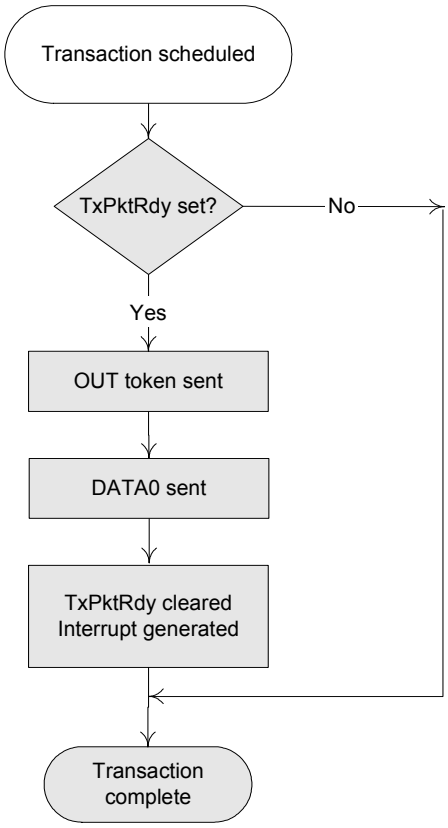


16.3. ISOCHRONOUS TRANSACTIONS

IN TRANSACTION



OUT TRANSACTION



17. HARDWARE CONFIGURATION READBACK

Information about the number of Tx and Rx endpoints have been configured, 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.

The FIFOSize register is included among the core's set of Indexed registers and is located at offset 0x1F. When read for endpoint numbers 1 – 15 (selected through the Index register), the register returns details of the hardware configuration for the selected Tx and Rx endpoints as follows:

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: This register does not return valid information where the Dynamic FIFO Sizing option is used. It also does not return valid information about Endpoint 0 for which there is a single 64-byte FIFO, used for both Rx and Tx transactions.

18. REVISION HISTORY

18.1. ISSUE 1

26th June 2002. First issue of document.

18.2. ISSUE 2

14th October 2002. Correction made to definition of TxInterval and RxInterval registers.

18.3. ISSUE 3

27th February 2003. Changes made following addition of Soft Disconnect feature, NAK Timeout and FIFOSize register to design.

18.4. ISSUE 4

21st May 2003. DMAMode bit added to TxCSR2 register. Enhancements made to descriptions of Soft Disconnect feature and FIFOs. Interrupt Service Routine flow diagram corrected (picture linking error).

18.5. ISSUE 5

25th November 2003. Corrections made to descriptions of AutoSet (TxCSR2.D7) and NAK Timeout (CSR0.D7; RxCSR1.D3). Clearing Session bit (DevCtl.D0) now also stops CLKOUT.

18.6. ISSUE 6

5th November 2007. Updated description for ReqPkt bit. Removed references to Inventra.

MUSBFDRC

http://www.mentor.com/supportnet

Corporate Headquarters
Mentor Graphics Corporation
8005 S.W. Boeckman Road
Wilsonville, OR 97070 USA
Phone: 503-685-7000
North American Support Center
Phone: 800-547-4303
Fax: 800-684-1795

Silicon Valley Headquarters
Mentor Graphics Corporation
1001 Ridder Park Drive
San Jose, California 95131 USA
Phone: 408-436-1500
Fax: 408-436-1501

Europe Headquarters
Mentor Graphics Corporation
Immeuble le Pasteur
13/15, rue Jeanne Braconnier
92360 Meudon La Foret
France
Phone: 33-1-40-94-74-74
Fax: 33-1-46-01-91-73

Pacific Rim Headquarters
Mentor Graphics (Taiwan)
Room 1603, 16F,
International Trade Building
No.333, Section 1, Keelung Road
Taipei, Taiwan, ROC
Phone: 886-2-87252000
Fax: 886-2-27576027

Japan Headquarters
Mentor Graphics Japan Co., Ltd.
Gotenyama Hills
7-35, Kita-Shinagawa 4-chome
Shinagawa-Ku, Tokyo 140
Japan
Phone: 81-3-5488-3033
Fax: 81-3-5488-3021

11/07 PG-40134.005-FC