User Guide: USB 2.0 Core

Rev. [6/2012]

6-19-2012

Table of Contents

General Description

Aldec USB 2.0 core is the RTL model of USB 2.0 Function Controller, which is fully compatible with the USB 2.0 specification. This core is available in VHDL along with behavioral testbench. The testbench gives code coverage at last 95% (statement, branch). The core has been optimized for popular FPGA devices and its functionality has been verified in the real hardware.

Features

- Fully compliant to USB 2.0 specification
- Supports full-speed 12Mbps and high-speed 480Mbps modes
- Supports USB 2.0 Transceiver Macrocell Interface (UTMI)
- Conformed to Virtual Component Interface Standard (VCI)
- Programmable number of endpoints
- Flexible endpoint configuration
- Support for bulk, interrupt and isochronous transfers
- Supports high-bandwidth mode
- Optionally maximum Packet Size for bulk, interrupt and isochronous endpoints
- Hardware enumeration manager
- Fully-synchronous design
- Interfaces to any application bus.

Block Diagram

The Protocol Manager is connected to UTMI transceiver. This module handles USB bus reset including high speed detection and suspend and resume signaling. The Protocol Manager handles token address decoding for USB packets. Each Endpoint Manager is informed by the Protocol Manager if a token addressed to the endpoint has been received. The Endpoint Managers handle communication between Protocol Manager and endpoint memory (Bulk/Interrupt Memory or Isochronous FIFO). Endpoint Managers handle ping protocol. The USB core contains additional module named Enumeration Manager. This unit handles the enumeration process when the device is first plugged in. The USB core handles all device requests over control endpoint zero, so the developer can immediately start writing code to transfer data over USB using these preconfigured endpoints. The Enumeration Manager can be turned off and the enumeration process may be handled by a microcontroller. Optionally, the Enumeration Manager may download firmware from USB host. The Bus Interface block provides interface between USB control registers and system bus. This interface is compatible with Virtual Component Interface Standard.

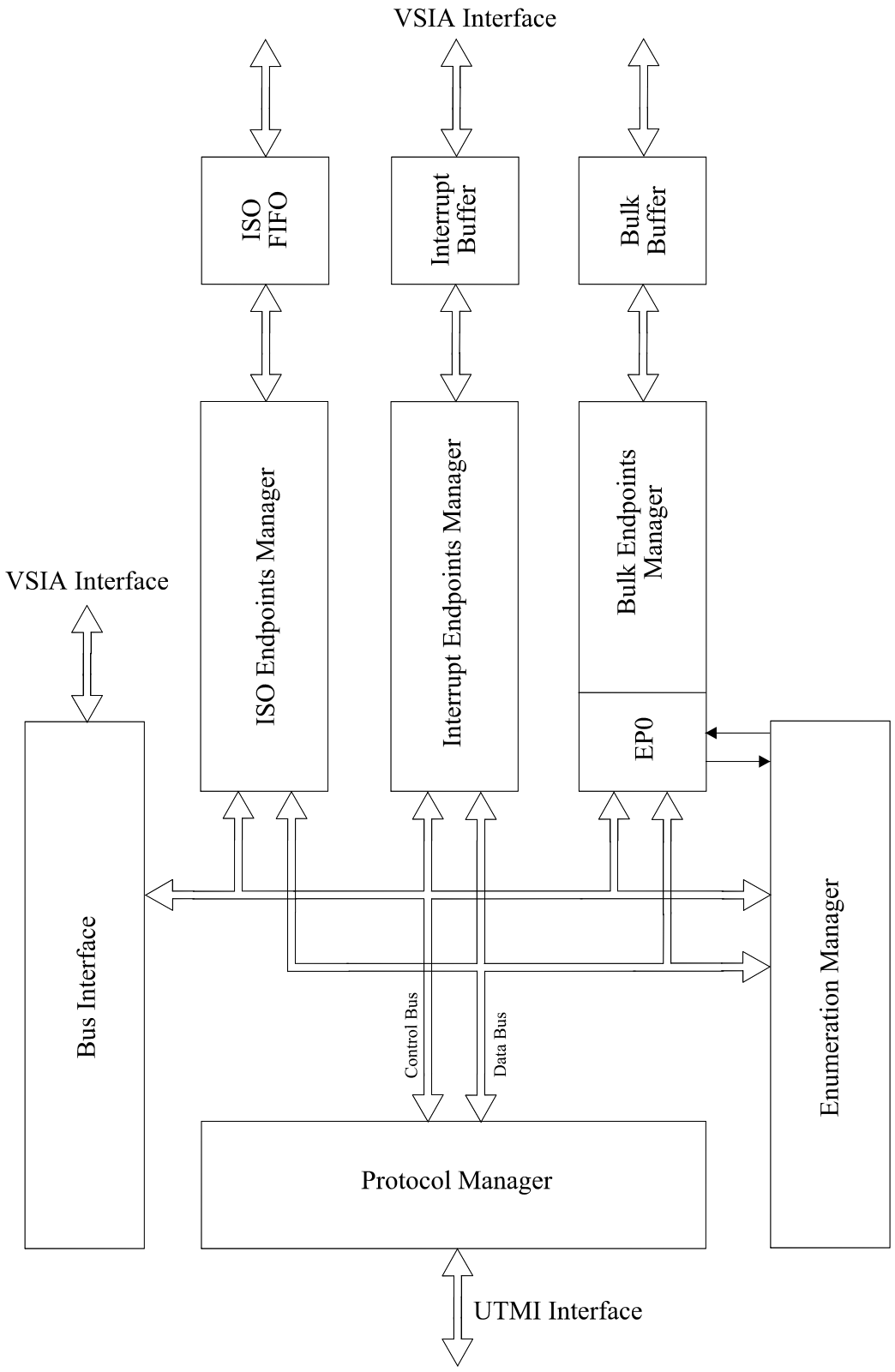

Figure 0.1 Simplified Block Diagram

Signal description

All ports of ALUSB 2.0 core are described in following table.

Pin name	direction	description	Remarks
CLK	in	System clock	Entire core is synchronized to rising edge of CLK
RESET	in	Asynchronous system reset	Active high
CLKEN	in	Global Clock Enable	Active high
DataIN(15:0)	in	Input data from UTMI	See section 4.2
RxActive	in	Receive Active – indicates that SYNC has been detected and UTM starts receiving data	See section 4.2
RxError	In	Error during receiving data	
RxValid	in	Receive Valid – indicates that data on DataIn bus are valid	See section 4.2
RxValidH	in	Receive Valid – indicates that data on DataIn(15:8) bus are valid	See section 4.2
TxReady	In	Transmit Data Ready – indicates that UTM will load data from DataOut bus on the rising edge of CLK	See section 4.2
LineState(1:0)	In	Line State – reflects the current state of the single ended receivers.	See section 4.2
DataInISO(63:0)	In	Input Data to Isochronous endpoint	See section 11
Ce_up(3:0)	In	Read Strob for Isochronous endpoints OUT	See section 12
TxValidHIso(3:0)	In	Strobe for writing higher byte of Isochronous IN endpoints	See section 11
TxValidIso(3:0)	In	Write strobe for Isochronous IN endpoints	See section 11
DataInBulk(191:0)	In	Input data for Bulk endpints	See section 6
TxActive(11:0)	In	Data transfer to endpoint in progress	See section 6
TxValidHBulk(11:0)	In	Strobes for writing higher byte of input data	See section 6
TxValidBulk(11:0)		Strobes for writing lower byte of input data	See section 6
RxReadyBulk(11:0)	In	External device ready for reading data from.	See section 6
Wdata(7:0)	In	VSI input data	See section 3.2
Addr(6:0)	In	VSI address	See section 3.2
VAL	In	Valid strobe (VSI standard)	See section 3.2
WR	In	Write strobe (VSI standard)	See section 3.2
DataExt(15:0)	In	Data from external descriptor's memory	See section 14.33
DataOut(15:0)	Out	Output data for UTMI	See section 4.2
TxValid	Out	Strobe for transmitted data	See section 4.2
TxValidH	Out	Strobe for higher byte of transmitted data	See section 4.2
OpMode(1:0)	Out	Operational Mode – select operation mode for UTMI transceiver	See section 4.2

TermSelect	Out	Termination Select – selects HS/FS termination.	See section 4.2
XcvrSelect	Out	Transceiver Select – selects HS/FS transceiver.	See section 4.2
SuspendM	Out	Suspend UTMI transceiver – macrocell circuitry drawing suspend current	See section 4.2
Suspend	Out	USB device is suspened	See section 4.2
Resume	Out	USB device resumes its operation	See section 4.2
DataOutIso(63:0)	Out	Output Data from Isochronous endpoints	See section 12
RxValidIso(3:0)	Out	Strobe for data read from Isochronous OUT endpoints	See section 12
RxValidHIso(3:0)	Out	Strobe for higher byte of data read from Isochronous OUT endpoints	See section 12
BusyIso(3:0)	Out	Isochronous In endpoints busy indicator	See section 12
Ftogg(7:0)	Out	Fifo toggle indicator of Isochronous endpoints	See section 12 and See section 11
PidErr(3:0)	Out	PID Error indicator of Isochronous Out endpoints	See section 8
CrcErr	Out	CRC16 Error indicator of Isochronous Out endpoints	See section 8
DataOutBulk(191:0)	Out	Output Data from Bulk out and Interrupt endpoints	See section 8
RxValidBulk(11:0)	Out	Strobe for data read from Bulk and interrupt OUT endpoints	See section 8
RxValidHBulk(11:0)	Out	Strobe for higher byte of data read from Bulk and interrupt OUT endpoints	See section 8 and See section 9
EmptyOut(11:0)	Out	Indicates that OUT endpoint is empty	See section 8 and 9
EmptyIN(11:0)	Out	Indicates that IN endpoint is empty	See section 10 and 11
BusyIn	out	Indicates given IN endpoint is busy	See section 10 and 11
BusyOut	Out	Indicates given OUT endpoint is busy	See section 8 and 9
RData(7:0)	Out	VSI output data	See section 3.2
AddExt(4:0)	Out	Address to external descriptor's memory	See section 14.33
INT	Out	Interrupt request signal	See section 3.2
ACK	Out	Acknowledge signal (VSI standard)	See section 3.2
nUSBRes	Out	Reset for Kawasaki Transciever.	See section 4.2
REPEATNEEDED(3:0)	Out	Debug Signal for ISO OUT endpoints.	See section 12.3
REPEAT(3:0)	In	Debug Signal for ISO OUT endpoints.	See section 12.3

VSIA interface

The user can communicate with the ALUSB 2.0 via set of 8-bit registers assembled in table 3.1. The access to these registers is realized by simple 8-bit interface that is compatible to Virtual Component Interface (VCI) standard. The interface assures flexible connection to any other user IP, especially microprocessors and microcontrollers.

For clearance purpose, in this document, any device connected to the interface is called microprocessor.

The core utilizes 7 address lines. All registers occupy addresses from 00h to 67h.

Some registers are read-only, some write-only and some are read-write. Writing data to read-only or not implemented register does not affect specified register. Reading from write-only or not implemented register causes the read data is 00h.

Table 0.1 ALUSB 2.0 registers

address	Name	Description	Dir
00h	OUT0CRTL	Sets toggle and resets Bulk0 OUT endpoint	W
01h	OUT1CRTL	Sets toggle and resets Bulk1 OUT endpoint	W
02h	OUT2CRTL	Sets toggle and resets Bulk2 OUT endpoint	W
03h	OUT3CRTL	Sets toggle and resets Bulk3 OUT endpoint	W
04h	OUT4CRTL	Sets toggle and resets Bulk4 OUT endpoint	W
05h	OUT5CRTL	Sets toggle and resets Bulk5 OUT endpoint	W
06h	OUT6CRTL	Sets toggle and resets Bulk6 OUT endpoint	W
07h	OUT7CRTL	Sets toggle and resets Bulk7 OUT endpoint	W
08h	OUT8CRTL	Sets toggle and resets Interrupt0 OUT endpoint	W
09h	OUT9CRTL	Sets toggle and resets Interrupt1 OUT endpoint	W
0Ah	OUT10CRTL	Sets toggle and resets Interrupt2 OUT endpoint	W
0Bh	OUT11CRTL	Sets toggle and resets Interrupt3 OUT endpoint	W
0Ch		N/A	
0Dh		N/A	
0Eh		N/A	
0Fh		N/A	
10h	IN0CRTL	Sets toggle and resets Bulk0 IN endpoint	W
11h	IN1CRTL	Sets toggle and resets Bulk1 IN endpoint	W
12h	IN2CRTL	Sets toggle and resets Bulk2 IN endpoint	W
13h	IN3CRTL	Sets toggle and resets Bulk3 IN endpoint	W
14h	IN4CRTL	Sets toggle and resets Bulk4 IN endpoint	W
15h	IN5CRTL	Sets toggle and resets Bulk5 IN endpoint	W
16h	IN6CRTL	Sets toggle and resets Bulk6 IN endpoint	W

17h	IN7CRTL	Sets toggle and resets Bulk7 IN endpoint	W
18h	IN8CRTL	Sets toggle and resets Interrupt0 IN endpoint	W
19h	IN9CRTL	Sets toggle and resets Interrupt1 IN endpoint	W
1Ah	IN10CRTL	Sets toggle and resets Interrupt2 IN endpoint	W
1Bh	IN11CRTL	Sets toggle and resets Interrupt3 IN endpoint	W
1Ch			
1Dh			
1Eh			
1Fh			
20h	Feature control 0	Resets and/or clears all Bulk and Interrupt endpoints IN and OUT	W
21h	Feature control 1	Resets all Bulk endpoints IN and OUT and/or set all Bulk and Interrupt endpoints IN and OUT	W
22h	Feature control 2	Resets all Bulk OUT and/or clears all Bulk and Interrupt endpoints IN	W
23h	Feature control 3	Resets all Bulk IN and/or sets all Bulk and Interrupt endpoints IN	W
24h	Feature control 4	Resets all Interrupt endpoints IN and OUT and/or clear all Bulk and Interrupt endpoints OUT	W
25h	Feature control 5	Resets all Interrupt endpoints OUT and/or set all Bulk and Interrupt endpoints OUT	W
26h	Feature control 6	Resets all Interrupt endpoints IN and/or clears all Bulk endpoints IN and OUT	W
27h	Feature control 7	Resets all Bulk and Interrupt endpoints OUT and/or sets all Bulk endpoints IN and OUT	W
28h	Feature control 8	Resets all Bulk and Interrupt endpoints IN and/or clears all Interrupt endpoints IN and OUT	W
29h	Feature control 9	Sets all Interrupt endpoints IN and OUT	W
30h–3Fh	N/A		
40h	Status_USB	Status Register	R/W
41h	DEVICE_addr	Device address	W
42h	Test	Test Settings	W
43h	Suspend	Suspend Settings	W
44h	Valid_in_L	Valid IN low byte	W/R
45H	Valid_in_H	Valid IN high byte	W/R
46h	Valid_out_L	Valid_out low byte	W/R
47h	Valid_out_H	Valid out high byte	W/R
48h	Stall_in_L	Stall in low byte	W/R
49h	Stall_in_H	Stall in High byte	W/R

4Ah	Stall_out_L	Stall out Low byte	W/R
4Bh	Stall_out_H	Stall out H byte	W/R
4Ch	Int_Enable	Interrupt Enable Register	W/R
4Dh		N/A	
4Eh		N/A	
4Fh		N/A	
50h	Max_Pack_size_L_C	Max packet size low byte ep 12	W
51h	Max_Pack_size_H_C	Max packet size high byte ep 12	W
52h	Max_Pack_size_L_D	Max packet size low byte ep 13	W
53h	Max_Pack_size_H_D	Max packet size high byte ep 13	W
54h	Max_Pack_size_L_E	Max packet size low byte ep 14	W
55h	Max_Pack_size_H_E	Max packet size high byte ep 14	W
56h	Max_Pack_size_L_F	Max packet size low byte ep 15	W
57h	Max_Pack_size_H_F	Max packet size high byte ep 15	W
58h	New_Frm_L	New Frame low byte	R
59h	New_Frm_H	New Frame high byte	R
5Ah	Counter_wr_low	Counter_wr low byte. Defines number of bytes send during the control transfer.	W
5Bh	Counter_wr_high	Counter_wr high byte. Defines number of bytes send during the control transfer.	W
5Ch	Counter_rd_low	Counter_rd_low byte. Defines number of bytes received during the control transfer.	R
5Dh	Counter_rd_high	Counter rd high byte. Defines number of bytes received during the control transfer.	R
5Eh		N/A	
5Fh		N/A	
60h	setup 0	Setup byte 0	R
61h	setup 1	Setup byte 1	R
62h	setup 2	Setup byte 2	R
63h	setup 3	Setup byte 3	R
64h	setup 4	Setup byte 4	R
65h	setup 5	Setup byte 5	R
66h	setup 6	Setup byte 6	R
67h	setup 7	Setup byte 7	R

The ALUSB 2.0 core is treated as *target*, however, the microprocessor is treated as *initiator* (see chapter 2.2 of VCI standard). According to VCI standard, the ALUSB 2.0 interface consists of following ports:

- VAL,
- ACK,
- RD,
- ADDRESS(6 downto 0),
- WDATA(7 downto 0).

Since the interface supports only 8-bit data width and supports neither burst transactions nor error signaling, these five ports suffice to perform any read and write operation. The figure 3.2 shows signals used by VCI interface in the ALUSB 2.0.

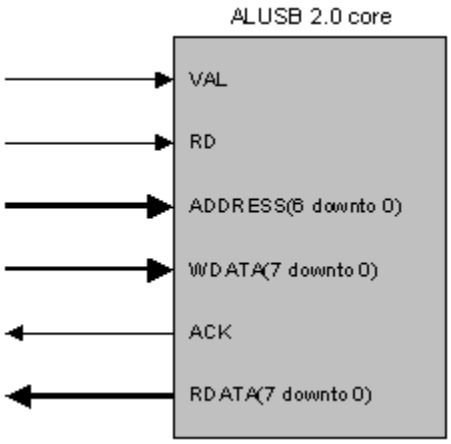

Figure 0.2 VSIA interface.

Write operation

During write operation RD signal is set to 0, microprocessor sets address of register to be written to. Simultaneously, it asserts the VAL signal and updates data on WDATA bus. The address, data and the VAL signal must be maintained until ACK has become asserted and there is a rising edge of CLK. On this edge data are written to ALUSB 2.0 interface.

During write, the ACK is asynchronously generated of VAL. Figure 3.3 shows write operation to the one of registers. Shaded ADDRES and DATA stand for do not care.

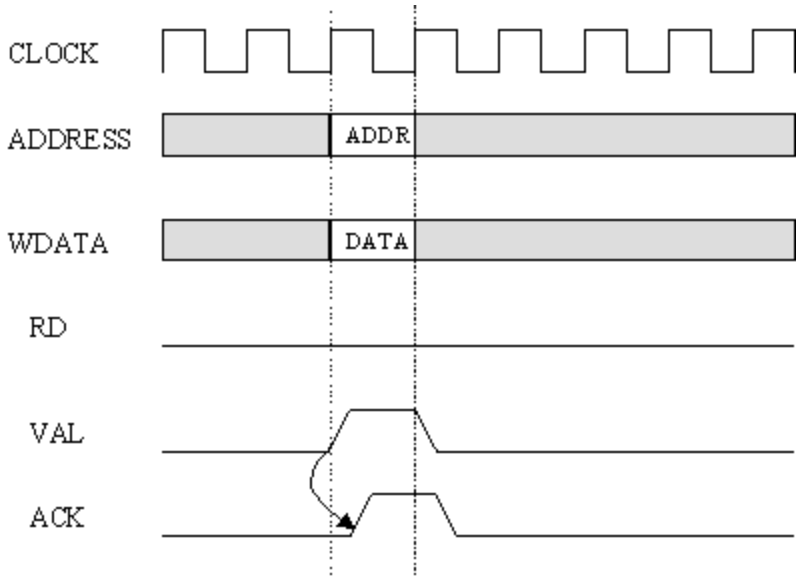

Figure 0.3 Write operation to the ALUSB 2.0

The VAL asserted indicates that microprocessor sets address and sets valid data on the bus. ACK asserted means that ALUSB 2.0 core can complete write operation.

Note:

ACK is not synchronized to CLK. In write operation ACK depends on VAL only.

If microprocessor meets ACK asserted on rising edge, it should deactivate VAL signal, otherwise the write operation will be performed again.

Read operation

During read operation RD is set to 1, microprocessor sets address of register to be read from. Simultaneously, it asserts the VAL signal. The address, data and the VAL signal must be maintained until ACK has become asserted and there is a rising edge of CLK. During read, the ACK is synchronously generated when VAL is 1 and rising edge on CLOCK occurs. For all registers except for 5Ch and 5Dh, ACK is asserted on next rising edge of CLOCK after VALID is set to 1. For 5Ch and 5Dh registers ACK is asserted two clock cycle after VALID is set to 1. Figure 3.4 shows read operation from Status Register. Shaded ADDRES and DATA stand for do not care.

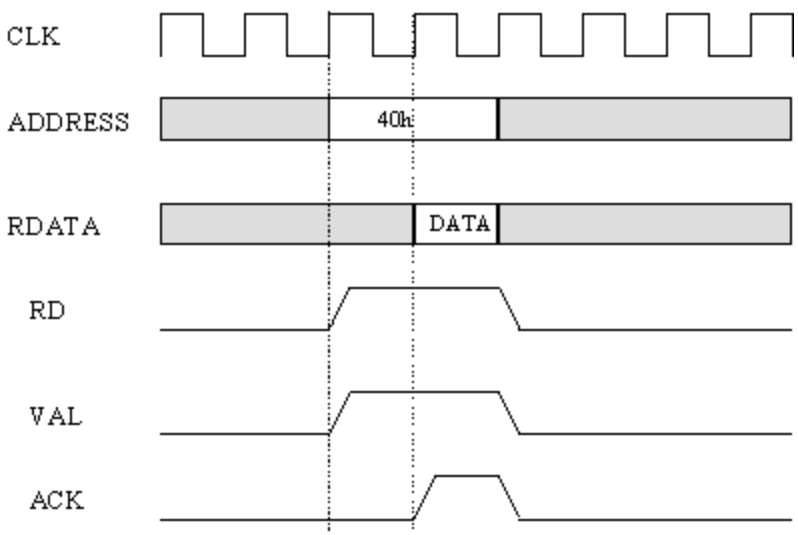

Figure 0.4 Read operation from Status Register

Figure 3.5 shows read operation from New_Frm_L register.

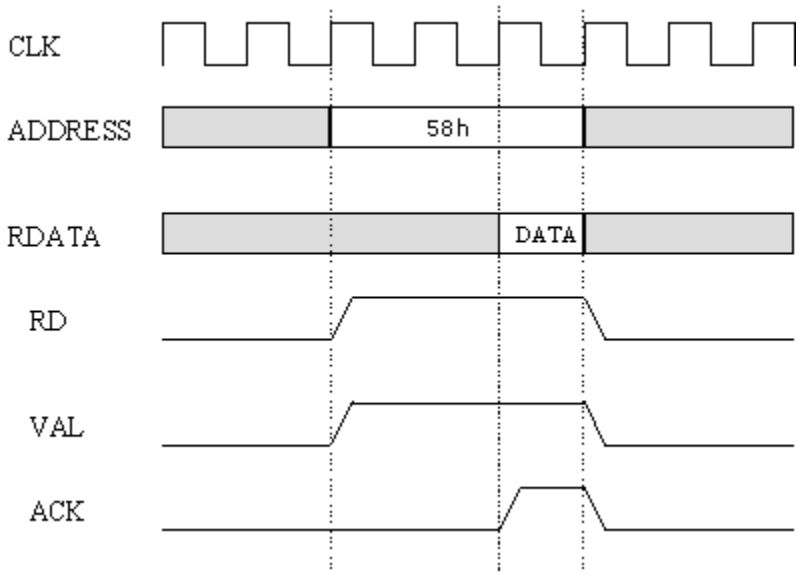

Figure 0.5 Read operation from New_Frm_L

Reading from Status Register

Status Register is special register that indicates current state of ALUSB 2.0. Status Register is associated with five sources of interrupt generated by the ALUSB 2.0 core. (For more details regarding interrupts see section 11). When any of these interrupts occurs, the INT output is asserted. INT is cleared when all

flags indicating interrupts sources are cleared. In order to clear specified flag in the Status Register, the one should be set to it by writing Status Register with proper mask (See section 11).

The Status register should be read by microprocessor after occurrence of rising edge on INT output.

If the read operation is being performed on the Status Register, and some interrupt occurs at the same time, then read operation is postponed. It means that ACK is asserted two or more CLK cycles from VAL set to one. Figure 3.6 shows the case where start of read operation and SOF interrupt occur simultaneously. SOF is internal signal that indicates occurrence of new frame (microframe) (for more detail see section 4).

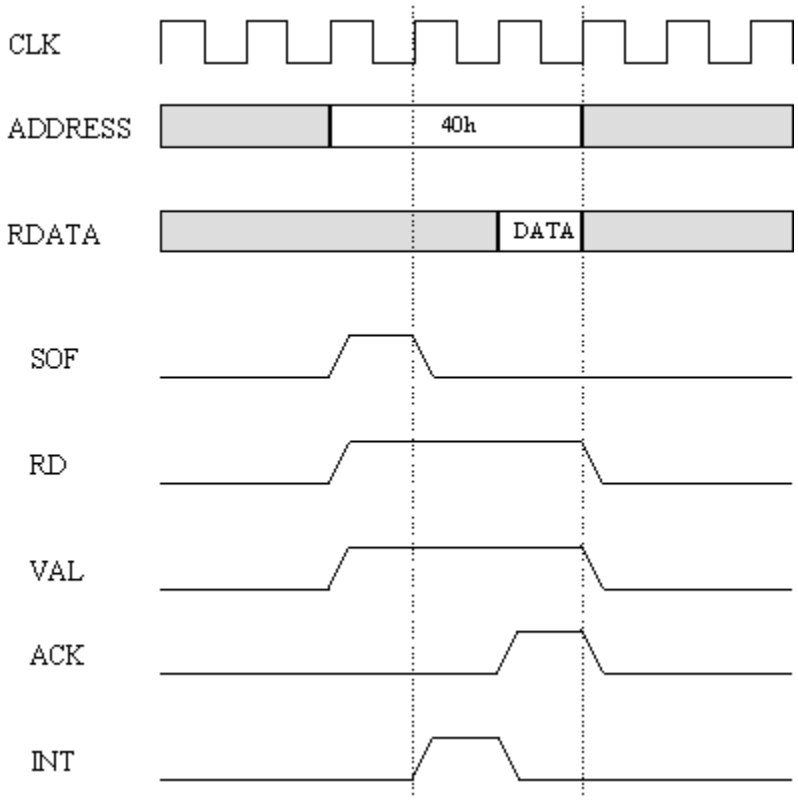

Figure 0.6 Read operation from Status Register when SOF interrupt occurs

The INT output is asserted one CLOCK cycle after SOF is set to one. ACK is asserted one cycle later. It causes that microprocessor reads Status Register that is updated and information of SOF interrupt is indicated during current read. If no interrupt occurs at the beginning of read operation, the read is performed as shown on figure 3.5.

If any interrupt occurs at the beginning of read operation of the others registers, the read is not postponed.

Protocol Management Block

Introduction

Because the USB 2.0 devices operate on very high frequency – 480MHz, data recovery cannot be handled using HDL technology. For full-speed devices ASIC vendors provide only simple level translator to meet the USB signaling requirements. This technology is not sufficient for USB 2.0 clock rate. The USB 2.0 transceiver must handle low level USB protocol to provide the data rate that is compatible with today's FPGA devices. Intel™ released specification for USB 2.0 transceiver. The UTMI transceiver handles data serialization and deserialization, bit-stuffing and clock recovery and synchronization. The USB data are provided in parallel form and the data rate does not exceed 60MHz. The Aldec USB 2.0 core is designed to interface with the UTMI transceiver. It is capable of transmitting data at both full-speed and high-speed data rates. The pinout conforms with the UTMI specification.

Interface to UTMI

The UTMI specification provides two interfaces. The 8-bit interface supports 60MHz performance and is intended for ASIC implementation. The UTMI specification provides also 16-bit interface, which is intended for FPGA based designs. The Aldec USB core is compatible with the 16-bit interface. However, the 8-bit UTMI can be also very easy connected. A simple logic must be added between the USB and UTMI interface. The USB core provides global clock enable signal, which can be used in this case to adjust clock frequency of the USB core. The Protocol Management block contains the following signals, which should be connected to the UTMI transceiver:

Table 0.2 UTMI interface signlas

Name	Direction	Active Level	Description
CLK	Input	Rising-Edge	Global Clock Signal
CLKEN	Input	High	Global Clock Enable Signal.
Reset	Input	High	Global asynchronous reset.
XcvrSelect	Output	N/A	Transceiver Select – selects HS/FS transceiver.
TermSelect	Output	N/A	Termination Select – selects HS/FS termination.
SuspendM	Output	Low	Suspend UTMI transceiver – macrocell circuitry drawing suspend current.
LineState (0-1)	Output	N/A	Line State – reflects the current state of the single ended receivers.
OpMode (0-1)	Output	N/A	Operational Mode – select operation mode for UTMI transceiver.
DataOut (0-15)	Output	N/A	DataOut – 16-bit parallel data bus.
TXValid	Output	High	Transmit Valid – indicates that the DataOut bus is valid.
TXValidH	Output	High	Transmit Valid High – indicates that DataOut(8-15) bus is valid.
TXReady	Input	High	Transmit Data Ready – indicates that UTM will load data from DataOut bus on the rising edge of CLK.
DataIn (0-15)	Input	N/A	DataIn – 16-bit parallel data bus.
RXValid	Input	High	Receive Data Valid – indicates that DataIn bus has valid data.
RXValidH	Input	High	Receive Data Valid H – indicates that DataIn(8-15) bus has valid data.
RXActive	Input	High	Receive Active – indicates that SYNC has been detected and UTM starts receiving data.
nUSBRes	Input	Low	Reset for Kawasaki transceiver.

The UTMI interface is defined in the "USB 2.0 Transceiver Macrocell Interface Specification" provided by Intel™. The specification contains details about functionality and timing of the interface. The UTMI transceiver handles the following functions:

- Data and clock recovery from serial stream on the USB.
- SYNC/EOP generation and checking.
- Bit-stuffing, unstuffing and bit-stuff error detection.
- Suspend/Resume signaling.
- USB 2.0 Test Mode.

The data received from USB cable are deserialized. The UTMI block removes suffed bits and SYNC and EOP fields from the USB packet. The PID and CRC fields remain unchanged. Then, the data are outputted using parallel interface. The UTM interface handles additionally Suspend, Reset and Resume signaling. The UTMI logic allows us to detect or send Chirp K, J states on the USB bus. The High-Speed or Full-Speed operation mode can be changed any time to handle Suspend or Resume signaling.

The USB core contains nUSBRes additional port. It is reset signal for Kawasaki KL5KUSB200 transceiver. The KL5KUSB200 chip is not fully compliant with UTMI specification. It requires asynchronous resetting while the USB Host performs bus reset signaling. The nUSBRes port should be connected directly with RSTN input of the KL5KUSB200 transceiver. Please, refer to KL5KUSB200 USB 2.0 compliant Transceiver Chip DataSheet Rev 0.21 for additional information. The nUSBRes signal is active by 330ns at the beginning of each USB bus reset sequence. It resets asynchronously the KL5KUSB200 transceiver and remains inactive until the next bus reset sequence. This port should be left unconnected while other UTMI transceivers are used.

USB Packets Decoding

The Protocol Management Block receives data form UTMI transceiver. First of all this block must find out what packet type is being sent by the USB bus. The packet identifier (PID) is decoded to determine the type of packet, the format of the packet and the tape of error detection applied to the packet. If the packet has a CRC filed the protocol management unit performs CRC checking. The packet is considered invalid if CRC checking fails. The packets are divided into four groups:

- Token packets.
- Data packets.
- Handshake packets.
- Special packets.

Each token packet excluding SOF contains address filed and endpoint filed.

If such token packet is received the protocol management unit checks the address field to determine if the token is addressed to its. If address checking fails the token is ignored. Otherwise, the endpoint field is decoded and corresponding endpoint manager is informed that following transaction is addressed to its.

The SOF packet is addressed to all possible USB devices connected to the bus. The SOF token consists of a PID and 11-bit frame number. The SOF token must be delivered once to all USB devices every 1ms for full-speed bus or every 125us for high-speed bus. If SOF token was corrupted due to bus errors, the USB device must synthesize existence of the damaged SOF. The protocol management unit performs all these operations. The ISO endpoints managers are informed if valid SOF PID was received. The frame number register is updated if SOF token was received and CRC sum is correct. The protocol management unit generates also the damaged SOFs. The contents of the frame number registers is shown the tabes below

Table 0.3 New Frame Low byte register

New_Frm_L		New Frame Low Byte				X58	
b7	b6	b5	b4	b3	b2	b1	b0
Frm4	Frm3	Frm2	Frm1	Frm0	Mfr 2	Mfr 1	Mfr0
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Table 0.4 New Frame High Byte

New_Frm_H		New Frame High Byte				X59	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	Frm10	Frm9	Frm8	Frm7	Frm6	Frm5
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

The three last significant bits contain the microframe number. These bits are reset to zero and remain unchanged if device works in full-speed mode.

A data packet consists of a PID, a data filed containing zero or more data bytes and CRC. The protocol management unit checks and removes the PID and CRC fields and moves the data to the endpoints managers. Each endpoint manager is informed what type of PID was received in current data packet. If an endpoint manager sends data the protocol management unit moves the data to UTMI transceiver and calculates CRC sum. The CRC sum is added at the end of the packet.

All special packets are ignored by USB function device excluding PING token. The PING token is handled similarly like other token packets. The protocol management block sends information to corresponding bulk manager that PING token was received.

Handshake packets consist of only a PID. The protocol management unit sends information to all endpoints managers if this packet was received.

Reset Signaling

If the USB function device works in HS mode and detects bus inactivity for more than 3ms then operation mode must be switched to full-speed. Then the protocol management block checks the LineState signals for SE0 condition. If SE0 is asserted the protocol management block forces reset sate to the device. If the USB function device operates in full-speed mode and sees an SE0 on its port for more than 2.5us treats this signal as a reset. If the reset was detected then device performs HS detection handshake protocol. The device sends Chirp-K on the bus and waits for an alternating sequence of Chirp-K and Chirp-J, which should be sent by a hub. If the device detects the chirp sequence then enters high-speed mode, otherwise remains in full-speed mode. The protocol management block sets HsFs flag in the Status Register if high-speed mode is detected. The status register contains USBReset flag. This flag is set to one if device is in the reset state. The USB function device may generate an interrupt if the USBReset flag is set. The interrupt must be enabled setting corresponding bit in the Interrupt Enable Register. Please, refer to the “Registers” section of this document.

Table 0.5 Status Register

Status_USB		Status Register					x40
b7	b6	b5	b4	b3	b2	b1	b0
EnumEnable	HsFs	Resume	SetDataIntr	SOF	TkSetup	SuspDetect	UsbReset
W	R	R	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Suspend and Resume signaling

If HS device detects SE0 asserted on the bus for more than 3ms then the operation mode is switched to full-speed. After that, the protocol management block checks the LineState signals for an 'J' state. If the 'J' state is asserted by time longer than 100us the device enters suspend state. The FS device goes into the suspend state after it sees a constant 'J' state for more than 3ms. After detecting the suspend signaling the USB core alerts external microcontroller by setting SuspDetect flag in the Status Register. This may cause generation of an interrupt if corresponding enable flag is set to one in the Interrupt Enable register. The microcontroller responds to the interrupt by performing any necessary operation as shutting off external power consuming devices. Next, the microcontroller sets the EnterSusp flag in the Suspend register. After that, the USB Core drives the Suspend pin to one. The high state on the Suspend pin should turn off the external clock oscillator. These actions put the USB core into low power mode, as required by the USB specification.

Table 0.6 Suspend Register

Suspend		Suspend Register					x43
B7	b6	B5	b4	b3	b2	b1	b0
D7	D6	D5	D4	D3	EnterSusp	ResumeEn	DriveRes
R	R	R	R	R	R/W	R/W	R/W
0	0	0	0	0	0	0	0

If a device is in the suspend state, its operations are resumed when any non-idle signaling is received on its upstream facing port. The device can also generate resume signaling to resume system operation if its remote wakeup capability, has been enabled by the USB host. Resume signaling always take place in FS mode, so the behavior for HS device is identical to a FS device. The protocol management block uses the LineState signals to determine when the transition from the 'J' to the 'K' state occurs.

If a non-idle state occurred on the bus, the USB core clears the Suspend pin and sets Resume flag in the Status Register. The external clock oscillator should restart its operation. The Resume flag can also generate an interrupt for external microcontroller. After these actions the USB device continues operation in the same speed mode, which was before suspending.

An event internal to the device may initiate the resume process (Remote Wakeup). The USB device must report that it is capable of signaling remote wakeup in the configuration descriptor. The remote wakeup is possible if the USB host enabled the remote wakeup capability by sending Set Feature/Device

request. The ResumeEn bit in the Suspend register should be set to one if the remote wakeup is enabled. The microcontroller or the Enumeration Manager can set this bit in response to the Set Feature/Device request. Assuming that ResumeEn bit is set, the microcontroller can set DriveRes bit in the Suspend register to drive remote wakeup. This signal must be asserted for at last 1ms. After this time the USB host performs all necessary operation to wake up system and start transmission on the bus.

USB 2.0 Test Mode Generation

Test mode of a port is entered using a device specific standard request. The microcontroller or the Enumeration Manager must enter the test mode in response to the SetFeature(Test Mode) request. The USB Core supports the following test modes:

Test Type	Test Number	Description
Test_J	01h	The USB core sends continuously the high-speed J state on the bus.
Test_K	02h	The USB core sends continuously the high-speed K state on the bus.
Test_SE0_NAK	03h	The USB core responds to any IN token packet with a NAK handshake.
Test_Packet	04h	The USB core sends the test packet.

The USB core contains the Test Mode register. The register consists of the test number bits and the test enable flag. The microcontroller sets the appropriate test mode on the four last significant bits of the registers. The list of available test modes is shown in the table above. If the test number was set properly, the microcontroller enables the test by setting the TestEN flag. The protocol management block performs all operation required by selected test. The test can be disabled by resetting the USB device.

Table 0.7 Test Mode register

TestMode		Test Mode				X42	
b7	b6	b5	b4	b3	b2	b1	b0
D7	D6	D5	TestEN	TestNR3	TestNr2	TestNr1	TestNr0
R	R	R	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

The Enumeration Manager may also select and run these tests if it was enabled to handle the enumeration process. The microcontroller does not have access to the register if the enumeration manager is enabled.

Enumeration Management block.

Control Endpoint Zero

The endpoint zero is a control endpoint and is required by every USB device. The endpoint accepts setup tokens and receives standard device requests. The control endpoint zero is defined by the USB specification as one bidirectional endpoint. The endpoint was implemented as two bulk IN and OUT endpoints in Aldec USB Core. Both of these endpoints are handled as normal bulk endpoints. The division of the control endpoint into two separate endpoints does not affect any control transaction. The control endpoint zero cooperates with the Enumeration Manager, which receives the setup packets. The setup data are never loaded to endpoint's zero buffers. The Enumeration Manager transfers the eight-byte packet to local data buffer for further processing. The setup packet is always received and acknowledged regardless of the control endpoint's valid, busy or stall bits. The setup packets contain an eight-byte data structure that provides information about the device request. The 8-byte buffer holds data that arrives in setup stage of the control transfer. The setup data can be processed by a microcontroller or by the Enumeration Manager. It depends on the EnumEnable flag in the Status Register.

Table 0.8 Status Register

Status_USB		Status Register				x40	
b7	b6	b5	b4	b3	b2	b1	b0
EnumEnable	HsFs	Resume	SetDataIntr	SOF	TkSetup	SuspDetect	UsbReset
R/W	R	R	R/W	R/W	R/W	R/W	R/W
1	0	0	0	0	0	0	0

If the flag is set to one the Enumeration Manager handles all standard USB requests. Otherwise, the microcontroller must interpret the 8-byte packet and respond to the request.

The USB core generates separate interrupt requests for the various control transfer phases, as shown on the figure below. These interrupt are generated to simplify handling the USB request.

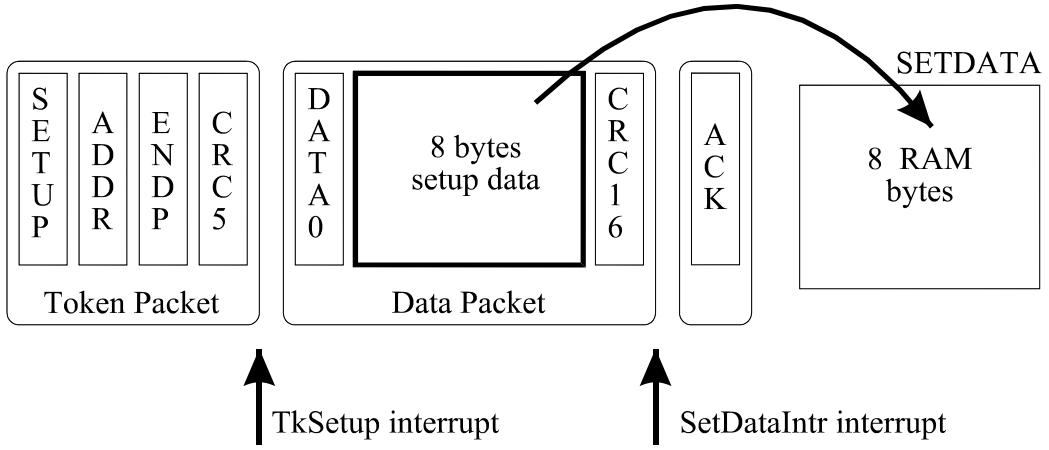

Figure 0.7 Interrupts associated with control transfer

The USB core generates TkSetup (Setup Token Arrived) interrupt when setup token was received. This interrupt is normally used for configuration of control endpoint zero. The microcontroller can set toggle bits and disarm the endpoint to be ready for data stage and status stage of the control transaction. The USB core generates SetDataIntr (Setup Data Arrived) interrupt when the eight bytes of setup data have been received error-free and transferred to local buffer starting at address SETDATA (0x60).

A microcontroller program responds to the SetDataIntr interrupt by inspecting the eight bytes of setup data. When the processing is finished the microcontroller arms the control endpoint to handle data stage or status stage of the control transaction.

The figure below shows the set of USB registers that deal with control transactions over endpoint zero.

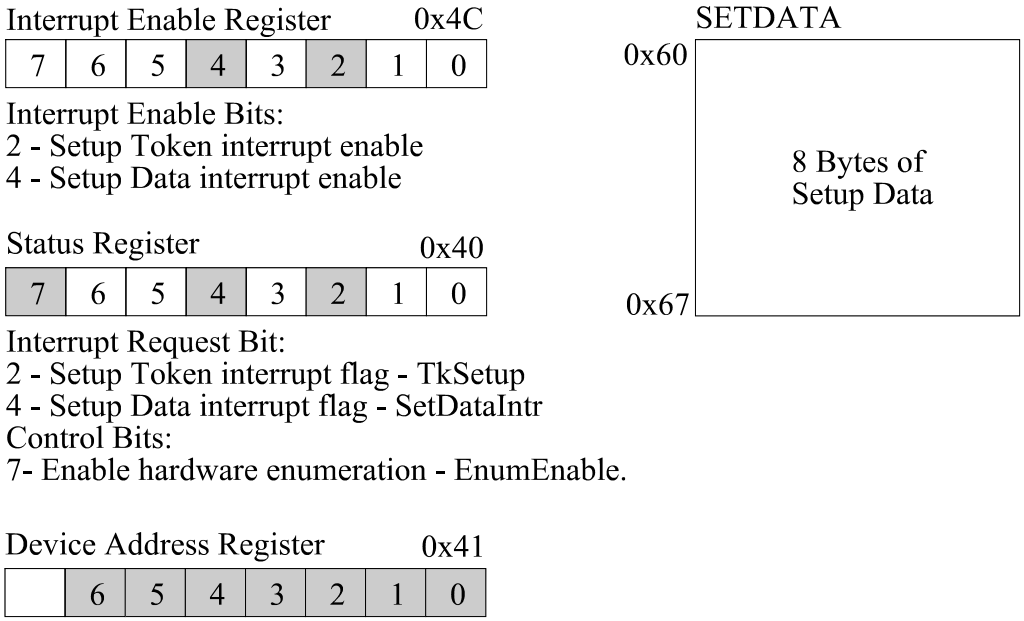

Figure 0.8 Registers for Handling Setup Transactions

Two bits in the Interrupt Enable Register enable the Setup Token and Setup Data interrupts. The USB core transfers eight setup bytes into the RAM buffer at SETDATA address. The Status register contains EnumEnable control bit. The Device Address Register contains USB device address. This register can be changed in response to Set Address request. The microcontroller may need access to the Stall and Valid registers to handle some request regarding bulk endpoints. Please, refer to the "Registers" section of this document for detailed description of these registers.

Enumeration Manager

The USB core contains the Enumeration Manager block to perform enumeration without external microcontroller. The Enumeration Manager contains the USB descriptors. This block sets the proper endpoint configuration bits to match the descriptors. The USB core handles all device requests over control endpoint zero, so the developer can immediately start writing code to transfer data over USB using these preconfigured endpoints. The enumeration process can be handled also by the external microcontroller.

The EnumEnable bit in the Status Register determines which entity the core or the microcontroller handles enumeration process. At power on the EnumEnable bit is one, indicating that the USB core

handles device requests. The Table below shows how the USB core responds to USB request when EnumEnable=1.

Table 0.9 USB Device Requests

Request	Action
Get Descriptor	Supplies table of descriptors
Get Configuration	Return configuration number
Get Interface	Return Alternate Settings number
Get Status/Device	Return two zero bytes
Get Status/Interface	Return two zero bytes
Get Status/Endpoint	Supplies Stall bit for indicated EP
Clear Feature/Device	None
Clear Feature/Endpoint	Clears Stall bit for indicated EP
Set Feature/Device	None
Set Feature/Endpoint	Set Stall bit for indicated endpoint
Set Configuration	Configure device and sets configuration number
Set Interface	Configure device and sets interface number
Set Address	Set device address
Set Descriptor	None
Sync Frame	None

The USB core has built-in descriptor data table. The Enumeration Manager sends the descriptor table in response to Get Descriptor request. The USB device consists of single USB configuration containing one interface with two alternate setting. The endpoints reported for this device are shown in the table below:

Table 0.10 The USB Core Endpoints

Endpoint	Type	Alternate Settings	
		0	1
		Max Packet Size	
0 INOUT	CTRL	64	64
1 IN	Bulk	0	64
1 OUT	Bulk	0	64
2 IN	Bulk	0	64
2 OUT	Bulk	0	64
3 IN	Bulk	0	64
3 OUT	Bulk	0	64
4 IN	Bulk	0	64
4 OUT	Bulk	0	64
5 IN	Bulk	0	64
5 OUT	Bulk	0	64
6 IN	Bulk	0	64
6 OUT	Bulk	0	64
7 IN	Bulk	0	64
7 OUT	Bulk	0	64
8 IN	Interrupt	0	64
8 OUT	Interrupt	0	64
9 IN	Interrupt	0	64
9 OUT	Interrupt	0	64
10 IN	Interrupt	0	64
10 OUT	Interrupt	0	64
11 IN	Interrupt	0	64
11 OUT	Interrupt	0	64
12 IN	ISO	0	64
12 OUT	ISO	0	64
13 IN	ISO	0	64
13 OUT	ISO	0	64

14 IN	ISO	0	64
14 OUT	ISO	0	64
15 IN	ISO	0	64
15 OUT	ISO	0	64

The alternate setting zero uses no interrupt or isochronous bandwidth as recommended by USB specification. If the Enumeration Manager is turned on the microcontroller may also read or write the endpoint's STALL registers. The microcontroller may have to halt an endpoint because of receiving incorrect data. The Enumeration Manager has a higher priority if both microcontroller and the Enumeration Manager write data to the register at the same time. The Enumeration Manager controls also Device Address Register. The microcontroller may write data to the register but the data are ignored as long as the EnumEnable bit is one. If the EnumEnable bit is reset to zero, the USB core passes all USB request onto the microcontroller via the SETDATA buffer. The STALL registers and Device Address register are fully controlled by the microcontroller if EnumEnable is reset to zero.

Note:

The USB configuration can be changed upon user request. Some of endpoints can be removed and the USB descriptors can by modified to meet user requirements.

USB Requests

The USB specification defines a set of Standard Device Requests. The request and the request's parameters are sent to the device in the setup packet. Every setup packet has eight bytes. USB device must respond to standard device request. This section describes how the USB device responds to individual request.

Get Status Request.

The USB specification defines three USB status requests. The requests are as follows:

1. Decvice Staus
    a. Remote Wakeup
    b. Self-Powered
2. Endpoint Staus
    a. Stall bit.
3. Interface Status – reserved.

The following tables show the eight setup bytes for the Get Status request:

Table 0.11 Get Device Status Request

Byte	Field	Value	Description	Response
0	bmRequest	0x80	Request to device - type In	Byte 0:
1	bRequest	0x00	Get Status request	Bit0 – Self Powered
2	wValueL	0x00		Bit1- Remote Wakeup
3	wValueH	0x00		
4	wIndexL	0x00		Byte 1: zero
5	wIndexH	0x00		
6	wLengthL	0x02	Two bytes must be returned	
7	wLengthH	0x00		

Table 0.12 Get Interface Status Request

Byte	Field	Value	Description	Response
0	bmRequest	0x81	Request to interface - type In	Byte0: zero
1	bRequest	0x00	Get Status request	
2	wValueL	0x00		Byte1: zero
3	wValueH	0x00		
4	wIndexL	INTR	Interface number	
5	wIndexH	0x00		
6	wLengthL	0x02	Two bytes must be returned	
7	wLengthH	0x00		

Table 0.13 Get Endpoint Status request

Byte	Field	Value	Description	Response
0	bmRequest	0x82	Request to Endpoint - type In	Byte0:
1	bRequest	0x00	Get Status request	Bit0 – Stall bit
2	wValueL	0x00		
3	wValueH	0x00		Byte1: zero
4	wIndexL	EP	Endpoint number	
5	wIndexH	0x00		
6	wLengthL	0x02	Two bytes must be returned	
7	wLengthH	0x00		

The Get Status Device request queries the state of two bits, Remote Wakup and Self-Powered. The Remote Wakeup bit indicates whether or not the device is enabled to request remote wakeup. The Self-powered bit indicates whether or not the device is self-powered.

The Enumeration Manager returns two zero bytes in response to Get Status Interface request. This request is reserved for future use.

Each bulk or interrupt endpoint has a Stall bit. If this bit is set the endpoint returns a STALL handshake in response to any USB transaction. The Get Status Endpoint request returns the Stall bit for corresponding endpoint.

Get Configuration Request.

Table 0.14 Get Configuration Request

Byte	Field	Value	Description	Response
0	bmRequest	0x80	Request to device - type In	Byte 0:
1	bRequest	0x08	Get Configuration Request	The configuration number
2	wValueL	0x00		
3	wValueH	0x00		
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	0x01	One bytes must be returned	
7	wLengthH	0x00		

The Enumeration Manager returns the current configuration number in response to this request.

Get Interface Request

Table 0.15 Get Interface Request

Byte	Field	Value	Description	Response
0	bmRequest	0x81	Request to Device - type In	Byte 0:
1	bRequest	0x0A	Get Interface Request	The alternate Setting for specified interface
2	wValueL	0x00		
3	wValueH	0x00		
4	wIndexL	INTR	Interface Number	
5	wIndexH	0x00		
6	wLengthL	0x01	One bytes must be returned	
7	wLengthH	0x00		

The Enumeration Manager returns the alternate setting for selected interface.

Get Descriptor Request

The USB specification defines four types of the Get Descriptor request:

- Get Descriptor – Device
- Get Descriptor – Configuration
- Get Descriptor - Device Qualifier
- Get Descriptor - Other Speed Configuration

The following tables show the setup packet bytes for the Get Descriptor requests:

Table 0.16 Get Device Descriptor Request

Byte	Field	Value	Description	Response
0	bmRequest	0x80	Request to Device - type In	The Device descriptor table.
1	bRequest	0x06	Get Descriptor request	
2	wValueL	0x00		
3	wValueH	0x01	Device Descriptor	
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	LenL	Number of returned bytes	
7	wLengthH	LenH		

Table 0.17 Get Configuration Descriptor Request.

Byte	Field	Value	Description	Response
0	bmRequest	0x80	Request to Device - type In	The Configuration descriptor table.
1	bRequest	0x06	Get Descriptor request	
2	wValueL	CFG	Configuration number	
3	wValueH	0x02	Configuration Descriptor	
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	LenL	Number of returned bytes	
7	wLengthH	LenH		

Table 0.18 Get Device Qualifier Descriptor Request

Byte	Field	Value	Description	Response
0	bmRequest	0x80	Request to Device - type In	The Device Qualifier descriptor table.
1	bRequest	0x06	Get Descriptor request	
2	wValueL	0x00		
3	wValueH	0x06	Device Qualifier Descriptor	
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	LenL	Number of returned bytes	
7	wLengthH	LenH		

Table 0.19 Get Configuration Descriptor Request

Byte	Field	Value	Description	Response
0	bmRequest	0x80	Request to Device - type In	The Other speed configuration descriptor table.
1	bRequest	0x06	Get Descriptor request	
2	wValueL	0x00		
3	wValueH	0x07	Other Speed Configuration Descriptor	
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	LenL	Number of returned bytes	
7	wLengthH	LenH		

The requests return the specified descriptor. The wLength filed specifies the number of bytes to return. A request for configuration descriptor returns the configuration descriptor, all interface descriptors and all endpoint descriptors for all of the interfaces in a single request.

Set Address Request

This request sets the device address for all future device accesses.

Table 0.20 Set Device Address request

Byte	Field	Value	Description	Response
0	bmRequest	0x00	Request to Device - type Out	
1	bRequest	0x05	Set Address request	
2	wValueL	ADDR	Device Address	
3	wValueH	0x00		
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	0x00		
7	wLengthH	0x00		

Set Interface Request

Some USB devices have configurations with different interfaces and alternate settings. This request allows the host to select an interface and an alternate setting.

Table 0.21 Set Interface request

Byte	Field	Value	Description	Response
0	bmRequest	0x01	Request to Interface - type Out	
1	bRequest	0x0B	Set Interface request	
2	wValueL	AS	Alternate Setting	
3	wValueH	0x00		
4	wIndexL	INTR	Interface	
5	wIndexH	0x00		
6	wLengthL	0x00		
7	wLengthH	0x00		

Set Configuration Request

Table 0.22 Set Configuration request

Byte	Field	Value	Description	Response
0	bmRequest	0x00	Request to Device - type Out	
1	bRequest	0x09	Set Configuration request	
2	wValueL	CFG	Configuration Number	
3	wValueH	0x00		
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	0x00		
7	wLengthH	0x00		

This request sets the device configuration. The lower byte of the wValue filed specifies the desired configuration. This configuration must be zero or match a configuration value from a configuration descriptor.

Set Feature Request

The Set Feature request is used to enable remote wakeup or stall an endpoint. This request is also used to select and enable the USB 2.0 Test Mode.

Table 0.23 Set Device Feature request

Byte	Field	Value	Description	Response
0	bmRequest	0x00	Request to Device - type Out	
1	bRequest	0x03	Set Feature request	
2	wValueL	FTR	Feature Selector	
3	wValueH	0x00		
4	wIndexL	0x00		
5	wIndexH	TEST	Test Mode Selector	
6	wLengthL	0x00		
7	wLengthH	0x00		

The Set Feature Device request sets the remote wake up bit if the wValueL filed is equal to 01H. If the wValueL filed indicates the test mode feature (02H), the Enumeration Manager enables the test mode selected by wIndexH field.

Table 0.24 Set Endpoint Feature request

Byte	Field	Value	Description	Response
0	bmRequest	0x02	Request to Endpoint - type Out	
1	bRequest	0x03	Set Feature request	
2	wValueL	0x00	Feature Selector – Stall bit	
3	wValueH	0x00		
4	wIndexL	EP	Endpoint Number	
5	wIndexH	0x00		
6	wLengthL	0x00		
7	wLengthH	0x00		

The Set Endpoint Feature request sets the Stall bit for selected endpoint.

Clear Feature Request

Table 0.25 Clear Device Feature request

Byte	Field	Value	Description	Response
0	bmRequest	0x00	Request to Device - type Out	
1	bRequest	0x01	Clear Feature request	
2	wValueL	0x01	Feature Selector – remote wakeup	
3	wValueH	0x00		
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	0x00		
7	wLengthH	0x00		

The Clear Device Feature request is used to clear a stalled endpoint. The Test Mode feature cannot be cleared by this request.

Table 0.26 Clear Endpoint Feature request

Byte	Field	Value	Description	Response
0	bmRequest	0x02	Request to Endpoint- type Out	
1	bRequest	0x03	Clear Feature request	
2	wValueL	0x00	Feature Selector – stall bit	
3	wValueH	0x00		
4	wIndexL	EP	Endpoint Number	
5	wIndexH	0x00		
6	wLengthL	0x00		
7	wLengthH	0x00		

The Clear Endpoint Feature request removes the stall condition from an endpoint.

Descriptors

Device Descriptor

The Device descriptor specifies a MaxPacketSize of 64 bytes for control endpoint 0. The descriptor contains Product and Release Number ID's, which can be specified by a user. The USB core returns this information in response to a Get_Descriptor/Device host request.

Table 0.27 The Device Descriptor

Offset	Field	Description	Value
0	BLength	Length of this descriptor = 18 bytes	12H
1	bDescriptorType	Descriptor Type = Device	01H
2	bcdUSB (L)	USB spec version 1.1 (L)	01H
3	bcdUSB (H)	USB spec version 1.1 (H)	01H
4	bDeviceClass	Device class	00H
5	bDeviceSubClass	Device sub-class	00H
6	bDeviceProtocol	Device Protocol	00H
7	bMaxPacketSize0	Max packet size for EP0 = 64 bytes	40H
8	IdVendor (L)	Vendor id (L)	XX
9	IdVendor (H)	Vendor id (H)	XX
10	idProduct (L)	Product id (L)	XX
11	idProduct (H)	Product id (H)	XX

12	bcdDevice (L)	Device release Number (BCD,L)	XX
13	bcdDevice (H)	Device release Number (BCD,H)	XX
14	iManufacturer	Manufacturer index string = none	00H
15	Iproduct	Product index string = none	00H
16	iSerialNumber	Serial number index string = none	00H
17	bNumConfigurations	Number of configurations in this interface = 1	01H

Device Qualifier Descriptor

The device qualifier descriptor describes information about high-speed device that would change if the device ware operating at the other speed. For example, if the device is currently operation at full-speed, the device qualifier returns information about how it would operate at high-speed and vice-versa. The USB host accesses this descriptor using the Get Descripor/Device_Qualifier. The USB core supports the device qualifier descriptor. The device configuration is the same for both full and high speed as shown in the table below. The descriptor can also be modified upon user request.

Table 0.28 The Device Qualifier Descriptor

Offset	Field	Description	Value
0	Blength	Length of this descriptor = 18 bytes	0AH
1	bDescriptorType	Descriptor Type = Device	06H
2	bcdUSB (L)	USB spec version 2.0 (L)	00H
3	bcdUSB (H)	USB spec version 2.0 (H)	02H
4	bDeviceClass	Device class	00H
5	bDeviceSubClass	Device sub-class	00H
6	bDeviceProtocol	Device Protocol	00H
7	bMaxPacketSize0	Max packet size for EP0 = 64 bytes	40H
8	bNumConfigurations	Number of configurations in this interface = 1	01H
9	BReserved	Reserved for future use, must be zero	00h

Configuration descriptor

The configuration descriptor includes a total length field that encompasses all interface and endpoint descriptors that follow the configuration descriptor. This configuration describes a single interface. The host selects the configuration using Set Configuration Request.

Table 0.29 The Configuration descriptor

Offset	Field	Description	Value
0	BLength	Length of this descriptor = 9 bytes	09H
1	bDescriptorType	Descriptor Type = Configuration	02H
2	wTotalLength (L)	Total length (L) including Interface & Endpoint descriptors	EDH
3	wTotalLength (H)	Total length (H)	00
4	bNumInterfaces	Number of interfaces in this configuration	01H
5	bConfigurationValue	Configuration value used by Set Configuration Request to select this interface	01H
6	IConfiguration	Index of string describing this configuration	00H
7	BmAttributes	Attributes - bus powered, no wakeup	80H
8	MaxPower	Max power	XX

Other Speed Configuration Descriptor

The Other Speed Configuration descriptor describes a configuration of a high-speed capable device if it can operate at its other possible speed. The structure of the other speed configuration is identical to a configuration descriptor.

Table 0.30 The Other Speed Configuration descriptor

Offset	Field	Description	Value
0	BLength	Length of this descriptor = 9 bytes	09H
1	bDescriptorType	Descriptor Type = Configuration	07H
2	wTotalLength (L)	Total length (L) including Interface & Endpoint descriptors	09H
3	wTotalLength (H)	Total length (H)	00
4	bNumInterfaces	Number of interfaces in this configuration	01H
5	bConfigurationValue	Configuration value used by Set Configuration Request to select this interface	01H
6	iConfiguration	Index of string describing this configuration	00H
7	bmAttributes	Attributes - bus powered, no wakeup	80H
8	MaxPower	Max power	XX

Interface descriptors

Interface 0, alternate setting 0 describes endpoint 0 only. This is a “zero bandwidth” setting. The interface has no string index.

Table 0.31 The Interface 0 Alternate Setting 0 Descriptor

Offset	Field	Description	Value
0	Blength	Length of the interface descriptor	09H
1	BdescriptorType	Descriptor Type = Interface	04H
2	bInterfaceNumber	Zero based index of this interface = 0	00H
3	bAlternateSetting	Alternate setting value = 0	00H
4	bNumEndpoints	Number of endpoints in this interface (not counting EP0) = 0	00H
5	bInterfaceClass	Interface class = vendor specific	FFH
6	bInterfaceSubClass	Interface sub-class = vendor specific	FFH
7	bInterfaceProtocol	Interface protocol	00H
8	Iinterface	Index to string descriptor for this interface = none	00H

Interface 0, alternate setting 1 has 30 endpoints, whose individual descriptors follow the interface descriptor. The alternate settings have no string indices.

Table 0.32 The Interface 0 Alternate Setting 1 Descriptor

Offset	Field	Description	Value
0	Blength	Length of the interface descriptor	09H
1	BdescriptorType	Descriptor Type = **Interface**	04H
2	bInterfaceNumber	Zero based index of this interface = **0**	00H
3	bAlternateSetting	Alternate setting value = **1**	01H
4	bNumEndpoints	Number of endpoints in this interface (not counting EP0) = **30**	1EH
5	bInterfaceClass	Interface class = vendor specific	FFH
6	bInterfaceSubClass	Interface sub-class = vendor specific	FFH
7	bInterfaceProtocol	Interface protocol = vendor specific	00H
8	Iinterface	Index to string descriptor for this interface = none	00H

Bulk and Interrupt Endpoints Descriptors

Interface 0, alternate setting 1 has fourteen bulk and four interrupt endpoints, whose individual descriptors follow interface descriptor. The table below shows descriptor for Bulk IN Endpoint number 1. The others Bulk IN Endpoints have exactly the same descriptor, only the BEndpointAddress field is changed. For example, the Bulk In Endpoint number 2 has the BendpointAddress field equal 82H.

Table 0.33 Bulk In Endpoint Descriptor

Offset	Field	Description	Value
0	Blength	Length of this endpoint descriptor	07H
1	BdescriptorType	Descriptor Type = Endpoint	05H
2	BEndpointAddress	Endpoint direction (1 is in) and address = IN1	81H
3	BmAttributes	xfr type = Bulk	02H
4	WMaxPacketSize (L)	Max packet size = 64 bytes	40H
5	WMaxPacketSize (H)	Max packet size – high	00H
6	Binterval	polling interval in milliseconds	00H

The Table below shows descriptor for Bulk OUT Endpoint number 1. The others Bulk OUT Endpoints have exactly the same descriptor only the BEndpointAddress field is changed.

Table 0.34 The Bulk Out Endpoint Descriptor

Offset	Field	Description	Value
0	Blength	Length of this endpoint descriptor	07H
1	BDescriptorType	Descriptor Type = Endpoint	05H
2	BEndpointAddress	endpoint direction (1 is in) and address = OUT1	01H
3	BmAttributes	xfr type = Bulk	02H
4	WMaxPacketSize (L)	max packet size = 64 bytes	40H
5	WMaxPacketSize (H)	max packet size – high	00H
6	Binterval	polling interval in milliseconds	00H

The interrupt endpoints have very similar descriptors. The field bmAttributes indicates the transfer type. This filed must have value 03h for interrupt endpoints. The default device configuratiom contains four interrupt endpoints.

Isochronous Endpoints Descriptors

Table 0.35 The Isochronous In Endpoint Descriptor

Offset	Field	Description	Value
0	Blength	Length of this endpoint descriptor	07H
1	BdescriptorType	Descriptor Type = Endpoint	05H
2	BendpointAddress	endpoint direction (1 is in) and address = IN8	88H
3	BmAttributes	xfr type = ISO	01H
4	wMaxPacketSize (L)	max packet size = 64 bytes	40H
5	wMaxPacketSize (H)	max packet size – high	00H
6	Binterval	polling interval in milliseconds (1 for iso)	01H

Table 0.36 The Isochronous OUT Endpoint Descriptor

Offset	Field	Description	Value
0	Blength	Length of this endpoint descriptor	07H
1	bDescriptorType	Descriptor Type = Endpoint	05H
2	bEndpointAddress	endpoint direction (1 is in) and address = OUT2	08H
3	BmAttributes	xfr type = ISO	01H
4	wMaxPacketSize (L)	max packet size = 64 bytes	40H
5	wMaxPacketSize (H)	max packet size – high	00H
6	Binterval	polling interval in milliseconds (1 for iso)	01H

Interface 0, alternate setting 1 has four isochronous endpoints with max packet size of 64 bytes. The tables above show descriptors for the isochronous IN and OUT endpoints. The other isochronous endpoints have the same descriptor excluding bEndpointAddress field, which contains endpoint address.

Note:

Some of descriptors contain fields, which have the XX value. These values should be specified by the user if hardware enumeration is used.

Bulk IN endpoints.

The Bulk endpoints typically transport large amount of data, such as that used for printers or scanners. The Bulk data is sequential. The reliable exchange of data is ensured at the hardware level by using the error detection in hardware and invoking a limited number of retries. The ALUSB 2.0 core supports up to

8 Bulk IN endpoints including control endpoint zero. All Bulk endpoints can operate in the Full Speed mode as well as in the High Speed mode. The Bulk IN endpoints are independent of each other and are referenced from 0 to 7.

The USB specification defines the maximum bulk data payload to be only 8, 16, 32, or 64 bytes for full-speed devices and 512 bytes for high-speed devices. The bulk endpoints are designed to support the maximum data payload size.

The Interface for all Bulk IN endpoints consists of the following ports:

- DataInBulk (127 downto 0),
- TxActive (7 downto 0),
- TxValidBulk (7 downto 0),
- TxValidHBulk (7 downto 0),
- EmptyIn (7 downto 0),
- BusyIn (7 downto 0).

Every Bulk IN endpoint uses these ports to communicate with an external device, which transfers the data. Figure 6.1 shows the ports of the ALUSB 2.0 core that are the Bulk IN endpoints interface.

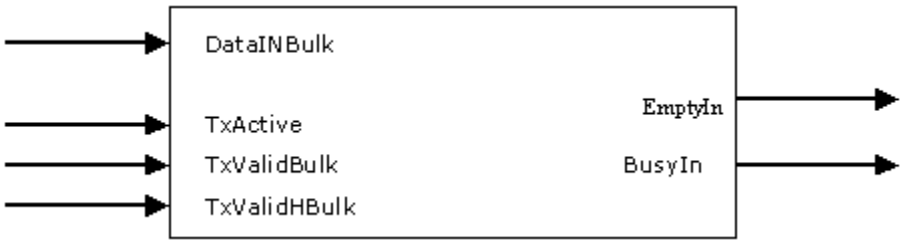

Figure 0.1 Bulk IN interface

Single endpoint utilizes:

- 16 bits of DataInBulk,
- 1 bit of TxActive,
- 1 bit of TxValidBulk,
- 1 bit of TxValidHBulk,
- 1 bit of EmptyIn,
- 1 bit of BusyIn.

The table 6.1 shows how these signals are assigned to the individual endpoints.

Table 0.1 Signal assignment to the individual Bulk In endpoints

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
DataInBulk	(15:0)	(31:16)	(47:32)	(63:48)	(79:64)	(95:80)	(111:96)	(127:112)
TxActive	0	1	2	3	4	5	6	7
TxValidBulk	0	1	2	3	4	5	6	7
TxValidHBulk	0	1	2	3	4	5	6	7
EmptyIn	0	1	2	3	4	5	6	7
BusyIn	0	1	2	3	4	5	6	7

For example endpoint 1 IN utilizes:

- DataInBulk (31 downto 16),
- TxActive (1),
- TxValidBulk (1),
- TxValidHBulk (1),
- EmptyIn (1),
- BusyIn (1).

Endpoint 3 IN utilizes:

- DataInBulk (63 downto 48),
- TxActive (3),
- TxValidBulk (3),
- TxValidHBulk (3),
- EmptyIn (3),
- BusyIn (3) etc.

The Bulk IN endpoints assure a data flow from an external device to the USB Host.

To use the given Bulk IN endpoint, the corresponding bit in the VALID_in_L register (44h) must be set and the corresponding bit in the STALL_in_L register (48h) must be cleared.

Table 0.2 The Valid Register - low byte

Valid_in_L		Valid Register low byte					x44
b7	b6	b5	b4	b3	b2	b1	b0
Bulk 7	Bulk 6	Bulk 5	Bulk 4	Bulk 3	Bulk 2	Bulk 1	Bulk 0
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	1

Table 0.3 The Stall Register - low byte

Stall_in_L		Stall Register low byte					x48
b7	b6	b5	b4	b3	b2	b1	b0
Bulk 7	Bulk 6	Bulk 5	Bulk 4	Bulk 3	Bulk 2	Bulk 1	Bulk 0
W	W	W	W	W	W	W	W
1	1	1	1	1	1	1	0

An external device writes data to double-buffered FIFO. The FIFO buffers are independent. The device has access only to one FIFO buffer while the second FIFO buffer is read by the corresponding endpoint controller.

When the endpoint controller finished reading data from its FIFO buffer, then corresponding bit of the EmptyIn port is set. It means that FIFO buffer on the endpoint side is empty.

When a device writes data to its FIFO buffer, then the corresponding bit of BusyIn port is set. The device clears the TxActive port when writing data to the FIFO buffer is finished. The FIFO buffers are switched when the FIFO buffer on the endpoint side is empty and the device FIFO buffer is full. It happens when the TxActive port is low, the BusyIn port is high and the EmptyIn port is high. Figure 6.2 shows how the FIFO buffers are accessed and switched.

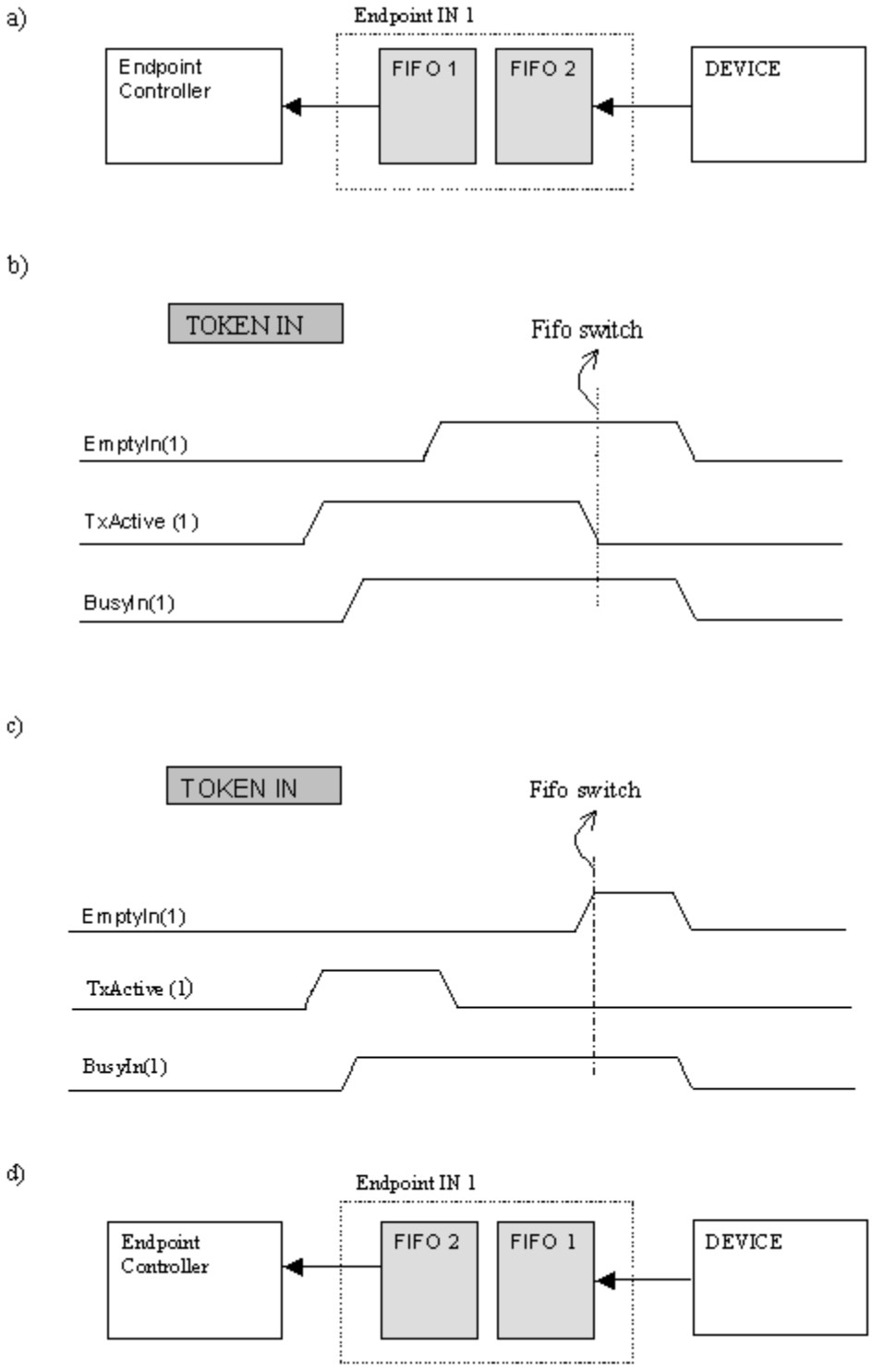

Figure 0.2 FIFO switch mechanism in Bulk IN endpoints

Figure 6.2a shows the FIFO buffers before switching. The FIFO1 buffer is on the endpoint side and the FIFO2 buffer is written by the device. Figure 6.2b shows a moment when buffers are switched. The endpoint controller reads data from FIFO1 buffer in response to Token IN. The EmptyIn port is low indicating that the FIFO1 buffer contains unread data. Meanwhile, the device is writing new data to the FIFO2 buffer. The BusyIn port is then high. When the endpoint controller finished reading data from FIFO1 then EmptyIn(1) port is high. The device clears TxActive signal when writing data is finished. The FIFO buffers are switched while TxActive signal goes low. Figure 6.2c shows FIFO buffers switch when the device first finishes writing data to the FIFO2 buffer. Then, switching occurs when the EmptyIn signal changes its value. Figure 6.2d shows how the FIFO buffers are connected after they have been switched.

In order to write data to FIFO a device must set valid data on the DataInBulk bus and set the TxActive port to one. Next, device must set the TxValidBulk port to one. When TxValidHBulk is set to one it indicates that all 16 bits of the DataInBulk bus are valid. If TxValidHBulk is not set it means that only the lowest 8 bits of the DataInBulk bus are valid. The data is written to FIFO on a rising edge of CLK when TxActive is asserted and TxValidBulk are asserted too.

When the BusyIn output port is low then the Device can start writing data to FIFO.

Figure 6.3 shows waveforms where seven bytes are written to Bulk IN 2 endpoint FIFO. Vertical, dashed lines indicate moments when the data are written to FIFO.

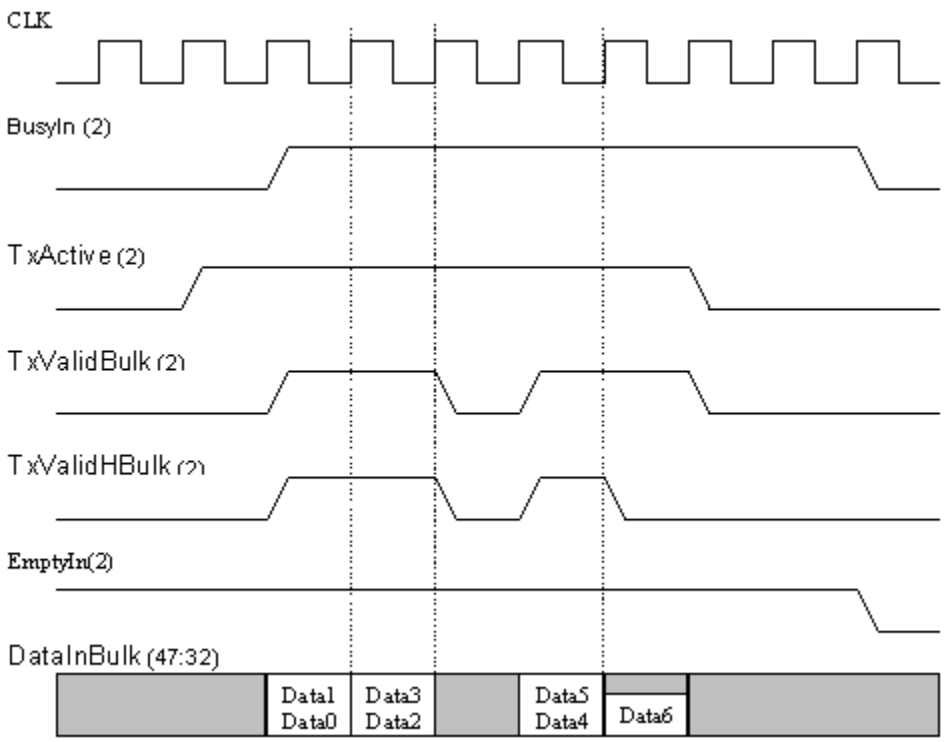

Figure 0.3 Writing data to FIFO

Only the last data word may contain one byte. All previous data words must be two byte wide. The USB host sees the bulk IN endpoint as single FIFO. If the USB host requires data from an endpoint, it sends IN token to this endpoint. Responding to the token, the endpoint sends data residing in FIFO. The data is sent in the same order as it is writen to FIFO by a device. The contents of the FIFO buffer must be sent in a single packet. The length of the packet cannot exceed the max packet size defined in the endpoint descriptor.

In order to send an empty packet, the external device sets the following state on input ports:

- TxActive is high,
- TxValidBulk is low,
- TxValidHBulk is low.

This condition must last at least one clock cycle. Figure 6.4 shows how to send an empty packet.

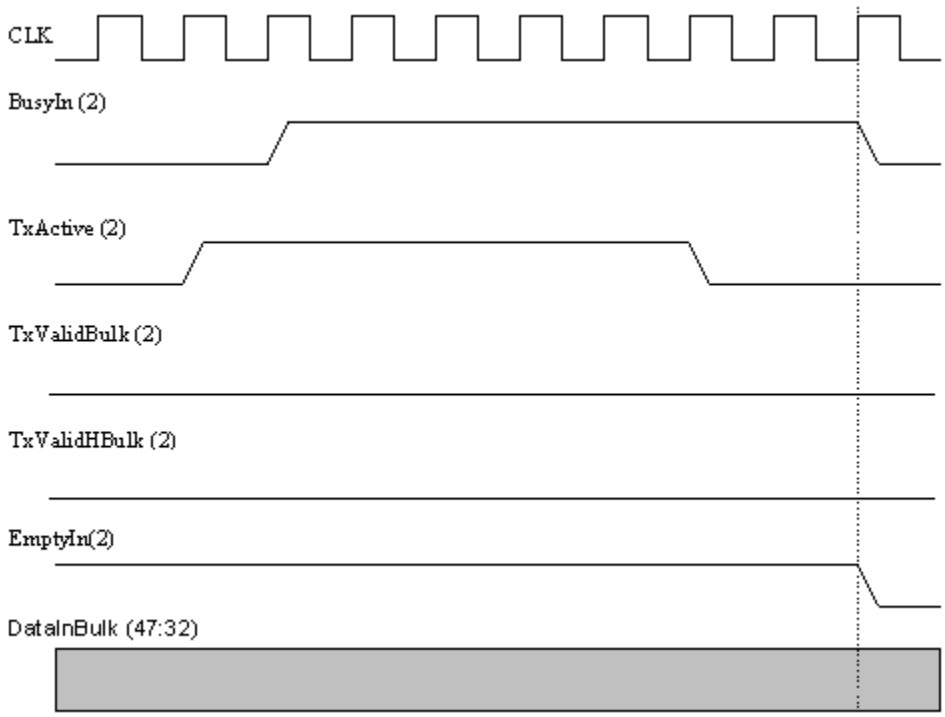

Figure 0.4 Sending empty packet to the Host

The USB core sends empty packet and clears the EmptyIn port. The USB core sends NAK handshake in response to IN Token if the endpoint's buffer is empty.

The USB core does not respond to IN Token addressed to an endpoint, which does not exist or is invalid. An endpoint is invalid if corresponding bit in the Valid register is not set. If endpoint's STALL bit is set then the USB core sends the STALL handshake in response to IN Token. The table below shows the set of control registers, which are used to reset an endpoint or to change its toggle bit.

Table 0.4 Bulk In control registers

address	Name	Description	Dir
10h	IN0CRTL	Sets toggle and resets Bulk0 IN endpoint	W
11h	IN 1CRTL	Sets toggle and resets Bulk1 IN endpoint	W
12h	IN 2CRTL	Sets toggle and resets Bulk2 IN endpoint	W
13h	IN 3CRTL	Sets toggle and resets Bulk3 IN endpoint	W
14h	IN 4CRTL	Sets toggle and resets Bulk4 IN endpoint	W
15h	IN 5CRTL	Sets toggle and resets Bulk5 IN endpoint	W
16h	IN 6CRTL	Sets toggle and resets Bulk6 IN endpoint	W
17h	IN 7CRTL	Sets toggle and resets Bulk7 IN endpoint	W

The following table shows one of the control registers. Each of them contains the same set of control bits for different endpoints.

Table 0. IN0CNTRL

IN0CNTRL		Set toggle and reset Bulk0 IN endpoint				X10	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP – Reset Endpoint - if high, it resets coresponding endpoint.

SET_ENDP – Change Toggle bit – if high, it changes value of corresponding toggle bit.

VAL_SET – Value of toggle bit – this value is written to toggle bit if SET_ENDP is high.

Interrupt IN endpoints.

The interrupt transfer type is designed to deliver data at a rate not slower than it is specified by endpoint's descriptor. The USB host queries the interrupt endpoints with this specified period of time. Retry of the transfer attempts at the next period in the case of occasional delivery failure due to error on the bus. The ALUSB 2.0 core supports up to 4 interrupt IN endpoints that can operate in High Speed mode as well as in Full speed mode. The interrupt endpoints are independent of each other and are referenced from 0 to 7. The endpoint's descriptor specifies the data payload size, which an endpoint can transmit. The maximum allowable interrupt data payload is 64 bytes or less for full-speed. High-speed endpoints are allowed to transfer maximum up to 1024 bytes data payload.

The Interface for all Interrupt IN endpoints consists of the following ports:

- DataInBulk (101 downto 128),
- TxActive (11 downto 8),
- TxValidBulk (11 downto 8),
- TxValidHBulk (11 downto 8),
- EmptyIn(11 downto 8),
- BusyIn (11 downto 8).

Every Interrupt IN endpoint uses these ports to communicate with an external device, which sends the data. Figure 6.1 shows ports of the ALUSB 2.0 core that are the Interrupt IN endpoints interface.

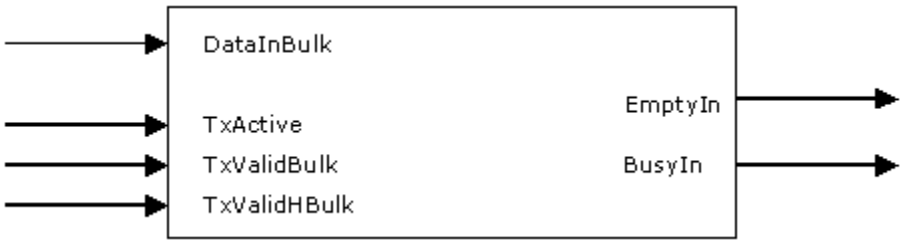

Figure 0.1 Interrupt In interface

Single endpoint utilizes:

- 16 bits of DataInBulk,
- 1 bit of TxActive,
- 1 bit of TxValidBulk,
- 1 bit of TxValidHBulk,
- 1 bit of EmptyIn,
- 1 bit of BusyIn.

The table 7.1 shows how these signals are assigned to individual endpoints.

Table 0.1 Signal assignment to individual interrupt In endpoints

	1	**2**	**3**	**4**
DataInBulk	(143:128)	(159:144)	(175:160)	(191:176)
TxActive	8	9	10	11
TxValidBulk	8	9	10	11
TxValidHBulk	8	9	10	11
EmptyIn	8	9	10	11
BusyIn	8	9	10	11

For example endpoint 1 IN utilizes:

- DataInBulk (159 downto 144),
- TxActive (9),
- TxValidBulk (9),
- TxValidHBulk (9),
- EmptyIn(9),
- BusyIn (9).

Endpoint 3 IN utilizes:

- DataInBulk (191 downto 176),
- TxActive (11),
- TxValidBulk (11),
- TxValidHBulk (11),
- EmptyIn(11),
- BusyIn (11) etc.

To use a given Interrupt IN endpoint, the corresponding bit in the VALID_in_H register (45h) must be set and the corresponding bit in the STALL_in_H register (49h) must be cleared.

Table 0.2 Valid In Register high byte

Valid_in_H		Valid In Register high byte				x45	
b7	b6	b5	b4	b3	b2	b1	b0
Iso3	Iso2	Iso1	Iso0	Interrupt 3	Interrupt 2	Interrupt 1	Interrupt 0
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

Table 0.3 Stall Register high byte

Stall_in_H		Stall In Register high byte				x49	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	Interrupt 3	Interrupt 2	Interrupt 1	Interrupt 0
W	W	W	W	W	W	W	W
1	1	1	1	1	1	1	1

Each Interrupt In endpoint contains a control register. This register can be used to reset an endpoint or to change the value of the toggle bit. The table below shows a set of the control registers.

Table 0.4 Conterol register of interrupt IN endpoints

address	Name	Description	Dir
18h	IN8CRTL	Set toggle and reset Interrupt0 IN endpoint	W
19h	IN9CRTL	Set toggle and reset Interrupt1 IN endpoint	W
1Ah	IN10CRTL	Set toggle and reset Interrupt2 IN endpoint	W
1Bh	IN11CRTL	Set toggle and reset Interrupt3 IN endpoint	W

The following table shows one of the control registers.

Table 0.5 IN8CRTL register

IN8CRTL			Set toggle and reset Interrupt0 IN endpoint			X18	
b7	b6	B5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP – Reset Endpoint - if high, It resets a corresponding endpoint.

SET_ENDP – Change Toggle bit – if high, it changes value of corresponding toggle bit.

VAL_SET – Value of the toggle bit – this value is written to toggle bit if SET_ENDP is high.

Bulk OUT endpoints.

The ALUSB 2.0 core supports up to 8 Bulk OUT endpoints (including control endpoint zero). All Bulk endpoints can operate in the Full Speed mode as well as in the High Speed mode. The Bulk OUT endpoints are independent of each other and referenced from 0 to 7.

The USB 2.0 specification defines the maximum bulk data payload sizes to be only 8, 16, 32 or 64 bytes for full-speed endpoints and 512 bytes for high-speed endpoints. Each bulk endpoint is designed to support the maximum data payload size. The interface for all Bulk OUT endpoints consists of the following ports:

- DataOutBulk (127 downto 0),
- RxValidBulk (7 downto 0),
- RxValidHBulk (7 downto 0),
- RxReadyBulk (7 downto 0),

- EmptyOut (7 downto 0),
- BusyOut(7 downto 0).

Each Bulk OUT endpoint uses these ports to communicate with an external device. Figure 8.1 shows the ports of ALUSB 2.0 that are interface of the Bulk OUT endpoints.

Figure 0.1 Bulk Out interface

A single endpoint utilizes:

- 1 bit of RxValidBulk (7 downto 0),
- 1 bit of RxValidHBulk (7 downto 0),
- 16 bits of DataOutBulk (127 downto 0),
- 1 bit of RxReadyBulk(7 downto 0),
- 1 bit of EmtpyOut (7 downto 0),
- 1 bit of BusyOut(7 downto 0).

The table 8.1 shows how these signals are assigned to individual endpoints.

Table 0.1 Signal Assignment to individual Bulk OUT endpoints

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
RxValidBulk	0	1	2	3	4	5	6	7
RxValidHBulk	0	1	2	3	4	5	6	7
DataOutBulk	(15:0)	(31:16)	(47:32)	(63:48)	(79:64)	(95:80)	(111:96)	(127:112)
RxReadyBulk	0	1	2	3	4	5	6	7
EmptyOut	0	1	2	3	4	5	6	7
BusyOut	0	1	2	3	4	5	6	7

For example endpoint 1 OUT utilizes:

- 1 bit of RxValidBulk (1),
- 1 bit of RxValidHBulk (1),
- 16 bits of DataOutBulk (31 downto 16),

- 1 bit of RxReadyBulk(1),
- 1 bit of EmptyOut (1),
- 1 bit of BusyOut(1).

Endpoint 3 OUT utilizes:

- 1 bit of RxValidBulk (3),
- 1 bit of RxValidHBulk (3),
- 16 bits of DataOutBulk (63 downto 48),
- 1 bit of RxReadyBulk(3),
- 1 bit of EmptyOut (3),
- 1 bit of BusyOut(3).

Bulk OUT endpoints assure data flow from USB Host to any device.

To use given Bulk OUT endpoint, the corresponding bit in the VALID_out_L register (46h) must be set to one and the corresponding bit in the STALL_out_L register (4Ah) must be reset to zero.

Table 0.2 Valid In Register low byte

Valid_in_L		Valid In Register low byte				x46	
b7	b6	b5	b4	b3	b2	b1	b0
Bulk 7	Bulk 6	Bulk 5	Bulk 4	Bulk 3	Bulk 2	Bulk 1	Bulk 0
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	1

Table 0.3 Stall In Register low byte

Stall_out_L		Stall In Register low byte				x4A	
b7	b6	b5	b4	b3	b2	b1	b0
Bulk 7	Bulk 6	Bulk 5	Bulk 4	Bulk 3	Bulk 2	Bulk 1	Bulk 0
W	W	W	W	W	W	W	W
1	1	1	1	1	1	1	0

The USB Host sends data to specific Bulk OUT endpoint. The data are received by an endpoint controller and written to double-buffered FIFO. The endpoint controller has access only to one FIFO buffer while the second buffer is accessed by an external device.

When the device has finished reading data then the USB core sets corresponding bit of the EmptyOut port to one. It means that the FIFO buffer on the device side is empty.

When the endpoint controller is writing data to the FIFO buffer, then the corresponding bit of the BusyOut port is set to one. If the endpoint controller has finished writing data to specific FIFO buffer and the second buffer is empty then the USB core switches these buffers.

The FIFO buffers are always switched when the FIFO buffer on the device side is empty and the second buffer is full. Both corresponding bits in BusyOut and EmptyOut ports go low after the buffer switch. Figure 8.2 shows how FIFO buffers are accessed and switched.

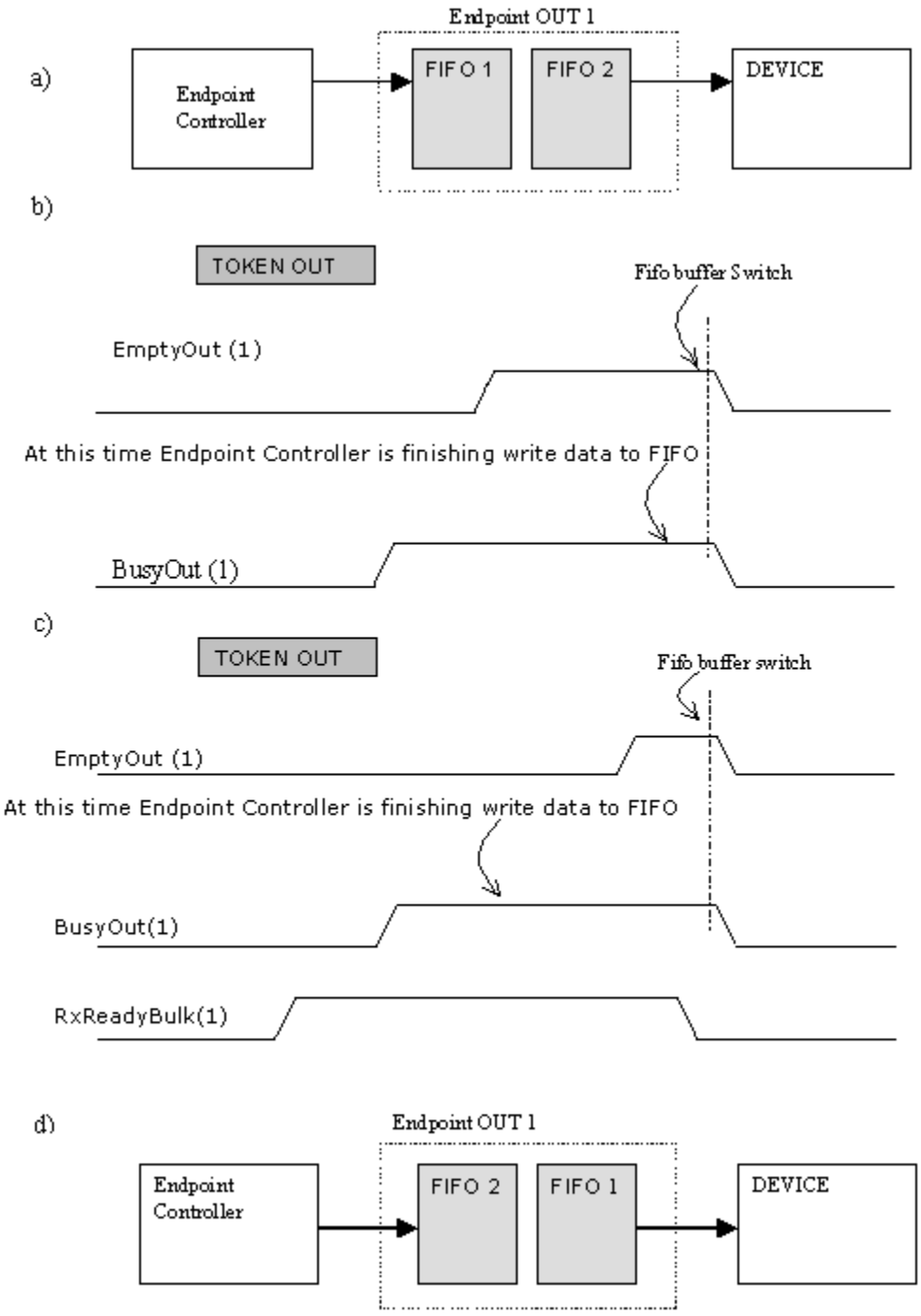

Figure 0.2 Fifo toggle in Bulk Out endpoints

Figure 8.2a shows the FIFO buffers before a switch. The endpoint controller writes data to the FIFO1 buffer and the FIFO2 buffer is read by an external device. The EmptyOut(1) port is low indicating that the FIFO2 buffer is not empty. The BusyOut(1) port is high indicating that the FIFO1 buffer is busy. Figure 8.2b shows a moment when the FIFO buffers are switched. If an external device finishes reading data from the FIFO buffer first, the USB core waits until the endpoint controller fills the second buffer. After that, these buffers are switched. Both EmptyOut(1) and BusyOut(1) flags go low after the switch.

If the endpoint controller finishes writing data first and the external device is still reading data from its buffer. The USB core waits until device finishes reading and switches the buffers. See Figure 8.2c. The FIFO buffers after the switch are shown on figure 8.2d. In order to read data from FIFO the external device must check the EmptyOut port. If this signal is low it means there is data in the FIFO buffer. The device sets the RxReadyBulk port to one to read this data. If this port is high the data residing in the FIFO buffer is outputted every rising edge of CLK. The RXValidBulk and RxValidHBulk ports are high indicating that the DataOutBulk bus is valid. If the RxValidHBulk is high it indicates that all 16 bits of DataOutBulk bus are valid. If RxValidHBulk is low it means that only the lowest 8 bits of the DataOutBulk bus are valid.

When RxReadyBulk is reset to zero then reading data from FIFO is suspended. The reading is resumed when RxReadyBulk is asserted again. The RxReadyBulk should be reset to zero when EmptyOut is set to one (all data has been read from FIFO).

Figure 8.3 shows waveforms where seven bytes are read from Bulk OUT 2 endpoints FIFO. The Data0. Data2, Data4, Data6 bytes appear on the DataOutBulk (7 :0) bus. The Data1, Data3, Data6 bytes appear on the DataOutBulk (15 :8) bus. The vertical, dashed lines indicate moments when the data is read from FIFO.

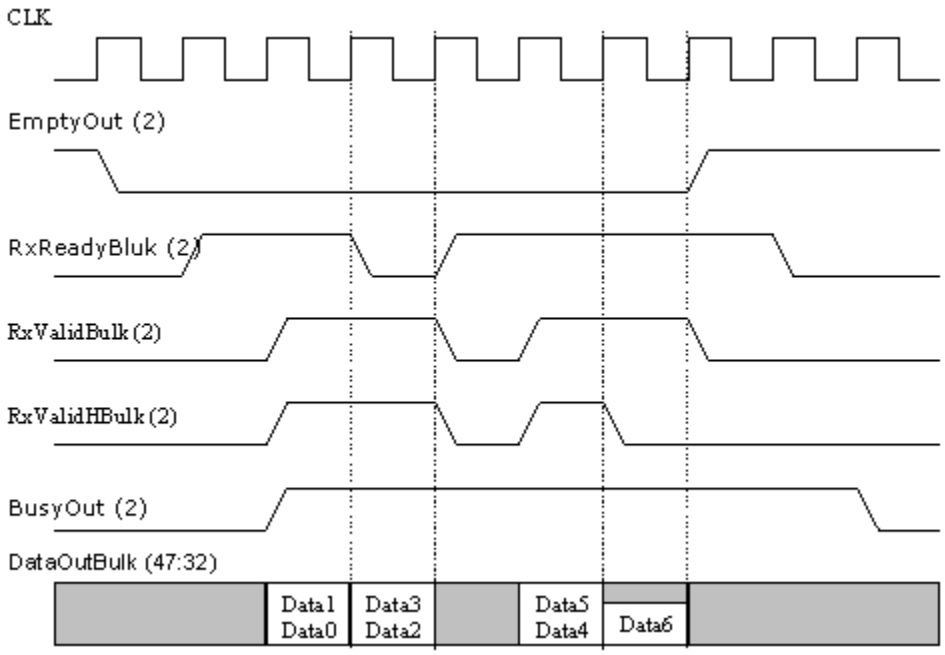

Figure 0.3 Receiving data from Bulk Out

From the device point of view, Bulk endpoint OUT is seen as single FIFO. If the device requires data from FIFO, it checks EmptyOut of specified endpoint. If this signal is zero then the device sets RxReadyBulk to

one. The data is sent in the same order as it is writen to FIFO by the USB host. Based on figure 8.3., after sending Token Out to endpoint 2, the USB host sends data in folowing order:

Data0, Data1, Data2, Data3, Data4, Data5, Data6.

Only the last data word can be read as one byte when RxValidBulk is low and RxValidHBulk is high.

The reception of an empty packet from the USB host is signaling with the following:

- RxReadyBulk is set to one (when EmptyOut is reset to zero),
- RxValidBulk is low,
- RxValidHBulk is low,
- EmptyOut is set to one after two clock cycles.

Figure 8.4 shows how empty packet is received.

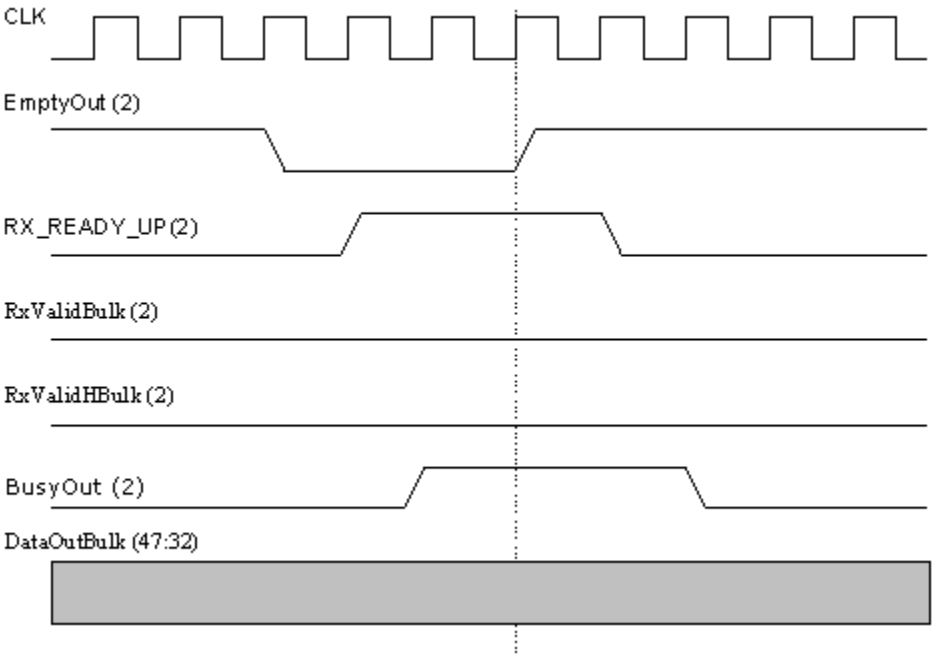

Figure 0.4 Receiving empty packet from the Host

The NAK handshake is sent to the USB host in response to the OUT token if the FIFO buffer is busy (BusyOut is set to one).

If the OUT Token is issued to not implemented endpoint or the Valid bit of the endpoint is not set, then no response is sent beck to the USB host.

If STALL bit of the endpoint is set, then STALL handshake is sent to the USB host.

In order to configure given endpoint, the corresponding OUTxCRTL register must be updated. The addresses of Bulk Out endpoints are shown in the following table:

Table 0.4 Bulk OUT endpoint control registers

address	Name	Description	Dir
00h	OUT0CRTL	Sets toggle and reset Bulk0 OUT endpoint	W
01h	OUT1CRTL	Sets toggle and reset Bulk1 OUT endpoint	W
02h	OUT2CRTL	Sets toggle and reset Bulk2 OUT endpoint	W
03h	OUT3CRTL	Sets toggle and reset Bulk3 OUT endpoint	W
04h	OUT4CRTL	Sets toggle and reset Bulk4 OUT endpoint	W
05h	OUT5CRTL	Sets toggle and reset Bulk5 OUT endpoint	W
06h	OUT6CRTL	Sets toggle and reset Bulk6 OUT endpoint	W
07h	OUT7CRTL	Sets toggle and reset Bulk7 OUT endpoint	W

The tabe below shows the OUT0CRTL register.

Table 0.5 OUT0CRTL register

OUT0CRTL			Set toggle and reset Bulk0 OUT endpoint		X00		
b7	B6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint.

When set to one then endpoint Bulk0IN is RESET.

SET_ENDP – sets the Toggle bit in Bulk0IN endpoint to a value depending on VAL_SET.

If VAL_SET and SET_ENDP are high then the toggle bit is set to one.

If VAL_SET is low and SET_ENDP is high then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

The above-mentioned description concernes all control registers of Bulk OUT endpoints (x10 - x17).

Bulk zero endpoint

All USB devices are required to implement a default control endpoint that uses both the input and output endpoint with endpoint number zero. The USB System uses control endpoint to initialize and generically manipulate the logical device (e.g., to configure the logical device). The default control endpoint provides access to the device's configuration information and allows the generic USB status and control access.

Interrupt OUT endpoints

The interrupt transfer type is designed to support those devices that need to send or receive data infrequently but with bounded service periods. The interrupt transfer provides:

- Guaranteed maximum service period for the pipe
- Retry of transfer attempts at the next period, in the case of occasional delivery failure due to error on the bus

The ALUSB 2.0 core supports up to 4 Interrupt OUT endpoints that can operate in High Speed mode as well as in Full speed mode. Each of Interrupt OUT endpoints is independent. The Interrupts endpoints are referenced from 8 to 11. The maximum allowable interrupt data payload size is 64 bytes or less for full-speed devices. High-speed endpoints are allowed to transmit the maximum data payload sizes up to 1024 bytes.

Interface for all Interrupt OUT endpoints consists of the following ports:

- DataOutBulk (191 downto 128),
- RxValidBulk (11 downto 8),
- RxValidBulk (11 downto 8),
- RxReadyBluk (11 downto 8),
- EmptyOut (11 downto 8),
- BusyOut(11 downto 8).

Each Interrupt OUT endpoint uses these ports to communicate with an external device. Figure 10.1 shows ports of the ALUSB 2.0 that are interface of Interrupt OUT endpoints.

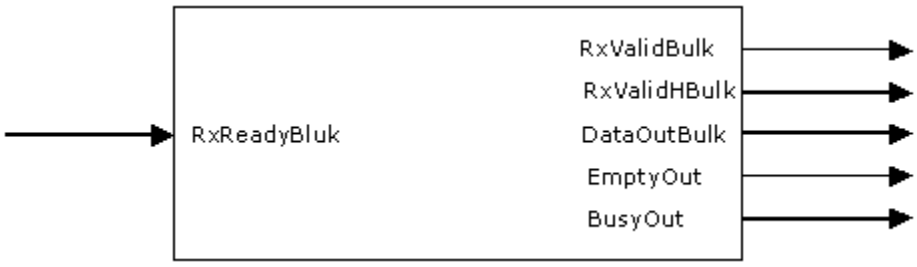

Figure 0.1 Interface of Interrupt endpoints OUT

Single endpoint utilizes:

- 1 bit of RxValidBulk (11 downto 8),
- 1 bit of RxValidHBulk (11 downto 8),
- 16 bits of DataOutBulk (191 downto 128),
- 1 bit of RxReadyBluk (11 downto 8),
- 1 bit of EmptyOut (11 downto 8),
- 1 bit of BusyOut(11 downto 8).

The table 10.1 shows how these signals are assigned to individual endpoints.

Table 0.1 Signal assignment to individual Interrupt OUT endpoints

	0	**1**	**2**	**3**
RxValidBulk	0	1	2	3
RxValidHBulk	0	1	2	3
DataOutBulk	(143:128)	(159:144)	(175:160)	(191:176)
RxReadyBluk	0	1	2	3
EmptyOut	0	1	2	3
BusyOut	0	1	2	3

For example endpoint OUT 1 utilizes:

- 1 bit of RxValidBulk (1),
- 1 bit of RxValidHBulk (1),
- 16 bits of DataOutBulk (159 downto 144),
- 1 bit of RxReadyBluk (1),
- 1 bit of EmptyOut (1),
- 1 bit of BusyOut(1).

Endpoint OUT 3 utilizes:

- 1 bit of TX_VALID_UP (3),
- 1 bit of TX_VALIDH_UP (3),
- 16 bits of DataOutBulk (191 downto 176),
- 1 bit of RxReadyBluk (3),
- 1 bit of EmptyOut (3),
- 1 bit of BusyOut(3).

The interrupt OUT endpoints assure data flow from USB Host to any device.

To use given Interrupt OUT endpoint, the corresponding bit in the VALID_out_H register (47h) must be set to one and the correspondent bit in STALL_out_H register (4Bh) must be reset to zero.

Table 0.2 Valid OUT Register high byte

Valid_out_H		Valid OUT Register high byte				x47	
b7	b6	b5	b4	b3	b2	b1	b0
Iso 3	Iso 2	Iso 1	Iso 0	Interrupt 3	Interrupt 2	Interrupt 1	Interrupt 0
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

Table 0.3 Stall OUT Register high byte

Stall_out_L		Stall OUT Register high byte				x4B	
b7	b6	b5	b4	b3	b2	b1	b0
Iso 3	Iso 2	Iso 1	Iso 0	Interrupt 3	Interrupt 2	Interrupt 1	Interrupt 0
W	W	W	W	W	W	W	W
1	1	1	1	1	1	1	1

Access to Interrupt endpoints is the same as to Bulk OUT endpoints. The behavior of the interface is exactly the same.

In order to configure a given endpoint, the corresponding OUTxCTRL register must be updated. The addresses of Interrupt Out endpoints are shown in the following table:

Table 0.4 Control registers of Interrupt OUT endpoints

address	Name	Description	Dir
08h	OUT8CRTL register	Set toggle and reset Interrupt0 OUT endpoint	W
09h	OUT9CRTL register	Set toggle and reset Interrupt1 OUT endpoint	W
0Ah	OUT10CRTL register	Set toggle and reset Interrupt2 OUT endpoint	W
0Bh	OUT11CRTL register	Set toggle and reset Interrupt3 OUT endpoint	W

The following table shows the Interrupt endpoint 0 OUT control register.

Table 0.5 OUT8CRTL

OUT8CRTL			Set toggle and reset Interrupt0 OUT endpoint			X08	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint. The endpoint is reset when this flag is high.

SET_ENDP – sets Toggle bit for Interrupt 8 OUT endpoint to value depending on VAL_SET. If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one. If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero. If SET_ENDP is reset to zero then toggle bit is not updated.

The above-mentioned description concernes all control registers of Interrupt endpoints (x08 – x0B).

Additional Features:

The USB 2.0 core contains additional control registers which perfom special operation on Bulk endpoints. The operation are listed in the following table:

Table 10.6 Feature Control Registers.

Name of register	**Write to register**	**Action**
Feature control 0	X01	reset all Bulk and Interrupt endpoints IN and OUT
Feature control 1	X01	reset all Bulk endpoints IN and OUT
Feature control 2	X01	reset all Bulk endpoints OUT
Feature control 3	X01	reset all Bulk endpoints IN
Feature control 4	X01	reset all Interrupt endpoints IN and OUT
Feature control 5	X01	reset all Interrupt endpoints OUT
Feature control 6	X01	reset all Interrupt endpoints IN
Feature control 7	X01	reset all Bulk and Interrupt endpoints OUT
Feature control 8	X01	reset all Bulk and Interrupt endpoints IN
Feature control 0	X02	set all toggle bits of Bulk and Interrupt endpoints IN and OUT to ‘0’
Feature control 1	X02	set all toggle bits of Bulk and Interrupt endpoints IN and OUT to ‘1’
Feature control 2	X02	set all toggle bits of Bulk and Interrupt endpoints IN to ‘0’
Feature control 3	X02	set all toggle bits of Bulk and Interrupt endpoints IN to ‘1’
Feature control 4	X02	set all toggle bits of Bulk and Interrupt endpoints OUT to ‘0’
Feature control 5	X02	set all toggle bits of Bulk and Interrupt endpoints OUT to ‘1’
Feature control 6	X02	set all toggle bits of Bulk endpoints IN and OUT to ‘0’
Feature control 7	X02	set all toggle bits of Bulk endpoints IN and OUT to ‘1’
Feature control 8	X02	set all toggle bits of Interrupt endpoints IN and OUT to ‘0’
Feature control 9	X02	set all toggle bits of Interrupt endpoints IN and OUT to ‘1’

The addresses of these registers are as follows:

Table 0.7 Control features registers

address	Name	Description	Dir
20h	Feature control 0	Resets all Bulk and Interrupt IN and OUT endpoints	W
21h	Feature control 1	Resets all Bulk IN and OUT endpoints and or set all Bulk and Interrupt endpoints IN and OUT	W
22h	Feature control 2	Resets all Bulk OUT and or clear all Bulk and Interrupt endpoints IN	W
23h	Feature control 3	Resets all Bulk IN and or set all Bulk and Interrupt endpoints IN	W
24h	Feature control 4	Resets all Interrupt endpoints IN and OUT and or clear all Bulk and Interrupt endpoints OUT	W
25h	Feature control 5	Resets all Interrupt endpoints OUT and or set all Bulk and Interrupt endpoints OUT	W
26h	Feature control 6	Resets all Interrupt endpoints IN and or clear all Bulk endpoints IN and OUT	W
27h	Feature control 7	Resets all Bulk and Interrupt endpoints OUT and or set all Bulk endpoints IN and OUT	W
28h	Feature control 8	Resets all Bulk and Interrupt endpoints IN and or clear all Interrupt endpoints IN and OUT	W
29h	Feature control 9	Sets all Interrupt endpoints IN and OUT	W

Isochronous IN endpoints

The ALUSB 2.0 core supports up to 4 Isochronous IN endpoints that can operate in Full Speed mode as well as in High speed mode. Each Isochronous In endpoint is independent on other endpoints. Isochronous endpoints are referenced from 12 to 15.

The interface for all IN isochronous endpoints consists of the following ports:

- DataInIso (63 downto 0),
- BusyIso(3 downto 0),
- TxValidIso(3 downto 0),
- TxValidHIso (3 downto 0),
- FTgg(3 downto 0).

Each Isocronous In endpoint uses these ports to communicate with an external device. Figure 11.1 shows ports that are interface of Isochronous IN endpoints.

Figure 0.1 Isochronous In interface

Single endpoint utilizes:

- 16 bits of DataInIso,
- 1 bit of BusyIso,
- 1 bit of TxValidIso,
- 1 bit of TxValidHIso,
- 1 bit of FTgg.

The table 11.1 shows how these signals are assigned to individual endpoints.

Table 0.1 Signal assignment to individual Isochronous IN endpoints

Endpoint number	DataInIso	BusyIso	TxValidIso	TxValidHIso	FTgg
12	(15 :0)	0	0	0	0
13	(31 :16)	1	1	1	1
14	(47 :32)	2	2	2	2
15	(63 :48)	3	3	3	3

It means that endpoint 12 IN utilizes:

- DataInIso(15 downto 0),
- Busy(0),
- TxValidIso(0),
- TxValidHIso(0)
- FTgg(0).

Endpoint 13 IN utilizes:

- DataInIso(31 downto 15),
- BusyIso(1),
- TxValidIso(1),
- TxValidHIso(1),
- FTgg(1) etc.

To use given isochronous IN endpoint, the corresponding bit in VALID_in_H register (45h) must be set to one.

Isochronous IN endpoints assure data flow from any device to USB Host in constant rate. A device writes data to FIFO organized as double, independent buffers located inside each Isochronous IN endpoint. The FIFO buffers are switched when first Token IN, addressed to specified endpoint, comes from the USB Host after SOF token occurrence. Thus, at a given moment a device has access to only one FIFO buffer, the second FIFO buffer can then be read by USB Host. FIFO buffers switch is indicated by generating impulse on the corresponding bit in the FTgg port. The pulse lasts one CLK cycle.

Figure 11.2 shows how FIFO buffers are accessed and switched.

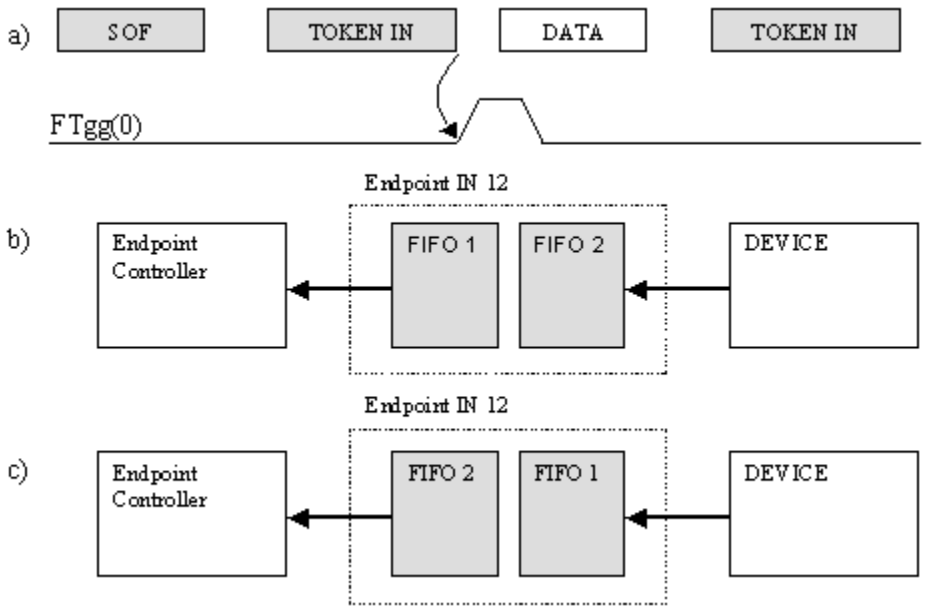

Figure 0.2 FIFO switch in Isochronous In Endpoint

Figure 11.2b shows how FIFO buffers are connected before the switch. The FIFO1 buffer is on the Endpoint Controller side and FIFO2 buffer is written by device.

Figure 11.2a shows moment when FIFO buffers are switched. The USB Host sends SOF token and subsequently issues Token IN to Isochronous IN Endpoint. In response to this token, FIFO buffers are switched and impulse on FTgg(0) is generated. The ALUSB core sends the data from FIFO2 buffer to the USB Host. After that, next IN Token is issued to this endpoint. This token does not cause the FIFO switch.

Note:

Only first Token In to specified endpoint in (micro)frame causes FIFO buffers switch in specified endpoint.

After the switch (figure 11.2 c), FIFO2 buffer is read by Endpoint Controller and FIFO1 buffer can be written by a device. This way, the Endpoint Controller can read data and send it to USB Host. The external device can write data to its FIFO buffer at the same time.

A device must fill its FIFO buffer between two rising edges on corresponding bit in the FTgg port. If there is no data to send in FIFO, empty packet is sent to the USB Host. If Token IN is issued to not implemented endpoint or endpoint's Valid bit is not set, then no response is set to the USB Host.

In order to write data to the FIFO a device must set valid data on DataInIso port and set TxValidIso signal. When the TxValidHIso is also set to one it indicates that all 16 bits of DataInIso bus are valid. If TxValidHIso is not set it means that only the lowest 8 bits of DataInIso bus are valid. The data are written to FIFO on rising edge of CLK when TxValidIso is asserted.

The Busy output port indicates that corresponding endpoint performs internal operation, and it is not able to read any data from a device. If a device sees BusyIso set to one, and rising edge on CLK occurs it must not activate TxValidIso signal. A device can write data when Busy is cleared and rising edge of CLK occurs. Figure 11.3 shows waveforms where seven bytes are written to ISO IN 12 endpoint buffer.

The Data0, Data2, Data4, Data6 bytes appear on the DataInIso(7 :0) bus. The Data1, Data3, Data6 bytes appear on DataInIso(15 :8) bus. The vertical, dashed lines indicate moments, when the data are written to FIFO.

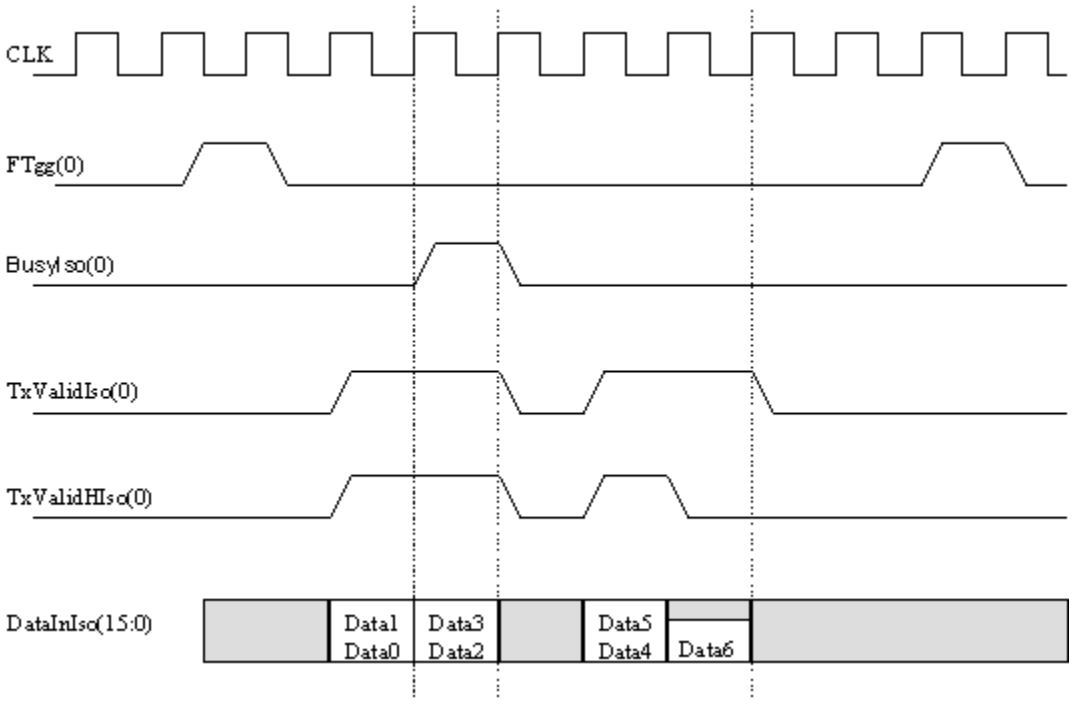

Figure 0.3 Writing data to Isochronous In endpoint

The FTgg(0) port is set to one indicating that FIFO buffers are switched. Next, device sets valid Data 0 and Data1 bytes on DataInIso bus. Data1 byte is written to FIFO because TxValidHIso is set to one.

Subsequently, Data2 and Data3 bytes are valid on the bus and the BusyIso output it asserted. Responding to that, device must suspend write operation until BusyIso goes low.

The Data4 and Data5 bytes are written to FIFO on the first rising clock edge when TxValid signals are assreted and BusyIso(0) is cleared.

From the USB Host point of view, isochronous IN endpoint is seen as single FIFO. If the USB Host requires data from FIFO, it send IN tokens to specified endpoint. Responding to these tokens, the ALUSB 2.0 core sends data residing in FIFO buffer located on the Endpoint Controller side. Data are sent in compliance with MAX_PACKET_SIZE descriptor (see chapter 5.9 in Universal Serial Bus Specification Revision 2.0) in the same order as they were writen to FIFO by a device. Based on figure 11.3, after sending IN token to endpoint 12, the USB Host receives data in folowing order: Data0, Data1, Data2, Data3, Data4, Data5, Data6.

A device must assure that data count written to FIFO does not exceed data count specified by MAX_PACKET SIZE descriptor. Moreover, a device must assure that all necessary data are written to FIFO between two impulses on FTgg port.

The empty packet is sent to the USB Host in response to Token IN if FIFO buffer is empty.

If Token IN is issued to not implemented endpoint or Valid bit of the endpoint is not set, then no response is sent to the HOST. If the Host retrieves bad data there is no possibility to re-send them to the Host.

In Full Speed mode Host issues only one Token IN to specified Isochronous In edpoint per frame.

In High Speed mode Host can issue up to three Tokens IN to specified Isochronous In edpoint per microframe. But only first token in microframe causes FIFO switch.

Isochronous OUT endpoints

Isochronous OUT endpoints interface.

The ALUSB 2.0 core supports up to 4 Isochronous OUT endpoints that can operate in High Speed mode as well as in Full speed mode. Each Isochronous Out endpoint is independent on other endpoints. Isochronous endpoints are references from 12 to 15.

Interface for all isochronous OUT endpoints consists of the following ports:

- DataOutIso(63 downto 0),
- Ce_uP(3 downto 0),
- RxValidIso(3 downto 0),
- RxValildHIso(3 downto 0),
- FTgg(7 downto 4)
- CRCErr(3 downto 0).

Each Isochronous OUT endpoint uses these ports to communicate with an external device. Figure 12.1 shows ports of the ALUSB 2.0 core that are interface of the Isochronous OUT endpoints.

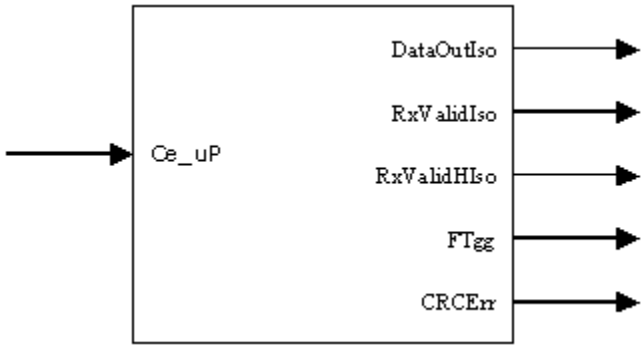

Figure 0.1 Isochronous Out interface

Single endpoint utilizes:

- 16 bits of DataOutIso,
- 1 bit of Ce_uP,
- 1 bit of RxValidIso,
- 1 bit of RxValidHIso,
- 1 bit of FTgg,
- 1 bit CRCErr.

The table 12.1 shows how these signals are assigned to individual endpoints.

Table 0.1 Signal assignment to individual Isochronous OUT endpoints

Endpoint number	DataOutIso	Ce_uP	RxValidIso	RxValidHIso	FTgg	CRCErr
12	(15 :0)	0	0	0	4	0
13	(31 :16)	1	1	1	5	1
14	(47 :32)	2	2	2	6	2
15	(63 :48)	3	3	3	7	3

It means that endpoint 12 OUT utilizes:

- DataOutIso (15 downto 0),
- CE_uP(0),
- RxValidIso (0),
- RxValidHIso (0),
- FTgg(4),
- CRCErr(0).

Endpoint 13 OUT utilizes :

- DataOutIso (31 downto 15),
- Ce_uP(1),
- RxValidIso (1),
- RxValidHIso (1),
- FTgg(1),
- CRCErr (1) etc.

To use given isochronous OUT endpoint, the corresponding bit in VALID_out_H register (47h) must be set to one.

Isochronous OUT endpoints assure data flow from USB Host to any device at constant rate. A device reads the data from FIFO organized as double, independent FIFO buffers located inside each Isochronous OUT endpoint.

The FIFO buffers are switched when the USB core receives Token OUT addressed to specified endpoint followed by SOF token.

If SOF token occurs but in the current (micro)frame the USB Host has not issued Token OUT to this endpoint then FIFO buffers are not switched.

At a given moment a device has access to only one FIFO buffer. The Endpoint Manager can write data to the second FIFO buffer at the same time. The FIFO buffers switch is indicated by pulse generation on the corresponding bit on the FTgg port. The pulse lasts one CLK cycle.

Figure 12.2 shows how FIFOS are accessed and switched.

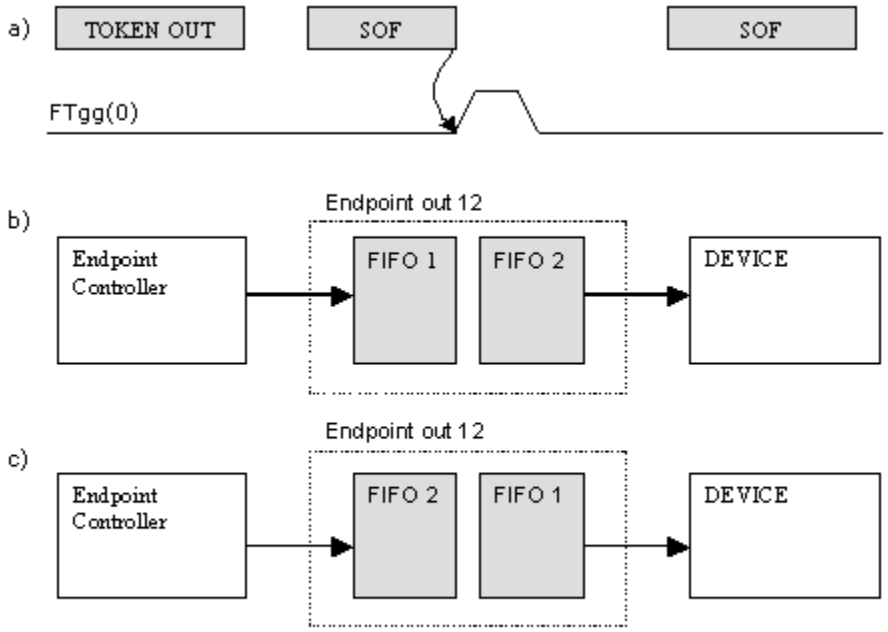

Figure 0.2 Fifo toggle in Isochronous OUT

Figure 12.2b shows how FIFOS are connected before the switch. The FIFO1 buffer is on the Endpoint Controller side and the FIFO2 buffer is being read by a device.

Figure 12.2a shows moment when FIFOS are switched. The Host sends Token OUT to Isochronous Endpoint OUT 12, and subsequently, it issues SOF packet. In response to this packet, FIFOS are switched and an impulse on FTgg(0) is generated. After falling edge on FTgg(0) the device can read data from FIFO.

Note:
Isochronous OUT Endpoints FIFOS are switched when SOF occurs after Host sent Token OUT to specified endpoint in previous frame. This mechanism is different from FIFO toggle applied to Isochronous IN Endpoints.

After the FIFO buffer switch (figure 12.2 c), the FIFO 2 buffer can be written by the endpoint controller and FIFO 1 buffer can be read by a device. This way, a device can read data that the Endpoint controller wrote to FIFO 2 before FIFO switch.

The device must manage to read its FIFO buffer between two rising edges on the corresponding bit in the FTgg port, otherwise, unread data will be lost.

In order to read the data from FIFO, the device must set CE_uP to one. After two cycles of CLK, valid data occurs on DataOutIso bus. RxValidIso and RxValidHIso signals indicate validity of the read data.

If the device has set CE_uP to one and after two cycles of CLK RxValidIso is not set, it means there is no data in the FIFO buffer.

If the device has set CE_uP to one and after two cycles of CLK RxValidIso is set, but RxValidHIso is not set, it means that only one byte is available in the FIFO buffer.

If a device has set CE_uP to one and after two cycles of CLK RxValidIso is set, and RxValidHIso is set to one as well, it means that two bytes on the bus are valid, and it is necessary to perform another read operation to check if there are any data in the FIFO.

Read operation can be performed as burst and as single read.

In burst, CE_uP is set to one and held in high state for more than one CLK cycle. Figure 12.3 shows reading from FIFO in burst mode. Vertical, dashed lines indicate moments when ALUB 2.0 issues valid data on DataOutIso bus.

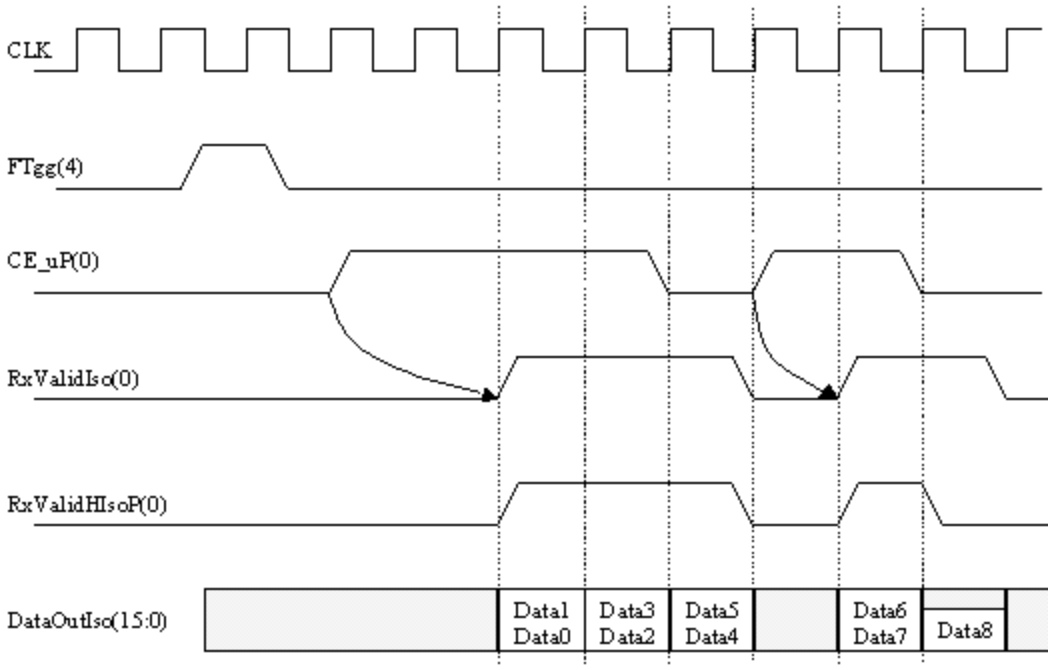

Figure 0.3 Reading data from Fifo using burst method

The USB Host sent 9 bytes to Isochronous OUT endpoint 12. After the FIFO switch the device wants to read these data. It sets Ce_uP(0) for 3 CLK cycles. Data0 and Data1 appears on DataOutIso(15:0) two CLK cycle after Ce_uP(0) was set to one. Data2 and Data3 appear during next cycle because Ce_uP(0) was still assereted. Since Ce_uP(0) lasts three CLK cycles, Data4 and Data5 are the last ones in this data blok.

In next burst Ce_uP(0) is asserted for 3 CLK cycles again. In respose to this assertion Data6 and Data7 appear. In next CLK cycle only Data8 occurs, and RxValidHIso(0) goes low. Data8 is the last byte in FIFO.

Note:
The case where RxValidHIso(0) and RxValidIso(1) occures only if a device is reading the last byte from FIFO and there was odd byte count in FIFO buffer.

In single mode, a device sets Ce_uP for one CLK cycle. Figure 12.4 shows the case when both modes are used by a device.

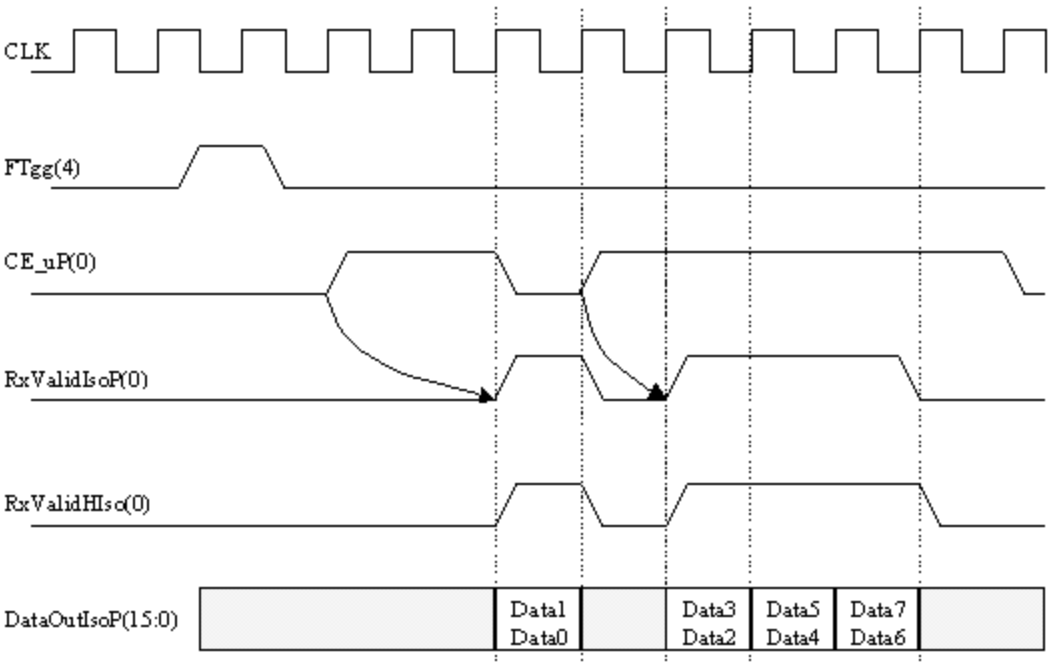

Figure 0.4 Reading data from Fifo using single and burst methods

If the CRC error occures during transmission from the USB Host to ALUSB 2.0 core, then after fifo switch, the coresponding bit of CRCErr is set to one. This way the device knows that all data residing in FIFO are corrupted. Correct data may be accesed after next fifo switch.

In Isochronous transfers the device has no possibility to inform the The USB Host about error occurance because Isochronous transmission supports neither handshakes nor re-transmission.

Debugging signals for ISO Out endpoints.

The isochronous transmission does not support handshakes and retransmission. The USB 2.0 core cannot notify the USB Host if the received packet is corrupted. The isochronous transmission is designed for real-time applications where correctness of the data is not so important as delivery on time.

The USB 2.0 core may not be able to detect the data phase of ISO transmission when PID of the received data packet is corrupted. In such case the received packet is ignored. It causes losing the entire data packet. The USB 2.0 core is able to detect such situation. Typically, data is transmitted to ISO endpoints every USB frame or microframe. The USB 2.0 core checks at the beginning of each frame/microframe if the ISO Out endpoints received data in the previous frame. The REPEATNEEDED flag is set when new frame starts and ISO Out endpoint has not received data during the previous frame. This signal informs that the packet might be lost. Each ISO OUT endpoint has individual REPEATNEEDED flag.

- REPEATNEEDED(0) – Endpoint ISO OUT 12.
- REPEATNEEDED(1) – Endpoint ISO OUT 13.
- REPEATNEEDED(2) – Endpoint ISO OUT 14.

- REPEATNEEDED(3) – Endpoint ISO OUT 15.

This signal can be used for detecting ignored packets. It should not be used during normal core operation.

The lost packet can be replaced with the previous one. If the entire packet was ignored it means that the previous packet still remains in the endpoint buffer. Activating the REPEAT signal forces the ISO OUT endpoint to treat the previous packet as the new one. The previous packet can be read by external device to fill the gap caused by ignored packet. Each ISO Out endpoint has individual REPEAT flag.

- REPEAT(0) – Endpoint ISO OUT 12.
- REPEAT(1) – Endpoint ISO OUT 13.
- REPEAT(2) – Endpoint ISO OUT 14.
- REPEAT(3) – Endpoint ISO OUT 15.

These signals are for debugging purposes only and should not be used during normal core operation. The REPEAT signal should be connected to ground and the REPEATNEEDED should be left unconnected.

Interrupts

The ALUSB 2.0 core has one INT output that is used to request interrupt to the processor.

This output is associated with Status register (40h) and Int_Enable register (4Ch).

Table 0.1 Status Register

Status_USB		Status Register				x40	
b7	b6	b5	b4	b3	b2	b1	b0
EnumEnabl	HsFs	Resume	SetDataIntr	SOF	TkSetup	SuspDetect	UsbReset
W	R	R	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Table 0.2 Interrupt Enable register

Status_USB		Interrupt Enable				x4C	
b7	b6	b5	b4	b3	b2	b1	b0
-	Disab	Res_En	SDI_En	SOF_En	TkS_En	SDet_En	UsbRe_En
R	R	R	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Status register contains flags that are updated when any of the six interrupt sources occur and Int_Enable register contains enable bits associate with flags. The Figure 13.1 shows Interrupt system in ALUSB 2.0 core.

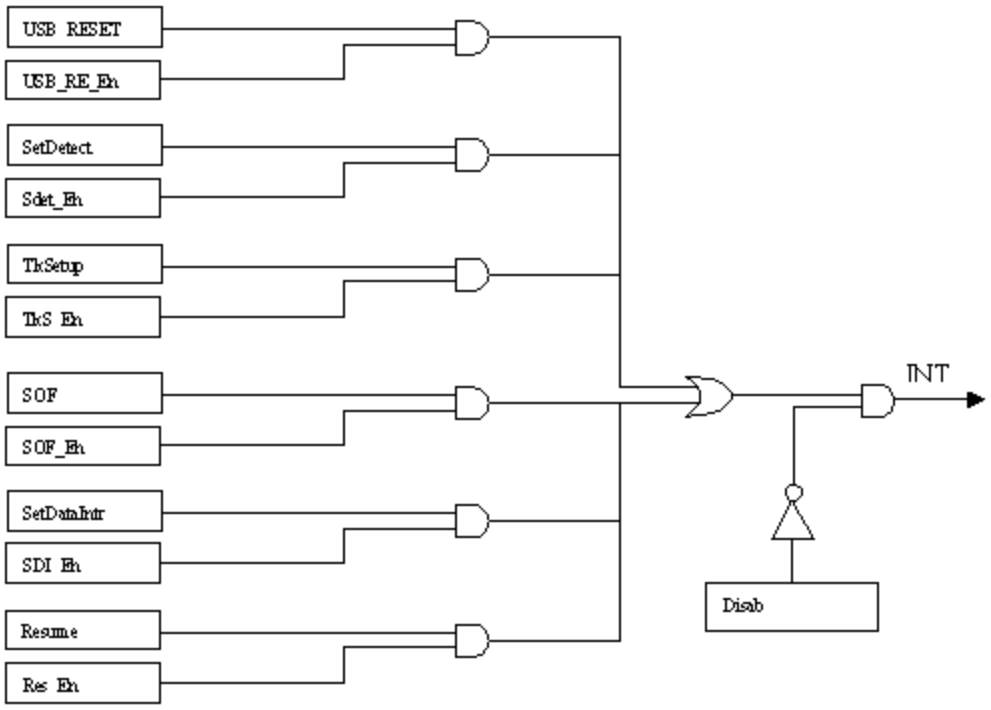

Figure 0.1 Interrupt system in the ALUSB 2.0 core

When Disab is set to one, then INT output is reset to zero even if some flags and their enable bits are set.

When Disab is reset to zero then interrupt system is enabled. If particular flags are set by the ALUSB 2.0 core and their enable bits are set to one, then INT is being set to one.

The ALUSB 2.0 core cannot clear flags associated with interrupt system.

During interrupt service, processor can clear them by writing **one** to particular flags (see section 14.1).

Registers

OUT0CRTL register

OUT0CRTL			Set toggle and reset Bulk0 OUT endpoint			X00	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint.

When set to one then OUT Bulk 0 endpoint is reset.

SET_ENDP – sets Toggle bit in Bulk 0 OUT endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

OUT1CRTL register

OUT0CRTL			Set toggle and reset Bulk1 OUT endpoint			X01	
b7	b6	b5	B4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint. When set to one then Bulk 1 OUT endpoint is reset.

SET_ENDP – sets Toggle bit in Bulk 1 OUT endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

OUT2CRTL register

OUT2CRTL			Set toggle and reset Bulk2 OUT endpoint			X02	
B7	b6	b5	B4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint. When set to one then endpoint Bulk 2 OUT is reset.

SET_ENDP – sets Toggle bit in Bulk 2 OUT endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero. If SET_ENDP is reset to zero then toggle bit is not updated.

OUT3CRTL register

OUT3CRTL			Set toggle and reset Bulk3 OUT endpoint			X03	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint. When set to one then Bulk 3 OUT endpoint is reset.

SET_ENDP – sets Toggle bit in Bulk 3 OUT endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

OUT4CRTL register

OUT4CRTL			Set toggle and reset Bulk4 OUT endpoint			X04	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint. When set to one then Bulk 4 OUT endpoint is reset.

SET_ENDP – sets Toggle bit in Bulk 4 OUT endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

OUT5CRTL register

OUT5CRTL			Set toggle and reset Bulk5 OUT endpoint			X05	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint.When set to one then Bulk 5 OUT endpoint is reset.

SET_ENDP – sets Toggle bit in Bulk 5 OUT endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero. If SET_ENDP is reset to zero then toggle bit is not updated.

OUT6CRTL register

OUT6CRTL			Set toggle and reset Bulk6 OUT endpoint			X06	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint. When set to one then Bulk 6 OUT endpoint is reset.

SET_ENDP – sets Toggle bit in Bulk 6 OUT endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero .

If SET_ENDP is reset to zero then toggle bit is not updated.

OUT7CRTL register

OUT7CRTL			Set toggle and reset Bulk7 OUT endpoint			X07	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint. When set to one then Bulk 7 OUT endpoint is reset.

SET_ENDP – sets Toggle bit in Bulk 7 OUT endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

OUT8CRTL register

OUT8CRTL			Set toggle and reset Interrupt0 OUT endpoint				X08
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint. When set to one then Interrupt 8 OUT endpoint is reset.

SET_ENDP – sets Toggle bit in Interrupt 8 OUT endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

OUT9CRTL register

OUT9CRTL			Set toggle and reset Interrupt9 OUT endpoint				X09
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint. When set to one then Interrupt 9 OUT endpoint is reset.

SET_ENDP – sets Toggle bit in Interrupt 9 OUT endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

OUT10CRTL register

OUT10CRTL			Set toggle and reset Interrupt10 OUT endpoint				X0A
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint. When set to one then Interrupt 10 OUT endpoint is RESET.

SET_ENDP – sets Toggle bit in Interrupt 10 OUT endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

OUT11CRTL register

OUT11CRTL			Set toggle and reset Interrupt11 OUT endpoint				X0B
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint. When set to one then Interrupt 11 OUT endpoint is RESET.

SET_ENDP – sets Toggle bit in Interrupt 11 OUT endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

IN0CRTL register

IN0CRTL			Set toggle and reset Bulk0 IN endpoint				X10
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint.

When set to one then endpoint Bulk 0 IN is RESET.

SET_ENDP – sets Toggle bit in Bulk 0 IN endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

IN1CRTL register

IN0CRTL			Set toggle and reset Bulk1 IN endpoint			X11	
b7	b6	b5	B4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint. When set to one then Bulk 1 IN endpoint is reset.

SET_ENDP – sets Toggle bit in Bulk 1 IN endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

IN2CRTL register

IN2CRTL			Set toggle and reset Bulk2 IN endpoint			X12	
B7	b6	b5	B4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint. When set to one then Bulk 2 IN endpoint is reset.

SET_ENDP – sets Toggle bit in Bulk 2 IN endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

IN3CRTL register

IN3CRTL			Set toggle and reset Bulk3 IN endpoint			X13	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint.

When set to one then endpoint Bulk 3 IN is RESET.

SET_ENDP – sets Toggle bit in Bulk 3 IN endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

IN4CRTL register

IN4CRTL			Set toggle and reset Bulk4 IN endpoint			X14	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint. When set to one then endpoint Bulk 4 IN is reset.

SET_ENDP – sets Toggle bit in Bulk 4 IN endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero. If SET_ENDP is reset to zero then toggle bit is not updated.

IN5CRTL register

IN5CRTL			Set toggle and reset Bulk5 IN endpoint			X15	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint. When set to one then Bulk 5 IN endpoint is reset.

SET_ENDP – sets Toggle bit in Bulk 5 IN endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero .

If SET_ENDP is reset to zero then toggle bit is not updated.

IN6CRTL register

IN6CRTL			Set toggle and reset Bulk6 IN endpoint			X16	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint. When set to one then Bulk 6 IN endpoint is reset.

SET_ENDP – sets Toggle bit in Bulk 6 IN endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

IN7CRTL register

IN7CRTL			Set toggle and reset Bulk7 IN endpoint			X17	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint. When set to one then Bulk 7 IN endpoint is reset.

SET_ENDP – sets Toggle bit in Bulk 7 IN endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

IN8CRTL register

IN8CRTL			Set toggle and reset Interrupt0 IN endpoint			X18	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint.

When set to one then endpoint Interrupt 8 IN is RESET.

SET_ENDP – sets Toggle bit in Interrupt 8 IN endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

IN9CRTL register

IN9CRTL			Set toggle and reset Interrupt9 IN endpoint			X19	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint.

When set to one then endpoint Interrupt 9 IN is RESET.

SET_ENDP – sets Toggle bit in Interrupt 9 IN endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

IN10CRTL register

IN10CRTL			Set toggle and reset Interrupt10 IN endpoint			X1A	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint.

When set to one then Interrupt 10 IN endpoint is reset.

SET_ENDP – sets Toggle bit in Interrupt 10 IN endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

IN11CRTL register

IN11CRTL			Set toggle and reset Interrupt11 IN endpoint			X1B	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	VAL_SET	SET_ENDP	RES_ENDP
-	-	-	-	-	W	W	W
0	0	0	0	0	0	0	0

RES_ENDP- Reset Endpoint.

When set to one then Interrupt 11 IN endpoint is reset.

SET_ENDP – sets Toggle bit in Interrupt 11 IN endpoint to value depending on VAL_SET.

If VAL_SET is set to one and SET_ENDP is set to one then toggle bit is set to one.

If VAL_SET is reset to zero and SET_ENDP is set to one then toggle bit is reset to zero.

If SET_ENDP is reset to zero then toggle bit is not updated.

Status Register

Status_USB		Status Register				x40	
B7	b6	b5	b4	b3	b2	b1	b0
EnumEna	HsFs	Resume	SetDataIntr	SOF	TkSetup	SuspDetect	UsbReset
W	R	R	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

The Status_USB register contains current status of the USB device. The Status_USB register is associated with six interrupt sources generated by USB. These sources are:

- USB Bus Reset,
- USB Bus Suspend,
- Token Setup Recived,
- Token SOF (Start Of Frame),
- Setup Data Recived,
- USB Bus Resume.

Occurance any of these five interrupt sources sets the corresponding flag in the Status_USB register. If the corresponding bit in the INT_Enable register is set to one and interrupt system is enabled, then interrupt output INT is set to one.

When a device detects active INT, it can read the Status_USB register to find out which source(s) caused the interrupt request.

In order to clear the flag the device must write **one** to it.

For example, if the Status_USB register has value 03h, and the device wants to clear only flag Susp_Detect, then the device must write 02h to the Status_USB register. After writing, the Status_USB register has value 01h.

Bit HsFs contains information about current speed mode.

HsFs = 0 – ALUSB works in full speed mode.

HsFs = 1 – ALUSB works in high speed mode.

Bit HsFs is not associated with interrupt system and it is read-only.

EnumEna – Enumeration Enable.

If EnumEna set to one then hardware enumerator is enabled,

If EnumEna reset to zero then hardware enumerator is disabled.

DEVICE_addr register

DEVICE_addr		Device address				x41	
b7	b6	b5	b4	b3	b2	b1	b0
DEVA7	DEVA6	DEVA5	DEVA4	DEVA3	DEVA2	DEVA1	DEVA0
W	R	R	R/W	W	W	W	/W
0	0	0	0	0	0	0	0

The Device Address Register contains USB device address. It is write only.

This register can be changed in response to Set Address request.

Test register

Test Mode register consists of the test number bits and the test enable flag. The microcontroller sets the appropriate test mode on the four last significant bits of the registers. If the test number was set properly, the microcontroller enables the test by setting the TestEN flag. The protocol management block performs all operation required by selected test. The test can be disabled by resetting the USB device only.

Table 0.1 Test Mode register

TestMode		Test Mode				X42	
B7	b6	b5	b4	B3	b2	b1	B0
D7	D6	D5	TestEN	TestNR3	TestNr2	TestNr1	TestNr0
R	R	R	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

The Enumeration Manager may also select and run these tests if it was enabled to handle the enumeration process. The microcontroller does not have access to the register if the enumeration manager is enabled i.e. if EnumEna in the status register is set to one.

Suspend register

Suspend		Suspend Register				x43	
B7	b6	B5	b4	B3	b2	b1	B0
D7	D6	D5	D4	D3	EnterSusp	ResumeEn	DriveRes
R	R	R	R	R	R/W	R/W	R/W
0	0	0	0	0	0	0	0

When microprocessor sets the EnterSusp flag in the Suspend register the USB Core drives the Suspend pin to one. The high state on the Suspend pin should turn off the external clock oscillator. These actions put the USB core into low power mode, as required by the USB specification.

The ResumeEn bit in the Suspend register should be set to one if the remote wakeup is enabled. The microcontroller or the Enumeration Manager can set this bit in response to the Set Feature/Device request.

The microcontroller can set DriveRes bit in the Suspend register to drive remote wakeup if ResumeEn bit is set to one.

Interrupt Enable Register.

The Interrupt Enable register contains enable bits associated with interrupt flags from the Status Register. The USB 2.0 interrupts are fully described in chapter 13. Please, refer to this chapter for more detailed description.

Status_USB		Interrupt Enable				x4C	
b7	b6	b5	b4	b3	b2	b1	b0
-	Disab	Res_En	SDI_En	SOF_En	TkS_En	SDet_En	UsbRe_En
R	R	R	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

New_Frm_L and New_Frm_L registers.

When the USB Host sends SOF token with 11-bit frame number and 3 bit microframe number, the ALUSB 2.0 core updates two registers: New_Frm_L and New_Frm_L with this 14-bit frame number.

Mfr2, Mfr1, Mfr0 – are located in New_Frm_L register and stand for current micro frame number.

Frm 0 to Frm 10 – are located in New_Frm_L and New_Frm_H registers and stand for current frame number.

Thereby, New_Frm_L and New_Frm_L keep current frame and micro frame number.

New_Frm_L		New Frame Low Byte				X58	
b7	b6	b5	b4	b3	b2	b1	b0
Frm4	Frm3	Frm2	Frm1	Frm0	Mfr 2	Mfr 1	Mfr0
R	R	R	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

New_Frm_H		New Frame High Byte				X59	
b7	b6	b5	b4	b3	b2	b1	b0
-	-	Frm10	Frm9	Frm8	Frm7	Frm6	Frm5
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

New_Frm_L and New_Frm_L registers are read-only.

Max Packet Size registers.

Each Isochronous Endpoint IN is associated with two registers where the lowest 13 bits of MAX PACKET SIZE are stored.

Tables below show MAX Packet Size of ISO 12 IN endpoint stored in two registers.

Register Max_Pack_size_L_n contains lowest 8 bits of MAX PACKET SIZE descriptor. N stands for endpoint number. For Isochronous endpoints n = 12 ..15.

Max_Pack_size_L_C		MAX Packet Size of endpoint ISO IN low byte				X50	
b7	b6	b5	b4	b3	b2	b1	b0
MPS7	MPS6	MPS5	MPS4	MPS3	MPS2	MPS1	MPS0
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

Register Max_Pack_size_H_n contains 5 bits of MAX PACKET SIZE descriptor.

Max_Pack_size_H_C		MAX Packet Size of endpoint ISO IN High byte				X51	
b7	b6	b5	b4	B3	b2	b1	b0
-	-	-	MPS12	MPS11	MPS10	MPS9	MPS8
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

Bits 15, 14 and 13 of MAX PACKET SIZE descriptor are not stored in described registers.

Setup registers.

Endpoint zero accepts a special setup packet, which contains an eight-byte data structure that provides host information about the control transaction. The USB core transfers the eight setup bytes into eight bytes of RAM starting from 60H address. The microcontroller can inspect the data and run processing of the request. Two interrupts are associated with control transfer. They allow easy and fast access to the eight request bytes. Please see chapter 5.1 for more details.

Counter_*WR and Counter*_RD registers.

The USB 2.0 core allows connecting external ROM memory with USB descriptors. The endpoint zero can read the descriptors directly from that memory. The data does not have to be copied into the endpoint 0 FIFO buffer. The external ROM memory should be connected to the following ports:

- DataExt(15:0) – input data bus. It should be connected with data output of the external memory.
- AddExt(4:0) – address output. It should be connected with address input of the external memory.

The endpoint zero packet size cannot be larger than 64 bytes. If a descriptor is larger, it must be divided into 64 bytes packets. The endpoint zero controller addresses the 64 bytes using AddExt(4:0) address port. The higher addresses of the external memory must be set correctly depending, which packet and descriptor is currently sent. The Counter_wr register defines number of bytes to send.

Counter_wr_low		Endpoint 0 send counter – low byte				X5A	
b7	b6	b5	b4	b3	b2	b1	b0
b7	b6	b5	b4	b3	b2	b1	b0
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

Counter_wr_high		Endpoint 0 send counter – high byte				X5B	
b7	b6	b5	b4	B3	b2	b1	b0
-	-	-	-	-	-	-	b8
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

This register must be set correctly before sending a packet from the external ROM memory. Writing a value to the register arms automatically the endpoint 0. The data are sent after receiving valid token addressed to the endpoint zero. The EMPTY flag of the endpoint zero can be checked to verify if sending the data is finished.

Counter_rd_low		Endpoint 0 receive counter – low byte				X5C	
b7	b6	b5	b4	b3	b2	b1	b0
b7	b6	b5	b4	b3	b2	b1	b0
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Counter_rd_high		Endpoint 0 receive counter – high byte				X5D	
b7	b6	b5	b4	B3	b2	b1	b0
-	-	-	-	-	-	-	b8
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

The USB 2.0 core allows also reading Counter_rd register. It contains number of received bytes by endpoint zero from USB Host. This register can be used to verify how many bytes ware received during the last transfer.