USB 2.0 (USB20HR) Device IP Core

User Guide

System Level Solutions, Inc. (USA)
14100 Murphy Avenue
San Martin, CA 95046
(408) 852 - 0067

http://www.slscorp.com

IP Core Version:	2.2
Document Version:	2.1
Document Date:	June 2008

ug_ipusb20hr_2.1

About this Guide

Introduction

This guide helps users to know about the basics of USB 2.0 (USB20HR) Device IP Core.

Table below shows the revision history of this user guide.

Version	Date	Description
2.1	June 2008	• Changed introduction, features. • Modified the document as per the new directory structure.
2.0	March 2008	• ULPI interface support added.
1.4	July 2007	• Test Mode support added for complient testing. • Timing issues solved for chirp communication.
1.3	March 2007	• Modified Endpoint0 Controller • Separate memory for enumeration data has been added to avoid overwrite problem of write function thtough Nios II.
1.2	November 2006	• Removal of Contorl Endpoint Registers and Modification of CSR Register of each endpoint.
1.1	May 2006	First Publication of the USB2.0 User Guide

How To Find Information

- The Adobe Acrobat Find feature allows you to search the contents of a PDF file. Use Ctrl + F to open the Find dialog box. Use Shift + Ctrl + N to open to the Go To Page dialog box.
- Bookmarks serve as an additional table of contents.
- Thumbnail icons, which provide miniature preview of each page, provide a link to the pages.
- Numerous links shown in Navy Blue color allow you to jump to related information.

How to Contact SLS

For the most up-to-date information about SLS products, go to the SLS worldwide website at http://www.slscorp.com. For additional information about SLS products, consult the source shown below.

Information Type	E-mail
Product literature services, SLS literature services, Non-technical customer services, Technical support.	support@slscorp.com

Typographic Conventions

The user guide uses the typographic conventions as shown below:

Visual Cue	Meaning
Bold Type with Initial Capital letters	All headings and Sub headings Titles in a document are displayed in bold type with initial capital letters; Example: **Core Architecture, Operation.**
Bold Type with Italic Letters	All Definitions, Figure and Table Headings are displayed in Italics. Examples: *Table 4-1. Control and Status Register Details*, *Figure 2-1. USB 20HR IP Core Architecture*
Italic type	Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: <*USB20HR Installation Path*>.
1., 2.	Numbered steps are used in a list of items, when the sequence of items is important. such as steps listed in procedure.
•	Bullets are used in a list of items when the sequence of items is not important.
☞	The hand points to special information that requires special attention
⚠ CAUTION	The caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process.

Visual Cue	Meaning
⚠ WARNING	The warning indicates information that should be read prior to starting or continuing the procedure or processes.
👣	The feet direct you to more information on a particular topic.

Contents

1. Introduction

The USB20HR Device IP Core is a RAM based USB 2.0 compliant device core with 32-bit Avalon interface and UTMI & ULPI interface support. The core supports both High Speed (480 Mbps) and Full Speed (12 Mbps) functionality. The core supports three preconfigured endpoints Control, Bulk IN, and Bulk OUT. It can be configurable for up to 15 IN/OUT endpoints on customer request. Each configurable endpoint has an endpoint controller that supports interrupt, bulk, and isochronous transfers.

The core is an RTL design in Verilog that implements an USB device controller on an ASIC or FPGA. The core has been optimized for popular FPGA devices and its functionality has been verified on the real hardware. It is provided as Altera Quartus II Megafunction (Altera SOPC Builder ready component) and integrates easily into any SOPC Builder generated system using Nios® II Avalon bus.

This user guide will provide you with some basic technical details of USB20HR device core acting as an Avalon slave in Altera’s SOPC builder. SOPC builder is a software tool that allows for the creation of a Nios II system module or a more general multi master **S**ystem **O**n A **P**rogrammable **C**hip module. A complete Nios II system module contains a Nios II (soft core 32 bit RISC) processor and its associated system peripherals.

☞ For development kits lacking USB interface, SLS has two types of Snap On Boards available, one is for UTMI interface and the other is for ULPI interface along with a USB 2.0 compliant device IP core integrated into SOPC builder. The Snap On Board snaps on to the Altera Standard Santa Cruz header, therefore, now your Nios development kit is with USB 2.0 interface without any extra effort. So now using SOPC builder one can build a system with USB interface in just a few minutes.

For the UTMI physical interface of the USB 2.0 Snap On Board refer to USB 2.0 - UTMI Snap On Board Reference Manual.

For the ULPI physical interface of the USB 2.0 Snap On Board refer to

USB 2.0 On-The-Go (ULPI) Snap On Board Reference Manual.

Features

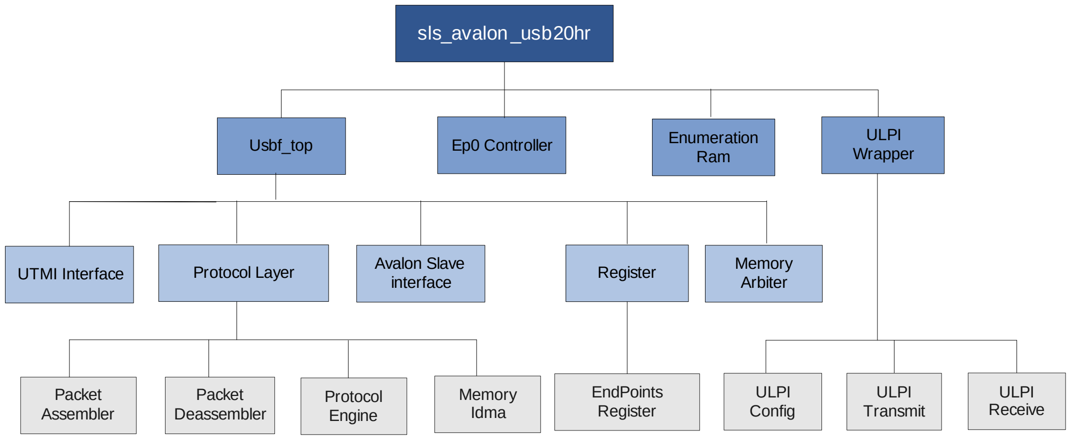
Following are the USB20HR Device IP core features:

- Certified USB 2.0 Device IP Core
- Supports both Full Speed (12 Mbps) and High Speed (480 Mbps)
- Supported Interfaces
  - UTMI (Cypress CY7C68000 PHY)
  - ULPI (NXP ISP 1504 PHY)
- Preconfigured for 3 endpoints
  - Control
  - Bulk IN
  - Bulk OUT
- Configurable for up to 15 IN/OUT endpoints including Isochronous and Interrupt on customer request at additional cost.
- Performs all USB enumeration process which is controlled with software command (usb connect control) and CRC through Hardware.
- Avalon Bus compliant
- Implemented in Verilog RTL
- Optimized LE count

Design Hierarchy

The design hierarchy & corresponding Verilog files are depicted in Figure 1-1.

Figure 1-1. Design Hierarchy of the USB20HR IP Core



Core Resources

Table 1-1 shows the LE usage of the Core.

Table 1-1. Core LE Usage Summary

Type	Endpoints				LEs	Memory Bits
	Control	Bulk	Iso	Int		
SLS SOPC UTMI Interface	1	2	-	-	2450	18432
SLS SOPC ULPI Interface	1	2	-	-	2520	18432
Extra Endpoints	-	1	-	-	130	0
Extra Endpoints	-	-	1	-	130	0
Extra Endpoints	-	-	-	1	130	0

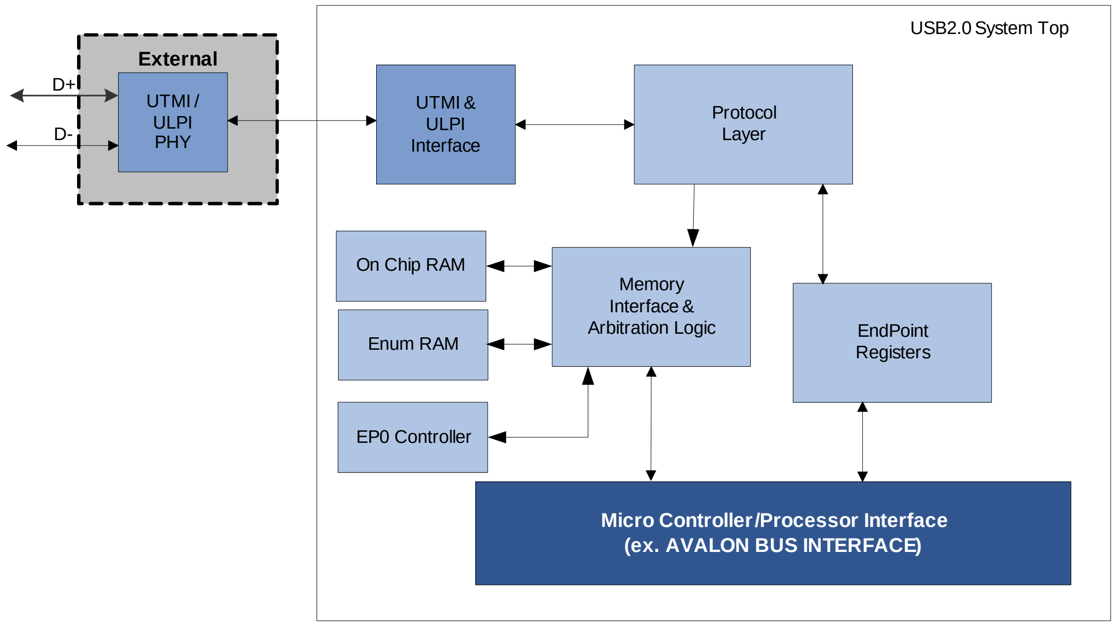
Further Information

For information about USB20HR IP Core installation directory structure and its content, licensing, component implementation and its support, refer **readme.text** located at <*USB20HR Installation Path*>\ **usb20hr_**<*licensetype*>. <*USB20HR Installation Path*> is the installation directory. The default installation directory is c:\altera\<version #>\ip\sls. <*licensetype*> is the ip core license type. The licensetype can be **ocp_eval** or **oc_full**.

2. Core Architecture

The USB 2.0 Device communicates through two differential lines (D+ & D-) that connect to a transceiver. The transceiver or physical interface, in turn, connects to the core via interface signals. Since the core supports all the transfer types--Control, Bulk, Isochronous and Interrupt, each transfer type retains a separate pipe for OUT and IN operation. Figure 2-1. illustrates overall architecture of the USB20HR IP core.

Figure 2-1. USB20HR Device IP Core Architecture



Each of the blocks is described in detail below:

The micro Controller/Processor interface provides a bridge between Host Interfaces (ex Nios Processor) and internal data memory and control registers.

UTMI / ULPI PHY

The UTMI or ULPI PHY chip is the external chip that provides a link between USB2.0 IP and the physical USB data lines D+ & D-. The maximum clock from the PHY is 60Mhz in 8 bit mode. All the blocks run from the clock provided by the UTMI or ULPI PHY for synchronization.

UTMI & ULPI Interface

The UTMI (USB2.0 Transceiver Macrocell Interface) or ULPI block connects through the external PHY. It controls speed negotiation as well as handles all functions related to the line signals. It also handles the ULPI PHY chip configuration for device operation.

Protocol Layer

This block Deassembles/Assembles the packet, handles all the standard USB 2.0 protocol handshakes and control correspondence.

Memory Interface & Arbitration Logic

This block has the arbitration logic that arbitrates between the USB core and host interface for memory access.

EndPoint Registers

This block has control and status registers for each endpoints. One can configure any specific endpoint through these registers via Avalon interface.

On chip RAM

This memory is used to store transmitted as well as received data for End-Points. Size of this memory is 2Kbyte. Micro controller and device interfaces with this memory through memory arbitration logic.

EP0 Controller

All the EndPoint 0 transfers are taken care by this module. It accommodates a state machine to decode different types of SETUP packets and gives a response. USB standard requests as well as mass storage class specific requests are supported.

Enum_RAM

This module contains all the information related to Device Descriptor, Configuration Descriptor, Interface Descriptor, EndPoint Descriptor and String Descriptor.

String descriptors , Vendor ID & Product ID of device, class, subclass and protocol of device and interface can be edited through usb20hr_enum_data_editor.exe utility.

The usb20hr_enum_data_editor.exe and readme.txt are given in *<USB20HR Installation path>*\**usb20hr_***<licensetype>*\***software\\ utilities\\enumdataeditor***.

WARNING

Don't edit User_defines.v or Enum_ram.hex file manually.

Micro Controller/Processor Interface

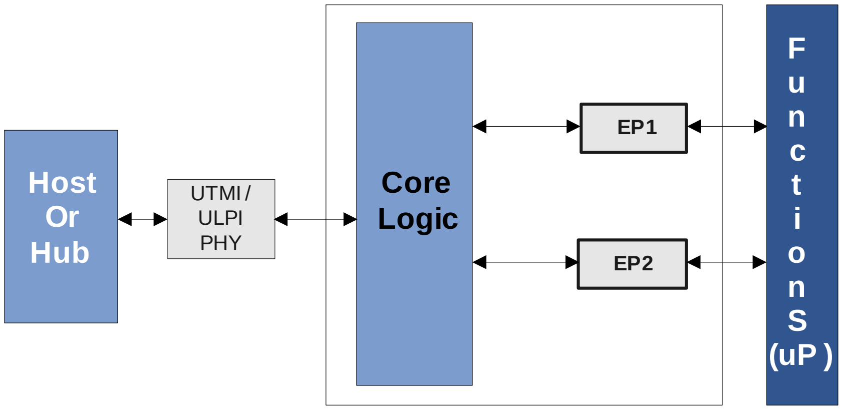
This block provides a consistent core interface between the internal functions of the core and the function-specific host or micro controller.

3. Operation

This core has been designed to compatible with avalon specific interface. The Figure 3-1. below shows how the core connects to the function micro controller and HOST.

IP core functionality is controlled by function controller as shown in Figure 3-1. through endpoint registers.

Figure 3-1. Operational Block Diagram



The USB core uses onchip memory (512*32) which is used as a temporary data storage. No software intervention is needed between endpoint access. Double buffer mechanism is used for reducing the latency requirement on the software, and increasing USB throughput.

EndPoints

This USB core supports 3 endpoints (Control , Bulk In , Bulk Out). The function controller must set up the endpoints (except for Control EndPoint) by writing to the endpoint register: EPn_CSR. EPn_INT, EPn_BUFx .

Buffer Pointers

The buffer pointers point to the input/output data structure in memory. If all buffers are not allocated, the core will respond with NAK acknowledge to the USB host.

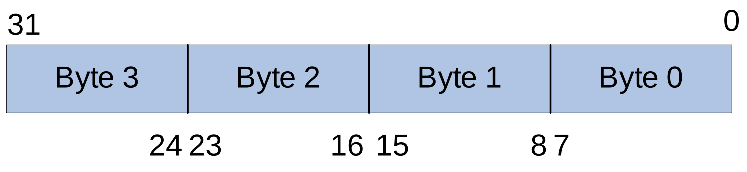
This core supports a double buffering feature which reduces the latency requirements on the functions micro controller and driver software. Data is being retrieved/filled from/to the buffers in a round robin fashion. When data is sent to/from an endpoint, first buffer0 is used. When the first buffer is empty/full, the function controller, may be notified via an interrupt. The function controller can refill/empty buffer0 now. The core will now use buffer 1 for the next operation. When the second buffer is full/empty, the function controller is interrupted, and the core will use buffer0 again, and so on. A buffer that has the used bit set cause the core for replying with NAK/NYET acknowledgments to the host.

The Buffer’s Used bits indicate when a buffer has been used (this information is also provided in the Interrupt Source Register). The function controller must clear these bits after it has emptied/refilled the buffer.

Data Organization

Since the buffer memory is 32 bits wide and USB defines all transactions on byte boundaries it is important to understand the relationship of data in the buffer to actual USB byte sequence. This USB core supports Little Endian byte ordering

Figure 3-2. Data Organization



The buffer pointer always point to byte 0. The USB core always fetches four bytes from the buffer memory. The actual final byte that is transmitted in the Nth transaction depends on the Buffer Size. The MaxPacketSize must always be a multiple of 4 bytes.

Interrupts

The USB core provides interrupt outputs. The output is fully programmable. The usage of the interrupt is up to the system into which the USB core is incorporated.

The interrupt mechanism in the USB core consists of a two level hierarchy:

1. **The main interrupt source register** (INT_SRC) indicates interrupts that are endpoint independent. These interrupts indicate overall events that have either global meaning for all endpoints or can not be associated with an endpoint because of an error condition.
2. **The endpoint interrupt source register** indicates events that are specific to an endpoint.

Timing

The interrupt output are asserted when the condition that is enabled in the interrupt mask occurs. They remain asserted until the main interrupt register is read.

Software Interaction

An interrupt handler should first read the main interrupt source register (INT_SRC) to determine the source of an interrupt. It must remember the value that was read until it is done processing of each interrupt source. If any of the bits 2 or 1 are set, the interrupt handler should also read the appropriate endpoint interrupt register to determine endpoint specific events. Multiple interrupt sources may be indicated at any given time. Software should be prepared to handle every interrupt source it cares about.

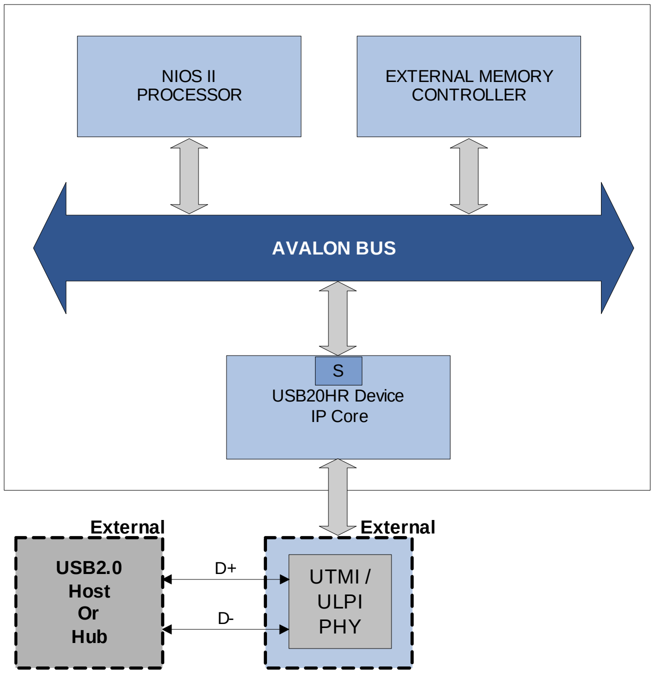
A care must be taken not to lose interrupt sources, as the main interrupt source register is cleared after a read.

4. Avalon Bus Connectivity

The Avalon bus is a simple bus architecture designed for connecting on-chip processors and peripherals together into a system–on–a–programmable chip (SOPC). The Avalon bus is an interface that specifies the port connections between master and slave components, and specifies the timing by which these components communicate.

Avalon Bus Connectivity shows how the system is connected with Avalon bus, Which one is Master, which one is slave and what are the devices connected in a system. In this system, USB20HR Device IP Core component acts as a slave. It is interrupt based component. Avalon slave interface is used to access the USB2.0 data. Figure 4-1. shows Avalon Bus Connectivity.

Figure 4-1. Avalon Bus Connectivity



5. Core Registers

This section describes all the registers inside the USB20HR Core. The **Register** field describes the name of the register. The **Offset** field describes the offset of register in USB IP. The **Access** field describes the type of access to the register that is read or write. Description field describes the Type and Function of register.Table 5-1 shows the register details.

Table 5-1. Control and Status Register Details

Sr. No.	Register	Offset	Width	Access	Description
1	MAIN_CSR	0x00	8	RO	Control/Status Register
2	INT_MSK	0x08	16	RW	Interrupt Mask for endpoint independent sources
3	INT_SRC	0x0c	32	ROC	Interrupt Source Register
4	FRM_NAT	0x10	32	RO	Frame Number and Time
5	D_CNCT	0x1fc	8	RW	Device Connect / Disconnect register
EndPoint Registers					
6	EP1_CSR	0x50	32	RW	EndPoint 1: CSR
7	EP1_INT	0x54	32	ROC	EndPoint 1: Interrupt Register
8	EP1_BUFFER0	0x58	32	RW	EndPoint 1: Buffer Register 0
9	EP1_BUFFER1	0x5c	32	RW	EndPoint 1: Buffer Register 1
10	EP2_CSR	0x60	32	RW	EndPoint 2: CSR
11	EP2_INT	0x64	32	ROC	EndPoint 2: Interrupt Register
12	EP2_BUFFER0	0x68	32	RW	EndPoint 2: Buffer Register 0
13	EP2_BUFFER1	0x6c	32	RW	EndPoint 2: Buffer Register 1

☞ Endpoint buffer memory (2K size) is starting from offset address 0X20000.

Control Status Register (MAIN_CSR)

This is the main configuration and status register of the core.

Table 5-2. CSR Register Details

Bit	Access	Description
7:6	RO	Reserved
5	RO	Reserved
4:3	RO	UTMI Line State
2	RO	Interface Status 1 = Attached
1	RO	Interface Speed 1 = High Speed; 0 = Full Speed Mode
0	RO	1 = Suspend Mode

This register provides current status of the device. This is RO register.

Value after Reset : **02h**

Interrupt Mask Register (INT_MSK)

The interrupt mask register defines the functionality of interrupt output with regards to events without associated endpoints.

A bit set to a logical 1 enables the generation of the interrupt for that source, a zero disables the generation of an interrupt. Table 5-3 shows the list of interrupts controlled by this register.

Table 5-3. Interrupt Mask Register Details

Bit	Access	Description
15:10	RW	Reserved
9	RW	Interrupt Enable : Speed Negotiation Done
8	RW	Interrupt Enable: Received a USB reset
7	RW	Interrupt Enable: Received a UTMI Rx Error
6	RW	Interrupt Enable:Assert interrupt when Detached
5	RW	Interrupt Enable: Device Attach

4	RW	Interrupt Enable : Leave Suspend Mode (Resume)
3	RW	Interrupt Enable: Enter Suspend Mode
2	RW	Interrupt Enable: No such endpoint
1	RW	Interrupt Enable: PID Error (PID check sum error)
0	RW	Interrupt Enable: Bad Token (CRC 5 error)

Value after Reset : **0000h**

Interrupt Source Register (INT_SRC)

This register identifies the source of an interrupt. Whenever the function controller receives an interrupt, the interrupt handler must read this register to determine the source and cause of the interrupt. Some of the bits in this register will be created after a read. The software interrupt handler must make sure it keeps whetever infomration is required to handle the interrupt.

Table 5-4. Interrupt Source Register Details

Bit	Access	Description
31:30	RO	Reserved
29	ROC	Speed Negotiation Done
28	ROC	USB Reset
27	ROC	UTMI Rx Error
26	ROC	Detached
25	ROC	Attached
24	ROC	Resume
23	ROC	Suspend
22	ROC	No Such Endpoint
21	ROC	PID Error (PID check sum error)
20	ROC	Bad Token (CRC 5 error)
19:3	RO	Reserved
2	RO	Endpoint 2 caused an interrupt
1	RO	Endpoint 1 caused an interrupt
0	RO	Reserved

Value after reset : **00000000h**

Frame Number and Time Register (FRM_NAT)

This register tracks the frame number as received from the SOF token, and the frame time.

Table 5-5. Frame Number and Time Register Details

Bit	Access	Description
31:28	RO	Number of frames with the same frame number (this field may be used to determine current microframe)
27	RO	Reserved
26:16	RO	Frame number as received from SOF token
15:12	RO	Reserved
11:0	RO	Time since last SOF in 0.5 uS resolution

Value after reset : **00000000h**

Device Connect / Disconnect register (D_CNCT)

This register is used to control device connect/ disconnect operation through processor. This is 8 bit register. For connection of the device with the host 0xFF should be written to the register. and to disconnect the device from host 0x00 should be written to the register.

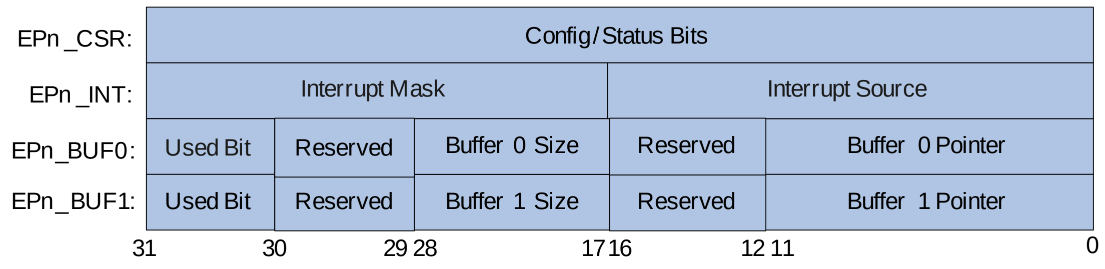
Table 5-6. Device Connect / Disconnect Register Details

Bit	Access	Description
7:1	RO	Reserved
0	RW	Connect/Disconnect Status 1 = Device Connected 0 = Device Disconnected

Endpoint Registers

Each endpoints has 4 registers associated with it. These registers have exactly the same definition for each endpoint.

Figure 5-1. Endpoint Register



Endpoint CSR Register (EP_CSR)

The configuation and status bits specify the operation mode of the endpoint and report any specific endpoint status back to the contoller.

Table 5-7. Endpoint CSR Register Details

Bit	Access	Description
31:30	RW	UC_BSEL Buffer Select This bit must be initialized to zero (first Buffer 0 is used). The USB core toggle these bits, in order to know which buffer to use for the next transaction. 00 : Buffer 0 01 : Buffer 1 1x :Reserved
29:28	RW	UC_DPD These two bits are used by the USB core to keep track of the data PID for high speed endpoints and for DATA0/DATA1 toggling.
27:26	RW	EP_TYPE Endpoint Type 00 : Control Endpoint 01 :IN EndPoint 10 : OUT Endpoint 11 : Reserved

25:24	RW	TR_TYPE Transfer Type 00 : Interrupt 01 : Isochronous 10 : Bulk 11 : Reserved
23:22	RW	EP_DIS Temporarily Disable EP 00 : Noraml 01 : Force the core to ignore transfer over this EP 10 : Set EP HALT 11 : Reserved
21:18	RO	EP_number
17	RW	LRG_OK 1 - Accept data packets of more than MAX_PL_SZ (RX only) 0 - Ignore data packets of more than MAX_PL_SZ (RX only)
16	RW	SML_OK 1 - Accept data packets of less than MAX_PL_SZ (RX only) 0 - Ignore data packets of less than MAX_PL_SZ (RX only)
15	RO	Reserved
14	RO	Reserved
13	RO	Reserved
12:11	RW	TF_FR Number of transaction per microframe (HS mode only)
10:0	RW	MAX_PL_SZ Maximum payload size (MaxPacketSize) in bytes

Value after Reset :

EP1 : **00040000h**

EP2 : **00080000h**

Endpoint Interrupt Mask/Source Register (EP_IMS)

The interrupt register for each endpoint has mask bits for interrupt output and bits that indicate the interrupt source when an interrupt has been received.

Table 5-8. Endpoint Interrupt Mask / Source Register Details

Bit	Access	Description
31:29	RO	Reserved
28	RW	Interrupt Enable : PID Sequence Error
27	RW	Interrupt Enable : Buffer Full/Empty
26	RW	Interrupt Enable : Unsupported PID
25	RW	Interrupt Enable : Bad packet (CRC 16 error)
24	RW	Interrupt Enable : Time Out (Waiting for ACK or DATA packet)
23:06	RO	Reserved
5	ROC	Interrupt Status :PID Sequence Error
4	ROC	Interrupt Status :Buffer 1 Full/Empty
3	ROC	Interrupt Status :Buffer 0 Full/Empty
2	ROC	Interrupt Status :Unsupported PID
1	ROC	Interrupt Status :Bad Packet (CRC 16 error)
0	ROC	Interrupt Status :Time Out (waiting for ACK or DATA packet)

Value after Reset : **00000000h**

Endpoint Buffer Register (EP_BUF)

The endpoint buffer register holds the buffer pointer for each endpoint. Each endpoint has two buufer registers, thus allowing double buffering. Each buffer register has exactly the same definition and functionality.

Table 5-9. Endpoint Buffer Register Details

Bit	Access	Description

31	RW	USED This bit is set by the USB core after it has used this buffer. The function controller must clear this bit again after it has emptied/refill this buffer. This bit must be initialized to 0.
30:29	RO	Reserved
28:17	RW	BUF_SIZE Buffer size (number of bytes in the buffer).
16:12	RO	Reserved
11:0	RW	BUF_PTR Buffer pointer (byte address of the buffer)

Value after Reset : **FFFFFFFFh**

6. Core IOs

Table 6-1 list all the IOs of the USB20HR IP core.

Table 6-1. Core IOs

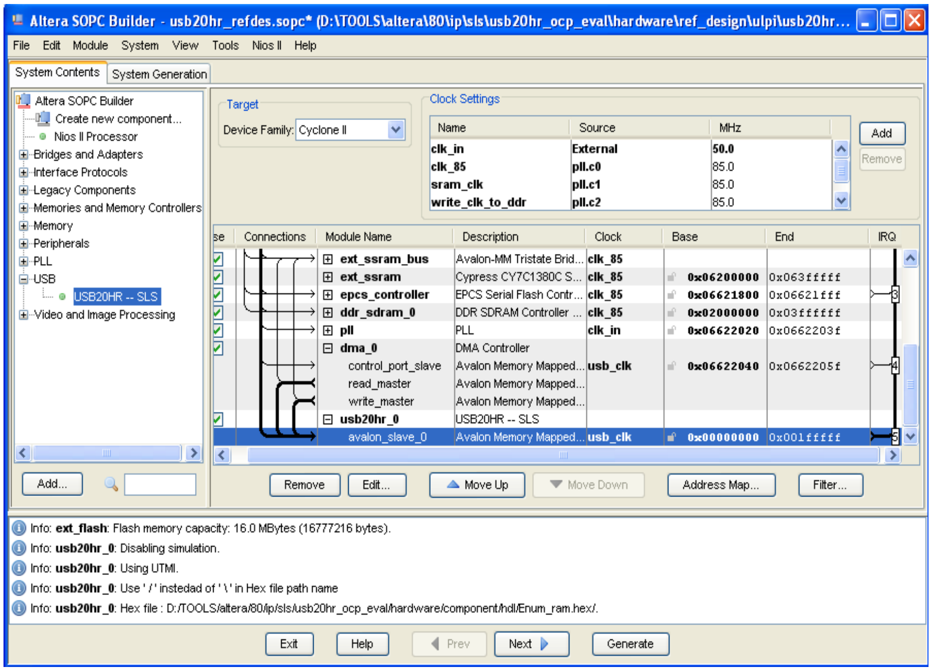
Interface Type	Name	Width	Direction	Description
UTMI	Data	8	Bi-Di	Bi-Direction Data Line for ULPI and UTMI interface.
	RxActive	1	In	Receive Active
	RxValid	1	In	Receive Data Valid.
	RxError	1	In	Receive Error.
	TxValid	1	Out	Transmit Valid.
	TxReady	1	In	Transmit Ready.
	XcvSelect	1	Out	1: Full Speed transceiver selected. 0: High Speed transceiver selected.
	TermSel	1	Out	1: Full Speed termination enabled. 0: High Speed termination enabled.
	SuspendM	1	Out	Not Supported.
	LineState	2	In	Line State.
	OpMode	2	Out	Operation Mode Select.
	usb_vbus	1	In	Should be connected to the Vcc pin of the USB connector. Used to detect if the core is connected to an USB interface.
ULPI	Data	8	Bi-Di	Bi-Direction Data Line for ULPI and UTMI interface.
	Dir	1	In	Direction line of ULPI interface.
	Nxt	1	In	Next line of ULPI interface.
	Stp	1	Out	Stop line for ULPI interface

7. Using USB20HR IP in SOPC Builder

To use USB20HR Device IP core in SOPC Builder follow the steps below:

1. Running **usb20hr_<*licensetype*>_v<*version #*>.exe** will automatically place the USB20HR component into SOPC Builder GUI. You can see that USB20HR component is now added in the final view of the SOPC builder system. See Figure 7-1.

Figure 7-1. USB20HR Device IP Core in SOPC Builder


Altera SOPC Builder - usb20hr_refdes.sopc* (D:\TOOLS\altera\80\ip\sls\usb20hr_ocp_eval\hardware\ref_design\ulpi\usb20hr...

Device Family: Cyclone II

Name	Source	MHz
clk_in	External	50.0
clk_85	pll.c0	85.0
sram_clk	pll.c1	85.0
write_clk_to_ddr	pll.c2	85.0

se	Connections	Module Name	Description	Clock	Base	End	IRQ
✓		ext_ssram_bus	Avalon-MM Tristate Brid...	clk_85			
✓		ext_ssram	Cypress CY7C1380C S...	clk_85	0x06200000	0x063fffff	
✓		epcs_controller	EPCS Serial Flash Contr...	clk_85	0x06621800	0x06621fff	3
✓		ddr_sdram_0	DDR SDRAM Controller ...	clk_85	0x02000000	0x03ffffff	
✓		pll	PLL	clk_in	0x06622020	0x0662203f	
✓		dma_0	DMA Controller				
		control_port_slave	Avalon Memory Mapped...	usb_clk	0x06622040	0x0662205f	4
		read_master	Avalon Memory Mapped...				
		write_master	Avalon Memory Mapped...				
✓		usb20hr_0	USB20HR -- SLS				
		avalon_slave_0	Avalon Memory Mapped...	usb_clk	0x00000000	0x001fffff	5

```
Info: ext_flash: Flash memory capacity: 16.0 MBytes (16777216 bytes).
Info: usb20hr_0: Disabling simulation.
Info: usb20hr_0: Using UTMI.
Info: usb20hr_0: Use ' / ' instedad of ' \ ' in Hex file path name
Info: usb20hr_0: Hex file : D:/TOOLS/altera/80/ip/sls/usb20hr_ocp_eval/hardware/component/hdl/Enum_ram.hex/.
```

2. Since the core is a slave, there must be an Avalon master. Usually, this master is the Nios processor. Using the SOPC builder tool, include the Nios processor (must have the Nios development kit). Then add any other peripherals required for the final design and assign the base address and irqs.

For further reference refer Core Registers Section

3. Run the provided C File, by uploading to the processor via the Nios II IDE or the ***.elf** through Cygwin bash shell and *nios-run* command for testing the behavior of USB20HR IP Core.

   ☞ The core will work on the basis of interrupting the Nios processor. That means whatever operation, either read or write, to any of the specified device would be driven by the Nios processor since it is the master. The USB core will deliver commands to the Nios processor through interrupts. The SLS supplied Sample C file (Look under **"..\\usb20hr_<***licensetype***>\software\example\ niosII\template** runs on top of the Nios processor allowing it to read the specified registers and decode opcodes and commands.