USB 2.0 Host Controller IP Core

Based on GRLIB, May 2007

Jan Andersson, Jonas Ekergarn

Table of contents

1 Introduction

1.1 Overview

Complex digital systems require high-performance external interfaces. Gaisler Research provides an advanced USB 2.0 Host Controller IP cores that interconnects the on-chip AMBA bus and external USB bus. The solution is a hardware validated, drop-in IP core that simplifies the design of System-On-a-Chip solutions.

The USB 2.0 Host Controller core (GRUSBHC) provides a link between the AMBA on-chip bus and the Universal Serial Bus (USB). The host controller supports High-, Full- and Low-Speed USB traffic. USB 2.0 High-Speed functionality is supplied by an enhanced host controller implementing the Enhanced Host Controller Interface (EHCI). Full- and Low-Speed functionality (USB 2.0 and USB 1.1) is supplied by one or more companion controllers implementing the Universal Host Controller Interface (UHCI). The Port Router supplies the dynamic connection between the host controllers and the USB transceivers.

The core can handle up to 15 downstream ports, where each port can handle all three USB speeds. Port routing within the core is highly configurable. The designer has choices ranging from handling Full/Low-Speed traffic with one companion controller per port, to having one companion controller handle all ports. The modularity of the core enables the designer to configure High- or Full/Low-Speed only products. Both controller types have support for big and little endian systems, with the option to adjust the register interfaces' byte order to fit the target platform.

The core supports UTMI+ 16-bit transceivers at 30 MHz and 8-bit transceivers at 60 MHz, and ULPI 8-bit transceivers at 60 MHz.

1.2 Features

Compatible with USB Specification Rev. 2.0 and Rev. 1.1

Supports all USB transfer types

Supports High- (480 Mbit/s), Full- (12 Mbit/s) and Low-Speed (1.5 Mbit/s) traffic

Enhanced Host Controller compatible with EHCI rev. 1.0

- Supports Asynchronous Park Mode
- Implements a NAK counter to limit unnecessary memory accesses
- Descriptor and data prefetch prevents buffer overrun and underrun errors
- AMBA AHB 32-bit master interface, APB 32-bit slave interface

Universal Host Controller compatible with UHCI rev. 1.1

- Extended to report over current conditions
- AMBA AHB master/slave interface
- Flexible configuration of port routing and number of companion controllers

Supports up to 15 downstream ports with up to 127 devices each

Easily adaptable to both big and little endian systems

Supports UTMI+ and ULPI transceivers

AMBA DMA interface with configurable burst length

AMBA Specification Rev. 2.0 compatible

1.3 System-on-a-Chip design

Gaisler Research's experience as developer of complex and high-performant System-On-a-Chip solutions ensures that their IP cores provide optimal performance and easy integration. The Gaisler Research IP cores have been designed to provide solutions that enable designers to meet their system performance goals whilst minimizing the cost of ownership associated with the acquisition and deployment of IP cores. The key benefits of the IP cores are:

- Configurability:

  Each IP core provides a large set of configurable parameters that are set at design time and drive the optimization of the design towards the needs of the designer's.

- Portability:

  The IP cores are developed in a tool independent manner, enabling portability between different tool sets and technologies.

- Plug & play:

  The IP cores are fully integrated in the AMBA AHB and APB bus fabric. The IP cores provide additional sideband signals to the AMBA bus fabric, allows plug & play information to be distributed amongst the cores. The information can then be used for probing a system and determining what drivers need to be loaded, or to provide dedicated support for software debug tools

- Compatibility:

  The IP cores have been verified for compatibility with the AMBA standard and have been validated to operate with the LEON3 32-bit SPARC processor.

1.4 Example application

The USB Host Controller IP core has been designed to fit into an architecture from which a large variety of applications can be derived. The architecture is centered on the AMBA Advanced High-speed Bus (AHB), to which the USB Host Controller IP core and other high-bandwidth units are connected. Low-bandwidth units connected to the AMBA Advanced Peripheral Bus (APB) which is accessed through an AHB to APB bridge.

An example of such an architectures is shown in figure 1.

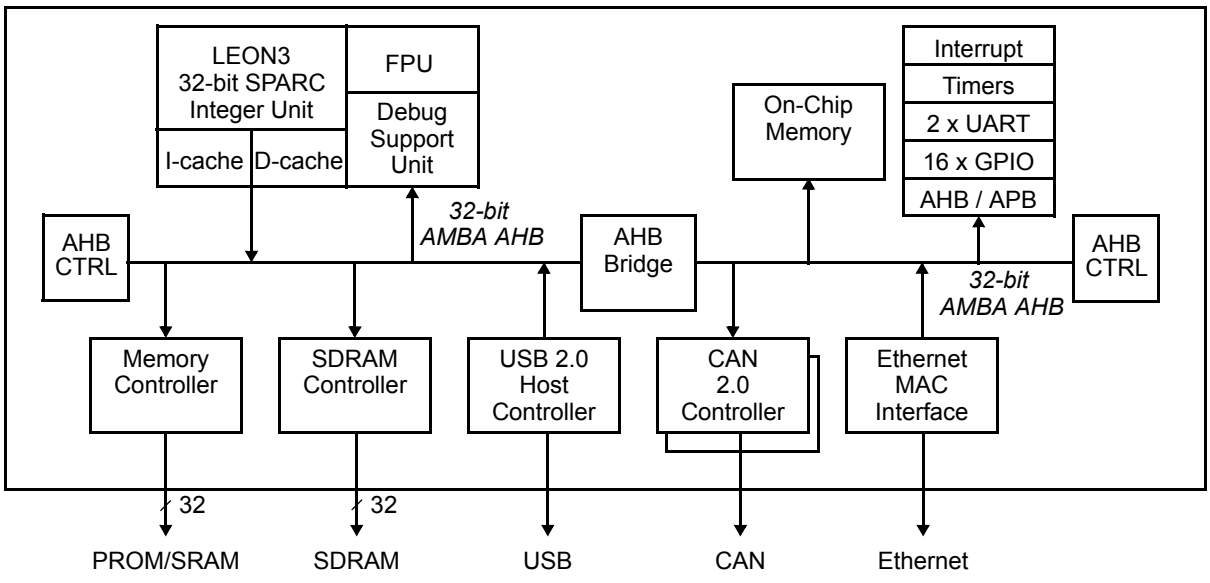

Figure 1. Architectural block diagram of a typical system using a USB Host Controller IP core

1.5 Implementation characteristics

The USB Host Controller IP core is inherently portable and can be implemented on most FPGA and ASIC technologies. Both area and timing of the core depends strongly on the selected configuration, target technology and the used synthesis tool.

Function	**Number of ports**	**Xilinx Virtex-4 ***			**ASIC *****
		LUT	**RAM**	**MHz**	**Gates**
Universal host controller (USB Full- and Low-Speed traffic only)	1	3 000	1	80	25 000
Enhanced host controller (USB High-Speed traffic only)	1	9 000	3	80	70 000
Enhanced host controller with one universal companion controller ** (supports all USB traffic speeds)	1	12 000	3	80	95 000
	2	13 000	4	80	105 000
	4	15 000	4	80	125 000
	8	20 000	4	80	155 000
	12	24 000	4	80	195 000
	15	27 000	4	80	215 000

* (LUTs / RAM blocks / AHB system clock MHz)

** The single companion controller handles all ports.

The core can also be configured to include multiple companion controllers.

*** ASIC gate count does not include required on-chip memory.

1.6 Licensing

The table below lists the provided IP core. The license column indicates if a core is available under GNU GPL and/or under a commercial license.

Note: The open-source version of GRLIB includes only cores marked with GPL or LGPL.

The core is available in VHDL RTL source code or netlist.

Function	**Name**	**License**
USB 2.0 Host Controller	GRUSBHC	COM

2 GRUSBHC - USB 2.0 Host Controller

2.1 Overview

The Gaisler Research USB 2.0 Host Controller provides a link between the AMBA AHB bus and the Universal Serial Bus. The host controller supports High-, Full-, and Low-Speed USB traffic. USB 2.0 High-Speed functionality is supplied by an enhanced host controller implementing the Enhanced Host Controller Interface. Full- and Low-Speed traffic is handled by up to 15 (USB 1.1) companion controllers implementing the Universal Host Controller Interface. Each controller has its own AMBA AHB master interface. Configuration and control of the enhanced host controller is done via the AMBA APB bus. Companion controller registers are accessed via an AMBA AHB slave interface. Figure 2 shows a USB 2.0 host system and the organization of the controller types. Figure 3 shows an example with both host controller types present.

The controller supports both UMTI+ and ULPI transceivers and can handle up to 15 ports.

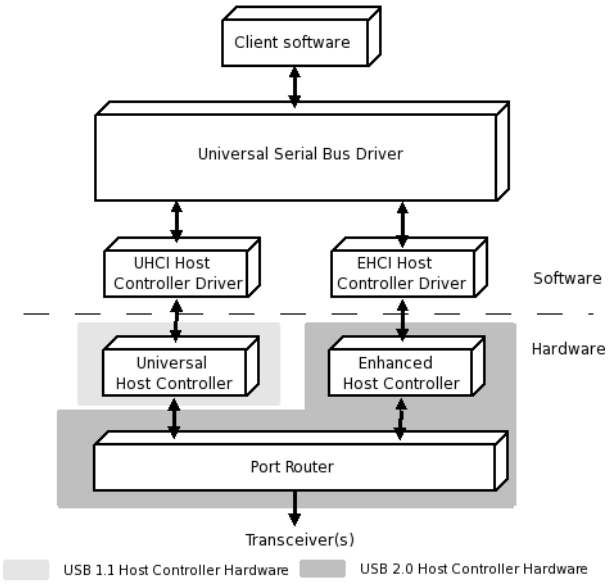

Figure 2. Block diagram of USB 2.0 host system

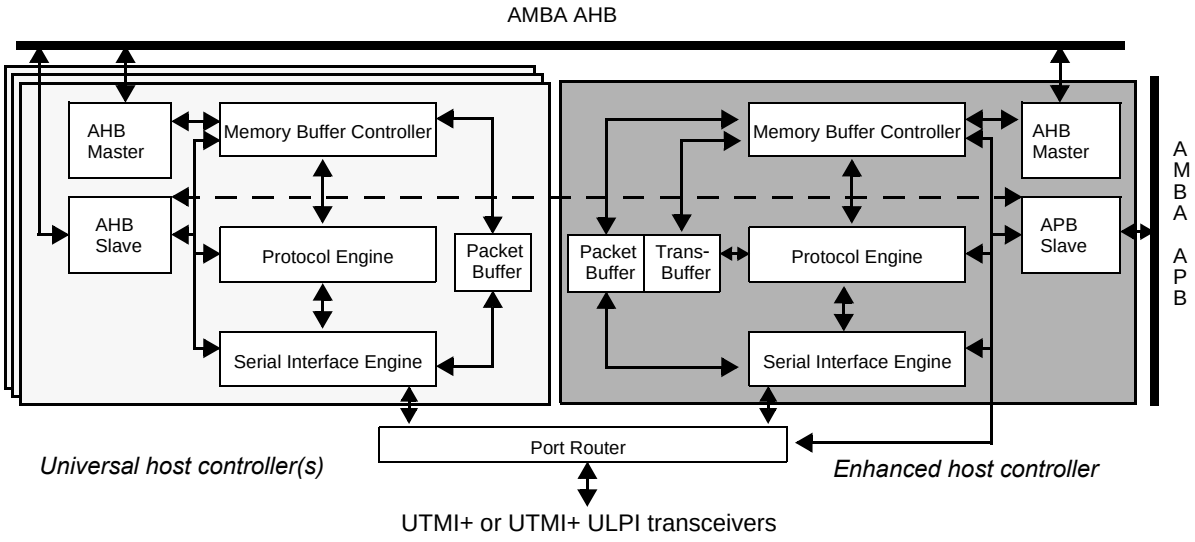

Figure 3. Block diagram of both host controller types

2.2 Operation

2.2.1 System overview

Depending on the core's configuration it may contain both controller types, one enhanced host controller, or up to 15 standalone universal host controllers. If both controller types are present, each universal host controller acts as a companion controller to the enhanced host controller.

The enhanced host controller complies with the Enhanced Host Controller Interface with the exception of the optional Light Host Controller Reset, which is not implemented.

The universal host controller complies with the Universal Host Controller Interface, with exceptions. The HCHalted field in the USB Command register is implemented as Read Only instead of Read/Write Clear. The Port Status/Control registers have been extended with Over Current and Over Current Change fields. Changes to both registers have been done in accordance with contemporary implementations of the interface. Both changes match the description of corresponding bits in the EHCI specification.

2.2.2 Protocol support

The enhanced host controller has full support for High-Speed traffic as defined in the USB Specification, revision 2.0. In addition Asynchronous Park Mode is supported, and the controller has a NAK counter.

The universal host controller supports Full- and Low-Speed traffic.

2.2.3 Descriptor and data buffering

The enhanced host controller prefetches one frame of isochronous descriptors. All payload data for a transaction is fetched before the transaction is executed. The enhanced host controller has a 2048 byte buffer for descriptors and a 2048 byte buffer for payload data, which can hold data for two transactions.

The universal host controller does not prefetch descriptors, depending on controller configuration a transaction on the bus may be initiated before all payload data has been fetched from memory. Each universal host controller has a 1024 byte buffer for payload data. A transfer descriptor in UHCI may describe a transaction that has a payload of 1280 bytes. The USB specification limits the maximum allowed data payload to 1023 bytes and the controller will not transfer a larger payload than 1023 bytes. If a descriptor has a, legal, larger payload than 1023 bytes, the controller will only attempt to transfer the first 1023 bytes before the transaction is marked as completed.

In the event that the host controller has just one port, the universal host controller and the enhanced host controller will share the data payload buffer. Thus only two 2048 byte buffers are required.

2.2.4 Clocking

The core has two clock domains; the system clock domain and the USB clock domain which includes the transceiver. All signals that cross a clock domain boundary are synchronized to prevent meta-stability.

2.2.5 Endianness

The core always accesses the least significant byte of a data payload at offset zero. Depending on the core's configuration, registers may be big endian, little endian, or byte swapped little endian.

2.2.6 RAM test facilities

The VHDL generic ramtest enables RAM testing. If RAM testing is enabled, controller's internal buffers are mapped into the register space. The universal host controller maps the packet buffer at off-

set 0x400 - 0x7FF. An enhanced host controller will map the packet buffer at offset 0x1000 - 0x17FFF and the transaction buffer at 0x1800 - 0x1FFF. The buffers can both be read and written, while both controller types are idle to ensure correct operation. A host controller is idle when Run/Stop is set to zero and HC Halted is one.

2.3 Port routing

Port routing is implemented according to the EHCI specification but functions regardless of whether the core is configured with or without an enhanced host controller. The VHDL generic prr enables or disables Port Routing Rules. With Port Routing Rules enabled, each port can be individually routed to a specific universal host controller via the VHDL generics portroute1 and portroute2. If Port Routing Rules is disabled the n_pcc lowest ports are routed to the first companion controller, the next n_pcc ports to the second companion controller, and so forth. The HCSP-PORTROUTE array is communicated via the portroute VHDL generics, which are calculated with the following algorithm:

$$\text{portroute1} = 2^{26}*CC_8 + 2^{22}*CC_7 + 2^{18}*CC_6 + 2^{14}*CC_5 + 2^{10}*CC_4 + 2^{6}*CC_3 + 2^{2}*CC_2 + CC_1 / 4$$

$$\text{portroute1} = 2^{26}*CC_{15} + 2^{22}*CC_{14} + 2^{18}*CC_{13} + 2^{14}*CC_{12} + 2^{10}*CC_{11} + 2^{6}*CC_{10} + 2^{2}*CC_9 + CC_1 \bmod 4$$

where CC_P is the companion controller that port P is routed to. Companion controllers are enumerated starting at 1.

When the enhanced host controller has not been configured by software, or when it is nonexsistent, each port is routed to its companion controller. This allows a universal host controller to function even if the host system does not have support for the enhanced host controller. Please see the EHCI specification for a complete description of port routing.

2.4 DMA operations

Both host controller types have configurable DMA burst lengths. The burst length in words is defined by the VHDL generic bwrd. The value of bwrd limits how many words a controller may access in memory during a burst and not the number of memory operations performed after bus access has been granted. When writing a data payload back to memory that requires half word or byte addressing the number of memory operations may exceed bwrd by one before the bus is released. If a host controller is given a byte-aligned data buffer its burst length may exceed the bwrd limit with one word when fetching payload data from memory.

The universal host controller uses a burst length of four words when fetching descriptors. This descriptor burst length is not affected by the bwrd VHDL generic. The universal host controller may be configured to start transactions on the USB before all data has been fetched from memory. The VHDL generic uhc_blo specifies the number of words that must have been fetched from memory before a USB transaction is started. Since the USB traffic handled by the universal host controller can be expected to have significantly lower bandwidth than the system memory bus, this generic should be set to a low value.

2.5 Endianness

The core works internally with little endian. If the core is connected to a big endian bus, endian conversion must be enabled. When the VHDL generic endian_conv is set, all AMBA data lines are byte swapped. With endian_conv correctly set the core will start accessing data payloads from byte offset zero in the buffer, this is the first byte that is moved on the USB. The VHDL generic endian_conv must be set correctly for byte and halfword accesses to work. Therefore it is not possible to change the byte order of the buffer by configuring the controller for a little endian bus when it is connected to a big endian bus or vice versa.

The VHDL generics be_regs and be_desc are used to place the controller into big endian mode when endian conversion is enabled. These configuration options have no effect when the core is connected to a little endian bus, as determined by the value of VHDL generic endian_conv. The VHDL generic be_regs arranges the core's registers in accordance with big endian addressing. In the enhanced host controller this will only affect the placement of the registers CAPLENGTH, HCIVERSION and the HCSP-PORTROUTE array. In the universal host controller be_regs will affect the placement of all registers. When be_regs is set, the bus to the register interface is never byte swapped. Tables 1 - 3 below illustrate the difference between big endian, little endian, and little endian layout with byte swapped DWORDs on two 16 bit registers. Register R1 is located at address 0x00 and register R2 is located at address 0x02.

Table 1. R1 and R2 with big endian addressing

31	16	15	0
R1(15:0)		R2(15:0)	

Table 2. R1 and R2 with little endian addressing

31	16	15	0
R2(15:0)		R1(15:0)	

Table 3. R1 and R2 with little endian layout and byte swapped DWORD

31	24	23	16	15	8	7	0
R1(7:0)		R1(15:8)		R2(7:0)		R2(15:8)	

The VHDL generic be_desc removes the byte swapping of descriptors on big endian systems. Tables 4 and 5 below list the effects of endian_conv and be_regs on a big endian and a little endian system respectively.

Table 4. Effect of endian_conv, be_regs, and be_desc on a big endian system

endian_conv	**be_regs**	**be_desc**	**System configuration**
0	-	-	Illegal. DMA will not function.
1	0	0	Host controller registers will be arranged according to little endian addressing and each DWORD will be byte swapped. In-memory transfer descriptors will also be byte swapped. This is the correct configuration for operating systems, such as Linux, that swap the bytes on big endian systems.
1	0	1	Host controller registers are arranged according to little endian addressing and will be byte swapped. Transfer descriptors will not be byte swapped.
1	1	0	Host controller registers will be arranged according to big endian addressing and will not be byte swapped. In memory transfer descriptors will be byte swapped.
1	1	1	Host controller registers will be arranged according to big endian addressing. In memory transfer descriptors will not be byte swapped.

Table 5. Effect of endian_conv, be_regs and be_desc on a little endian system

endian_conv	**be_regs**	**be_desc**	**System configuration**
0	-	-	Host controller registers will be placed as specified in the register interface specifications.
1	-	-	Illegal. DMA will not function.

2.6 Transceiver support

The controller supports UTMI+ and ULPI transceivers. All connected transceivers must be of the same type. Transceiver signals not belonging to the selected transceiver type are not connected and do not need to be driven.

2.7 PCI configuration registers and legacy support

The controller does not implement any PCI configuration registers. Legacy support is not implemented.

2.8 Software drivers

The core implements open interface standards and should function with available drivers. Gaisler Research supplies initialization code for both controllers for the Linux 2.6 kernel.

2.9 Registers

2.9.1 Enhanced host controller

The core is programmed through registers mapped into APB address space. The contents of each register are described in the EHCI specification.

Table 6. Enhanced Host Controller capability registers

APB address offset	Register
0x00	Capability Register Length
0x01	Reserved
0x02	Interface Version Number
0x04	Structural Parameters
0x08	Capability Parameters
0x0C	Companion Port Route Description

Table 7. Enhanced Host Controller operational registers

APB address offset	Register
0x14	USB Command*
0x18	USB Status
0x1C	USB Interrupt Enable
0x20	USB Frame Index
0x24	4G Segment Selector (Reserved)
0x28	Frame List Base Address
0x2C	Next Asynchronous List Address
0x54	Configured Flag Register
0x58 - 0x90	Port Status/Control Registers**

*Light Host Controller reset is not implemented.

**One DWORD register for each port.

2.9.2 Universal host controller

The core is programmed through registers mapped into AHB I/O address space. The contents of each register are described in the UHCI specification.

Table 8. Universal Host Controller I/O registers

AHB address offset	Register
0x00	USB Command
0x02	USB Status*
0x04	USB Interrupt Enable
0x06	Frame Number
0x08	Frame List Base Address
0x0C	Start Of Frame Modify
0x10 - 0x2C	Port Status/Control**

*The HCHalted bit is implemented as Read Only and has the default value 1.

**Over Current and Over Current Change fields have been added. Each port has a WORD register.

Table 9. Changes to USB Status register

15	6	5	4	0
UHCI compliant		HCH	UHCI compliant	

15: 6	UHCI compliant
5	Host Controller Halted (HCH) - Same behaviour as specified in the UHCI specification but the field has been changed from Read/Write Clear to Read Only and is cleared when Run/Stop is set. The default value of this bit has been changed to 1.
4:0	UHCI compliant

Table 10. Changes to Port Status/Control registers

15	11	10	9	0
UHCI compliant	OCC	OC	UHCI compliant	

15: 12	UHCI compliant
11	Over Current Change (OCC) - Set to 1 when Over Current (OC) toggles. Read/Write Clear.
10	Over Current Active (OC) - Set to 1 when there is an over current condition. Read Only.
9:0	UHCI compliant

2.10 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research), the enhanced host controller has device identifier 0x026, the universal host controller has device identifier 0x027. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

2.11 Configuration options

Table 11 shows the configuration options of the core (VHDL generics).

Table 11. Configuration options

Generic name	Function	Allowed range	Default
ehchindex	Enhanced host controller AHB master index	0 - NAHBMST-1	0
ehcpindex	Enhanced host controller APB slave index	0 - NAPBSLV-1	0
ehcpaddr	Enhanced host controller ADDR field of the APB BAR.	0 - 16#FFF#	0
ehcpmask	Enhanced host controller MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#

Table 11. Configuration options

Generic name	Function	Allowed range	Default
ehcpirq	Enhanced host controller interrupt line	0 - NAHBIRQ-1	0
uhchindex	Universal host controller AHB master index. If the core contains more than one universal host controller the controllers will be assigned indexes from uhchindex to uhchindex+n_cc-1.	0 - NAHBMST-1	0
uhchsindex	Universal host controller AHB slave index. If the core contains more than one universal host controller the controllers will be assigned indexes from uhc_hsindex to uhchsindex+n_cc-1.	0 - NAHBSLV-n_cc	0
uhchaddr	Universal host controller ADDR field of the AHB BAR. If the core contains more than one universal host controller the controllers will be assigned the address space uhchaddr to uhchaddr + n_cc.	0 - 16#FFF#	0
uhchmask	Universal host controller MASK field of the AHB BAR.	0 - 16#FFF#	16#FFF#
uhchirq	Universal host controller interrupt line. If the core contains more than one universal host controller the controller will be assigned interrupt lines uhc_hirq to uhchirq+n_cc-1.	0 - NAHBIRQ-1	0
memtech	Memory Technology used for buffers.	0 - NTECH	inferred
nports	Number of USB ports	1 - 15	1
ehcgen	Enable enhanced host controller	0 - 1	1
uhcgen	Enable universal host controller(s)	0 - 1	1
n_cc	Number of universal host controllers. This value must be consistent with nports and n_pcc, or portroute1 and portroute2, depending on the value of the generic prr. This value must be at least 1, regardless the value of generic uhcgen.	1 - 15	1
n_pcc	Number of ports per universal host controller. This value must be consistent with n_cc and nports: nports <= (n_cc * n_pcc) < (nports + n_pcc) when Port Routing Rules is disabled. The only allowed deviation is if (nports mod n_cc) < n_pcc in which case the last universal host controller will get (nports mod n_cc) ports. This generic is not used then Port Routing Rules (prr) is enabled.	1 - 15	1
prr	Port Routing Rules. Determines if the core's ports are routed to companion controller(s) with n_cc and n_pcc or with the help of portroute1 and portroute2.	0 - 1	0
portroute1	Defines part of the HCSP-PORTROUTE array	-	0
portroute2	Defines part of the HCSP-PORTROUTE array	-	0
endian_conv	Enable endian conversion. When set, all AMBA data lines are byte swapped. This generic must be set to 1 if the core is attached to a big endian bus, it must be set to 0 if the core is attached to a little endian bus.	0 - 1	1
be_regs	Arrange host controller registers according to big endian addressing. When set no endian conversion is made on the AMBA data lines connected to the host controller registers, regardless of endian_conv. Valid when endian_conv is enabled.	0 - 1	0
be_desc	Disable byte swapping of in-memory descriptors. Valid when endian_conv is enabled.	0 - 1	0

Table 11. Configuration options

Generic name	Function	Allowed range	Default
uhcblo	Universal Host Controller Buffer Limit Out. A universal host controller will start OUT bus transactions when uhcblo words of payload data has been fetched from memory.	1 - 255	2
bwrd	Burst length in words. A controller will not burst more than bwrd words when fetching data payloads. A universal host controller has a fixed, not affected by bwrd, burst length of four words when fetching transfer descriptors.	0 - 256	2
utm_type	Transceiver type: 0: UTMI+ 16 bit data bus 1: UTMI+ 8 bit data bus 2: UTMI+ ULPI	0 - 2	2
ext_vbus	Selects USB power source and fault detection: 0: Internal power source and no external fault indicator 1: External power source but no external fault indicator 2: External power source and external active high fault indicator 3: External power source and external active low fault indicator This generic is only valid for ULPI transceivers. UTMI+ transceivers must use a external power source and an external fault indicator.	0 - 3	3
ramtest	When set each controller maps its internal buffers into the controllers register space.	0 - 1	0

2.12 Signal descriptions

Table 12 shows the interface signals of the core (VHDL ports).

Table 12. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
UCLK	N/A	Input	USB clock	-
APBI	*	Input	APB slave input signals	-
EHC_APBO	*	Output	APB slave output signals	-
AHBMI	*	Input	AHB master input signals	-
AHBSI	*	Input	AHB slave input signals	-
EHC_AHBMO	*	Output	AHB master output signals.	-
UHC_AHBMO[]	*	Output	AHB master output vector.	
UHC_AHBSO[]	*	Output	AHB slave output vector.	-
O[]	xcvrsel[1:0]	Output	UTMI+ output signals	-
	termsel	Output	UTMI+ output signal	-
	suspendm	Output	UTMI+ output signal	Low
	opmode[1:0]	Output	UTMI+ output signals	-
	txvalid	Output	UTMI+ output signal	High
	drvvbus	Output	UTMI+ output signal	High
	datah[7:0]	Output	UTMI+ 16-bit interface output data bus	-
	validh	Output	UTMI+ 16-bit interface output signal	High
	host	Output	UTMI+ output signal	High
	csn	Output	ULPI output signal	Low
	resetn	Output	ULPI output signal	Low
	id	Output	ULPI output signal	-
	enablen	Output	ULPI output signal	Low
	fault	Output	ULPI output signal	-
	stp	Output	ULPI output signal	High
	data[7:0]	Output	UTMI+/ULPI output signals	-
	utm_rst	Output	UTMI+/ULPI output signal	Low
	dcrtl	Output	Data bus direction control; High: Input data	-
I[]	linestate[1:0]	Input	UTMI+ input signals	-
	txready	Input	UTMI+ input signal	High
	rxvalid	Input	UTMI+ input signal	High
	rxactive	Input	UTMI+ input signal	High
	vbusvalid	Input	UTMI+ input signal	High
	datah[7:0]	Input	UTMI+ 16-bit interface input data bus	-
	validh	Input	UTMI+ 16-bit interface input signal	High
	hostdisc	Input	UTMI+ input signal	High
	nxt	Input	ULPI input signal	High
	dir	Input	ULPI input signal	-
	data[7:0]	Input	UTMI+/ULPI input data bus	-

* see GRLIB IP Library User's Manual

2.13 Library dependencies

Table 13 shows the libraries used when instantiating the core (VHDL libraries).

Table 13. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	USBHC	Signals, component	Component declaration

2.14 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.usbhc_pkg.all;

-- USB Host controller with 2 ports. One enhanced host controller
-- and two universal host controllers.

entity usbhc_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    -- USBHC signals
    usbh_clkin     : in std_ulogic;
    usbh_clkout    : out std_ulogic;
    usbh_d         : inout std_logic_vector(15 downto 0);
    usbh_reset     : out std_logic_vector(1 downto 0);
    usbh_nxt       : in std_logic_vector(1 downto 0);
    usbh_stp       : out std_logic_vector(1 downto 0);
    usbh_dir       : in std_logic_vector(1 downto 0);
    usbh_id        : out std_logic_vector(1 downto 0);
    usbh_fault     : out std_logic_vector(1 downto 0);
    usbh_enablen   : out std_logic_vector(1 downto 0);
    usbh_csn       : out std_logic_vector(1 downto 0);
    usbh_faultn    : in std_logic_vector(1 downto 0);
    usbh_clock     : out std_logic_vector(1 downto 0);
    usbh_vbus      : in std_logic_vector(1 downto 0)
    );
end;

architecture rtl of usbhc_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);

  -- USBHC signals
  signal usbhci : usbhc_in_vector((CFG_USBHC_NPORTS-1) downto 0);
  signal usbhco : usbhc_out_vector((CFG_USBHC_NPORTS-1) downto 0);
  signal uhclko, urst, uhclk : std_ulogic;

begin

  -- AMBA Components are instantiated here
```

```
...

-- Instantiate pads, one iteration for each port
multi_pads: for i in 0 to 1 generate
    usbh_d_pad: iopadv
      generic map(tech => padtech, width => 8)
      port map (usbh_d((i*2**3+7) downto (i*2**3)), usbhco(i).data,
                usbhco(i).dctrl, usbhci(i).data);
    usbh_nxt_pad : inpad generic map (tech => padtech)
      port map (usbh_nxt(i),usbhci(i).nxt);
    usbh_dir_pad : inpad generic map (tech => padtech)
      port map (usbh_dir(i),usbhci(i).dir);
    usbh_resetn_pad : outpad generic map (tech => padtech)
      port map (usbh_reset(i),usbhco(i).utm_rst);
    usbh_stp_pad : outpad generic map (tech => padtech)
      port map (usbh_stp(i),usbhco(i).stp);
    usbh_csn_pad : outpad generic map (tech => padtech)
      port map (usbh_csn(i),usbhco(i).csn);
    usbh_enablen_pad : outpad  generic map (tech => padtech)
      port map (usbh_enablen(i), usbhco(i).enablen);
    usbh_fault_pad : outpad generic map (tech => padtech)
      port map (usbh_fault(i),usbhco(i).fault);
    usbh_id_pad : outpad generic map (tech => padtech)
      port map (usbh_id(i), usbhco(i).id);
    usbh_clock_pad : outpad generic map (tech => padtech)
      port map (usbh_clock(i),usbhco(i).clock);
    usbh_faultn_pad : inpad generic map (tech => padtech)
      port map (usbh_faultn(i), usbhci(i).faultn);
    usbh_vbus_pad : inpad generic map (tech => padtech)
      port map (usbh_vbus(i),usbhci(i).vbus);
  end generate;
  -- Clock generation. This example generates a clock to the
  -- transceivers. If the clock is genereated outside the FPGA
  -- no clock generator or clock out pads are necessary.
  usbh_clkgen : clkmul_virtex2
    generic map (clk_mul => 6, clk_div => 10)
    port map ('1', lclk, uhclko, urst);

  -- Clock to transceivers
  usbh_clkout_pad : outpad
      generic map (tech => padtech)
      port map (usbh_clkout, uhclko);

  -- Input clock for USB clock domain. Note: arch => 3 creates
  -- a clock pad which removes input delay by instatiating a BUFGDLL
  -- or similar.
  usbh_clkin_pad :
    clkpad generic map (tech => padtech, arch => 3)
    port map(usbh_clkin, uhclk);

   usbhostcontroller0: usbhc
    generic map (
      ehchindex => 5,
      ehcpindex => 14,
      ehcpaddr => 14,
      ehcpirq => 9,
      ehcpmask => 16#fff#,
      uhchindex => 6,
      uhchsindex => 3,
      uhchaddr => 16#A00#,
      uhchmask => 16#fff#,
      uhchirq => 10,
      memtech => memtech,
      nports => 2,
      ehcgen => 1,
      uhcgen => 1,
      n_cc => 2,
      n_pcc => 1,
      prr => 0,
      portroute1 => 0,
      portroute2 => 0,
```

```
        endian_conv => 1,
        be_regs => 0,
        be_desc => 0,
        utm_type => 2,
        uhcblo => 5,
        bwrd => 4,
        extvbus => 3,
        ramtest => 0)
      port map (
        clkm,uhclk,rstn,urst,apbi,apbo(14),ahbmi,ahbsi,
        ahbmo(5),
        ahbmo(7 downto 6),
        ahbso(4 downto 3),
        usbhco,usbhci);
end;
```

Gaisler Research AB
Första Långgatan 19
413 27 Göteborg
Sweden

tel +46 31 7758650
fax +46 31 421407
sales@gaisler.com
www.gaisler.com