FPGA prototyping of an USB Host Controller

Hudson Veloso, Diego Melo, Renata Garcia, Marcelo Lucena, Antonyus Pyetro, Edna Barros

Federal University of Pernambuco

ABSTRACT

This paper presents the FPGA prototyping of an USB Host Controller IP-core developed in behavioral SystemC. The IP-Core was designed using the Cynthesizer® behavioral synthesis tool, provided by Forte Design Systems. The USB Host Controller has been prototyped and validated in the Stratix II based Altera® platform. This platform contains a Nios II processor where runs a driver developed to work with the USB HC and the µClinux OS.

1. INTRODUCTION

This paper presents the architecture of an USB Host Controller IP-Core designed using behavioral synthesis tools. How the architecture has been conceived at a high level of abstraction and the problems during the prototyping phase is the focus of this work.

The IP-Core has been specified in behavioral SystemC and its functionalities have been verified in different abstraction levels: behavioral, RTL and in FPGA. It was also developed a driver supporting the communication between the controller and the µClinux OS. The complete system runs in an Altera® platform.

This paper is divided as follows: section 2 shows an overview of the USB Host; section 3 explains the architecture of the USB Host Controller and the modules that compose it; section 4 explains the platform used for prototyping the HC and how problems that emerged from a physical level were resolved; section 5 presents some results obtained; finally section 6 presents the conclusions and future works.

RELATED WORKS

Due the complexity increase of the electronic devices and the need to decrease the time design, the high-level specification approach has been used as an alternative to improve the design process. This approach is based on the system specification through of a behavioral model and communication structures in high-level abstraction.

A library for modelling of high-level communication is proposed as a SystemC extension for providing the modelling in high-level serial protocols, such as USB, CAN and FireWire [11]. But the library is limited because it isn't enough to define completely the architecture with the control and data channel.

The behavioral specification must be converted to RTL level that realizes the expected behavior [10]. For performing this, there are some tools that generate automatically the RTL code from the behavioral specification. Besides this, it also is used to refine the model, solving the most common design problems.

2. USB HOST OVERVIEW

The USB Host is the way that allows any system to communicate with USB devices through the USB protocol. It is composed by hardware and software. The hardware, called Host Controller (HC), is a controller that provides the mechanical and electrical interfaces to connection of devices. It also implements a data channel to share data and information among devices and others system peripherals. The Host Controller Driver (HCD) is the software that provides an abstraction, for the system higher level, allowing access to the HC functionalities. The HC and the HCD are based on the USB 2.0 specification [1] and on the OpenHCI specification [4]. The complete USB System is composed by layers, from the operating system to the device, as shown in Figure 1.

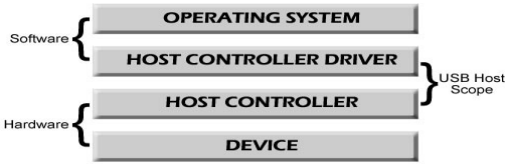

Figure 1: USB Host layers

The HCD may exchange information with the HC using Open Host Controller Interface (OHCI). The OHCI was the interface chosen for this design, due to the fact that its design that is more hardware oriented.

The HC communicates with the device through a physical interface, or PHY. Its role is to take care of the digital and modulated parts of the interface (encoding and decoding). Figure 2 shows the logic view of the USB Host architecture.

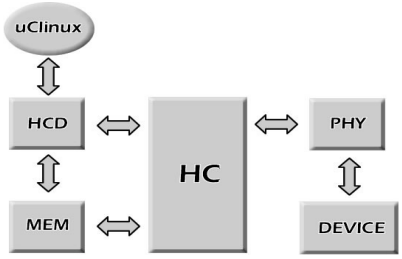

Figure 2: USB Host logic view

The MEM shown in Figure 2 is a shared memory used to exchange information between the HC and HCD; this memory is defined by the OpenHCI specification [4].

3. HC ARCHITECTURE

The HC is basically a data channel but, due to the complexity and requirements of an USB Host Controller, needs a control module. The modules from the HC can be divided into two basic groups. One implements the Data Channel and the other implements the System Control.

Each module has been verified separately before its integration to the whole system. The Figure 3 shows an overview of the main HC modules.

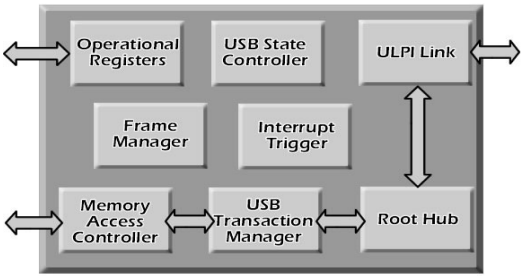

Figure 3: HC Architecture

The communication among the modules has been done at pin-accurate and TLM level, using FIFO's and buffers to achieve the desired data flow rate and adjusting the latency for synchronisation with the external clock source. In the next two subsections it will be explained the architecture of this modules and its functionalities.

3.1 Data Channel Modules

The Data Channel Modules is composed by Memory Access Control, USB Transaction Manager, Root Hub and ULPI Link. The details about them are exposed in the following subsections.

3.1.1. Memory Access Control (MAC)

The MAC connects the memory to the others HC internal modules through the use of modular interfaces and pin-accurate interfaces.

The main function of MAC is to schedule the data received from HC modules in order to control the memory accesses.

3.1.2. USB Transaction Manager

The USB Transaction Manager module was divided in two sub-modules: Prepare Transaction Control (PTC) and Execute USB Transaction.

The PTC is responsible to check if the devices, that will be served, are ready to communicate with the host. The Execute USB Transaction is activated by the PTC when exists a device ready to change information.

The algorithm is mainly concerned with the type of packet transaction, which could be "SETUP", "IN" or "OUT" packets. It depends on the type of transaction ("Control", "Bulk" or "Interrupt").

When a transaction ends, the USBT Controller must update the system to reflect the last transaction performed. This consists of updating the endpoint and transfer descriptors of the OHCI specification.

3.1.3. Root Hub

The RootHub has responsibilities like monitoring the Host Controller state, transmitting and receiving data from the Host Controller core to the downstream ports and transmitting StartOfFrame (SOF). It has the knowledge of inter packet transaction sequences to set up its counters to signal the correct timing requirements of the ULPI Specification.

3.1.4. ULPI Link and the PHY

The ULPI Link Module and the physical interface, or PHY (USB 3300 from SMSC® [7]), are based on the ULPI specification [3]. The ULPI is a low-pin interface version of the UTMI+ specification [2].

The ULPI Link receives processes and informs other modules from events such as device connection or disconnection; device reset completion, transmission or reception errors and accomplishments among others USB operations.

3.2 System Control Modules

The Operational Registers module is a register bank that stores information about the status and configuration of the HC and its transmissions.

The HCD uses the Operational Registers to configure and control the HC by writing the registers.

The USB State Controller takes care of the USB states transition. The USB specification defines four possible states for the HC: "USBSuspend", "USBResume", "USBOperational" and "USBReset".

The Frame Management (FM) takes care of the USB host controller and accomplishes the timing requirements specified in [1].

The Interrupt Trigger module is in charge of informing to the HCD that a transaction has been accomplished.

4. PLATFORM AND PROTOTYPING

The HC IP-core has been prototyped on a Stratix II EP2S60F672 FPGA [6] and integrated in a platform, using the Quartus II and SOPC Builder [9]. The final platform includes a memory, the NIOS II processor (used to run the OS), the USB Host Controller and the Avalon bus [8], to interconnect all the components.

Figure 4 shows the components and connections that compose the platform.

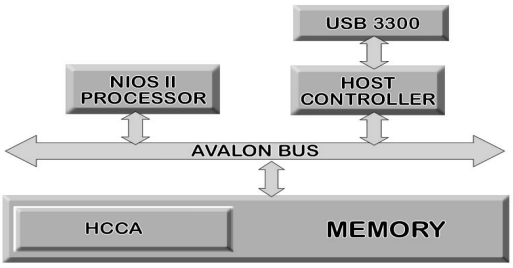

Figure 4: Platform for the USB Host

The HCCA is a shared memory used to exchange information between HC (hardware) and the HCD (software) modules. The HCD is running in the NIOS processor. The memory area is allocated by the HCD at the initialization and its address is stored in one of the HC registers, usually the HCCA stores information about the transmissions status.

The following sections talk about the problems found in this phase. They were related to the synchronization between the interfaces HC/PHY and HC/Nios II processor.

4.1 HC Two Clock Domain

The Nios II processor operates at a maximum clock of 50 MHz, but the PHY generates a 60 MHz clock. To solve this problem, a PLL has been used to correct the clock phase delay and generates a 30 MHz clock. The reason for this is the multiplicity behind the numbers 60 and 30 allowing a correspondence in the number of cycles, 2:1.

The MAC and Operational Register have a HC interface that receives 60MHz and an Avalon Bus interface that receives 30MHz of clock. This solution allowed that the NIOS II and the Avalon Bus could operate at 30 MHz and was able to communicate with the HC operating at 60 MHz.

4.2 PHY and HC Synchronisation (Clock Phase Adjustment)

This PLL adjusts the duty cycle to 50% and advances the phase. That clock adjustment compensates the delay of the communication between HC and the PHY and allows the prototyped HC to write the "STP" and "DATA" (out direction) signals to the PHY in the required timing.

However this clock adjustment has introduced a negative effect in the incoming communications that would not need any compensation. The use of high-level directives of input and output delay in the ULPI Link's ports introduce a delay to read the input signals balancing, therefore, the action of the clock phase advancing. Figure 5 shows how the PLL is connected to the platform.

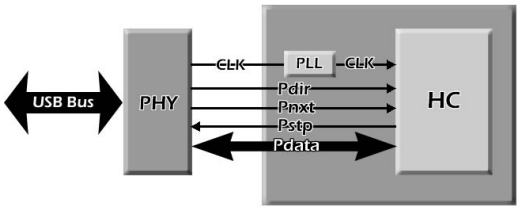

Figure 5: PLL interconnecting PHY and HC

5. RESULTS

The result of this work is an USB Host IP-core prototyped in FPGA occupying 54% of the total area and the HC occupying 48%. The HC data channel has been verified and validated through the sending and receiving of data packages. In our test cases the USB Host recognizes a pen-drive, reset it and requests information about its configuration; validating the communication with the device.

Table 1 shows some implementation and prototyping numbers that shows the complexity of the designed USB Host IP-core.

Results	
Requirements	147
Architecture	12 modules
Source Code	9158 code lines
Test-bench	16400 code lines
Prototyping	20599 ALUS

Table 1: USB Design results

Concerning functional verification, the test vectors cover two implemented transmission types: Bulk and Control transmission.

Table 2 shows the number of source code lines of the HC in different abstraction levels: behavioural SystemC and SystemC RTL. It shows the programming effort difference between these two abstractions levels.

Level	Code Lines
Behavioral SystemC	9158
RTL SystemC	81066

Table 2: Code Lines

Table 3 shows the simulation time spent to accomplish two protocols tests, each one executing all the channel tests in different abstraction levels: behavioral and RTL. It shows that the behavioural simulation time is about 50% shorter than the RTL simulation Time.

Protocol Test	Behavioral	Verilog
ACK	53m54s	110m26s
TIMEOUT 3	69m23s	143m0s

Table 3: Simulation Time

The test named ACK is the ideal flow protocol test, where the HC receives only ACK tokens; this test is the

fastest protocol test. Timeout 3 is the test where the Timeout token is issued three times in the same transmission; this is the slowest test. The tests have been performed in a machine including an AMD Sempron(tm) Processor +2800, 1GByte with CentOS Operating System.

6. CONCLUSION AND FUTURE WORK

The advantages in using a behavioral description in the project of digital systems, principally for complex projects like the USB Host Controller, justify its adoption. The reduction in time of development and simulation allows architecture exploration and rapid prototyping, increasing the designer productivity.

During the prototyping phase occurred some problems how the two clock domains and the de-syncronisation between the electrical interface and FPGA. They were resolved by the introduction of high-level directives during the synthesis and adjusting clock phase using a PLL.

With the validated IP-Core in FGPA, the next work consists of generating the layout of the USB Host Controller with the objective to validate the IP-core as ASIC.

REFERENCES

[1] COMPAQ; HEWLETT-PACKARD; INTEL; LUCENT; MICROSOFT; NECK; PHILIPS. Universal Serial Bus Specification. Revision 2.0. April, 2000. URL: http://www.usb.org/developers/docs/

[2] UTMI+ Specification, Revision 1.0. URL: http://www.smsc.com/main/catalog/ulpi.html

[3] UTMI+ Low Pin Interface (ULPI) Specification, Revision 1.1. URL: http://www.smsc.com/main/catalog/ulpi.html

[4] COMPAQ; MICROSOFT; NATIONAL SEMICONDUCTOR. OpenHCI – Open Host Controller Specification for USB. Release: 1.0a September, 1999 URL: ftp://ftp.compaq.com/pub/supportinformation/papers/hcir1_0a.pdf

[5] Brazil-IP Network. Brazilian network for the design of digital systems and IP-Cores.
URL: http://www.brazilip.org.br/

[6] ALTERA. Stratix II Handbook, Volume 1 and 2. May, 2007.
URL: http://www.altera.com/literature/lit-stx2.jsp

[7] SMSC® USB3300 Datasheet – Hi-Speed USB Host, Device or OTG PHY with ULPI Low Pin Interface, Revision 1.08, July 11, 2007.
URL: http://www.smsc.com/main/datasheets/3300.pdf

[8] Avalon Interface Specification, Version 1.0. May, 2005 – Altera®.

[9] ALTERA. Quartus II Version 7.2 Handbook, Volume 4. October, 2007.
URL: http://www.altera.com/literature/hb/qts/qts_qii5v4.pdf

[10] Siegmund, Robert; Mueller, Dietmar – Automatic Synthesis of Communication Controller Hardware from Protocol Specifications - http://ieeexplore.ieee.org/iel5/54/21911/01018137.pdf?tp=&isnumber=&arnumber=1018137

[11] McFarland, Michael C.; Parker, Alice C.; Camposano, Raul - The High-Level Sysnthesis of Digital Systems – URL: http://ieeexplore.ieee.org/iel1/5/1878/00052214.pdf?tp=&isnumber=&arnumber=52214