High Speed USB 2.0 Interface for FPGA Based Embedded Systems

Fatemeh Arbab Jolfaei, Neda Mohammadizadeh, Mohammad Sadegh Sadri, Fatemeh FaniSani
Isfahan University of Technology, Department of Electrical & Computer Engineering
f.arbabjolfaee@ec.iut.ac.ir

Abstract

FPGA implementation of high speed serial peripherals such as USB 2.0 are of great use. The Cypress SX2 USB 2.0 controller is one of the suitable choices for developing FPGA based USB peripherals. A simple interface module capable of transferring data rates above 400Mbits/s can be implemented to communicate with SX2. FPGAs can efficiently be used for building embedded systems. Xilinx complete set of development tools make implementation of large System-On-Chip designs feasible. We present two complete architectures for connecting SX2 to FPGA. First design minimizes FPGA resource usage while keeping a reasonable speed. In the second design, optimizations are done to reach maximum USB 2.0 interface speed at the cost of some additional logic. In order to use developed module in Xilinx embedded design flow, we make a custom peripheral which includes SX2 interface as its core and additional logic capable of connecting to OPB and PLB.

1. Introduction

High speed easy to use peripheral interfaces like USB 2.0 are of great use in today's world. USB 2.0 standard specifies a transfer rate of 480 Mega Bits per second for it's peripheral devices [1,7]. The high data transfer rate of USB 2.0 interface makes it a suitable choice for many different purposes such as data acquisition and processing.

The Xilinx Spartan-3 FPGA family, built on the success of four previous generations of cost-optimized Spartan FPGAs, offers platform capabilities with a wide range of I/O and density options. Efficiently utilizing 90 nanometer technology, Spartan™-3 devices provide aggressive speed, pricing, density and feature-rich low-cost FPGAs to customers [2].

Implemented hardware on a Spartan-3 FPGA is easily capable of handling data throughputs as high as what USB 2.0 demands. Using dedicated resources such as multipliers it is also possible to do signal processing tasks on these data. In addition, the large amount of available logic resources on FPGA allows integration of a complete system on a single chip. This makes Spartan-3 FPGA a suitable choice for use in USB peripheral designs.

In this paper, we connect a USB 2.0 controller to an FPGA to build a complete USB 2.0 system. Generally, commercial USB controllers can be divided into two categories: The first category includes stand alone controllers which contain an integrated microcontroller in addition to USB interfacing modules. The second category includes USB controllers which do most of needed USB interactions automatically and for other operations, rely on an external master, so user will develop all of the needed functionalities on his own FPGA, DSP or CPU.

Cypress is one of the most famous providers of USB 2.0 controllers. SX2 is a Cypress's offering that has a built-in USB transceiver and Serial Interface Engine (SIE), along with a command decoder for sending and receiving USB data [3]. It automatically responds to USB standard requests without any external master intervention. The SX2 presents two interfaces to the external master, a FIFO interface and a command interface. FIFO/Command interface can be asynchronous or synchronous. At startup the external master use the default descriptor built into the SX2 or load a complete descriptor into SX2. Here we use the second choice.

SX2 is mainly designed to be connected to microcontroller like devices. So, one idea for FPGA implementation of an interface to SX2 is to use a microprocessor core.

Xilinx provides different microprocessor cores. Virtex-4 family contains a high performance PowerPC hard core inside FPGA. However for other families such as Spartan-3, soft microprocessor cores are available. Examples include MicroBlaze and PicoBlaze soft cores. MicroBlaze provides a complete microprocessor system on an FPGA while providing a powerful 32 bit processing engine for different tasks. PicoBlaze can also be used in simpler designs and developers can get familiar with it in an hour.

KCPSM3 (PicoBlaze) is a very simple 8-bit microcontroller [4], primarily designed for the Spartan-3, Virtex-II, Virtex-II PRO, Virtex-4 and Virtex-5 devices. Although it could be used in processing data, it is most likely employed in applications requiring a complex but non-time critical state machine.

* Paper presented in EM-COM 2009 Korea

The revised version of the popular KCPSM macro occupies just 96 spartan-3 slices which is just 5% of XC3S200 device and less than 0.3% of XC3S5000 device. Together with this small amount of logic, a single block RAM is used to form a ROM store for a program of up to 1024 instructions. Even with such size constraints, the performance is respectable at approximately 43 to 100 MIPS depending on device type and speed grade.

Xilinx provides a complete set of tools for developing FPGA based embedded systems. Xilinx Embedded Development Kit (EDK) allows one to easily make a complete system containing CPU, memory controller and peripherals with just a few mouse clicks. EDK contains a large set of ready to use cores for all parts of the system.

As an example EDK provides Microblaze core as a 32bits CPU, a dedicated DDR SDRAM controller core and various kinds of peripherals such as: UART, I2C, SPI, VGA, PS/2, GPIO and Audio interfaces.

EDK uses Peripheral Local Bus (PLB) and On-Chip Peripheral Bus (OPB) as means of connecting CPU and other modules together. These busses, mainly designed by IBM, are one of the most efficient and most widely used set of buses for System-On-Chip applications.

One of the keys to success in making FPGA based embedded systems is the ability to make custom peripherals capable of connecting to PLB and OPB. You can build your own peripheral and add it to your design repository. After that the core can be used easily in any embedded system using EDK.

One of the main goals of this paper is to design a fully functional USB 2.0 interface and to add it to Xilinx embedded base system. This way we can take advantage of USB 2.0 high speed peripheral in our FPGA based embedded designs. This can be useful for anybody who is using EDK and needs a reliable high speed data transfer interface to outside world.

There are some problems regarding practical design of USB 2.0 interface. 400MBits/s is a very high data transfer rate, therefore data transmission between FPGA and USB 2.0 chip should be done with highest possible speed. Correct clocking schemes should be utilized to ensure that all of hold/setup timing requirements are met. Connecting the embedded USB interface module to PLB bus, there are some additional considerations. In addition to necessary HDL coding the related software, executed on Microblaze or PPC, should be developed carefully.

This report provides a complete design of a USB 2.0 peripheral using Spartan-3 FPGA. This includes system schematics, FPGA related state machines and flow charts and description of program running on PC side along with needed drivers. We try to reduce logic resource usage as much as possible while keeping data transfer rate of 480Mbits/s.

2. USB Interface Description

Figure 1 shows how SX2 is connected to an FPGA. Required tasks for FPGA in this work could be divided into two sections: 1- Complex tasks and 2- Usual tasks. Most of the times complex tasks are not time critical, but some usual tasks should be performed as fast as possible. Thus we provide two choices to the user. One choice is FPGA resource utilization of less than 100 slices and limited transfer rate of up to 100Mbits/s. The other choice is More complicated design, resource usage of above 300 slices and higher performance of more than 400Mbits/s.

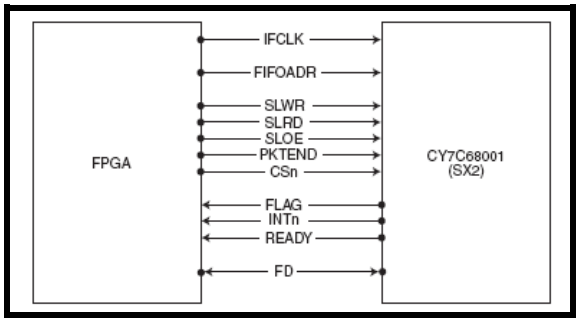

Figure 1. FPGA - SX2 connection

The simplified design only consists of the KCPSM module and a very little amount of FPGA logic. The block diagram of the high performance design on FPGA is shown in Figure 2. It consists of the following modules: ClockGen module Generates needed clock pulses of the entire system, with desired frequencies and phase shifts. PicoBlaze module Performs needed initialization tasks and manages exceptions and interrupts. FSM module is a high speed module which acts as the main control unit and provides synchronization between different parts of the system. IOInterface module Handles FPGA connection to SX2. DataManager module connects USB related modules to the rest of the modules on FPGA.

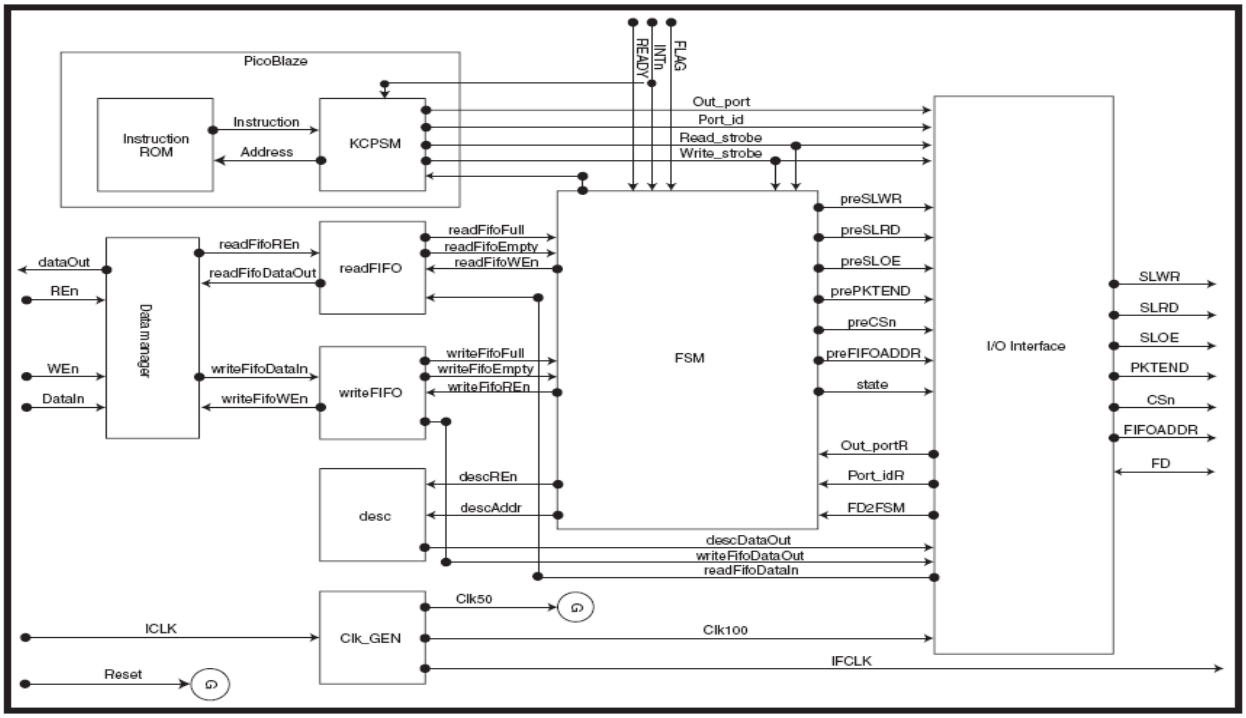

Figure 2. USB interface block diagram in high performance mode

2.1. Clock Generator

Reaching a rate of above 400Mbits/s needs SX2 device to be clocked at more than 30MHz when in synchronous mode. In order to meet SX2's required setup and hold times, a special clocking scheme is used. Figure 3 shows clock generator's block diagram.

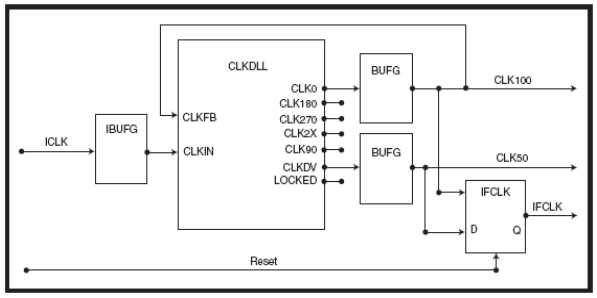

Figure 3. Clock generator block diagram

2.2. PicoBlaze

Figure 4.a and 4.b show execution flowcharts in the simplified design and high performance design respectively. Making SX2 enabled by setting CS# active, PicoBlaze waits for ready interrupt by running checkInt routine. After interrupt acquires, PicoBlaze initiates a read request. The received byte is interrupt status which indicates the type of interrupt.

After receiving ready interrupt, SX2 registers are configured by PicoBlaze. Some of the configurations are as follows:

A suitable value for INTENABLE register is chosen and corresponding interrupts are enabled/disabled. IFCONFIG register contains important settings such as selecting SX2's clock source (internal/external) and SX2's Synchronous/Asynchronous mode. In simplified design, SX2 is configured to use its own internal clock source and data transfers between FPGA and SX2 will be in asynchronous mode. For high performance design, synchronous data transfers between FPGA and SX2 will be done using an FPGA provided clock source.

In high performance design, FSM module is responsible for sending descriptor to SX2 by receiving "DESC report" signal from PicoBlaze. In simplified version, this task is done by PicoBlaze itself.

After the descriptor is sent, PicoBlaze waits for ENUMOK interrupt to ensure that enumeration process has been successful. It then checks if SX2 has enumerated in either full or high speed mode.

Finishing the initialization process, PicoBlaze handles the received interrupts by executing the related routines.

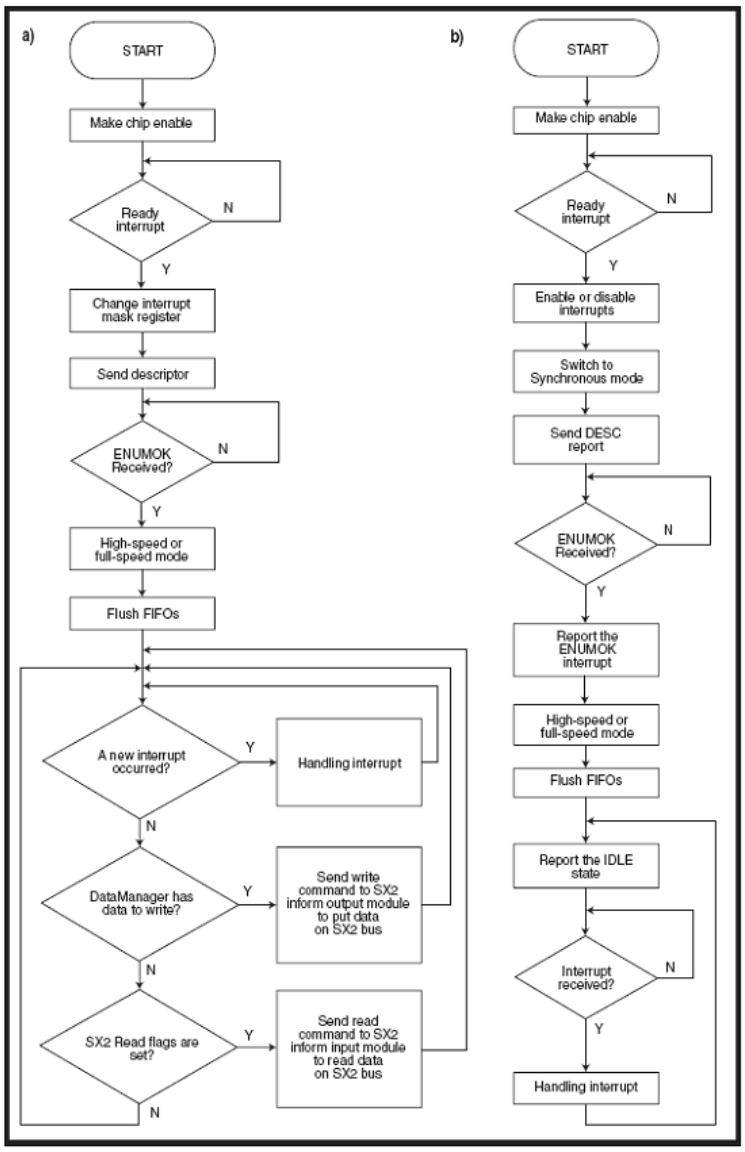

Figure 4. PicoBlaze execution Flow a) simplified mode b) high performance mode

2.3. FSM

The designed FSM consists of 6 states (Figure 5). All of the transitions in this FSM are synchronized with positive edge of clock. The FSM has also an asynchronous reset input.

Table 1 shows the function of each state briefly. At reset, The FSM enters PICOBLAZE state unconditionally in which PicoBlaze will do the required initialization tasks. When finished, Pico Blaze will send a message to FSM indicating that it can go to DESC state. In the DESC state, the descriptor will be transferred to SX2's descriptor RAM in command synchronous mode. After the entire descriptor has been transferred, FSM enters the IDLE state. When enumeration is completed, the SX2 will notify the FSM with an ENUMOK interrupt which is handled properly. Now the system is ready for its normal operation.

FSM leaves the IDLE state in three cases. If an interrupt occurs, FSM enters the INT_HANDLE state in which the FSM reads the interrupt status byte to determine the interrupt source, and then system control will be passed to PicoBlaze which performs required interrupt tasks and then informs FSM to continue its normal operation. If write FIFO has data to send, FSM goes to WRITE state. Data will be sent to SX2 using slave FIFO synchronous mode. FSM stays in the

WRITE state until all the data in write FIFO is sent or an interrupt occurs. In the latter case, transition to INT_HANDLE state happens after the write operation for the current 16-bit word is completed. When the USB host writes to one of the OUT endpoints, the related FIFO flag will be asserted and FSM enters the READ state. Similar to the WRITE state, FSM stays in this state until all data is read or an interrupt occurs.

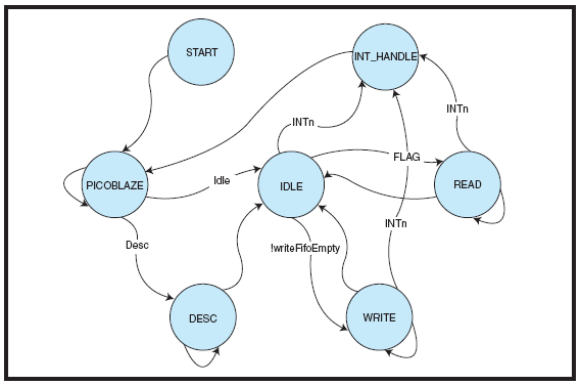

Figure 5. FSM execution diagram

2.4. I/O Interface

The IOINTERFACE module is responsible for controlling the I/O pins connected to SX2.

This block uses the 100MHz clock pulse in order to provide appropriate timings for control signals. For the rest of the operations, this module uses the same 50MHz clock pulse as other modules.

3. Custom Module for EDK

Using Embedded Development Kit create and import peripheral wizard, we made a custom module for our SX2 interface. Created custom module can easily connect to PLB bus. The base system CPU can be either Microblaze or PowerPC. Microblaze can be used in designs with any of Spartan-II/III or Virtex targets. PowerPC hard CPU core is available just in Virtex-IIP or Virtex-4/5FX FPGAs. Our custom module uses three different ways to talk to base system CPU and other modules on PLB: 1- Soft registers: dedicated registers inside custom module, mainly used for initialization and confirmation. 2- Interrupt: Our custom module is capable of interrupting the CPU when an event happened. For example when a new USB packet is arrived. 3- FIFOs : These are the main logic inside custom module responsible for transmission and reception of data from PLB.

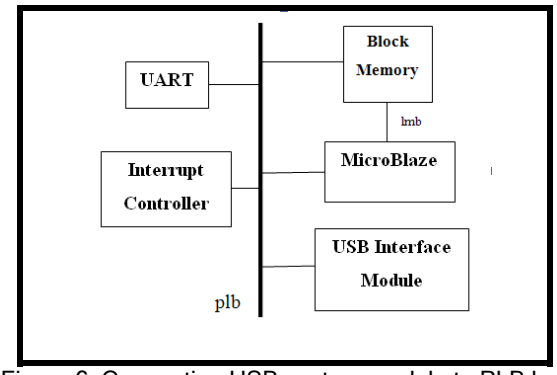

Figure 6. Connecting USB custom module to PLB bus

Designed SX2 interface module, described in section 2 will be added inside custom module as user logic. Additional code is developed in order to connect signals and make required timings.

Initialization, and control of the custom module will be done using CPU. Software running on CPU is responsible for this task. A set of library routines is developed which provide the capability for user to configure custom module, send/receive data and get its status.

Developed custom module can be easily inserted in any EDK base system. By the addition of library routines to the project, user can take advantage of the module easily. Figure 6 is the block diagram of a sample EDK project with custom module included in it.

Table 1. FSM states' descriptions

State	Description
PICOBLAZE	Performing basic initializing tasks and responding to the interrupts
DESC	Sending the descriptor to cypress
IDLE	System has no special task
INT_HANDLE	Determining the interrupt source
WRITE	Writing data to USB IN endpoint
READ	Reading from USB OUT endpoint

4. PC Side Software

Two complete sets of exmple programs are developed for both Windows and Linux. The most important point in USB software development is the kernel mode driver needed for communication with USB device.

Sample codes for sending data to USB peripheral and receiving data from it, using libUSB [5], is developed and tested for both windows and Linux operating systems. Qt development environment is used for Linux based samples. Another solution is to use USB 2.0 drivers provided by Cypress [6]. This

solution is solely for windows Operating System. Sample Windows applications are developed using MFC (MS Visual Studio).

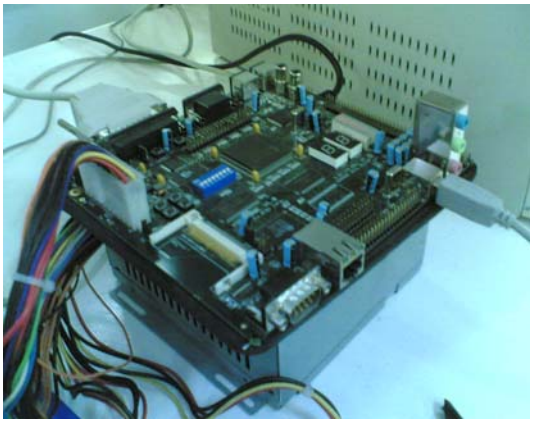

Figure 7. USB 2.0 interface on XSB300E board

5. Practical Implementation

This design is implemented on a Xess© XSB300E board which mounts a Xilinx® Spartan–IIE XC2S300E FPGA device connected to Cypress CY7C68001 USB controller. We also developed a new board using Spartan-3 FPGA, composite video interfaces and SX2 USB2.0 interface. The board is capable of transferring digitized video over USB to PC.

Table 2. Performance test results

Experiment Description	Device	Operating System	Achieved Rate
Transfer a 400 MByte file to board in high performance mode	XC2S300E	Windows (CyUSB)	432 Mbits/s
Transfer a 400 MByte file to board in high performance mode	XC2S300E	Linux (libUSB)	410 Mbits/s
Transfer digitized video to PC using simple design	XC3S400	Linux (libUSB)	113 Mbits/s

In high performance design 16-bit SX2 bus is selected and a 50 MHz clock pulse is used to synchronize FPGA and SX2. So data transmission rate between FPGA and SX2 is high enough to make the 480 Mbits/s rate of USB 2.0 standard achievable.

A wide range of USB 2.0 products, design cores and IPs are available in the market, however the authors could not find any solution capable of being implemented on all of the Spartan2/3, Virtex4,5 and Virtex-6 FPGAs. Besides, in many cases the provided core supports just USB full speed interface and not 480Mbits/s rate.

Table 3. FPGA resource usage

Design Type	Device	Slices	BRAMs	DLLs
High performance	XC2S300E	344 out of 3072 (11%)	9 out of 16 (56%)	1 out of 4 (25%)
Simple design	XC2S300E	191 out of 3072 (6%)	5 out of 16 (31%)	---
Simple design	XC3S400	34 out of 1792 (1%)	1 out of 16 (6%)	---

6. Conclusion

USB 2.0 interfaces can play an important role in embedded systems. FPGAs provide a high level of flexibility and processing power and so can be widely used in various kinds of embedded systems for different applications. Xilinx provides a complete set of development tools for building a complete system on one FPGA. It is a big advantage to be able to add a high speed USB 2.0 interface to an FPGA based embedded system. We have developed and tested a complete design of a USB 2.0 interface which provides data transfer rate of over 400Mbits/s. Finally we have integrated the developed high speed module into Embedded Development Kit. One can easily use the module in his/her FPGA based embedded design for whatever application needed.

7. References

[1] Jan Axelson, *USB Complete - Everything you need to develop custom USB peripherals*, Lakeview research, 3rd Edition, Aug 2005

[2] "Spartan-3 FPGA Family: Complete Data Sheet", Xilinx Corp., Aug 2005

[3] "CY7C68001 EZ-USB SX2 High-Speed USB Interface Device", Cypress Corp., Jun 2005

[4] "PicoBlaze 8-Bit Embedded Microcontroller User Guide", Xilinx Corp, Nov 2005

[5] "libUSB Manual", libUSB Project, Available at: http://libusb.sourceforge.net/

[6] "Cypress CyUSB.sys programmer's reference", Cypress Corp, 2003

[7] "USB in a nutshell, Making sense of the USB standard", Craig Peacock, Nov 2002, Available at: www.beyondlogic.org