Low cost USB - local bus interface for FPGA based systems

Wojciech M. Zabołotny[a] and Grzegorz Kasprowicz[a]

[a]Institute of Electronic Systems, Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warszawa, Poland

ABSTRACT

This paper presents a concept of simple interface allowing to control the local bus of the FPGA based system from the computer working as USB host. Such approach may significantly simplify and decrease cost of development of FPGA based systems.

Keywords: FPGA, Local Bus,

1. INTRODUCTION

When designing the electronic systems containing relatively complex digital part, we usually use the local bus, allowing to access the registers and memories.

Typical solution, based on industrial standard, used in many data acquisition systems is the VME bus. However usage of the VME bus requires quite sophisticated and costly infrastructure. One needs the VME crate, which provides the power supply, and mechanically supports boards. There is also necessary a VME controller. Use of VME for simple systems, consisting of a single board, or of a few boards is simply unjustified.

Our goal was to create simple, low cost interface, which can provide efficient access to the parallel local bus, using a typical interface available in modern computers (which will be used during the development of the system) and embedded systems (which will be used as system controller after development is finished). The desired architecture of proposed interface is shown in the Fig. 1.

2. SELECTION OF COMMUNICATION INTERFACE

If we look for a relatively fast interface widely available in typical computers and embedded systems, we can see three possibilities:

- Ethernet
- PCIe
- USB

Further author information: (Send correspondence to W.M.Z.)
W.M.Z.: E-mail: wzab@ise.pw.edu.pl, Telephone: +48 22 234 7717

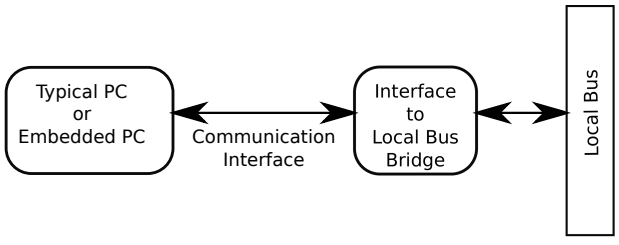

Figure 1. Required architecture of our interface.

Not all of the above interfaces are equally well suited for our purpose.

The Ethernet interface is usually used to provide network connectivity. It offers high speed and performance, but itself does not provide reliable transfer of data. To fix it we need to implement e.g. the TCP/IP stack which significantly increases the resources consumption on the interface side. In fact the interface should be equipped with an embedded PC or at least a microcontroller.

The PCIe interface offers high speed communication, but has limited possibilities to connect external boards. The "External PCIe" connectors are rarely available in typical computers. Theoretically the "ExpressCard" connector present in most modern laptop computers provides the PCIe bus (together with USB3.0), and there are solutions allowing to is as an External PCIe bus,[1] but such solution is still quite expensive.

Another disadvantage is a necessity to use an FPGA with PCIe support[2] or specialized PCIe bridge[3–5] with relatively high price.

The USB interface offers the following features:

- high speed interface designed for connection of external peripheral devices
- cheap cables, hubs and other infrastructure
- possibility to assure low price connectivity to FPGA chip

When we compare the USB interface with the previous two, it seems, that USB is the best interface for our controller.

3. CONNECTION OF FPGA TO USB

Unfortunately no FPGAs offer direct connectivity to USB. When connecting FPGA to the USB bus we have a few possibilities

- Connect FPGA directly, even though it is not fully USB compliant. This solution is however limited to USB1.1 with very low transmission speed of 1.5Mb/s.[6,7] Additionally, when using this approach, we still need to support the USB protocol, which is quite complicated. In fact we should provide the "soft CPU" in our FPGA to implement necessary procedures (e.g. device enumeration).
- Connect FPGA using the physical layer interface chip (PHY). There are IP cores for FPGA,[8] allowing us to connect it to typical USB2.0 PHY implementing the UTMI[9] interface. Unfortunately in this case we still need to implement USB protocol support in our FPGA.
- Connect FPGA via the specialized USB bridge. Such solution is better the the previous two, as we do not spend the FPGA resources to support the USB protocol. Additionally with prices of USB bridges comparable to prices of USB PHY chips, and with much better availability of the bridges this approach seems to be optimal.

4. SELECTION OF THE USB BRIDGE

Different firms offer chips allowing to connect the USB interface to local buses or interfaces, which can be used to establish connection between a FPGA chip and USB host. Probably the most popular are:

- CP210x family manufactured by Silabs.[10] Unfortunately those devices support only USB full speed – 12Mb/s.
- FX2LP family manufactured by Cypress.[11] Those device support the USB2.0 high speed communication, and multiple local side protocols (FIFO, I2C). These bridges include also fast version of 8051 microcontroller, however its use for high speed communication is quite limited.
- The USB bridges produced by the Future Technology Devices International Ltd. (FTDI).[12] Those devices offer both USB2.0 full speed and high speed. Additionally they offer multiple modes of operation, including the JTAG mode, which makes them very interesting solution for interfacing FPGA chips.

Finally we have decided to use the FT2232H chip manufactured by FTDI,[13] which provides our interface not only with the high speed FIFO, but also with JTAG mode, which may be used to configure and debug the interface FPGA.

5. SELECTION OF MODE IN FT2232H

The FT2232H chip offers wide choice of operating modes with different properties regarding their possible application in our interface

- FT245 Synchronous FIFO Interface Mode
  According to the manufacturer's data in this mode it should be possible to achieve throughput above 25MB/s. Analyzing the timing diagram itself, it seems, that the achievable throughput should reach value of 60MB/s, but probably this value is limited due to gaps between packets. Unfortunately the big disadvantage of this mode is the fact, that it utilizes resources of both communication channels. Therefore, if we switch on that mode in channel A, the channel B will be useless.

- FT245 Asynchronous FIFO Interface Mode
  According to the datasheet - the achievable throughput is up to 8MB/s, however analysis of the timing diagrams shows, that the minimum length of the read cycle is equal to 80ns, while the minimum length of the write cycle is 50ns. This mode even though slower than the Synchronous Fifo Mode, seems to be better suited to our need, as it leaves the second channel free for use in another communication mode.

- MPSSE Interface Mode
  In this mode we can use different specialized modes like SPI. I2C, JTAG, however the throughput is limited. Anyway this mode may be used in the second channel, e.g. in JTAG mode to provide programming and debugging functionality to the interface FPGA. What is important, the MPSSE-JTAG mode in FT2232H is well supported by Xilinx tools like Impact included in ISE suite[14] and ChipScope,[15] if used with the open source driver developed by Michael Gernoth.[16] It is also supported by the open source JTAG tool - UrJTAG.[17]

- MCU Host Bus Emulation Mode
  [18] In theory this should be the mode, which could allow us to directly control the local bus with 16-bit addresses and 8-bit data. According to the timing diagrams provided in the documentation, at highest speed the write cycle uses 10 cycles of 60MHz clock, while read cycle uses 8 cycles of 60MHz clock. This should assure the throughput of 6MB/s for write operations and 7.5MB/s for read operations (of course this throughput may be further reduced due to handshake and gaps between the packets). However in this mode we are forced to use the 8-bit data bus, and also the IORDY signal, intended to extend the read and write cycles has limited functionality.

- Other modes
  The other modes are dedicated for special kinds of interfaces, or offer too low throughput to be used in our interface:

  - Fast Opto-Isolated Serial Interface Mode
  - CPU-style FIFO Interface Mode
  - Synchronous and Asynchronous Bit-Bang Interface Mode

Finally we have decided to use channel A in the Asynchronous FIFO Interface Mode, sacrificing some speed to retain availability of the B channel which is used in MPSSE-JTAG mode.

The final architecture of our proposed interface is shown in the Fig. 2 Of course it is also possible to connect the FT2232H bridge directly to the FPGA in the developed system - in this case the IP core of our interface should be simply integrated with the IP cores implementing other functionalities in this FPGA.

6. COMMUNICATION PROTOCOL

The A channel of the USB bridge from the FPGA side is visible as a simple FIFO, which can only receive and transmit series of bytes. To transmit different commands and data, we have to create a communication protocol allowing to recognize particular commands.

In theory we need to transmit two commands - read and write. If the amount of address bits needed in the Local Bus is $N \cdot 8 - 1$, and if the width of the data bus is equal to $M \cdot 8$ bits, then we could simply encode those two commands as shown in the Table 1.

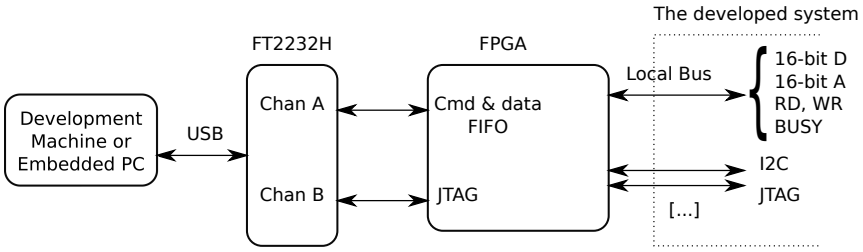

Figure 2. Block diagram of our interface.

Table 1. Encoding of commands for the simplest protocol with only two commands - read and write.

operation	command	response
Write	N bytes coding operation and address: 0, $A_{8\cdot N-2}$, ..., A_0 | M bytes of data: $D_{8\cdot M-1}$, ..., D_0	no response
Read	N bytes coding operation and address: 1, $A_{8\cdot N-2}$, ..., A_0	M bytes coding read data: $D_{8\cdot M}$, ..., D_0

Please note, that even such small details as order of address and data parts in the write command may affect performance of our interface. As soon as the FPGA receives the read/write bit and the address part, it may start to drive the address lines of the bus, giving more time for address decoders. Then, after the data part is received, the write strobe can be activated much faster.

Such simple encoding of commands may be quite efficient, but it still leaves some place for optimization's. Very often we may need perform writes to neighboring addresses, and reads from neighboring addresses. In both cases we can send only the starting address and the length of the block. Unfortunately, when we introduce such "block" commands, we need two bits to encode four possible commands (read, write, block read, block write).

Possible encoding of commands with $(N \cdot 8) - 2$ address lines , $M \cdot 8$ data lines and maximum length of the block transfer equal to 256 is shown in the Table 2:

Looking for the most efficient set of commands for our interface, we can add yet another combination - the "scattered block" read and write, which after the single command allow us to send the list of addresses to be read, or list of pairs (address, data) to be written.

Table 2. Encoding of commands for the protocol with added block commands.

operation	command	response
Write	N bytes – operation and address: 0, 0, $A_{8\cdot N-3}$, ..., A_0 | M bytes of data: $D_{8\cdot M-1}$, ..., D_0	no response
Read	N bytes – operation and address: 0, 1, $A_{8\cdot N-3}$, ..., A_0	M bytes coding read data: $D_{8\cdot M-1}$, ..., D_0
Block write	N bytes – operation and address: 1, 0, $A_{8\cdot N-3}$, ..., A_0 | Block length: L | $L \cdot M$ bytes of data: $D_{8\cdot M-1}$, ..., D_0	no response
Block read	N bytes coding operation and address: 1, 1, $A_{8\cdot N-3}$, ..., A_0 | Block length: L	$L \cdot M$ bytes coding read data: $D_{8\cdot M-1}$, ..., D_0

The above described list of commands may be further extended, as we can add different interfaces to our FPGA chip. It is possible e.g. to add the I2C interface, the SPI interface and the JTAG interface. Theoretically it may seem that in this way we unnecessarily duplicate the functionality of the FT2232H bridge, which supports those interfaces in the MPSSE mode. However there is a significant advantage of having those interface implemented independently.

In the MPSSE mode it is not possible to use the I2C, SPI and JTAG interfaces simultaneously (see the Table 3.6 in Ref. 13). Additionally, implementing those interfaces in our FPGA, we can provide them with additional "intelligence" implementing high level commands like "send the following N bytes to address 0x52 in the I2C interface".

Of course introduction of new commands, related to servicing of additional interfaces would result in further increase of number of bits needed to decode the command. To avoid it however we can simply reserve some special addresses for each interface.

For example, in 14-bit address space we can reserve the upper 0x3fff address for I2C access, so that writing of block of 13 bytes starting from this address will be interpreted as writing of series of bytes to the I2C interface, where the I2C address is defined by the first byte in the block, and data are the next bytes in the block. Such solution significantly improves speed of communication through the I2C interface.

Similarly the JTAG interface provided by the FT2232H in MPSSE-JTAG mode is well suited enough to configure relatively small FPGA in the interface itself, but to provide JTAG for FPGAs in the whole developed system we have created another, faster JTAG interface in our interface FPGA. As in case of the I2C interface we have reserved another address for this JTAG interface. Writing of a byte to this address sets the TMS and TDI lines accordingly, transmits the expected TDO state, and verifies the TDO line if requested, and finally generates the TCK pulse. Reading of this address returns information about the current state of the TDO line, and informs whether this line was in any TCK cycle after the previous reading in an incorrect state. Such functionality allows easy interfacing with the lib(X)SVF library[19]

The final, optional, improvement of our communication protocol includes changes increasing reliability.

The USB interface assures reliable transfer of data in the bulk mode (which is used, when communicating with the FT2232H), but the transferred data are visible simply as a stream of bytes. This creates a danger, that if two different applications will use our interface, in case of synchronization problems, commands sent by those applications may mix, leading to difficult to detect errors. The solution is to encapsulate of commands in frames, assuring reliable detection of the beginning and end of the command. Different schemes of encapsulation may be used, based e.g. on special "escape" characters and sequences, or on reservation of particular bits for marking of the start and end of the frame.

In our system we have decided to reserve the MSB bit in each byte for a "start of the frame" marker. This decreases the throughput by factor of $7/8$ but increases probability of detection of the corrupted commands. The above is especially important at the software development stage. We have also added the obligatory response for each command. Even after the write command the FPGA returns at least a single byte, with MSB set and with the status bit set depending on the status. Similarly, when reading, the first byte of the response has MSB set and contains the status bit. This allowed to add full wait states support with timeout detection on our Local Bus.

The cost of those modifications is that the address and data bytes are splited between different bytes when transferred through USB, but this can be very easily solved in our interface FPGA, and does not significantly complicate the PC software driving the interface.

7. SOFTWARE SUPPORT FOR THE INTERFACE

In case of interface, which is supposed to be used both for debugging and control applications, it is essential to provide broad software support.

Currently the PC side of our interface is serviced by a library written in C, which is responsible for building of USB packets containing encapsulated commands, and for unpacking of encapsulated responses. To communicate with the FT2232H chip we use the open source libftdi[20] library.

It is also possible to access our interface from the Python language in three different ways - either directly, using the python-ftdi module associated with the libftdi library, or via our library written in C, using the SWIG[21] generated wrapper, or finally via dedicated TCP/IP server.

To make our interface available for other environments, we have also developed the TCP/IP server which allows to send transmission commands via TCP/IP connection. This server allowed to access our system from Matlab environment running on remote Windows machine.

8. RESULTS

The developed interface was successfully implemented using the FT2232H chip connected to the XC3S50AN-TQG144 FPGA, and used to communicate with the data acquisition system[22] for GEM detectors during the development and debugging phase. Unfortunately, due to use of relatively small FPGA, not all proposed features could be implemented simultaneously. The finally used version was oriented on scattered block read and write transfers with support for JTAG and I2C interfaces. Because the interface was used for testing of prototype hardware and software, the most significant bit was used to mark the beginning of the command. As only 7 bits remained available in every USB-transmitted byte, the address (16-bit) and value (also 16-bit) pairs in write commands were encapsulated in 5-byte records, while addresses in read commands were encapsulated in 3-byte records, which decreased the throughput.

In the scattered block mode both read and write consumed 1.3 μs per operation. Considering the encapsulation of commands, it corresponded to 30Mb/s transfer rate through USB. Achieved transfer rate was significantly reduced by the wait states which have been added, to assure reliable transfer to the local bus, which was connected via the flexible ribbon.

Performed tests confirmed correct operation of the interface and its usefulness for controlling of the local bus, of the I2C interface, and of the JTAG interface.

9. CONCLUSIONS

Widely available USB bridges, supported with simple FPGA IP core allow to create efficient interface able to control the parallel local bus and other interfaces (e.g. I2C, JTAG).

The proposed interface may be implemented either as a separate unit (as in our prototype) or as an integral part of the FPGA based system.

Proposed interface may significantly increase comfort of development of simple FPGA based systems, which themselves do not require the VME or similar infrastructure.

The achieved operation speed is acceptable, even though it is not as high as in professional VME/USB interfaces.

Further improvement of speed can be probably achieved by utilization of the Synchronous FIFO Mode in FT2232H, or by use of USB3.0 compatible bridge.

REFERENCES

[1] "PE4L V2.1 (PCIe Adapter)." http://www.hwtools.net/adapter/PE4L%20V2.1.html.

[2] "PCI Express (PCIe) The High-Bandwidth Scalable Solution." http://www.xilinx.com/technology/protocols/pciexpress.htm.

[3] "PEX 8311, x1 Lane PCI Express Bridge, 21 x 21mm PBGA ." http://www.plxtech.com/products/expresslane/pex8311.

[4] "Semtech GN4121 PCI SIG Compliant Single Lane PCI Express Bridge." http://www.semtech.com/apps/product.php?pn=GN4121.

[5] "Semtech GN4124 PCI-SIG Compliant Four Lane PCI Express to Local Bus Bridge." http://www.semtech.com/apps/product.php?pn=GN4124.

[6] "USB 1.1 PHY." http://opencores.org/project,usb_phy.

[7] "USB 1.1 PHY (VHDL)." http://opencores.org/project,usb11_phy_translation.

[8] "USB 2.0 Function Core." http://opencores.org/project,usb.

[9] "USB 2.0 Transceiver Macrocell Interface (UTMI) Specification ." http://www.intel.com/technology/usb/download/2_0_Xcvr_Macrocell_1_05.pdf.

[10] "USB to UART Bridge." http://www.silabs.com/products/interface/usbtouart/Pages/usb-to-uart-bridge.aspx.

[11] "Usb hi-speed peripherals ez-usb fx2lp." http://www.cypress.com/?id=193.

[12] "FT Series ICs - USB Slave Converters." http://www.ftdichip.com/Products/ICs.htm.
[13] "FT2232H Dual High Speed USB to Multipurpose UART/FIFO IC." http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT2232H.pdf.
[14] "ISE Design Suite." http://www.xilinx.com/products/design-tools/ise-design-suite/index.htm.
[15] "ChipScope Pro and the Serial I/O Toolkit." http://www.xilinx.com/tools/cspro.htm.
[16] Gernoth, M., "XILINX JTAG tools on Linux without proprietary kernel modules." http://rmdir.de/~michael/xilinx/.
[17] "UrJTAG - Universal JTAG library, server and tools." http://urjtag.org/.
[18] "Command Processor For MPSSE and MCU Host Bus Emulation Modes." http://www.ftdichip.com/Support/Documents/AppNotes/AN_108_Command_Processor_for_MPSSE_and_MCU_Host_Bus_Emulation_Modes.pdf.
[19] "Lib(X)SVF - A library for implementing SVF and XSVF JTAG players." http://www.clifford.at/libxsvf/.
[20] "libftdi - ftdi usb driver with bitbang mode." http://www.intra2net.com/en/developer/libftdi/.
[21] "Simplified wrapper and interface generator." http://www.swig.org/.
[22] Kasprowicz, G., Czarski, T., Chernyshova, M., Czyrkowski, H., Dabrowski, R., Dominik, W., Jakubowska, K., Karpinski, L., Kierzkowski, K., Kudla, I. M., Pozniak, K., Rzadkiewicz, J., Salapa, Z., Scholz, M., and Zabolotny, W., "Readout electronics for the GEM detector," *Proc. SPIE* **8008**, 800080J (2011). http://dx.doi.org/10.1117/12.905492.