Bulletin of the *Transilvania* University of Braşov • Vol. 7 (56) No. 2 - 2014
Series I: Engineering Sciences

ENHANCING FPGA I/O COMMUNICATION USING WEB SERVICES

O.M. MACHIDON[1] **F. SANDU[1]**

Abstract: *FPGAs (Field Programmable Gate Arrays) are increasingly being used for both commercial and educational applications that take advantage of their re-configurability and computational power. The practice of offloading intensive computations to the FPGA where they can benefit from the computing parallelism and hardware acceleration is burdened by the need to overcome I/O bottlenecks generated by the lack of a fast and easy-to-use interface for transferring data. This paper presents the design and implementation of a service-based solution for interconnecting a PC and an FPGA over the Ethernet that offers a flexible, high-speed link for transferring data using an easy to use web interface.*

Key words: *FPGA, IP Core, web service, SOA, Ethernet.*

1. Introduction

FPGA devices have become very popular solutions for developing digital embedded systems, being an ideal platform for both commercial and educational applications due to their re-configurability, high flexibility and processing power. Thus FPGAs are increasingly being used for deploying compute-intensive tasks and for prototyping hardware systems.

A general challenge for FPGA designers is overcoming the I/O bottlenecks generated by the relatively difficult and slow communication between the chip and the external environment - typically a PC that offloads intensive computational processes to the device. FPGA chips have become more and more powerful lately - gaining in reconfigurable hardware resources, higher clock frequencies, optimized power consumption. Consequently a general trend is currently underway to use the FPGAs as auxiliary devices connected to the PC, where specific computational tasks can be offloaded. Such tasks can then benefit from the FPGAs computing parallelism and hardware acceleration, leading to an overall speedup of the applications. This raises the key issue of data transfer to and from an FPGA device connected to a general-purpose computer.

There are several interconnect technologies available for implementing the communication between and FPGA and PC: PCI Express, Ethernet, RS232 - which are available on nearly all recent FPGA platforms, and other technologies like Bluetooth, Wi-Fi, Infrared, that are not so widespread and usually imply connecting adaptors or extension cards.

PCI Express is without a doubt the fastest way to transfer data between FPGA and PC, but also the most expensive and one that needs dedicated PCI Express driver on the

[1] Dept. of Electronics and Computers, *Transilvania* University of Braşov.

PC, and an FPGA with PCI Express capabilities - where a specialized (usually commercial and expensive) PCI Express IP Core needs to be instantiated [7].

RS232, while easy-to-use, is outdated and cannot cope with the performance and speed requirements of today's digital applications.

The Ethernet solution remains the most accessible for interconnecting the two devices, however it brings on its own complexity and implementation issues [1]. The majority of existing Ethernet transfer solutions implement either the UDP/IP or TCP/IP protocol stack in the FPGA hardware for communicating with the PC. These solutions, although viable and with good performance, are suitable for complex applications that need extended network capabilities, and are also quite difficult to integrate and use in larger designs.

2. Objectives

This paper presents the design and implementation of a service-based solution for interconnecting a PC and an FPGA over the Ethernet that offers a flexible, high-speed link for transferring data using an easy to use web interface.

We have chosen Ethernet as the underlying technology for our development since most FPGA platforms contain an EMAC (Ethernet Media Access Controller) module that allows direct, data link level access to the PHY (physical layer) on-board device.

For successfully implementing this solution, our efforts have targeted:

- The design and implementation of an EMAC controller as an HDL IP Core for Ethernet communication.
- The development of a PC software application communicating with the embedded EMAC controller on the FPGA board over the Ethernet, using a proprietary protocol over the data-link layer.
- Abstracting away the complexity of using the software application by implementing a web application for transferring data to/from the FPGA using a web service and a JSP (Java Server Pages) web page as an interface to the user.

3. Design and Implementation

The communication solution developed, illustrated in Figure 1 above, is composed of two main parts: the embedded hardware component - running on the FPGA board - representing the *EthController* IP Core (that controls the on-board EMAC interfacing the ETH PHY) implemented in Verilog HDL (Hardware Description Language), and the software component (RAW socket application - responsible for the Ethernet communication with the FPGA, and Web application for exposing the communication on a web-based interface using a web service).

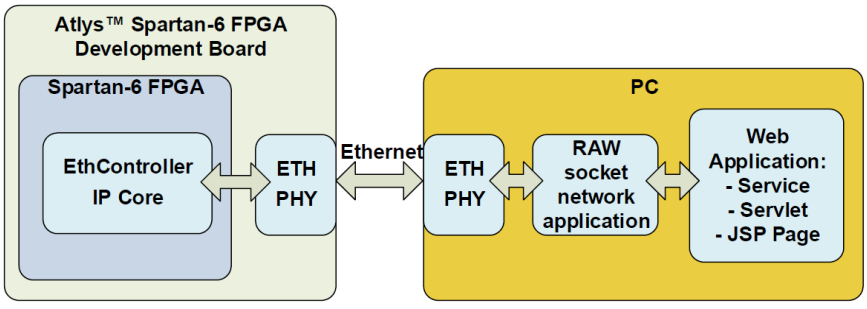

Fig. 1. *Communication system - general overview*

3.1. EthController IP Core

The hardware implementation is based on a Atlys Spartan-6 FPGA Development Board that integrates a Xilinx Spartan-6 LX45 FPGA and a Marvell Alaska Tri-mode Ethernet PHY paired with a Halo HFJ11-1G01E RJ-45 connector.

The ETH PHY's timing is controlled by an on-board 25MHz oscillator. The IP Core was implemented in Verilog HDL, an architectural view is available in Figure 2 below.

EthReceive Module

This module is responsible with receiving data or acknowledgement packets sent by the raw socket software application running on the PC. It takes the following input signals from the ETH PHY: clock, data bus and data validation, and provides to the EthSend module the index of the received packet in order to send back the corresponding acknowledgement packet.

The internal architecture of the EthReceive module is basically composed of a checksum compute module (CRC32_D8) and internal logic. The logic contains several counters and registers for discriminating inside the received packet flow. The packet inspection is being performed "real-time" which means that, as data is being received, it is also interpreted and actions are being taken based on it: the packet is validated, the length is extracted and the actual data is being grouped on a 16 bit wide bus and forwarded by an output port to a synchronization FIFO.

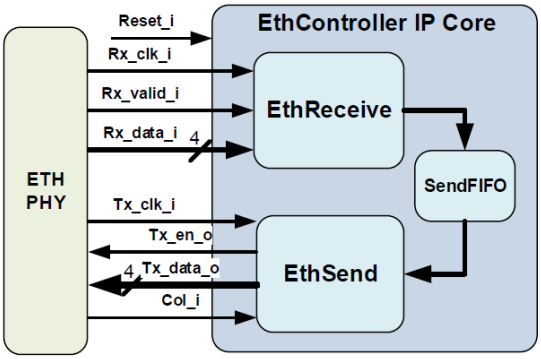

Fig. 2. *EthController IP Core internal architecture and interface with the PHY*

When the module's logic detects *rx_valid_i signal=High*, a packet is about to be received. The packet structure is presented in Figure 3. At first, the validity of the packet is checked - the existence of the Preamble and SFD (Start of Frame Delimiter) fields at the beginning of the incoming data. After confirming that the packet is valid, the module proceeds to checking the packet type - either confirmation packet or one containing user data - by analysing the first 4 bytes of the data frame.

Afterwards the remaining data is interpreted: the following 2 bytes represent the packet number, and another 2 bytes - the length of the contained data.

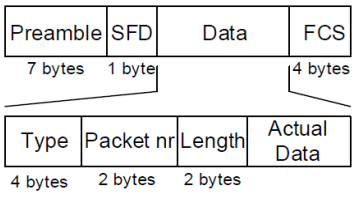

Fig. 3. *Diagram of an Ethernet packet from the communication PC-FPGA*

The user data is received as one nibble per clock and is read into a 16 bit shift register that once every 16 bit are read, is outputted together with a validation signal to be further used in the FPGA logic.

While data is being read from the received packet, the CRC is computed on-the-fly in order to validate its integrity. This is accomplished using the CRC32_D8 module instantiated in the EthReceive module. Once the packet has been read completely, if the CRC32_D8 module's output value is 0, the packet is validated, and its index number is written in the synchronization FIFO to be used by the EthSend module for sending the acknowledgement packet, otherwise it is ignored and discarded.

EthSend Module

This module is responsible with sending packets - either for acknowledgement of the received ones or containing data - from the FPGA system to the PC. It reads data from the FIFO module, when available, and encapsulates it into a packet that is sent through the PHY on the Ethernet interface. It uses several counters and registers, together with the CRC module for building the packet "on-the-fly", outputting on each clock cycle a nibble on the data bus.

When acknowledgement information - the number(s) of the received packets that need to be confirmed - is available as output data from the FIFO module, a new Ethernet frame is sent containing the specific packet number. The same process is performed using standard user data read from the FIFO module.

The sent packet structure is the same as in Figure 3. The module "builds" the packet according to the IEEE 802.3 protocol standard [5], starting with the Preamble and SFD fields and continuing with the data field. This field has a first byte indicating the type of packet (acknowledgement or containing ordinary data), followed by either the confirmed packet's number, or the length followed by the actual data. The data before being sent is stored in a shift register that is shifted 4 bits every clock period, since the Ethernet output data bus is 4 bits in width. The module's logic ensures the fact that in the case of shorter packets, an extra padding is appended (byes having a zero value) in order to guarantee a minimum length of 64 bytes/packet.

The last field of the packet is the FCS (Frame Check Sequence), with a length of 4 bytes, containing the 32 bit CRC value computed on-the-fly based on the packet's data. The CRC is computed using an instance of the same CRC32_D8 module.

The Ethernet 802.3 protocol regulates the standard minimal interval needed to exist between two consecutive sent packets. For our current case, an Ethernet 100Mbit/s transmission, this interval has a value of 960ns, which equals with 24 clock periods. This is ensured by interrogating a timer which is counting the clock periods after every sent packet.

Guaranteeing this interval, together with the absence of a possible collision on the Ethernet interface (signalled by the *col_i* signal received from the PHY), and receiving confirmation for the last packet sent are the three conditions that enable sending a new packet. If any of them is false, the module waits until all requirements are met before sending a new frame.

The packet acknowledgement mechanism was implemented in order to avoid congestions and to be able to guarantee the integrity of the transmitted data. The same counter mentioned above is used in the EthSend module to count 25000 clock periods (the equivalent of 1ms), an interval in which an acknowledgement for the last sent packet is expected. If not received, the same packet is re-transmitted - the data

being read from ResendFIFO, a structure that stores the content of the last sent packet.

3.2 Software Applications

The software applications designed to run on a PC with a Linux operating system are:

- a C network application that implements the actual Ethernet communication with the FPGA board;
- a Web Application composed of a web service, a Java servlet, and a JSP web page; it represents a service-based middleware that abstracts away the complexity and functional details of the C application and communication flow with the FPGA by exposing it on an easy-to-use web-based interface.

C network application

This application implements the communication between the Atlys FPGA development board and the PC over the Ethernet using RAW sockets. The program opens two RAW sockets: one for sending packets and another for receiving them. Thus, a bidirectional data exchange flow is supported: packets containing user data and the corresponding acknowledgements are being sent to/from the FPGA board.

Since we have implemented our own proprietary protocol, the communication had to be implemented at the data-link layer (level 2 from the OSI stack), so the software application is working with RAW sockets that allow receiving the entire Ethernet frame [2]. This is a key feature of our application, only such an approach allowing the application running on the PC to receive packets with customized protocols.

The communication domain of the socket opened for sending/receiving packets is set to PF_PACKET - which allows working with RAW packets at the data-link layer. The socket's type is SOCK_RAW and the specified protocol, ETH_P_ALL, which enables packet traffic regardless of protocol [3].

In order to guarantee the integrity of the data and to manage possible congestions, specific measures were implemented in the application, which work together with the ones developed in the EthController IP Core.

When sending data to the FPGA system, after each sent packet the application waits for an acknowledgement a certain interval, and tries to re-send the same packet three times. If the acknowledgement has still not been received, the application terminates. A diagram illustrating the functionality of the application - when sending data to the FPGA - is shown in Figure 4.

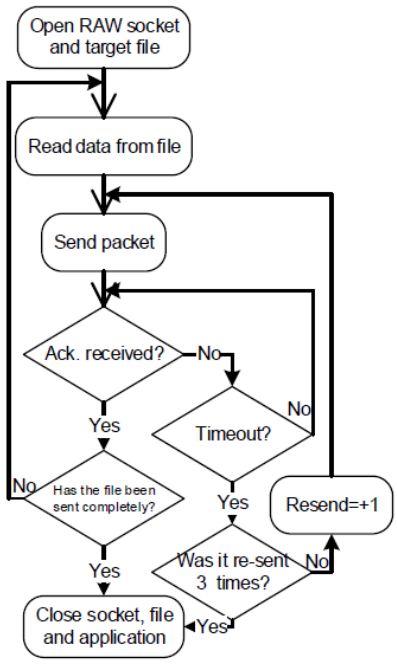

Fig. 4. *C network application functionality*

When receiving data from the FPGA the flow is simpler, since the incoming data is stored in a file until and EOF (end of file) character is received in a packet's data and each packet is confirmed when received.

Web applications

The web application has three components: the web service that acts like a middleware between the user and the underlying network application, a JSP web page representing a friendly, easy to use web-based user interface, and a Java servlet which is an intermediate entity between the two, facilitating the data flow amongst them.

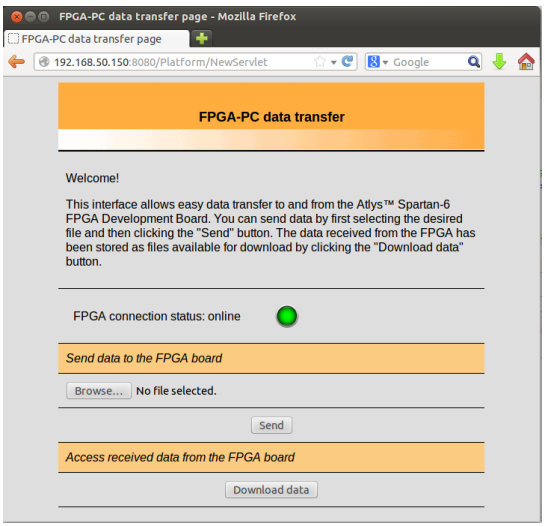

Fig. 5. *Screen capture showing JSP page web user interface for transferring data*

The servlet uses SOA (Simple Object Access Protocol) for communicating between the service and the JSP page [4]. It acts like an extension to the service by enhancing its functionality [6]; the HTML data and files from the JSP page are encapsulated by the servlet into a SOAP message that is sent to the service in order for it to be sent further away to the FPGA system. Any data received from the system follows a similar path: it is sent inside SOAP messages by the service to the servlet, and then it becomes available to the user on the JSP page.

The web service was implemented in Java and runs on a Glassfish 4.0 Server instance. It contains a method that interfaces the RAW socket network application by automatically creating and configuring a script-enabled working environment for executing the C application. Through this application it is managing the incoming/outgoing data flow, in order to send and receive data from the FPGA board.

4. Validation and Results

The testing and validation of the communication solution started by simulating the hardware design: the EthController IP Core. Individual simulations were performed targeting each modules (EthSend, EthReceive, SendFIFO), and also an overall simulation of the entire design (Figure 6).

A test bench module was developed for generating input signals (emulating the PHY): transmit and receive clocks (25MHz each, asynchronous), active-high reset, input data (4 bit bus) and data validation signal. Thus, the test bench generates packets received by EthReceive and verifies that the corresponding acknowledgements are being sent back by EthSend.

Also, this simulation environment allows an in-depth visual verification of the Ethernet frames sent by EthSend (valid structure - according to the 802.3 standard's specifications, correct CRC etc.).

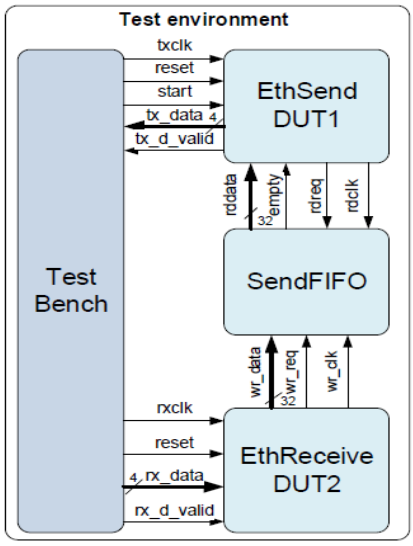

Fig. 6. *EthController simulation test environment*

Besides simulation, the hardware design was also tested while running on the FPGA board. The insertion of specific faults was implemented using the switches and buttons available on the board in order to test the behaviour of the design in critical situations that may happen during run-time:

- the possibility to turn off the sending of acknowledgement packets by the FPGA using an on-board switch for testing if the resending operation is working properly.
- sending packets that do not comply with the proprietary protocol's specifications to see if it causes malfunctions in the RAW socket application.

A benchmarking of the communication solution was made in order to evaluate the performance in terms of data transfer speed and also to stress the software and hardware components by emulating high traffic load scenarios. This was accomplished by sending a continuous data flow and monitoring the communication link using the IPTraf Linux utility. The results are shown in Table 1 below.

Table 1

IPTraf utility statistics of a data transfer from PC to FPGA

Metric	Value
Incoming packets	328655
Incoming bytes	248463 K
Outgoing packets	328655
Outgoing bytes	331256 K
Total packets	657310
Total bytes	579719 K
Incoming rates	2935.6 packets/s
	17754.5 Kbits/s
Outgoing rates	2935.4 packets/s
	23670.0 Kbits/s
Total rates	5871.0 packets/s
	41424.5 Kbits/s

The figures above show that during a data transfer from the PC to the FPGA, the speed achieved was approx. 2.9 MB/s, while the acknowledgement packets were sent back to the FPGA system at a rate of 2.2 MB/s. This transfer speed can be improved, since it is limited by the acknowledgement mechanism that was implemented, that is waiting for an individual confirmation for each sent packet before sending the next one. We are considering as a future development to substitute this mechanism with a sliding-window type that would result in a potential speed-up.

In Table 2 a summary of the FPGA utilization is presented, showing that the EthController IP Core occupies less than 2% of the resources available in terms of slices (the Xilinx technology basic unit composed of LUTs and FFs), and only 8% of the IOBs (Input/Output Buffers - the effective number of FPGA pins used). These results, indicating a very low area

and IO usage, show that EthController IP Core can easily be integrated into a larger design since it does not need many resources and provides an important enhancement regarding I/O communication of the FPGA.

Table 2
Spartan 6 Device Utilization Summary

Slice Logic	Utilization	
	Nr.	%
Number of Slice Registers	349	1
Number of Slice LUTs	378	1
Number of occupied Slices	145	2
Number of MUXCYs used	108	1
Number of bonded IOBs	19	8
Number of RAMB16BWERs	1	1
Number of BUFG/BUFGMUXs	3	18

The low resource usage is an advantage over the existing IP protocol based solutions [1], which makes our development ideal for FPGA hardware applications needing a basic connectivity with a PC for data transfer.

5. Conclusions

In this paper we have presented a new solution that implements an efficient communication interface between a PC and an FPGA over the Ethernet. The resource utilization is low - the IP Core running on the FPGA needs less than 2% of the available resources - and the transmission rates are around 3MB/s.

This communication is further enhanced by the integration of a web service and a JSP page acting as a user interface that abstracts the complexity, lower-level implementation and architectural details of the RAW socket application that operates on the data-link layer.

Future work will focus on improving the transfer speed by implementing an improved acknowledgement mechanism. Also, we are planning to enhance the EthController IP Core by implementing the TCP/IP stack, thus extending its application range.

Acknowledgement

This paper is supported by the Sectoral Operational Programme Human Resources Development (SOP HRD), ID134378 financed from the European Social Fund and by the Romanian Government.

References

1. Alachiotis, N., et al.: *Efficient PC-FPGA Communication over Gigabit Ethernet.* In*:* IEEE 10th International Conference on Computer and IT (CIT), Bradford, UK, 2010, p. 1727-1734.
2. Donahoo, M., Calvert, K.: *TCP/IP Sockets in C: Practical Guide for Programmers*. San Francisco, CA. Morgan Kaufmann, 2009.
3. Hall, B.B.: *Beej's Guide to Network Programming: Using Internet Sockets*. Riverside, CA. Jorgensen Publishing, 2011.
4. Hansen, M.: *SOA Using Java Web Services*. Upper Saddle River, NJ. Pearson Education, 2007.
5. IEEE 802.3™-2012: *IEEE Standard for Ethernet*. Available at: http://standards.ieee.org/about/get/802/802.3.html. Accessed: 10.08.2014.
6. Perry, B.: *Java Servlet & JSP Cookbook*. O'Reilly Media, Inc., 2004.
7. Xilinx: *Bus Master DMA Performance Demonstration Reference Design for the Xilinx Endpoint PCI Express Solutions*. Available at: http://www.xilinx.com/support/documentation/application_notes/xapp1052.pdf. Accessed: 28.08.2014.