A Gigabit UDP/IP Network Stack in FPGA

Fernando Luis Herrmann, Guilherme Perin, Josue Paulo Jose de Freitas, Rafael Bertagnolli and Joao Baptista dos Santos Martins

Federal University of Santa Maria (UFSM) - Microelectronics Group (Gmicro)
Post-Graduate Program in Informatics (PPGI)
Av. Roraima n. 1000, Santa Maria, Rio Grande do Sul, Brazil
Email: (herrmann, guilhermeperin, josue.freitas, rafaelbertagnolli)@mail.ufsm.br, batista@inf.ufsm.br

***Abstract*—This paper presents a proposal of a Gigabit UDP/IP network stack in FPGA, which is the stack of the widely used in VoIP and Video-conference applications. This network node implements the Network, Transport and Link Layer of a traditional stack. This architecture is integrated and developed using Xilinx ISE tool and synthesized to a Spartan-3E FPGA. We show architecture details, timing and area results of a practical prototyping. Also, we compare our prototype and results with other works in terms of area (Xilinx slices), speed (MHz), maximum Ethernet frame length (bytes) and maximum Ethernet speed (Mbps). Comparing to these works our architecture obtained a intermediate solution in area and is the best implementation in terms of speed (MHz).**

***Index Terms*—UDP/IP, network stack, FPGA.**

I. Introduction

Nowadays, the great need of communication in society has collaborated to appear news forms of communication, that are more accessible and lower cost, for example Voice over IP (VoIP) or video conference. But, in a microprocessor of general purpose, this applications compete equally in processing time with other applications, causing a overload in the processing. In order to solve this problem, solutions implemented in dedicated hardware, ASICs or FPGAs become available. This solutions allow that part of the processing, instead of being realized by the microprocessor of general purpose, now can be executed by a dedicated hardware.

The UDP/IP protocol has applications on audio and video streaming of VoIP and Video Conference communications. Being real times conversations mode, the UDP protocol employment is explained by the need of low delay datagrams tranfer, due to unrealiable service [1] [2].

Some hardware UDP/IP stacks have already been realized. In [3], the author describes an analysis of FPGA-based UDP/IP stack parallelism for embedded Ethernet connectivity. In [4] an analysis of the TCP/IP sub-functions are made and the work describes the performance-critical functions that can be accelerated in FPGAs, how these sub-functions may be implemented and what speed-up gains that can be achieved. [5] Shown a RTP/UDP/IP protocol in Virtex FPGAs to accelerating VoIP applications. [6] Proposed five design guidelines and a corresponding architecture in TCP/IP Offload Engine (TOE). [7] Propose an implementation of UDP/IP protocol stack on FPGA and its performance evaluation.

This work presents a Gigabit UDP/IP network stack implementation in FPGA and makes a comparison with other existing works. For integration with the Application Layer, we also propose data and configuration packets formats. This paper is organized as follows. Section II presents an overview of the OSI model layers. The proposed implementation is detailed on Section III, while evaluation and results are exposed in Section IV. Finally, Section V presents our conclusion.

II. Network stack

The UDP/IP protocol is part of the Open Standards Interconnect (OSI) model. OSI is a theoretical model and is used to describe the behavior of a network and also to describe networking issues. The OSI model consists of seven layers and the layers are named (starting from the highest layer): Application, Presentation, Session, Transport, Network, Link and Physical Layer. From a TCP/UDP/IP viewpoint the Session and Presentation Layers are often included in the Application Layer. The OSI layers are frequently referred in this paper, but it's not further explained. For a detailed description of the layers and protocols, see [2].

A. Link Layer

As for serial links, the Link Layer provides data exchange between neighboring computers as well as data exchange between computers within a local network. For the Link Layer, the basic unit of data transfer is the data link packet frame. A data frame is composed of a header, payload, and trailer.

A frame carries the destination link address, source link address, and other control information in the header. The trailer usually contains the checksum of the transported data. By using the checksum, we can find out whether the payload has been damaged during transfer. The Network Layer packet is usually included in the payload.

B. Network Layer

The Network Layer is responsible for establishing the route to be used between the originating and destination computers. End-to-end reliability across several physical links is more of a function of the Transport Layer.

The basic unit of transfer is a datagram that is encapsulated in a frame. The datagram is also composed of a header and data field. Trailers are not very common in network protocols.

978-1-4244-5091-6/09/$25.00 ©2009 IEEE

The datagram header, together with data (Network Layer payload), creates the payload or data field of the frame.

The Network Layer is used to establish communications with computer systems that lie beyond the local LAN segment. It can do so because it has its own routing addressing architecture, which is separate and distinct from the Link Layer. Such protocols are known as routed or routable protocols, for example Internet Protocol (IP) [8]. The Internet Protocol (IP) is the most important protocol of the Network Layer. IP attempts to deliver messages to the destination, which is selected by a unique IP address [9].

C. Transport Layer

The Network Layer make the connection between two remote computers easiear. As far as the Transport Layer is concerned, it acts as if there were no modems, repeaters, bridges, or routers along the way. The Transport Layer relies completely on the services of lower layers. It also expects that the connection between two computers has been established, and it can therefore fully dedicate its efforts to the cooperation between two distant computers. Generally, the Transport Layer is responsible for communication between two applications running on different computers [9].

In this case, the basic transmission unit is the segment that is composed of a header and payload. The Transport packet is transmitted within the payload of the network packet.

The Transport Layer provides end-to-end reliability by having the destination host communicate with the source host. The idea here is that even though lower layers of protocols provide reliable checks at each transfer, the end-to-end layer double checks to make sure that no machine in the middle failed [10].

The Transmission Control Protocol (TCP) is the most used protocol of Transport Layer which gives a connection-oriented communication with reliable data delivery, duplicate data suppression and flow control. Another Transport Layer protocol is the User Datagram Protocol (UDP), which provides an unreliable and connectionless communication service. However, UDP is very effective when TCP is not suited for the application needs, e.g. for real-time applications like audio and video or in applications where low latency and low delay is preferred over reliable data delivery [11].

III. Implementation

The hardware UDP/IP stack core that's described in this paper is shown in Figure 1. Figure 1(a) showed the architecture of a traditional TCP/IP stack. In Figure 1(b), as the shadowed area, is showed the UDP/IP stack which are entirely implemented in the FPGA. The Layers, Transport, Network and Link in UDP/IP stack are designed using Verilog and VHSIC Hardware Description Language (VHDL).

In our implementation the UDP protocol was implemented in the Transport Layer while TCP packets are passed directly to the Application Layer, to be processed by software.

For Network Layer the Internet Protocol version 4 (IPv4) is used, which gives a more area-effective design compared to the more recent IPv6 protocol.

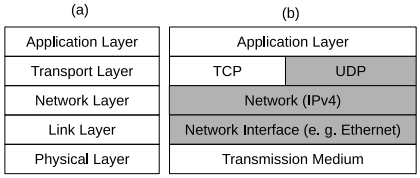

Fig. 1. (a) Architecture of a traditional TCP/IP stack. (b) Architecture used by this work and which layers are implemented. Redraw from [1].

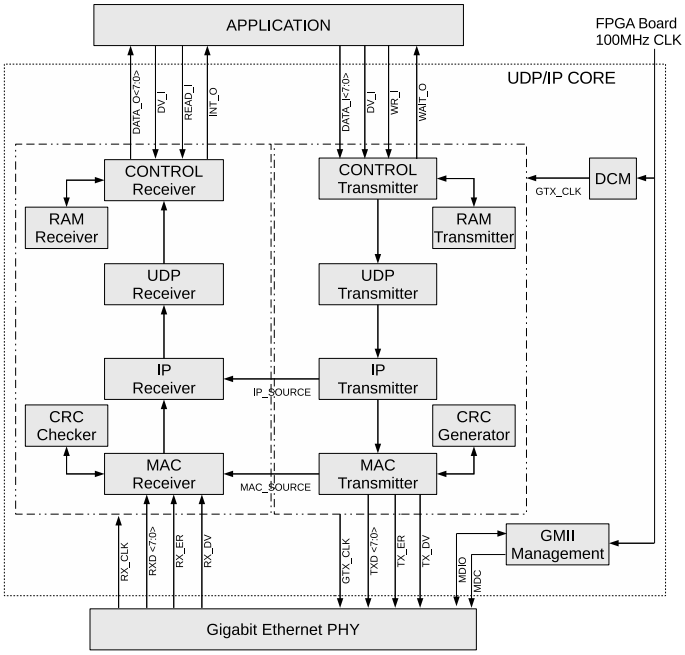

Fig. 2. Block diagram of the UDP/IP stack

The hardware UDP/IP stack implementation use the design structure which is showed in Figure 2. This implementation is full-duplex because the Transmitter and the Receiver works simultaneous and independent. The subsections below provide a detailed description of the functionality of each block.

A. Control Transmitter/Receiver

These blocks provide the communication with the Application Layer. The Control Transmitter receives the packet from the application and stores it in the RAM Transmitter for sending forward to the block UDP Transmitter.

The Control Receiver writes the packet from UDP Receiver in RAM Receiver and sends it to the application layer.

B. UDP Transmitter/Receiver

These blocks represent the Transport Layer and manage UDP packets. If the packet is a TCP, these two blocks just send the packet out. The block UDP Transmitter encapsulates the packet with the UDP header and sends out to block IP Transmitter.

The block UDP Receiver checks the packet and sends it to block Control Receiver without UDP header information.

C. IP Transmitter/Receiver

Represents the Network Layer manage IPv4 packets receiving and sending process. The IP Transmitter block calculates the checksum and encapsulate the packet with the IP header.

The IP Receiver block verifies the packet checksum and the destination IP-address. Only IP-address that matches the core's IP-address and broadcast IP-addresses are accepted and send to block UPD Receiver. If the packet check fails the packet will be rejected.

D. MAC Transmitter/Receiver

These blocks represent the Link Layer and manage the outcoming and incoming packets. The MAC Transmitter sends packets to the PHY. At the beginning, the preamble is sent, where the last nibble is a start of frame delimiter. The MAC transmitter then puts out the transmit packet to the PHY data bus and sets control signals. Each byte is sent to the CRC generator, which progressively calculates the CRC. When the packet end is reached the calculated 32-bit CRC is sent.

The MAC Receiver will check for a new packet (preamble from Ethernet PHY). Once a new packet is detected the CRC checker is notified and progressively calculates the checksum. When the end of frame is signaled from the Ethernet PHY the CRC check will be completed and the destination MAC-address will be verified. Only MAC-addresses that matches the UDP/IP stack MAC-address and broadcast MAC-addresses are accepted. If the packet check fails the packet will be rejected.

E. RAM Transmitter/Receiver

These blocks just temporarily store the packets and both has 1500 bytes in size.

F. CRC Checker/Generator

These blocks are identical and progressively calculate the Cyclic Redundancy Check (CRC). It uses the CRC32 polynomial for Ethernet. The polynomial is shown below:

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} +$$
$$X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X^{1} + 1$$

A 32-bit CRC provides error detection in the case where line errors (or transmission collisions in Ethernet) result in corruption of the MAC frame. Any frame with an invalid CRC is discarded by the MAC Receiver without further processing. The MAC protocol does not provide any indication that a frame has been discarded due to an invalid CRC. The Link Layer CRC therefore protects the frame from corruption while being transmitted over the physical medium.

G. DCM

This is a Digital Clock Manager (DCM) and it implements a Digital Frequency Synthesizer (DFS) to generate a 125MHz clock, see pin GTX_CLK in Figure 2. The GTX_CLK clock is used by all the Transmitter blocks and also by the Gigabit Ethernet PHY.

H. GMII Management

This block manages the Gigabit Media Independent Interface (GMII) to full-duplex and 1000Mbps operations using the Management interface I/O pin (MDIO). This block also implements a DFS to generate a 5MHz clock for the Management Interface Clok pin (MDC). The GMII Standard interface is defined in IEEE Standard 802.3 [12].

I. Overall behaviour

The operation of the system is based on the exchange of configuration and data packets between the Application and UDP/IP stack, see Table I.

TABLE I
PACKETS TYPE

Packet	Type	Value (HEX)
UDP	Data	11
TCP	Data	06
MAC Source	Configuration	81
IP Source	Configuration	82
MAC Gateway	Configuration	88

The configuration packets are send by the Application to the UDP/IP stack and are used only for configure the IP Source, MAC Source and the MAC Gateway. The structure of this three configuration packets is showed in the Table II.

TABLE II
CONFIGURATION PACKETS

Type (8 bits)	Configuration
81	MAC Source (48 bits)
82	IP Source (32 bits)
88	MAC Gateway (48 bits)

The data packets are send by the Application to UDP/IP stack which in turn sends data packets to the Application. There are two data packets, TCP and UDP, and it has the structure as showed in Table III. The Table III shows the structure of the data packets that are send by the Application to the UDP/IP stack. The data packets that coming from the UDP/IP stack to the application has the same structure, but instead the Destination IP address it has the Source IP address.

TABLE III
DATA PACKETS - APPLICATION TO UDP/IP STACK

Bits	7 - 0	23 - 8	39 - 24
Field Name	Packet Type	Packet Length	Source Port
Bits	55 - 40	87 - 56	(...)
Field Name	Destination Port	Destination IP address	Data

The Figure 3 shows how the Application must be send the packets to the UDP/IP stack. This Figure shows the sending of a MAC Gateway configuration packet.

The Figure 4 shows how the UDP/IP stack send a packet to the Application. The pin int_o indicates that the UDP/IP stack has an packet available. Also, this Figure shows the a receiving of a UDP packet.

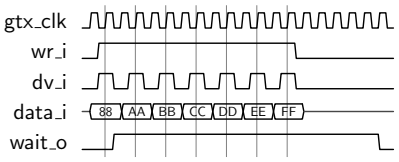

Fig. 3. Timing diagram - Application to UDP/IP stack

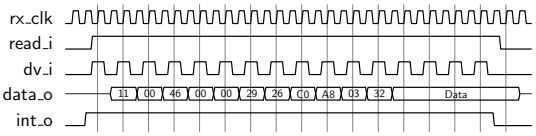

Fig. 4. Timing diagram - UDP/IP stack to Application

IV. Evaluation and Results

The UDP/IP stack was developed under Xilinx ISE 10.1 and passed through procedures of "Synthesis" and "Place and Route" for the Xilinx Spartan 3E, XC3S500e-4FG320 FPGA. Three simulation sub procedures were performed: "Functional", "Post Synthesis" and "Post Place and Route". Input values were supplied by macro files and test benches. Test benches also provided an overall report which includes signal comparison and signal sequence comparison, incoming TCP and UDP packets.

TABLE IV
Projects comparison

Projects	Xilinx Slices	Speed (MHz)	Max. Frame Length(bytes)	Max. Ethernet Speed(Mbps)
Our implementation	1580	125	1518	1000
Löfgren "Minimum"	517	90.7	256	100
Löfgren "Medium"	1022	60.3	256	100
Löfgren "Advanced"	1584	105.6	1518	1000
Dollas	1557	77	—	100

We compare our work in terms of area (Xilinx Slices), speed (MHz), maximum Ethernet frame length (bytes) and maximum Ethernet speed (Mbps) with other implementations and we get the results that can be seen in the Table IV. The chosen implementations were Löfgren "Minimum", Löfgren "Medium" and Löfgren "Advanced" of [3] and Dollas in [13]. [13] Presents a complete TCP/IP stack implementation, therefore only related data to UDP/IP implementation were observed.

In occupied area terms, our implementation can be considered an intermediate solution. In speed terms our work presented the best performance. For the maximum Ethernet frame length only our implementation and Löfgren "Advanced" are capable to manage frames with 1518 bytes. Löfgren "Minimum" and Löfgren "Medium" just manage frames with maximum 256 bytes. Dollas doesn't provide information about maximum Ethernet frame length. For the maximum Ethernet speed only our implementation and Löfgren "Advanced" are support 1000Mbps connections. Löfgren "Minimum", Löfgren "Medium" and Dollas just support 100Mbps connections.

For example, if we compare our implementation with Löfgren "Minimum", we can observe that our work occupies three times more area than Löfgren Minimum and just 35 MHz better in speed, but our implementation is capable to processing frames with 1518 bytes and supports 1000Mbps connections while Löfgren "Minimum" only processing frames with 256 bytes and works just with 100Mbps connections.

V. Conclusion

This paper shows a Gigabit UDP/IP network stack in FPGA. We present a UDP/IP stack that is successfully implemented and verified in Xilinx Spartan 3E. The hardware UDP/IP network stack used 1,580 Xilinx slices of Xilinx Spartan 3E FPGA and its maximum frequency operation is 125MHz.

Also, we present a comparison with other four works, in terms of area (Xilinx slices), speed (MHz), maximum Ethernet frame length (bytes) and maximum Ethernet speed (Mbps). In occupied area, our implementation is an intermediate solution. But, in speed our work obtained the best results among the compared works. Depending on the application purposes, is well accepted more few elements in area terms to achieve a better performance in data communication. This is the case of VoIP commucation once the delay problem could be a relevant issue in many cases.

References

[1] A. Rodriguez, J. Gatrell, and R. Peschke, *TCP/IP Tutorial and Technical Overview*, 7th ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2001.

[2] A. S. Tanenbaum, *Computer Networks*, 4th ed. Prentice Hall, August 2002.

[3] A. Löfgren, L. Lodesten, S. Sjoholm, and H. Hansson, "An analysis of fpga-based UDP/IP stack parallelism for embedded ethernet connectivity," in *Proceedings of Norchip Conference, Oulu, Finland, 23rd*, Nov. 2005, pp. 94–97.

[4] L. Weidong, "Designing TCP/IP functions in FPGAs," Master's thesis, Code Number CE-MS-2003-09.

[5] A. Tavoularis, M. G. Manousos, D. Economou, and G. Lykakis, "Accelerating voip applications using virtex FPGAs," *FPGA and Structured ASIC Journal*, 2004.

[6] S.-M. Chung, C.-Y. Li, H.-H. Lee, J.-H. Li, Y.-C. Tsai, and C.-C. Chen, "Design and implementation of the high speed TCP/IP offload engine," in *Proceedings of the 7th International Symposium on Communications and Information Technologies*, Oct. 2007, pp. 574–579.

[7] K. Morita and K. Abe, "Implementation of UDP/IP protocol stack on FPGA and its performance evaluation," in *Proceedings of IPSJ General Conference*, 2001, pp. 157–158.

[8] K. S. Siyan and T. Parker, *TCP/IP Unleashed*, 3rd ed. Sams Publishing, August 2002.

[9] L. Dostálek and A. Kabelová, *Understanding Tcp/ip: A Clear And Comprehensive Guide*. Packt Publishing, April 2006.

[10] D. E. Comer, *Internetworking with TCP/IP*, 5th ed. Prentice Hall, July 2005, vol. 1.

[11] P.-K. Lam and S. Liew, "UDP-liter: an improved UDP protocol for real-time multimedia applications over wireless links," in *1st International Symposium on Wireless Communication Systems, 2004.*, Sept. 2004, pp. 314–318.

[12] *IEEE 802.3 LAN/MAN Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, IEEE Std. 802.3, 2008.

[13] A. Dollas, I. Ermis, I. Koidis, I. Zisis, and C. Kachris, "An open TCP/IP core for reconfigurable logic," in *13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2005. FCCM 2005.*, April 2005, pp. 297–298.