An analysis of FPGA-based UDP/IP stack parallelism for embedded Ethernet connectivity

Andreas Löfgren Lucas Lodesten Stefan Sjöholm
Department of Computer Science and Electronics, Mälardalen University, Västerås, Sweden
RealFast Hardware Consulting AB, andreas.lofgren@realfast.se
Hans Hansson
Department of Computer Science and Electronics, Mälardalen University, Västerås, Sweden,
hans.hansson@mdh.se

Abstract:

When designing FPGA-based Ethernet connected embedded systems the priority and necessity of requirements such as cost, area, flexibility etc. varies for each system. Simplified for most systems, it can be stated that no extra functionality than required is desired. Hence, when designing a UDP/IP stack in an FPGA a single UDP/IP stack "template" design is not suitable to effectively realize the different embedded network system requirements.

We present three different UDP/IP stack cores, with different grades of parallelism and suited for various network demands. We show that the UDP/IP core area can be reduced to 1/3 of the original size with an appropriate implementation, accomplished by a trade-off between parallelism/latency and area. Furthermore guidelines are proposed on how to perform the trade-off between parallelism, area (cost), flexibility and functionality when designing an UDP/IP stack for compact embedded network systems.

1 Introduction

Ethernet is a very popular and commonly used network standard when connecting to a Local Area Network (LAN) or the Internet. In the past, when connecting an embedded system to a LAN even just for simple point-to-point communication, it was necessary to use additional network circuits that had more functionality than required, which came at a high cost. Furthermore, a processor was needed to implement the network stack. Now, with the existing FPGA (Field-Programmable Gate Array) technology it is feasible to implement an application-tailored subset of a UDP/IP (User Datagram Protocol/Internet Protocol) stack to achieve a straight-forward and cost-effective connection to a network.

Although the UDP/IP stacks can be configured for various numbers of network interfaces, this paper will focus on the Ethernet network protocol and local area networks.

A number of hardware UDP/IP stacks have already been realized. FPGA-based UDP/IP stacks for various networks, such as Ethernet [2][3], remote control applications [4], FireWire [5], and ATM [6], are described.

In [1] 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.

The specification and implementation of a TCP/IP stack core in a FPGA is presented in [7]. The design handles one communicating network node and uses external memory for storage.

[8] describes how a FPGA implementation of a TCP/UDP/IP stack core can be realized with Handel-C as input language.

It is clear when studying the related work that the focus is not on how to streamline an UDP/IP stack for various network demands. Instead a single representation is presented for each work. Our work presents three on-chip streamlined stacks for different embedded network system requirements. This approach is not used in the related research areas.

2 Background

This section covers the most important terms and methods used throughout the paper.

2.1 OSI model

The Open Standards Interconnect (OSI) model is theoretical 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 are not further explained. In section 2.2 and 2.3 the used protocols are shortly explained. For a detailed description of the layers and protocols, see [9].

2.2 Network Level Protocols

The most important protocol at this level is the *Internet Protocol (IP)*. IP attempts to deliver messages to the destination, which is selected by a unique IP address.

Internet Control Message Protocol (ICMP) is a network diagnostics protocol and is used to report problems with delivery of IP datagrams within an IP network.

Address Resolution Protocol (ARP) allows requests of

the Medium Access Control (MAC) address from other nodes when only the IP address is known.

Reversed Address Resolution Protocol (RARP) is used to determine the node's own IP address, often at initialization. It uses a mechanism similar to ARP, but here the MAC address of the host is the known parameter, and the IP address the requested.

2.3 Transport Level Protocols

The most used Transport Layer protocol is the *Transmission Control Protocol (TCP)*, 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. UDP is basically an interface for the packets sent by the network protocol IP. Hence, UDP does not give any delivery guarantees and do not provide error handling and retransmission of missing packets. 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. UDP may also be used as a carrier protocol in systems where the application itself includes the functionality for a reliable communication.

3 UDP/IP stack implementation

The hardware UDP/IP stack cores that are described in this paper can be viewed as tailored subsets of the TCP/IP stack and one possible configuration is represented, related to the OSI model, as the shadowed area in Figure 1.

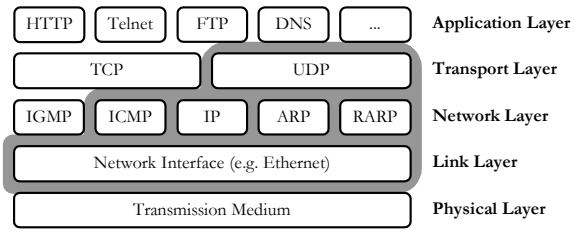

Figure 1. UDP/IP stack in OSI model layers, redrawn from [9].

All three UDP/IP cores include the Transport Layer, the Network Layer and the Link Layer, which are entirely implemented in the FPGA. UDP is used for the Transport Layer.

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

The Application Layer may be a software- or hardware application that communicates with the used UDP/IP core through the Transport Layer. By using a hardware user application in the FPGA the system performance can be boosted drastically. A software application can be implemented in a separate microprocessor chip alternatively as a soft- or hard processor in the FPGA (System-on-Chip).

The Transport-, Network- and Link Layers in the UDP/IP stack cores are designed using VHSIC Hardware Description Language (VHDL) as input. This means that the designs are not restricted to a specific FPGA technology.

The Physical Layer consists of an external Physical Layer Device (PHY) device that preferably handles an Ethernet connection at 10/100/1000 Mbps. Hence, a natural interface between the FPGA with the UDP/IP core and the PHY is to use a Media Independent Interface (MII) for 10/100 Mbps Ethernet and/or a Gigabit Media Independent Interface (GMII) for Gigabit Ethernet. These interfaces are similar to each other and are supported by the UDP/IP stack cores. If the network speed is changed the interface can be switched between MII- and GMII mode without re-configuring the FPGA.

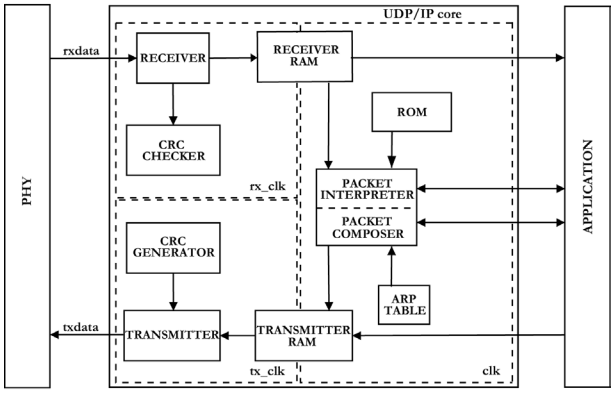

Figure 2. Block diagram of the UDP/IP cores.

All three UDP/IP stack implementations use the same basic design structure, which can be viewed in Figure 2. The designs consist of three clock domains; one system clock used for control purposes, one receive clock and one transmit clock. A short description of each common block follows:

Receiver: Manages incoming packets. The Receiver will check for a new packet (preamble from Ethernet PHY). Once a new packet is detected the packet will be saved byte-wise to the Receiver RAM (rx_ram). Each byte is also sent to the CRC checker, which 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 core's MAC-address and broadcast MAC-addresses are accepted. If the packet check fails the packet will be rejected.

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:

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

Receiver RAM: The Receiver RAM temporarily stores the entire received packet, i.e including the whole frame with Ethernet header, IP header etc. Packets with incorrect MAC-addresses will be filtered out in the Receiver and are not stored in the Receiver RAM.

ROM: To reduce the gate count the designs contain elements that are considered as static in the transmitted Ethernet frame. These elements are stored in a ROM and examples of the static IP header elements are: protocol type, version, header length, differential services, fragmentation flags and fragment offset.

Packet Composer/Interpreter: These are the main control blocks, which are closely related to each other. The blocks checks that the incoming packet is valid, manages UDP packets in both directions, manages ARP requests, generates ARP responses, manages ICMP requests and generates ICMP responses. The UDP checksum will always be calculated for transmitted packets by the Packet Composer. For this work the Packet Interpreter is configured to check the UDP checksum on incoming packets. However, this feature is optional and may be turned off to further reduce gate count. The checksum calculation is based on a 16-bit two's complement adder and is executed in parallel with the data copy to the Transmitter- or Receiver RAM from the Application Layer respectively from the Receiver.

ARP lookup table: Contains IP- and MAC-addresses for up to 4 simultaneous network nodes with a FIFO priority queue of the entries.

Transmitter RAM: The Transmitter RAM temporarily stores the entire packet, i.e including the whole frame with Ethernet header, IP-header etc., which shall be transmitted.

Transmitter: The Transmitter will check for a send flag from one of the packet types (UDP, ARP, RARP or ICMP). At the beginning, preamble is sent, where the last nibble is a start of frame delimiter. The Transmitter then reads data from the Transmitter RAM and 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.

Both the RAMs are implemented as double-ported FPGA Block RAMs, which works as separators to the clock domains. A maximum of two packets can be stored simultaneously. Communication statistics, such as UDP length errors and UDP checksum errors of received packets, can optionally be implemented and communicated to the Application Layer. A simple handshake interface is implemented as interface to the application.

The embedded network architecture is closely connected to the intended function of the system and also directly affects the UDP/IP core implementation. We will address the need for several variants of the UDP/IP core by giving three network architecture examples in sections 3.1 to 3.3.

3.1 "Minimum" UDP/IP core

In this type of network the hardware application is assumed to be either transmit-intensive or receive-intensive. One example of such an application is a data acquisition system where at least one hardware node is connected to a sensor. Raw data from the sensor is pre-processed, e.g. digital filtering, in the FPGA and then transmitted to a central unit in the network. For this small point-to-point network architecture category we propose a "minimum" UDP/IP core that utilizes logic sharing of the Packet Composer and Packet Interpreter parts of the design (i.e. less parallelism). The core is able to transmit and receive packets simultaneously, but since the control logic is shared between the transmit- and receive side in the design, thus dependent of each other, the operation may be expressed as a hybrid between half-duplex and full-duplex mode. In this type of network system the Application Layer can be implemented directly in hardware and thus omitting the need for an external (or an on-chip) CPU. Furthermore, the hardware UDP/IP stack can use a fix IP-address for the central unit and also exclude the ARP/RARP/ICMP functionality in the core. This gives a very area-effective design, at the expense of flexibility.

3.2 "Medium" UDP/IP core

For this implementation the network architecture requires full-duplex operation. Several nodes must be able to communicate, in both directions, with the hardware UDP/IP stack. The "Medium" core is based on the "Minimum" implementation, but to attain higher flexibility the Packet Interpreter and Packet Composer are configured to process simultaneously. Furthermore ARP functionality is included, allowing the core to request the MAC address from other nodes when only the IP address of its neighbours is known. The core will also respond to ARP requests from neighbouring nodes. An ARP table will simultaneously handle the MAC- and IP addresses of 4 nodes. ICMP response functionality is included as well, which responds to ping requests with a fixed ICMP length and data. This core is suitable as a trade-off between the "Minimum" and "Advanced" implementation to achieve the most effective design.

3.3 "Advanced" UDP/IP core

For this implementation the embedded network system requires a connection with a reliable communication link for frames containing control data as well as a fast data communication UDP link. The network speed is 1000 Mbps in full-duplex mode and the Ethernet frame length must be maximized. Here an "Advanced" UDP/IP core is presented that uses the "Medium" implementation and additionally also manages Gigabit Ethernet and an Ethernet frame length of 1518 bytes (Ethernet Jumbo frames are not yet supported by the core). A TCP "channel" is incorporated with UDP as a carrier protocol, enabling slow but reliable TCP/IP communication handled by an external (or on-chip) application, see Figure 3.

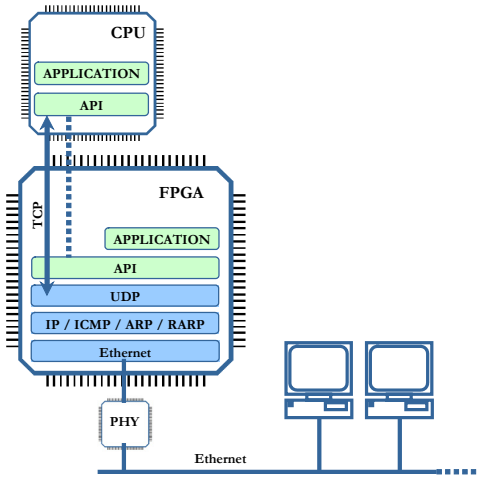

Figure 3. Advanced network architecture.

Due to the longer supported packet length this implementation requires two extra Block RAMs compared to the other two designs. The impressive performance for pure UDP data is ~957 Mbps in both directions when excluding overhead, such as Ethernet preamble, UDP/IP headers, CRC and Ethernet interframe gap, in the calculation. The overall performance is, of course, dependent of the latency at the Application Layer. Refer to Table 1 for a comparison of features and synthesis results for the three cores.

4 Evaluation and Discussion

Hardware embedded system design requires decisions regarding simplifications, parallelization and trade-off between functionality and performance/area. At least the following issues must be taken into account when designing an UDP/IP core: Allowed logic utilization, network speed, number of simultaneous communicating network nodes, packet loss limit, supported protocols, duplex mode and packet size.

Table 1. Comparison of the implementations.

Spartan-3, xc3s200-4ft256			
	"Minimum"	"Medium"	"Advanced"
Xilinx Slices	517	1022	1584
Xilinx BRAMs	3	3	5
Fmax (MHz)	90,7	60,3	105,6
Length (bytes) *	256	256	1518
Duplex mode	Full **	Full	Full
Speed (Mbps)	10/100	10/100	10/100/1000
ARP	No	4 entries	4 entries
RARP	No	No	Yes
ICMP	No	Yes	Yes
TCP "channel"	No	No	Yes
Flexibility	Low	Medium	High

* Ethernet preamble not included, ** Refer to section 3.1

As seen in Table 1 the "Minimum" core implementation only occupies approximately 1/3 of the area compared to the "Advanced" representation. Additionally for the "Minimum" core, the Application Level is easier to implement in hardware, giving a compact on-chip solution. For many point-to-point embedded network systems this area saving is definitely a vital factor when choosing the most cost-effective FPGA chip. However, the "Minimum" implementation has less performance, is not that reliable and requires more adaptation of the entire network system. All cores have been verified successfully in real hardware and the "Medium" and "Advanced" versions have both been incorporated in industrial projects.

As a guideline we recommend to start with a flexible "Advanced" UDP/IP solution and rationalize the design to an implementation that still fulfils the network system demands. Major area savings are achieved by reducing the supported packet size, excluding protocols and decrease the allowed network speed, but if the goal is a "Minimum" core static IP addresses and parallelization of sub-functions, such as checksum calculations, will further contribute to the area reduction. For efficiency the VHDL design should be parameterized to the highest possible degree to simplify these changes.

5 Conclusion

This paper shows the necessity of multiple representations of hardware UDP/IP stack cores. We present three core versions, all successfully implemented and verified in hardware, which exploit different network system characteristics. The "Minimum" core is preferred for point-to-point networks and allows a simple application that offers a cost-effective on-chip solution. The "Advanced" solution gives a reliable, high performance and flexible core suited for general-purpose networks. The "Medium" representation is used as a trade-off between the "Minimum" and "Advanced" core to achieve an effective design mainly for networks with a small number of communicating nodes. In many systems, that utilize an FPGA, the ambition is to fit the design in a cost-effective device and consequently area constraints are of high priority. As shown, the UDP/IP core area can be reduced to 1/3 of the original size by adapting the design to the network system requirements. This adaptation is accomplished by a parallelism/latency vs. area/cost trade-off. The FPGA technology makes the adaptation easy and the primary function in the chip is not unnecessary affected by excessively area consuming Ethernet connectivity logic.

We suggest that the UDP/IP stack should be customized when specifying the entire embedded network system, starting from a flexible solution and trimming down the core to fulfill the requirements of the embedded network system that is being designed.

6 References

[1] *Designing TCP/IP Functions In FPGAs* by Weidong Lu, MSc Thesis, Code number CE-MS-2003-09

[2] *Internet Connected FPGAs* by H Fallside, M J S Smith, Field-Programmable Custom Computing Machines, 2000 IEEE Symposium on , 17-19 April 2000 Pages:289 – 290

[3] *Implementation of UDP/IP Protocol on FPGA and Its Performance Evaluation* by K Morita, K Abe, IPSJ General Conf. Special5, pages 157–158

[4] *Specification of TinyIPv6 Protocol Stack for Remote Control and Its Implementation on FPGA* by Y Izuhara, K Morita, T Tateoka, K Abe, IPSJ Journal, Vol.43, No.11, pp.3540-3548, 2002

[5] *Hardware Design and Implementation of IP-over-1394 Protocol Stack and Its Evaluation* by M Yusairi, B Abu, K Abe, Technical Report of IPSJ, Vol.IAC2002, No.5, pp.51-58, Mar. 2003

[6] *Protocol Wrappers for Layered Network Packet Processing in Reconfigurable Hardware* by F Braun, B Abu, J Lockwood, M Waldvogel, Micro, IEEE ,Volume: 22 , Issue: 1 , Jan.-Feb. 2002 Pages:66 – 74

[7] *Design and Implementation of a TCP/IP Core for Reconfigurable Logic* by Christophoros Kachris, Technical University of Crete

[8] *Rapid Development of Reconfigurable Systems* by S. P. G. Chappell, IEEE 12th International Workshop on Rapid System Prototyping, Monterey California, June 2001

[9] *TCP/IP Tutorial and Technical Overview (7th Edition)* by Adolfo Rodriguez, John Gatrell, John Karas, Roland Peschke (ISBN: 0130676101).