# FPGA Implementation of IPv6 Header Processor

Zdravko Todorov, Danijela Efnusheva, Ana Cholakoska and Marija Kalendar
*Computer Science and Engineering Department, Faculty of Electrical Engineering and Information Technologies, Ss. Cyril and Methodius University, Rugjer Boshkovik 18, PO Box 574, 1000 Skopje, N. Macedonia*
*z_todorov@outlook.com, {danijela, acholak, marijaka}@feit.ukim.edu.mk*

Keywords: IPv6 Protocol, FPGA, IP Header Processing, Multi-Gigabit Networks.

Abstract: With the increasing number of Internet devices, the emergence of IoT, 5G and the increased traffic between the devices, the IPv6 is complementing IPv4. As IPv6 is becoming the protocol of choice by the new technologies, in order to accommodate for the features demanded by these technologies it is necessary to have high speed and low latency between the connected nodes. This paper introduces a hardwired IPv6 FPGA node, which processes IPv6 packets and is focused on high-speed transmission. Although, the code is written VHDL, it is written in a way which enables the user to easily add new features and implement new extension headers. The implementation of this IPv6 header processor is done on a Virtex7 VC709 FPGA development board.

## 1 INTRODUCTION

As of 2021, almost three decades after the appearance of IPv6, only 35% of all accesses to Google have been made with IPv6. Google's chart of accesses starts to grow throughout the year 2011, and the official date of exhaustion of the IPv4 address-space was on 31 January 2011. Even though the adoption rate is not equal in all countries, some countries reach over 50% levels, and some have less than 1% adoption in 2021 [1]. The IPv4 protocol has 2^32 (<4.3e9) possible addresses with total available addresses ~3.7e9, the World's population counts more than 7.5e9, leaving ½ devices for every human. In comparison, IPv6 has 2^128 (~3.4e38) possible addresses (~8e28 more than IPv4). Processing the IPv6 header is different than processing the IPv4. The main difference is the checksum check, which in IPv6 is removed, and instead, bit-level error detection for the entire IPv6 packet is performed by the link layer [2]. Additionally, each device in IPv6 will have its public routable address, which makes it very suitable for the new wireless devices and IoT devices.

The IPv6 processor logic is simple to design, provided that it will be specially adapted to work with IPv6 headers. The proposed header processor will be used to read a single IPv6 header, modify the header where necessary, and then send it to the next node in the network. One of the unique features of the IPv6 protocol compared to IPv4 are the extension headers, which now are of variable non-fixed size, can be placed in mixed order, and only the ones used need to be sent. That means that the protocol by itself requires a certain degree of customizability. Manufacturing such a processor on an ASIC proposes great challenges because of the fast-paced development in the networks and, on the other side, slow-paced IC development (½ to 2 years). Because this type of technology is not suitable for such logic, we are exploring other types of technologies [3].

The FPGA technology suits our requirements with fast-paced development and customizability, and where speed is necessary, we can easily modify the code and trade chip resources for lower latency. In other words, we get a good compromise of performance, price, and re-programmability [4]. This design is developed on a reconfigurable hardware platform – FPGA development board Virtex7 VC709 [5].

The rest of this paper is organized as follows: Section 2 gives an overview of different state of the art solutions. Section 3 describes the proposed IPv6 header processor and explains its ability to provide fast IPv6 header processing. Section 4 presents simulations and synthesis results from the FPGA implementation of the VHDL IP header processor model. Section 5 concludes the paper, outlining the benefits of the proposed IP header processor.

## 2 STATE OF THE ART

Every network device that is part of a computer network is intended to examine field in the packet headers to decide what to do with each packet. This process of identifying and extracting fields in a packet header is subject to a vast amount of research [6]. With the ever-increasing speed of network links, the research is mostly focused on hardware acceleration for achieving suitable processing speeds [7]. This is mainly achieved by combining application-specific coprocessors with general-purpose multiprocessor systems, or reconfigurable FPGA platforms. In most cases, network processors (NPs) [3] and [8], are used to perform fast data plane packet processing. This includes processing of the IP header, by analysing, parsing and modifying its content. NPs might include some specialized hardware units to perform task offloading, such as lookup and pattern matching, classification of packets, queue management and traffic control [9].

The most popular NPs used today, include one or many parallel homo- or heterogeneous processing cores. For instance, Intel's IXP2800 processor [10], includes 16 identical multi-threaded general-purpose RISC processors organized as a pool of parallel homogenous processing cores that can be easily programmed with great flexibility towards ever-changing services and protocols. Furthermore, EZChip has introduced the first NP with 100 ARM cache-coherent programmable processor cores [11], that is by far the largest 64-bit ARM processor yet announced.

The discussed NPs confirm that most of the operations in NPs are performed by general-purpose RISC-based processing cores as a cheaper but slower solution, combined with custom-tailored hardware that is more expensive but also more energy-efficient and faster. If network packet processing is analysed on general-purpose processing cores, then it can be easily concluded that a significant part of processor cycles will be spent on IP packet header parsing and processing.

On the other hand, some proposals of TCP/IP offload engines [12] provide a certain amount of processing relief compared to a classical network interface card, but still, it requires a huge portion of data processing from the main processor. The sequential software flow, i.e. protocol processing consumes CPU time and resources, creating a dependency between processor load and available throughput as well as latency. This reveals a major drawback, especially for embedded systems where resources are even more limited and CPU time is needed for application-specific tasks. To overcome these system-dependent limitations in throughput and latency, the authors of [13], implement a complete TCP/IP stack in hardware. This 10 GbE hardware-based TCP/IP stack can handle a single physical network interface and contain IPv4, ICMPv4, TCP and UDP protocols.

Furthermore, the authors of [14] introduce a novel architecture implementing a TCP/IP stack capable of processing 10 Gb/s data full-duplex, while handling thousands of concurrent sessions. The architecture's resource requirements scale linearly with the number of supported sessions to over 115,000 given today's 20 nm devices. Similar types of architectures appear in [15] and [16] - the first being an open-source Gigabit Ethernet TCP/IP IPv6 networking architecture, designed for packet processing, IoT, test & measurement, and control (e.g., sensors, motors, etc.) applications and the second implementing a UDP/IP hardware protocol stack that enables high-speed communication over a LAN or a point-to-point connection. The core, designed for standalone operation, is ideal for offloading the host processor from the demanding task of UDP/IP encapsulation and enables media streaming with speeds up to 100Gb/s even in processor-less SoC designs. Assuming that most of the available research presents a hardware implementation of IPv4 protocol, our research is focused on developing a dedicated processor module for IPv6 protocol. The emergence of novel technologies, such as 5G, together with the significant expansion of devices on the Internet and the Internet of Things, makes IPv6 protocol a necessity, compared to IPv4.

## 3 DESIGN OF IPV6 HEADER PROCESSOR

The IPv6 header processor consists of three processors: main processor, memory processor, and error processor, which is shown in Figure 1. The main processor is processing the header information while receiving data from the memory processor and is sending error data to the error processor. The memory processor is the bridge between the on-board RAM memory and the main processor. Its purpose is to get the necessary data from the external memory and prepare it for the main processor. The error processor reads error data from the main processor and sends the error messages back to the source if necessary.

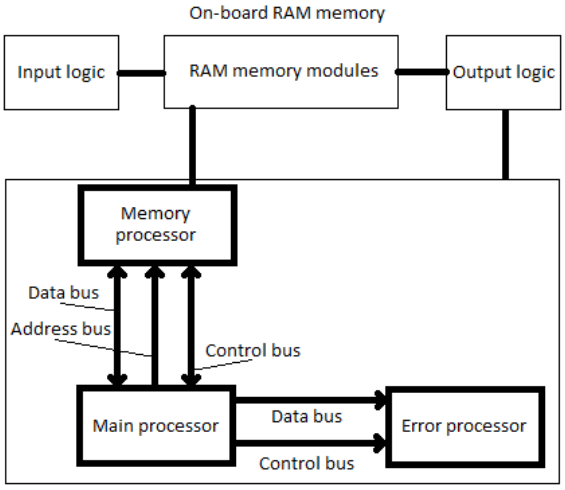


Figure 1: IPv6 header processor internal structure.

This device is designed to work with external on-board RAM. The type of RAM is defined in the memory processor. IPv6 headers have a maximum length of 64kB, meaning that storing the whole header inside the core would make the hardware too complex. The main processor consists of three data buffers, one main 16 octet data buffer and two shared 16 octet data buffers. The communication with the memory processor is realized through a 128-bit data bus and 16-bit address bus. The main data buffer is used for reading the current header continuously. Because there can be more than one main processor in the whole implementation, the shared memory can be reserved for use by each separate main processor.

The size of the main buffer can be changed. The secondary buffers are set to be 16 octets because we can exchange two addresses in a single cycle.

Transferring data from memory to processor takes three cycles with a 128bit data bus and a 16bit address bus. The formula for necessary cycles (NC) to transfer 128bit data given in (1),

$$NC = \lceil 128/N \rceil * 3 \qquad (1)$$

where 1<=N<=128, and N is the width of the data bus in bits. The address bus can be further narrowed. This changes the formula for cycles and adds further complexity to the circuit. For devices with low latency, N=128 is recommended, and for devices with lower logic space, N=16 is recommended. Transferring a whole datagram of 40B takes 9 cycles.

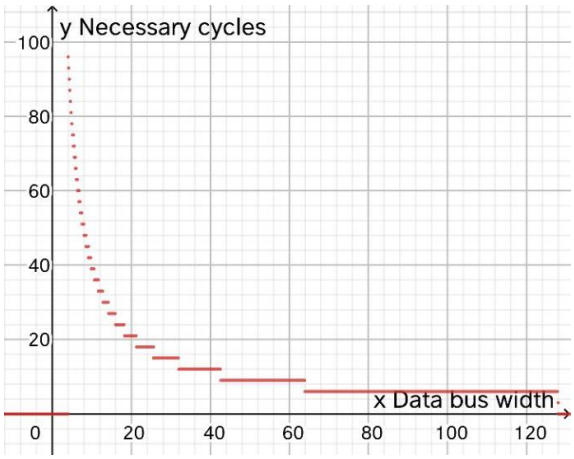


Figure 2: Necessary cycles to transfer 16 data octets to the main processor.

Processing the IPv6 header is done in two phases. The first phase is with the first eight octets of header data, which are always in the same position. Therefore, this data is processed in parallel, and if any errors are detected, the processor sends unique error detection bits to the error processor. If the header needs to be destructed, the memory processor gets this information so that the header can be deleted from the memory.

The first 8 octets of the IPv6 header contain the following information:

- Version - 4 bits are used to indicate the version of IP and is set to 6;
- Traffic class – is available for use by originating nodes and/or forwarding routers to identify and distinguish between different classes or priorities of IPv6 packets;
- Flow label – may be used by a source to label sequences of packets for which it requests special handling by the IPv6 routers, such as non-default quality of service or "real-time" service;
- Payload length – Length of the IPv6 payload, i.e., the rest of the packet following this IPv6 header, in octets;
- Next header – Identifies the type of header immediately following the IPv6 header;
- Hop limit – Decremented by 1 by each node that forwards the packet. The packet is discarded if Hop Limit is decremented to zero;

The following 32 octets contain address information, and once this information is checked then the first phase is finished.

Once the first phase is complete, the main processor starts to process the extension headers in the second phase. The extension headers are continually processed. In this implementation, we are working on a node that is placed between the source and the destination host. Therefore, we are processing only the extension headers which are subject to a change, and we are checking for errors in the fields which need not be changed. The extension headers are processed in the order in which they are present. In RFC 2460 [17], it is recommended that the extension headers are placed in a particular order, but that is not necessary. Additionally, not all extension headers are required to be present. Because of these requirements, the extension headers are processed continually.

Processing the first eight octets of the IPv6 header takes one cycle. Processing of the extension headers depends on whether the header is changed or checked.

In this implementation, we added processing of the routing header extension as an example of the possibilities that this device provides. As an example, once the routing header extension 43 is detected, the processor detects errors and processes the header. The header states that two addresses need to be exchanged and the addresses are stored in the shared memory in order to be exchanged. The shared memory is reserved only for a small portion of time in order to provide possibility of multiple main processors inside a single IC.

Once all extension headers are processed, the second phase is finished. When the second phase finishes, the processor waits for the next IPv6 header. The complete data flow diagram of IPv6 header processing is shown in Figure 3.

The error processor is a separate module which communicates with the memory processor and each main processor. Each main processor can signal the error processor for an error. The error processor then decides whether the packet should be discarded and if so, the error processor sends location information to the memory processor.

The memory processor is a bridge module that connects the main processor with the on-board RAM. When the main processor signals the memory processor for necessary data, the memory processor calculates the location of the data in the RAM. The memory processor has IP (intellectual property) core for communication with onboard DRAM provided by VIVADO Suite. In this way the data is read and written from the DRAM and sent to the main processor.

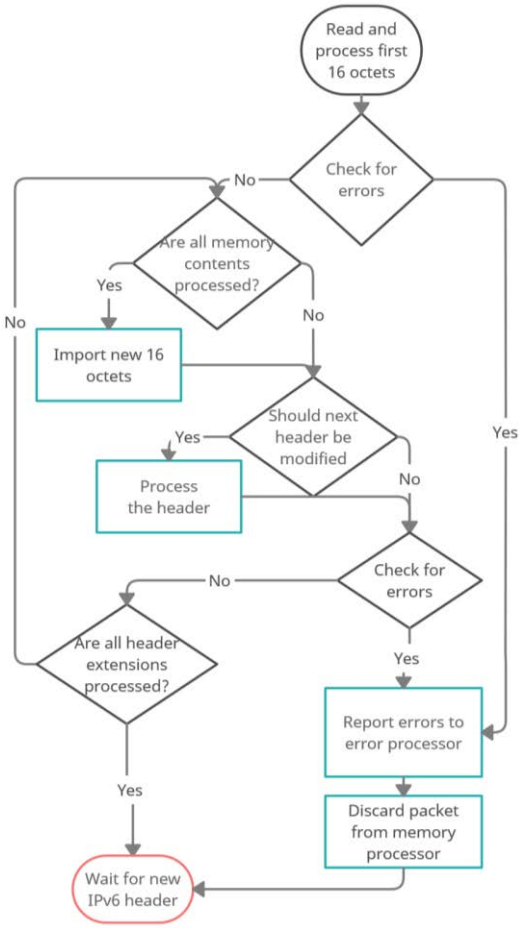


Figure 3: Main processor logic diagram.

# 4 FPGA IMPLEMENTATION OF IPV6 HEADER PROCESSOR

The proposed IPv6 processor was described in VHDL using the Xilinx VIVADO Design Suite tool. This software environment includes a simulator for performing functional analysis of VHDL models and several other hardware syntheses and FPGA implementation tools.

Simulations and functional analysis were made only for the main processor and the error processor, because the memory processor is implementation-dependant.

Once the analysis is finished, the IPv6 header processor is synthesized and implemented in Virtex7 VC709 evaluation board [5]. The synthesis results

show that the IPv6 header processor can be implemented in the Virtex7 VC709 development board, by utilizing 962 FF and 2653 LUT without the memory processor. More detailed results of the FPGA utilization, after the synthesis of the proposed IP header processor in VC709 FPGA board is shown in Table 1. Furthermore, Figure 4 presents the implemented IP header processor, after the place and route on the appropriate VC709 FPGA board is finished.

Table 1: Utilization of Virtex7 VC709 FPGA resources for the proposed IP header processor.

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 2653 | 433200 | 0.61 |
| LUTRAM | 16 | 174200 | 0.01 |
| FF | 962 | 866400 | 0.11 |

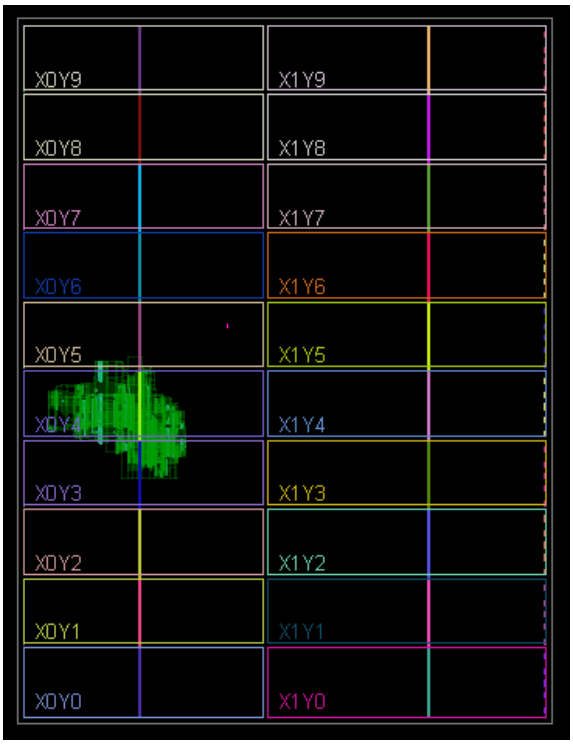


Figure 4: Implemented IP header processor on Virtex7 FPGA board (after place and route).

As a result of the low FPGA resource utilization, the processor can be further extended and then implemented on the same development board. According to this, the FPGA technology makes the proposed processor very flexible and cheap for implementation. Additionally, the ability for ease FPGA reconfiguration, makes the IP header processor implementation suitable for further modifications and improvements.

# 5 CONCLUSION

The main focus of this paper is the FPGA implementation of the proposed IPv6 header processor. Considering that the implemented IP header processor utilizes less than 0.11% FF and 0.61% LUT FPGA resources, future work would include the whole implementation of the IPv6 processor, including communication with on-board RAM and ethernet port IO. It is evident that these modifications would require more resources than previously used, but this makes the IPv6 packet processor a whole. The possibility of generating various bus widths and different logic will make this kind of processor suitable for less resourceful and powerful FPGA boards.

This device will be very practical in device-to-device communication, because with the implementation of this code in every device, all of the devices will be able to be used as a link between the source and destination node.

This approach makes use of FPGA re-configurability, which has proven to be an ideal solution for achieving reasonable speed at low price.

# REFERENCES

[1] Google, IPv6 adoption in the Internet [Online]. Available: https://www.google.com/intl/en/ipv6/statistics.html, 2021.

[2] Ch. M. Kozierok, The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference, 1st ed. CA: No Starch Press, 2005.

[3] P. C. Lekkas, Network Processors _ Architectures, Protocols and Platforms (Telecom Engineering). McGraw-Hill Professional, 2003.

[4] J. M. P. Cardoso and M. Hubner, Reconfigurable Computing: From FPGAs to Hardware/Software Codesign, NY: Springer-Verlag, 2011.

[5] Xilinx, VC709 Evaluation Board for the Virtex-7 FPGA, User guide, 2016.

[6] G. Gibb, G. Varghese, M. Horowitz, and N. McKeown, “Design principles for packet parsers,” in Proc. of ACM/IEEE Symposium on Architectures for Networking and Communications Systems, pp. 13–24, 2013.

[7] J. Kořenek, “Hardware acceleration in computer networks,” in Proc. of 16th International Symposium on Design and Diagnostics of Electronic Circuits Systems, 2013.

[8] R. Giladi, Network Processors - Architecture, Programming and Implementation, Ben-Gurion

University of the Negev and EZchip Technologies Ltd., 2008.

[9] B. Wheeler, A New Era of Network Processing. LinleyGroup Bob Wheeler's White paper, 2013.

[10] Intel, Intel® IXP2800 and IXP2850 network processors, Product Brief, 2005.

[11] B. Doud, "Accelerating the data plane with the Tilemx manycore processor," in Linley Data Center Conference, 2015.

[12] Z. Bokai, Y. Chengye, and C. Zhonghe, "TCP/IP Offload Engine (TOE) for an SOC System" in Nios II Embedded Processor Design Contest-Outstanding Designs, 2005.

[13] U. Langenbach, A. Berthe, B. Traskov, S. Weide, K. Hofmann, and P. Gregorius, "A 10 GbE TCP/IP Hardware Stack as part of a Protocol Acceleration Platform," in Proc. of 3rd IEEE International Conference on Comsumer Electronics, 2013.

[14] D. Sidler, G. Alonso, M. Blott, K. Karras, K. Vissers, and R. Carley, "Scalable 10 Gbps TCP/IP Stack Architecture forReconfigurable Hardware," in Proc. of 23rd IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, 2015.

[15] Mind Chasers, Private Island: Open Source FPGA-Based Network Processor for Privacy, Security, IoT, and Control, White paper, 2020 [Online]. Available: https://mindchasers.com/education.

[16] Xilinx, UDPIP-100G100G UDP/IP Hardware Protocol Stack, Product Brief, 2020.

[17] Internet Society, "Internet Protocol, Version 6 (IPv6)," RFC 2460, 1998.