1 Gbps Ethernet TCP/IP and UDP/IP Header Compression in FPGA

Milan Štohanzl, Zbyněk Fedra
Department of Radio Electronics
Faculty of Electrical Engineering and Communication
Brno University of Technology
Brno, Czech Republic
e-mail: stohanzl@phd.feec.vutbr.cz,
fedraz@feec.vutbr.cz

Marek Bobula
Research and Development Department
Racom Ltd.
Nové město na Moravě, Czech Republic
marek.bobula@racom.eu

***Abstract* — This paper presents a study about the hardware implementation of the TCP/IP and UDP/IP headers compression for the point-to-point communication. The implementation is focused on the achievement of minimum latency and high compression ratio. The applied compression technique is a dictionary-based method. For the TCP/IP, the fixed length of the compressed header was implemented. On the contrary, the variable length of the compressed header was implemented for the UDP/IP. The dictionaries are filled from the original data on both sides. No additional transmissions are used for retaining the continuity of the dictionary content.**

Keywords-Field Programable Gate Array (FPGA); Ethernet; header; compression; connection; IP; TCP; UDP.

I. INTRODUCTION

In order to reduce the amount of transmitted data over physical layer of the IP network, lossless data compression can be implemented. This can be useful for the one hop of the point-to-point radio or optical Ethernet link especially for solution integrated with a modem. The lossless data compression is used for reducing the data redundancy at the transmitter side (in a compressor). This redundancy is renewable at the receiver side (in a decompressor). In general, the random character of the Ethernet data traffic must be considered [2]. For this reason, any lossless data compression method may fail as the statistical-based and dictionary-based lossless data compression methods are based on the data redundancy (with specific sequence and/or time dependency). Statistical-based compression methods are unsuitable more than dictionary based methods because they need more time for preprocessing (creation of data statistics) [1]. However, the character of data in the headers of data link layer, network layer and transport layer is well known and/or predictable [6] [7] [8].

The data cannot be transferred trough IP networks without headers [9]. The second, the third and the fourth layers add the headers to the data. These headers are used for routing and transferring data through network. A single TCP connection or UDP data flow is usually hundreds or thousands of packets transmitted from source to destination. There are a lot of data that do not change during the connection in headers TCP/IP connection or UDP/IP flow. Repeated transfers of headers add the redundancy (unchanging data in headers) or the redundancy in long numbers transmitting which are changed only minimally packet by packet. Such items can be transferred in the form of differences. The main idea of the header compression is in the replacement of long raw headers by headers without redundancy. The original headers are identically restored at the receiver side [1] [2] [4].

The length of original headers is strictly defined, or is defined in some extent [6] [8]. On the contrary, the length of transmitted payload is entirely random. The headers may take up to 50 % of the volume of the transmitted data at the physical layer (when a very short payload is transmitted). When the frames over the MTU (Maximum Transmission Unit) (1500 B) are transmitted the headers take less than 5 % of volume transported on physical layer.

There are two general ways of implementing a compression algorithm. Software implementation is costs effective for low speed connections. For real-time applications with high speed connections, hardware implementation is better. Nowadays, the compression hardware exists in many forms, for example Lempel-Ziv algorithms [2], X-Match Pro Algorithms [3] and many more. The information about hardware implementation specialized in the Ethernet headers compression was not found. The software implementation of the TCP/IP headers compression was designed and published in RFC 1144 [4] and RFC 2507 [5]. Therefore, this study deals with hardware implementation of the Ethernet headers compression (TCP/IP and UDP/IP).

This paper is divided into eight sections. The original headers and compressed headers formats are introduced in Section II. Section III describes the applied method. The hardware arrangement is presented in Section IV. The compressor is discussed in Section V and the differences between compressor and decompressor are discussed in Section VI. The results of this research are summarized in Section VII. The conclusion of the present study can be found in Section VIII.

II. ORIGINAL AND COMPRESSED HEADERS

In this section, the original and compressed headers content and its meaning is described.

The IP header format is shown in Figure 7. IP version is a four-bit item. The IP length is a four-bit item and it represents one quarter of the total IP length. The basic length

 ISBN: 978-1-61208-231-8

of an IP header is 20 bytes. All the IP headers with different lengths are not compressed. The item ToS (Type of Services) contains the Differentiated Services Codepoint and the Explicit Congestion Notification. One of the compression conditions is the ToS equal to zero. The total IP length is the length of the IP packet. It is a sum of the IP header length, the higher layer protocol header length and the data (payload) length. This item is transmitted in the compressed header without any changes. IP Identification (IP ID) is the number of IP packets sent by the station. The IP ID in the TCP/IP stream is changed only minimally being sent packet by packet and therefore it is transmitted in the compressed header as an eight-bit difference. The field flags and the fragment offset are set only when the IP packet is fragmented. This field must be zero (compression condition). Only the DF (Don't Fragment) bit can be written to one. In this case, the seventh bit in the compressed header is set to one. The TTL (Time To Live) is a number that limits the lifespan of the IP packets in the IP networks. This value is constant for one TCP/IP stream at one line (when the IP packets are routed by the same way trough the IP network). The item protocol identifies higher layer protocol. For example TCP is presented by number 6, UDP is presented by number 17. The IP checksum is the checksum of all bytes in the IP header. This value need not be transmitted in the compressed header and can be evaluated during the decompression process. As a result, however, the error protection would be decreased. This value is transmitted in the compressed header without any changes. The source and destination IP addresses are the IP addresses of the end nodes and they identify the TCP/IP data stream and/or the UDP/IP data flow. [6]

The basic length of the TCP header is 20 bytes. The source and destination ports together with the IP addresses identify the TCP/IP data stream. The Sequence Number (SEQ) and the Acknowledgement Number (ACK) are changing (increasing) in TCP/IP stream packet by packet. Therefore they are transmitted in compressed header as a 16-bit difference. The data offset represents the length of the TCP header. The implemented compression algorithm allows the compression of the TCP headers with basic length. The TCP flags field contains the TCP flags. Only the headers with a set Acknowledgement flag can be compressed. If the Push flag (PSH) is set, compression is allowed and the eighth bit in the compressed header is set to one. Other flags (Urgent, Reset, Syn, Fin) must not be set. The Window size value is transmitted in the compressed header without any changes. The TCP checksum is the checksum of all bytes in the TCP header and the data payload. The evaluation of this checksum during the decompression process would cause the decrease of the error protection in a similar way as with the IP checksum. Moreover, the whole TCP packet including the payload would have to be buffered. This would disproportionately increase the latency. Therefore, the TCP checksum is transmitted in the compressed header without any changes.The urgent pointer is only set if the urgent flag is set to one. Therefore the urgent pointer is not transmitted in the compressed header. The original TCP header format is shown in Figure 8 [8].

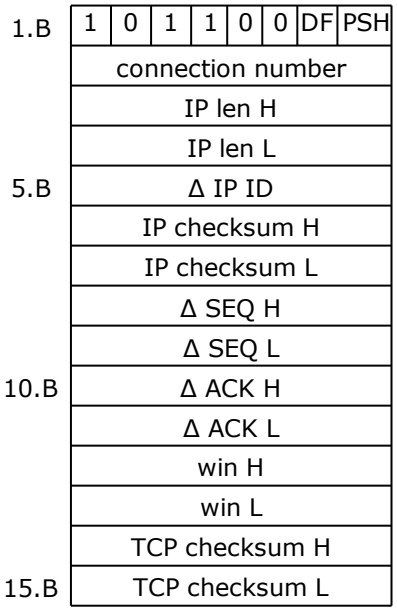

Figure 1. Compressed header format (TCP/IP).

The UDP header consists of eight bytes. Four bytes are port numbers. The item UDP length is the length of payload and the UDP header. This item can be calculated by subtracting the IP header length of the (20 B) from the IP length. The UDP checksum is the checksum of all bytes in the payload and the UDP header. This item is transmitted in the compressed header. The original headers format is shown in Figure 8. In Figure 2, the formats of the compressed UDP/IP headers are shown. As can be seen, the length of the header is variable. The item IP ID is not included in the compressed header when the IP ID is zero in the original data flow. The Δ IP ID is transmitted in when the difference between the previous and the current value is less or equal to the eight-byte-value. Otherwise, the IP ID is transmitted without changes. The format of the compressed header is indicated by the seventh and the eighth bit in the compressed header. The compressed UDP/IP header formats are shown in Figure 2 [7].

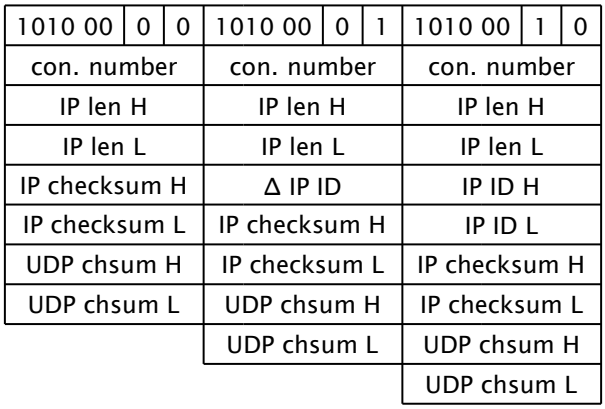

Figure 2. Compressed header formats (UDP/IP).

III. METHOD

The implemented header compression technique is a dictionary method. The compressor and decompressor dictionaries contain data from raw (original) headers that are transmitted in the compressed headers as the difference or are not transmitted in the compressed headers at all. The dictionaries are divided into cells. Every cell contains data for one TCP/IP or UDP/IP flow. The width of the words in a cell is defined for each word separately. Each cell also contains the "connection number" item and the "Counter of Use" (CoU). The CoU is used to detect the flow inactivity (see below).

At the beginning (after start-up), the compressor and decompressor dictionaries are empty. With the arrival of the first packet that meets the conditions of the compression, the raw data from headers are stored in one dictionary cell. The packet is sent without any changes. At the receiver side, the decompressor identifies this compressible packet and stores the raw data in one cell as does the compressor. After the arrival of the next packet of this flow, the compressor finds a match in the dictionary and compresses the headers. During the compression process, the compressor actualizes data in the cell from the raw headers (only changing data like IP ID, ACK, SEQ that are transmitted like differences) and increments the CoU. The decompressor detects the compressed header and decompresses this header using the data from dictionary in the same way the compressor actualizes data in the dictionary and increments the CoU at the decompressor side. Described procedure ensures that the compressor and the decompressor have the same content of dictionary at all times. This is the main condition of this method.

When the compressor finds a match in the dictionary but the raw headers are not compressible (a difference is greater than one byte and/or the flags in raw headers do not allow compression), the raw data are transmitted and the compressor/decompressor actualizes the data in the dictionary. There is a chance that the next packet of this flow will be compressible.

When all the cells of the dictionary are full and the compressor identifies compressible packet without a match in the dictionary, a revision of the dictionary is made. All cells with the CoU equal to zero (only one packet of the flow which allowed the compression passed) are released. For easier implementation, the packet is transmitted without any changes and is not stored in the dictionary. The decompressor does the same. It is also possible for the CoU in all cells not to be equal zero. In this case, the "Counter of Raw Data" (CoRD) is incremented. If the CoRD is equal to the pre-set threshold every CoU is reset. Subsequently, the packets from the stored flows may come. These packets are compressed and the CoUs in the cells of the dictionary are incremented. The first compressible packet which is not in the dictionary causes the release of cells with the CoU equal to zero. The method of zeroing CoU in case of full dictionary and transmitting unstored flows guarantees the old data in the dictionary are cleared and the dictionary is ready for the current flows. The data in the dictionary will never grow so old so as to render the storing of new data impossible.

IV. HW SOURCES

The data rate 1 Gbps (Ethernet IEEE 802.3z) corresponds to modulation rate 1.25 GBaud on the physical layer. After the deserializer, the data rate on GMII busses between MAC/PHY (Media Access Control, Physical Layer) circuits and the FPGA in Figure 3. is 125 Mbit/s. The main clock in the FPGA is the same (125 MHz corresponds to 8 ns). The hardware arrangement is presented in Figure 3. The GMII bus contains eight wires for the data, one wire for clock signal and one wire for validation signal.

The original signal from the Ethernet line is modified in the MAC/PHY circuit and brought into the FPGA. The modification (header compression) can be performed there. Then the data flow is connected to the second PHY/MAC circuit. This circuit must not monitor the CRC (Cyclic Redundancy Check) of the Ethernet frame and the headers content. The decompression must be implemented from the second side. The TX/RX (Transmit, Receive) line is ideal (without losses and interferences).

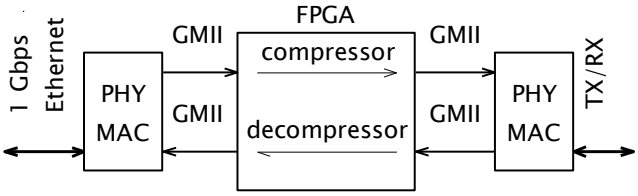

Figure 3. HW arrangement.

V. THE COMPRESSOR

The compressor processes (as well as the decompressor processes) are divided into three basic blocks. These blocks are shown schematically in Figure 4. The first block recognizes the header items of the TCP/IP and UDP/IP, in the second block the data flow is delayed in order to allow the third block to evaluate the header items before it starts to process or compress them in the original data flow.

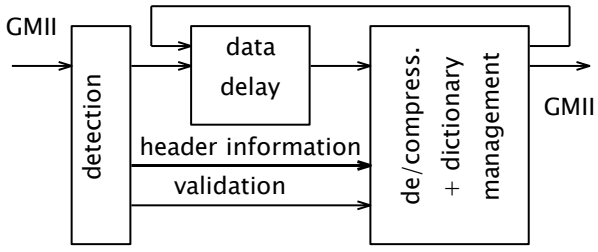

Figure 4. Block diagram of the de/compressor.

A. Detection

The main task of the header detector is to identify the header items from the TCP/IP and UDP/IP headers. This header information is copied on parallel busses. After the

stabilization of the data in the busses, the single-bit validation signal is set. The validation of the parallel data is necessary because the data are set to the busses byte by byte but the bus(ess) width corresponds to the word width of the current information item. The example of parallel bus setting is shown in Figure 9. The detection process is realized by the combinational state machine which secures the content and the order of the data in the Ethernet frame, shown in Figure 6. The byte order is determined by the internal counter of bytes. This counter is activated or reset by the validation signal of the GMII bus.

The example of the VHDL (Very High Speed Integrated Hardware Description Language) code from detector follows. There are four states from combinational state machine shown when the source IP address is detected. One-bit combinational variable the "Comb_ip_source" is set to one in these states. In other states, it is set to zero.

```
When st_sniff_IP_12 =>
        Comb_ip_source <= '1';
        Next_State_sniff <= st_sniff_IP_13;
When st_sniff_IP_13 =>
        Comb_ip_source <= '1';
        Next_State_sniff <= st_sniff_IP_14;
When st_sniff_IP_14 =>
        Comb_ip_source <= '1';
        Next_State_sniff <= st_sniff_IP_15;
When st_sniff_IP_15 =>
        Comb_ip_source <= '1';
        Next_State_sniff <= st_sniff_IP_16;
```

The variable "Comb_ip_source" is tested in the sequential part. The vectors "Loc_ip_src_N" and "data_inner" are eight-bit vectors. The "data_inner" contains the current data byte of the data flow.

```
If Comb_ip_source = '1' then
        Loc_ip_src _0 <= data_inner;
        Loc_ip_src _1 <= Loc_ip_src _0;
        Loc_ip_src _2 <= Loc_ip_src _1;
        Loc_ip_src _3 <= Loc_ip_src _2;
End if;

IP_SOURCE_OUT <= (Loc_ip_src_3 & Loc_ip_src_2
& Loc_ip_src_1 & Loc_ip_src_0);
```

The last line of the example is the combinational output. "IP_SOURCE_OUT" is 32-bits output vector. An example of the filling of this vector is presented in Figure 9. This example was taken by the JTAG (Joint Test Action Group) from the FPGA. Each vector is in hexadecimal format. The first line of this example shows the vector-data IP_SOURCE_OUT, the second line shows the IP_DEST_OUT (destination IP address). The third line shows the data_inner and the last line shows the states of the state machine. As can be seen, the state "st_sniff_IP_12" corresponds to 23_h. The byte 93_h (147_{10}) is the highest byte of the source IP address. This byte affects the value of the output vector in the next clock cycle. Every byte of the source IP address is gradually processed in the same way. The vector "IP_SOURCE_OUT" is valid in the state 27_h. Then, the following data are processed in a similar way (destination IP address,...). A validation signal is set at the moment when all data at all parallel buses are valid.

B. Data delay

There are several ways of creating a delay in the data stream in FPGA. A long shift register is easy to implement in VHDL, but it is very demanding for hardware resource consumption. A FIFO (First In First Out) memory allows reading the data flow with the delay but this delay can not be changed during the reading process. On the contrary, a RAM (Random Access Memory) memory allows the reading from any address. For this reason, the delay is implemented by the RAM memory and it is driven by the special driver (written in VHDL). The RAM memory is implemented as soft IP core and uses memory blocks of the FPGA. It is the Simple Dual-port RAM [10] with 128 word depth (every word is one byte). Every incoming byte is written in the RAM and the writing address (controlled by the RAM controller) is incremented. The data reading from RAM starts when writing address exceeds some threshold. The change of the delay is initialized by the third block (compressor). It is shown by the feedback from the third block to the delay block in Figure 4. The connection between RAM and RAM driver is shown in Figure 5. (WR means writing and RD means reading).

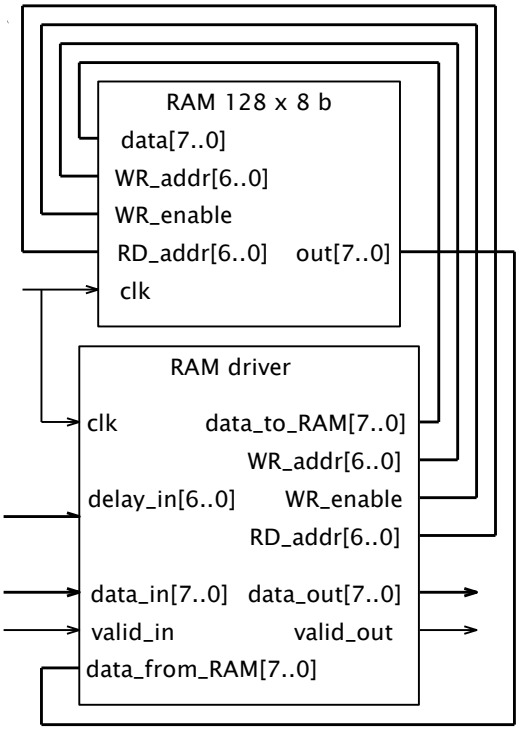

Figure 5. Delay block interconnection.

There is a delay between the writing of reading address and getting the data from the RAM output. This delay is

caused by the gating signals in RAM memory at inputs and outputs (two clock periods) and the actual reading process (one clock period). For this reason, the reading data from RAM memory are connected to the RAM driver and they are supplemented by the correct validation signal.

C. Compressor and dictionary management

The compressor block also performs the dictionary management, as was described in Section III. The compressor evaluates the original header data from the parallel busses and decides whether they meet the compression conditions, finds a match in the dictionary, compress header and/or actualizes the dictionary. The dictionary is also actualized when the match in the dictionary is found while the original header did not meet the compression conditions.

A function of the compressor is controlled by the combinational state machine, the calculations are included in the sequential part. While the compression is running, the original headers (IP, TCP and UDP) are replaced by the compressed header. The data (payload) immediately follow the compressed header. The length of the compressed header is shorter than the original headers. It means that the delay of the data flow must be changed during the compression process. In a design with the variable length of the compressed header the change of the delay must be variable. For this reason, the vector "delay_in" has the same length as the reading address "RD_addr". The value of delay change is added to the reading address value. The example is in Figure 10. The reading address is in the first line, the second line shows the validation signal, the third line shows the reading data from RAM. The last line shows the "delay_in" vector. This vector is set to zero and contains the value 19_h (25_{10}) only in one clock tact. In the next clock tact, the reading address is changed from 24_h to $3E_h$. The value of the reading address is increased by one (in every clock pulse) plus extra 19h. As described above, the output data from the RAM are delayed by three tacts following the reading address. The bytes 47_h, 45_h and 54_h in the data flow are ASCII symbols 'G', 'E' and 'T'. It is the start sequence of payload in packet (access to the web). The byte 00_h in the data flow before this sequence is caused by the change of the reading address. Nevertheless, this error is of no importance as this byte as well as the preceding bytes are replaced by the compressed header.

The example of the header compression is shown in Figure 11. In the first line internal byte counter is shown, the "delay_in" is shown in the second line. The original data flow at the input of the compressor is shown in the third line. The sequence starts by the start of the IP header. The IP length is 0024_h, IP ID is $07B7_h$ and TTL is 80_h. The payload immediately follows the TTL (the change of the delay is reflected). The payload sequence means "Hello." in ASCII. In the last line of the example, the output data flow with the UDP/IP compressed header is shown. The first byte is $A1_h$, it presents format with Δ IP ID (01_h). Connection number is 63_h.

VI. THE DECOMPRESSOR

The decompressor is divided into three blocks like the compressor. These blocks are similar to the blocks at the compressor side. The differences between the two are described bellow.

A. Detection

The detector at the decompressor side allows the detection of the compressed headers. The parallel busses between the detector and the decompressor include wires for items from the compressed headers.

B. Data delay

The delay of the data flow is realized in a similar way as at the compressor side. The difference is only in the sign for "delay_in" vector. During the decompression process, the delay must increase because the length of the recovered headers is larger than the length of the compressed header.

C. Decompressor and dictionary management

The dictionary management at the decompressor side is the same as that at the compressor side. The decompressor (like the compressor) searches matches in the dictionary and performs the basic dictionary tasks (storing data, actualizing data, releasing cells,...).

When the decompressor processes the data from the compressed header, it seeks consensus by the connection number in the dictionary. Then, the decompressor calculates original header items from the dictionary data and the received differences, actualizes the dictionary and stores the recovered header items in the internal vectors. These data are put in the output data flow in the correct order (controlled by the internal byte counter in sequential part. The restored header must be closely followed by the payload and therefore the delay is changed during this process.

VII. RESULTS

The described header-compressor and decompressor were implemented in FPGA Altera Cyclone III, namely the EP3C40F484C7 with speed grade -8, 39,600 Logic Elements (LE), more than 1 Mbits Random Access Memory, 126 M9K (special memory) blocks, 126 multipliers (18 x 18), four PLLs (Phase Locked Loop [11].

The TABLE I. contains the FPGA synthesis result. The compressor and decompressor dictionaries have five cells for the TCP/IP streams and five cells for the UDP/IP flows. The decompressor detector is more demanding on hardware resources than the compressor detector for its ability of detection also of the compressed headers. The decompressor calculates original header items and stores them in the internal vectors in contrast with the compressor which put the compressed header data directly into the output data flow. Therefore, the decompressor implementation is more hardware demanding. The difference between the data delay blocks at the compressor and decompressor sides is very simple (see in Section VI). Nevertheless, the consumption of the LEs and LUTs is higher at the decompressor side.

 ISBN: 978-1-61208-231-8

TABLE I. REQUIRED HW SOURCES

		LEs	M9Ks	LUTs
Compressor	***Detector***	404	0	109
	Data delay	74	1	24
	Compressor	3661	0	1635
Decompressor	***Detector***	526	0	156
	Data delay	82	1	32
	Decompressor	4240	0	1760

The above described implementation allows the compression of TCP/IP and UDP/IP headers with the compression from 40 to 15 bytes for TCP/IP and from 28 to 8, 9 or 10 bytes for UDP/IP. This compression and decompression processes were designed with respect to minimum delay in the data flow, in contrast to the systems that perform the compression of the headers after the buffering the whole packet. The delay is variable for Ethernet frames with compression while the delay is constant for the frames without compression. In the worst case, the flow delay is less than 50 clock periods at the compressor side and less than 30 clock periods at the decompressor side. It is less than 640 ns.

VIII. CONCLUSION

The paper presented the implementation of the header compressor for one Gbps Ethernet in the FPGA. The compression and decompression of the TCP/IP and UDP/IP headers in presented form was implemented and tested.

The variable length format of the compressed header for the TCP/IP and IPv6 header compression will also be incorporated. The future work would also include mechanisms for handshaking between the compressor and the decompressor dictionaries and for the data protection. These mechanisms would allow the use of the TX/RX real line (with losses and interferences). However, these mechanisms should be implemented on the lowest layer. It would require implementation of the MAC/PHY into FPGA.

ACKNOWLEDGMENT

This paper was supported by the project Systems of Wireless Internet Communication (SYWIC) LD11081 in frame of COST IC 0906 action. The research published in this submission was financially supported by the project CZ.1.07/2.2.00/20.0007 WICOMT of the operational program Education for competitiveness and by grant no. FEKT-S-11-12 (MOBYS).

The described research was performed in laboratories supported by the SIX project; the registration number CZ.1.05/2.1.00/03.0072, the operational program Research and Development for Innovation.

REFERENCES

[1] D. Salomon and G. Motta, Handbook of Data Compression, Springer-Verlag, London, 2010.

[2] R. Mehboob, S. A. Khan, and Z. Ahmed, "High speed lossles data compression architecture," Multitopic Conference, 2006. INMIC '06. IEEE, pp. 84-88, 23-24 Dec. 2006, doi: 10.1109/INMIC.2006.358141.

[3] J.L. Nunez-Yanez and V.A. Chouliaras, "Gigabyte per second streaming lossless data compression hardware based on a configurable variable-geometry CAM dictionary," Computers and Digital Techniques, IEE Proceedings, vol. 153, no. 1, pp. 47-58, 10 Jan. 2006, doi: 10.1049/ip-cdt:20045130.

[4] V. Jacobson, B. Nordgren and S. Pink "Compressing TCP/IP headers for low-speed serial links," RFC 1144, Feb. 1990.

[5] M. Degermark, "IP header compression," RFC 2507, Feb. 1999.

[6] "Internet protocol DARPA internet program protocol specification," RFC 791, Sep. 1981.

[7] J. Postel, "User Datagram protocol," RFC 768, Aug. 1980.

[8] "Transmission Control Protocol," RFC 793, Sep. 1981.

[9] "IEEE 802.3. Part 3: Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications," New York: IEEE Computer Society, pp. 49 2008.

[10] www.altera.com, "Internal memory (RAM a ROM) User Guide," Jan. 2012

[11] www.altera.com, "Cyclone III Device Family Overview," vol. 1. 2012.

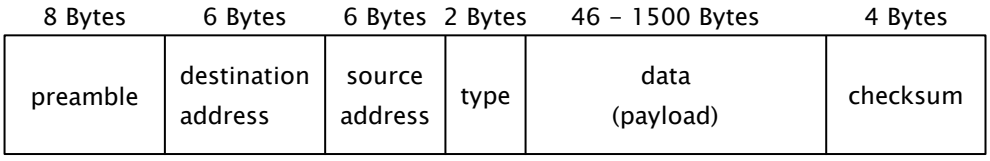

Figure 6. Ethernet frame (Ethernet II). [9]

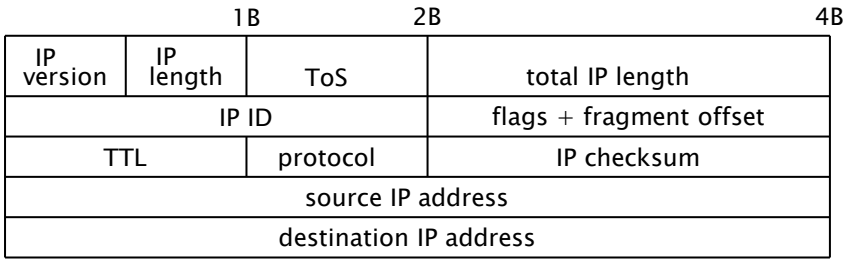

Figure 7. IP header (IPv4). [6]

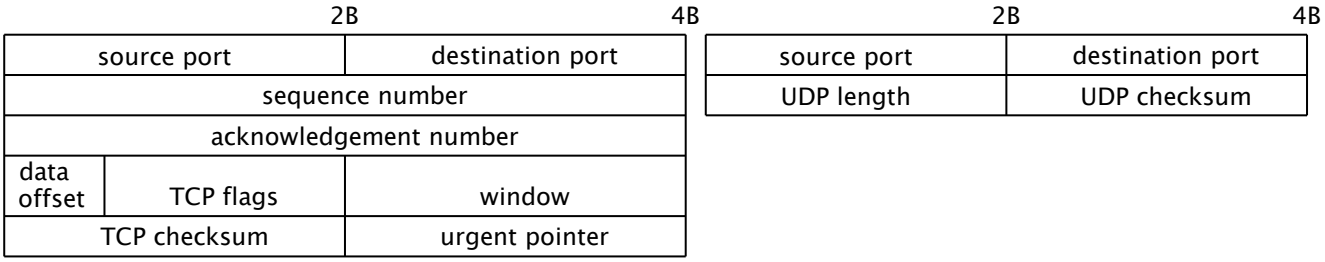

Figure 8. TCP header format (left), UDP header format (right). [7] [8]

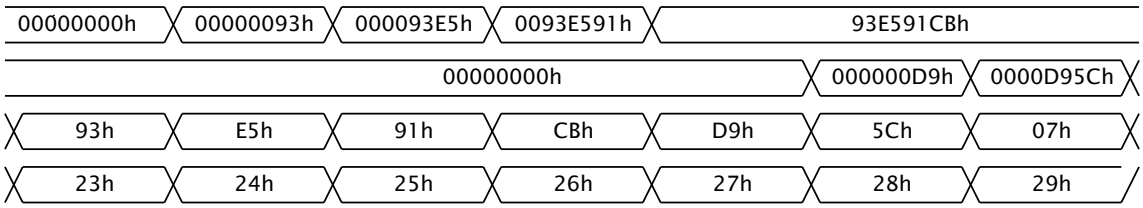

Figure 9. The example of filling of the “IP_SOURCE_OUT” data vector.

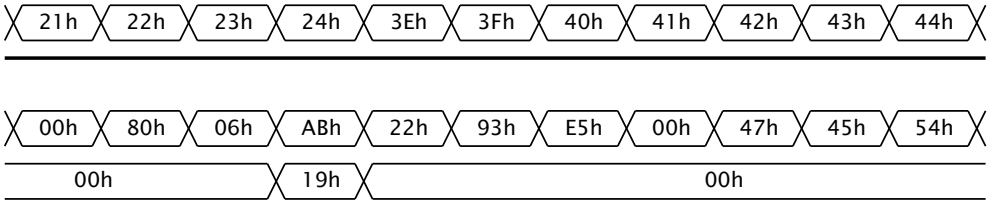

Figure 10. The example of changing of the reading address.

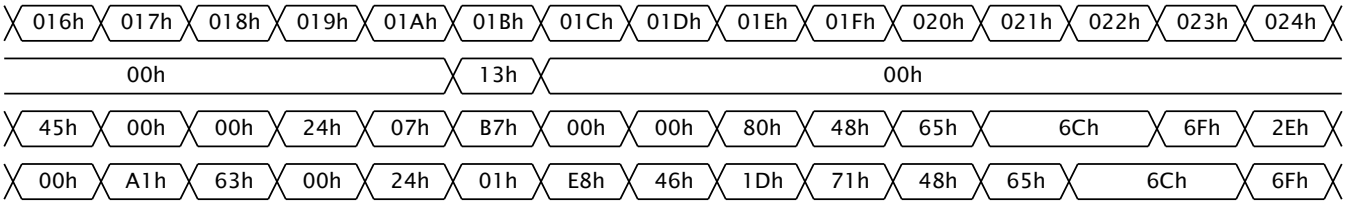

Figure 11. UDP header compression.