Feasibility Study Of Hardware Acceleration In Gif Compression Algorithm

Ivan V. Mozghovyi
Dept. of Computer Engineering
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
Kyiv, Ukraine
mozg.v34@gmail.com

Anatoliy M. Serghienko
Dept. of Computer Engineering
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
Kyiv, Ukraine
a.ser@i.ua

Roman D. Yershov
Dept. of Electronics, Automatics, Robotics and Mechatronics
Chernihiv Polytechnic National University
Chernihiv, Ukraine
roman.d.yershov@gmail.com

***Abstract*— Increasing requirements for data transfer and storage is one of the crucial questions now. There are several ways of high-speed data transmission, but they meet limited requirements applied to their narrowly focused specific target. In addition, such solutions do not solve the problem of data storage. The data compression approach gives the solution to the problems of high-speed transfer and low-volume data storage. This paper is devoted to the compression of GIF images, using a modified LZW algorithm with a tree-based dictionary. It has led to a decrease in lookup time and an increase in the speed of data compression, and in turn, allow developing the method of constructing a hardware compression accelerator during the future research.**

Keywords— FPGA, GIF, lossless compression, image compression, dictionary, hardware acceleration

I. INTRODUCTION

Nowadays, the problem of data transferring optimization is becoming one of the most significant. Whereas the size of data increases, there should be a way to transfer it with the highest speed. A solution to this problem depends on the branch of its application. There is a list of the solutions shown in Fig. 1.

One of them is parallel buses. They are used mostly for:

- Peripheral connections to the computer motherboard (e.g. PCI express bus for connection of GPU module);
- System on chip interconnection buses (e.g. Avalon interface for connection of Intel FPGA modules);
- Standardized system buses for microcontrollers (e.g. AHP APB busses of ARM ® Cortex ® processors).

Another approach is high-speed serial interfaces. An application of it can be found in:

- Network interfaces;
- The connections between modules on a single board;
- Data transmission for high-speed ADC modules;
- Low-voltage differential signaling [14, 15].

Despite the different areas of application, these solutions have **common problems**. The first one is that they are used **only for data transmission.** As a result, they cannot solve the problem of data storage, which is also important. The next one is a **narrowly focused area of application**. It means that each solution has a specific target, which is non-scalable to another.

It is well known, that the usage of data compression can be found in more branches than high-speed interfaces or parallel buses. In addition, a combination of high-speed interfaces and data compression is a good practice. For example, in the latest versions of the HDMI interface, the highest data rates are possible only using Display Stream Compression [12, 13]. The data compression is used not only for data transfers but also for storage. Therefore, developing an efficient approach of data compression solves not only the transmission problem but also the storage. It does not matter if it is big data storage or just a memory block to keep the buffered image. Decreasing the size of a single file optimizes both cases.

The main point is that usage of the different formats can modify the entire image corresponding to the specific algorithm. In most cases, it is a compression method algorithm. The main difference between all compression methods is that if it is lossless or with losses. The term or "information loss" means that some compression methods cannot guarantee exactly the same image after its decompression. It will definitely differ from the original. However, in some cases, a human eye cannot notice the difference between the source and decompressed image, and using the method with losses is acceptable. Mostly they are used for multimedia, where little distortion after decompression is insufficient. In addition, there are other features of compression methods, e.g. dictionary or run-length compression, compression ratio, compression speed [7], etc.

II. SELECTION OF A COMPRESSION METHOD

There are some points about the selection of the compression method. In general, all the compression methods are divided into:

- Run-Length Encoding;
- Statistical Methods;
- Dictionary methods.

All of these classifications find their implementations in software, but the question of their hardware implementation remains actual. In the comparative table of compression

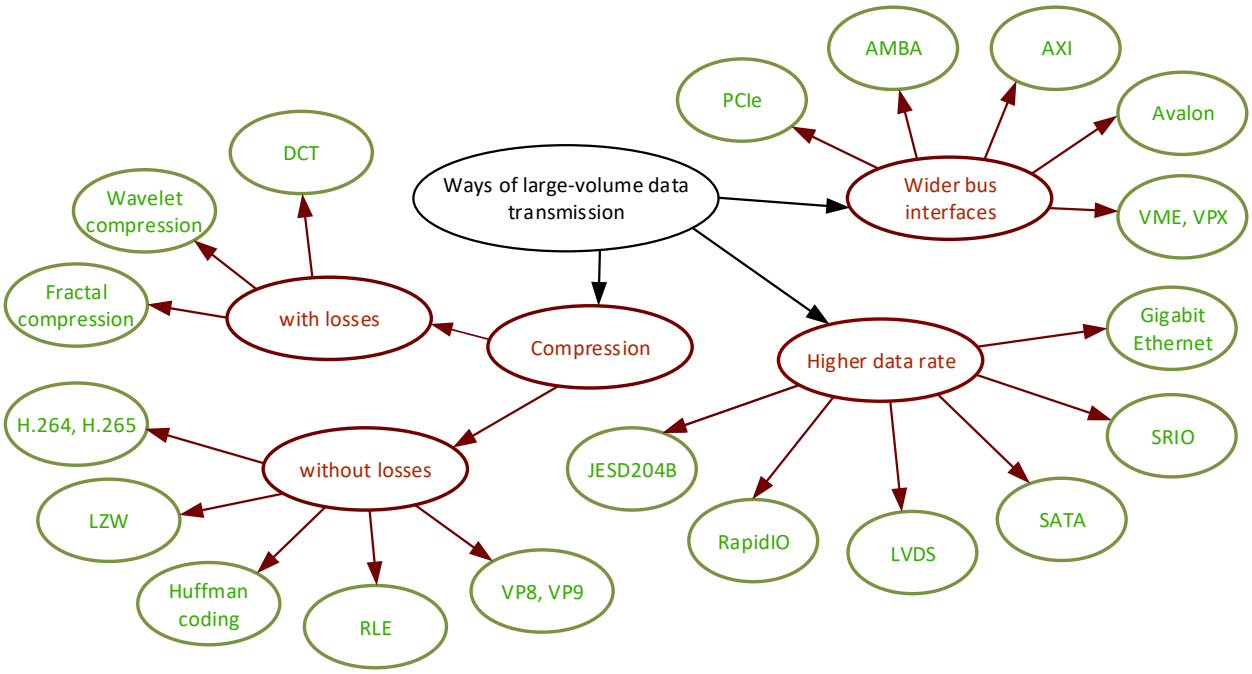

Fig. 1. Data transmission solutions

in paper [1] was mentioned that run-length encoding does not give a good compression ratio, that is why they are omitted, despite it is lossless. There are left only two variants to choose from: statistical or a dictionary method. The general scheme of a compression method consists of two parts: model and coder. A model finds the redundancy in the input data and sends it to the coder, which replaces the repetitive fragments with the corresponding codes.

A. Statistical methods

There is a clean separation onto model and coder parts. The statistical model assigns the values to the events (some data fragment found) depending on the probability of their appearance in the input data sequence. The more frequency of the event occurrence the more the value. The main problem of a hardware implementation of a statistical method is that they are mostly based on Markov stochastic modeling. In paper [4] was mentioned that the compression ratio of the statistical methods is limited by the usage of multi-symbol alphabets zeroth-order modeling, and on the other hand, the speed is limited by the usage of binary alphabets in high-order modeling. The author's explanation of it is in that the data in binary symbol alphabets only a few bits are processed in each cycle. Finally, the paper [4] represents the next features of hardware implementation:

- Hardware complexity of zeroth-order modeling and not impressive productivity results.
- High-order modeling also does not give good performance characteristics. They are not comparable with the dictionary methods' results.
- Tree-based implementations using Huffman coding showed better results but the problem of adaptation to the difference in input image sequence remains. In addition, the best performance was achieved only using content-addressed memory. In the addition, the best-mentioned compression ratio of 0.5 is also not impressive.

III. Review Of LZW Compression Method

The target of current research is to find the way to improve the existing GIF [5] image format. The main benefit of using GIF is that image compression is provided using LZW [8] dictionary lossless compression method. In some cases, it is necessary to keep the image as it was before the compression. For example, the image of schematics with small notations or values. In addition, a GIF file can be represented as an animation, due to the compressed sequence of image frames inside a file [5]. Different solutions can be found to improve the existing GIF image format. Generally, they can be divided into the optimization of the color table and improving the LZW compression method. Our research is about the modification of the LZW compression method.

Some research has addressed the problem of its hardware implementation. The authors of the paper [9] propose an FPGA-based implementation of the LZW algorithm. The results showed that such a solution gives a speed factor up to 23.51 over the sequential implementation on the CPU. Nevertheless, it is possible due to the implementation of 24 instances of compression module on a single chip. This research also approves the possibility of algorithm parallelization. There is another study [3], proposing to use the custom compression method that implements a bit plane slicing and adaptive Huffman encoding for the LZW dictionary. This approach gives a result of a higher compression ratio up by 2 times more than the original method. One more way [10] is to improve the utilization of the dictionary by dividing it into sets. This allows decreasing the lookup time and partially operating in a parallel way. Combining all of the recommended methods, an own FPGA implementation can be designed. Hardware implementation can find its application in different branches, e.g. space technologies [11].

To obtain an efficient implementation of a hardware compressor, the answers for 3 questions should be found:

- What might be pipelined and parallelized and in what way?
- What processing stages depend on the results of the previous ones?
- What parts of algorithm might be scalable?

IV. Implementation Aspects

Our method of hardware compressor implementation includes both hardware and software parts. For today, several companies (e.g. Intel, Xilinx) have suggested a solution to such implementation using the technology of the "System-on-Chip" (SoC). For example, Intel has a family of FPGAs Cyclone® V SoC, which implements an FPGA and an ARM dual-core processor ARM® Cortex®-A9 on a single chip. The communication between hardware and software subsystems is performed using the hardware processor system IP Core, which allow interconnects FPGA interfaces with ARM processing core [16]. Fig. 2 represents the scheme, where can be seen the connections between each parts of the system. Other aspects of implementation give answers to 3 questions from the review section.

Firstly, the simplified structure of the GIF file shown in Fig.3 should be analyzed. There are many things, which can be modified to get higher performance, but in our case, it should be focused on the fact that GIF format supports displaying a sequence of images as frames. Therefore, it should be considered that pipelining and parallelization might be applied either to the image regions (after dividing the image into regions) or to each image from the distributed sequence into each processing branch if the sequence of frames is processed. For example, if 4 instances of a hardware accelerator are available, an each image on each processor for the frame sequence can be distributed. And, properly to the sequence, add the compressed frames to the result file.

Another point is that the LZW algorithm mostly consists of sequential processes. The first step of the algorithm is to initialize the first 255 dictionary records with default values from 0 to 255. This step cannot be parallelized for obvious reasons. To decrease the lookup time of occasion search, the modified tree-based structure of the compression dictionary can be used. Each record of this dictionary consists of the fields shown in table 1.

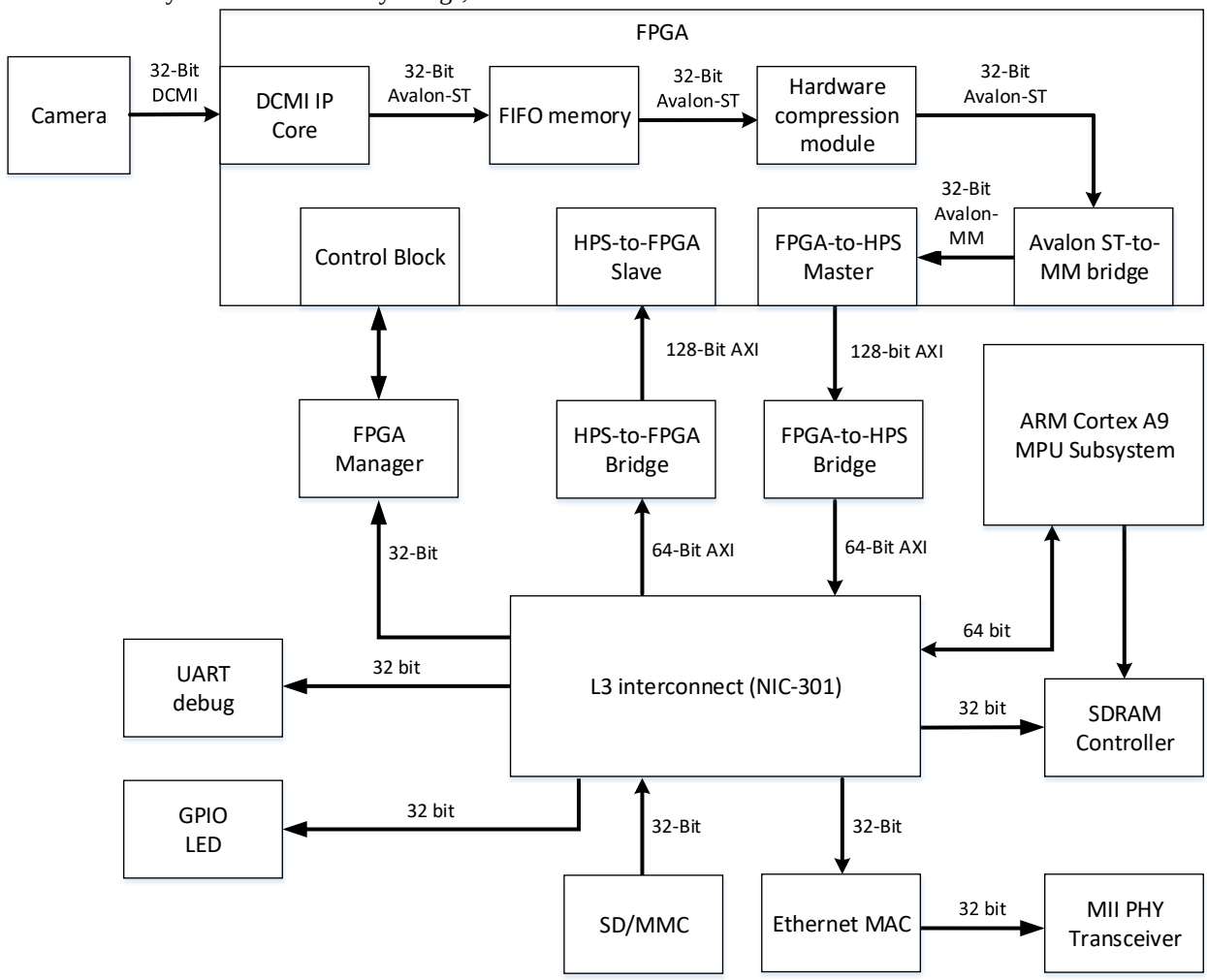

Fig. 2. Final Device Scheme

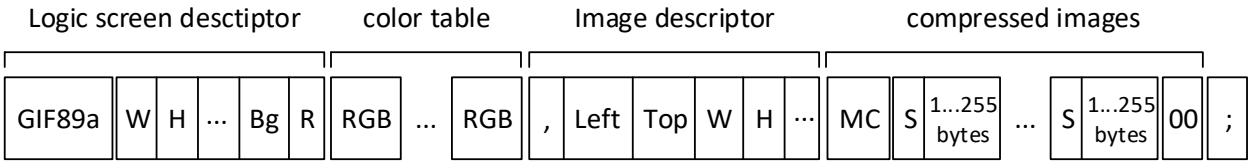

Fig. 3. GIF File Structure

TABLE I. TREE-BASED DICTIONARY NODES

	Nodes		
Address	97	268	297
Value	(-: «a»)	(97: «b»)	(268: «c»)

The address represents the actual offset in memory (RAM) to access the byte value from the dictionary default values or the ancestors. It is similar to the pointer in the C programming language. This field is necessary for getting the access to the proper nodes.

The Value is the combination of the ancestor's address and the default record. For example, if the text string is "ABC", the node has the next structure:

Address 297 is the newly assigned address of the node. 268 is the address of the ancestor's, which has the address 268 and the value 97: 268, where 97 is the address of its ancestor - default record "a".

Another point is to choose the correct form of a tree structure. The AVL [18] structure was selected because it is balanced, and the balancing process is performed on a step of adding a new node.

Scaling might be applied to different features. For example, service data of GIF file allow configuration of such parameters:

- Number of bits per color;
- Amount of frames(images) per file;
- Image resolution.

However, the main scaling parameter, in our case, is that an FPGA allows multiple instance implementations. The scaling of this parameter is limited only to the amount of the FPGA components.

V. DISCUSSION

Summarizing the above study, the developed hardware compression unit does not show breakthrough characteristics. However, during analyzing the FPGA resources that is actually used [16], it can be seen that it has enough space to implement at least 20 instances of the hardware part of investigated compression unit. Moreover, it is not the highest-performance Intel FPGA, which can offer. The leading Intel FPGAs Stratix 10, which has many more resources than any Cyclone V FPGA, allows increasing the performance characteristics by several times. Assuming the above, the limitations of the software embodiments of image compressors can be overcame.

Another benefit of an FPGA using is that a low-power consumption feature can be achieved under changing of some parameters of the synthesis constraints files (*.ucf). Therefore, one of the aspect of the future research can also be dedicated to developing the image compression unit embodiment optimized by low-power consumption criteria

REFERENCES

[1] M. Sharma, "Compression Using Huffman Coding" *International Journal of Computer Science and Network Security*, vol.10, 5, (5 2010).

[2] H. Rubaiyat, "Data Compression using Huffman based LZW Encoding Technique" *International Journal of Scientific & Engineering Research*, vol. 2, 11, (11 2011).

[3] A. Sawsan, T. Abu, M. M. J. Hossam, M. K. Asma'a, I. K. Gharaybih, "Improving LZW Image Compression", *European Journal of Scientific Research* ISSN 1450-216X vol.44, .3, (2010), pp.502-509.

[4] K. Papadopoulos and I. Papaefstathiou, "Titan-R: A Reconfigurable Hardware Implementation of a High-Speed Compressor," *2008 16th International Symposium on Field-Programmable Custom Computing Machines*, Palo Alto, CA, (2008), pp. 216-225, doi: 10.1109/FCCM.2008.14

[5] CompuServe "Graphics Interchange Format (GIF)", *CompuServe Incorporated*, (1987)

[6] J. Miano "Compressed image file formats" Addison-Wesley, New York, NY. (2001).

[7] D. Solomon, "Data Compression The complete reference", 4-th Ed., *Springer-Verlag London Limited*, (2007).

[8] T. Welch, "A Technique for High-Performance Data Compression", *Computer*, vol. 17, 3 (6 1984), 8–19. doi: 10.1109/MC.1984.1659158

[9] X. Zhou, Y. Ito, K. Nakano "An Efficient Implementation of LZW Compression in the FPGA", *International Conference on Algorithms and Architectures for Parallel Processing*, 10048, 3 (11 2016). doi: 10.1007/978-3-319-49583-5_39

[10] W. Cui, "New LZW Data Compression Algorithm and Its FPGA Implementation", *Picture Coding Symposium* (11, 2007).

[11] P.-S. Yeh, "Implementation of CCSDS Lossless Data Compression for Space and Data Archive Applications" NASA/Goddard Space Flight Center, Code 564, Greenbelt, MD, 20771 USA

[12] HDMI "HDMI 2.1 Overview". *HDMI Forum, Inc. hdmi.org*. 4 January 2017. Retrieved 2017-01-10.

[13] HDMI "HDMI 2.1 Press Release". *HDMI Forum, Inc. hdmi.org.* 4 January 2017. Retrieved 2017-01-10.

[14] P. Heydari, "Design and Analysis of Low-Voltage Current-Mode LogicBuffers", *Department of Electrical and Computer Engineering University of California*, Irvine, CA 92697-2625

[15] A. Boni, A. Pierazzi, D. Vecchi "LVDS I/O Interface for Gb/s-per-Pin Operation in 0.35-um CMOS", *IEEE Journal Of Solid-State Circuits*, vol. 36, 4, (4, 2001)

[16] Intel FPGA "Cyclone V Hard Processor SystemTechnical Reference Manual", *Intel Corporation, 101 Innovation Drive, San Jose*, CA 95134, (2018).

[17] Terasic, "DE0-Nano-SoC User Manual", *Terasic Inc*, December 28, 2015.

[18] G. M. Adel'son-Vel'skii, E. M. Landis, "An algorithm for organization of information", *Dokady Akademii Nauk SSSR*, 146:2 (1962), 263–266