UDC 004.383
DOI: https://doi.org/10.20535/2708-4930.2.2021.244189

I. MOZGHOVYI
A. SERGIYENKO
R. YERSHOV

GIF IMAGE HARDWARE COMPRESSORS

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. 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, allows developing the method of constructing a hardware compression accelerator during the future research.

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

1. 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 the parallel busses. 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 busses (e.g. Avalon interface for connection of Intel FPGA modules);
- Standardized system busses for microcontrollers (e.g. AHP APB busses of ARM ® Cortex ® processors).

Another approach is the 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 a set of 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 one.

It is well known, that the use of the data compression can be found in more branches than for the high-speed data transfer. 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 the Display Stream Compression mode [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 not. The term of the “information loss” means that some compression methods cannot guarantee exactly the same image after its decompression. It will definitely differ from the original image. 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.

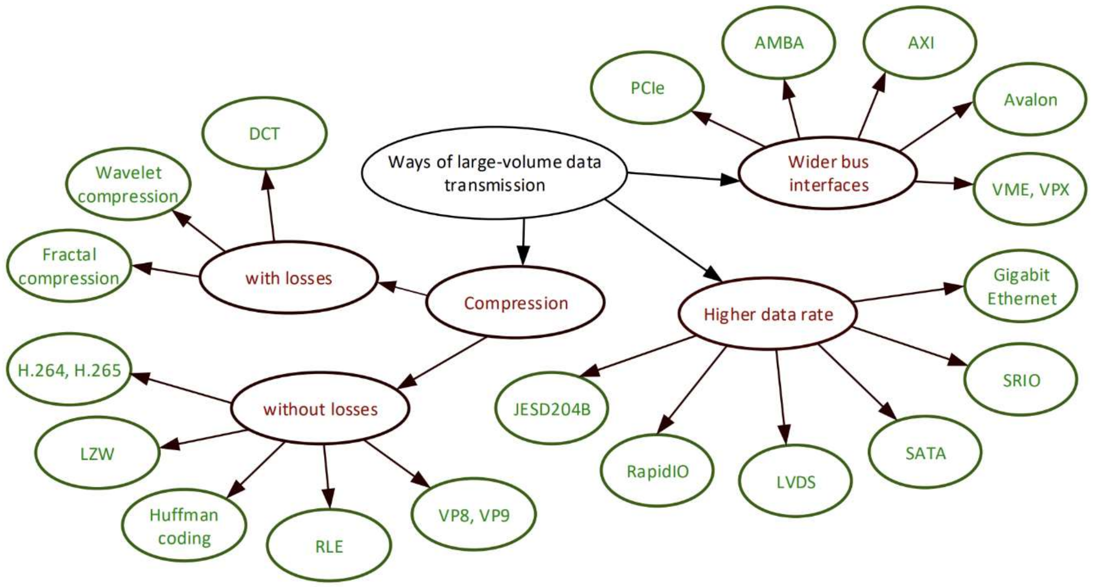

Fig. 1. Data transmission solutions

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 not. The term of the “information loss” means that some compression methods cannot guarantee exactly the same image after its decompression. It will definitely differ from the original image. 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.

Nevertheless, the lossless compression is obligatory for the original photo, medical image, or document image storage. For this purposes the compression programms are usually used. But the hardware compressor could both speed-up the compression, decrease the energy consumption, and probably, decrease the stored file volume.

This article considers the questions of the lossless compression algorithm selection and its hardware implementation issues.

2. Selection of a compression method

There are some points about the selection of the compression method. In general, all the lossless 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 the compression methods in [1] it was mentioned that the run-length encoding methods do not give a good compression ratio, therefore, they are omitted, despite their losslessness. Only two variants are left to choose from: the statistical method or the dictionary method. The general scheme of a compression method consists of two parts: the model and the coder. The model finds the redundancy in the input data and sends it to the coder, which replaces the repetitive fragments with the corresponding codes. But there is a difference between statistical and dictionary methods, so the short overview should be provided to clear the details.

2.1 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 the Markov stochastic modeling. In the paper [4] it was mentioned that the compression ratio of the statistical methods is limited by the usage of the multi-symbol alphabet zeroth-order modeling. On the other hand, the speed of the compression is limited by the use of the binary alphabets in the high-order modeling. The author's explanation of it is in that the data in the binary symbol alphabet, only a few bits are processed in each cycle. Finally, the paper [4] represents the next features of the hardware implementation:

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

2.2 Dictionary methods

In contradiction to the statistical methods, the dictionary methods provide compression using a special dictionary which is pre-determined, or filled during the input data processing. Such a technique does not use a statistical model or variable-sized codes. Each subsequence of bits from the input data stream is represented as a token, or a record, in a compression dictionary.

Depending on the method, the dictionary may be static or dynamic. The static one is filled before the compression process is started and the repeated blocks of data are replaced only in case when they are available in the pre-determined dictionary. The dynamic dictionary is adaptive and is partially filled at the step of initialization and then appended with the new records during the data processing.

When the dictionary is set up, the compression of input data is performed in a way of replacing the repeated portions of bits, strictly according to the dictionary table. In addition, such compressors are not narrowly focused on target data format, and may be used for general purposes. They can compress the audio data as well as text, what makes them popular.

In a case of implementation of a compressor with a wide application spectrum, the dynamic dictionary is more suitable than the static one [7]. Furthermore, in accordance to the principle of the data processing, there is an assumption that the dictionary compressing methods are more suitable for the hardware implementation.

The paper [4] also supports the hypothesis that dictionary compression methods are better to be implemented in hardware then the statistical methods. First of all, they are due to achieving good throughput and the competitive compression ratio. In addition, these methods are good for compression of non-streaming data, what widens its area of application. So, to clear up the benefits of using the dictionary compression methods for hardware implementation some examples should be provided.

All four software and hardware examples, described in [4] use the derivatives methods from the Lempel-Ziv-1 (LZ1) algorithm. As a first example an ALDC algorithm was represented which is implemented in a 0.8-um CMOS technology and clocked at 40 MHz obtaining a throughput of 320 Mb/s. This algorithm was developed by IBM which is used in utilities like Pkzip and ARJ. The implementation of AHA coprocessor gives a performance of the same 320 Mb/s with the 40MHz clock frequency but in the 0.5-um CMOS technology.

The next example is the STAC/Hifn device, representing the LZS algorithm. It was implemented in a 0.35-μm CMOS technology, was clocked at 80 MHz and showed a throughput of 640 Mb/s. This device consists of a full-duplex architecture meaning that it can compress and decompress the data simultaneously. Both of these chips use the CAM memory to store the dictionary and enable the parallel searching and adaptation.

Another example of the hardware implementation of a dictionary-based method is a PE-based processing element architecture for LZ1 algorithm. With the constant data input rate the post-layout simulation showed a performance of 700Mb/s in the 0.5-um CMOS technology. However, this implementation is applicable only for compressing the ASCII coded models due to the 7-bit basic symbol width.

In comparison to the dictionary methods, the statistical methods showed worse performance characteristics. The same paper [4] describes an example of a chip representing a tenth-order Markov model with the associated binary arithmetic coder, which is implemented in a 0.8 μm CMOS technology and is clocked at 25 MHz. Its compression ratio is in the order of 0.5, while the speed is data dependent but typically is around not impressing 3Mbit/s.

The examples with the Huffman coding technology showed better performance, but worse than LZ1-based ones did. The first one showed 95.2 Mb/s for compression and 60.6 Mb/s for decompression in a 2-μm SCMOS technology with a clocking frequency of 83.3MHz. To achieve this result a CAM memory modules were used to speed up the tree adaptation process.

3. LZW compression method

The goal of the current research is to find a way to improve the existing GIF [5] image format. The main benefit of using GIF is that the 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 main architectural feature of this FPGA implementation is an FPGA-suitable hash table that consists of buckets each of which is composed of 8 entities. Each entity stores a 12-bit pointer, 8-bit character, and 12-bit back pointer. The data table is divided into 8 parts what facilitates the reading of 8 values at one time. Having a back pointer in a record makes easier the search of included values without checking eight entities in the bucket one by one. Also, there is a 4-bit value in the bucket record which can easily determine if the element is already stored. For this hash table was used three operations: initialization, search, and adding.

As for the hardware aspects, in order to implement that hash table was used block RAMs, configured in dual-port mode. The total amount is eighteen 18 Kb block RAMs. The final device was implemented on a circuit with the Xilinx Virtex-7 FPGA. Implementation of 1 instance will take:

- 104 (0.02% of available on FPGA) Slice registers;
- 346 (0.11%) Slice LUTs;
- 18 (0.87%) 18K block RAMs;
- The clock frequency is 179.99 MHz.

The experimental results showed not a big difference with sequential LZW compression on the Intel Core i7-4790 with a 3.6 GHz clock frequency. If the test image has more common regions the sequential implementation is even faster. Testing one image the FPGA speed-up factor was 0.34:1 over the CPU. But if we take a look at the percentage of used hardware components, only a small part of them were actually active. If the implementation consisted of 24 circuits, the hardware parameters are the next:

- 3120 (0.51%) Slice registers;
- 7782 (2.56%) Slice LUTs;
- 432 (20.97%) 18K block RAMs;
- The clock frequency is 163.35 MHz.

However, the maximum clock frequency decreases with the growth of instances the results showed that such a solution gives a speed factor up to 23.51 over the sequential implementation on the CPU [9].

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, the 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 to 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 the algorithm might be scalable?

4. Implementation aspects

The proposed method of the 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 allows interconnects FPGA interfaces with the ARM processing core [16]. Fig. 2 represents the scheme, where can be seen the connections between each part 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, each image on each processor for the frame sequence can be distributed. And, properly to the sequence, the compressed frames are added 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.

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 access to the proper nodes.

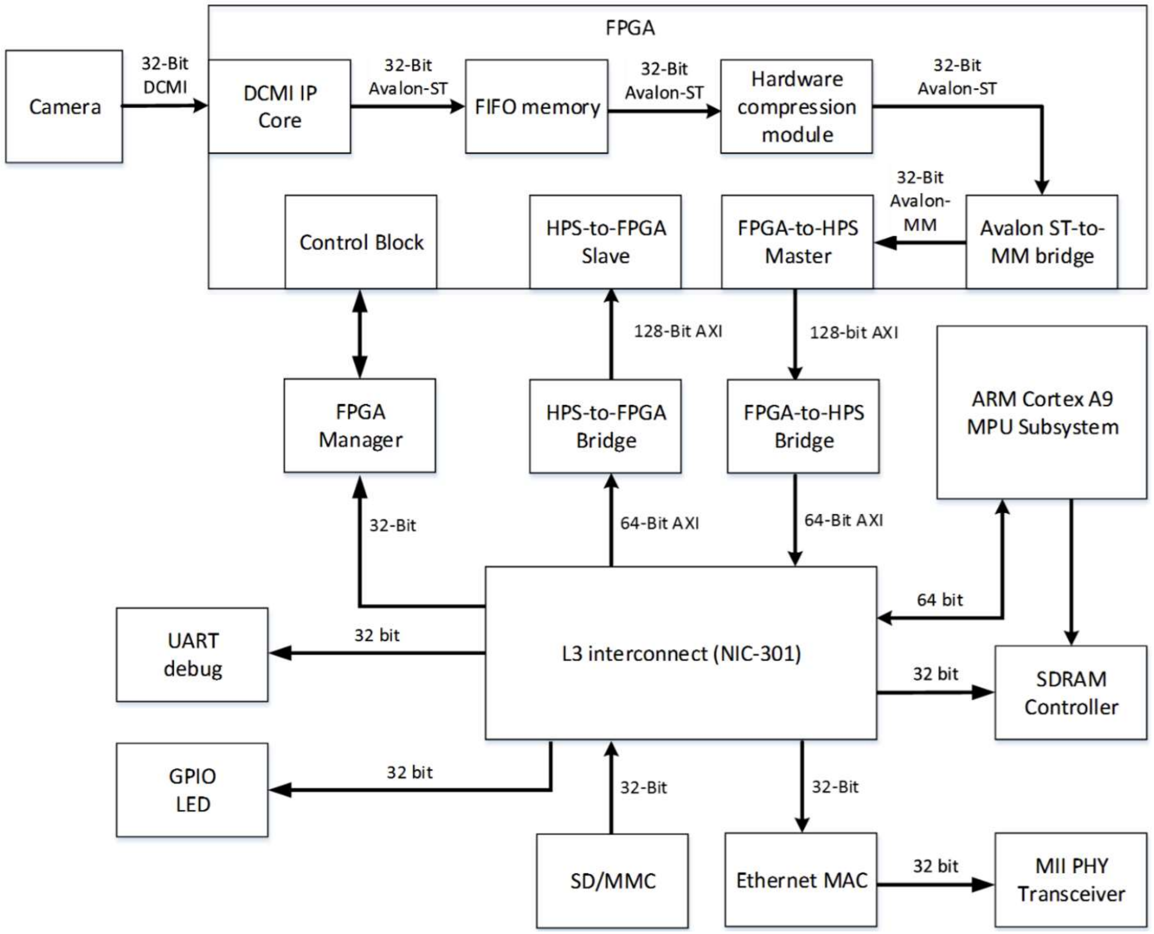

Fig. 2. Final Device Scheme

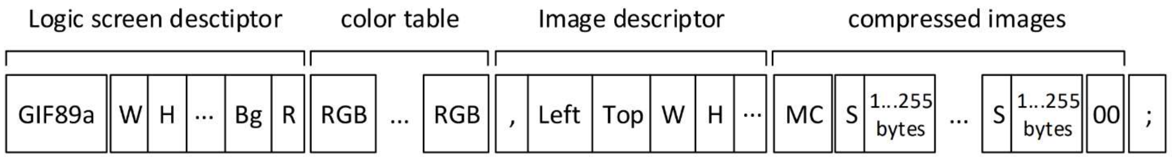

Fig. 3. GIF File Structure

Table 1

Tree-based dictionary nodes

	Nodes		
Address	97	268	297
Value	(-: «a»)	(97: «b»)	(266: «c»)

The Value is the combination of the ancestor's address and the default record. For example, if we have the text string “ABC”, the node will have the next structure:

Address 297 is the newly assigned address of the node. 268 is the address of the ancestor, 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. We decided to use the AVL [18] structure because it is balanced, and the balancing process is performed on a step of adding a new node.

The 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.

5. Discussion

Summarizing the above study, the developed hardware compression unit does not show breakthrough characteristics. However, during analyzing the FPGA resources that are actually used [16], it can be seen that it has enough space to implement at least 20 instances of the hardware part of the investigated compression unit. Moreover, it is not the highest-performance Intel FPGA, which can be offered. 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 the image compressors can be overcome.

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 aspects of future research can also be dedicated to developing the image compression unit embodiment which is optimized by the hardware volume and low-power consumption criteria.

6. Conclusions

This paper describes, in general, the scientific problem of data transmission, and what benefits data compression gives, comparing to other solutions. To choose the correct way of further research, all the factors were cleared up and given some proves about choice of compression method to implement. The main benefit of using dictionary compression methods against statistical was the compression speed. As a compression method to review, the LZW algorithm was chosen for several reasons. The first one is that the LZW method performs data compression without losses. In addition, this compression method is used in GIF files compression, so it seems that it is good for both data and image compression.

To have an efficient implementation, some aspects were discussed: what parts of the algorithm implementation should be parallelized and pipelined; what processing stages depend on the results of the previous ones; what parameters might be scalable. To show a possible implementation as a final device was shown the scheme with all the main processing units and interconnection between them. It was described as a device, implemented using the technology of "System-on-Chip" which represents a complex device of an FPGA with the ARM processor on a single chip. This technology is widely used nowadays which makes it possible to use the ready-made drivers and solutions to implement the compressor as a final device, so facilitate its development. In the discussion section, the possible ways of the future research were presented.

References

[1] M. Sharma, "Compression Using Huffman Coding," *JCSNS International Journal of Computer Science and Network Security*, vol. 10, no. 5, May 2010.

[2] H. Rubaiyat, "Data Compression using Huffman based LZW Encoding Technique," *International Journal of Scientific & Engineering Research*, no. 11, 2011.

[3] A. Taleb, H. Mustafa, A. Khtoom, and I. Gharaibeh, "Improving LZW image compression," *European Journal of Scientific Research*, vol. 44, no. 3, pp. 502–509, Aug. 2010.

[4] K. Papadopoulos and I. Papaefstathiou, "TitanR: A Reconfigurable Hardware Implementation of a HighSpeed Compressor," in *2008 16th International Symposium on FieldProgrammable Custom Computing Machines*, pp. 216–225. doi: https://doi.org/10.1109/FCCM.2008.14.

[5] CompuServe Incorporated, “CompuServe Graphics Interchange Format (GIF),” 1987

[6] J. Miano, *Compressed image file formats: JPEG, PNG, GIF, XBM, BMP*. New York: Addison-Wesley, 1999.

[7] D. Salomon, *Data Compression*. London: Springer London, 2007. doi: https://doi.org/10.1007/978-1-84628-603-2.

[8] T. Welch, “A Technique for High-Performance Data Compression,” *Computer*, vol. 17, no. 6, pp. 8–19, Jun. 1984, doi: https://doi.org/10.1109/mc.1984.1659158.

[9] X. Zhou, Y. Ito, and K. Nakano, “An Efficient Implementation of LZW Compression in the FPGA,” in *International Conference on Algorithms and Architectures for Parallel Processing*, Jan. 2016, pp. 512–520. doi: https://doi.org/10.1007/978-3-319-49583-5_39.

[10] W. Cui, “New LZW data compression algorithm and its FPGA implementation,” in *Picture Coding Symposium*, Jan. 2007.

[11] P.-S. Yeh, “Implementation of CCSDS Lossless Data Compression for Space and Data Archive Applications,” *SpaceOps 2002 Conference*, Mar. 2002, doi: https://doi.org/10.2514/6.2002-t5-12.

[12] “HDMI 2.1 Overview,” *HDMI Forum, Inc*, 2017. hdmi.org (accessed Jan. 10, 2017).

[13] “HDMI 2.1 Press Release,” *HDMI Forum, Inc*, 2017. hdmi.org (accessed Jan. 10, 2017).

[14] P. Heydari, “Design and analysis of lowvoltage currentmode logic buffers,” in *Fourth International Symposium on Quality Electronic Design, 2003. Proceedings.*, pp. 293–298. doi: https://doi.org/10.1109/ISQED.2003.1194748.

[15] A. Boni, A. Pierazzi, and D. Vecchi, “LVDS I/O interface for Gb/sperpin operation in 0.35/spl mu/m CMOS,” *IEEE Journal of SolidState Circuits*, vol. 36, no. 4, pp. 706–711, doi: https://doi.org/10.1109/4.913751.

[16] “Cyclone V Hard Processor System Technical Reference Manual,” *Intel*, 2018. https://www.intel.com/content/www/us/en/docs/programmable/683126/17-1/introduction.html

[17] “DE0-Nano-SoC User Manual,” Terasic Inc, 2019. Available: https://www.terasic.com.tw/attachment/archive/941/DE0-Nano-SoC_User_manual_rev.D0.pdf

[18] M. Adelson-Velskii, “An algorithm for organization of information,” *Doklady Akademii Nauk SSSR*, pp. 263–266, 1962.