# FPGA implementation of Lempel-Ziv data compression

**Gehad Mohey[1], Abdelhalim Zekry[2], Hatem Zakaria[3]**
[1]Electronics and Communications Engineering Department, El-Madina Higher Institute for Engineering and Technology, Egypt
[2]Electronics and Communications Department, Faculty of Engineering, Ain shams University, Egypt
[3]Electrical Engineering Department, Benha Faculty of Engineering, Benha University, Egypt

**Article Info**

***Article history:***

Received Oct 30, 2020
Revised Jan 10, 2021
Accepted Apr 29, 2021

***Keywords:***

Data compression
FPGA
Lossless compression
LZ77
LZSS
Systolic array design
VHDL

**ABSTRACT**

When transmitting the data in digital communication, it is well desired that the transmitting data bits should be as minimal as possible, so many techniques are used to compress the data. In this paper, a Lempel-Ziv algorithm for data compression was implemented through VHDL coding. One of the most lossless data compression algorithms commonly used is Lempel-Ziv. The work in this paper is devoted to improve the compression rate, space-saving, and utilization of the Lempel-Ziv algorithm using a systolic array approach. The developed design is validated with VHDL simulations using Xilinx ISE 14.5 and synthesized on Virtex-6 FPGA chip. The results show that our design is efficient in providing high compression rates and space-saving percentage as well as improved utilization. The Throughput is increased by 50% and the design area is decreased by more than 23% with a high compression ratio compared to comparable previous designs.

***Corresponding Author:***

Gehad Mohey
Department of Electronics and Communications
El-Madina Higher Institute for Engineering and Technology
Faisl street, Giza, Egypt
Email: gehadeldonea91@gmail.com

## 1. INTRODUCTION

Computers can deal with several different sorts of data like text, games, sound, photos, and film. A percentage of these information sources need a large amount of data which can also quickly fill up your hard disk or take a long time to transmit over a network. It is regularly an issue to be able to store a lot of digital information using a limited amount of space. For this reason, it is interesting to check if the data can be rewritten such that it takes up less space. This may appear like magic, but does, in fact, work well for some data types. Data compression is used multimedia formats for images, Video and audio [1, 2]. The lossless data compression indicates that data is the same at the source and destination [3, 4]. Huffman code [5, 6], run-length code [7], arithmetic code [8], and Lempel-Ziv (LZ) compression algorithms [9] are a widely used [10] lossless data compression technique. Among them, the LZ algorithm that is a dictionary-based algorithm that can achieve an average compression ratio for lossless data compression and is considered universal. Statistical lossless data compressors are better than dictionary-based in cost, area requirement and compression ratios [11]. In the hardware implementation of dictionary-based methods, three approaches are distinguished: CAM (Content Addressable Memory) approach [12], the microprocessor approach [13] and the systolic array approach [5]. The main advantage of the Systolic array approach is that it is easily implemented and a higher clock rate can be achieved [14]. Comparison between the three approaches is shown in Table 1. Due to the considerable amount of parallel comparison involved by LZ algorithm, so achieving a very high throughput using software approaches may be difficult. Systolic Array approach will be used in this research to achieve high throughput with lower H/W requirements.

The remaining of this paper is organized as: This paper consists of six sections; the related work is explained in section 2. LZSS compression algorithm is explained in Section 3. Section 4 describes the systolic array design for LZ. Section 5 contains the simulation and implementation results of our design. Finally, conclusions are given in section 6.

Table 1. Comparison between hardware approaches of dictionary-based method

| Approach | Features |
|---|---|
| Microprocessor | Large amount of available flexibility-loss of performance-does not completely explore parallelism |
| Content addressable memory (CAM) | Very fast-high power consumption-expensive due to high hardware requirements-full paraller searching |
| Systolic array | Lower hardware requirements-better testability-pipeline searching–higher clock rate |

## 2. RELATED WORK

Since lossy data compression allocates the bits necessary for data restoration within a specified fidelity level measured by a distortion feature. This theory is called rate-distortion [8]. In lossless data compression [7], the data should be precisely reconstructed [15]. Lempel–Ziv compression method is a dictionary method based on the substitution of text substrings with its previous occurrences. The Lempel-Ziv compression dictionary starts with a certain predetermined state, but during the encoding process, the content changes depending on the data that has already been encoded. LZ77 [9] and LZ78 [16] are the most famous algorithms. LZSS is the most popular versions of LZ77 [17-19]. There are many researches works on LZ design for data compression. We will introduce some of the recent and previous works such as in [20], in [14] and in [21].

In [22], Marsh and Knapp presented a detailed analysis of how the size of the buffers in the LZ77 algorithm affects the throughput and compression ratio. By choosing a specific buffer size, the required area can be evaluated, the compression ratio, and the throughput that the compressor can achieve. Using a Xilinx XC2V1000 FPGA device, the implementation of the compressor was done using a 512-byte search buffer and a coding buffer of 15-byte. Based on post-layout simulations, architecture can achieve a 11 Mbps throughput. In [23], By using systematic design methodologies, an area/power- architecture for LZ data compression was developed. In order to indicate early completion, they used a control variable to improve the latency. Their architecture allows a high-level understanding of the tradeoffs involved. By using a common estimation framework, a broad range of options can be considered, since the architecture is scalable and parameterized.

In [24], A LZ compression parallel algorithm was described by Mohamed A. Abd El Ghany. To display early completion, a control variable was used to further improve the latency. The proposed implementation is efficient in terms of speed and area requirements. The design area is decreased by more than 30% and the compression rate is increased by more than 40%. His compression rate was about 13Mbps. In [25], Design and FPGA implementation for GZIP compressor based systolic array was presented. A single GZIP compression core was implemented in Virtex 6 FPGA ML605 development board, data transfers Xillybus utilization was done over PCI Express. The throughput of their implementation was over 1.3 Gbps and the software average throughput was 52 Mbps using the Calgary corpus. In [26], H.Luo, Ye Cai, and Q.Mao presented a multi-core GZIP compressor for HDFS. To increase throughput, the core was designed via expanding multiple systolic array compression cores. The Hardware implementation was evaluated using Alpha Data Adm-Pcie-KU3 FPGA development board, RIFFA data transfers utilization was done over PCI Express. The peak throughput of the compressor exceeds 1.1 GB/s.

## 3. LZSS COMPRESSION ALGORITHM

LZSS is one of the improvements of LZ77 that will be used in this paper. a window ($n$ = 9) shown in Figure 1 and look-ahead buffer ($L_s = 3$) as an example. Assume that $X_i$, $i$ = 0, 1... $n\text{-}1$ will be represented as the window content and that $Y_j$, $j$ = 0, 1… $L_s\text{-}1$ (i.e., $Y_j = X_{i+n-Ls}$) as the look-ahead buffer content. The look-ahead buffer content is compared with the content of the dictionary according to LZ concept to find the length of the longest match Lmax which start from $I_p$ position. Then output will be represented by a codeword ($I_p$, $L_{max}$). The code word length $L_c$ is given by:

$$Lc = log2(n - Ls) + Log2\,(Ls) + 1\,bits \qquad (1)$$

To represent a symbol in the window, *w* bits are needed, l = *$log_2$ ($L_s$)* bits to represent $L_{max}$, and $p$ = *$log_2$ (n-Ls)* bits to represent $I_p$. Then (l + *p*) / ($L_{max}$ * *w*) is the compression ratio.

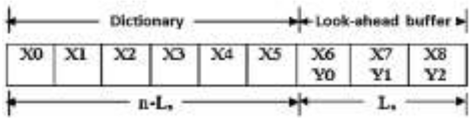

Figure 1. Example of LZ compressor window

In the DG I Figure 2, match length and match signal are propagated from cell to cell. The window content (X) and the look-ahead buffer content (Y) are broadcast to all cells horizontally and diagonally respectively. By the DG projection into the surface normal to the projection vector selected, the processor assignment can be done.

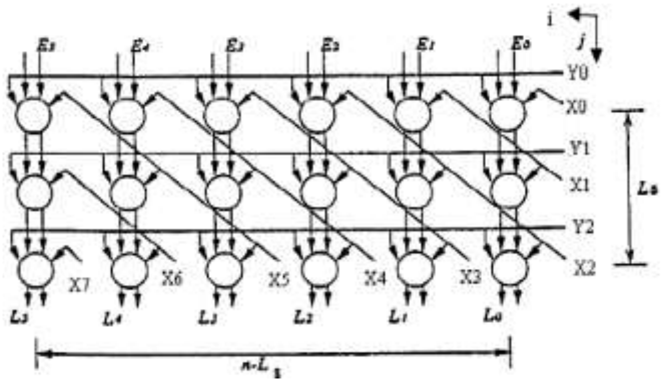

Figure 2. Dependence graph of the LZ compression algorithm

## 4. SYSTOLIC ARRAY DESIGN FOR LZ DATA COMPRESSION

The compression design of Lempel Ziv is shown in Figure 3. The systolic array design architecture consists of three major components: the SALZC compressor module, the RAM block, and the host controller. SALZC module doesn't include block RAM. The dictionary size can be increased by exchanging the block RAM with a larger one. Also, the host controller is not combined into the SALZC module, to be able to modify when the dictionary size is changed. The window size length *n in our* implementation 1K, and the length of look-ahead buffer $L_s$ = 16.

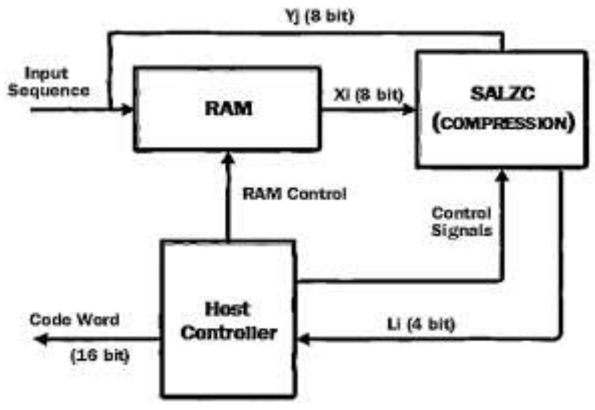

Figure 3. Lempel-Ziv compression chip

### 4.1. SALZC module

SALZC module contains 16 processor elements (PES), one L-encoder, 16 bytes shift register, and 4-bit counter. The SALZC module is depicted in Figure 4.

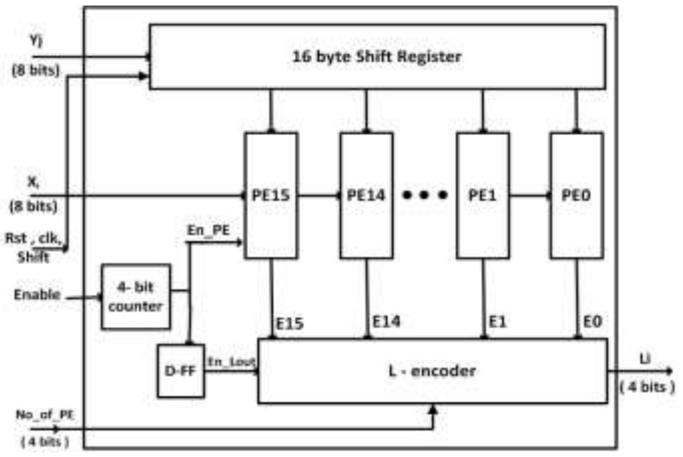

Figure 4. SALZC module block diagram

From the DG shown in Figure 2, all the nodes in a specific row are integrated into a single processor element (PE). A linear array of length $L_s$ is produced. The layout is simple due to the regularity of the array. A single cell (PE) only will hand – laid out, then the other 15 PEs are its copies. Routing is also simplified due to systolic array design. The resulting array of Design-P are given in Figure 5 and the space-time diagram is shown in Table 2.

As shown in Figure 5 the architecture consists of 16 processing elements that is used for comparison, and L-encoder that is used for matching length output. Thus, the look-ahead buffer symbols Yj that remain in PEs during the encoding step and do not change. The Xi dictionary variable moves systolically from left to right, with 1 clock cycle delay. The processing element's match signal Ei moves onto the L-encoder. The encoder's output Li is the matching longitude resulting from the i-1 comparisons. After one clock cycle, the first Li will be obtained and each clock cycle will be obtained for the following ones. The Yj is preloaded to be processed before the encoding process and this will take Ls extra cycles. The time of preloading new source symbols during the encoding process depends on the number of source symbols will be compressed in the preceding compression step, Lmax.

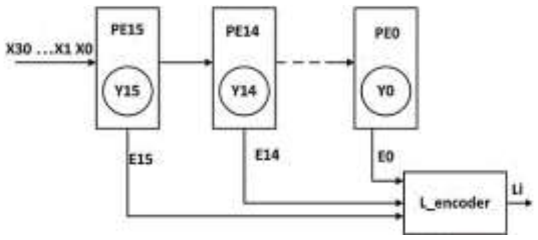

Figure 5. Array of design-p

Table 2. Space-time diagram

| space / time | PE15 | PE14 | …. | PE1 | PE1 | Li |
|---|---|---|---|---|---|---|
| 1 | X15– Y15 | X14– Y14 | …. | X1– Y1 | X1– Y1 | |
| 2 | X16– Y15 | X15– Y14 | …. | X2– Y1 | X2– Y1 | L0 |
| 3 | X17– Y15 | X16– Y14 | …. | X3– Y1 | X3– Y1 | L1 |
| …. | …. | …. | …. | …. | …. | …. |
| 16 | X30– Y15 | X29– Y14 | …. | X16– Y1 | X16– Y1 | L14 |
| 17 | | | | | | L15 |

The PE block diagram is presented in Figure 6. The comparison of $Y_j$ and incoming $X_i$ requires only one equality comparator. The $E_i$ (match signal) result for the comparator propagates to the L-encoder. The L-encoder block diagram is depicted in Figure 7. The match-length $L_i$ is computed according to match signals.

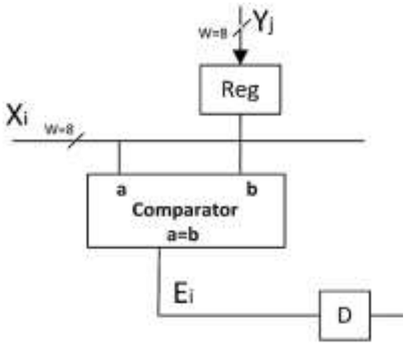

Figure 6. Functional block of a processing element

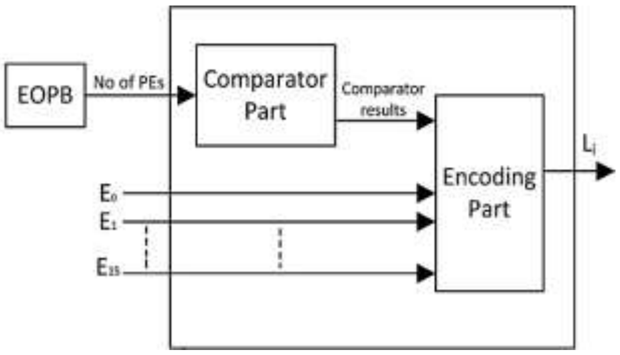

Figure 7. The l-encoder

### 4.2. Host controller

The Host controller includes match results block (MRB), code word generator, and end of processing block (EOPB), as shown in Figure 8. From Figure 5, it is clear that the L-encoder doesn't generate the maximum matching length. So, in order to determine $L_{max}$ among the generated $L_i$s', a match results block (MRB) is needed as shown in Figure 9.

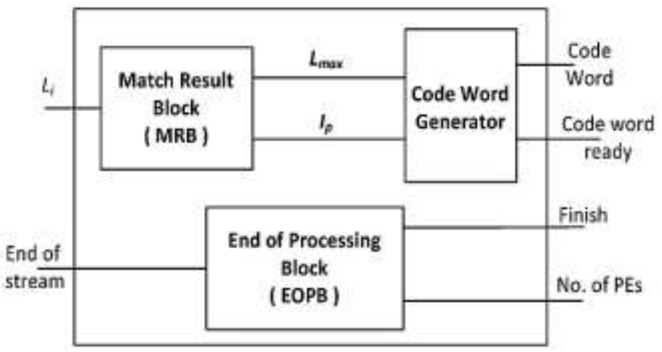

Figure 8. Host controller block diagram

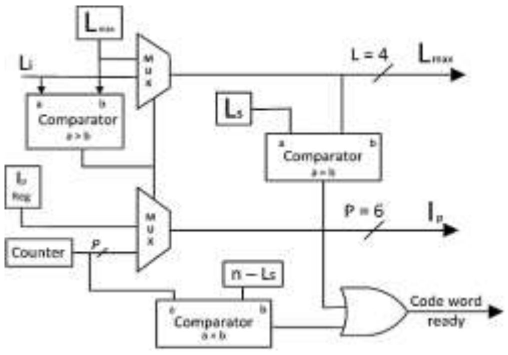

Figure 9. Match results block

The end of processing block as shown in Figure 10 includes a 4-bit counter and Determination block (DB). This counter is needed to successfully handle the last part of the data stream. End of stream signal does not mean the end of the compression operation, but once the end of stream signal is generated using the 4-bit counter It's used to trigger the encoding process of the unprocessed data in the look-ahead buffer. After receiving the enable signal the counter will count the number of shift operations. DB determines the number of process elements that will operate during the encoding step according to the counter output and generates the end signal after the compression operation is complete. Determination block (DB) is shown in Figure 11. Without the DB the last part will be compressed incorrectly. The number of PEs in the forward buffer should be equal to the number of unprocessed data. Comparator and Subtractor are the principal components of DB. If the counter output (the number of data processed in the look-ahead buffer) is less than the number of PEs, they can be subtracted by the Subtractor. The number of PEs is created which will operate during the encoding stage. If the counter output is equal to the PES number, it means the entire look-ahead buffer data is processed. Hence the end signal (finish) will generate.

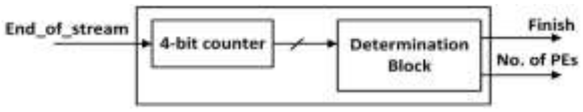

Figure 10. End of processing block (EOPB)

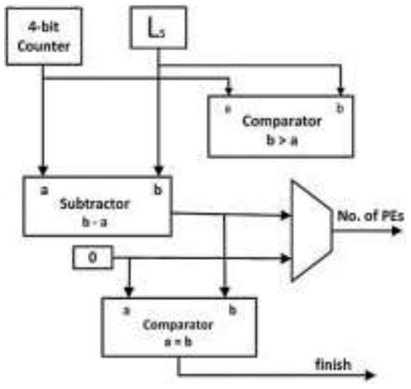

Figure 11. Determination block (DB)

### 4.3. Block RAM

We use the block RAM as the first-in-first-out (FIFO), so we need to use two counters, as illustrated in Figure 12. The first one is to generate a write address. At first, it is loaded by the look-ahead buffer's first address, then it counts to initialize the look-ahead, buffer. Afterward, it will point to where an input symbol should be inserted. The second one is to generate the address for reading. It will point to the FIFO's first location (equal to the address written + 1). Upon reaching the maximum value one of two counters. In the next step, it'll immediately go down to 0.

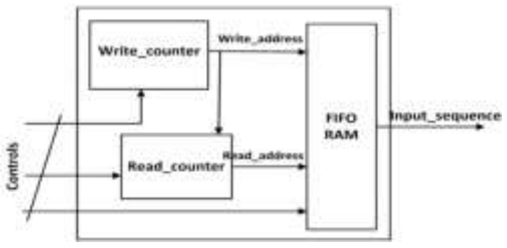

Figure 12. First-in-first-out (FIFO)

## 5. RESULTS

### 5.1. Software simulations

The RTL architecture of SALZC module depicted in figure 4 is VHDL modeled with its simulation result as shown in Figure 13. The SALZC receives a sequence of 16 bytes of data from a text vector file. Thus, the first 16-bytes of data stored in $Y_j$ then it reads $X_i$ and then it compared $Y_j$ with $X_i$ and the result is in $L_i$ and $Y_0$-out since $L_i$ = 1111 and this is due to the first 16-byte of $X_i$ equal the first 16-bytes of $Y_j$ and $Y_0$-out = 01110011 and this is due to the first byte of file = 01110011.

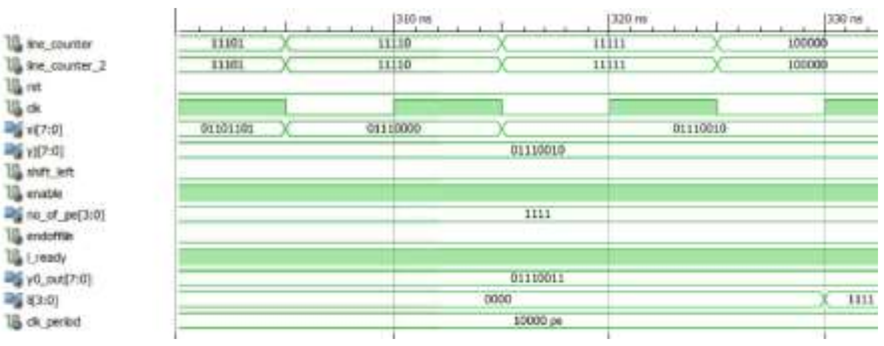

Figure 13. Simulation results of SALZC module

The simulation result of the Host controller is shown in Figure 14. The code will output the code word due to the received signal from SALZC since if there is no match it will output codeword that contains $M_0$ if there is a match it will output the code word that contains Length of the match and its pointer. The first bit in the codeword specifies that if there is a match or not.

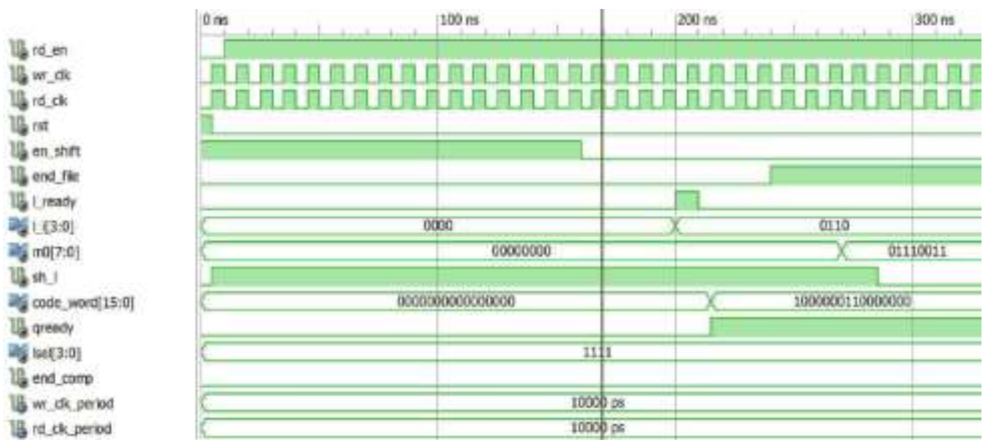

Figure 14. The host controller simulation result

The code also will do shift if *en-shift* = 1 or if *load* =1 since *en-shift* is a control signal to do 16 shifts initially then if *load* =1 it will load a new byte. The Host controller output also depends on *L-ready*, which shows that the match is ready or not. $L_i$ shows the length of the match and according to this length, the code will do shift (Li number of bytes). If we assume that $L_i$ = 0110 then *Q-Ready* = 1 then *shift-left* = 1 for 6 clock cycle then shift left return to zero waiting for a new condition of Li or load if there is no match. as shown in Figure 15 Window act as a dictionary in our code since RAM is FIFO its depth = 1024 and width = 8. The code reads the data input from the text file then the output is.

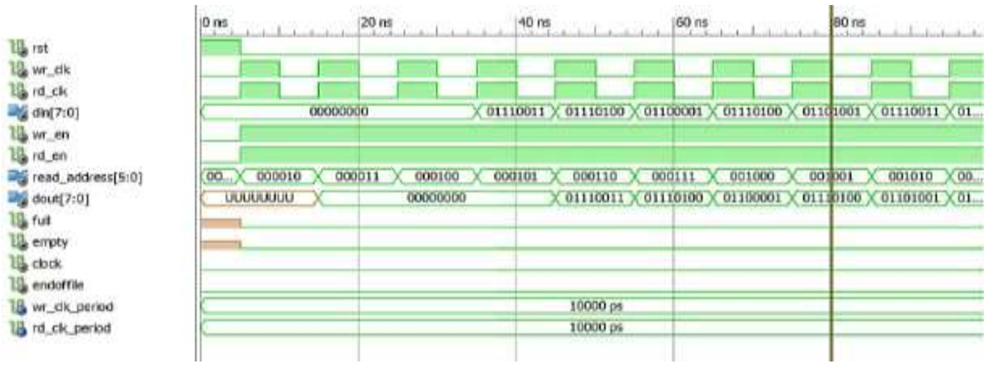

Figure 15. Window simulation result

After verifying the VHDL code of all the component Window, SALZC and Host controller, the match-length of comparison and the first byte stored in the first PE is fed to Host controller then it decides if it was a match-length then it compares it with the maximum length stored previously then it outputs the codeword that consists of (16-bits) contain match-length of compression and the pointer of this length, then it does several shifts equal to the match-length and load a new number of byte to the shift register and compare again. If it wasn't a successful comparison it output the first byte that was stored in the first PE and it does one shift (load one new byte) and do the comparison again as shown in Figure 16. If it has a match-length after the comparison, SALZC module has a signal that shows that the code has a match-length as shown in Figure 16 (*Q_ready*) signal = 1 at the time the output has a length and pointer and the first bit of the codeword equal one this is another verify for the output, but if the output has zero length it will out a signal (*load*) = 1 that verify there is no correct comparison.

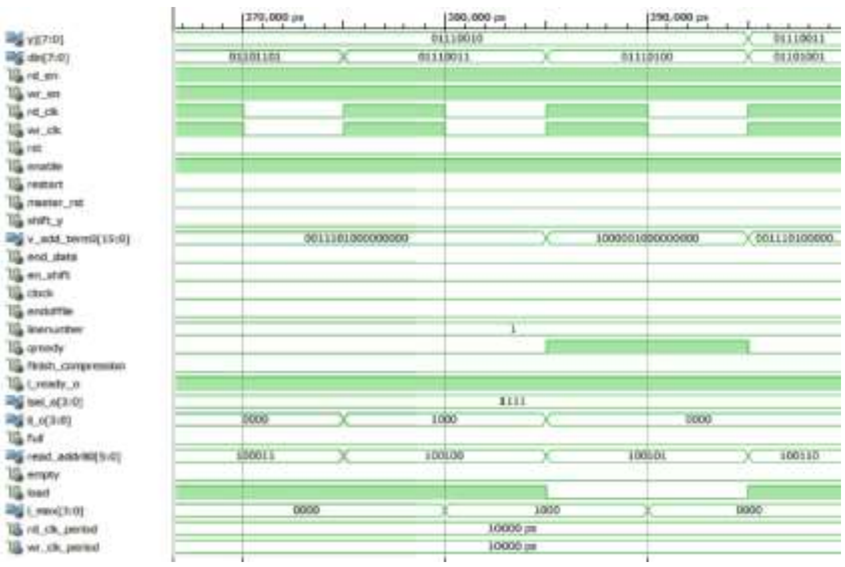

Figure 16. The LZ compression chip simulation result

### 5.2. Implementation

In this section, we present the achieved design lossless compression efficiency. The implementation of our design is carried out using Xilinx Virtex-6 FPGA, for $n$ = 1k, $L_s$ = 16, $w$ = 8. FPGA utilization summary is shown in Table 3. The compression rate $R_c$ can be estimated as:

$$Rc = clk \times (Ls * W)/(n - Ls + 1) \tag{2}$$

In our implementation, we use window size (n) is 1K, $L_s$= 16, $w$= 8, and $CLK$= 175.408, Our module does space-saving about 55% and on average compression rate up to 25.75 Mbps. Saving percentage in our FPGA implementation is 55% and the compression ratio is 67.8%. The total on-chip power is 3.422 W.

Table 3. Implementation result of the proposed design

| Device utilization summary (estimated values) | | | |
|---|---|---|---|
| Logic Utilization | Used | Available | Utilization |
| Number of Slice Registers | 373 | 301440 | 0% |
| Number of Slice LUTs | 262 | 150720 | 0% |
| Number of Fully Used LUT-FF Pairs | 117 | 518 | 22% |
| Number of bounded IOBs | 65 | 600 | 10% |
| Number of Block RAM/FIFO | 1 | 416 | 0% |
| Number of BUFG/BUFG/CTRL/BUFHCEs | 3 | 176 | 1% |

Table 4 depicts the comparison between Compression rate of the proposed design and the literature. In [26], presents parallel multi-core GZIP compressor via HDFS and implemented the design using Adm-Pcie-KU3 FPGA device. They achieved compression rate about 22%. In [25], presented a design and implementation of a complete GZIP core architecture. They do the implementation using Virtex 6 ML605 and achieved compression ratio about 21.7%. In [24], Presents design and implementation of LZSS using Sparten-II FPGA device and achieved 13 Mbps. Compared to the results in [16] the throughput is increased by 50% and the design area is decreased by more than 23% that provides an excellent platform for Real-time compression applications.

Table 4. Compression rate and compression ratio comparison

| Design | Dictionary size | Compression Ratio | Compression Rate | FPGA Device |
|---|---|---|---|---|
| Design, 2019 | ------- | …… | 22% | Adm-Pcie-KU3 FPGA |
| Design, 2017 | 1024 | ….. | 21.7% | Xilinx Virtex 6 |
| Design, 2007 | 1024 | 13 Mbps | ….. | Sparten-II |
| The proposed design | 1024 | 26 Mbps | 67.8% | Xilinx Virtex 6 |

## 6. CONCLUSIONS

In this paper, the design and implementation of lossless data compression was described using the LZ algorithm. Xilinx ISE 14.5 tool is used. The programming is done in VHDL language and the whole algorithm is described in that language. Our systolic array LZ compression (SALZC) module provides space-saving about 55% and on average compression rate up to 25.75 Mbps. Comparing to literature work we proved that LZSS based systolic array design can achieve high compression ratio compared to GZIP and also can achieve high compression rate compared to other LZ design. As future work, one may modify the host controller since it can be used for other algorithms string-matching based LZ, such as LZW and LZ78.

## REFERENCES

[1] J. Latif, P. Mehryar, L. Hou, Z. Ali, "An efficient data compression algorithm for real-time monitoring applications in healthcare," *2020 5th International Conference on Computer and Communication Systems (ICCCS)*, 2020, pp. 71-75.

[2] J. Uthayakumar, T. Vengattaraman, "Performance evaluation of lossless compression techniques: An application of satellite images," *2018 Second International Conference on Electronics, Communication and Aerospace Technology ICECA*, 2018, pp. 750-754.

[3] H. D. Kotha, M. Tummanapally, V. K. Upadhyay, "Review on lossless compression techniques," *Journal of Physics: Conference Series, Volume 1228, International conference on computer vision and machine learning*, Andhra Pradesh, India, 27-28 Dec. 2018.

[4] A. Gopinath, M. Ravisankar, "Comparison of lossless data compression techniques," *2020 International Conference on Inventive Computation Technologies (ICICT)*, 2020, pp. 628-633.

[5] A. Moffat, "Huffman coding," *ACM Computing Surveys*, vol. 52, no. 4, pp. 1-35, 2019.

[6] S. T. Klein, S. Saadia, D. Shapira, "Forward looking Huffman coding," *Theory of Computing Systems*, 2020.

[7] M. Pandey, S. Shrivastava, S. Pandey, S. Shridevi, "An enhanced data compression algorithm," *International Conference on Emerging Trends in Information Technology and Engineering ic-ETITE*, 2020, pp. 1-4.

[8] C. W. Huang, J. J. Ding, "Efficient EEG Signal compression algorithm with long length improved adaptive arithmetic coding and advanced division and encoding techniques," *2018 IEEE 23rd International Conference on Digital Signal Processing DSP*, vol. 2018-Nov., no. 1, pp. 1-5, 2019.

[9] J. Ziv, A. Lempel, "A universal algorithm for sequential data compression," in *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337-343, May 1977.

[10] Parekar P. M, Thakare S. S, "Lossless Data compression algorithm-a review," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 1, pp. 276-278, 2014.

[11] A. Gupta, A. Bansal, V. Khanduja, "Modern lossless compression techniques: Review, comparison and analysis," *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, 2017, pp. 1-8.

[12] K. Pagiamtzis, A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," in *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712-727, Mar. 2006.

[13] Udaya Kumar H, "Design and Implementation of lossless data compression coprocessor using FPGA," *International Journal of Engineering Research & Technology*, vol. 4, no. 05, pp. 818–822, 2015.

[14] Shih-Arn Hwang, Cheng-Wen Wu, "Unified VLSI systolic array design for LZ data compression," in *IEEE Transactions on Very Large Scale Integration VLSI Systems*, vol. 9, no. 4, pp. 489-499, Aug. 2001.

[15] Himali Patel, Unnati Itwala, Roshni Rana, Kruti Dangarwala, "Survey of lossless data compression algorithms," *I International Journal of Engineering Research & Technology (IJERT)*, vol. 4, no. 4, pp. 926-929, 2015.

[16] J. Ziv, A. Lempel, "Compression of individual sequences via variable-rate coding," in *IEEE Transactions on Information Theory*, vol. 24, no. 5, pp. 530-536, Sep. 1978.

[17] J. A. Storer, T. G. Szymanski, "Data compression via textual substitution," *Journal of the Association for Computing Machinery*, vol. 29, no. 4, pp. 928-951, 1982.

[18] S. Belu, D. Coltuc, "RoLZ - The reduced offset LZ data compression algorithm," *2019 International Symposium on Signals, Circuits and Systems ISSCS*, 2019, pp. 1-4.

[19] G. Wang, H. Peng, Y. Tang, "Repair and restoration of corrupted LZSS files," in *IEEE Access*, vol. 7, pp. 9558-9565, 2019.

[20] T. Bell, D. Kulp, "Longest-match string searching for Ziv-Lempel compression," *Software: Practice and Experience*, vol. 23, no. 7, pp. 757-771, 1993.

[21] N. Ranganathan, S. Henriques, "High-speed VLSI designs for Lempel-Ziv-based data compression," in *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 40, no. 2, pp. 96-106, 1993.

[22] E. R. Marsh, B. R. Knapp, "On the design and implementation of an instrumented grinding testbed," *Sensor Review*, vol. 25, no. 2, pp. 155-161, 2005.

[23] B Bongjin Jung, W. P. Burleson, "Efficient VLSI for Lempel-Ziv compression in wireless data communication networks," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 3, pp. 475-483, 1998.

[24] M. A. A. El ghany, A. E. Salama, A. H. Khalil, "Design and Implementation of FPGA-based Systolic Array for LZ Data Compression," *2007 IEEE International Symposium on Circuits and Systems*, 2007, pp. 3691-3695.

[25] O. Plugariu, A. D. Gegiu, L. Petrica, "FPGA systolic array GZIP compressor," *2017 9th International Conference on Electronics, Computers and Artificial Intelligence ECAI*, 2017, pp. 1-6.

[26] H. Luo, Y. Cai, Q. Luo, R. Mao, "FPGA-based parallel multi-core GZIP compressor in HDFS," *2019 20th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, 2019, pp. 31-35.

## BIOGRAPHIES OF AUTHORS

**Eng. Gehad Mohey** was born on April 1991. She received the B.Sc. With Honor in 2013 in Computer Department, from Benha Faculty of Engineering at Benha University, Egypt. She is a demonstrator at Communication Engineering Department – Madina Higher Institute of Technology – Egypt. Her research interest includes embedded systems and digital systems design.

**Prof. Abdelhalim Zekry** is a professor of electronics at faculty of Engineering, Ain Shams University, Egypt. He worked as a staff member in several universities. He published more than 250 papers. He also supervised more than 104 Master thesis and 28 Doctorate. Prof. Zekry focuses his research programs on the field of microelectronics and electronic applications including communications and photovoltaics. He got several prizes for his outstanding research and teaching performance.

**Dr. Hatem M. Zakaria** received his Ph.D. degree in 2011 in Micro and Nano electronics from Grenoble University, Grenoble, France. He is currently an Assistant Professor at MSA University. Dr. Hatem has more than 13 years of experience in asynchronous circuit design, nanoscale CMOS technology and Complex SOC design techniques. His research interests include hardware implementations of FEC algorithms for communications applications, and physical layer hardware design for wireless communications systems such as DVB-C, DVBS/S2 and DVB-RCS2.