**DOI: https://doi.org/10.15276/aait.06.2023.6**
**UDC 004.383**

# Decompressor for hardware applications

**Vitalii O. Romankevych[1)]**
ORCID: https://orcid.org/0000-0003-4696-5935; romankev@scs.kpi.ua. Scopus Author ID: 57193263058
**Ivan V. Mozghovyi[1)]**
ORCID: https://orcid.org/0000-0001-5469-486X; mozg.v34@gmail.com.
**Pavlo A. Serhiienko[1)]**
ORCID: https://orcid.org/0000-0003-3030-0074; paulsrgnk002@gmail.com. Scopus Author ID: 57204497516
**Lefteris Zacharioudakis[2)]**
ORCID: https://orcid.org/0000-0002-9658-3073; compusci@cytanet.com.cy.
[1)] National Technical University of Ukraine “Igor Sikorsky Kyiv Politechnic Institute”, 37, Peremogy Ave. Kyiv, 03056, Ukraine
[2)] Neapolis University Pafos, 2, Danais Ave. Paphos, 8042, Cyprus

## ANNOTATION

The use of lossless compression in the application specific computers provides such advantages as minimized amount of memory, increased bandwidth of interfaces, reduced energy consumption, and improved self-testing systems. The article discusses known algorithms of lossless compression with the aim of choosing the most suitable one for implementation in a hardware-software decompressor. Among them, the Lempel-Ziv-Welch (LZW) algorithm makes it possible to perform the associative memory of the decompressor dictionary in the simplest way by using the sequential reading the symbols of the decompressed word. The analysis of the existing hardware implementations of the decompressors showed that the main goal in their development was to increase the bandwidth at the expense of increasing hardware costs and limited functionality. It is proposed to implement the LZW decompressor in a hardware module based on a microprocessor core with a specialized instruction set. For this, a processor core with a stack architecture was selected, which is developed by the authors for the tasks of the file grammar analyzing. Additional memory block for the dictionary storing and an input buffer which converts the byte stream of the packed file into a sequence of unpacked codes are added to it. The processor core instruction set is adjusted to both speed up decompression and reduce hardware costs. The decompressor is described by the Very high-speed integral circuit Hardware Description Language and is implemented in a field programable gate array (FPGA). At a clock frequency of up to two hundred megahertz, the average throughput of the decompressor is more than ten megabytes per second. Because of the hardware and software implementation, an LZW decompressor is developed, which has approximately the same hardware costs as that of the hardware decompressor and has a lower bandwidth at the costs of flexibility, multifunctionality, which is provided by the processor core software. In particular, a decompressor of the Graphic Interchange Format files is implemented on the basis of this device in FPGA for the application of dynamic visualization of patterns on the embedded system display.

**Keywords:** lossless compression; field programable gate array; hardware-software co-design; intellectual property core

***For citation:*** **Romankevych V. O., Mozghovyi I. V., Serhiienko P. A., Zacharioudakis L. “Decompressor for hardware applications”. *Applied Aspects of Information Technology.* 2023; Vol. 6 No.1: 74–83. DOI: https://doi.org/10.15276/aait.06.2023.6**

## INTRODUCTION

It is necessary to solve such contradictional problems as hardware minimization, energy consumption, and performance optimization when developing the embedded digital systems. The compression of information makes it possible to reduce the volume of storage devices in this situation. It can also increase the bandwidth of data transmission channels, but also minimize power consumption.

For this purpose, the lossless data compression application specific hardware should be used [3].

Such minimization is achieved by reducing the intensity of data exchanges, especially between remote transmitters and receivers [1]. For example, the data exchanges between the processor and memory during the execution of programs can consume up to 75 % of the energy consumed due to the wire heating and high-current buffer switching [2]. Therefore, storing data and programs in a compressed form and decompressing them in real time is effective due to reducing the memory volume, speeding up data loading and reducing power consumption.Consider the compression of programs and data which are loaded into the internal memory of the microcontroller of the Internet of Things module. This makes it possible to almost halve the power consumption of this module with insignificant overhead costs for the decompression implementation [4]. Likewise, the compression of the bit stream for the field programable gate array (FPGA) speeds up its loading [5].

If the decompressor is performed as an application specific module, its capabilities are greatly expanded. For example, a data decompressor is now an integral part of ultra-high-resolution displays. Then, it becomes possible to transfer large volumes of data through the display interface [6, 7]. The decompressor of the test data, if it has low hardware costs, significantly increases the efficiency of self-diagnosis systems [8] and diagnostics of complex systems [9] by reducing the volume of stored test data.

In the application specific systems for modeling and management of complex objects, the hardware generators of graphic elements are used, which are displayed on the screen in real time [10, 11]. However, implementing real-time rendering of these elements requires significant computational overhead using a graphics accelerator and often has excessive hidden latency. In addition, the development of programs that generate such images is time-consuming. Therefore, the use of a decompressor in the GIF format, in which the LZW algorithm is implemented, makes it possible to implement such hardware generators of various dynamic graphic elements for display on the screen in real time. These can be, for example, images of measuring devices, such as arrow indicators or patterns of the object position in space [12].

The decompressing files in the LZW standard using hardware-software module is proposed in this article. At the same time, the proper decompression speed is ensured by the implementation of the decompressor in a processor core with a specialized architecture, which is configured in FPGA.

## ALGORITHMS AND TOOLS FOR DECOMPRESSION

The overall goal of data compression is to reduce the number of bits needed to represent information. The lossless compression means that the original data can be accurately reproduced by the decompressor. In this way, it is possible to compress software code, digital input data, medical data, text and other content that is sensitive to distortion.

During Huffman compression, the unique code words are assigned to a bit sequence, the length of which is inversely proportional to the frequency of such lines [13]. A more complex but slower method of arithmetic coding gives a slightly higher compression ratio $\eta$, which is the ratio of file lengths before and after compression. Both methods require statistical analysis of the processed data [14].

Methods based on a dynamically generated dictionary are suitable for data compression without prior analysis. Thus, the LZ77 algorithm uses a dictionary buffer and a preview buffer [15]. The longest line in the preview buffer that matches a line from the dictionary buffer is converted to a code that is the index of the dictionary buffer. However, it is not suitable for hardware implementation because the dictionary buffer and preview buffer sizes are too large for hardware implementation. The LZRW3 algorithm is a variant of the LZ77 algorithm, in which the length of lines in the dictionary is significantly limited, and the search in it is accelerated using a hash table. Thanks to this, it has become widespread in FPGAs and its implementation provides a throughput of up to 180 MB/s at a clock frequency of 220 MHz [16].

The LZ77 algorithm together with Huffman coding is used in the Deflate method and is widely used in hardware decompressors of GZIP files [17, 18].

The LZ78 algorithm creates a dictionary table and finds in it the longest line corresponding to the input line [19]. If there is no string matching the input in the dictionary table, the index of the recognized string, which is one character shorter, and the last character of this string are outputted.

The LZW algorithm is a variant of the LZ78 algorithm, which outputs only the index of the corresponding row of the dictionary table in a compressed file [20]. In addition, the length of the string replaced by the index reaches several hundred bytes, due to which the compression of images with a uniform background is achieved more than ten times. The work [21] shows that the LZW algorithm is not inferior to the Deflate method when compressing images with eight-bit pixels into TIFF files.

According to LZ77, LZ78 algorithms, the decompression complexity is much lower than the compression one due to the fact that the reproduction of the dictionary and the search in it are performed much easier by the decompression. Because of this, the LZW algorithm is common in systems where decompression occurs more often than compression, for example, when decompressing the GIF files. Therefore, this algorithm is chosen for implementation.

There are several implementations of the LZW decompressor in FPGA, among them the decompressor [22] uses fixed-length words and one dictionary table and achieves a maximum decompression speed of 160 MB/s at a clock frequency of 50 MHz. The decompressor [23] has the highest bandwidth of 280 MB/s at a clock frequency of 300 MHz. This speed is achieved due to the parallelization of the processes of creating the dictionary, reading a line from it, and forming the output sequence, as well as an increased number of memory blocks.

Table 1 shows a comparison the decompressor parameters executing the LZ77, Deflate and LZW algorithms when they are implemented in FPGA. Here, hardware costs are expressed in the number of look-up tables (LUTs) which are the main elements of FPGA, as well as built-in two-port memory blocks (BRAMs), which have a volume of 18kB. Such a comparison makes it possible to conclude that the decompressors with the LZW algorithm have a gain in the used hardware volume, and the ratio of hardware volume per unit of bandwidth is more than seven times higher. It also requires a smaller amount of RAM in comparison with decompressors that other algorithms perform.

*Table 1.* **Parameters of some decompressors given in the references**

| Parameter | Reference | | | |
|---|---|---|---|---|
| | [24] | [17] | [18] | [23] |
| Algorithm | LZ77 | Deflate | Deflate | LZW |
| Hardware volume, LUTs | 56000 | 3362 | 15691 | 307 |
| BRAMs | 50 | 16 | 30 | 13 |
| Throughput, MB/s | 7200 | 5.4 | 97.4 | 280 |
| Hardware volume per unit of bandwidth, LUTs/MB/s | 7.8 | 623 | 161 | 1.1 |

***Source*: compiled by the authors**

It is worth mentioning that the LZW algorithm was not widely used among scientists, programmers and engineers for decades due to the fact that it was patented and intellectual property rights prevented its implementation [21]. Moreover, the patent holder Unisys defended its rights to use the LZW algorithm, especially in hardware devices. Now the patent has expired and conditions have appeared to consider this algorithm in more detail and implement it more widely.

In many cases, the extreme decompressor bandwidth is not required, as in the examples in the introduction. But at the same time, the speed of software decompression may not be sufficient or the necessary time resources of the central processor are spent during its execution. Therefore, this article considers the idea of implementing LZW decompression in an application specific compact processor core, the architecture of which is configured for labor-intensive algorithm operations. At the same time, such a processor core, in addition to decompression, is capable of performing other algorithms, for example, decompressing GIF files, implementing data exchange protocols, and organizing system testing.

## PURPOSE AND OBJECTIVES OF THE RESEARCH

**The purpose** of this research is to develop a hardware-software decompressor that performs the lossless LZW algorithm, which, comparing to the hardware decompressor, has the same hardware volume but is able to perform many other algorithms and has the property of reconfiguration.

**To achieve this goal**, the following tasks are performed in this work:

1. Analysis of the LZW decompression algorithm in order to determine the operations that should be performed in hardware and software.
2. Research and modification of the architecture of the microprocessor core configured in FPGA with the correction of its instruction set and the addition of hardware blocks that accelerate decompression.
3. Creating a VHDL model of a hardware-software decompressor, programming its processor core, configuring it into FPGA and determining the parameters of the resulting decompressor module. Comparison of the new decompressor with existing samples.
4. Analysis of the possibilities of a new decompressor implementation.

## LZW DECOMPRESSION

The LZW decompression algorithm is described by the following program text:

```
i = 0;
  while ($y_i \neq 257$){
      if ($y_i = 256$) InitT($C$);
      if($y_i \in C$){
          Out($C(y_i)$); //a string is read from dictionary.
          AddT($C(y_{i-1}) + C1(y_i)$); // a string is
                                //added to dictionary
          $y_{i-1} = y_i$;        //pointer of $y_i$ is stored
          }
          else {
            Out($C(y_{i-1}) + C1(y_{i-1})$);
            AddT($C(y_{i-1}) + C1(y_{i-1})$);
          }
          i++;
    }
}
```

Here, $Y = y_0, y_1, \ldots, y_{m-1}$ is the input sequence of codes from the compressed file,

$X = x_0, x_1, \ldots, x_{n-1}$ is the uncompressed string forming the output file, the characters of which are selected from an alphabet of $k$ characters $\alpha_0, \alpha_1, \ldots, \alpha_{k-1}$,

$C1(y_i)$ is the first character of the string, which is encoded by the code $y_i$,

Out(C($y_i$)) – the function of adding the word C(yi) from the dictionary to the output file,

AddT() is the function of adding a new word to the dictionary,

«+» is the concatenation operation,

InitT($C$) — is the initialization function of the dictionary $C$, which forms the rows of alphabet symbols in the table $C$, the rest of the rows are empty, and the index field of the previous symbol is zero.

As a rule, $k = 256$. For example, if the index AB is 264, then $C$ (264) = AB. Consider the codes 0 – 255 are the character codes of the ASCII table. The code 256 is the command to clear the dictionary, and code 257 is the end-of-compressed code.

Consider an example of a compressed file in the form of the sequence 256, 66, 65, 95, 258, 258, 95, 65, 261, 257. The step-by-step process of unpacking this sequence is shown in Table 2.

*Table 2.* **Eexample of decompressing a file using the LZW algorithm**

| $y_i$ | $y_{i\text{-}1}$ | Dictio-nary reading | Dictionary writing | Decompressor output |
|---|---|---|---|---|
| 256 | – | | Clearing | |
| 66 | 256 | B | | B |
| 65 | 66 | A | 258->BA | BA |
| 95 | 65 | _ | 259->A_ | BA_ |
| 258 | 95 | BA | 260->_B | BA_BA |
| 258 | 258 | BA | 261->BAB | BA_BABA |
| 95 | 258 | _ | 262->BA_ | BA_BABA_ |
| 65 | 95 | A | 263->_A | BA_BABA_A |
| 261 | 65 | BAB | 264->AB | BA_BABA_ ABAB |
| 257 | 261 | (eof) | End | BA_BABA_ ABAB(eof) |

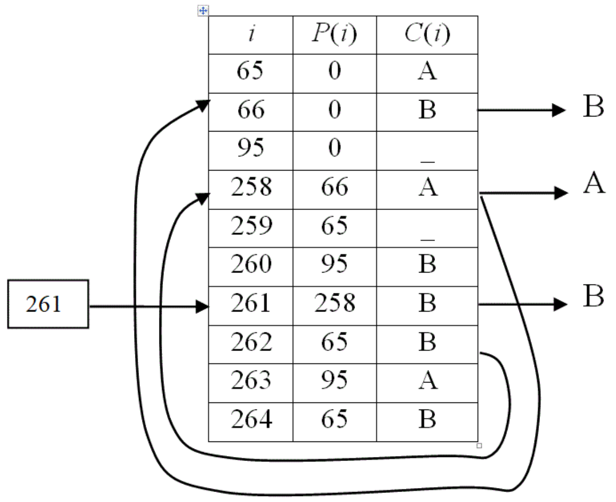
***Source*: compiled by the authors**

The length of strings in a $C$ dictionary is variable and can reach hundreds of characters. Therefore, it is impractical to implement the dictionary in a hardware decompressor in the form of a random-access memory (RAM) device, the word of which has a length of this order. Because of this, in all known hardware decompressors, this dictionary is implemented as a list.

The Fig. 1 illustrates the contents of the dictionary RAM, which has an address-index $i$, the field $P(i)$ of the pointer to the previous character of the line and the field $C(i)$ of the next character of the string.

According to the example in the Table 1, consider the code-index 261 is inputted to the dictionary input. Then the last symbol of line B and the index of the previous symbol 258 are selected from RAM, after that the symbol A of the string and the index of the first symbol B are selected. The first symbol B is selected last. At the same time, the formation of the next word AB in the dictionary consists in writing in the table at index 264 the symbol B and the index of symbol A, which is the input code $y_{i-1}$ that is delayed for one period of the algorithm execution.

To increase the compression ratio, the $y_i$ codes have different bit widths that change dynamically. Initially, it is equal to 9, and the next code after code 511 already has the bit width 10. In practice, the bit width of the code does not exceed 12. Therefore, after the code 4095, the code of the dictionary clearing command comes.



| $i$ | $P(i)$ | $C(i)$ |
|---|---|---|
| 65 | 0 | A |
| 66 | 0 | B |
| 95 | 0 | _ |
| 258 | 66 | A |
| 259 | 65 | _ |
| 260 | 95 | B |
| 261 | 258 | B |
| 262 | 65 | B |
| 263 | 95 | A |
| 264 | 65 | B |

*Fig. 1.* **Contents of the dictionary and reading from it the word by the index $y_i$**
***Source*: compiled by the authors**

So, when the file $Y$ is unpacked, the codes $y_i$ are extracted from it. According to this code, the word is read from the dictionary $C$, which is joined to the string of the resulting file $X$. Moreover, the read word has the reverse order of symbols, which is corrected either by writing-reading from the stack memory, or by writing to the output buffer with index addressing [23].

The compression ratio of the LZW algorithm depends on the quality of the compressor and the content of the compressed file. In real applications, it is equal to η = 1.6-38 and meets the practical needs in many cases [25].

## MODULE FOR LZW DECOMPRESSION

The module for LZW decompression is based on the SM16 processor core, described in [26]. The structure of the 16-bit processor core SM16 as part of the decompressor is shown in Fig. 2. This processor core has a well-known dual-stack architecture [27]. The processor core contains a program counter (PC), data RAM DataRAM, program ROM ProgramROM, instruction register (IR), return address stack (RStack), data stack (DStack), arithmetic and logic unit ALU. Registers T, N are the outermost registers of the DStack. The R register is the top of the RStack, which also acts as a loop counter. Registers A and B serve as address pointers to speed up data transfer.

The input file *Y* is fed into the FIFO buffer IBuf. It has a built-in scheme for the selection of individual codes $y_i$ from the sequence of bytes of the file *Y*, which is controlled by an automaton. It has interrupt signal outputs for events of filling the buffer and detecting the the dictionary cleaning and end-of-file commands from it.

The DICT dictionary stores up to 4095 *P*(*i*) indices and *C*(*i*) symbols and is implemented as RAM. The DataRAM data memory, in addition to storing variables, is used to change the order of characters in a word that is read from the dictionary before writing it to the OBuf output buffer. That is, with the help of the pointer register A, a stack is organized in it. Its depth is determined by the maximum length of the string encoded in the *Y* file.

The instructions of the processor core are executed in one cycle of the clock signal, with the exception of branching, constant loading, and memory reading instructions, which are executed in two cycles. Due to the fact that the processor core has stack architecture, the routine call is executed in only two clock cycles. At the same time, the context switching is performed naturally through the RStack and DStack. The interruption from the input buffer filling signals and the arrival of the dictionary cleaning command is also implemented quickly. These properties make the architecture friendly to the bitstream parsing and encoding algorithms that often use branching.

A 16-bit instruction has one to three opcode fields, F1, F2, F3, and a variable-length D field that stores a constant or jump displacements. The processor core can perform up to three operations F1, F2, F3 in parallel. Several instructions have been added to the processor core instruction set to speed up the dictionary access and data exchanges.

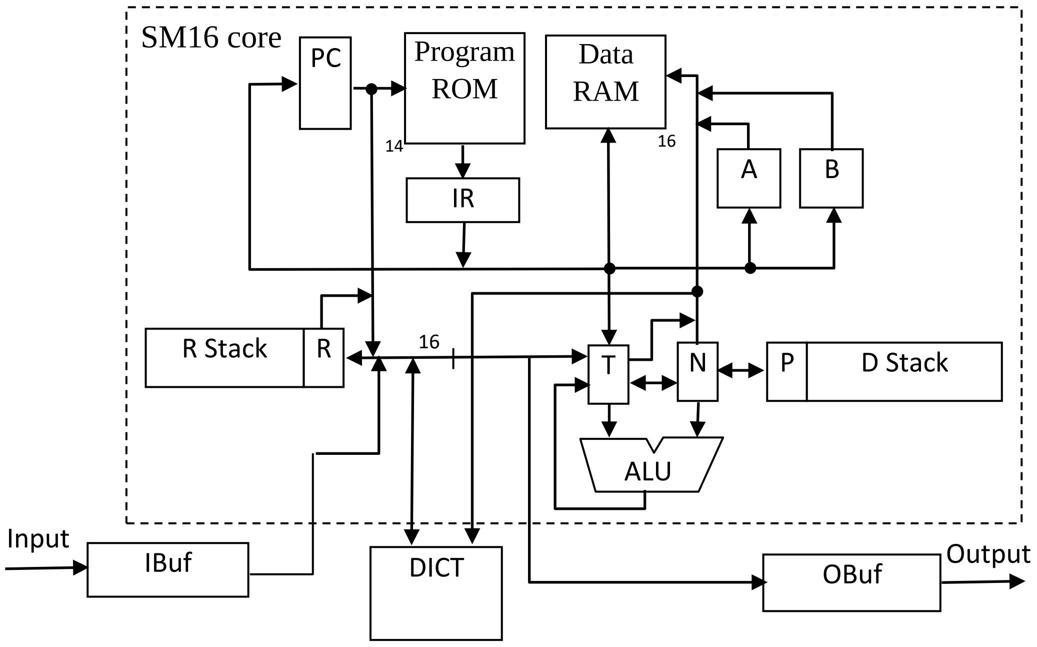


*Fig.2.* **SM16 processor core included in the LZW decompressor**
***Source*: compiled by the authors**

Due to the following additional instruction

:L1 OUTB @ab- L1 DJNZ

the symbol from the register T is written into the output buffer OBuf, and the next symbol from DataRAM at the address of the index register A is written in the register T. After that, the register A is decremented. If the state of the register R is non-zero, then the control flow is transferred to the label L1, and the register R is decremented. Otherwise, the loop is exited. Thus, the decompressed word is rewritten from the stack organized in DataRAM to the output buffer using a single instruction executed in a program cycle.

It should be noted that the syntax of the assembly language of the SM16 processor core is largely the same as the syntax of the Forth language. Just like the Forth programs, the compiled programs for SM16 processor core take up little memory compared to programs in other languages [26]. The user of the SM16 core can add new instructions to the instruction set and specific hardware if they help to speed up the execution of the algorithm. The program written in assembly language is compiled and simulated in the developed framework containing the simulator with a graphical interface.

## RESULTS

The SM16 processor core is described in VHDL language. When placed in a Xilinx Spartan-6 FPGA, it occupies only 632 LUTs and has a maximum clock frequency of 95 MHz. For comparison, the hardware volume of the 16-bit OpenMSP430 core [28] is three times higher, and its clock frequency is lower than in the proposed core.

The selection of $y_i$ codes from the input stream, as well as the fixing the events of dictionary cleaning, end of the file, or the code length change are performed in the hardware of the IBuf buffer. The DICT dictionary RAM has direct access by the index *i*. The OBuf output buffer is hardware-implemented FIFO. The rest of the algorithm is performed in the program.

Thanks to the implementation of the specified hardware blocks, it was possible to halve the cycle of unpacking the $y_i$ code by throwing out the instructions from the program that extract the $y_i$ code from the input stream, monitor the arrival of the dictionary cleaning command, or the end-of-file symbol, form the index address of the dictionary.

The obtained parameters of the decompressor module are given in Table 3. The hardware costs in it are expressed in the number of configurable logic block slices (CLBs), each of which includes from 1 to 4 LUTs, depending on the specific placement in FPGA. It should be noted that, on average, there are 160 CLBs per memory block BRAM. So these blocks are a valuable hardware resource. The quality factor *K* is also presented there, which is equal to the product of hardware costs and the amount of memory. That is, the lower the *K* factor, the more effective the project is. The decompressor executing the LZW algorithm spends, on average, 15 cycles to decompress one character. At a processor core clock frequency of 190 MHz, decompression is carried out at a speed of 12.6 Mbytes/s.

For comparison, a decompressor model was tested in which the LZW algorithm is executed only by programming the SM16 processor core in FPGA. In order to obtain an acceptable decompression speed during the algorithm software execution, it was necessary to add the shift left and shift right instructions to the instruction set. It is necessary for extracting the variable bit length codes from the input byte stream. As a rule, such instructions are included in the instruction set of most known processor architectures. Such instructions require the additional barrel shifter circuit, which takes a lot of hardware. As a result, the hardware costs of the processor core increased to 239 CLBs, that is, by 24% compared to the hardware and software implementation. Also, the decompression throughput is halved. So, this example testifies in favor of the hardware and software implementation of the LZW algorithm.

Compared to hardware decompressors, this decompressor loses in throughput. However, it has half the amount of BRAM blocks than the device [23] and is able to decompress files with longer lines $x_i$ than other analogs, i.e., it has a potentially larger value of the compression ratio $\eta$.

The hardware decompressor cores are listed in Table 3 for comparison. They are designed according to the technology of the register transfer level description which is compiled into a circuit at the gate level by the proper FPGA compiler-synthesizer. Therefore, for their modernization, they need to perform a repeated design cycle, which is time-consuming and associated with the addition of equipment volumes that are worth the added functionality. For example, in [18] a decompressor with a static Huffman table is proposed, in which the hardware costs double when adding a dynamic Huffman table. Unlike the hardware modules, in the proposed decompressor module it is easy to add functionality without changing the structure of the module. So, the advantage of this decompressor is the possibility of reconfiguration, which consists in

*Table 3.* **Decompressors cores parameters**

| Decompressor core | Helion [16] | Source [18] | Source [23] | Proposed, no IBuf, DICT | Proposed | Proposed |
|---|---|---|---|---|---|---|
| Algorithm | LZRW3 | Modified LZW | LZW | LZW | LZW | LZW |
| Xilinx FPGA chip | Virtex-5 | Virtex-2 | Virtex-7 | Spartan-6 | Spartan-6 | Kintex-7 |
| Hardware costs, CLBs | 166 | 247 | 139 | 239 | 193 | 224 |
| RAM, BRAMs | 4 | 8 | 13 | 7 | 7 | 7 |
| RAM size, kB | 9 | 18 | 29.25 | 15.75 | 15.75 | 15.75 |
| Clock frequency, MHz | 226 | 50 | 300 | 95 | 95 | 190 |
| η, MBytes/s | 180 | 140 | 280 | 3.1 | 6.3 | 12.6 |
| Qualitative index K | 1494 | 4446 | 4066 | 3764 | 3040 | 3528 |
| Introduction of additional functions | Unavailable | Unavailable | Unavailable | Available | Available | Available |

***Source*: compiled by the authors**

programming the processor core to perform many other algorithms, such as unpacking GIF and TIFF files, data exchange protocols, system testing, control algorithms. At the same time, a slight increase in the volume of program memory is possible, which is already small compared to the volume of similar programs for RISC processors [27]. To increase the functionality, a developed framework is used with a simulator of a microprocessor core together with added hardware units, which has a built-in assembler [26].

The multifunctionality of the module is confirmed by the fact that it is easy to combine this module with the device described in [26]. Both devices have the same processor core. Therefore, in order for this module to be able to perform both file unpacking and grammatical analysis of its content, it is only necessary to add three blocks of stack memory to it, which occupy 45 LUTs each, and increase the amount of program memory.

To achieve an even higher speed of decompression, it is possible to create a multi-processor system based on a set of the configurable SM16 processor cores, which decompress independent data blocks in parallel.

## CONCLUSIONS

Hardware modules for the lossless data decompression make it possible to reduce the amount of data stored or transmitted over communication channels, as well as to reduce the power consumption of devices for embedded applications. Among many lossless compression algorithms, the LZW algorithm is the most suitable for hardware implementation due to low hardware costs for its implementation with an acceptable compression ratio. A hardware and software module for LZW decompression has been developed, which can be configured in FPGAs of various series. The module is built on the basis of a processor core with stack architecture.

Thanks to the hardware and software implementation, a decompressor module is designed, which, with a hardware cost of 193 CLBs in Xilinx FPGA, has a decompression speed of 12.6 MB/s and, unlike hardware decompressors, has the ability to be reconfigured and increase the number of algorithms performed with no or small additional hardware costs. The project quality factor *K* is low, and the project tends to be able to its functionality is proven. So, the goal of the research has been achieved. Specifically, the module is configured to decompress the GIF files. The module is intended for use in embedded systems. The throughput of decompression increases proportionally to the number of such modules that work in parallel. Thus, the proposed decompression module can be useful when it is implemented in many embedded systems implemented in FPGAs.

## REFERENCES

1. Ritter, D., Dann, J., May, N. & Rinderle-Ma, S. “Hardware accelerated application integration processing: Industry Paper”. *DEBS '17: Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems.* June 2017. p. 215–226, https://www.scopus.com/record/display.uri?eid=2-s2.0-85023209536&origin=resultslist&sort=plf-f. DOI: https://doi.org/10.1145/3093742.3093911.

2. Lafond, S. & Lilius, J. "An energy consumption model for java virtual machine". *TUCS Technical Report*. 2004; No. 597. DOI: https://doi.org/10.1007/11682127_22.

3. Li, X., Mu, L., Zang, Y. & Qin, Q. "Study on performance degradation and failure analysis of machine gun barrel". *Defence Technology*. 2020; 16. (2): 362–373, https://www.scopus.com/record/display.uri?eid=2-s2.0-85069836705&origin=resultslist&sort=plf-f. DOI: https://doi.org/10.1016/j.dt.2019.05.008.

4. Zervas, N. "Firmware compression for lower energy and faster boot in IoT devices". *October 2015*. – Available from: https://www.design-reuse.com/articles/38541/firmware-compression-for-lower-energy-and-faster-boot-in-iot-devices.html. – [Accessed: Jan., 2022].

5. Beckhoff, C., Koch, D. & Torresen, J. "Portable module relocation and bitstream compression for Xilinx FPGAs". *24th International Conference on Field Programmable Logic and Applications (FPL)*. Munich: Germany. 2014. p. 1–8. DOI: https://doi.org/10.1109/FPL.2014.6927480, https://www.scopus.com/record/display.uri?eid=2-s2.0-84911191271&origin=resultslist&sort=plf-f.

6. Walls, F. G. & MacInnis, A. S., "VESA display stream compression for television and cinema applications". *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*. 2016; 6(4): 460–470, https://www.scopus.com/record/display.uri?eid=2-s2.0-85027032558&origin=resultslist&sort=plf-f. DOI: https://doi.org/10.1109/JETCAS.2016.2602009.

7. "Chips&Media releases CFrame30, its groundbreaking hardware design for loss frame buffer compression". Seoul: Korea. 2015. – Available from: https://www.design-reuse.com/news/37671/chips-media-lossy-frame-buffer-compression.html. – [Accessed: Jan., 2020].

8. Touba, N. A. "Survey of test vector compression techniques". *IEEE Design & Test of Computers*. 2006; 23 (4): 294–303, https://www.scopus.com/record/display.uri?eid=2-s2.0-33748510387&origin=resultslist&sort=plf-f. DOI: https://doi.org/10.1109/MDT.2006.105.

9. Romankevitch, A., Morozov, K., Mykytenko, S. & Kovalenko O. "On the cascade GL-model and its properties". *Applied Aspects of Information Technology*. 2022; 5 (3): 256–271. DOI: https://doi.org/10.15276/aait.05.2022.18

10. Ponce-Cruz, C. & Ramirez-Figueroa, F. D. "Intelligent control systems with LabVIEW". *Springer*. 2010. DOI: https://doi.org/10.1007/978-1-84882-684-7.

11. Kovačec, D. "FPGA IP cores for displays". *In: Handbook of Visual Display Technology*. J. Chen, W. Cranton, M. Fihn-Eds. *Springer*. 2012. p. 512–530, https://www.scopus.com/record/display.uri?eid=2-s2.0-85027007071&origin=resultslist&sort=plf-f. DOI: https://doi.org/10.1007/978-3-540-79567-4_40.

12. Mozghovyi, I., Sergiyenko, A. & Yershov, R. "GIF image hardware compressors". *Information, Computing and Intelligent systems*. 2021; 2: 48–55. DOI: https://doi.org/10.20535/2708-4930.2.2021.244189.

13. Gallager, R. "Variations on a theme by Huffman". *IEEE Transactions on Information Theory*. 1978; 24. (6): 668–674, https://www.scopus.com/record/display.uri?eid=2-s2.0-0018032133&origin=resultslist&sort=plf-f. DOI: https://doi.org/10.1109/TIT.1978.1055959.

14. Salomon, D. & Motta, G. "Handbook of data compression". *5th Ed. Springer*, 2010. 1360 p. ISBN: 978-1-84882-903-9, https://www.scopus.com/record/display.uri?eid=2-s2.0-84865192560&origin=resultslist&sort=plf-f. DOI: https://doi.org/10.1007/978-1-84882-903-9.

15. Ziv, J. & Lempel, A. "A universal algorithm for sequential data compression". *IEEE Transactions on Information Theory*. 1977; 23 (3): 337–343, https://www.scopus.com/record/display.uri?eid=2-s2.0-0017493286&origin=resultslist&sort=plf-f. DOI: https://doi.org/10.1109/TIT.1977.1055714.

16. "LZRW3 data compression core for Xilinx FPGA. Full Datasheet". *Helion Technology*. 2008. p. 1–3. – Available from: https://www.heliontech.com/downloads/lzrw3_xilinx_datasheet.pdf. – [Accessed: Jan. 2020].

17. Hwang, G. B., Cho, K. N., Han, C. Y., Oh, H. W., Yoon, Y, H. & Lee S. E. "Lossless decompression accelerator for embedded processor with GUI". *Micromachines,* 2021; 12 (2), https://www.scopus.com/record/display.uri?eid=2-s2.0-85100608354&origin=resultslist&sort=plf-f. DOI: https://doi.org/10.3390/mi12020145.

18. Ledwon, M., Cockburn, B. F. & Han, J. "High-Throughput FPGA-Based hardware accelerators for deflate compression and decompression using high-level synthesis". *IEEE Access*. 2020; 8: 62207–62217, https://www.scopus.com/record/display.uri?eid=2-s2.0-85083429723&origin=resultslist& sort=plf-f. DOI: https://doi.org/10.1109/ACCESS.2020.2984191.

19. Ziv, J. & Lempel, A. "Compression of individual sequences via variable-rate coding". *IEEE Transactions on Information Theory*. 1978; 24 (5): 530–536, https://www.scopus.com/record/display.uri?eid=2-s2.0-0018019231&origin=resultslist&sort=plf-f. DOI: https://doi.org/10.1109/TIT.1978.1055934.

20. Welch, T. "A technique for high-performance data compression". *Computer*. 1984; 17 (6): 8–19. https://www.scopus.com/record/display.uri?eid=2-s2.0-0021439618&origin=resultslist&sort=plf-f. DOI: https://doi.org/10.1109/MC.1984.1659158.

21. May, P. & Davies K. "Practical analysis of tiff file size reductions achievable through compression". *iPRES 2016: 13th International Conference on Digital Preservation*. Bern: Switzerland. 2016. p. 1–10.

22. Navqi, S., Naqvi, R., Riaz, R. A. & Siddiqui F. "Optimized RTL design and implementation of LZW algorithm for high bandwidth applications". *Przeglad Electrotechniczny* (*Electrical Review*). 2011; 4: 279–285, https://www.scopus.com/record/display.uri?eid=2-s2.0-79955025506&origin=resultslist&sort=plf-f.

23. Zhou, X., Ito, Y. & Nakano, K. "An efficient implementation of LZW decompression in the FPGA". *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. Chicago: IL, USA. 2016. p. 599–607, https://www.scopus.com/record/display.uri?eid=2-s2.0-84991665925&origin=resultslist&sort=plf-f. DOI: https://doi.org/10.1109/IPDPSW.2016.33.

24. Fang, J., Chen, J., Lee, J. et al. "An Efficient High-Throughput LZ77-Based Decompressor in Reconfigurable Logic". *J. Sign. Process. Syst*. 2020; 92: 931–947, https://www.scopus.com/record/display.uri?eid=2-s2.0-85085762888&origin=resultslist&sort=plf-f. DOI: https://doi.org/10.1007/s11265-020-01547-w.

25. Funasaka, S., Nakano, K. & Ito, Y. "A Parallel algorithm for LZW decompression, with GPU Implementation". *In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K., Kitowski, J., Wiatr, K. (eds) Parallel Processing and Applied Mathematics. PPAM 2015.* LNCS. *Springer, Cham*. 2016; Vol. 9573: 228–237, https://www.scopus.com/record/display.uri?eid=2-s2.0-84968531501&origin=resultslist&sort=plf-f. DOI: https://doi.org/10.1007/978-3-319-32149-3_22.

26. Sergiyenko, A., Orlova, M. & Molchanov, O. "Hardware/Software Co-design for XML-Document Processing". *In: Hu, Z., Petoukhov, S., Dychka, I., He, M. (eds) Advances in Computer Science for Engineering and Education III. ICCSEEA 2020. Advances in Intelligent Systems and Computing, Springer, Cham*. 2021; Vol 1247: 373–383, https://www.scopus.com/record/display.uri?eid=2-s2.0-85089717777&origin=resultslist&sort=plf-f. DOI: https://doi.org/10.1007/978-3-030-55506-1_34.

27. Koopman, P. "Stack computers: the new wave". *Ellis Horwood, Mountain View Press*, CA. 1989.

28. Oliver, J. P., Acle, J. P. & Boemo, E. "Power estimations vs. power measurements in Spartan-6 devices". *2014 IX Southern Conference on Programmable Logic (SPL)*. Buenos Aires: Argentina. 2014. p. 1–5, https://www.scopus.com/record/display.uri?eid=2-s2.0-84922109257&origin=resultslist&sort=plf-f. DOI: https://doi.org/10.1109/SPL.2014.7002214.

**Conflicts of Interest:** the authors declares no conflict of interest

Received 16.01.2023
Received after revision 12.03.2023
Accepted 17.03.2023

**DOI: https://doi.org/10.15276/aait.06.2023.6**
**УДК 004.383**

# Декомпресор для апаратних застосунків

**Романкевич Віталій Олексійович**[1)]
ORCID: https://orcid.org/0000-0003-4696-5935; romankev@scs.kpi.ua. Scopus Author ID: 57193263058
**Мозговий Іван Владиславович**[1)]
ORCID: https://orcid.org/0000-0001-5469-486X; mozg.v34@gmail.com
**Сергієнко Павло Анатолійович**[1)]
ORCID: https://orcid.org/0000-0003-3030-0074; paulsrgnk002@gmail.com. Scopus Author ID: 57204497516

**Lefteris Zacharioudakis**[2)]
ORCID: https://orcid.org/0000-0002-9658-3073; compusci@cyt anet.com.cy
[1)] **Національний Технічний Університет України “КПІ ім. Ігоря Сікорського”, пр. Перемоги, 37. Київ, 03056, Україна**
[2)] **Неапольский Університет у Пафосі, пр. Данайський, 2. Пафос, 8042, Кіпр**

## АНОТАЦІЯ

Застосування безвтратної компресії в спеціалізованих обчислювальних засобах дає такі переваги, як мінімізація об’єму пам’яті, збільшення пропускної здатності інтерфейсів, зменшення енергоспоживання, покращення систем автотестування. В статті розглянуті відомі алгоритми безвтратної компресії з метою вибору такого, що найбільш підходить для реалізації у апаратно-програмному декомпресорі. Серед них алгоритм Lempel-Ziv-Welch (LZW) дає змогу найпростішим чином виконати асоціативну пам’ять словника декомпресора за рахунок послідовного зчитування символів слова. Аналіз існуючих апаратних реалізацій декомпресорів показав, що при їх розробці основна мета була збільшити пропускну здатність за рахунок збільшення апаратних витрат та обмеження функціональності. Запропоновано виконати декомпресор LZW апаратно-програмним чином на основі ядра мікропроцесора зі спеціалізованою системою команд. Для цього вибрано процесорне ядро зі стековою архітектурою, розроблене авторами для задач граматичного аналізу. Додано блок пам’яті для зберігання словника та вхідний буфер, який конвертує потік байтів запакованого файлу у послідовність розпакованих кодів, що додані до нього. Система команд процесорного ядра скоректована з метою як пришвидшення декомпресії, так і зменшення апаратних витрат. Декомпресор описаний мовою Very high-speed integral circuit Hardware Description Language і реалізований у програмовній логічній інтегральній схемі. При тактовій частоті двісті мегагерц, середня пропускна здатність декомпресора – понад десять мегабайтів на секунду. Завдяки апаратно-програмній реалізації, одержано LZW-декомпресор, який має при приблизно тих самих апаратних витратах як у апаратного декомпресора меншу пропускну здатність за рахунок гнучкості, багатофункціональності, які дає програмовне процесорне ядро в його складі. Зокрема, на основі даного пристрою реалізується декомпресор Graphic Interchange Format файлів для застосунку динамічної візуалізації патернів на дисплеї вбудованої системи.

**Ключові слова:** безвтратна компресія; програмовна логічна інтегральна схема; апаратно-програмна розробка; віртуальний модуль

## ABOUT THE AUTHORS

**Vitalii O. Romankevych** - Doctor of Engineering Sciences, Professor, Professor of System Programming and Special Computer System Department. National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”, 37, Peremogy Av. Kyiv, 03056, Ukraine
ORCID: http://orcid.org/0000-0003-4696-5935; romankev@scs.kpi.ua. Scopus Author ID: 57193263058
***Research field*:** Dependability of Fault-Tolerant Multiprocessor Control Systems. Self-Testing of Multiprocessor Systems

**Романкевич Віталій Олексійович -** доктор технічних наук, професор, професор кафедри Системного програмування та спеціальних комп’ютерних систем. Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», пр. Перемоги, 37. Київ, 03056, Україна

**Ivan V. Mozghovyi** - PhD student of Department of Computer Engineering. National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”, 37, Peremogy Av. Kyiv, 03056, Ukraine
ORCID: https://orcid.org/0000-0001-5469-486X; mozg.v34@gmail.com
***Research field*:** Pattern recognition in images; embedded high-performance manycore systems in FPGA

**Мозговий Іван Владиславович -** аспірант кафедри Обчислювальної Техніки. Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», пр. Перемоги, 37.
Київ, 03056, Україна

**Pavlo A. Serhiienko** - PhD student, Assistant of Department of System Programming and Specialized Computer Systems. National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”, 37, Peremogy Av. Kyiv, 03056, Ukraine
ORCID: http://orcid.org/0000-0003-3030-0074; paulsrgnk002@gmail.com. Scopus Author ID: 57204497516
**Research field:** Pattern recognition in images; embedded high-performance manycore systems in FPGA

**Сергієнко Павло Анатолійович –** аспірант, асистент кафедри Системного програмування та спеціальних комп’ютерних систем. Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», пр. Перемоги, 37. Київ, 03056, Україна

**Lefteris Zacharioudakis** - PhD, Visiting Lecturer of Neapolis University Pafos, 2, Danais Avenue, Paphos, 8042, Cyprus
ORCID: https://orcid.org/0000-0002-9658-3073; compusci@cytanet.com.cy.
**Research field:** Computer/network security and has a number of publications in cryptography and authentication/identification methods

**Лефтеріс Захаріудакіс –** PhD, викладач у Неапольскому Університеті у Пафосі, пр. Данайський, 2.
Пафос, 8042, Кіпр