I.J. Modern Education and Computer Science, **2021, 6, 11-24**
Published Online December 2021 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2021.06.02

Comparative Analysis of Performance Run Length (RLE) Data Compression Design by VHDL and Design by Microcontroller

Marvin Chandra Wijaya
Department of Computer Engineering, Maranatha Christian University, Bandung, Indonesia
Email: marvin.cw@eng.maranatha.edu

Received: 26 June 2020; Revised: 15 July 2020; Accepted: 26 August 2020; Published: 08 December 2021

Abstract: Compression is a way to compress data to produce a file with a size smaller than its original size. Compression techniques can be performed on text data or binary, image (JPEG, PNG, ...), Audio (MP3, AAC, RMA, WMA, ...) and video (MPEG, H261, H263, ...). Compression Data is a way to process information using bits or other information units lower than the representation of data that is not encoded with a particular encoding system.

Data compression has a function to condense, shrink data to its size becomes smaller. With the smaller size of storage space required then less to make it a more efficient storing process, but it also can shorten the time of the data exchange.

Data compression using the run-length encoding (RLE) is a technique used to compress the data contains recurring characters. Run-length encoding (RLE) is a very simple form of data. In RLE running data (sequence data value is the same with many of the data elements in a row) is stored as the value of a single data and calculated the length of the data. This method is useful for data that contains a lot of data, such as simple graphic images (icons, line drawings, and animation). Data compression can be realized in various ways. Data compression can be designed using the VHDL language and can also use a microcontroller. Every realization of data compression has different performances. In this research, the performance was analyzed at the speed of compression. From the experiments conducted, the results of compression speed using VHDL implementation are 6.95 KB / s and microcontroller implementation is 5.34 KB/s. Based on the experimental results from the implementation of data compression using VHDL proposed in this study has a speed of 30.11% better.

Index Terms: Data Compression, Run Length Encoder, FPGA, VHDL, Microcontroller

1. Introduction

The need for large storage areas has increased greatly lately. This requirement, caused by data that must be stored more and more, especially for the banking and business data. They seem to desperately need a very large capacity to store all your data and important files. The storage is not only allocated to one place. But they will also store data or files in another place. Although saved data is the same, it is useful to backup data. The backup needs to be done because there is no means of ensuring that the data on the storage area in the computer will not be damaged. Then, many storage capacities must be provided to accommodate all of those things [1].

There are a lot of theories and methods for data compression. One theory is simple enough to use the Huffman code (Huffman coding). In the Huffman code encoding, used the concept of a binary tree data structure (binary tree). Theory Huffman code itself is not just one, but there are some variations, optimization, and combinations thereof. In computer science, data compression is the art of representing digital data sources and the other into a more compact form [2]. One variance of Huffman code is dynamic Huffman and Run Length Encoding to get the best possible compression ratio and compression time [3].

Digital data that has been compressed can be restored to its original form of digital data (decompression) where this depends on the application software that supports the compression [4]. When an application is able to 'eliminate' or compress the data that is not required then the application is also able to restore the data that is eliminated so that a digital data originally (decompression), but there is also an application that can compress but when decompressing can use other applications (for example applications WinZip with WinRAR application).

The method used in this algorithm is to find characters that are repeated more than 3 times in a file for later converted into a bit marker (marker bit) and then followed by a bit that provides information on the number of characters that are repeated. After that it closes with compressed characters. The meaning of the bit marker is a row of 8 bits that make up an ASCII character. So that if a file contains a recurring character, for example, or in binary 01000001

eight times, then the data is compressed into 11,111,110 00001000 01000001. Thus we can save as much as 5 bytes. Run Length algorithm can be expressed as follows: Rows of Data the left is a row of data in the original file, while a row of data to the right is a row of result data compression algorithms Run Length.

The purpose of this research was to produce a comparative analysis of performance data compression method RLE implementation using VHDL and using a microcontroller. The objective of this research is to find the best form of implementation in terms of speed for data compression using the Run Length Encoder method. The solution for implementing hardware to create a compression encoder is two ways, namely with general-purpose computers and special-purpose computers. In general-purpose, computers can use a microprocessor or microcontroller programmed with the Run Length Encoder compression algorithm. On special-purpose computers can be made using FPGA with the VHDL programming language. The VHDL program will be created using the Run Length Encoder compression algorithm. Each of these solutions has its advantages and disadvantages. In solutions using a microcontroller have advantages in terms of modification of the program if there are various possible inputs. While the disadvantage is that the speed depends on the microprocessor clock and is more energy-consuming. Solutions using FPGA programmed using VHDL are usually not energy-intensive and have high processing speeds because they are directly applied to the logic circuit. The disadvantage is that flexibility is still lacking. The author's expectation in this research is to improve compression speed, especially solutions using FPGA programmed using VHDL.

2. Related Works

There are some works related to this research. A power-efficient System-on-a-Chip test data compression method using alternating statistical run-length coding is designed by Haiying Yuan [5]. The experimental results show a high compression ratio, low scan-in test power dissipation with System-on-a-Chip scan testing.

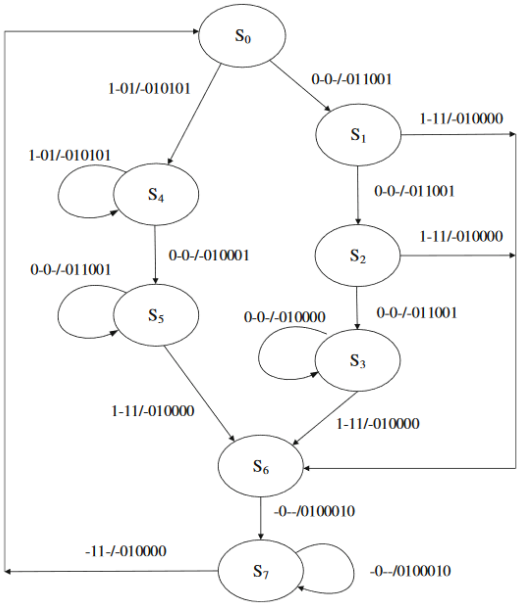

Fig.1. Finite State Diagram (Haiying Yuan)

Haiying Yuan uses eight states (S0, S1, S2, S3, S4, S5, S6, S7) in making on-chip system programs (figure 1). The number of states used makes the program more efficient and faster, but flexibility is not too large. The author proposes a system with a large number of more states to make the program more flexible without reducing speed too much.

Makinen conducted research entitled "Approximate Matching of Run-Length Compressed Strings" which modified the algorithm for Run Length Encoder [6]. A Greedy algorithm for the longest common subsequence (LCS) to the Levenshtein distance achieving O(m'n+n'm) complexity extends to a weighted edit distance model for matching of strings that have been compressed using run-length encoding. This algorithm is useful for the existence of the longest common subsequence between the two strings. Other methods are still needed for normal circumstances. The worst-case of complexity to compute the edit distance between two run-length encoded strings depends on the uncompressed string lengths, it can cause a long time processing. The solution is to break the foundational gap "a fully compressed" algorithm whose running time depends solely on the compressed string lengths[7].

Also, there is an analysis for time processing with a set of data compression techniques that are used for on-chip VHDL programming by EL-Maleh [8]. The techniques used are based on data compression and decompression code. The decompression and compression algorithm decreases test data volume and the amount of data that has to be transported from the source to the chip.

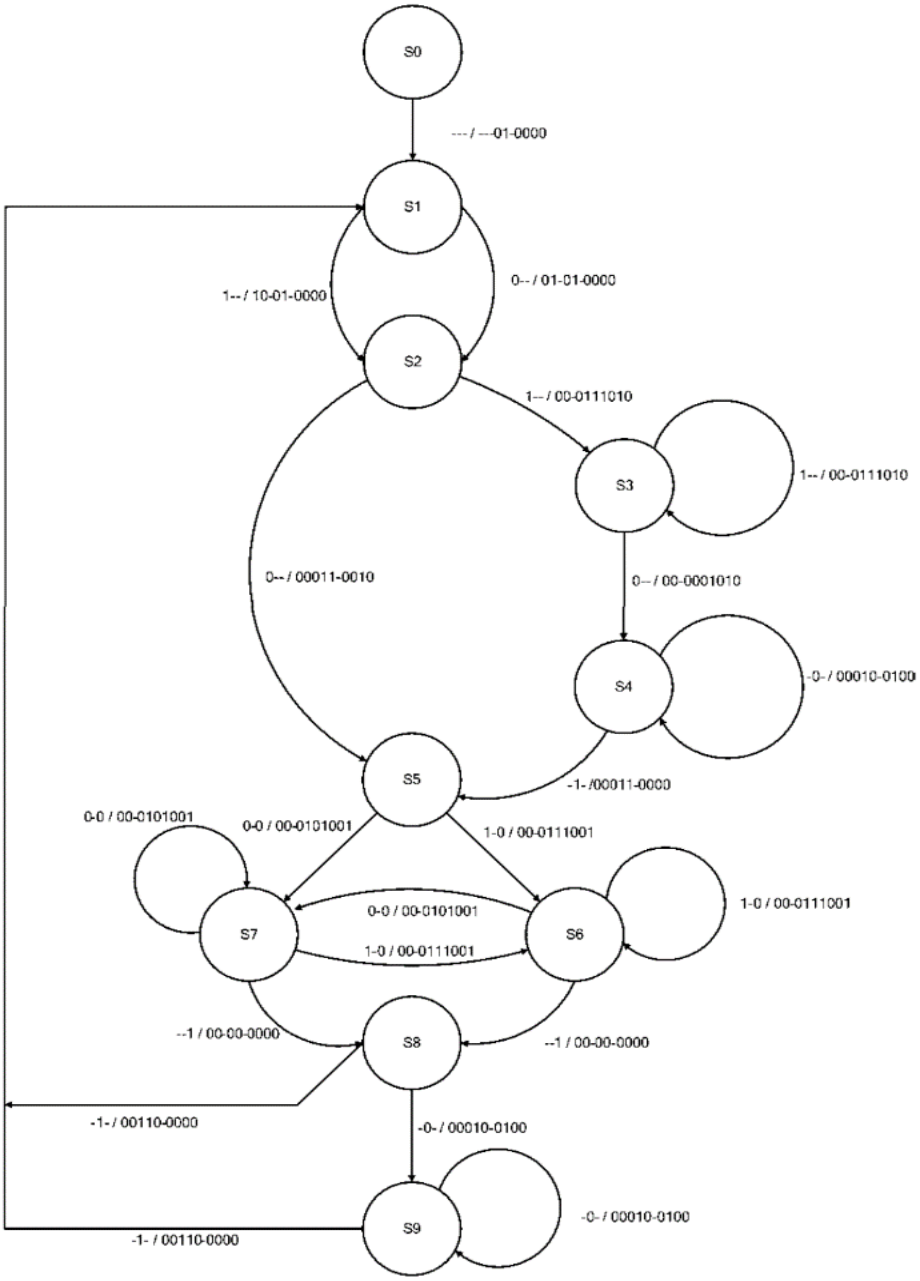

Fig.2. Finite State Diagram (El-Maleh)

El-Maleh uses ten states (S0, S1, S2, S3, S4, S5, S6, S7, S8, S9) in making on-chip system programs (figure 2). The use of these ten states has also resulted in the data being compressed correctly.

Wang Dauh Tseng researched multiple and large test volumes on a chip design, the test in the test set is divided into several sets. Each set is constituted by several compatible patterns in which information such as length and number of patterns is encoded [9].

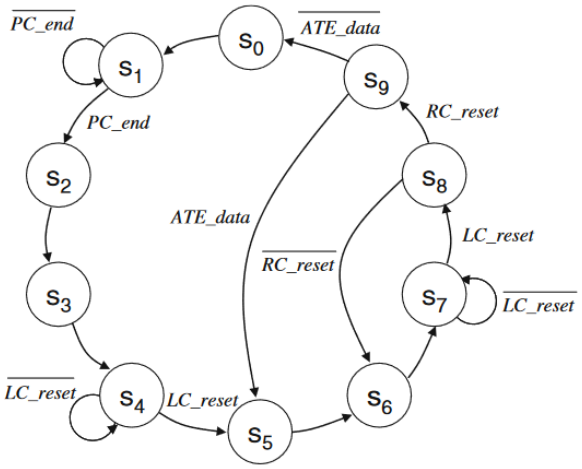

Fig.3. Finite State Diagram (El-Maleh)

Wang Dauh Tseng uses ten states (S0, S1, S2, S3, S4, S5, S6, S7, S8, S9) in the form of circles in making ob-chip system programs (figure 3).

Another test data compression for system-on-chip using Count Compatible coding call Count Compatible Pattern Run Length (CCPRL) is made to improve the compression ratio. This experimental result shows that the compression ratio achieved maximal to 71.73% [10]. For more complex data tests such as polynomially samplable sources, there is an efficient prefix-free compression algorithm and decompression algorithm. With source X is samplable compress (support subset of $\{0,1\}^n$), then [11] :

$$\text{Expected length is } H(X) + O(1) \tag{1}$$

If $H(X) \leq k = n - (\log n)$ then expected length is

$$\text{Expected length is } K + \text{polylog}(n-k) \tag{2}$$

If X is the witness set for a self reducible NP relation then

$$\text{Expected length is } H(X) + 5 \tag{3}$$

Big data with a structured data analysis use organize data in a column-wise manner. The column-wise often referred to as column stores. Columns structured such as spreadsheet applications are easily compressible. Many compression algorithms including Run Length Encoding, exploit the similarity between column value. With the comprehensive implementation of heuristic, Run Length Encoding can produce a very good compression ratio [12].

3. Method

The method for this research using the Modified Waterfall model by comparing two RLE implementations, as shown in figure 4. Step method of Modified Waterfall Model, initially with making a data collection analysis for compressing RLE using VHDL (Data Analysis). After Data Analysis, design the state diagram for HDL. Data compression design will use a way to determine the finite state diagram. The finite state diagram will be translated into the VHDL programming code. The results of compression using VHDL implementation will be measured its performance. Performance is measured based on the processing time needed to perform compression.

After conducting experiments using the VHDL implementation, then conducting experiments using the microcontroller implementation. It starts with analyzing the data set for RLE compression using assembler language. Then design a flowchart and make coding for RLE compression using assembler language. Compression results using a microcontroller implementation will also measure its performance based on the processing time.

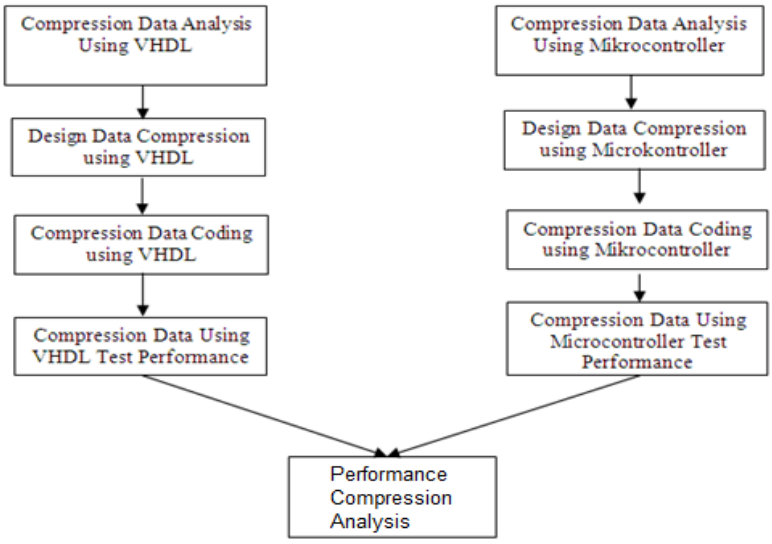

Fig.4. Modified Waterfall Model

The results of the performance measurements of the two methods will be compared. Comparisons were analyzed for compression ratio and compression time. Based on the compression ratio and compression processing time, which method is better can generally be analyzed.

Figure 5 represents a chip input-output pin design by author. Pin on-chip consist of :

- There are set data bus (from Input 0 to Input 7) for source data set.
- Output pins are output 0 to output 7
- "Take" pin is used to send an input data signal
- "ReadIn" pin is used to send a signal that the input data bus is occurring.
- When Compression is done then there is a signal from "Readdone" output pin.
- "Result" pin is used by the chip/system to send that output bus output is finished.
- For input and output synchronization then "clock" input is used.
- "Reset" pin is used to reset the system.

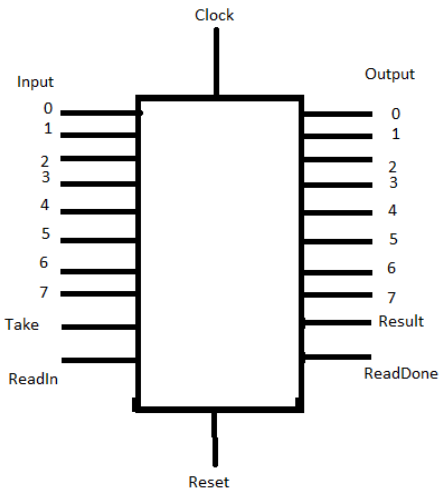

Fig.5. Pin on chip design by author

Pin Clock will be given a timing signal with the same clock frequency as the clock frequency given to the microcontroller. The data input signal is given in the form of 8 bits (Input 0 - Input 7) after the data is given then the Take pin will be given when the data is ready to be converted. The on-chip system will perform conversion processing. When the on-chip system starts to produce compression results, the signal from the ReadDone pin will be given by the system.

After the input chip output design is done, the next step is to define the finite state that is designed (figure 6). There are15 states that used for VHDL programming namely S0, S1, S2, S3, S4, S5, S5A, S6, S7, S7A, S8, S9, S10, S11, and, S12 represent state condition.

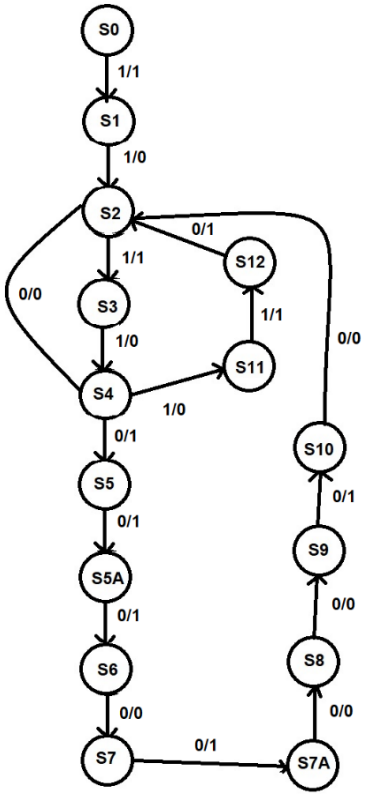

Fig.6. Finite-state Diagram

VHDL syntax for pin declaration :

```
Entity comrun is
Port (     clk, reset, readin, take : in std_logic;
           input : in std_logic_vector (7 downto 0);
           output : out std_logic_vector (7 downto 0);
                    readdone, result : out std_logic);
End comrun;
```

VHDL code for the finite state diagram model that has been designed:

```
Architecture a of comrun is
type state is (s0,s1,s2,s3,s4,s5,s5a,s6,s7,s7a,s8,s9,s10,s11,s12);
signal current_state : state;
signal satu,dua : std_logic_vector(7 downto 0);

begin

 process(clk,reset)
 variable b : integer range 0 to 255;
  begin
    if rising_edge(clk) then
```

```
 case current_state is
 when s0 => B:=0;
   if readin='1' then
    current_state<=s1;
   end if;
when s1 => B:=1;
          Satu<=Input;
          Readdone<='1';
          current_state <= s2;
 when s2 =>Readdone <= '0';
          if readin='1' then
          current_state<=s3;
          end if;
  when s3 =>Dua <= Input;
          Readdone <= '1';
          B:=B+1;
          Current_state <= s4;
  when s4 =>Readdone<='0';
          if satu = dua then
            current_state<=s2;
          else
            if b<5 then
               current_state <= s11;
           else
              current_state <= s5;
           end if;
          end if;
  when s5 =>Hasil <='0';
          B:=B-1;
          current_state <= s5a;
  when s5a=>      Hasil <= '1';
          Output <= "11111110";
          If ambil = '1' then
          current_state <= s6;
           end if;
  when s6 =>      Hasil <= '0';
          current_state <= s7;
          When s7 =>  case b is
          when 4 => output <="00000100";
          when 5 => output <="00000101";
           when 6 => output <="00000110";
           when 7 => output <="00000111";
           when 8 => output <="00001000";
           when 9 => output <="00001001";
           when 10 => output <="00001010";
           when 11 => output <="00001011";
           when 12 => output <="00001100";
           when 13 => output <="00001101";
           when 14 => output <="00001110";
           when 15 => output <="00001111";
           when others => output <="00000000";
           end case;
           Hasil <= '1';
           current_state <= s7a;
  when s7a =>     if ambil='1' then
          current_state <= s8;
          end if;
  when s8 =>      hasil<='0';
          Current_state <= s9;
```

```
        when s9 =>      Output <= Satu;
                Hasil <= '1';
                if ambil ='1' then
                        current_state <= s10;
                end if;
        when s10 => Hasil <='0';
                Satu <= dua;
                B:=0;
                Current_state <= s2;
        when s11 =>     Output <= satu;
                Hasil <='1';
                if ambil='1' then
                current_state <= s12;
                end if;
        when s12 => B:=B-1;
                Hasil <='0';
                if B<=1 then
                current_state <= s2;
                else
                current_state <= s11;
                end if;
 end case;
 end if;
end process;
end a;
```

Figure 7 represents a flowchart for data input stream programming that will be implemented to a microcontroller. Flowchart for data stream input using a standard system that has been studied by Yasmine M. Tabra [13].

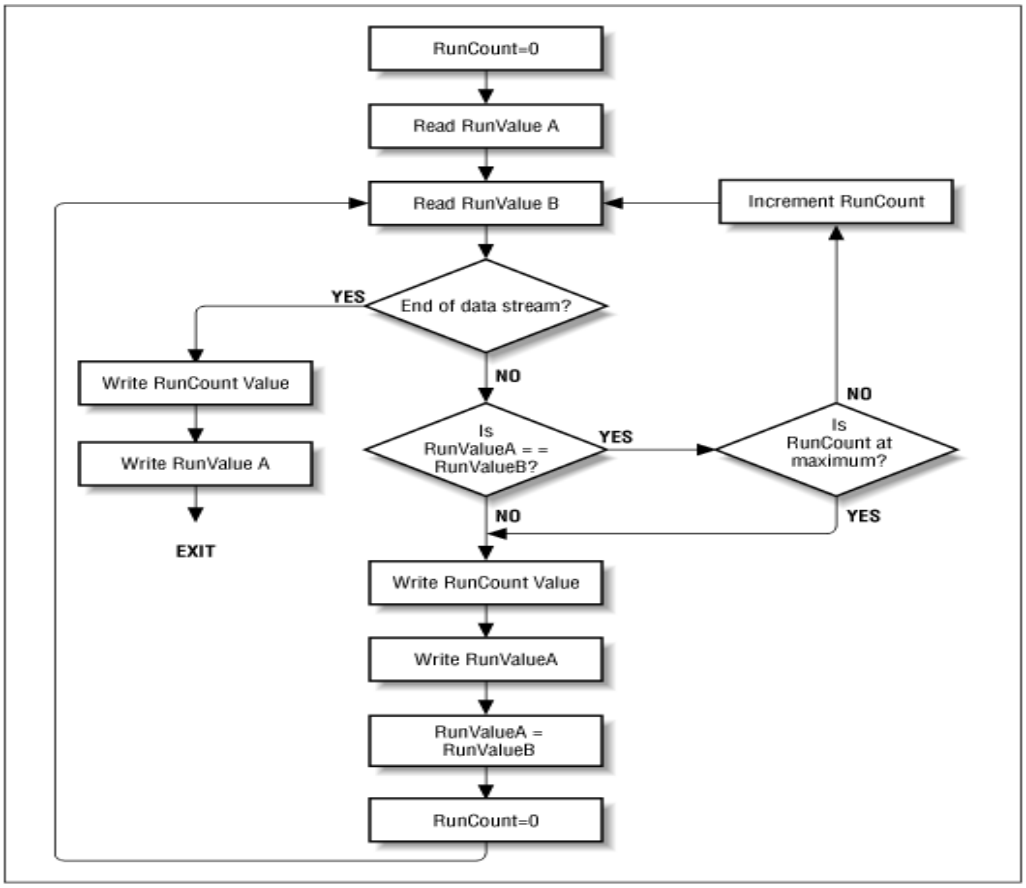

Fig. 7. Basic Flowchart for Run Length Encoding (RLE)

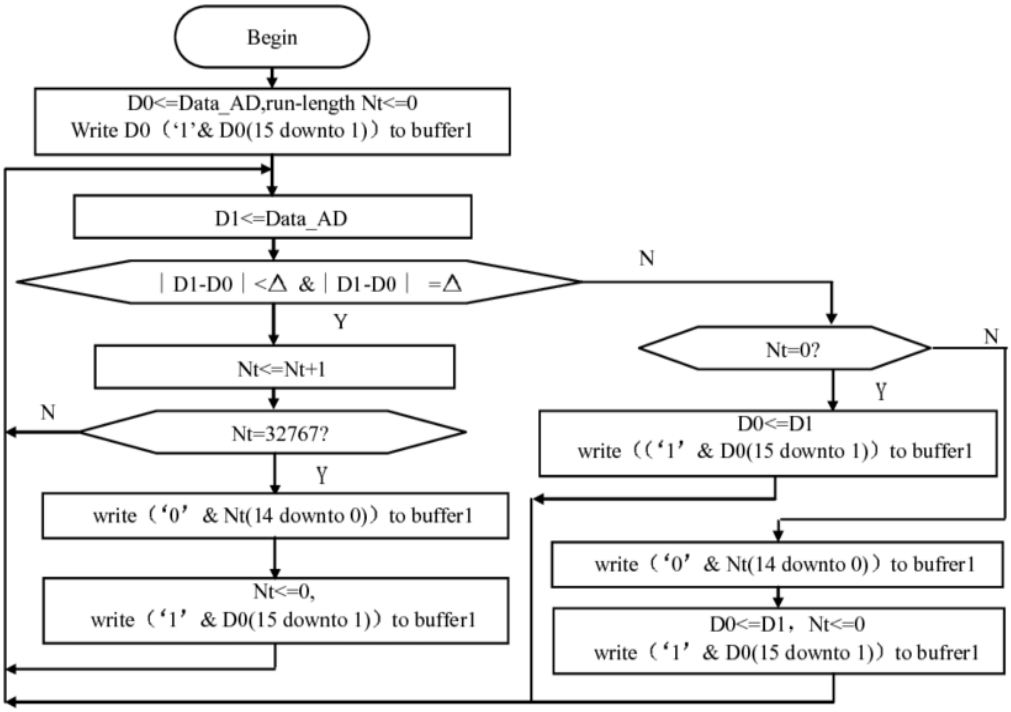

Fig. 8. Standard Run Length Encoder Flowchart

Initialization is to set RunCount to 0, then read the input data stream twice (namely RunValue A and RunValueB). Compare RunValueA and RunValue B, if both data is the same then increment RunCount value. This loop continues until RunValueA and RunValueB are different, then write RunCount value and RunValueA. Then write RunValueB into RunValueA and set RunCount to 0 again. This loop continues until the end of the data stream [14].

Flowchart for data compression algorithm using Run Length Encoder using flowchart that has been studied by Yanhu Shan as shown in figure 8 [15].

The RLE flowchart used, starts from calculating the length of the existing data, then the amount of data that has the same value. Every time there is the same data do the writing '0' and '1' depending on the results of calculations. The processing is done repeatedly until the input data stream is no longer available.

4. Results

Experiments will be conducted by providing the same set of inputs for both implementations. The expected outcome of the performance to be calculated is the speed of the data compression process with the on-chip (VHDL) implementation having a higher speed.

Table 1 represents the input given to the system and the result. If given data is 10101010 as many as five (5) times and then a signal is 11001100. This can be seen from the Readdone signal that appears 6 times. Then the output is 11111110 00000101 10101010 and 11001100. This can be seen from the results signal that appears 4 times.

So it appears first output signal and the signal marker that tells 11111110 many repetitions 5 times for example 00000101, then the signal character is 10101010. Then, if there is one signal output 110011, added to 110011 that is not a signal compression result, because the data is only supplied one time.

Table 1. Data Compression Input Output

Input
10101010
10101010
10101010
10101010
10101010
11001100

Output
11111110
00000101
11001100

The input data set from table 2 send both to VHDL programming implementation and assembler programming implementation. Both systems got the same data set and produce the same output too. It means that both system have the same RLE algorithm.

The next experiment is to provide a large data set. These data are created semi-randomly. The purpose of this experiment is not to see the value of the compression ratio, but to see a comparison of VHDL implementation and microcontroller implementation. As for the ratio compression formula is [16]:

$$\frac{Input\ data\ size - Ouput\ data\ size}{input\ data\ size} \times 100\% \quad (4)$$

Table 2. Random Input data set for VHDL programming implementation

Input data size (byte)	Output size (byte)	Compression Ratio (%)
64	22	65,63
128	36	71,88
256	80	68,75
512	166	67,58
1024	334	67,38
2048	598	70,80
4096	1234	69,87
8192	2468	69,87
16384	4980	69,60
32768	9886	69,83
65536	18902	71,16

The same data in the previous experiment was inputted into the microcontroller implementation. The results are shown in table 3.

The measurement results of the two implementations as shown in table 4 produce the same compression ratio. This means that the implementation using VHDL programming and microcontroller programming has the same compression ratio performance.

Table 3. Random Input data set for Microcontroller programming implementation

Input data size (byte)	Output size (byte)	Compression Ratio (%)
64	22	65,63
128	36	71,88
256	80	68,75
512	166	67,58
1024	334	67,38
2048	598	70,80
4096	1234	69,87
8192	2468	69,87
16384	4980	69,60
32768	9886	69,83
65536	18902	71,16

Table 4. Compression ratio comparison

Compression Ratio using VHDL implementation (%)	Compression Ratio using microcontroller implementation (%)	Difference
65,63	65,63	0
71,88	71,88	0
68,75	68,75	0
67,58	67,58	0
67,38	67,38	0
70,80	70,80	0
69,87	69,87	0
69,87	69,87	0
69,60	69,60	0
69,83	69,83	0
71,16	71,16	0

The compression ratio is not always the same for each input data given because it depends on the contents of the data provided. But for the same data, the two types of implementation are included, which will produce the same compression ratio

The next experiment is measuring compression time. A data set is entered into the VHDL implementation and microcontroller implementation. Then the results of these experiments were measured when the compression. The measurement starts from the start of compression until the compression is complete.

In this experiment, the clock given for both RLE algorithm implementations is the same. The results are as shown in table 5.

Table 5. Compression time

Input data size (byte)	VHDL implementation Compression time (ms)	Microcontroller implementation Compression time (ms)
640	112	142
1280	210	280
2560	405	550
5120	801	1025
10240	1531	2040
20480	3050	4050
40960	6075	8025
81920	12081	15978
163840	23902	31860
327680	47604	63401
655360	94206	122001

Figure 9 represents time compression for both implementations. In Figure 9, it shows that the compression ratio for the RLE algorithm using the VHDL implementation has a better compression ratio compared to the microcontroller implementation.

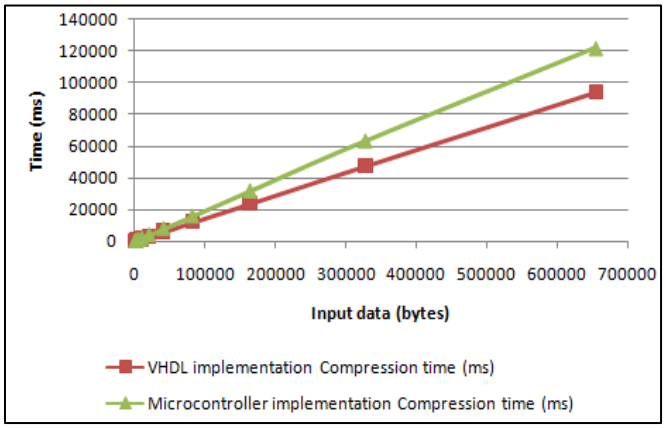

Fig. 9. Comparison of compression ratio for RLE algorithm using VHDL and Microcontroller

To find the speed of compression for each implementation of the RLE algorithm, a linear regression equation is used [17].

$$Y = a\,X + b \tag{5}$$

$$b = \frac{(n)(\Sigma XY) - (\Sigma X)(\Sigma Y)}{(n)(\Sigma X^{2}) - (\Sigma X)^{2}} \tag{6}$$

$$a = \bar{Y} - b.\bar{X} \tag{7}$$

$$\text{Compression speed} = 1 / b \tag{8}$$

Linear regression calculations will be used for both RLE algorithm implementations. The value of X is the number of data input (bytes) and the Y value is calculated twice, first the compression time (ms) for VHDL implementation and the second compression time (ms) for microcontroller implementation.

Table 6 shows linear regression analysis to find the compression speed per byte or per second in the VHDL implementation.

Table 6. Linear Regression for VHDL Implementation

	X	Y	X^2	Y^2	XY
	640	112	409600	12544	71680
	1280	210	1638400	44100	268800
	2560	405	6553600	164025	1036800
	5120	801	26214400	641601	4101120
	10240	1531	104857600	2343961	15677440
	20480	3050	419430400	9302500	62464000
	40960	6075	1677721600	36905625	248832000
	81920	12081	6710886400	145950561	989675520
	163840	23902	26843545600	571305604	3916103680
	327680	47604	1.0737E+11	2266140816	15598878720
	655360	94206	4.2949E+11	8874770436	61738844160
Σ	1310080	189977	5.7266E+11	11907581773	82575953920

n=11

$\bar{X} = 119098.2$

$\bar{Y} = 17270.64$

$$b = \frac{\left((11 \times 82575953920) - (1310080 \times 11907581773)\right)}{(11 \times (5.7266E + 11)) - 1310080^2}$$

b = 0.143891

Compression speed = 1/b = (1/0.143891) = 6.949687

This means that the compression speed in the VHDL implementation is 0.143891 milliseconds/Byte or 6.949687 KByte/second.

Table 7 shows linear regression analysis to find the compression speed per byte or per second in the VHDL implementation.

Table 7. Linear Regression For Microcontroller implementation

	X	Y	X^2	Y^2	XY
	640	142	409600	20164	90880
	1280	280	1638400	78400	358400
	2560	550	6553600	302500	1408000
	5120	1025	26214400	1050625	5248000
	10240	2040	104857600	4161600	20889600
	20480	4050	419430400	16402500	82944000
	40960	8025	1677721600	64400625	328704000
	81920	15978	6710886400	255296484	1308917760
	163840	31860	26843545600	1015059600	5219942400
	327680	63401	1.0737E+11	4019686801	20775239680
	655360	122001	4.2949E+11	14884244001	79954575360
Σ	1310080	249352	5.7266E+11	20260703300	1.0769E+11

n=11

$\bar{X} = 119098.2$

$\bar{Y} = 22668.36$

$$b = \frac{\left((11 \times (1.0769E + 11)) - (1310080 \times 20260703300)\right)}{(11 \times (5.7266E + 11)) - 1310080^2}$$

b = 0.187217

Compression speed = 1/b = (1/0.187217) = 5.341397

This means that the compression speed in the microcontroller implementation is 0.187217 milliseconds/byte or 5.341397 Kilobyte/second.

From the results of these experiments, it was found that the implementation using VHDL has more compression speed than implementation using Microcontroller. The calculation of the percentage of compression speed using VHDL has a greater value:

$$\frac{6.949687 - 5.341397}{5.341397} \times 100\% = 30.11\%$$

From the calculation of RLE implementation using VHDL, it has a greater compression speed of 30.11%. Data retrieval is collected randomly with a large number of bytes to reduce bias from the input data sample and to increase the reliability and accuracy of the study.

5. Conclusions

From the two data retrieval to analyze the performance of data compression for two types of implementation: implementation using VHDL programming and implementation using a microcontroller (assembler language) obtained the following results.

In the first data collection to analyze the compression ratio, it was found that the two implementations produced the same compression ratio. The compression ratio is not always the same for each input data given because it depends on the contents of the data provided. But for the same data, the two types of implementation are included, which will produce the same compression ratio.

In the second data collection to calculate the compression speed, it was found that the compression speed using the VHDL implementation was 6.949687 Kilobytes/second and the speed of compression using the microcontroller implementation was 5.341397 Kilobyte/second. This means that the compression speed of the RLE algorithm using the VHDL implementation has a higher speed than the one that uses the microcontroller implementation.

The compression speed improvement that was carried out successfully was carried out by conducting several experiments making finite state diagrams with many different states. The finding in this study found that by using 15 states obtained good results.

Acknowledgment

Thank you to Computer Engineering Department - Maranatha Christian University for helping carry out this research, for assistance in terms of finance and facilities for this research.

References

[1] Nelson, Gailly, M.J.L. *The Data Compression Book*. Second Edition. M&T Books, 1995.
[2] Pu, I. M. *Fundamental Data Compression*. London: Butterworth Heinemann, 2006.
[3] Patil, R.B, Kulat, K.D., *Image and Text Compression Using Dynamic Huffman and RLE Coding*, Proceedings of the International Conference on Soft Computing for Problem Solving, pp. 701-708, 2011.
[4] Sujaini, H., Mulyani. Y., *Pemampatan File*. Institut Teknologi Bandung. Bandung, 2000.
[5] Yuan, H., Guo, K., Sun, X., Ju. Z., *A Power-Efficient Test Data Compression Method for SoC using Alternating Statistical Run-Length Coding*, Journal of Electronic Testing, Vol 32(1), pp. 59-69, 2016.
[6] Makinen, Ukkonen, Navarro, *Approximate Matching of Run-Length Compressed Strings*, Algorithmica, Vol 35(4), pp 347-369, 2003
[7] Chen, K.Y., Chao, K.M., *A Fully Compressed Algorithm for Computing the Edit Distance of Run-Length Encoded Strings*, Algorithmica, vol 65(2), pp 354-370, 2013
[8] Chandra, A., Chakrabaty, K., *Analysis of Test Application Time for Test Data Compression Methods Based on Compression Codes*, Journal of Electronic Testing, vol 20(2), pp 199-212., 2004.
[9] Tseng, W.D., Lee, L.J., *Test Data Compression Using Multi-dimensional Pattern Run-length Codes*, Journal of Electronic Testing, vol 26(3), pp 393-400, 2010.
[10] Yuan, H., Mei, J., Song, H., Guo, K., *Test Data Compression for System-on-a-Chip using Count Compatible Pattern Run-Length Coding*, Journal of Electronic Testing, vol 30(2), pp 237-242, 2014.
[11] Trevisan, L., Vadhan, S., Zuckerman, D., *Compression of Samplable Sources*, Computationally complexity, vol 14(3), pp 186 – 227, 2005
[12] Jovanovski, J., Arsov, N. Stevanoska, E., Simons, M.S., Velinov, G., *A meta-heuristic approach for RLE compression in a column store table*, Soft Computing, pp. 1-22, 2018.

[13] Tabra, Y.M., Sabbar, B., F*PGA Implementation of New LM-SPIHT Colored Image Compression with Reduced Complexity and Low Memory Requirement Compatible for 5G*, International Journal of Reconfigurable and Embedded Systems, pp 1-13, 2019.
[14] Murray, James, D., Van Ryper, William, *Encyclopedia of Graphics File Format*, 2nd edition, O'Reilly & Associates, Inc, 1996.
[15] Shan, Y., Ren, Y., Zhen., Wang, K., *An Enhanced Run-Length Encoding Compression Method for Telemetry Data*, Polska Akademia Nauk, pp 551-562, 2017
[16] Mahammad, F.S., Viswanatham, V.M., *Performance analysis of data compression algorithms for heterogeneous architecture through parallel approach*, The Journal of Supercomputing, pp 1-14, 2018
[17] James, G., Witten. D., Hastie, T., Tibshirani, R., *Linear Regression*, An Introduction to Statistical Learning, pp 59-126, 2013

Author's Profile

Marvin Chandra Wijaya, a senior lecturer of Computer Engineering Departement, Maranatha Christian University, Indonesia. He received a bachelor's degree in Electrical Engineering at Maranatha Christian University – Indonesia. He received a master's degree in Management at Parahyangan Catholics University - Indonesia. He also received a master's degree in Computer Engineering at Institute Teknologi Bandung – Indonesia. His research interests are Multimedia, Microcontroller, and artificial intelligence.

How to cite this paper: Marvin Chandra Wijaya, " Comparative Analysis of Performance Run Length (RLE) Data Compression Design by VHDL and Design by Microcontroller", International Journal of Modern Education and Computer Science(IJMECS), Vol.13, No.6, pp. 11-24, 2021.DOI: 10.5815/ijmecs.2021.06.02