International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 11, November - 2013

FPGA Realization of RLE Scheme for Highspeed Data Transmission with Improved Compression Rate

K. Babulu,
Professor of ECE, Department of ECE, UCEK, JNTUK, Kakinada,

Y. Vijay kumar,
Project Associate, Department of ECE, UCEK, JNTUK, Kakinada,

Abstract

This paper describes a modified scheme for Run Length Encoding with Tolerance factor. The significant improvement of compression ratio for almost any kind of data can be achieved by the proposed scheme. An improved Run Length Encoding scheme with the introduction of Tolerance factor is described. The proposed solution has been suggested and performed for each problem to achieve efficient coding. Some of major problems are with design of RLE for very large efficient data compression. This has been resolved by Introducing Tolerance factor in RLE. So that larger sequences that affects compression ratio are broken into small sequences Using Tolerance factor. By using Tolerance factor we can consider the any two approximate bit sequences are as one bit and hence we can save more memory. Run Length Encoding (RLE) scheme is described using VHDL language and is simulated on Xilinx ISE environment. By introducing Tolerance factor for this Run Length Encoding (RLE) scheme compressor rate is greatly improved there by storage space required also reduces greatly.

1. INTRODUCTION

The data compression is a process that reduces the amount of data in order to reduce the data transmitted and decreases transfer time because the size of data is reduced [2]. The data compression is commonly used in the modern database systems. The Compression can be utilized in the different reasons including:

1) Reducing the storage/archival costs, this is particularly important in large data warehouses.

2) The Improving query of Workload performance by reducing I/O costs [3].

The data compression involves the transforming a string of characters in some representation into a new string which contains same information but with the smallest possible length. The data compression very has important application in the areas of data transmission and the data storage [6]. The Compressing data reduces storage and the communication costs. Similarly, the compressing a file to the half of its original size is equivalent to doubling the capacity of storage medium. The Data compression is rapidly becoming a standard component of the communications hardware and data storage devices [4].

2. RUN LENGTH ENCODING

The Run Length Encoding (RLE) is a very simple form of the data compression in which runs of the data are stored as a single data value and count, rather than as original run. This is the most useful on data that contains many such runs: for example, the simple graphic of the images such as icons, line drawings, and the animations. This is not useful with files that do not have many runs as it could greatly increase size of the file.

Run Length Encoding (RLE) algorithm performs a lossless compression of the input data based on sequences of identical values. This is a technique of the historical, and it is originally exploited by fax machine and later adopted in the image processing. This algorithm is quite easy: each run, instead of the being represented explicitly, is translated by encoding algorithm in a pair (l, v) where l is the length of run and v is the value of run elements. The longer the run in sequence to be compressed better is the compression ratio [5].

A. Working of Run Length Encoding(RLE)

The n bit of data is compressed by arranging it in the form of run and count of each run. The count of each run is then represented by binary for the case of binary data. The Amount of data compressed is directly related to the length and the number of longer runs. Run Length Encoding (RLE) when applied on the data with information bits ‘‘1111111111110000000000001111” gives us a subset of the 3 pairs, each pair representing the number of runs and bit. Hence, the above mentioned bit pattern in Run Length Encoding (RLE) scheme is represented as (12, 1) (11, 0) (4, 1). In binary form the latter pairs are expressed as follows: 12=01100, 11=01011 and 4=00100. The final output comes out to be 01100**1**01011**0**00100**1**. In this way, the original pattern of the n bits can be compressed to a great extent thereby reducing data.

This type of encoding scheme does not always perform data compression. In some scenarios where the runs of smaller length are in excess, this scheme performs poorly and instead of compressing the data, the resultant output is an expanded form of the input. Consider a pattern "101010" when applied by Run Length Encoding, the final output comes out to be an expanded form of input data. The final output is (1,1)(1,0)(1,1)(1,0)(1,1)(1,0)or 111011101110 which is larger in size than the input.

In this case of large consecutive runs of 1's or 0's, RLE performs efficient compression whereas in case of a data with large number of single 0's or 1's, the output is an expanded form of input sometimes the output is twice the size of input. This expansion of data instead of compression proves RLE technique less reliable. That is why run length encoding is a poor technique and practically not efficient for larger data. The core objective of this research paper is to improve this encoding technique.

The length of largest run decides the number of bits needed to represent the count or length of each run in run length encoding technique. Consider a bit pattern 01010011111111111. The largest run in the data is 11 number of 1's appearing consecutively. Therefore this run decides the number of bits needed to represent length of each run in data. The number of bits needed to represent the length of run is 4 or the given scenario. The above sequence is written in run length encoding as,

Table I. RLE Pairs for the above Data

BIT	RUN LENTH ENCODING
0	0001,0
1	0001,1
0	0001,0
1	0001,1
00	0010,0
11111111111	1011,1

B. Problem with Run Length Encoding

There are two basic problems that degrade the performance of Run Length encoding schemes. Most of the Bits in a data are arranged in runs of smaller lengths that include single zeroes or single ones, double zeroes or double ones. Such combination requires more number of bits than their actual size to represent them in run length encoding technique. A single 0 may be represented as (000001, 0) which is seven times larger than the input data. This is one of the major performances limiting factor and results in expansion of data instead of compression. Sometimes a data may contain a very large sequence of consecutive ones or zeros. Such sequences are represented in fewer numbers of bits in RLE but they might affect the overall compression in a negative way as largest sequence decides the number of bits to represent the length of a run in each pair. As a result the length of run in all the other sequences is also represented by the same number of bits for which the largest run is represented. For the scenario given below we need 4 extra bits to represent the length each single bit 0/1 because the longest sequence is represented by 4 bits as shown below.

Table II. 4-bit Long Sequence

0	0001,0
111111111111111	1111,1

3. MODIFIED RLE SCHEME

3.1. Proposed Solution

As indicated above, there are two very clear problems that decrease the performance of run length encoding scheme. We have proposed some modifications in run length encoding scheme. These modifications are specially designed to counter the above mentioned problems. The modified run length encoding scheme gives a significant improvement in compression ratio for almost any kind of data.

The above mentioned problem in Run Length Encoding Scheme can be overcome by intelligent compression. Analyzing the input data is the first and core step. We analyze data to highlight if there are any largest numbers of sequences that may increase the number of bits to represent the length of each run. Secondly we highlight the smaller sequences of single zeroes/ones double zeroes/ones or triple zeros/ones that may result in expansion of data instead of compression.

3.2. Tolerance factor

Tolerance factor is used for various purposes, such as for bringing bit streams that do not necessarily have the same or rationally related bit rates up to a common rate, or to fill buffers or frames. The location of the tolerance factor bits is communicated to the receiving end of the data link, where these extra bits and approximate bits are removed to return the bit streams to their original bit rates or form [1].

Compression rate can be improved by combining two or more adjacent values. While counting for a run, if the next run to be counted is very small distance for the current run then the two runs can be merged. The distance between two runs is called tolerance factor. Those two runs can be merged only when the effect of merging is within the considerable range. When the two runs are merged with a common run, while decompressing original data retrieved may not exactly same with the original one. There may be some deviations. Merging is to be done only when the effect of merging is within the considerable range [9].

Let us consider some raw data that was observed with a temperature sensor which is located in a furnace. If the sample raw data that was collected in a furnace is 45, 45, 45, 45, 45, 46, 46, 46, 48, 48 then it can be compressed by considering a tolerance factor. If the tolerance factor is one then the compressed data is observed like (45, 8) (48, 2). Without tolerance factor the compressed data may appear like (45, 5) (46, 3) (48, 2). By using the tolerance factor the compression rate is more. But when in decompression we may lose some information. The decompressed information may appear like 45, 45, 45, 45, 45, 45, 45, 45, 48, 48.

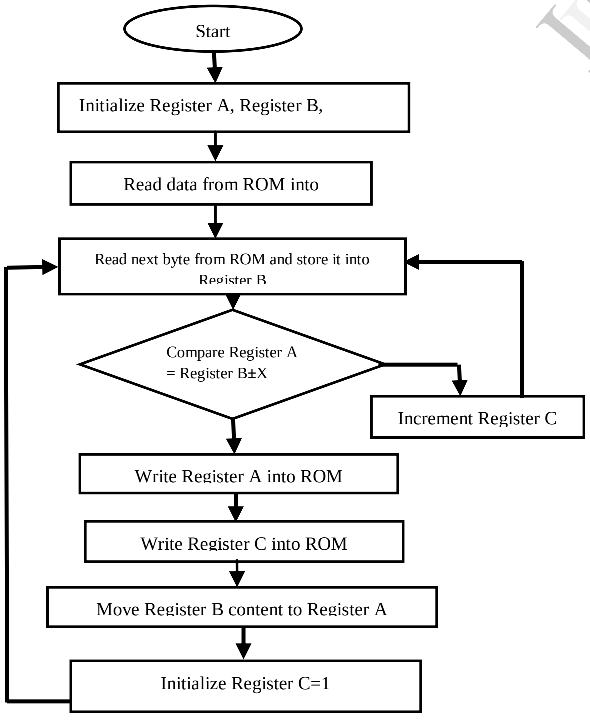
Start
Initialize Register A, Register B,
Read data from ROM into
Read next byte from ROM and store it into Register B
Compare Register A = Register B±X
Increment Register C
Write Register A into ROM
Write Register C into ROM
Move Register B content to Register A
Initialize Register C=1

Figure.1: Flow Diagram of Modified RLE Scheme

The decompressed information may not match exactly with the original raw data but this change may not affect the information to be communicated.
In the tolerance factor repeated bits and approximate bits are considered as single bit. One example for the compression process is shown in Table I. By using the tolerance factor technique memory required to store information can be reduced, also in communication systems the data transfer rate is improved. It is considered that the tolerance factor, X= ±1 so that the adjacent value of the byte is considered as the same byte and both they are merged at an initial value. Figure.1 shows the algorithmic flow how the Run Length Encoding scheme is implemented and also how the tolerance factor is included to the Run Length Encoding technique.

4. HDL IMPLEMENTATION

The proposed technique is developed with VHDL which contain five modules and those are compressor, de-compressor, controller, and memory. The block diagrams of compressor and De-compressor are shown in Figures 2 and 3 respectively.

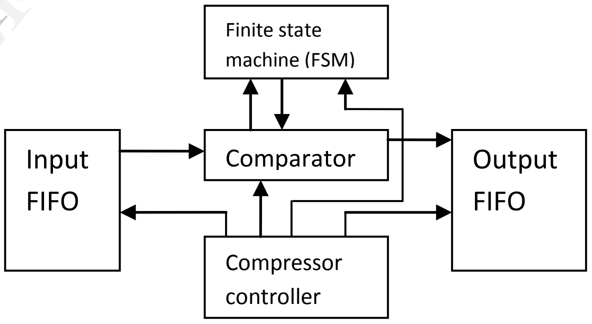

Figure.2: Block diagram for Compressor

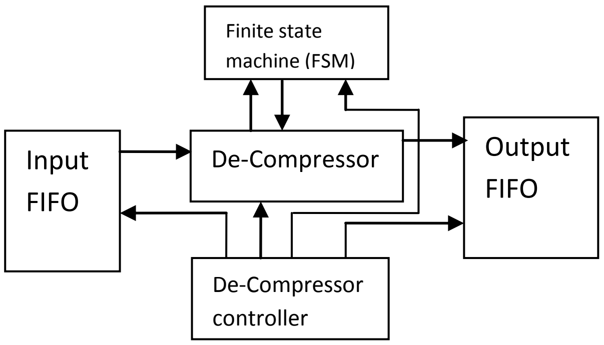

Figure.3: Block diagram for de-compressor

In compressor section the data is taken from the ROM/ FIFO and then given to the Compressor, so that the compressor is used to compress the input data and it given to the output ROM. In the de-compressor section the input is compressed data and it is given to the de-compressor, then the de-compressed data is send to the output ROM/ FIFO. So that the actual information is retrieve from the de-compressor section. The Finite state machine (FSM) is used to control the compressor and de-compressor's position. The RLE compressor is implemented according to the state diagram shown in Figure.4.

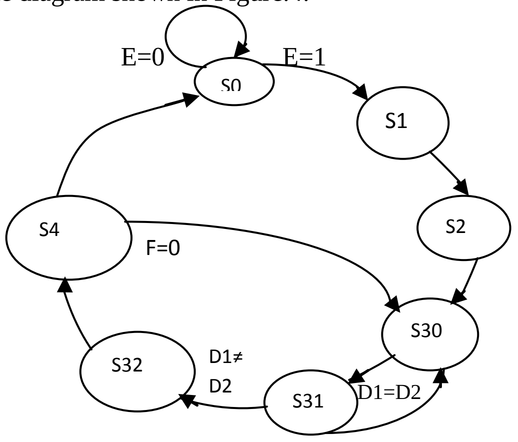

Figure.4: State Diagram for Compressor.

In this state machine enable=1 then it will goes to FIFO read1 state otherwise it will present in same state. Then after it will goes to Data1 state after that it will goes to FIFO read2 state. FIFO read2 state will goes to Data2 state. When Data1≠ Data2, then its changes to the Forward data state otherwise Data2 state goes to FIFO read2 state. The Forward data state goes to Forward count. When Finnish=0 this state changed to FIFO read2 state otherwise Forward count state goes to ideal state.

TABLE III: Expansion of States

States	Expansion
E	Enable
F	Finish
S0	Idle
S1	FIFO read1
S2	Data1
S30	FIFO read2
S31	Data2
S32	Forward Data
S4	Forward count

The RLE De-compressor is implemented according to the Stat diagram shown in Figure.5.

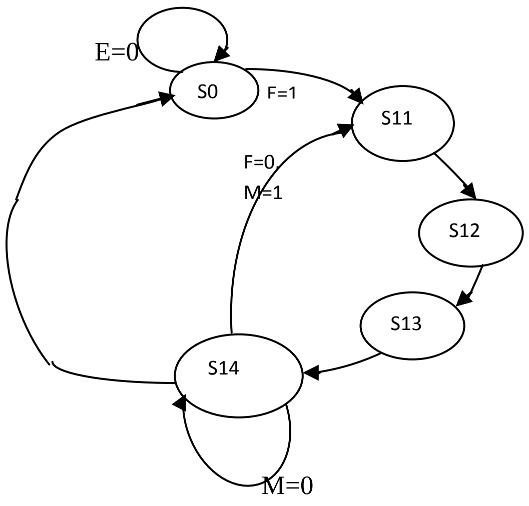

Figure 5: State Diagram De-Compressor.

In this state machine enable=1 then it will goes to Data1 state after that it will changed to Data2 state. After changed to the Forward data state from Data2 state register is compare with counter. If Register is equal to counter then Match=1 and Finish=0 so that Forward data state is changed to FIFO read1 state, it is not equal Match=0. When Finish=1 the loop will goes to the ideal state.

TABLE IV: Expansion of States

States	Expansion
E	Enable
F	Finish
M	Match
S0	Idle
S11	FIFO read1
S12	Data1
S13	Data2
S14	Forward Data

5. SIMULATION RESULTS

5.1. Schematic view

The schematic view shown in Figure.6 has two sub modules namely transmitter controller and compression controller. In transmitter controller section ComEna signal is used as compression enable signal. All the other control signals are generated and are controlled by the transmitter controller. In compression controller section the 8-bit data is giving to the compression controller through the channel as shown in the Figure.6. So that we can collect the compressed data from the data output signal.

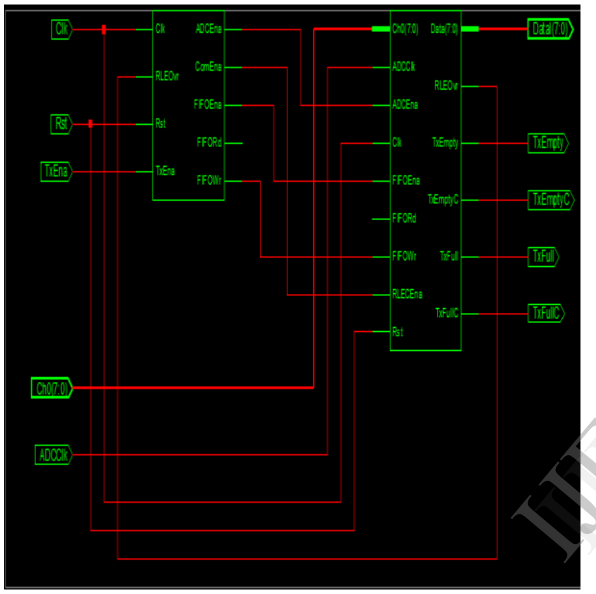

Figure 6: Schematic view for Top order Module

The simulation results of the top order module are given below. An Analog to digital converted (ADC) output is given to the Compressor by the help of FIFO controlled by the compressor controller so that we can get compressed output. A compressor output is given to the de-compressor with the help of FIFO controlled by the de-compressor controller so that we can get the de-compressed output.

5.2. Simulation results

Simulation results for the proposed technique is shown in Figure.7 Simulation wave forms shows the raw data that is supplied to the input FIFO and the compressed data as well as decompressed data in two different FIFOs. This table contains Input data, compressed data, Decompressed data and saved bytes are given below. We have given 15 bytes to the compressor then it will give 6 bytes of compressed data. So that we have saved total 9 bytes.

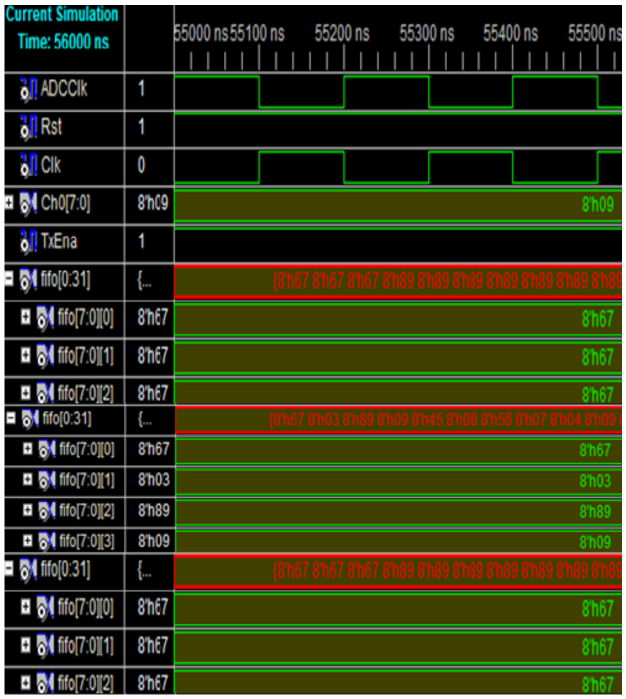

Figure.7: Simulation Results of Modified RLE Scheme

It is observed from the following Table V that 67 67 67 is input data to the compressor then it will give 3x67. But 89 89 89 89 89 89 88 data contains one 88, so that 88 also can taken as an approximate value of the actual data by using Tolerance factor technique. Hence the obtained compressed output is 7 x 89.

TABLE V: Modified RLE Algorithm showing Data Compression

Input Data	Compressed data	Bytes saved	Decompressed data
67 67 67	3 x 67	1	67 67 67
89 89 89 89 89 89 88	7 x 89	5	89 89 89 89 89 89 89
45 45 45 45 44	5 x 45	3	45 45 45 45 45
Total bytes=15	Total bytes=6	Total Bytes saved=9	Total bytes=15

5.3. Synthesis report

Modified RLE scheme is described using VHDL language and is simulated using Xilinx ISE. Synthesis has been carried out by Xilinx synthesizer. This design is targeted on Xilinx Spartan3E FPGA. PACE is the application where the pin assignment is done for Spartan 3E. Pin locations for the ports of the design are specified so that they are connected correctly on the Spartan-3E. After pin assignment, re-implemented the design and verified that the ports of the counter design are routed to the package pins specified. Impact is an application used to download.

TABLE: VI SYNTHESIS REPORT

Device utilization summary:	
Selected Device :	3s400tq144-5
Number of Slices:	94 out of 3584 2%
Number of Slice Flip Flops:	88 out of 7168 1%
Number of 4 input LUTs:	189 out of 7168 2%
Number used as logic:	141
Number used as RAMs:	48
Number of IOs:	24
Number of bonded IOBs:	24 out of 97 24%
Number of GCLKs:	2 out of 8 25%

The Synthesis Report is shown in the above Table VI which gives the Device Utilization Summary.

6. CONCLUSION

In this paper a new and more reliable technique for data compression is proposed and is realized on Spartan 3E FPGA. It solves the limitations present in run length encoding scheme. Problems in run length encoding are highlighted and discussed in detail. A solution to each problem is then proposed in modified run length encoding scheme. The sequence is taken and analyzed. To make run length encoding more functional Tolerance factor is included. Here in this project tolerance factor is considered as 1. It can be extended such that the decompressed data should not affect the information to be transmitted. When tolerance factor is more compression also more but the decompressed data may not exactly match with the original raw data. There should be a compromise between tolerance factor input data compression ratio. In this paper tolerance factor is selected as one. If the tolerance factor increased further compression ratio also increases.

REFERENCES

[1] Asjad Amin, Haseeb Ahmad Qureshi, Muhammad Junaid, Muhammad Yasir Habib, WaqasAnjum, “Modified Run Length Encoding Scheme with Introduction of Bit Stuffing for efficientData Compression” in 6th International conference on internet Technology and secured IEEE Transactions, 11-14 December 2011, 978-1-908320-00-1/11.

[2] Eug`enePamba Capo-Chichi, Herv´eGuyennet, Jean-Michel Friedt,“A new Data Compression Algorithm for Wireless Sensor Network,”in Proc Third International Conference on Sensor Technologies and Applications,2009, pp.1-6 DOI 10.1109/SENSORCOMM.2009.84.

[3] StratosIdreos, RaghavKaushik, VivekNarasayya, Ravishankar Ramamurthy, “Estimating the Compression Fraction of an Index using Sampling,”in*Proc. International Conference on Data Engineering (ICDE)*, 2010, doi. 10.110/ICDE.2010.5447694.

[4] James A. Storer, “Data Compression Methods and Theory,”Computer Science Press, 1988, 413 pp, ISBN-10: 0716781565.

[5] Stefano Ferilli, “Automatic Digital Document Processing and Management: Problems, Algorithms and techniques,”ISBN: 0857291971.

[6] Viswabharathi, D., K. Raghuram, and G. Rajesh Kumar. "High Speed Test Architecture for SRAM using Modified March Algorithm." International Journal of Engineering Research and Applications (IJERA) Vol. 2, Issue 6, November- December 2012, pp.1654-1659.

[7] Y. Wang and K. Roy, Maximum power estimation for CMOS circuits using deterministic and statistical approaches, in Proc. VLSI Conf., pp. 364–369, January 1995.Applications,2009, pp.1-6 DOI 10.1109/SENSORCOMM.2009.84

[8] Ghosh, S. Devadas, K. Kuetzer, J. White, “Estimation of average switching activity in combinational and sequential circuits”, Proc. of IEEE/ACM DAC, 1992, pp. 253-259

[9] SIA, The International Technology Roadmap for Semiconductors: 2005 Edition, Semiconductor Industry Association, San Jose, CA, 2005

[10] Pujar, J.H.; Kadlaskar, L.M. (May 2010). "A New Lossless Method of Image Compression and Decompression Using Huffman Coding Techniques". Journal of Theoretical and Applied Information Technology 15 (1): 18–23