RESEARCH ARTICLE OPEN ACCESS

Improved Run Length Encoding Scheme For Efficient Compression Data Rate

S. Sarika*, S. Srilali**
(Department of Electronics and Communication Engineering)
(Swarnandhra College of Engineering and Technology)

ABSTRACT
Recent technological breakthrough in high speed processing units and communication devices have enabled the development of high data compression schemes. This paper presents a modified scheme for Run length encoding (RLE). Run length encoding algorithm performs compression of input data based on sequences of identical values. RLE is having some limitations and they have been highlighted and discussed in detail in this paper. In RLE largest number of sequences may increase the number of bits to represents the length of each run, which may increase the size of memory stack which may results in performance degradation. For n-bit run it requires 2n memory stack. If run is greater than n bits we require 2n+1 memory stack to store the run value. An efficient coding technique, Bit stuffing has been suggested in this paper. A new bit different from the original sequence is added in between reduces the repeat length, thereby with the same stack we can represent length as well. This technique is described using VHDL and is implemented on Saprtan3 FPGA

Keywords - bit stuffing, compression, memory stack, run, Run length encoding (RLE)

I. INTRODUCTION

Data Compression Is A Process To Reduce The Number Of Bits Used To Store Or Transmit Information And Decreases Space Because Size Of Data Is Reduced, Time To Transmit, And Cost. Data Compression Is Commonly Used In Modern Data Base Systems. Data Compression Is Used For Different Reasons. 1) To Decrease Data Bulk And To Decrease Data Transportation.2) Make Optimal Use Of Limited Storage Spacedata Compression Involves Transferring Of A String Of Data In Some Representation (Such As BINARY, ASCII) Into New String Which Contain The Same Information But In Reduced Length. The Compression Technique Is To Identify Redundancy And To Eliminate It. Data Compression Removes The Useless And Reforms The Data. There Are Two Categories Of Compression Techniques. They Are: 1) Lossy Compression Technique. 2) Lossless Compression Technique. Lossy Compression Methods Include DCT (Discreet Cosine Transform), Vector Quantization And Huffman Coding And Lossless Compression Methods Include RLE (Run Length Encoding), String-Table Compression, LZW (Lempel Ziff Welch).

This paper is organized as follows: Section II presents original run length encoding scheme, Section III represents improved run length encoding scheme with bit stuffing, Section IV presents the VHDL implementation of RLE, Section V verifies the result of proposed encoding scheme, Section VI represents conclusion.

II. RUN LENGTH ENCODING

Run Length Encoding (RLE) is a simple and popular data compression algorithm. It is based on the idea to replace a long sequence of the same symbol by a shorter sequence and is a good introduction into the data compression field for newcomers.

The RLE algorithm performs a lossless compression of input data based on sequence of identical values (runs).In this algorithm is represents explicitly by a pair (v, l) where v is the value and l is the length of the value.

For instance, Run-length encoding when applied on the data with information bits “11111111111111110000000000011111”. The above input sequence in the run length encoding scheme is represented as (1,16)(0,11)(1,5).in binary forms expressed as 16=10000, 11=01011, 5=00101.Final output comes out to be 11000000101110010.

The length of largest run will decides the number of bits to represent the count or length of each run in the run length encoding scheme. Consider the bit pattern 110111111111111.The largest run in the data is 12 number of 1’s appearing consecutively. Therefore this run decides the number of bits needed to represent the length of each run in the data. The number of bits needed to represent the length of the run is 4 or the given scenario. The above sequence is written in run length encoding as:

Table I: RLE encoding example

BIT	RUN LENGTH ENCODING
11	1,0010
0	0,0001
111111111111	1,1100

The basic problem that degrades the performance of run length encoding technique is sometimes a data may contain a very large sequence of consecutive ones or zeros. In such sequences as the largest sequence of consecutive ones/zeros decides the number of bits to represent the length of the run. As a result the length of the run in all other sequences is also represented by the same number of bits. This in turn increases the size of memory stack and decreases the transmission speed of data. The main objective of this paper is to improve this encoding technique.

III. IMPROVED RUN LENGTH ENCODING SCHEME

To increase the performance and transmission speed of run length encoding scheme we have proposed some modifications. The modified run length encoding scheme gives improvement in compression ratio.

Bit stuffing is the process of inserting non information bits into data to break up bit patterns to affect the synchronous transmission of information. It is widely used in network and communication protocols, in which bit stuffing is a required part of the transmission process. The location of the stuffed bit is communicated to the receiving end and extra bits are removed from the original data at the receiving end. The Receiver end requires the information of maximum allowable consecutive bits

Bit stuffing works to limit the number of consecutive bits of the same value included in the transmitted data for run-length limited coding. This procedure includes a bit of the opposite value after the maximum allowed number of consecutive bits of the same value. Stuffed bit should not confuse with overhead bits. In modified RLE, 15 consecutive ones are represented by 4 bits and 17 consecutive ones are represented by 5 bits. In this paper the length of the sequence after which the bit will be stuff is 15 consecutive ones/zeros.

20 consecutive ones are represented by 5 bits of run.

$$\underbrace{11111111111111111111}_{10100,\ 1}$$

By Bit Stuffing is used 4 bits are sufficient to represent run

$$\underbrace{111111111111111}_{1111,\ 1}011111$$

Therefore by bit stuffing it limits the more number of consecutive ones/zeros which decreases the number of bits to represent the run value and decreases the memory stack and increase in turn increases the transmission speed.

IV. HDL IMPLEMENTATION OF RLE

This paper discusses the RLE implementation in VHDL, in which the total module is divided into two different sub modules they are i)compression module and ii) decompression module.

In compression module, it has input FIFO (First In First Out) which takes the data and gives it to the Compressor which compress the input data and gives it to output FIFO. There is a controller which controls the input FIFO, compressor and Output FIFO. At Decompression module, it has input FIFO which takes the data and gives it to the DeCompressor which retrieves the original data and gives it to the output FIFO.

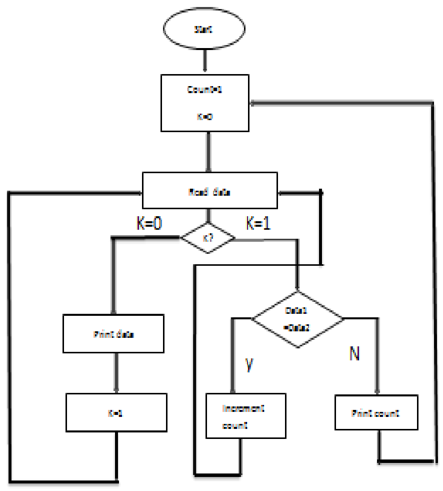

Fig 1: Flow chart for compression

Compression Algorithm:
Step-1: Start on the first element of input.
Step -2: Initialize the values with count=1, k=0.
Step-3: Read the first element of input data1.
Step-4: As the value of K is '0' it will print the input data1 and then it increments the K value to '1'.
Step-5: Again it goes to step-3 takes the second data2 and next the checks the value of k
Step-6:As the value of k is 1 it now it will checks for whether data1=data2,if the data1=data2 it will increment the count value if not equal it prints the count value after the data1 value and again goes to step 2.

Decompression Algorithm:
Step-1: Start on the first element of the data input
Step-2: Read the data and store it in a register A and initialize C with '1'
Step -3: Print the data which is in register A
Step-4: Take the second data from input and store it on register B.
Step-5: If C=B then go to step 2 else print A and increment C and repeat.

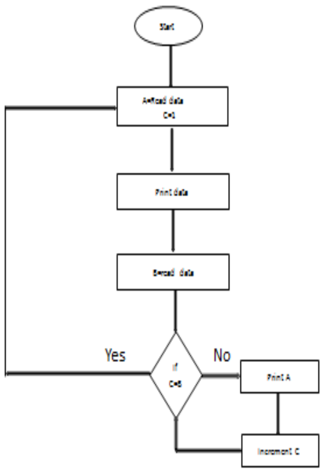

Fig: 2 Flow chart for decompression

V. Simulation Results

Raw data that is to be supplied to the compressor can be collected from any analog sensor or transmitter. That analog information should have redundant data. Then only this compression technique works well. The analog information that is collected must be converted to digital form using Analog to Digital Converter. That digital information first stored in a FIFO. First In First Out Queue is used because the transmitter and the compressor may not operate with the same frequency. In order to make them work properly a queue is used in between. Working of compressor and the queues that are placed on input side and output side are explained with the simulation results.

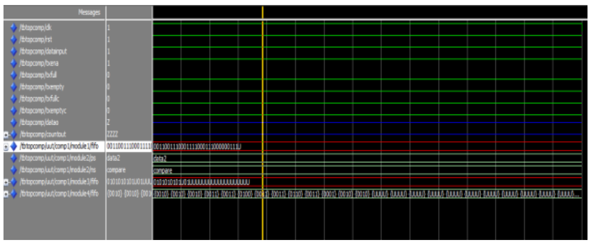

Fig 3: Compressor simulation results

Figure 3 shows the compressor simulation results. When serial data that is coming from Digital to Analog Converter is supplied to the compressor, first of all it was stored in a FIFO. After that when the compressor is enabled the compressor starts working. Compressed data is stored in out FIFOs. How the compressor works and where the raw data and the compressed data is stored can be viewed in the simulation waveforms.

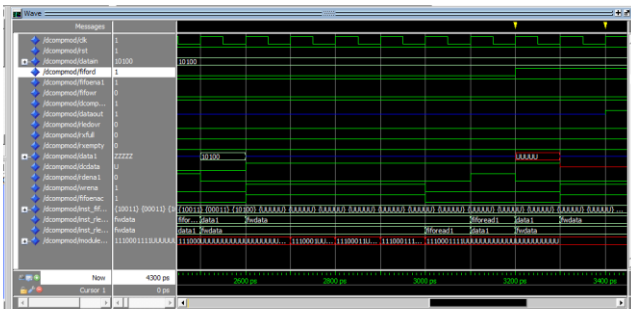

Fig 4: Decompressor simulation results

Fig 4 shows the simulation results for decompressor. For this decompressor input data coming from the compressor. First of all Data coming from the compressor is stored in a FIFO. When decompressor is enabled Decompressor retrieves information from input FIFO and then decompress the input information. The decompressed information exactly equal to the input information that is supplied to the compressor. Decompressed information is stored in output FIFO.

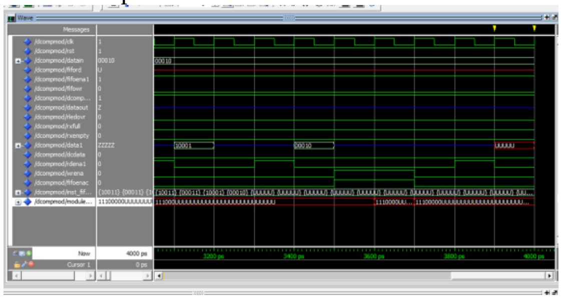

Fig 5: Bit Stuffing Decompressor simulation results.

Fig 5 shows the simulation results for Bitstuffed decompressor. For this decompressor input data coming from the compressor. First of all Data coming from the compressor is stored in a FIFO. When decompressor is enabled Decompressor retrieves information from input FIFO and then decompress the input information. The decompressor here removes stuffed bits. Decompressed information is stored in output FIFO.

VI. Conclusion

This paper provides a new and more reliable technique for data compression. To make RLE work we use bit stuffing to break larger sequences and which decreases the number of bits to represent the run value and decreases the memory stack and in turn increases the transmission speed. With a slight change in the compressor architecture, the compression ratio greatly affected.

References

[1] Eug`ene Pamba Capo-Chichi, Herv´e Guyennet, Jean-Michel Friedt, “A new Data Compression Algorithm for Wireless Sensor Network,” in Proc Third International Conference on Sensor Technologies and Applications, 2009

[2] James A. Storer, “Data Compression methods and theory” Computer Science Press, 1988
[3] C. E. Shannon, “A mathematical theory of communication,” Bell Syst. Tech. J., vol. 27, no. 3 and 4, pp. 379–423, July and Oct. 1948.
[4] S. Tate. Complexity Measures. In K. Sayood, editor, Lossless Compression Handbook, pages 35–54. Academic Press, 2003.
[5] N. Faller. An Adaptive System for Data Compression. In Record of the 7th Asilomar Conference on Circuits, Systems, and Computers, pages 593–597. IEEE, 1973.
[6] StratosIdreos, Raghav Kaushik, vivek Narasayya, Ravi shankar Ramamurthy, “Estimating the Compression Fraction of an Index using Sampling,” in Proc. International Conference on Data Engineering (ICDE), 2010
[7] X. Wu and N.D. Memon. CALIC—A context based adaptive lossless image coding scheme. IEEE Transactions on Communications, May 1996.
[8] “Efficient coding schemes for the hard-square model,” IEEE Trans. Inform. Theory, vol. 47, pp. 1166–1176, Mar. 2001.
[9] B. Ramamurthi and A. Gersho. Classified Vector Quantization of Images. IEEE Transactions on Communications, COM-34:1105–1115, November 1986.
[10] M. Hans and R.W. Schafer. AudioPak—An Integer Arithmetic Lossless Audio Code. In Proceedings of the Data Compression Conference, DCC ’98. IEEE, 1998.
[11] G. Langdon and J.J. Rissanen. Compression of black-white images with arithmetic coding. IEEE Transactions on Communications, 29(6):858–867, 1981.
[12] J. Ziv and A. Lempel. A universal algorithm for data compression. IEEE Transactions on Information Theory, IT-23(3):337–343, May 1977.
[13] M. Nelson and J.-L. Gailly. The Data Compression Book. M&T Books, CA, 1996.