ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 2, Issue 12, December 2013

Modified Run Length Encoding Scheme for High Data Compression Rate

M.VidyaSagar, J.S. Rose Victor
Amrita Sai Institute of Science and Technology, Paritala, Kanchikacherla Mdl, Vijayawada

Abstract— **Now a day's communication plays a vital role and it is the main aspect in the present world. Due to of this rapid changes are occurred and to transmit the data effectively and efficiently and it takes large time and power conception to transmit it. So it needs to decrease the data rate, the data can be compressed and it would be transmitted in an efficient manner. The efficient manner of compression data is to use Run Length Encoding (RLE) method. In this paper data compression uses Run Length Encoding (RLE) because the compression is very ability and would be having an exact output and an easy way to understand to implement. This algorithm is implemented by writing VHDL description and is simulated using Xilinx ISE software simulation tools. The paper deals mainly about data compression. There are two types of compressions - one is loss less data compression and other is lossy data compression. In this paper loss less data compression technique is used. Both techniques have their own advantages. Advantage of lossless compression is that it can send the data efficiently. It has low power consumption, reduced time delay. This include audio, video, images and detailed graphics for screen design (computers, TVs, projector screens). One of the disadvantages of loss less compression is that if the data has less repentance bytes then it takes more bits than the original data.**

Index Terms— **Lossless data compression, Lossy data compression, Run Length Encoding (RLE), VHDL**

I. INTRODUCTION

Now-a-days many data compression techniques are evolved with high compression rate. In these Techniques input date is to be encoded. By using the techniques the input data is to be compressed. To compress the original input data or to reduce the input data it will never effect the original information and it is possible to consume less power for data transmission. It is possible to transmit the data efficiently by using these methods; there two compression methods. One is Lossless compression and the other is Lossy compression method. In this paper loss less compression technique is implemented. This algorithm is described using VHDL language and is simulated using Xilinx ISE simulator.

The aim is to compress the input data using loss less compression method. It has low power consumption, reduced time delay. Loss less is only effective with media elements that can still 'work' without all their original data. These include audio, video, images and detailed graphics for screen design (computers, TVs, projector screens). One of the disadvantages of loss less compression is that if the compressed file size is more due to of redundancy to be compressed, then the quality will degrade drastically.

In this paper loss less data compression technique is used. Loss less data compression looks for 'redundant' pixel information is to be defining like a function (x, y). This means that when the file is decompressed the original data will not retrieve with a small error. Here both the techniques have their own advantages. The data from the user is obtained in which the data would be stored in FIFO. So as to get synchronization between receiver and the compressor, FIFOs are used and the FIFO would assign the priorities to the data obtained. Then the data is transferred to the COMPRESSOR block at which it would compress the data and would give the data to the output FIFO. Output FIFO is used to synchronize data coming from the compressor and the data that is transmitted through the communication channel. The decompression was also the same fashion there would be two FIFO'S and a DECOPRESSOR block which would decompress the data obtained from the FIFO and the FIFO gets the data from the transmitter.

Run-Length Encoding, as an important compression method for raster data, is broadly applied in storing GIS spatial data. The present researches on Run-Length Encoding pay more attention on the realization and optimization of the compression, encoding and decoding, while less on the raster operations that on the basis of Run-Length Encoding data structure. The raster operations, easy to be realized, are all kinds of algebraic and logical operations targeted at the attribute of raster grid; Run-Length Encoding could not only compress the raster data effectively, but also retain the intuition feature of raster data [1]. The RLE algorithm performs a lossless compression of input data based on sequences of identical values (runs). It is a historical technique, originally exploited by fax machine and later adopted in image processing. The algorithm is quite easy: each run, instead of being represented explicitly, is translated by the encoding algorithm in a pair (l, v) where l is the length of the run and v is the value of the run elements.

The longer the run in the sequence to be compressed, the better is the compression ratio. All the smaller sequences that may result in expansion of data are kept out of RLE. The single zeros/ones, double zeros/ones that contribute in expanding data are ignored. Such sequences are left untouched and run length encoding scheme is not applied on them. The v in (l ,v) or value of a run is not a single zero or one, as a single zero or one will be mixed with the ignored sequences. The value of v is taken as the smallest consecutive sequence that is considered for RLE. Consider a data 444444555566666. If the modified scheme is applied and defines the data which is repeated as a function, the output comes out to be (4, 6), (5, 4), (6, 5). The value of a run is defined to send efficiently. Before compression it needs 15 bytes, but after compression the total bytes required to transmit is 6 bytes only. So the total file size is to be reduced from 15 to 9 bytes and need less power to transmit the data.

II. Run Length Encoding Methods

Run-length encoding (RLE) is a very simple form of data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. This is most useful on data that contains many such runs: for example, simple graphic images such as icons, line drawings, and animations. It is not useful with files that don't have many runs as it could greatly increase the file size.

RLE may also be used to refer to an early graphics file format supported by CompuServe for compressing black and white images, but was widely supplanted by their later Graphics Interchange Format. RLE also refers to a little-used image format in Windows 3.x, with the extension 'rle', which is a Run Length Encoded Bitmap, used to compress the Windows 3.x startup screen.

Typical applications of this encoding are when the source information comprises long substrings of the same character or binary digit.

All the smaller sequences that may result in expansion of data are kept out of RLE. The single zeros/ones, double zeros/ones that contribute in expanding data are ignored. Such sequences are left untouched and run length encoding scheme is not applied on them. The v in (l,v) or value of a run is not a single zero or one as a single zero or one will be mixed with the ignored sequences. We take the value of v as the smallest consecutive sequence that is considered for RLE. Consider a data 444444555566666. If we apply modified scheme and define the data which is repeated as a function. The output comes out to be (4, 6), (5, 4), (6, 5). The value of a run is defined to send efficiently.

A. *Lossless compression method*

Most lossless compression programs do two things in sequence: the first step generates a statistical model for the input data, and the second step uses this model to map input data to bit sequences in such a way that "probable" (e.g. frequently encountered) data will produce shorter output than "improbable" data.

The primary encoding algorithms used to produce bit sequences are Huffman coding (also used by DEFLATE) and arithmetic coding. Arithmetic coding achieves compression rates close to the best possible for a particular statistical model, which is given by the information entropy, whereas Huffman compression is simpler and faster but produces poor results for models that deal with symbol probabilities.

There are two primary ways of constructing statistical models: in a static model, the data is analyzed and a model is constructed, then this model is stored with the compressed data. This approach is simple and modular, but has the disadvantage that the model itself can be expensive to store, and also that it forces using a single model for all data being compressed, and so performs poorly on files that contain heterogeneous data. Adaptive models dynamically update the model as the data is compressed. Both the encoder and decoder begin with a trivial model, yielding poor compression of initial data, but as they learn more about the data, performance improves. Most popular types of compression used in practice now use adaptive coders.

Lossless compression methods may be categorized according to the type of data they are designed to compress. While, in principle, any general-purpose lossless compression algorithm (general-purpose meaning that they can accept any bit string) can be used on any type of data, many are unable to achieve significant compression on data that are not of the form for which they were designed to compress. Many of the lossless compression techniques used for text also work reasonably well for indexed images.

B. *Lossy compression method*

Lossy data compression has of course a strong negative connotation and sometimes it is doubted quite emotionally that it is at all applicable in medical imaging. In Transform encoding one performs for each image, run a mathematical transformation that is similar to the Fourier Transform thus separating image information on gradual spatial variation of brightness (regions of essentially constant brightness) from information with faster variation of brightness at edges of the image (compare: the grouping by the editor of news according to the classes of contents). In the next step, the information on slower changes is transmitted essentially lossless (compare: careful reading of highly relevant pages in the newspaper), but information on faster local changes is communicated with lower accuracy (compare: looking only at the large headings on the less relevant pages). In image data reduction, this second step is called quantization. Since this quantization step cannot be reversed when decompressing the data, the overall compression is 'lossy' or 'irreversible'.

III. HDL Implementation

"A CODEC (compressor/decompression) is used carry out the algorithm to save a file in a compressed format and open a compressed file. CODECs can be implemented in either hardware or software. Hardware CODECs are more expensive but, because they use dedicated chips instead of the computers CPU time, they are significantly more efficient.

Lossy and lossless techniques use different algorithms to help achieve good quality graphics with smaller file sizes. The technique that is chosen depends on the type of graphic working with.

As with lossless algorithms, lossy algorithms also look for repeat pixel values and assign less data to common values. If a photo of a sunset over the sea is taken, for example there are going to be groups of pixels with the same color value, which can be reduced. Lossy algorithms tend to be more complex, as a result they achieve better results for bitmaps and can accommodate for the loss of data. The compressed file is an estimation of the original data.

Other advantage of lossy compression is that it can reduce file sizes much more than lossless compression and you can determine how much compression can be applied. For example JPEG is a lossy format and GIF is lossless. With JPEG you can determine how much compression is applied, whereas with GIF you can't, it's set for you. Also JPEG can compress files much more than GIFs - JPEGs can reduce a file to up to 5% of its original size.

One of the disadvantages of lossy compression is that if the compressed file keeps being compressed, then the quality will degraded drastically. The diagram alongside illustrates this process.

Most loss less compression algorithms allow for variable levels of quality (compression) and as these levels are increased, in loss less data compression technique the data is to be count by comparing its previous binary values, if it is equal then the count will increase by one and finally file size is reduced. At the highest compression levels, image deterioration becomes noticeable.

A. *Compression Block*

In this compression block the input data is to give to the "INPUT FIFO BLOCK". This block takes the input data in first in first out sequence and then it is applied to the "COMPRESSION BLOCK". This compression block compresses the input digital data which can decrease the total size of the input. Decreasing of the input data is done by compressing the given data in which the bits that has a difference one to each and to eliminate the one bit. After compression the compressed data will go to "OUTPUT FIFO BLOCK". This block will give the reduced compressed output data. This compressed data is then applied to the "DECOMPRESSOR BLOCK".

In "RLE Compressor block" can compress the input data. In initializing the flow chart to start and consider the register 1 and register 2.The input byte is to be read when it the reset signal is enable it starts to read the data and then write it. If next input will come it will store the next FIFO address and then to decrease the count and the final output is decompressed like reverse order of compression method. When the total address is filling it is not possible to store the next input so that the out is "Data full". When the data full that input data is to be carry forward then the FIFO is in "*Idle*" state then the next input will store.

When the second bit enters into FIFO block it compare it's pervious bit value, if both bits are having same value then t then he count increase by one .Example: If 6 bytes are came continuously by 8 times then it will represent as like a function (6, 8),if 4 bytes are repeated for 5 times then it will represent as (4,5).by using this functional logic the continues data is to be count by its number of times then the compressed data stores in output FIFO. In this block is modified output data which gets compressed file which reduces the file size.

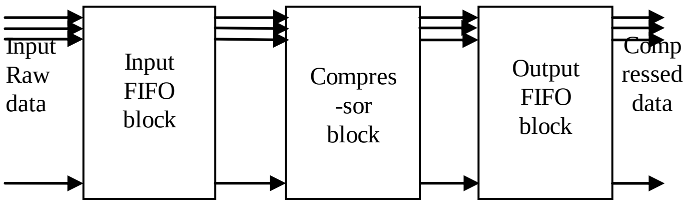

Fig 1: Compressor Block diagram

B. *Decompression Block*

In the "De-Compressor block" the input which takes from the "Compressor block"is applied to the "Input FIFO". In "RLE decompression block" can extract the compressed output.. In decompression of the compressed data flows to start and consider the register 1 and register 2.The input byte is to be read when it the reset signal is enable it starts to read the data and then write it. If next input will come it will store the next FIFO address and then to decrease the count and the final output is decompressed. When the total address is filling then it is in ideal state, then at the out of the output FIFO to get the original input.

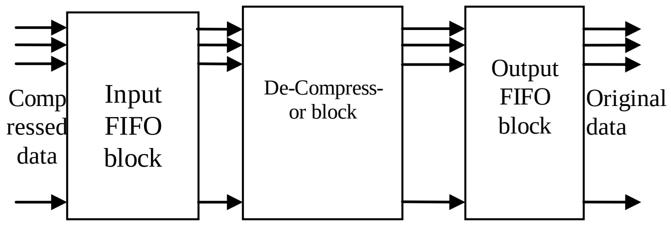

Fig 2: De-Compressor Block diagram

In this block, the sequential input decompresses the input in "De-Compressor Block" and that data which decompressed is to given the "Output FIFO". When the first input is entering into the decompression block then input values are decompress by basing of its functional value. if the compressed input is (6, 8),that means 6 is repeated by 8 times, so that the out of decompressed block will is to print 6 for 8 times by comparing its functional value and to decrease the count by using counter signal and the remaining functional values are also be decompress as same passion

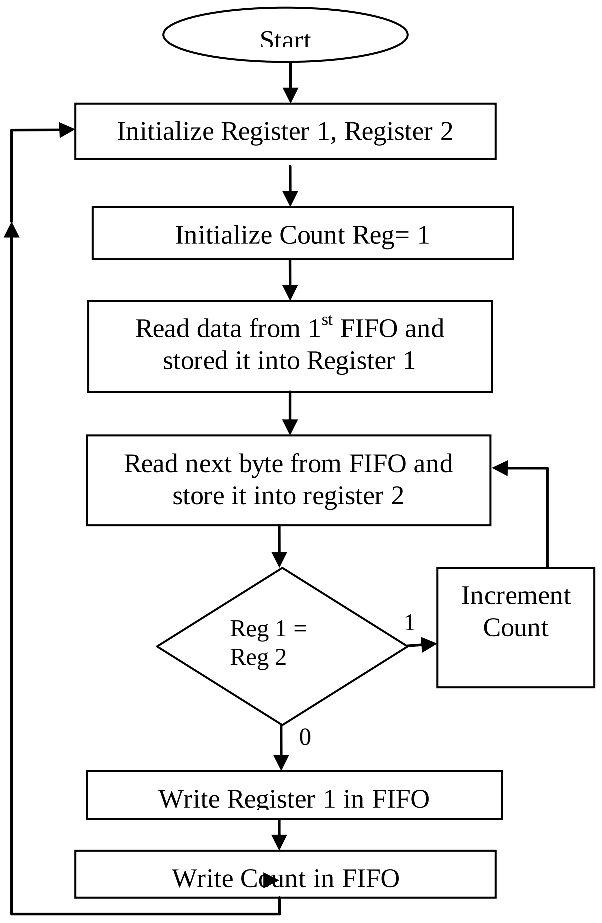
Start
Initialize Register 1, Register 2
Initialize Count Reg= 1
Read data from 1st FIFO and stored it into Register 1
Read next byte from FIFO and store it into register 2
Reg 1 = Reg 2
1
Increment Count
0
Write Register 1 in FIFO
Write Count in FIFO

Fig 3: Compressor flow chart

IV. Simulation Results

The simulation results are shown in the below. The input data is given to the compressor. The data is given below. So that the data is given to the compressor then it will gives the compressed data.

Fig 4: Compressor Simulation Results

The compressed output data is taken from the compressor and it is shown below. The data is taken from the input ROM and the data is given to the compressor then compressed data is available. By using compressor we can save the memory and data sending process is very easy.

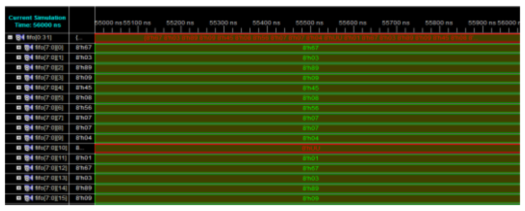

Fig 5: Simulation results showing Compressed data on output FIFO

The received compressed data from the receiver we should retrieve the actual data. So we are using the de-compressor technique. The received data is given to the de-compressor from the ROM; it will give the actual data and can be visible or readable. The input of the de-compressor Figure 5 is as shown below.

Fig 5: De-Compressor Simulation results

The decompressed data coming from the decompression is exactly equal to the original data that is supplied at the compressor input terminal. This decompressed data is stored in output FIFO for further storage or processing purpose.

V. Conclusion

Run-length encoding performs lossless data compression and it is well suited to the digital data which is having more redundancy. This loss less data compression technique is very useful for efficient data transmission in less duration of time. So it takes less power conception and reduces the time delay. These Common formats for run-length encoded data include True vision TGA, Pack Bits, PCX and ILBM. Run-length encoding is used in fax machines (combined with other techniques into Modified Huffman coding). It is relatively efficient because most faxed documents are mostly white space, with occasional interruptions of black. This technique is very useful to transmit the data having redundancy but this technique is fails to transmit the which is having less redundancy, it requires to send more compressed input bits than original data. This is the main limitation of this method.

References

[1] 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
[2] James A. Storer, "Data Compression Methods and Theory,"Computer Science Press, 1988, 413 pp, ISBN-10: 0716781565
[3] 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 Vol. 2, Issue 6, November- December 2012, pp.1654-1659

[4] Salomon, David (2008). A Concise Introduction to Data Compression. Berlin: Springer. ISBN 9781848000728.

[5] Navqi, Saud; Naqvi, R.; Riaz, R.A.; Siddiqui, F. (April 2011). "Optimized RTL design and implementation of LZW algorithm for high bandwidth applications". Electrical Review 2011 (4): 279–285.

[6] Mahmud, Salauddin (March 2012). "An Improved Data Compression Method for General Data". International Journal of Scientific & Engineering Research 3 (3): 2. Retrieved 6 March 2013

[7] D.A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", Proceedings of the I.R.E., September 1952, pp 1098–1102.

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 16.3, pp. 385–392.

[9] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. Managing Gigabytes. New York: Van Nostrand Reinhold, 1994. ISBN 978-0-442-01863-4.

[10] Burrows, Michael; Wheeler, David J. (1994), A block sorting lossless data compression algorithm, Technical Report 124, Digital Equipment Corporation

[11] Gordon Cormack and Nigel Horspool, "Data Compression using Dynamic Markov Modelling", Computer Journal 30:6 (December 1987)

[12] Cleary, J.; Witten, I. (April 1984). "Data Compression Using Adaptive Coding and Partial String Matching". IEEE Trans. Commun. 32 (4): 396–402. doi:10.1109/TCOM.1984

M.VidyaSagar received his B.Tech degree in Electronics and Communication Engineering from Akula Sree Ramulu College of Engineering affiliated to JNTU, Kakinada in 2011.He is currently doing his M.Tech P.G degree in VLSI ES from Amrita Sai Institute of technology Affiliated by JNTU Kakinada,Paritala,Vijayavada,A.P. His Research interests include studies in RTL Design Using VHDL, Verilog and verification using System Verilog.

J.S. Rose Victor received her Master of Technology (M.Tech) in DSCE from the Jawaharlal Nehru Technological University, Anantapur in 2003. She is currently Associate Professor in the Department of Electronics & Communication Engineering, Amrita Sai Institute of Science and Technology, ,Paritala,Vijayavada, Affiliated to JNTU Kakinada, A.P. Her research interests include Signal Processing.