International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 02 Issue: 04 | July-2015 www.irjet.net p-ISSN: 2395-0072

An Optimum Algorithm for Data Compression using VHDL

Mr. Pralhad Sarkar[1], Prof. Prashant Indurkar[2], Prof. Ravindra Kadam[3]

[1]*M.TechStudent, BDCE, Sewagram, India*
[2]*Associate professor,BDCE, Sewagram, India.*
[3]*Assistant professor,BDCE, Sewagram, India.*

Abstract: -This paper describes a method of data compression for FPGA systems called by us GCC (Golomb Compressed Code algorithm). This method is widely used for lossless Data compression due to its lower complexity in encoding & decoding methods. The main objective of data compression is to find out the redundancy and eliminate them through Golomb algorithm, so that the data which is reduced require less memory as well as the size of the data decreases hence the cost of transmission is also reduce. This method gives Lossless data compression recreates the exact original data from the compressed data while lossy data compression cannot regenerate the perfect original data from the compressed data. Our method reduces code size up to 38.63% (including different code word size).In order to prove its validity, the developed algorithm is simulated using the Modelsim Altera Starter Edition 6.4a.

Keywords – GCC; CC-MLD; pattern blocks; encrypt; decrypt

INTRODUCTION

With the increase in the requirement of online real time data, data compression algorithms are to be implemented for increasing throughput. Compression is the art of representing information in a compact form rather than its original or uncompressed form. The compression code is place in main memory and/or instruction cache memory, thereby increasing the number of stored instruction, then increasing the cache hit rate and decreasing the search into the main memory, thus increasing the system performance and reducing power consumption. [4] The original file which is to be compressed is first coded which is then known as encrypted file. For any efficient compression algorithm file size must be less than the original file. To get back the original file we need to 'decrypt' the encoded file. Data compression methods are sometime very difficult as it require hardware for its implementation ants of maintains.

The compressed code is placed in main memory and/or instruction cache memory, thereby increasing the number of stored instructions, then increasing the cache hit rate and decreasing the search into the main memory, thus increasing system performance and reducing energy consumption [6]. Thus, during the program execution, the compressed code is taken to decompression and sent to the next level of memory or directly to processor.[1] The compression rate (CR) is widely accepted to measure efficiency of a compression method and it is defined according to(1) .

$$CR = \left(\frac{\text{Original code size–Compressed code size}}{\text{original code size}} x\ 100\right)\% \quad (1)$$

Fig. 1 shows an overview of code compression method using GCC algorithm. These algorithms used to encrypt works properly; there should be a significant difference between the original file and the compressed file. When data compression is used in a data transmission application, speed is the primary goal.[5] Speed of transmission depends upon the number of bits sent, the time required for the encoder to generate the coded message and the time required for the decoder to recover the original ensemble.

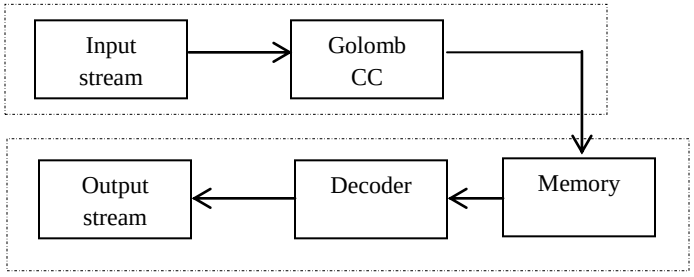


Fig.1 Overview of Golomb CC algorithm

In this paper, a detail study of Golomb coding algorithms for test vector compression and decompression is presented. In order to have simplicity in development and testing, the Golomb coding parameter m is set to 2 [7].The goal of this work was to increase the compression ratio as high as possible without any loss in the original data.

RELATED WORK

Author Wander Roger Azervedo Dias, Edward David Moreno and Issanc Palmeria give a new method of code compression

for embedded systems by them as CC-MLD (Compressed Code using Huffman-Based Multi-Level Dictionary). It applies two compression techniques and it uses the Huffman code compression algorithms. A single dictionary is divided into two levels and it is shared by both techniques. They performed simulations using application from MiBench and they had used four embedded processor (ARM, MIPS, PowerPC and SPARC). Their method reduces code size up to 30.6% (including all extra costs for these four platforms).[1] They had implemented the decompressor using VHDL and FPGA and they had obtained one clock from decompression process.

Author Ivan Scherbakov, Christian Weis and Norbert When had describe a design and develop a data compression engine on a single FPGA chip that is used as part as part of text-classification application. The implementation of the prediction by partial matching algorithm and arithmetic coding data compression is totally in hardware as well as in software code. Their design implements a dynamic data structure to store the symbol frequency counts up to maximal order of 2. The computation of the tag-interval that encodes the data sequence in arithmetic coding is done in parallel architecture that achieves a high speed up factor. Even with a relatively slow 50MHz clock their hardware engine performs more than 70 times faster than a software based implementation in C on a CPU running on a 3 Ghz clock. [3]

Author Joel Ratsaby and Vadim Sirota had presented a flexible high-performance implementation of the LZSS compression algorithm capable of processing up to 50 MB/s on a Virtex-5 FPGA chip. They exploit the independently addressable dual-port block RAMs inside the FPGA chip to achieve an average performance of 2 clock cycles per byte. To make the compressed stream compatible with the ZLib library they encode the LZSS algorithm output using a fixed Huffman table defined by the Deflate specification. They also demonstrate how changing the amount of memory allocated to various internal tables impacts the performance and compression ratio. [2] They provide a cycle-accurate estimation tool that allows finding a trade-off between FPGA resource utilization, compression ratio and performance for a specific data sample.

Author Vijay G. Savani, Piyush M. Bhatasana describes the methods of creating dedicated hardware which can receive uncompressed data as input and transmit compressed data at the output terminal. This method uses FPGA for the same, wherein the hardware part has been created using Xilinx Embedded Development Kit (EDK) and data compression algorithms have then been implemented on the same hardware. The EDK helps creating a Soft Core Processor on the FPGA with desired specifications. The data compression algorithm can be implemented on this processor. The advantage of this kind of a system is that, without changing the hardware, the FPGA can be reprogrammed with a new algorithm whenever a better technique is discovered. For the proof of concept the Huffman coding technique has been implemented. The Soft Core Processor uses serial port and for direct input the GPIO of the processor were used. The user enters text data through this port, and the soft core processor using Huffman's data compression algorithm gives compressed data as the output [4].

Author Arohi Agarwal and V.S.Kulkarni had discussed about Data transmission, storage and processing are very necessary nowadays. Data can be represented in compact form using data compression for transmitting and storing a huge volume of data required large space which is an issue. In order to transmit and store such a large volume of data it requires large memory space and large bandwidth avability. Because of which the hardware increases as well as cost increases. Hence to solve this it is necessary to reduce the size of the data which is to be transmitted without any information loss .For this purpose they have taken the following algorithm. LZMA is a lossless dictionary based algorithm which is used in 7zip was proving to be effective in unknown byte stream compression for reliable lossless data compression. Here the algorithm LZMA is implemented on SPARTAN 3E FPGA to design both the encoder and the decoder which reduces the circuit size and its cost [8].

METHODOLOGY – GOLOMB CC

The details regarding Golomb Coding basic background information is described. Golomb coding uses a tunable

parameter *m* to divide an input value into two parts: q, the result of a division by *m*, and r, the remainder. The quotient in unary code followed by the remainder in truncated binary encoding. W Golomb coding is equivalent to unary coding. In Golomb Coding, the group size, m, defines the code structure. Thus, choosing the *m* parameter decides variable length code structure which will have direct impact on the compression efficiency [9].

After finalization of parameter m, a table which maps the runs of zeros or ones is created [7]. A Run length of multiples of *m* are grouped into *Ak* and given the same prefix, which is (k – 1) number of one's followed by a zero, which can also termed as quotient and can be represented in the form of unary codes. A tail is given for each member of the group, which is the binary representation of log2m bits.[5] The other term for tail is Remainder of the division of run length by m. The codeword is then produced by combining the prefix and the tail.

In order to avoid this problem, the algorithm must be capable of detecting the end of data and if the last bit is a '0' then additional '1' must be added during the encoding process and at the time of decompressing the encoded data, this extra appended 1 should be removed by the decoder.

I. ALGORITHM

1. Fix the parameter *M* to an integer value.
2. For *N*, the number to be encoded, find
   a. quotient = *q* = int[*N*/*m*]
   b. Remainder = *r* = *N* modulo *m*.
3. Generate Codeword

The Code format: <Quotient Code><Remainder Code>,

A. WhereQuotient Code (in unary coding)

   i. Write a *q*-length string of 1 bits
   ii. Write a 0 bit.

B. Remainder Code (in truncated binary encoding)

   I. If *M* is power of 2, code remainder as binary format. So log2(m) bits are needed.
   II. If *M* is not a power of 2, set b= log2(m).
      a. If r < 2b - m code *r* as plain binary using *b*-1 bits.
      b. If r>= 2b - m code the number r + 2b–m in plain binary representation using b bits.

TABLE I. EXAMPLE OF GOLOMB CODING

Numbers	Divisor	Quotient	Remainder	Code
5	4	1	1	0101
10	4	2	2	00110
15	4	3	3	000111
61	8	7	5	00000001101
64	8	8	0	000000001000

In Golomb coding, we code an integer (m), by the quotient (q) and remainder (r) of division by the divisor (d). We write the quotient bi=dc in unary notation and the reminder i mod d in binary notation [10]. We need a stop bit after the quotient. We can use 1 as stop bit if the quotient is written as 0 to represent the unary form. In case of Rice coding, we use the divisor as a power of 2. For example if we are coding a number 15 with divisor 4, the code will be 000111 (See TABLE I). Golomb-Rice coding will achieve good compression ratio. This algorithm is applied though VHDL to achieve the GCC.

II. SIMULATION

Fig. 2 shows the simulation of Golomb code where input stream is applied to GCC with the clock having 50 duty cycles, it gives the compress data as shown in Table II.

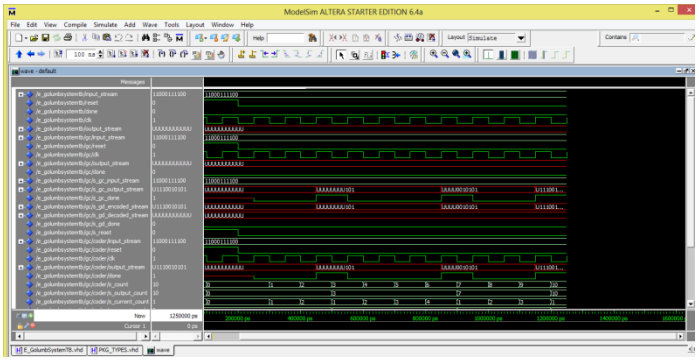


Fig. 2 Simulation wave form

III. RESULT and COMPRESSION TABLE

Table II gives the compression between the CC-MLD[1] and GCC method. The golomb code is synthesis on Modelsim Altera Starter Edition 6.4a.The simulation were performed with unitary code and patterns block (since PB- is "pattern blocks" with different pattern)

TABLE II. COMPRESSION TABLE

	Word size			
	128	256	512	1024
Unitary CCMLD	15.7%	20.5%	20.5%	27.2%
Unitary GCC	25%	27.77%	32.5%	38.63%
PB-2 CCMLD	8.3%	10.8%	12.5%	7.9%
PB-2 GCC	25%	22.22%	30%	18.18%
PB-3 CCMLD	4.6%	5.8%	3.4%	-8.1%
PB-3 GCC	12.5%	44.44%	40%	9.09%
PB-4 CCMLD	3.1%	3.3%	-3.8%	-19.3%
PB-4 GCC	25%	33.33%	18.18%	9.09%

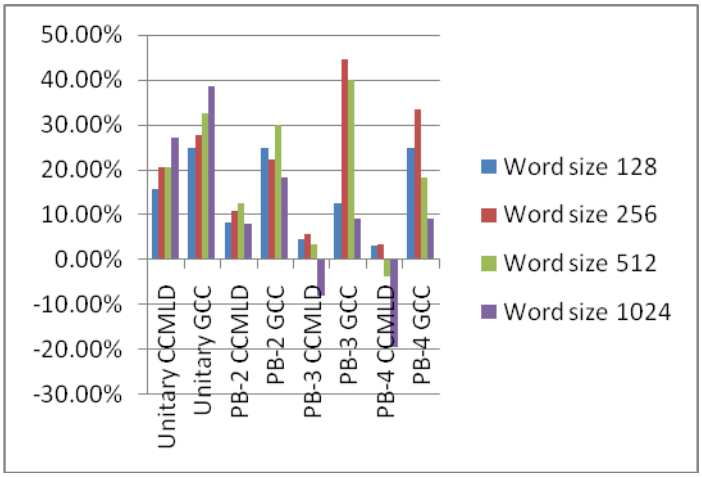
Table III. shows the overall comparison of 512 and 1024 bits word size level. In which the compression of GCC is more continent and the compression rate is better.

TABLE III. OVERALL AVERAGE COMPRESSION RATE

Algorithm	Level	
	512	1024
CC-MLD	28.3%	30.6%
GCC	32.5%	38.63%

Golomb compress code algorithm can perform 'n' bit compression depending upon the size of m (length of input stream). Fig. 3 shows the compression graph with various PB. Since the length of the sample to be read is not known when the decoder loads data to its registers, it need to load packages of data of a fixed size. The package size must be equal or greater than the maximum bit length of the encoded data. As mentioned, we assume that the maximum length is 16 bits, and that the decoder receives packages of this size. Further, the register size in the decoder must be 3 times larger than this, resulting in 48 bits, in order to avoid buffer under run.

FIG. .3 INPUT BIT STREAM COMPARISON.



V. CONCULSION

In this paper we present an optimum code compression amethod (GCC – Golomb Compress Code) that uses a technique which is based on Golomb Algorithm. The compression method is implemented in VHDL. Through the simulation we can see that the method GCC reached, on average compression rate up to 38.6%. Also this VHDL based method is 70% faster than C based compression. Thus we conclude that our method present in this paper, is an efficient and can be used in FPGA based system which is give a well compression ratio. As a future work different compression code algorithm can be simulate using VHDL.

REFERENCES

[1] Wander Roger Azevedo Dias, Edward David Moreno, Issanc Palmeria, " A New code compression Algorithm and its Decompressor in FPGA – Based Hardware" IEEE, 2013.

[2] Joel Ratsaby, vadim Sirota, " FPGA – based data compression based on Prediction by Partial Matching, IEEE, 2012.

[3] A High-Performance FPGA-Based Implementation of the LZSS Compression Algorithm by Ivan Shcherbakov, Christian Weis, and Norbert Wehn, 2012 IEEE.

[4] Implementation of Data compression Algorithm on FPGA using soft core processor by Vijay G. Savani , Piyush M. Bhatasana and Akash Mecwan, 2012 IJICT.

[5] Data compression Methodologies for Lossless Data and Compression between Algorithms, 2013 IJESIT.

[6] W. R. A. Dias, and E. D. Moreno, "Code Compression in ARM Embedded Systems using Multiple Dictionaries". In Proc. of 15th IEEE CSE 2012, Paphos, Cyprus, pages 209-214, December 2012.

[7] G. H. H'ng, M. F. M. Salleh and Z. A. Halim," Golomb Coding Implementation in FPGA", School of Electrical and Electronics Engineering, Universiti Sains Malaysia,Seri Ampangan,14300 Nibon Tebal, Pulau Pinag, Malaysia. VOL. 10, NO. 2, 2008, 36-40.

[8] "FPGA based implementation of Data compression using Dicitionary based 'LZMA' Algorithm" by Arohi Agrawal, V.S.Kulkarni, IRF international Conference.

[9] Hong-Sik Kim, Joohong Lee, Hyunjin Kim, Sungh Kang, and Woo Chan Park, A Lossless Color Image Compression Architecture using a Parallel Golomb- Rice Hardware CODEC, IEEE transactions on circuits and systems for video Technology, vol. 21, no. 11, November 2011.

[10] J. Zhang, X. Long, and T. Suel. Performance of compressed inverted list caching in search engines. pages 387–396, 2008

