A High Performance And Area Efficient Golomb Coding For H.264 Entropy Encoder In FPGA

Jaikaran Singh, Khyati Borad, Mukesh Tiwari, Sanjay Rathod

Abstract- In this paper we present a new implementation method of golomb coading. Due to lower complexity this techniques is well known for data compression . Thus it is mainly used in mobile multimedia communication . in this paper we present the golomb coding compression algorithm in field programmable gate array (FPGA) . The proposed coding is implemented to reduce the hardware cost . The coding scheme development utilises the VHDL . The development algorithm is simulated using the ALTERA Quartus II software

Keywords- FPGA , VHDL , ALTERA Quartus II, Exp-Golomb.

1. INTRODUCTION

The speedy extensive rowth of digital technologies such as internet access ,digital television, video and video calls have increased the demand for high storage and transmission capacity in order to fit the growing needs [1]. This has called upon for the need of effectual data compression techniques, where the original data is compressed into a smaller data size. This process also reduces the transmission bandwidth needed for data transfer. [2]. Golomb coding is the best lossless data compression techniques for h.264 entropy encoder. It is capable of compressing larger sized data into a smaller sized data and also allowed the encoded data to be reconstructed back after decompression techniques. Moreover there is another high performance lossless compression algorithm such as arithmetic coding [4] etc. Though, this algorithm has higher design complexity. One another way of compression technique is lossy compression. In lossy data compression, the decoded data loses some of the information, So consequence a lower quality data. Golomb coding has been used in the latest H.264 video standard [5] as part of its entropy coding. This is presente the hardware implementation of H.264 video codec baseline profile [6]. It's contains the hardware implementation of the Exp-Golomb coding for its entropy coding of h.264 entropy encoder. Another Golomb coding applications are the use in system-on-chip (SoC) test-data compression system as presented in [7]. In [8] Jung and Chong also use Golomb coding for SoC test data compression, where the new algorithm developed can reduce the scan-in power and test data volume. In this paper, the development of Golomb coding algorithms for data compression and their implementation in Field Programmable Gate Array (FPGA) is presented. The algorithm is developed using the ALTERA Quartus II software [9]. The remainder of the paper is organised as follows[10]. Section II presents the details of Golomb Coding and the basic compression method. Section III presents the modified version of the Golomb coding compression methods in practical FPGA implementation. Section iv represents the simulation results of the golomb algorithms generated in order to prove their validity. Lastly, Section v concludes the paper.

2. BACKGROUND

This section covers the details background information regarding Golomb Coding.In entropy coading of H.264 the data are coading using either context-adaptive variable length coding (CAVLC) or using Exp-Golomb codes depending on the entropy encoding mode.

When entropy coding mode is set to 0 then residual block data is coded by a context-adaptive variable length coding (CAVLC) scheme and When entropy coding mode is set to 1 then variable-length coded units are coded by Exp-Golomb codes.

TABLE:1 Exp-Golomb codewords

Code_ num	Codeword
0	1
1	010
2	011
3	00100
4	00101
5	00110
6	00111
7	0001000
8	0001001
…	…

Exp-Golomb codes (Exponential Golomb codes) are variable length codes with a regular construction. Table 1 lists the first 9 codewords .It is clear from examining the first few codewords from Table-1 that they are constructed in a logical way as follows :

[M zeros][1][INFO]

Where the INFO is an *M*-bit field carrying information.As shown in table-1, the first codeword has no leading zero or trailing INFO and Codewords 1 and 2 have a single-bit INFO field and codewords 3–6 have a two-bit INFO field and so on. The length of each Exp-Golomb codeword is (2*M* + 1) bits.

Each codeword can be constructed by the encoder based on its index *code_num*:

M = floor(log2[code_num + 1])
INFO = code num + 1 – 2^M

A Decoder can be decode codeword as follows:
i. Read in M leading zeros followed by 1.
ii. Read M-bit INFO field.
iii. code_num = 2^M + INFO – 1
[For codeword 0: INFO and M are zero.]

A parameter *h* to be encoded is mapped to code_num in one of the following ways:
ue(h) : Unsigned direct mapping, code_num = h. Used for reference frame index , macroblock type and others.

se(h) : Signed mapping, used for motion vector difference, delta QP and others. h is mapped to code_num as follows (Table-2).

code num = 2|h| (h ≤ 0)
code num = 2|h|−1 (h> 0)

TABLE-2 Signed mapping se(h)

h	code_num
0	0
1	1
−1	2
2	3
−2	4
3	5
.....	.....

me(h) : Mapped symbols; parameter h is mapped to code_num according to a table specified in the standard parameter. This mapping is used for the coded_block_pattern parameter as shown in Table 3.As Table 3 lists a small part of the table for Inter predicted macroblocks , coded_block_pattern indicates which 8x8 blocks in a macroblock have non-zero coefficients. Each of these mappings (ue(h), se(h) and me(h)) are designed to produce short codewords for repeated values and longer codewords for less common parameter values.

For example, inter macroblock type P_L0_16 × 16 (prediction of 16 × 16 luma partition from a previous picture) is assigned code num 0 because it occurs frequently; macroblock type P_8× 8 (prediction of 8 × 8 luma partition from a previous picture) is assigned code_num 3 because it occurs less frequently; the commonly-occurring motion vector difference (MVD) value of 0 maps to code_num 0 whereas the less-common MVD= −3 maps to code_num 6.

TABLE 3 Part of coded block pattern table

coded block pattern (Inter prediction)	code_num
0 (no non-zero blocks)	0
16 (chroma DC block non-zero)	1
1 (top-left 8 × 8 luma block non-zero)	2
2 (top-right 8 × 8 luma block non-zero)	3
4 (lower-left 8 × 8 luma block non-zero)	4
8 (lower-right 8 × 8 luma block non-zero)	5
32 (chroma DC and AC blocks non-zero)	6
3 (top-left and top-right 8 × 8 luma blocks non-zero)	7
.....	.....

So using this four mapping we can calculate the value of code_num. And using the value of code_num we can find the value of M and then using the value of code_num and M we can calculate the value of INFO.

3. FPGA IMPLEMENTATION

The Golomb encoder algorithm shown in Figure 1 is a complete design for hardware implementation. After the system has been reset , the system will move into state 1 , which is remains ideal if enable signal is zero and goes to state 2 if enable signal is one. In state 2 the system is latch the value of code_num and calculate M= (log2[code_num+1]).Value of code_num is decided from the mapping of ue(h) , se(h) and And this value is decided using Table-1, Table-2 and Table-3. If calculation is completed the system is goes in to the state 3 and if the value of code_num is zero then the system goes direct to state 6. After calculating the value of M In state 3 the system is calculate 2^M for calculating the value of INFO in next state. In state 4 the system calculate the value of INFO

using the value of code_num and 2^M . After calculating the value of INFO and M the system goes to the state 5 where it concatenate M,1 and INFO and generate the encoded byte of code_num. After concatenate of M ,1 and INFO the system goes into the state 6 to cheak if total encode byte is reach or not. If total encode byte is reach then system generate the output and if total encoded byte does not reached the system further goes to state 2 to latch the code_num and then the system will then stop handling any process until a reset signal is given.

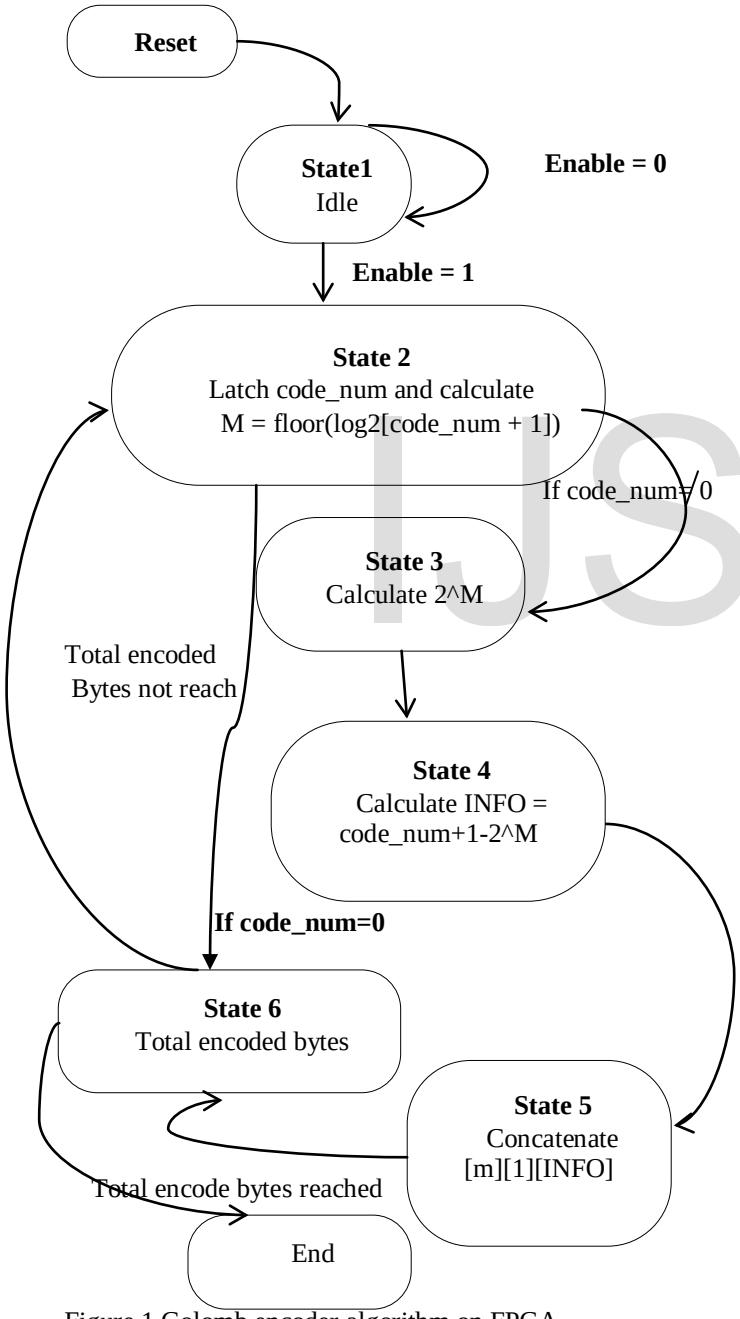

Figure 1 Golomb encoder algorithm on FPGA

4. RESULTS

The results for exponential Golomb encoder algorithm implementation on FPGA are simulated and verified for their validity. Firstly, the implementation of Golomb encoder shown in Figure 1 was simulated using ALTERA Quartus II software with 10 bytes sample data, 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08 and 0x09. The simulated result is shown in Figure 2 From the observation of the simulation diagram of Figure 2, input data signal of code_num will only be accepted by the system if *en_codenum* signal is asserted HIGH. Data will only be accepted when the system is not busy, in other words, the signal *reset* is asserted HIGH. This is to avoid the system from being flooded with data when the system is not ready for the next data. Output data signal of codew(data out) is only valid when the *valid* signal is asserted as HIGH. The expected output is shown in Figure 3 and the expected output is compared with the simulated output. It is shown that the simulated conforms to expected output, thus showing the design is valid

5. CONCLUSION

Through simulation of the Golomb Coding, it can be concluded that the algorithm of Golomb Coding on FPGA had been successfully developed. This had been proven by comparing the results generated using simulation of the Golomb Encoder with the expected results which showed result was identical.

The use of encoded data as the input data for Golomb Decoder managed to generate back the original data proved that the Golomb Decoder System may also successfully developed. Hence, the whole system of Golomb Coding was completed.

ACKNOWLEDGMENT

The authors are grateful to the RGPV for funding this work as a part of M.Tech thesis.

REFERENCES

[1] M. Ghanbari, *Video Coding an Introduction to Standard Codecs*. London: The Institution of Electrical Engineering, UK, 2003.

[2] T. Sikora, "Trends and perspectives in image and video coding," *Proceedings of IEEE*, vol. 93, no. 1, pp. 6-17, Jan 2005.

[3] S. W. Golomb, "Run Length Encodings," *IEEE Transactions on Information Theory,* vol. 12, pp. 399-401, 1966.

[4] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic Coding for Data Compression" *CACM Journal*, vol. 30, no. 6, pp.520–540, June 1987.

[5] J. Ostermann, J. Bormans, et al, "Video coding with H.264/AVC: Tools, Performance, and Complexity," *IEEE Circuits and System Magazine,* Vol. 4, No. 1, pp. 7-28, First Quarter 2004.

[6] T. Silva, et. al., FPGA based design of CAVLC and Exp-Golomb coders for H.264/AVC baseline entropy coding," *Proc 3rd IEEE Southern Conference on Programmable Logic*, pp. 161-166, Feb 2007.

[7] A. Chandra and K. Chakrabarty, "System on-a-chip test data compression and decompression architectures based on Golomb codes," *IEEE Trans. Computer-Aided Design*, vol. 20, pp.355-368, Mar.

2001.
[8] J. M. Jung, and J. W. Chong, "Efficient test data compression and low power scan testing in SoCs," *ETRI Journal*, vol. 25, no. 5, pp. 321- 327, Oct 2003.
[9] Quartus II development software literature, availableAt http://www.altera.com/literature/lit-qts.jsp.
[10] G. H. H'ng, M. F. M. Salleh and Z. A. Halim" Golomb Coding Implementation in FPGA" Faculty of Electrical Engineering Universiti Teknologi Malaysia , VOL. 10, NO. 2, 36-40, 2008 .
[11] White Paper: H.264 / AVC Context Adaptive Variable Length Coding Iain Richardson
[12] Iain E.G. Richardson , "H.264 and MPEG-4 Video Compression" John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England.

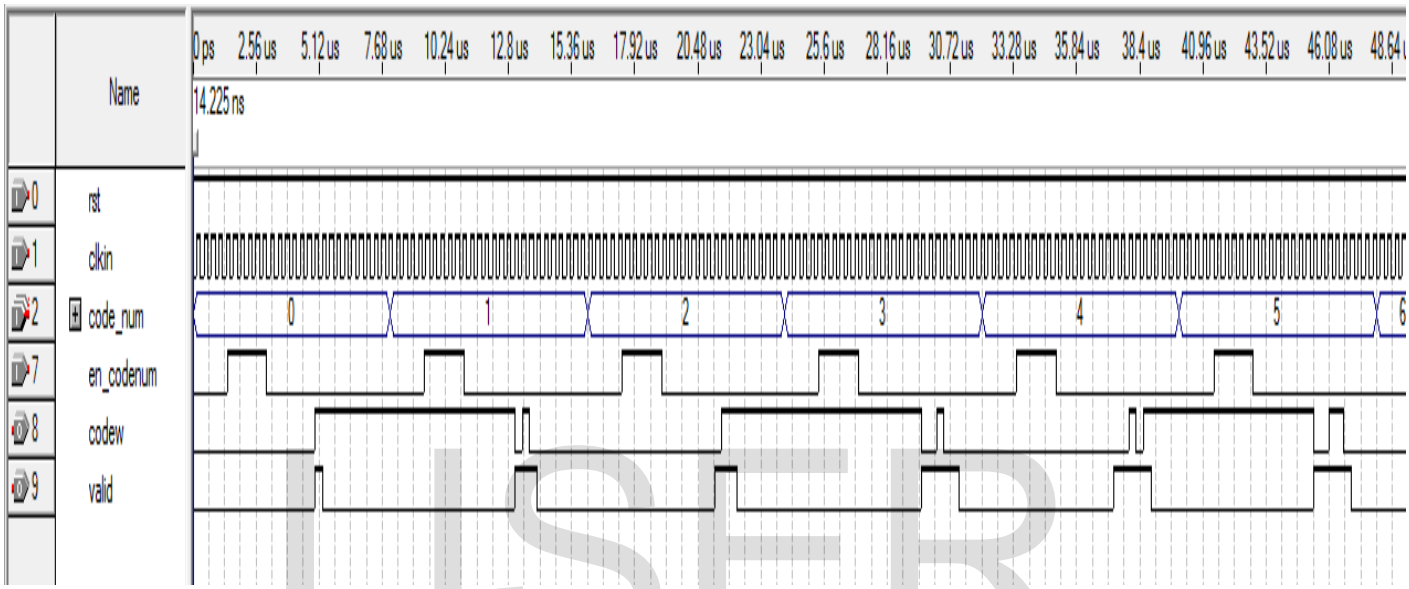

Figure 2 Simulation Result Of Exp-Golomb Encoder.