Efficient Data compression using variable length Huffman coding

Reva Joshi[1], G. Githika[2], Ch. Prudvi[3],Dr.SK.Fairooz[4], Shaik Mohammed Rafi[5]

[123] *UG Scholar , Sreyas Institute of Engineering & Technology, Hyderabad.*
[4] *Dr SK.Fairooz, Associate Professor , Dept of ECE, Sreyas Institute of Engineering & Technology, Hyderabad.*
[5] Shaik Mohammed Rafi, *Assistant Professor , Dept of ECE, Sreyas Institute of Engineering & Technology, Hyderabad.*

ABSTRACT : *In this paper, we are developing a new compression approach that relies on the diversity of the proposed approach and the high and low data blocks in Huffman encoding. The proposed curriculum in VHDL languages will be developed and tested for implementation via the targeted Xilinx FPGA. The proposed high-speed decoding system takes approximately 43.75 seconds to decode a 12x32 test dataset. The current tree-based Huffman decoder takes about 83.15µs to decode the same data set. The high-speed decoder implemented takes about 400 hours less than the current system.*

Keywords : *Huffman , Variable length , FPGA , Xilinx , compression.*

I. INTRODUCTION

Huffman encoding is a widely used lossless data compression algorithm that assigns shorter encoded words for more frequent symbols and longer encoded words for relatively rare symbols to reduce the size of the original information [1]. Huffman encoding is also known as optimal variable length encoding, and the algorithm provides the highest lossless compression ratio for compression methods that encode symbols separately [2]. VLSI implementation of Huffman encoders is an important ongoing topic, and several VLSI designs have been suggested in the literature. However, current Huffman encoding schemes require considerable time and computational resources to create the Huffman tree and generate encoded words, resulting in lower encoding speed and higher hardware costs [3]. In this article, we suggest a new data structure to build a Huffman tree, and the proposed data structure is used to develop a new algorithm to improve the efficiency of Huffman coding by creating a Huffman tree simultaneously with word generation encoded, where the width of the symbols does not affect the compression time. The proposed new data structure does not require storing the Huffman tree in fixed RAM, which increases the available memory resources. High-speed serial data encryption software is essential for many applications, such as image compression, data transmission, and data communications. Therefore, the proposed technological innovation is used in VLSI high-speed serial data encryption using Verilog's Hardware Description Language (HDL), and is simulated using the Modelsim Electronic Design Automation (EDA) tool. The simulation results show that the proposed encoding scheme provides more encoding speed and lower hardware cost compared to current Huffman parallel encoding applications [4] [5]. The proposed design requires an average of 3.28 cycles per clock to compress the 8-bit code, which is much lower than the reference [5]. An average of 7-clock cycles requires compression of the 8-bit code. The organization of the paper as section II is proposed architecture, section III as decoder , section IV as experimental results and section V as conclusions.

II. PROPOSED ARCHITECTURE

The high-speed decoder receives the serial input bitstreams from the encoder, and proceeds to the N-bit shifter where the serial data becomes parallel. This parallel data is passed to the decoder and the longitudinal decoder to decode the encoded words to the original data. The symbol decoders are provided with segmented notebooks. Symbol books are aligned as LUTs with the same symbol length. Each LUT contains all the code words of equal length and is shortened depending on their length.

Depending on the length of the code words, the current high-speed decoder scans the received code words using LUT. Since each LUT consists of only a specific length code, the scan time to trace the code is reduced.

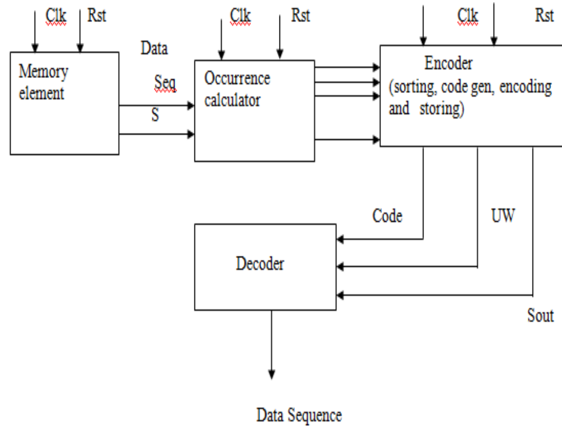

Fig.1 :Proposed for Encoder and Decoder

The method proposed here consists of the following units;

1. Input Memory Element,
2. Occurrence Calculator,
3. Code Generator
4. Encoder,
5. Decoder,
6. Output Memory Element.

II. *A) INPUT MEMORY ELEMENT*

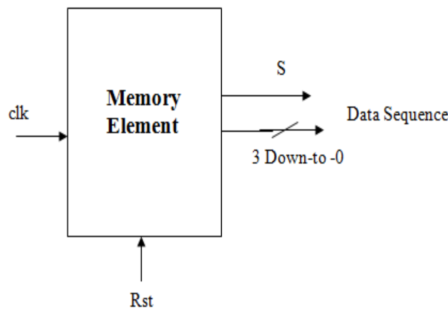

Fig 1: Input Memory Elements

The input memory component consists of twelve locations with a length of 32 bits each. This block stores data that is passed to the frequency calculator. Signal 'S' is the status signal, made '1' once the complete data is passed to the repeat calculator, clk is the clock signal that activates the presence calculator on the rising edge, "first" is the reset signal, which when high, initializes all memory locations to Zero External data is the output vector, which transfers data from memory to an occurrence calculating unit. The memory object transports all the data stored in it as 4-bit blocks to an iteration calculator, and each 4-bit block is individually called as a single word. Therefore, the number of unique words is sixteen (that is, 0000 to 1111).

II. *B) OCCURRENCE CALCULATOR*

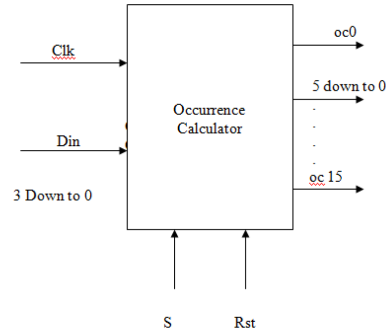

Fig 2: Occurrence Calculator unit

This block counts the number of unique word occurrences in the data. Generates a control signal to the encryption block with status S, to indicate that the recurrence calculation is complete, and enables encryption (when S = '1'). clk is the clock signal that activates the repeat calculator on the rising edge, it is reset to first, all repeat values are initialized to zero, and Sout (3 to 0) is the four-bit input data which are fed into the occurrence calculator from the input memory item, OC0 to OC15 are the occurrence values for single words.

II C) Code Generator:

This Block can be treated as the combination of the following Sub-Modules:

1. SUMMER
2. SORTER

1) SUMMER:

The summer is to calculate the sum of the two values fed and provides the sum of these two inputs. Summer is used in encryption to calculate the sum of the last two elements after each type.

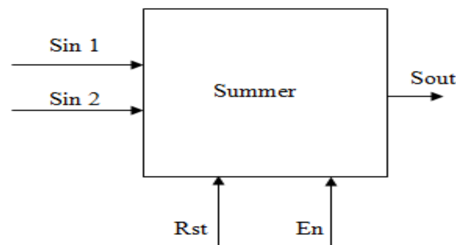

Fig3: Summer Module

2) SORTER:

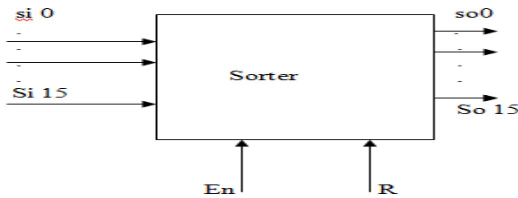

Fig 4:Sorter Module

The classifier contains si0, si1, ... si15 as input values taken from the frequency calculator and ordered in descending order, when the enable signal "on" is active high and the reset signal "activates first" it is low. The ordered values are passed to the code generator.

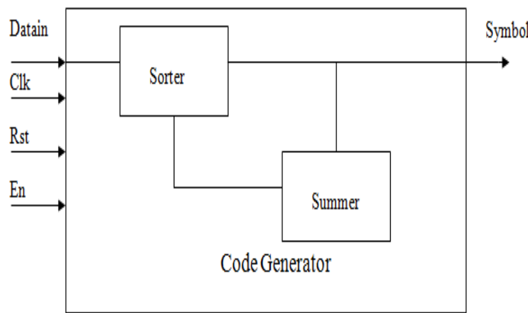

Fig 5: Code Generation unit

Huffman's code is created in this block. Perform the following process.

• Click replacing each unique word with an equivalent word

• Stores compressed data, after encryption, to be sent to the decoder.

III. DECODER CIRCUIT

This block receives the bits encoded as serial bits and is decoded. Please go back to get the original data and block consists of:

III. *A) COMPARATOR*

This part compares the compressed data received from the encoder with the predefined codewords stored in the LUT.

III. *B) LOOK UP TABLE (LUT)*

This block stores a predefined code sequence from which the comparator takes data and uses it to decrypt. This block gets values from the Encoder block.

IV. EXPERIMENTAL RESULTS

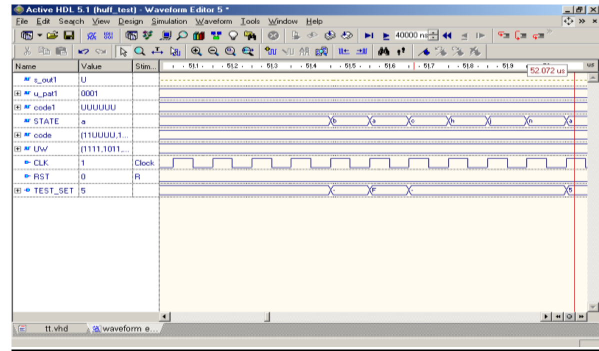

Fig 6: Simulation result showing decoding state transition and the decoded data set

The simulation result obtained for the current Huffman coding system implemented. The simulation result shown above shows that a decoding state transmission can be observed from the specified simulation result. A total of 14 decoded cases are observed for a full set of cipher data. The "Test_Set" flag shows the decoded data that the encoder has retrieved from the obtained encrypted data.

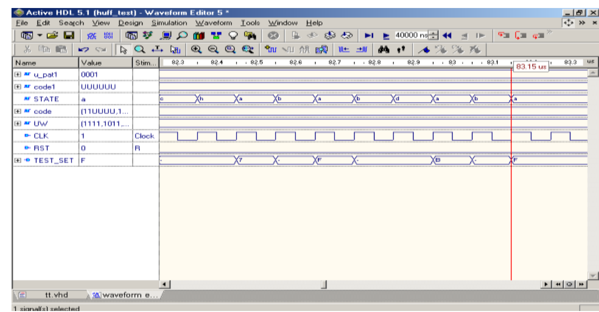

Fig 7 : Simulation result showing the final data block decoded at 83.15µs.

Figure 7 above illustrates the simulation result obtained for the current Hofmann coding system. From the simulation result it was observed that a total of 83.15 µm was taken to decrypt the entire set of encoded data. The "Test_set" flag shows the recovered decoded data set for the encoded data bits obtained.

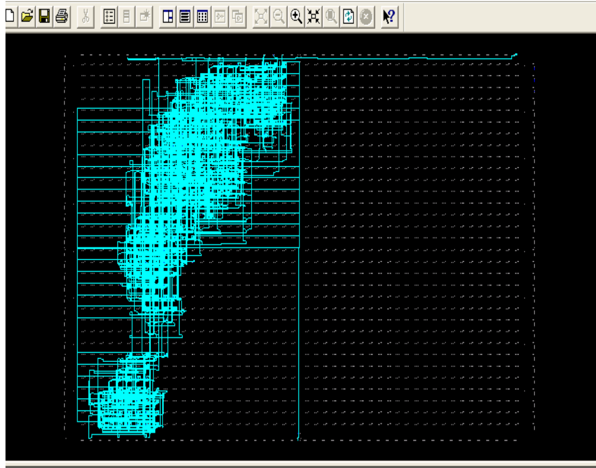

Fig8 : Showing the implementation of Huffman design on to the targeted FPGA (Xcv300-6bg432) generated on FPGA Editor Tool.

Figure depicts resource use for the FPGA target (Xcv300-6bg432) and channeled to the implemented Huffman coding system. The directive is implemented using the FPGA editor. It was noted that about 50% of the resources are used to implement the current Hofmann coding system at the target FPGA

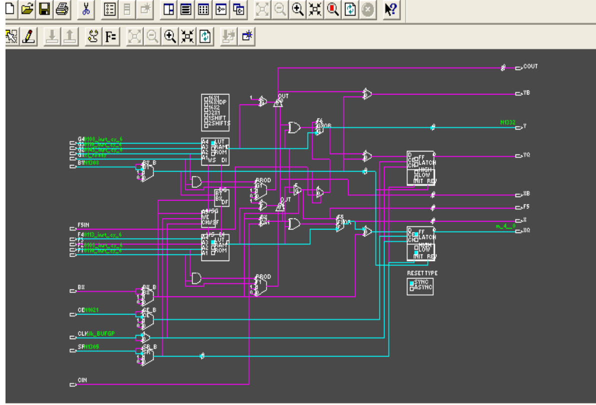

Fig 9 : Logical Placement of the implemented Huffman System

Figure 9 shows the implementation of the logical block in a configurable logical block (CLB) taken from the Huffman design as shown above, indicating the FPGA (Xcv300-6bg432) created in the FPGA Editor tool.

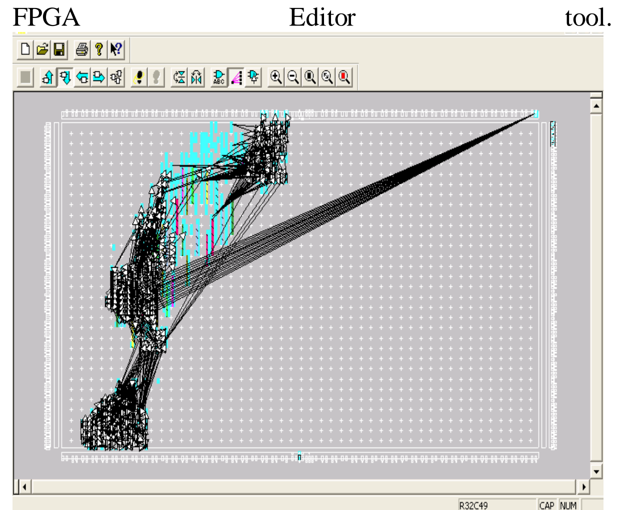

Fig 10 :Floor plan for the Huffman design developed on FPGA Floor planner tool.

The figure shows the blueprint of the system implemented for Huffman encoding in the target FPGA. The floor design provides a list of interworking networks between each node in a specific programmed FPGA structure.

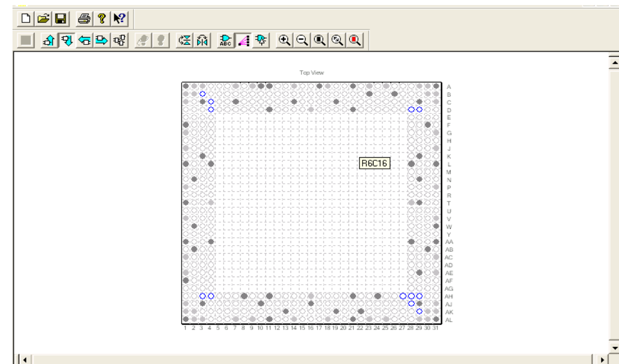

Fig 11: Chip view for the Huffman design developed on FPGA floor Planner tool.

The Figure shows the default chip configuration for the implemented Huffman encoding system. The figure shows the custom screws, VCC and GND pins for the executed design.

V. CONCLUSION

In this article, a length variable decoder is applied to decode. From the simulation results obtained, it can be seen very clearly that the proposed high-speed decoding system takes approximately 43.75 µs to decode a 12x32 test data set. The current tree-based Huffman decoder takes approximately 83.15 µs to decode the same data set. The implemented high-speed decoder takes approximately 400 clocks less than the current system. From these observations, it can be

concluded that the proposed system could give a higher decoding rate compared to the current Huffman coding system.

REFERENCES

[1] Rongshan Weia , Xingang Zhang: *" Efficient VLSI Huffman Encoder Implementation and its Application in High Rate Serial Data Encoding"* , IEICE Electronics Express, October 24, 2017

[2] D. A. Huffman: "*A method for the construction of Minimum-Redundancy Codes,"* Proc. I.R.E. #40 (1952) 1098.

[3] P. K. Shukla, et al. "*Multiple Subgroup Data Compression Technique Based on Huffman Coding*." First International Conference on Computational Intelligence, Communication Systems and Networks IEEE Computer Society, 2009:397-402.

[4] W. W. Lu and M. P. Gough: "A fast-adaptive Huffman coding algorithm," IEICE Trans. Commun. 41 [4] (1993) 535.

[5] V. K. Prasanna and H. Park: "*Area Efficient VLSI Architectures for Huffman Coding*," IEEE Trans. Circuits and Syst. (1993) 568.

[6] A. J. Mukherjee, et al. "*MARVLE: A VLSI Chip for Variable Length Encoding and Decoding*." IEEE International Conference on Computer Design on Vlsi in Computer & Processors IEEE Computer Society, (1992)170.

[7] L. Y. Liu, et al.: "*Design and hardware architectures for dynamic Huffman coding,"* IEE Proc. – Comput. Digit. Tech. 142 [6] (1995) 411.

[8] Y. S. Lee, et al.: "*A memory-based architecture for very-high-throughput variable length codec design,*" IEEE International Symposium on Circuits and Systems IEEE, 1997:2096-2099 vol.3.

[9] Babu, K. Ashok, and V. S. Kumar. "*Implementation of data compression using Huffman coding*." International Conference on Methods and MODELS in Computer Science IEEE, 2010:70 - 75.

[10] H. C. Chang, et al.: "*A VLSI architecture design of VLC encoder for high data rate video/image coding,"* Circuits and Syst. 4 (1999) 398.

[11] A. Mukherjee, N. Ranganathan, and M. Bassiouni: "*Efficient VLSI designs for data transformations of tree-based codes,"* IEEE Trans. Circuits and Syst. 38 [3] (1991) 306.

[12] T. Kumaki, et al.: "*CAM-based VLSI architecture for Huffman coding with real-time optimization of the code word table [image coding example],"* IEEE International Symposium on Circuits and Systems IEEE, (2005)5202 Vol. 5.

[13] Chen, Bei, et al. "*Huffman Coding Method Based on Number Character."* International Conference on Machine Learning and Cybernetics IEEE, 2007:2296-2299.