Journal of Engineering and Applied Sciences 11 (3): 402-407, 2016
ISSN: 1816-949X

A New Lossless Method of Huffman Coding for Text Data Compression and Decompression Process with FPGA Implementation

Maan Hameed, Asem Khmag, Fakhrul Zaman and Abd. Rahman Ramli
Department of Computer and Communication Systems Engineering,
University Putra Malaysia (UPM), Selangor, Malaysia

Abstract: Digital compression for reducing data size is important because of bandwidth restriction. Compression technique is also named source coding. It defines the process of compressed data using less number of bits than uncompressed form. Compression is the technique for decreasing the amount of information used to represent data without decreasing the quality of the text. It also decreases the number of bits needed to storage or transmission in different media. Compression is a method that makes keeping of data easier for a large size of information. In this study, proposed Huffman design includes encoder and decoder based on new binary tree for improving usage of memory for text compression. A saving percentage of approximately 47.95% was achieved through the suggested way. In this research, Huffman encoder and decoder were created using Verilog HDL. Huffman design was achieved by using a binary tree. ModelSim simulator tool from Mentor Graphics was used for functional verification and simulation of the design modules. FPGA was used for Huffman implementation.

Key words: Binary tree, data compression, decoding algorithm, Huffman Decoder, Verilog, FPGA

INTRODUCTION

Compression process is very important in the modern era of technology which is focused on speed and efficiency. Hence, large pieces of data are replaced by small bits of information which can be distributed between peers at faster rates (Swapna and Ramesh, 2015). The two main kinds of algorithms are a lossless and lossy. Lossless compression is utilized for the target that needs an accurate rebuilding of the original text while lossy process is employed when the user can allow some variation between the original and compressed data (Kate, 2012) Data compression which utilizes lossy method regularly cannot be reproduced accurately. Therefore, decompression process of the compressed file can be closer to the source data (Sharma, 2010). The fact that information requires storage and transfer has given rise to demands for best transmission and storage techniques. Different lossless compressions such as Shannon Fano, Arithmetic coding, Huffman coding and Run Length encoding algorithm are some of the techniques in use presently (Kodituwakku and Amarasinghe, 2010). David Huffman has tried to improve Huffman coding techniques. Huffman codes are prefixed codes and are optimum for a set of possibilities (Senthil and Robert, 2011). It is based on two considerations. Firstly, symbols that occur more frequently have lower codewords than symbols that occur less frequently in the best code. Secondly, the two symbols that occur least frequently have the same length in an optimum code. The Huffman procedure is accomplished by adding a simple requirement to these two pronouncements. This condition which corresponds to the lowest probability symbols differ only in the last bit.

Lossless compression: This type of compression indicates that no information is lost and the exact in the original file can be recovered by decrypting the encrypted file. In this model of compression the encrypted file is usually utilized for storing or sending information (Bhattacharjee *et al.*, 2013). Lossless process can be broadly classified into two types. The first one is entropy based encoding. In this method, calculation of repeated appearance of any unique representation in the text is done. Then, the compression reinstates the representatives with the produced symbols. The second type is dictionary based encoding. In this case, the encoder maintains data construction which identified as "Dictionary" (Wu *et al.*, 2006).

Huffman coding: Huffman code is one of the Variable Length Codes (VLC) which compresses data size

Corresponding Author: Maan Hameed, Department of Computer and Communication Systems Engineering,
University Putra Malaysia (UPM), Selangor, Malaysia

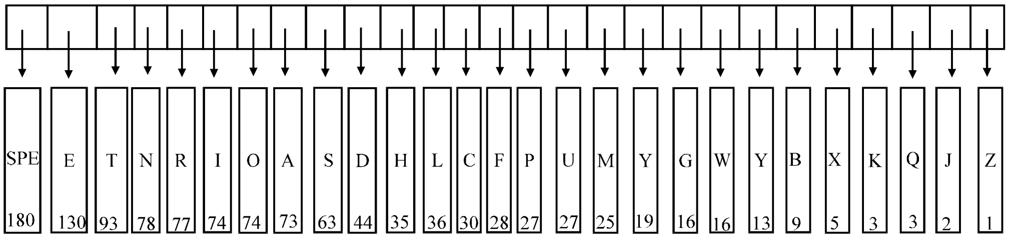


Fig. 1: Queue of data

(Beak *et al*., 2010). Static Huffman compression assigns variable length codes to symbols based on their rate of events in a given text. Low-frequency representations are encoded using a number of bits and high-frequency symbols are encoded using some bits. The coding manner creates a binary tree. The tree of Huffman with sections is labeled bits 0 and 1. Huffman tree is necessarily sent with compressed data to allow the receiver decode the information. The binary tree is constructed bottom-up and left-to-right. Firstly, the symbols are sorted in ascending order of their probabilities. In each step, two leaves with the least probabilities are placed in the tree. Their parent node has the sum of the probabilities of the two leaves. This step should be repeated until it reaches the root node which has probability of 1. After the tree is completed, 1's and 0's are assigned to left and right branches respectively. However, 1's and 0's can also be assigned to right and left branches respectively but the assignment must be consistent. Then, the coding for any value-symbol is created by traversing the binary tree from the source (root) of the tree to the particular leaves of interest. Tree construction involves merging the joints (nodes) step-by-step till each of them is installed in a rooted tree. This process always connects the two nodes presenting the lowering recurrence in a bottom-up method (Suvvari and Murthy, 2013).

Steps of building Huffman tree: Huffman's codes are run by substituting each alphabetical representation by VLC. ASCII employs 8-bits per symbol in English document. This is supposed wasteful because some special characters can occur more frequently than others (Asha Latha and Rambabu, 2012). Optimal Huffman codes can be created utilizing an effective algorithm. This is achieved by sorting the symbols by frequency in increasing order. This operation will be repeated n-1 times until all symbols are merged together. Each merging operation represents a node in a binary tree and the left or right choices on root-to-leaf path represent the bit of the binary codeword for each symbol. Production of a table of symbols distributed by rotation can be performed utilizing priority queues (Aarti, 2013). Huffman compression process has been confirmed to be effective in reducing the overall size of the data and employs the procedure of substituting fixed length bit of codes by VLC (Nourani and Tehranipour, 2005). The procedure for constructing Huffman tree involves making a list of free nodes and then selecting two nodes with the lowest weight from the list. After that, there will be creation of a parent node for the two selected nodes and its weight is equal to the total of nodes. Then, the parent node is added to the list of free nodes. This process is repeated until the construction becomes a single tree (Kodituwakku and Amarasinghe**,** 2010). Huffman generates code for the entire symbol of alphabet by traversing the binary tree from the root to the node. It assigns 0 to left hand branches and 1 to right hand branches. Building Huffman tree starts by arranging the text data according to frequencies to extract the code for each character. Initially, each node contains a symbol and its probability (Chen *et al*., 2006). Huffman tree is employed by both encoder and decoder. The alphabet consists of the uppercase letters and space. Huffman tree is based on the following assumed frequencies: E 130 T 93 N 78 R 77 I 74 O 74 A 73 S 63 D 44 H 35 L 35 C 30 F 28 P 27 U 27 M 25 Y 19 G 16 W 16 V 13 B 9 X 5 K 3 Q 3 J 2 Z 1. Every 1000 letters must have 180 spaces. Arrange the alphabet characters in ascending order of their frequencies for generating Huffman tree. There should be a table of free leaves wherever any leaf is corresponds to a symbol in the alphabet in ascending order according to their frequencies. Figure 1 shows the queue of arranging data in ascending order starting from the lowest frequency to the highest one.

MATERIALS AND METHODS

Construction of the tree: A binary tree consists of a collection of nodes and leaves. Each node connects a pair of nodes or leaves. Node at the height of the binary tree is named the source (root) of the tree, it also represents the parent for two leaves and each node should have at least one leaf. The initial process to build Huffman tree by selecting two free leaves with the lowest weight from the

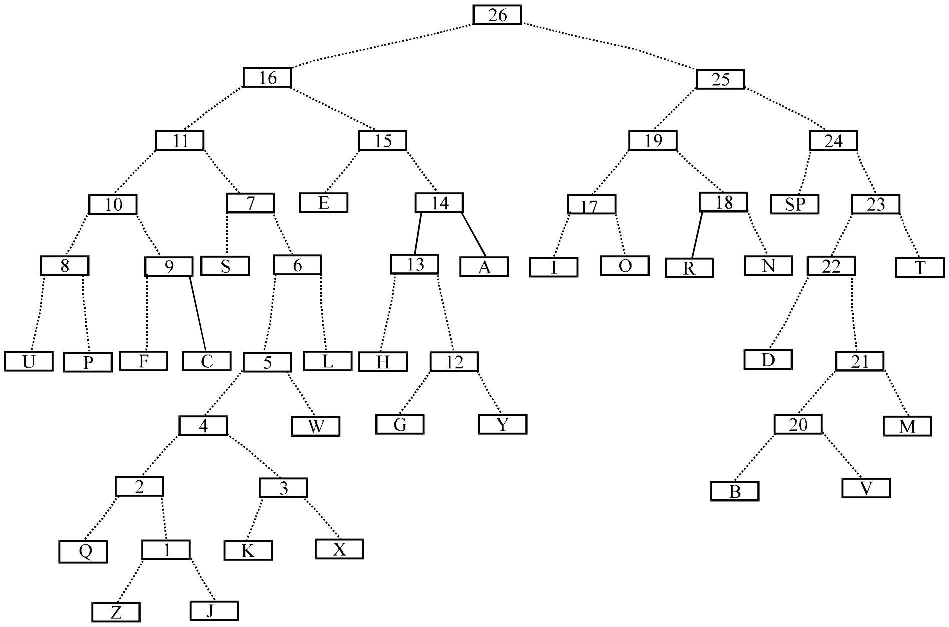


Fig. 2: Huffman binary tree

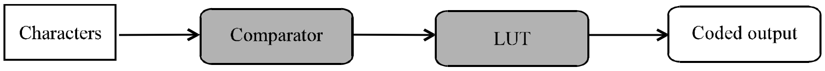


Fig. 3: Show the block diagram of encoder

list to create new node and make the frequency of new node equals to sum of frequencies of left and right children (Pujar and Kadlaskar, 2010). In this method, a Huffman design is implemented using binary tree to get the smallest size of data compression which is built upon using the frequencies corresponding to characters. Where during this work achieved the smallest size by arranging the branch values for tree based on its construction using the characters and respective frequencies in the way leads to the smallest size. In Fig. 2, shows a binary tree in which the branch values are arranged in order to get the best compression. The main idea of coding is to assigns shorter codes to symbols that occur more frequently and longer codes to those that occur less frequently. Both encoding and decoding process should be done on the same tree. Hence, the data stored in the encoder is stored in the encoder LUT Gonzalez and Woods 2002.

Implementation of encoder: Huffman coding process using VLC leads to the best compression rate of encoding length values (Chung and Wu, 1999) In this research, the encoder is implemented using Huffman tree. Huffman tree used by encoder and decoder is used to estimate codeword bits (Brown, 2007). The encoder retrieves the code for each symbol from a map and shifts it out one bit at the time. The decoder is obtained from the tree by adding acts from the leaves back to the top of the tree. If a state is not a leaf of the tree and its encoding is n, then the encodings of its two children are 2n+1 and 2n+2, respectively. Character input which is given to the encoder acts as input to the LUT which gives corresponding encoded word on the data bus. This is then given to a shift register to serially shift the data out. As it is a variable length coding, in order to determine the end of the codeword for each character while shifting out, one more bit is added to the end of the code word in the LUT which should made 1.

The codeword is logically shifted out till it contains only 1 at its LSB. Then, next character is loaded from the comparator. Apart from this, the encoder should generate an enable signal to the decoder so that the decoder knows when the valid data is presented to it. Figure 3 shows the block diagram of an encoder and codes for each character which comes from the tree as presented before is stored in LUT.

Implementation of decoder: In the decoder, the coded value is first stored in the buffer and then shifted using a LIFO. The shifted value is then stored in the 9-bit temporary register which is then compared with respective codes stored in the LUT. Then, the character is finally

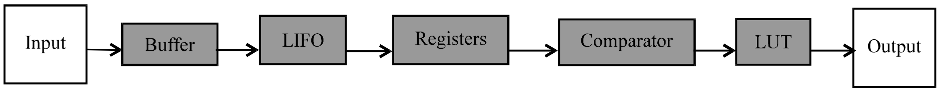
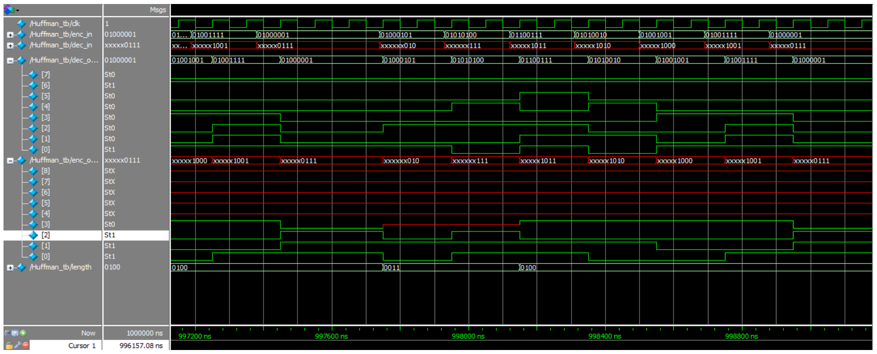


Fig. 4: Block diagram of Huffman decoder



```
                               Msgs
/Huffman_tb/clk                1
/Huffman_tb/enc_in             01000001
/Huffman_tb/dec_in             xxxxx0111
/Huffman_tb/dec_o...           01000001
  [7]                          St0
  [6]                          St1
  [5]                          St0
  [4]                          St0
  [3]                          St0
  [2]                          St0
  [1]                          St0
  [0]                          St1
/Huffman_tb/enc_o...           xxxxx0111
  [8]                          StX
  [7]                          StX
  [6]                          StX
  [5]                          StX
  [4]                          StX
  [3]                          St0
  [2]                          St1
  [1]                          St1
  [0]                          St1
/Huffman_tb/length             0100

Now                            1000000 ns
Cursor 1                       996157.08 ns
```

Fig. 5: Full Huffman design waveform validation

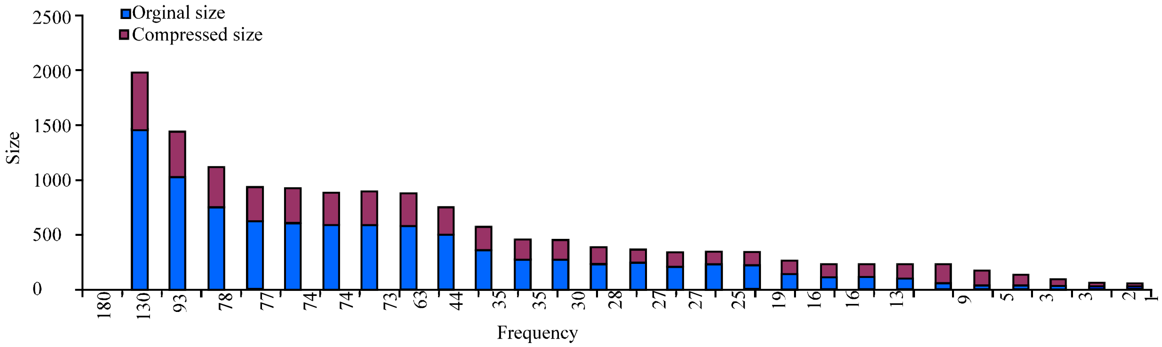


Fig. 6: Comparison of compressed and uncompressed data sizes

decoded. Both encoding and decoding should be done with respect to the same tree. In this method, inside the decoder block first a buffer is first presented inside the decoder block in order to store the output of the encoder part. A LIFO which will shift the coded values stored inside the buffer is presented next to it. This shifted code is then stored inside a temporary register of 9-bit size. Both the coded value and Huffman tree which are stored inside the LUT are compared to obtain a decoded output with respect to the corresponding coded state. In Fig. 4, the block diagram of Huffman decoder clearly explains the operation as shown.

RESULTS AND DISCUSSION

HDLs and their simulators allow designers to partition their designs into components that can work concurrently and communicate with one another (Arya and Tato, 2014). Additionally, encoder inputs involve a Clk signal and 8-bit ASCII which represent input data for Huffman encoder to generate 9-bit output data with variable length coding. However, decoder module consisted of Clk signal with 9-bits as input data to generate 8-bit ASCII which represents decoder output. Figure 5 shows all inputs and outputs for Huffman design and top-level Huffman simulation.

Measuring performance of huffman compression: When estimating the quality of execution, the main difficulty will be size competence. Considering coding mode which depends on the repetition of components in the reference data, it is difficult to estimate the quality of a coding process in 1. The quality of compression depends on the model and the arrangement of the input source. Furthermore, compression process behavior is based on the type of the compression process. Estimation of compressed size will be done by determining the size of the uncompressed data. This is equal to addition of the frequencies of all the letters of alphabet. Calculation of total size for original data is shown in Eq. 1. Equation 2 shows the process of calculating compressed data size (Sharma, 2010).

$$\text{Original data size} = \text{Total frequency} \times \text{ASCII} \tag{1}$$

$$\text{Original data size} = 1180 \times 8 = 9440 \text{ bits}$$

$$\text{Compressed data size} = \text{Codeword bits} \times \text{frequency} \tag{2}$$

$$\text{Compressed data size} = 4913 \text{ bits}$$

Saving percentage is derived through calculating the shrinkage of the source file in percentages. This is widely accepted as the measure of efficiency of a compression method and is defined in percentage (Mohammed *et al*., 2015).

$$\text{Saving percentage} = \left(\frac{\text{(original size-new size)}}{\text{(original size)}}\right)100\% \tag{3}$$

$$\text{Saving percentage} = 47.95\%$$

The proposed method of Huffman design has high saving percentage of data size up to 47.95%. Figure 6 shows the comparison of compressed and uncompressed sizes.

CONCLUSION

This study has shown that higher level of information helps increasing compression quality. The newly presented coding and decoding processes depend on binary tree for compression and decompression of data to reduce data size. The proposed design is used to compress text files of size 9440 bits to achieve a new compressed size of 4913. The proposed Huffman design has saving percentage of about 47.95% of the original size. Future works need to be carried out to improve the area. Compared to other different compression techniques, we conclude that Huffman compression is a more efficient compression for image coding and decoding to an appreciable degree.

REFERENCES

Aarti, A., 2013. performance analysis of huffman coding algorithm. Int. J. Adv. Res. Comput. Sci. Software Eng., 3: 615-619.

Arya, K.V. and N. Tato, 2014. A lossless compression algorithm for video frames. Proceedings of the 9th International Conference on Industrial and Information Systems, December 15-17, 2014, Gwalior, pp: 1-5.

Asha Latha, P. and B. Rambabu, 2012. A new binary tree approach of huffman code. Int. J. Soft Comput. Eng., 1: 59-62.

Beak, S., B. Van Hieu, G. Park, K. Lee and T. Jeong, 2010. A new binary tree algorithm implementation with Huffman decoder on FPGA. Proceedings of the Digest of Technical Papers International Conference on Consumer Electronics, January 9-13, 2010, Halmstad, pp: 978-979.

Bhattacharjee, A.K., T. Bej and S. Agarwal, 2013. Comparison study of lossless data compression algorithms for text data. IOSR J. Comput. Eng., 11: 15-19.

Brown, S.D., 2007. Fundamentals of Digital Logic with Verilog Design Tata. McGraw-Hill, New York.

Chen, C.Y., Y.T. Pai and S.J. Ruan, 2006. Low power Huffman coding for high performance data transmission. Proceedings of the International Conference on Hybrid Information Technology, Volume 1, November 9-11, 2006, Cheju Island, pp: 71-77.

Chung, K.L. and J.G. Wu, 1999. Level-compressed huffman decoding. IEEE Trans. Commun., 47: 1455-1457.

Gonzalez, R.C. and R.E. Woods, 2002. Digital Image Processing. Prentice Hall, New Jersey.

Kate, D.M., 2012. Hardware implementation of the huffman encoder for data compression using altera DE2 board. Int. J. Adv. Eng. Sci., 2: 11-15.

Kodituwakku, S.R. and U.S. Amarasinghe, 2010. Comparison of lossless data compression algorithms for text data. Indian J. Comput. Sci. Eng., 1: 416-425.

Mohammed, M.H., A. Khmag, F.Z. Rokhani and A.R. Ramli, 2015. VLSI implementation of huffman design using FPGA with a comprehensive analysis of power restrictions. Int. J. Adv. Res. Comput. Sci. Software Eng., 5: 49-54.

Nourani, M. and M.H. Tehranipour, 2005. RL-Huffman encoding for test compression and power reduction in scan applications. ACM Trans. Design Automation Electr. Syst., 10: 91-115.

Pujar, J.H. and L.M. Kadlaskar, 2010. A new lossless method of image compression and decompression using huffman coding techniques. J. Theoretical Applied Inform. Technol., 15: 18-22.

Senthil, S. and L. Robert, 2011. Text compression algorithms-a comparative study. J. Commun. Technol., 2: 444-451.

Sharma, M., 2010. Compression using Huffman coding. Int. J. Comput. Sci. Network Secur., 10: 133-141.

Suvvari, V. and M. Murthy, 2013. VLSI implementation of huffman decoder using binary tree algorithm. Int. J. Electr. Commun. Eng. Technol., 4: 85-92.

Swapna, R. and P. Ramesh, 2015. Design and implementation of huffman decoder for text data compression. Int. J. Curr. Eng. Technol., 5: 2032-2035.

Wu, K., E.J. Otoo and A. Shoshani, 2006. Optimizing bitmap indices with efficient compression. ACM Trans. Database Syst., 31: 1-38.