www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 3 Issue 10 October, 2014 Page No.8854-8858

Efficient Data Transmission by Introducing Stuffing Bits in HUFFMAN Coding Technique

G. S. L. Alekhya, M. S. S. Bhargav, A. Narayana Kiran

Electronics and Communication Engineering Department
Shri Vishnu Engineering College for Women, India
alekhya.gsl76@gmail.com

Electronics and Communication Engineering Department
Vishnu Institute of Technology, India
medicharla.bhargav1@gmail.com

Electronics and Communication Engineering Department
Shri Vishnu Engineering College for Women, India
narayanakiran.akondi@gmail.com

Abstract— **In recent years, image and video encoding has become more popular in network access. The rapid development in wired and wireless digital communication has made the extensive use of the text data. However, there are few researches focusing on encoding data and memory usage. The basic characteristics of text data like transmission rate, bandwidth, redundancy, bulk capacity and co-relation among text data makes basic compression algorithms mandatory. Therefore this paper considers the problem of memory usage and encoding scheme to provide low bit rate transmission based on HUFFMAN coding. As for n bit being transmitted it requires 2n memory stack for further increase in the data bits it requires 2n+1 memory stack which is wide waste of memory if there presents redundancy bits. Image transmission has large repeated sequences at some places which can be considered as redundant. The proposed method uses stuffing bits in order to provide high speed and low cost transmission**

.

Keywords— **Decoding, HUFFMAN coding, Image encoding, Redundancy, Stuffing bits etc.**

I. Introduction

In recent years, compressing an image before transmitting has gained a lot of interest with a rapid growth of multimedia and presence of wide network access, as uses of this compressing of data ranges from mobiles, laptops to high quality satellite communication. Compressed data is the art of presenting data in its compact form which is decompressed i.e., the original form of data is observed. Compression techniques are used to reduce the amount of data that would otherwise be needed to store, handle, and/or transmit the represented content. Using compression technique provides high bandwidth rate, as HUFFMAN coding is a variable length coding, it provides an advantage of increased compression rate. Hence it is widely used as compression technique during transmission of images and videos. In this STUFFING bits are used during compression of data which are observed and removed during decompression [1][6]. Here using of stuffing bits provides advantage of decreasing the memory size which thereby reduces the cost [2].

Without using stuffing bits the memory required should change dynamically. As the sequence of same value bits increases the count value which there by increases the memory width and size. The memory width is the count of the largest sequence of the incoming data, it is waste of memory as rest of the sequence count may not require that much width. The overall cost of the encoder and decoder increases.

To avoid such disadvantages the concept called bit stuffing is introduced to the encoding technique. As stuffing an opposite bit after the largest allowing sequence allow the available memory to be used efficiently thereby decreasing the overall cost.

II. HUFFMAN CODING TECHNIQUE

The technique works constructing a binary tree of nodes. The size of the tree depends on the number of symbols [8].

The simplest construction algorithm uses a priority queue where the node with lowest probability is given highest priority [3][7]:

1. Create a node called the leaf for each symbol and add it to the priority queue.
2. If there is more than one node in the queue:

    a. The nodes having the highest priority are to be removed (lowest probability) from the queue
    b. Create a new internal node with these two nodes as children and with probability equal to the sum of these two nodes' probabilities.
    c. Add the new node to the queue.

3. The remaining node is the root node and the tree is complete.

In figure 1 as shown Huffman coding is a variable length coding. In a variable-length code the code words may have different lengths. Here are examples of fixed and variable length codes for our problem (note that a fixed- length code must have at least 3 bits per code word).

No. bits for fixed length coding is 150*3=450 and for variable length coding is 60*3+5*3+30*3+5*3+50*1=350

10% memory is saved.

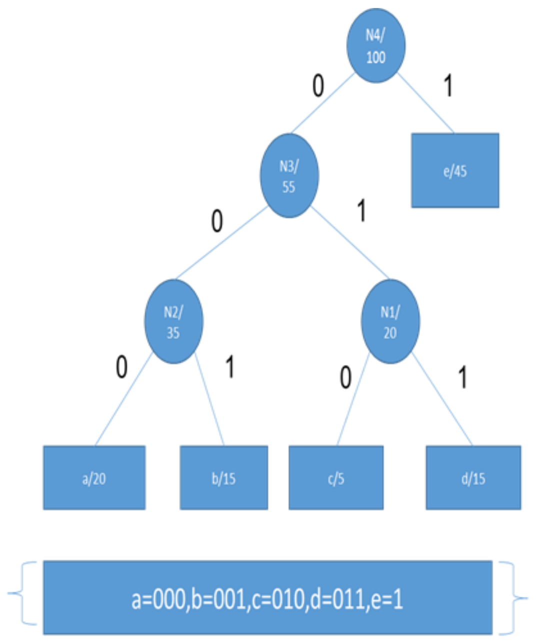

Fig 1: Huffman tree.

Table 1: Huffman coding example.

	a	b	c	d	e
Frequency	**60**	**5**	**30**	**5**	**10**
Fixed length	**000**	**001**	**110**	**101**	**111**
Variable length	**000**	**001**	**010**	**011**	**1**

III. Bit Stuffing

Stuffing bits are mainly used to limit the occurrence of consecutive bits having the same value. To limit this occurrence, a bit of opposite value is inserted after allowing maximum number of consecutive bits [4]. Bit stuffing is mainly used to limit the width of memory and decrease the cost. As memory cannot be changed dynamically and is also a cost issue, increased occurrence of same value consecutively which exceeds the existing memory width may cause loss of data which cannot be renewed during decoding and increasing the width of the memory also increases the memory size which there by increases the cost. To avoid this problem and for efficiently using the already existing memory bit stuffing plays a prominent role.

25 consecutive ones are represented by 5 bit width memory.

$$\underbrace{1111111111111111111111111}_{11001,\ 1}$$

By Bit Stuffing 4 bit width memory is sufficient to represent the given pattern

$$\underbrace{111111111111111}_{1111,\ 1}\underset{\text{stuffed bit}}{\underset{\Uparrow}{0}}\underbrace{1111111111}_{1010,1}$$

At the receiver end this stuffing bit is removed and original data is obtained. Here stuffed bit should not confuse with the overhead bits. Receiver should have the information of the value of maximum number of consecutive bits that is being sent.

Therefore, increased speed of transmission, reduced cost of memory and efficient usage of available memory are achieved by bit stuffing.

IV. HDL Implementation of Encoder and Decoder.

In this paper VHDL implementation of the module which is used for data transmission is observed. This module is again divided into two sub modules they are i) Encoder ii) Decoder.

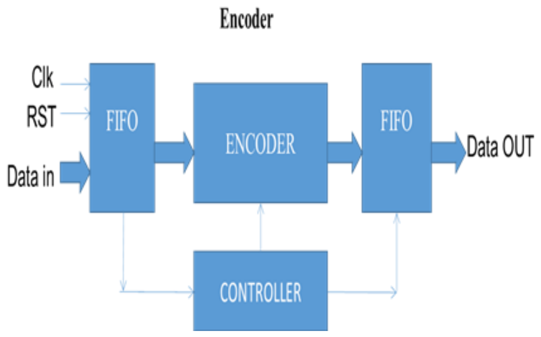

Fig2: Block diagram of encoder.

The Encoder block again contains sub modules such as a) FIFO b) Encoding module c) Controller. In Encoder, the incoming binary data is stored in FIFO module (First in First out) which is given to the encoder module as shown in Fig2. In encoder module the counter is present which counts the repeated binary value and when another binary value appears next it stores the previous value and the count

[5]. This continues as per the input data. If the count increases the width of the FIFO stack, stuffing bit is inserted so that the remaining repeated binary values are stored at next address after the stuffing bit .Here the controller gives the signals to the encoder such as FIFO full, FIFO empty, FIFO read, FIFO write etc.

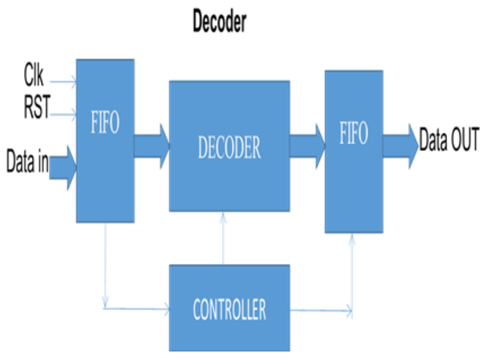

Fig3: Block diagram of decoder.

At the receiver, the FIFO which has the stored data and the count is being decoded simultaneously. Here the data is decoded and the binary value is given out as much as the count value. Stuffing bit is observed and removed as it is identified at the decoder as the appearance of opposite binary after the repeated number of binary bits of same value. So, the original data is obtained and is given to the output FIFO.

The controller gives the control signals to the encoder and decoder such as when to write, read, FIFO is full, where to insert stuffing bit etc as shown in fig3.

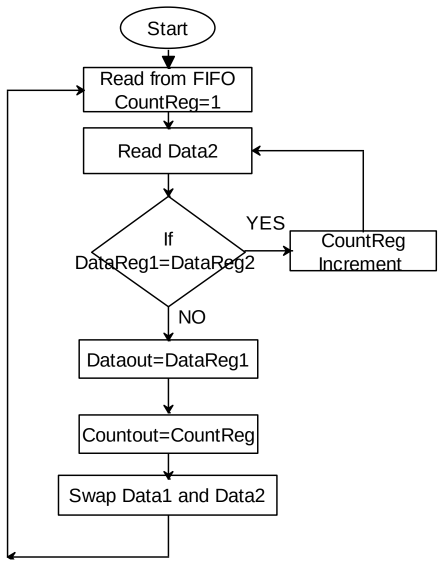

Fig 4: Flow chart of the encoder.

The incoming data which is to be transmitted is stored in FIFO which is read from when data is being encoded during transmission. first data is read and stored in register1 and second data is read and stored in register2. These two information bits ie, binary data are compared if the two binary bits are same then count register is incremented and again the next data is fetched and compared, if not equal send the data to the data out and count to count out. Now the two data register values are swapped and again fetch the data to the second register.

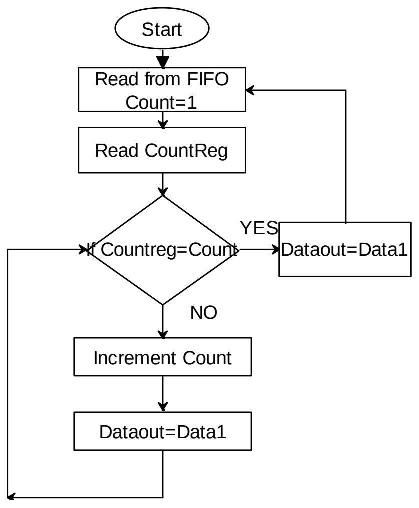

Fig 5: Flow chart of the Decoder.

The above flow chart describes the functioning of the decoder. The count value which stored in count register represents the repeated binary values is read and the data bit being repeated is also read. Initially the count is made 1 and it is compared with the value in the count register. If the values are equal then data is given to the output ie, to the targeted point. If they are not equal then count is incremented and it checks until count is equal and gives it to output.

A. *FIFO Stack*

FIFO is used to store the incoming data and send to any other module when necessary and to maintain same frequency between transmitter and the compressor. As there may be some frequency difference between transmitter and the compressor unit which may lead to in appropriate working of the unit.

B. *Encoder Algorithm*

Step1: Read data from the input FIFO and assign it to Datareg1, again read next data from FIFO and assign it to Datareg2.
Step2: Compare the data present in datareg1 and datareg2.
Step3: If the binary bits are equal increment the countreg value.
Step4: If count reg value reaches the maximum value, send the data to the output FIFO and also send the count value to output FIFO.
Step5: Now insert a opposite value bit to the one present in data out, which is called the stuffing bit.
Step6: Assign the value of the bit present in datareg2 to datareg1 and fetch next binary value from the input FIFO to datareg2.
Step7:Again compare the binary values present in datareg1 and datareg2 and go to step3.
Step8:If the binary values present in the datareg1 and datareg2 are not equal then send the data present in datareg1 to output FIFO and countreg value to output FIFO and go to step6.

C. *Decoder Algorithm:*

Step1: Read the data from the output FIFO of the encoder.
Step2: Assign data bit to datareg1 and count to datareg2.
Step3: Initialize count to 0 and compare count and datareg2.
Step4: If the binary values are equal then go to step2.
Step5: If the binary values are not equal then send the data present in the datareg1 to data out.

V. Experimental Results

The data which is to be transmitted is first collected through transmitter or analog sensor. This analog data is sent through analog to digital converter as the data being transmitted is sent in digital format. Converted digital data is stored in FIFO which is encoded for speed transmission. The encoded data is again stored in FIFO for decoding. Working of encoder and queues that are used to store the data can be explained by seeing the simulation results.

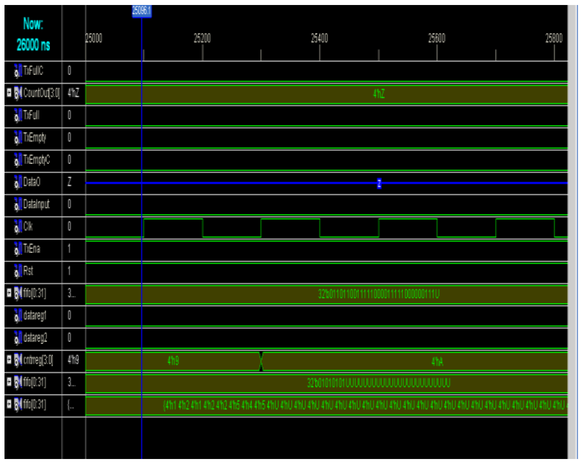

Fig 6:simulation results of compressor without stuffing bits.

Incoming serial data is stored in FIFO which is encoded and stored in another FIFO. This encoded data is given to another FIFO to extract the original data ie, decoding the encoded data.

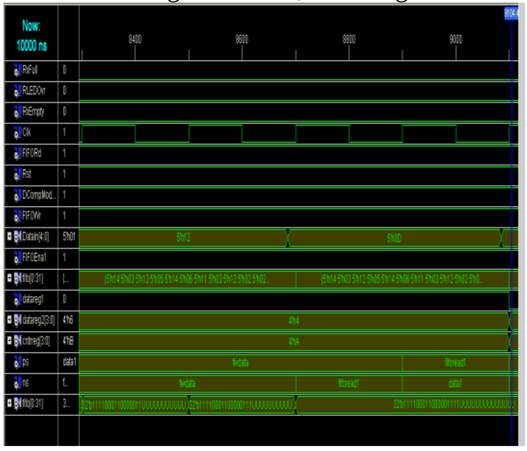

Fig 7:Simulation results of Decoder without stuffing bits.

The data coming from the compressor is sent to decoder by first storing the compressed data in FIFO. Stored data is passed through the decoder module which decodes the data and abstracts the original data.

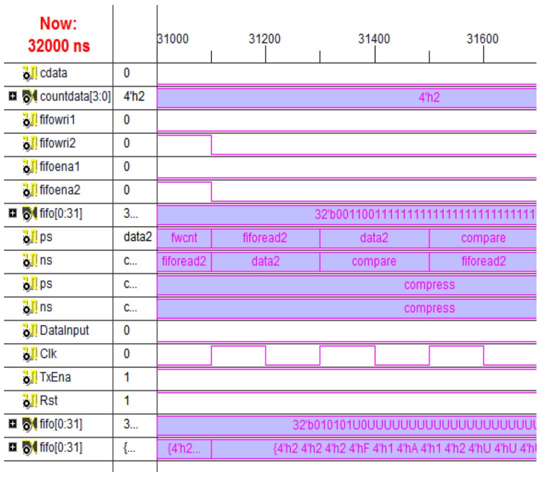

Fig 8: Simulation results of Encoder using Stuffing bits. Using stuffing bits, as we can see it avoids the repeating of consecutive binary digit of same value.

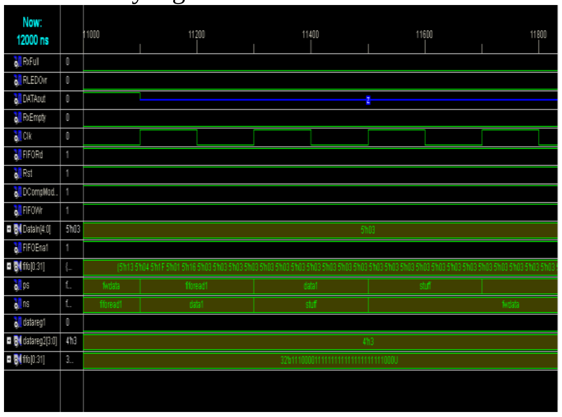

Fig 9: Simulation results of decoder using stuffing bits.

The original data is extracted from the encoded data and is given to the targeted device.

VI. Conclusions

The encoding and decoding technique used is highly reliable and Huffman technique used here makes the data uniquely decodable. Using Stuffing bits provides efficient use of memory and increase in speed factor as it helps in breaking the long sequence of same binary value. With this proposed method of adding stuffing bits there is a small burden on memory because of the extra stuffing bits but the effect of stuffing bits is very much high while reducing the memory required and also cost of the integrated circuit. Overall memory required to store the complete data sequence reduces. In the case of communication systems, the transmission rate can be increased with the introduction of stuffing bits. The overall cost is also reduced. By adopting some other compression techniques the transmission rate can be further improved.

References

[1] Dr. Muhammad Younus Javed and Mr. Abid Nadeem,"Data Compression Through Adaptive Huffman Coding Scheme" , IEEE-2000, Vol. II, pp.187-190.

[2] D.A.Huffman,A method for the constraction of Minimum Redundancy Codes , Proceedings of the IRE, 1952 ,pp. 1098-1101.

[3] Hirschberg , D.S. and Lelewer,D.A. ,Efficient Decoding of prefix codes ,Communication of the ACM, pp.449-458, 1990.

[4] J. Feng and G. Li, " A Test Data Compression Method for System-on-a-Chip" 4th IEEE International Symposium on Electronic Design, Test and Applications, 2008.

[5] Nadeem , A. ,Design and Implementation of Data Compression System , College of Electrical and Mechanical Engineering ,Rawalpindi ,Thesis, 1995.

[6] P. Bender and J. K. Wolf "A universal algorithm for generating optimal and nearly optimal run-length-limited, charge constrained binary sequences", Proc. IEEE Int. Symp. Information Theory, 1993

[7] Shukla, P.K. ; Rusiya, P. ; Agrawal, D. ; Chhablani, L. ; Raghuwanshi, B.S. "Multiple Subgroup Data Compression Technique Based on Huffman Coding" International conference on Computational Intelligence, Communication Systems and Networks, pp. 397- 402, 2009.

[8] Vitter, S. V.,Design and analysis of dynamic Huffman codes ,Journal of the Assocition for Computing Machinery ,Vol. 34, No. 4, pp. 825-845, 1987.