International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 2, February - 2014

FPGA Implementation of Huffman Encoder and Decoder for High Performance Data Transmission

Shireesha Thummala[1] ,Thrisul Kumar. J[2], Swarna latha. E[3]
[1] Vignan Institute of Technology and Aeronautical Engineering, Vignan Hills,Hyderabad
[2] Vignan Institute of Technology and Aeronautical Engineering, Vignan Hills,Hyderabad
[3] Vignan Institute of Technology and Aeronautical Engineering,Vignan Hills,Hyderabad

Abstract - In computer science and information theory, Huffman coding is an entropy encoding algorithm used for lossless data compression. The purpose of this paper is to present and analyze HUFFMAN CODING ALGORITHM for the data compression and decompression. Huffman coding is a minimal variable character coding based on the frequency of each character. First, each character becomes a trivial binary tree, with the character as the only node. The character's frequency is the tree's frequency. Two trees with the least frequencies are joined as the sub trees of a new root that is assigned the sum of their frequencies. This is repeated until all characters are in one tree. One code bit represents each level. Thus more frequent characters are near the root and are coded with few bits, and rare characters are far from the root and are coded with many bits. In this paper Huffman encoder and decoder are designed in VHDL. Huffman decoding is done by using a state diagram approach. ModelSim simulator tool from Mentor Graphics will be used for functional simulation and verification of the encoder & decoder modules. The Xilinx Synthesis Tools (XST) will be used to synthesize the complete design on Xilinx family FPGA.
Keywords: VHDL,FPGA,XST,ASCII,VLC

1. INTRODUCTION

Compression means storing data in a format that requires less space than usual. Data compression is particularly useful in communications because it enables devices to transmit the same amount of data in fewer bits.The bandwidth of a digital communication link can be effectively increased by compressing data at the sending end and decompressing data at the receiving end. Huffman coding is a popular compression technique that assigns variable length codes (VLC) to symbols. On decompression the symbols are reassigned their original fixed length codes. The compression or decompression scheme described in this paper is based on statistical coding. In statistical coding, variable length code words are used to represent fixed-length blocks of bits in a data set. For example, if a data are divided into four-bit blocks, then there are 16 unique four-bit blocks. Each of the 16 possible four-bit blocks can be represented by a binary codeword. The size of each codeword is variable (it need not be four bits). The idea is to make the code words that occur most frequently have a smaller number of bits, and those that occur least frequently to have a larger number of bits. This minimizes the average length of a codeword. A Huffman code is an optimal statistical code that is proven to provide the shortest average codeword length among all uniquely decodable variable length codes.

A Huffman code is obtained by constructing a Huffman tree. The path from the root to each leaf gives the codeword for the binary string corresponding to the leaf.Huffman decoder uses a lookup table for retrieving the original or transmitted data from the encoder. This lookup table consists of all the unique words and their corresponding code vectors. Once the Huffman coding is performed, the compressed data is transmitted serially to the decoder. Once the complete data has been received, decoding of compressed data has done by the state diagram approach and the decoded data is said to be decompressed and the complete process of decoding is called as decompression of data.

2. DESIGN BLOCK DIAGRAM

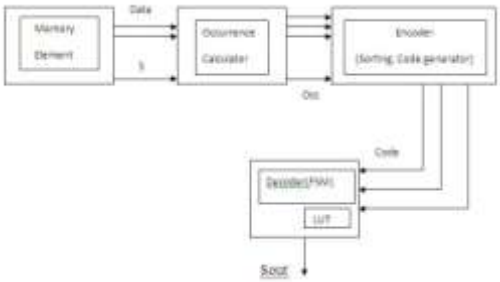

Figure 1. Block Diagram of Design

2.1 Input memory element

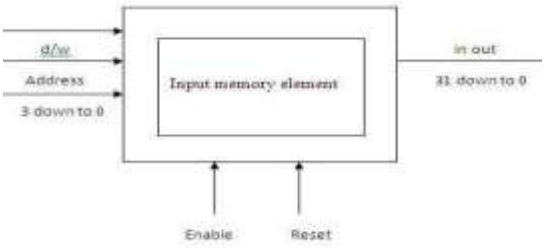

Figure 2.Input memory element

Input Memory Element consists of twelve locations of 32 – bit length each. This block stores the data which is passed down to the Occurrence Calculator. Signal 'S' is the

status signal, made '1' once the complete data is passed on to the Occurrence Calculator, 'clk ' is the clock signal which activates the occurrence calculator on rising edge, 'reset' is the reset signal, which when high , initializes all the memory locations to zero.'en' when " active –high " only allows all the transactions from or to the memory element. 'Address (3 down to 0) 'is the address bus which takes a number lying in between 0 to 12 , 'rw' is the read /write signal, when '1', data is written in to the memory location given by the 'Address' bus and when '0' , data is read from the location specified by the 'Address' bus. 'in out' is a bidirectional bus, which transfers data from the memory or into the memory.Memory Element transfers all the data stored in it as 4-bit blocks to the occurrence calculator and each 4 –bit block is individually called as a unique word. Thus the number of unique words is sixteen (0000 to 1111).

2.2 Occurrence calculator

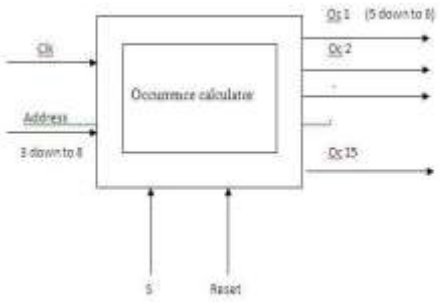

Figure 3.occurrence calculator

This block calculates the occurrences of the number of unique words present in the data. It generates a control signal to the Encoder block as status 'S' ,to indicate the completion of occurrence calculation and activates the Encoder(when S=1) , clk is the clock signal which activates the occurrence calculator on rising edge, rst is reset when high, all the occurrence values are initialized to zero, din (3 down to 0)is the 4 –bit input data fed to the occurrence calculator from the input memory element , oc1 to oc15 are the occurrence values of the unique words.

2.3 Code Generator

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

1. Sorter 2. Adder

2.3.1 Sorter

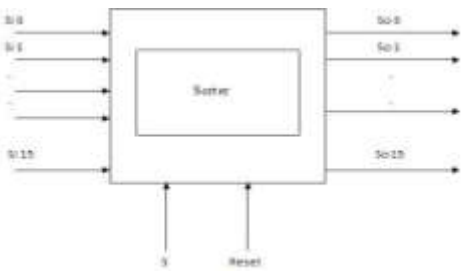

Figure 4.Sorter of code generator

Sorter has si0 , si1 , si2…..si15 as input values that are taken from the occurrence calculator and sorts them in descending order, when the enable signal 'en' is 'active-high' and the reset signal 'rst' is 'active -low'. The sorted values are passed to the code generator.

2.3.2 Adder

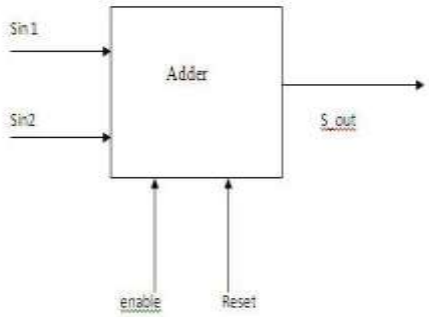

Figure 5.Adder

The functionality of the adder is to calculate the sum of two values passed to it provide the sum of those two inputs. The summer is used in encoder so as to calculate the sum of last two elements after each sort.

2.4 Code Generator

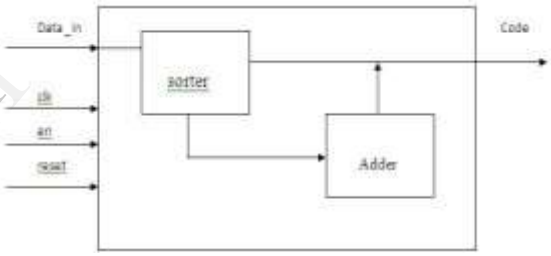

Fig 6.Code generator

This is the block where the actual Huffman code is generated .The code generated in the code generator is passed to the encoder block as input where each input data is encoded with its corresponding code. If the input data is divided into n-bit blocks, the Huffman Code contains the at most n+2 number of bits. It performs the compression by replacing each of the unique word with its equivalent code words and stores the compressed data.

2.5 .Huffman Decoding

This block receives the encoded data bits in the form of serial bits and are decoded back to get the original data. This block consist of Comparator and Look up table. Comparator compares the received compressed data from the encoder with the predefined code words stored in LUT. Look up table (LUT) stores a predefined code sequence from which the comparator takes the data and uses for decoding. Huffman decoding is done by using a state diagram approach and decoding of compressed data is done based on comparison of each incoming bit with that of the available lookup table of code vectors. Each bit of data after reception is compared with the code vectors of the look up table , if matched its state is returned to initial state else it state goes to some other state. If the data does not match

with any of the code vectors from the lookup table, it accepts the next occurring data and tries to match with the code vector from the lookup table. Once a match of code vector with the received data pattern occurs, the control is transferred to the initial state, indicating that the data has been matched and the decoder is ready to accept the next bit. If unmatched, control is transferred to next state and again makes an attempt to match the look up. Similar process continues till the complete incoming data is decoded.Once the complete data has been received,decoding of compressed data has done as said earlier by the state diagram approach and the decoded data is said to be decompressed and the complete process of decoding is called as decompression of data.

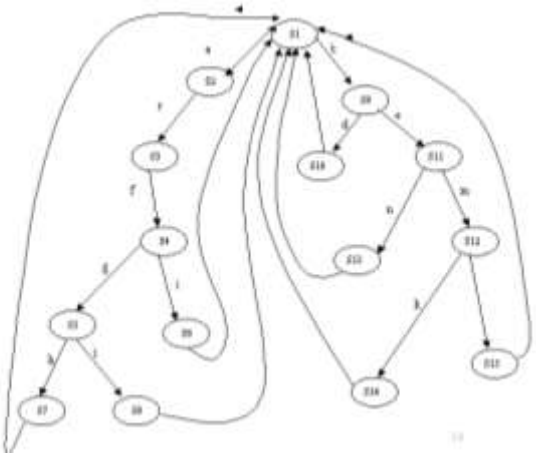

Figure 7. FSM for Huffman Decoder

Figure 8.State diagram

3. EXAMPLE FOR HUFFMAN ENCODER AND DECODER

Table 1. Input sequence considered

1111	0101	0011	1111	1011	1110	1101	1011
1111	1111	1111	1111	0000	0000	0000	0000
1001	0010	0110	0111	1110	1101	0110	1110
1111	1111	1110	0110	1111	1001	1111	1011
1111	1111	0111	1010	1111	1111	0111	1111
0010	1110	1000	0100	1111	1111	1111	1111
0110	1011	1011	1011	1101	0111	1011	0111
1100	1000	1010	0111	0101	1011	1111	1101
1011	0100	1101	1101	1110	1111	1111	1111
1111	1011	0111	0101	1111	0111	1111	1101
1100	1101	1001	1110	1110	1101	1011	0110
0111	0010	1111	1111	0111	1111	1011	1111

Table 2.Huffman Encoder output data

11XX	01111	10100	11XX	100X	000X	001X	100X
11XX	11XX	11XX	11XX	10101	10101	10101	10101
01110	10100	0110	010X	000X	001X	0110	000X
11XX	11XX	000X	0110	11XX	1001	11XX	100X
11XX	11XX	010X	101111	11XX	11XX	010X	11XX
10100	000X	101100	101101	11XX	11XX	11XX	11XX
0110	100X	100X	100X	001X	010X	100X	010X
101110	101100	101111	010X	01111	100X	11XX	001X
100X	101101	001X	001X	000X	11XX	11XX	11XX
11XX	100X	010X	01111	11XX	010X	11XX	001X
101110	001X	01110	000X	000X	001X	100X	0110
010X	10100	11XX	11XX	010X	11XX	100X	11XX

Table 3.Look up table

I	*4bit data*	*Occurrence*	*Pi*	Huffman Code words
1	1111	30	0.3125	11
2	1011	12	0.1250	100
3	0111	10	0.1042	010
4	1101	09	0.0938	001
5	1110	08	0.0833	000
6	0110	05	0.0521	0110
7	0000	04	0.0417	10101
8	0010	04	0.0417	10100
9	1001	03	0.0313	01110
10	0101	03	0.0313	01111
11	1100	02	0.0208	101110
12	0100	02	0.0208	101101
13	1000	02	0.0208	101100
14	1010	02	0.0208	101111
15	0011	0	0	UUUU
16	0001	0	0	UUUU
# states of FSM				15

Table 4.Huffman decoder output data.

1111	0101	0011	1111	1011	1110	1101	1011
1111	1111	1111	1111	0000	0000	0000	0000
1001	0010	0110	0111	1110	1101	0110	1110
1111	1111	1110	0110	1111	1001	1111	1011
1111	1111	0111	1010	1111	1111	0111	1111
0010	1110	1000	0100	1111	1111	1111	1111
0110	1011	1011	1011	1101	0111	1011	0111
1100	1000	1010	0111	0101	1011	1111	1101
1011	0100	1101	1101	1110	1111	1111	1111
1111	1011	0111	0101	1111	0111	1111	1101
1100	1101	1001	1110	1110	1101	1011	0110
0111	0010	1111	1111	0111	1111	1011	1111

4. SIMULATION RESULTS

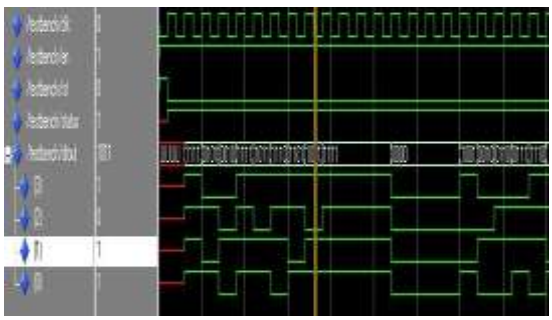

Figure 9.Input memory element

Figure 10. Occurrence SCalculator

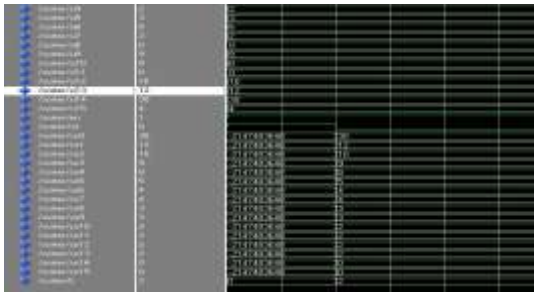

Figure 11.Occurrence calculator

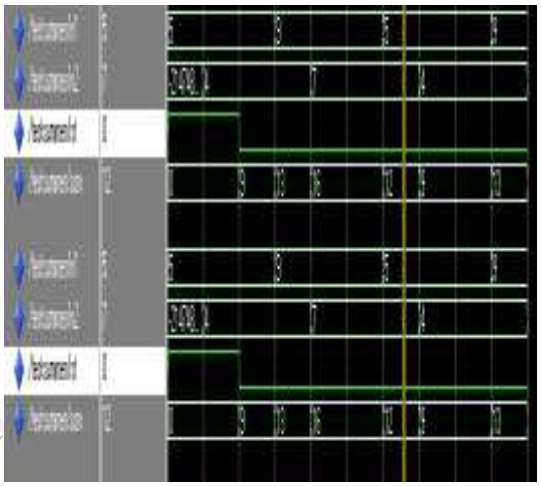

Figure 12. Adder sub module

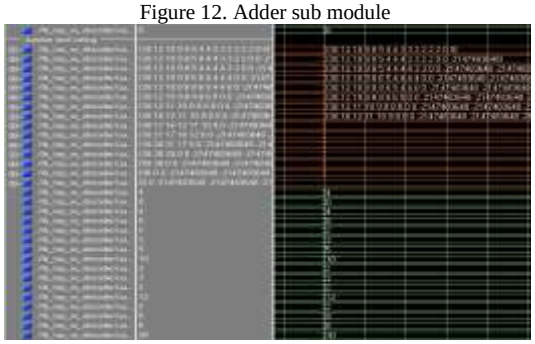

Figure 13.Code generator

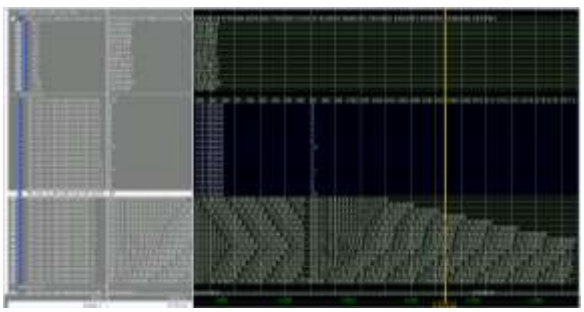

Figure 14.Huffman Encoder and Decoder

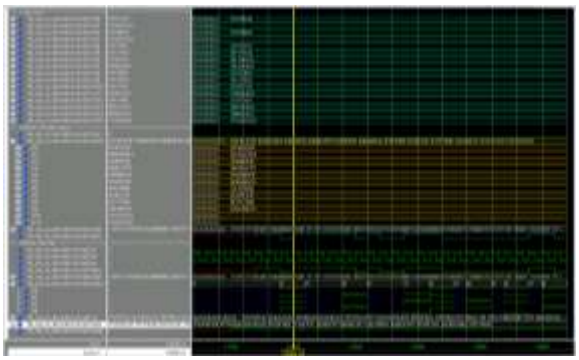

Figure 15.Huffman Encoder and Decoder

5. CONCLUSIONS

In this paper, we designed Huffman encoder and decoder using VHDL. ModelSim simulator tool is used for functional simulation and verification of the encoder & decoder modules. The Xilinx Synthesis Tool (XST) is used to synthesize the complete design on Xilinx family FPGA. Xilinx Placement & Routing tools will be used for backend, design optimization and I/O routing. Further work is to apply Huffman Coding in text, image, and video compression such as JPEG,MPEG2,etc.,and also used in digital compression devices. It is often combined with other coding schemes because of enhanced compression ratio and entropy.

6. REFERENCES

1) M. Y. Javed and A. Nadeem.: 'Data compression through adaptive huffman coding scheme'. In *Proceedings of TENCON 2000*, volume 2, pages 187.190, Sep. 2000.
2) MacKay, D.J.C.: 'Information Theory, Inference, and Learning Algorithms'. Cambridge University Press, 2003.
3) R. Hashemian,: 'Memory efficient and high-speed search Huffman coding'. IEEE Trans. Commun. 43 (1995) 2576- 2581.
4) D.A. Huffman.: 'A Method for the Construction of Minimum-Redundancy Codes'. Proceedings of the I.R.E., September 1952, pp 1098–1102.
5) XILINX Documentation, System Generator User Guide, Version 8.1.
6) Kesab K. Parhi. 1992.: 'High-speed VLSI Architectures for Huffman and Viterbi Decoders'.IEEE TRANSACTION ON CIRCUITS AND SYSTEMS-11: ANALOG AND DIGITAL SIGNAL PROCESSING. 39, no. 6 (JUNE): 385-391.
7) Richard W. Hamming, (1986).: 'Coding and
8) Information Theory'. New Jersey: Prentice-Hall.