VARIABLE LENGTH DECODER FOR STATIC HUFFMAN CODE

Mrs. T.G.Panse
Department of Electronics Engg.
Yeshwantrao Chavan College of Engg.
Nagpur, India
tejaswini.deshmukh@gmail.com

Abstract— **The proposed work is aimed at designing Variable-Length Decoder for Huffman code. Using the decoder we can obtain the original data from the compressed data .The system is designed using VHDL and the source code can be targeted to any FPGA for several applications. sound, voice, image compressions are frequently used in a real time application such as the broadcasting, the Variable-Length Decoder can be designed for a real time application. Emphasis has been given to design a simple working solution with minimum possible circuitry.**

***Keywords*—Huffman encoder, Variable length decoder, VHDL**

I. Introduction

In coding theory a variable-length code is a code which maps source symbols to a *variable* number of bits.Variable-length codes can allow sources to be compressed and decompressed with *zero* error and still be read back symbol by symbol. With the right coding strategy an i.i.d. source may be compressed almost arbitrarily close to its entropy. This is in contrast to fixed length coding methods, for which data compression is only possible for large blocks of data, and any compression beyond the logarithm of the total number of possibilities comes with a finite probability of failure. Some examples of well-known variable-length coding strategies are Huffman coding, Lempel–Ziv coding and arithmetic coding

CLASSES OF VARIABLE LENGTH DECODER

Variable-length codes can be strictly nested in order of decreasing generality as non-singular codes, uniquely decodable codes and prefix codes. Prefix codes are always uniquely decodable, and these in turn are always non-singular.

II INTRODUCTION TO HUFFMAN ENCODER

In computer science and information theory, Huffman coding is an entropy encoding algorithm used for lossless data compression. The term refers to the use of a variable-length code table for encoding a source symbol where the variable-length code table has been derived in a particular way based on the estimated probability of occurrence for each possible value of the source symbol. Huffman coding uses a specific method for choosing the representation for each symbol, resulting in a prefix code that expresses the most common characters using shorter strings of bits than are used for less common source symbols. Huffman was able to design the most efficient compression method *of this type*: no other mapping of individual source symbols to unique strings of bits will produce a smaller average output size when the actual symbol frequencies agree with those used to create the code. Huffman coding is equivalent to simple binary block encoding, e.g., ASCII coding. Huffman coding is such a widespread method for creating prefix codes that the term "Huffman code" is widely used as a synonym for "prefix code" even when such a code is not produced by Huffman's algorithm. Although Huffman coding is optimal for a symbol-by-symbol coding with a known input probability distribution, its optimality can sometimes accidentally be over-stated. For example, arithmetic coding and LZW coding often have better compression capability. Both these methods can combine an arbitrary number of symbols for more efficient coding, and generally adapt to the actual input statistics, the latter of which is useful when input probabilities are not precisely known or vary significantly within the stream**.** Our design target is for decoding an image Figure 1 shows a one example of the black and White image data. Each pixel is organized by 3 bits. Then there are 8 levels of the pixel intensity.

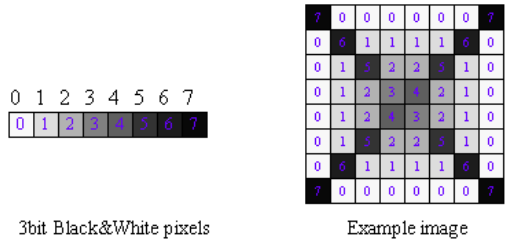


Figure 1. Black and White Image

The left figure in the figure 1 is 8 kinds of pixel intensities. The center number is the level. The most bright pixel is 0. The larger the number increases, the darker the pixel gets. Since there are 8 levels, 3 bits can express the levels. For decoding coloured images we can convert various types of images into grayscale images by using MATLAB software .

HUFFMAN CODING OF IMAGE DATA

The table 4 shows a example to apply the Huffman code to this image. Most frequently appeared pixel number of 0 corresponds to the code '1'. In total, the cross mark image can be expressed in 168 bits. Then the compression ratio is 168/192=87.5%.

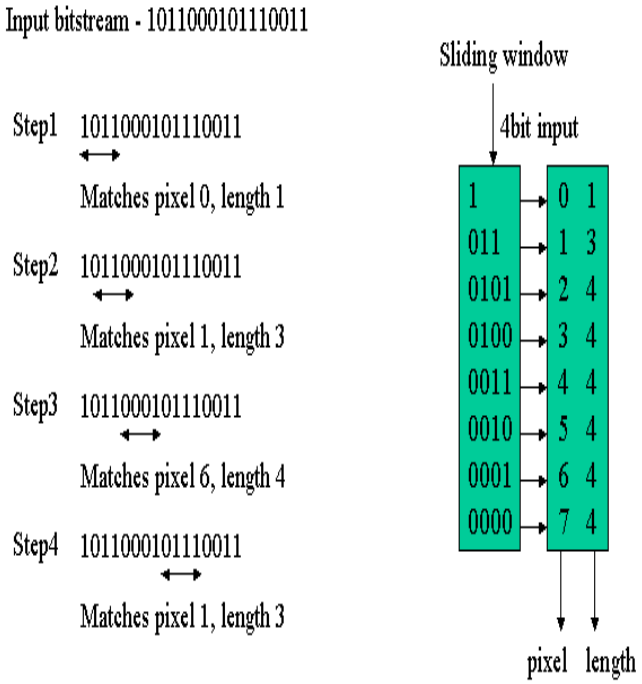
Pixel Number	Frequency	Huffman Code	Code length	Total bits
0 (white)	24	1	1	24
1	16	011	3	48
2	8	0101	4	32
3	2	0100	4	8
4	2	0011	4	8
5	4	0010	4	16
6	4	0001	4	16
7 (black)	4	0000	4	16
				Total 168bit

Table 1. Huffman Coded Cross Image

Table 2. Pixel Number and Huffman Code of Crossmark

Pixel Number								Huffman Code							
7	0	0	0	0	0	0	7	0000	1	1	1	1	1	1	0000
0	6	1	1	1	1	6	0	1	0001	011	011	011	011	0001	1
0	1	5	2	2	5	1	0	1	011	0010	0101	0101	0010	011	1
0	1	2	3	4	2	1	0	1	011	0101	0100	0011	0101	011	1
0	1	2	4	3	2	1	0	1	011	0101	0011	0100	0101	011	1
0	1	5	2	2	5	1	0	1	011	0010	0101	0101	0010	011	1
0	6	1	1	1	1	6	0	1	0001	011	011	011	011	0001	1
7	0	0	0	0	0	0	7	0000	1	1	1	1	1	1	0000

VARIABLE LENGTH DECODER ALGORITHM



STEP 1: In the algorithm, 4 bit-wide sliding window is used. The 4 bit data corresponding the sliding window is compared with the righthand side table. The top of the table is '1'. The MSB of the input 4 bits is '1' then match the top of the table in spite of the other 3 bits and outputs the pixel number of '0' and the code length of 1.

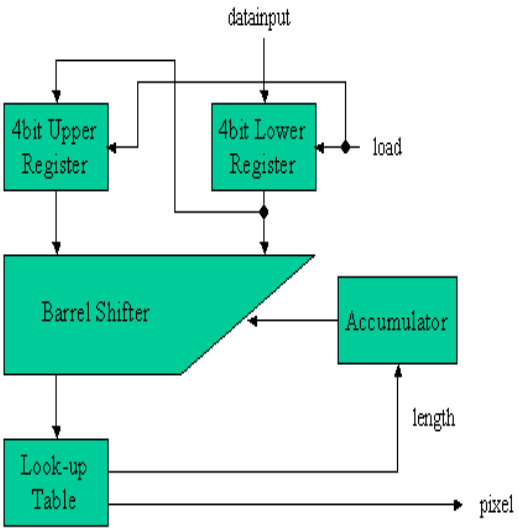
STEP 2 : In the STEP1, the code length is 1. Then shift the sliding window by 1 bit then get the 4 bit inputs corresponding the new sliding window and proceed the same procedure as STEP1. In the example, the new 4 bits is '0110' then the MSB 3bits will match '011'. As a result, pixel number of 1 and the code length of 3 will be obtained.

STEP 3

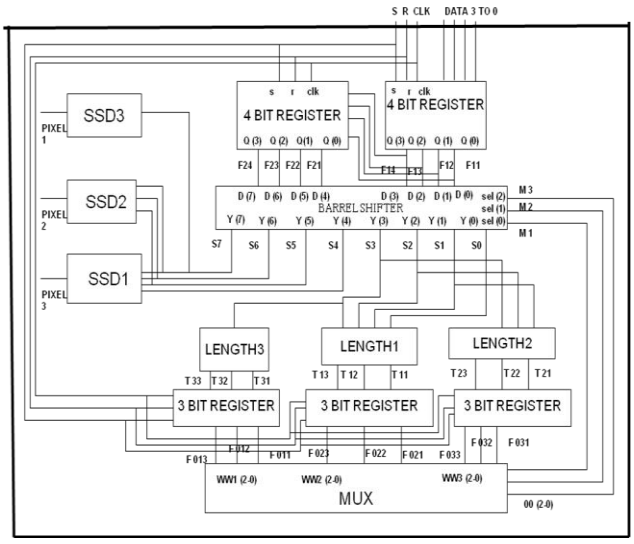
In the STEP2, the code length was 3. Then shift the sliding window by 3 bits then get the next 4 bits inputs. And repeat the same operation.

ARCHITECTURE OF VARIABLE LENGTH DECODER

In this architecture, two sets of 4 bit registers are included. The 4 bit width sliding window will be inside of this 8 bit width. Then the 4 bit inputs corresponding the sliding window has to be compared with the Look-up table.The look-up table outputs both the pixel number and the code length. According to the code length, sliding window has to be shifted. This shift operation is realized by the accumulator and Barrel Shifter.The timing critical path will be Barrel Shifter -> Look-up Table -> Accumulator -> Barrel Shifter. In order to achieve the higher clock rate, circuit optimization such as pipelining will be necessary.We have used the paper titled



Block Diagram of Variable Length Decoder



The Variable Length Decoder consists of 4-Bit Registers , Barrel Shifter , Seven Segment Display ,3 Bit Register and Multiplexer.

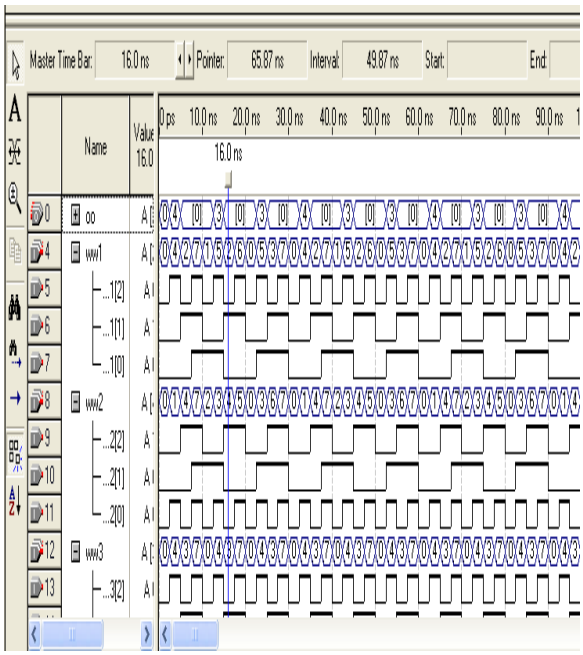
The Huffman coded image consisting of a bit stream is fed to the two 4-Bit registers .The registers stores the code and the output of both the registers is given to the Barrel Shifter. The Barrel Shifter is used to shift the bit stream as per our requirement. Since the output of both the

4-Bit registers is the same the first four bits and the last four bits of the Barrel Shifter are identical. The first four bits are used to identify the pixel no and the last four bits are used to detect the length of pixel. There are three seven segment displays i.e SSD 1 , SSD 2 , SSD 3.

SSD 1 detects all pixels having bit length 1 and displays the pixel no. Similarly SSD 2 and SSD 3 detect pixels having bit length 3 and 4 respectively. The Length 3 block detects 1 bit length, Length 2 block detects 3 bit length and Length 1 block detects 4 bit length . This length is stored in the 3- Bit Register. The output of each of the 3 Bit registers is given to the Multiplexer. The Multiplexer will select the length and gives the output to the select lines of the Barrel Shifter .

The Barrel Shifter detects the length and shifts the bit stream by that length. The process continues till the entire bit stream has been decoded and each pixel has been identified. Thus once all the pixels have been identified we get the entire image.

VHDL Simulation Result for Barrel shifter



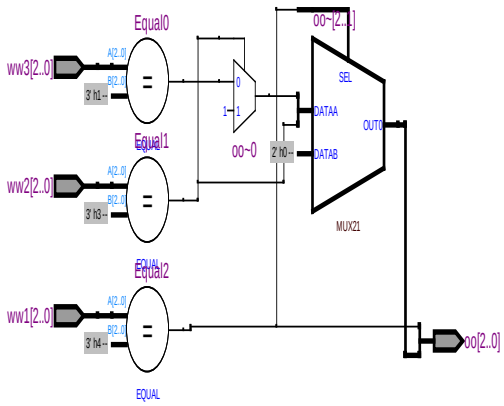
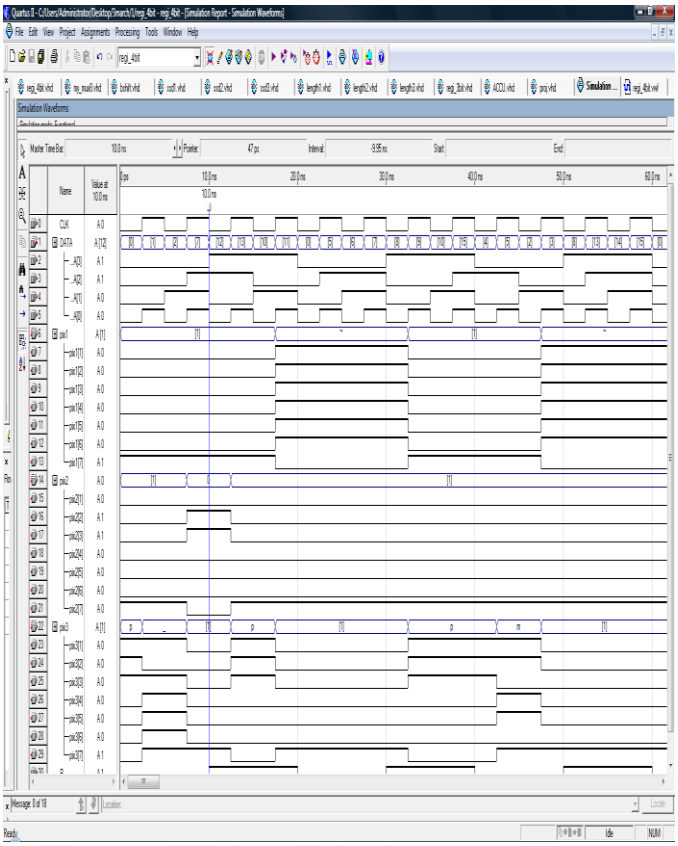


Figure 2 RTL View of Barrel Shifter

VHDL Simulation result for Variable length decoder



CONCLUSION

Thus the variable length decoder has been designed successfully by VHDL .The main advantage of using Variable length decoder is that data can be transmitted by using lesser number of bits. This in turn results in less utilization of hardware and quicker and more efficient transmission of data. Although this is a simple form of a decoder it can be suitably modified to decode different types of images. To generalize the above designed decoder we can decode different types of images by converting them into grayscale images using MATLAB software. Future scope of this Variable Length Decoder is we can decode images having higher no of pixel intensities by suitably modifying the design.

Also the variable length decoder can be designed so as to be able to decode large amount of information and can be used in motion picture technology.

REFERENCES

[1] C.-Y. Chang and K. Yao, “Systolic array processing of the Viterbi algorithm,” *IEEE Trans. Inf. Theory*, vol. 35, no. 1, pp. 76–86, Jan. 1989.

[2] C. G. Caraiscos and K. Z. Pekmestzi, "Low-latency bit-parallel systolic VLSI implementation of FIR digital filters," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 43, no. 7, pp. 529–537, Jul. 1996.

[3] V. Boriakoff, "FFT computation with systolic arrays, a new architecture," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 44, no. 4, pp. 278–284, Apr. 1994.

[4] I. K. Proudler, J. G. McWhirter, M. Mooner, and G. Hekstra, "Formal derivation of a systolic array for recursive least squares estimation,"
IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process., vol. 43, no.
3, pp. 247–254, Mar. 1996.

[5] S. Swaminathan, R. Tessier, D. Goeckel, and W. Burleson, "A dynamically reconfigurable adaptive Viterbi decoder," in *Proc. FPGA'02*, Monterey, CA, Feb. 24–26, 2002.