Parallel Huffman Decoder with an Optimize Look UP Table Option on FPGA

By Zulfakar Aspar, Zulkalnain Mohd Yusof, Ishak Suleiman
Faculty of Electrical Engineering, Universiti Teknologi Malaysia

Abstract

Compression is very important for system with limited channel bandwidth and/or limited storage size. One of the main components in image/video compression is a variable length coding (VLC). This paper would discuss about one of the most popular VLC known as Huffman Coding. In our present work, a real time hardware parallel Huffman decoder has been successfully designed and implemented using 50,000 gate FPGA (FLEX10K20 from Altera). The parallelism is important to be exploited in the design to achieve the high frame rate such as in JPEG and MPEG implementation. Using parallel technique, a codeword is guaranteed to be processed within a single clock cycle. The codeword to be processed is matched with the one stored in a Look Up Table (LUT). A LUT is needed during Coder and Decoder process. In order to saves memory cost, an optimize LUT is suggested. This paper does not intend to complete an optimized operating speed design, instead this paper only concentrate on producing a workable real-time decoder design.

1. Introduction

Image/video compression is one of the major components used in video-telephony, videoconferencing and multimedia-related applications. Compression allows efficient utilization of channel bandwidth and/or storage size. One of the main components in image/video compression is entropy coding. The most important part in entropy coding is variable length coding (VLC). One of the most well known VLC is Huffman Coding. Using Huffman coding, lossless compression is achieved by exploiting statistical redundancy that exist in the data. In order to get a high throughput, the Huffman algorithm is implemented in hardware.

This paper is organized as follow. Section 1 is the introduction on Huffman Coding. Section 2 is a brief explaination on Huffman algorithm. Section 3 is the bit serial Huffman decoder implementation. Section 4 is the bit parallel Huffman decoder implementation. Section 5 is the incoperation of Huffman Decoder into JPEG Huffman Decoder. Section 6 is the optimization of the Look Up Table to reduce the hardware consumption. was implemented. Section 7 is the conclusion of this paper. The design work done in this paper does not target for an optimum clock speed. Instead, a minimum acceptable working design is presented. As a comparison, the effect of using different type of FPGA's on the operating speed performance, Flex10K20RC240-3 and Flex10K20RC240-4 were presented. Combining with the FPGA usage the design was successfully done in a short development cycle.

2. Huffman Algorithm

Shanon entropy defines that any given set of raw data can be arranged in a more efficient manner provided if the probability of each data symbol is known. The basic idea in Huffman algorithm is to assign a shorter codeword for a more frequent (probability) data and to assign a longer codeword for a less frequent data. Following Huffman algorithm, a set of data sample is prepared as shown in table 2.

One of the main problems in Huffman Decoder is due to the varying length of the coded codes. Therefore, if the Huffman Decoder is implemented using bit serial technique, the number of cycles required to decode a symbol varies depending on the codelength. Therefore, bit serial technique is slow for a real time implementation. Another major problem is a single bit error propagation. If any of the bits is corrupted by noise during transmission, the entire group of data become useless. Possible way of limiting the error propagation error is to insert synchronous code at certain interval.

Table 2: The Huffman Codes for a set of sample of Fix Length Codes

Code	Fix Length Code	Pn	VLC (Huffman)	Total Bit = Pn X Bit Length
A	0 0 0	22	0 0	44
B	0 0 1	8	0 1 0 1	32
C	0 1 0	15	0 1 1	45
D	0 1 1	33	1 0	66
E	1 0 0	4	0 1 0 0 1	20
F	1 0 1	16	1 1 0	48
G	1 1 0	2	0 1 0 0 0	10
H	1 1 1	20	1 1 1	60
Total Occurrence = 120			Total All Bit = 325	

3. Bit Serial Huffman Decoder

The simplest and easiest way to design the serial Huffman decoder is to decode the data bit by bit or in serial fashion (manner). Using Moore model, the decoder can easily be constructed. Given any set of Huffman codes, the finite state machine can be built by

0-7803-6355-8/00/ $10.00 © 2000 IEEE

treating each node (which is assigned by 1 or 0) as a different states. The branches are treated as the path to the different states or the path that the states need to follow. The sample data in table 2 has a Huffman tree as in figure 3. Then based on the guidance given and figure 3, the state diagram can be constructed.

To speed up design time, VHDL technique was choose to build the circuit. The main problem with this method is if the probability of the symbols change, the Huffman tree also changes. Therefore, the design must be changed. Another major problem is the number of clock cycles to finish decode a codeword depends on the length of the codeword. This will cause inconsistency bit rate which require additional hardware to solve the problem.

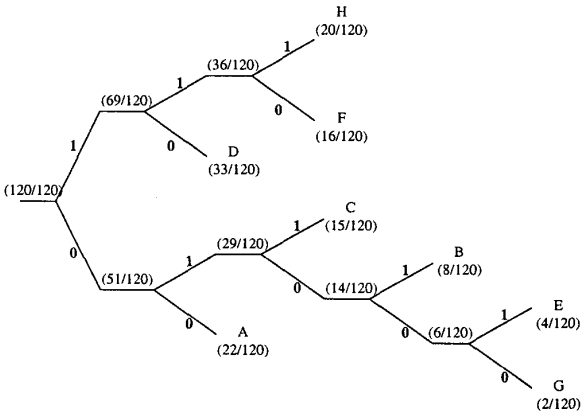

Figure 3: The Huffman tree for sample data in table 2

Although bit serial design consumes little hardware (3 flip-flops only for this example), the speed is very poor for a real time implementation. Thus, bit serial Huffman is not suitable for a real time implementation. A better alternative is to use bit parallel Huffman coding.

4. Bit Parallel Huffman Decoder

The bit parallel technique used is based on the work done by Shaw-Min Lei, and Ming-Ting Sun [2]. The advantage of this technique is a decoded code or a codeword is produced in every clock cycle regardless of the length of the Huffman codes. The increase in hardware complexity is justified by the speed gained. The basic idea of this method is to find the matching coded codeword with the stored decoded codewords in a Look Up Table (LUT). The length of the data input stream is fixed such as 8 bits. As shown in table 2, 8 bits length of data input stream may contain more than one coded codeword. Therefore, a buffer is needed to hold the input data stream. Another purpose of using the buffer is to ensure that the stored data can be shifted to the correct position after a codeword was decoded. If there is no more uncoded codeword, the buffer will receive new data stream and the process above is repeated. As a result, more than one cycle may be needed to decode a data stream, which may contain more than one coded Huffman codeword (decoded on every cycle). In addition, for storing the match coded codeword with the decoded codeword, LUT also contains the length of the coded codeword (codelength). Therefore, after a coded codeword is decoded, the length of the coded codeword will be accumulated by the Accumulator. The Accumulator has a dual purposes: First is to point to the correct location for the data input stored in Buffer before the data can be decoded. This is done by accumulating the codelength as every time a codeword is decoded. Second is to trigger Buffer to load a new data input if all coded codeword in previous data input were decoded. The Look-up table (LUT) consists of data pointer and ROM/RAM/PLA, which consists of a table of, predetermine Huffman codes[2]. For example,

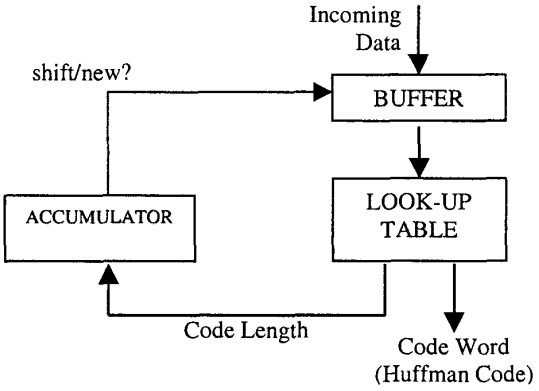

Figure 2: The Bit Parallel Decoder technique is important to speed up the decoding process

a first stream of data consists of a total of 8 bit stream of 00100110. Initially, the Accumulator is zero. Thus, the decoded codeword is A (000) or 00 with total codelength as two. On the second cycle, the Accumulator is two, forcing the Buffer to shift by two bits. Thus, the decoded codeword is D (011) or 10 with total codelength as two. On the third cycle, the Accumulator is four, forcing the Buffer to shift by four. Thus, the decoded codeword is C (010) or 011 with total codelength as three. On the fourth cycle, the Accumulator is seven while the remaining bit one. Combining this remaining bit with a second 8 bit stream of data (10010101) which is stored in the second Buffer layer, the LUT will see the shifted by seven bit data as 01001010. Thus, the decoded codeword is E (100) or 01001. On the fifth cycle, the Accumulator is twelve, which is more than eight. As a result, the Accumulator will be deducted by eight to point to the next undecoded data. The data in the second layer Buffer is put into the first layer Buffer and the second layer Buffer receives a new stream of 8 bit data. These are all done in the same cycle.

The example above shows the input data stream may need more than one cycle to be decoded while decoded data is always available on every clock cycle. Any changes to the Huffman Code can be adjusted by changing the LUT content.

The operating speed of the Bit Parallel Huffman Decoder by using Altera FPGA Flex10K20RC240-4 is 9.24MHz. If Altera FPGA Flex10K20RC240-3 is used, the decoder operating speed is 10.89MHz. Both FPGA's total logic cells utilization is 37% or 436 over 1152 logic cells.

5. Bit Parallel JPEG Huffman Decoder

The successful design of the bit parallel Huffman Decoder was incoperated in JPEG Decoder standard. However, Parallel Huffman Decoder produces a decoded codeword in every clock cycle whereas in JPEG Huffman Decoder, the output of the decoder is alternated between Huffman Codeword and Coded Amplitude as shown in Table 5.1 [3], [4]. The Huffman Codeword is

Table 5.1: The difference between Huffman Decoder and JPEG Huffman Decoder

Decoder	Cycle	Data Output
Huffman	1	Huffman Codeword
	2	Huffman Codeword
JPEG Huffman	1	Huffman Codeword
	2	Coded Amplitude

the fixed length codes or the codes before compressed. The Coded Amplitude is the value of a DCT coefficient. Therefore, the original Huffman Decoder design must be modified to decode the respective Huffman Codewords and Coded Amplitudes. The detail of the actual JPEG Decoder design can be referred to [3] and [4].

According to JPEG standard, the decoded Huffman Codeword consists of number of Run and Category. Number of Run is the number of zeros between coefficient or the coefficient location. Category is the number of bits assigned for the Amplitude or Amplitude bit length. Amplitude is the value of DCT coefficient. The Amplitude bit length tells how much the data in the Buffer should be shifted due to Amplitude Length. Hence, in the first cycle the accumulator will accumulate the CodeLength while in the second cycle the accumulator will accumulate the Amplitude length.

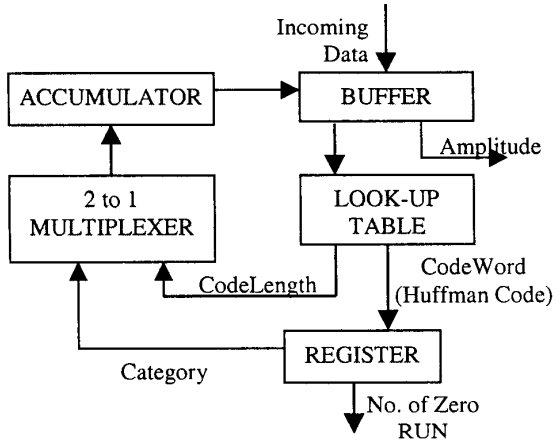

Figure 5: The modified Bit Parallel Huffman Decoder to follow JPEG standard

The operating speed of the JPEG Huffman Decoder by using Altera FPGA Flex10K20RC240-4 is 9.30MHz. If Altera FPGA Flex10K20RC240-3 is used, the decoder operating speed is 10.95MHz. Both FPGA's total logic cells utilization is 47% or 547 over 1152 logic cells.

6. Optimize Look Up Table

From experience above, it was found that the larger the LUT, the more efficient the compression will be. However, it is impossible to keep the LUT as large as possible without adding extra cost. In JPEG Huffman Decoder LUT mention before, the input bit width is restricted to 16 bit length (refer to JPEG Luminance table). As a result, the Huffman Codewords can be divided into two main groups separated by leading zeros and leading ones. Using this knowledge, the LUT size can further be reduced. Instead of using a 262 entries (17 bits input wide) times 13 bits width per entry, the LUT can reduced to 262 entries (10 bits input wide) times 13 bits width per entry. The only addition hardware is a leading zeros or leading ones detector so that the buffer can be adjusted to the correct position.

Even though, the total logic get consumption increases, the size of the memory used is drastically reduced. The only problem encounter in this optimization method is the stable time for the buffer to position itself at the right position increases. The modified LUT is shown in Figure 6.

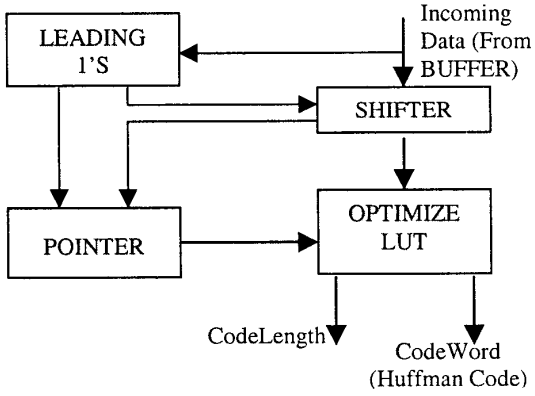

Figure 6: The stable time increases as an additional layer of data Shifter/Positioner is needed.

When a data stream arrives from Buffer, the Shifter will reposition itself into a new position based on the information detected by Leading 1's Detector (L1D). As a result, the shifter can only reposition itself after L1D has stable. Hence, additional time is needed. The Pointer is used to select the right MSB bits of the LUT if needed. The combination of the Pointer and the Shifter will ensure the optimize LUT is decoded at the correct position.

The operating speed of the Bit Parallel JPEG Huffman Decoder by using Altera FPGA Flex10K20RC240-4 is 9.91MHz. If Altera FPGA Flex10K20RC240-3 is used, the decoder operating speed

is 11.54 MHz. Both FPGA's total logic cells utilization is 99% or 1145 over 1152 logic cells.

7. Conclusion

The Parallel Huffman Decoder is a very important technique to ensure that decoded data is produced in every clock cycle. An example of Parallel Huffman Decoder application shows that the decoder is easily modified to customize for a dedicated application. Further work proves that with another small modification, the total memory consumption to build LUT can be further reduced with small additional logic gets. Preliminary results show that the optimize LUT performance is good enough to be implemented in any application such as JPEG Decoder. The speed may be further improved by designing a customize IC using VLSI technology.

Further work is to integrate this technology into a complete compression standard such JPEG Decoder, MPEG or MPEG II Decoder, MP3 Decoder and others.

REFERENCES

[1] Richard W. Hamming, (1986) Coding and Information Theory. New Jersey: Prentice-Hall.

[2] Shaw-Min Lei, and Ming-Ting Sun. "An Entropy Coding System for Digital HDTV Applications.", IEEE TRANSACTION ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY. 1, no. 1, 147-154, Mar 1991.

[3] M. Kovac, N. Ranganathan. "JAGUAR: A Fully Pipelined VLSI Architecture for JPEG Image Compression Standard", Proceedings of the IEEE, 83, no.2, Feb 1995.

[4] Kesab K. Parhi. 1992. "High-Speed VLSI Architectures for Huffman and Viterbi Decoders." IEEE TRANSACTION ON CIRCUITS AND SYSTEMS-II: ANALOG AND DIGITAL SIGNAL PROCESSING. 39, no. 6 (JUNE): 385-391.

[5] Peter Pirsch, Nicolas Demassieux, and Winfried Gehrke. 1995. "VLSI Architectures for Video Compression – A Survey." Proceeding of IEEE, 83, no. 2 (February): 220-244.

[6] Peter A. Ruetz, Po Tong, Douglas Bailey, Daniel A. Luthi, and Peng H. Ang. 1992. "A High-Performance Full-Motion Video Compression Chip Set." IEEE TRANSACTION ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY. 2, no. 2 (JUNE): 111-133.

[7] "Altera Megafunction Partners Program", http://www.altera.com