A Memory-Based Architecture for Very-High-Throughput Variable Length Codec Design*

Yew-San Lee Jin-Jer Jong Tsyr-Shiou Perng Li-Chyun Hsu Ming-Yang Jaw Chen-Yi Lee

Institute of Electronics Engineering, National Chiao Tung University,

1001, Ta Hsueh Road, Hsinchu, 300, Taiwan, R.O.C. China

Tel: 886-3-5712121 ext. 54238; E-mail: mountain@royals.ee.nctu.edu.tw

Abstract

Variable-length code/decode (VLC/VLD) is the most popular data compression technique which can reduce the storage and communication channel bandwidth needed to transmit a large amount of data. In this paper, we present a new memory-based VLSI architecture for VLC/VLD codec system* . Both coding and decoding procedures are mapped onto a memory which has been minimized by using a two-bit structure. The proposed architecture mainly consists of memory and simple arithmetic unit, making it very suitable for VLSI implementation. Simulation results show that based on 0.35um CMOS process, both compression rate and decompression rate up to 1.2Gbits/s can be achieved.

1. Introduction

Many applications and image processing standards use lossless coding at the end of the encoder to obtain high compression ratio. In addition, lossless coding can recover exact copy of the original data from the compressed data. Several lossless coding schemes have been proposed for these purposes. Every source data has some finite information content called the entropy of the source. This entropy is the limiting factor for lossless data compression. Once the entropy of the source is reached, no further compression is possible without some loss of information. The variable-length-code (VLC), also called the Huffman code [1], is an optimal code with the average codeword length approximating the source entropy. The idea is to assign shorter codewords to high probability symbols and longer codewords to low probability symbols. The procedure is shown in Fig.1 with a codebook size of 4. The total average length of the code can be minimized.

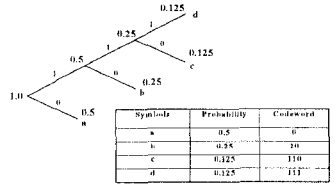

Fig. 1. An example of the VLC code.

In recent years, several special-purpose VLSI architectures have been proposed to implement VLC/VLD codec system. There are two classes of architectures, namely the tree-based and the PLA-based architectures, have been discussed in the literature. A class of PLA-based architectures have been proposed in [4], [5], [13]. This method can increase throughput by using parallel and pipelined architectures. But the flexibility is poor since it hardwires the code in hardware allowing only one code to be implemented. For large codebook sizes, this approach will not be very practical. The speed will be degraded a lot by the nature of the PLA architectures. A set of tree-based architectures have been discussed in [2], [3], [7], [10]. It has good flexibility and allows the codebook to be stored in an on-chip memory so that the codebook can be changed by users to meet the needs of various applications. In addition, this approach can achieve high speed operation even when codebook size becomes large.

The motivation behind our research is to develop a high-throughput VLSI architecture for the VLC/VLD codec system. In this paper, we propose a new scheme for mapping Huffman tree onto memory which leads to an area-efficient solution. We break the recursive dependence among the tree search steps which will increase the throughput effectively. We also design (fully-custom) a scalable inout bandwidth synchronous SRAM to achieve more flexibility and efficiency. For the control unit and arithmetic unit, an in-house 3-V cell library is exploited to generate the final layout.

The organization of this paper is as follows. In section 2, we first present the encoding algorithm, memory mapping method and hardware architecture. After that, we will describe the decoding part in section 3. Finally, we integrate the above two parts into a codec system which will be discussed in section 4. Concluding remarks are made in section 5.

2. Encoding algorithm and architecture

Huffman code is a tree-based code, it can be represented by tree structure such as binary trees, two-trees, quad trees, etc. The Huffman tree is an unbalanced tree since no code can be a prefix of any other code. If we use high order tree to represent it, more memory space and address will be wasted to store the unused node information. As a result, we choose two-bit tree [9], [10] structure to map the Huffman code onto a memory in our codec system. For encoding, the internal nodes are labeled top-down from root to terminal level and within each level the nodes are labeled from left to right as shown in Fig.2.

* WORK SUPPORTED BY THE NATIONAL SCIENCE COUNCIL OF TAIWAN, ROC, UNDER GRANT NSC86-2221-E-009-016

0-7803-3583-X/97 $10.00 ©1997 IEEE

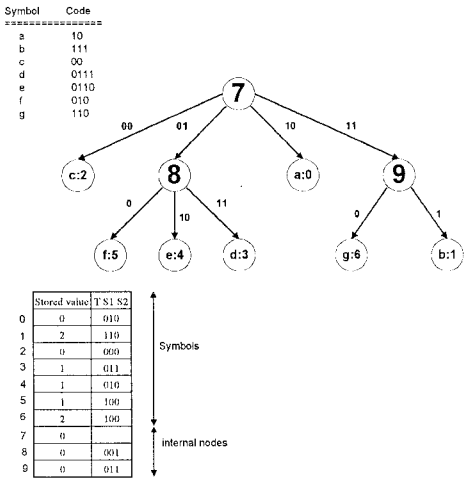

Fig 2 : Two-bit tree for encoding

Assume there have n symbols and m internal nodes in the tree. The pointer of the i-th symbols (terminal nodes) are stored in the i-th location using log n bits. After that, the pointer of the i-th internal nodes are stored in location $n+i$ sequentially. The stored value is [label of the parent node - n]. We only need to store the subtracted result, and then save the memory space especially for the tree which has many terminal nodes. Besides, three additional bits (T, S1, S2) are added for each node. They will be used to control the encoding steps. Both S1 and S2 bits contain the encoding result (label of the edge between the two nodes). Because some edges have 1-bit result and some have 2-bits result, we use T bit to indicate these two cases, i.e. T=0 for two bits and T=1 for one bit. The encoding algorithm flow chart is shown in Fig. 3.

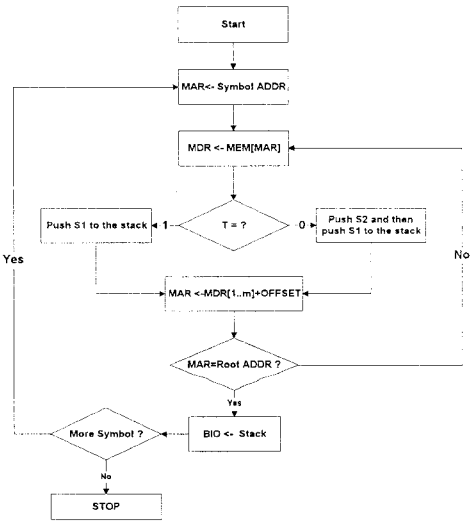

Fig 3 : Encoding algorithm flow diagram

Since the synchronous SRAM is pipelined to one stage, we can get the next node information at the next positive edge of the clock. Simultaneously, the encoding architecture uses this information to calculate next parent node pointer address. These actions are processed until the root of the tree is reached. The encoding system architecture is shown in Fig. 4.

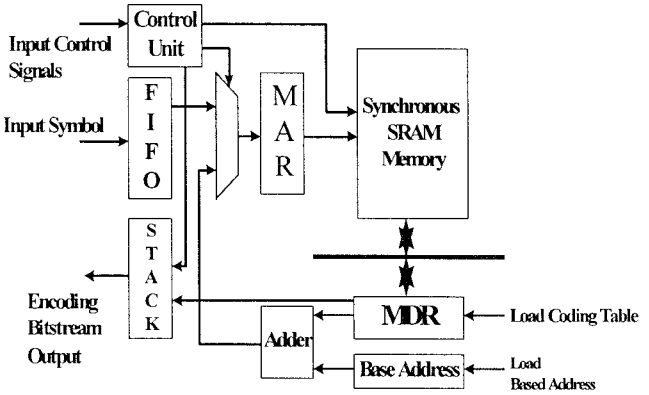

Fig 4 : The encoding system architecture

We will use a barrel shifter to control the codec system I/O interface. It will transform the sequential output format to parallel output format. As a result, we don't need to do "pop" action on the stack. And hence the encoding steps and cycle time can be reduced. The most area consuming part in the codec system is the memory, which is a key issue to reduce its requirement [8]. For the same Huffman code, our algorithm can save about 25% and 10% memory space compared to Ref.[2] and [3] respectively. In addition, we can merge the internal nodes information and decrease the memory space again. Assume S-bits coding symbols, C % compression ratio and f Hz clock rate, the comparison result of throughput are listed as bellow :

$$\text{ref [2]} : \quad \frac{f}{(\frac{S \times C}{2} \times 3)+1} \times S \quad \text{Mbps}$$

$$\text{ref [3]} : \quad \frac{f}{(S \times C) \times 2} \times S \quad \text{Mbps}$$

$$\text{our system} : \quad \frac{f}{(\frac{S \times C}{2})} \times S \quad \text{Mbps}$$

From above description, it can be found that our architecture's throughput is about four times of Ref [2] and [3].

3. Decoding algorithm and architecture

In the decoding algorithm, we also use two-bits Huffman tree to map the Huffman code onto a memory. We merge the child nodes of the same parent node into a "Loc" node as shown in Fig. 5. The stored values are four sets of "T S1 S2" data. These control bits assignment for decoding are listed as bellow :

	T	S1	S2
Case 1 : Regular Node	0	0	0
Case 2 : Special Node			
i) Two 1-bit labels	0	1	1
ii) One 1-bit label = 0	0	1	0
iii) One 1-bit label = 1	0	0	1
Case 3 : Terminal Node	1	0	0
Case 4 : Unused Node	1	1	1

We search from the symbol nodes to the root of the tree in the encoding process. In contrast, we search from the root of the tree to the terminal nodes in the decoding process. We use two memory modules to store the "Loc" values and symbols (codes) separately. Besides, we construct two register files "T" and "C". The *i*-th location of the "T" register file stores the total number of terminal nodes which appears from 0 to (*i* -1)-th "Loc" nodes. In the same way, the *i*-th location of the "C" register file stores the total number of nodes which have child nodes in 0 to (*i* -1)-*th* "Loc" node. Finally, we use these information to search the next "Loc" address and control the decoding steps. The decoding algorithm flow chart is given in Fig. 6.

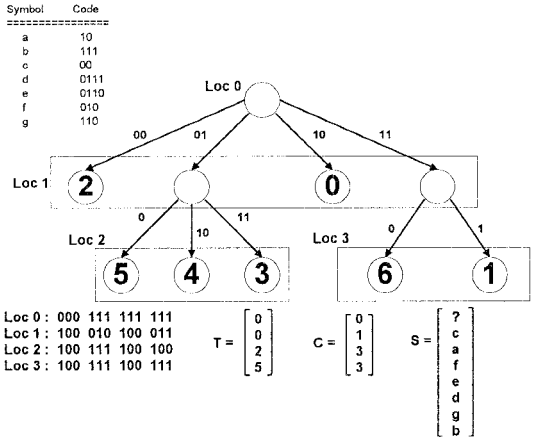

Fig 5 : Decoding tree with "Loc" nodes

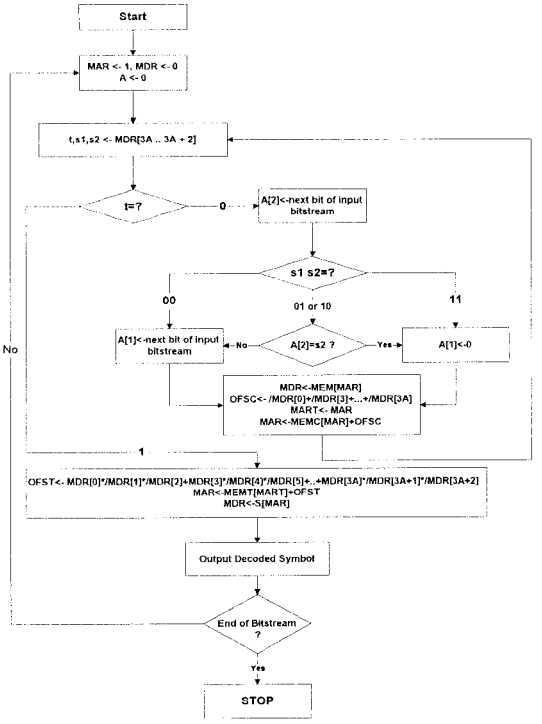

Fig 6 : Decoding algorithm flow diagram

We can use current "Loc" value and "C" value to calculate the next "Loc" node address before we get the next input stream data. As a result, we don't need to wait the input stream data for calculating next "Loc" node address. We break the recursive dependence of the searching steps. Thus the throughput rate can be enhanced effectively. Since we merge four child nodes into a "Loc" node, more amount of memory space can be saved for the tree which has many internal nodes (large code table size). Besides, we can continue to decode the next bit stream data and read the decode result (symbol code) from the symbols memory in parallel since "Loc" memory and symbols memory are two separate modules. It has additional latency of one cycle for the first decoded symbol. After that, we can decode the bit stream with a speed of 2bits per cycle by using the above parallel operations. The overall decoding system architecture is shown in Fig. 7.

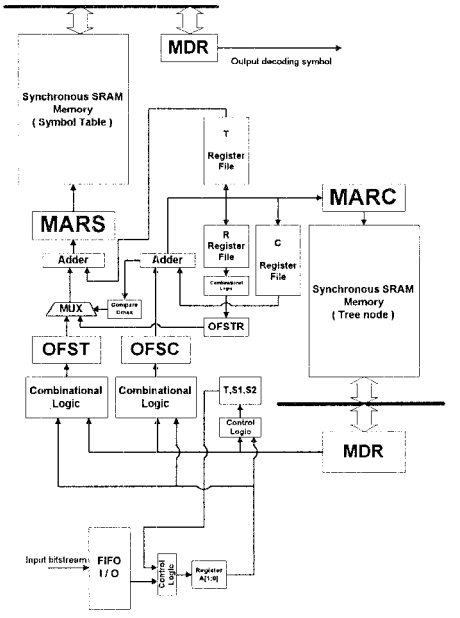

Fig 7 : The decoding system architecture

For the same Huffman code, our decoding algorithm can save about 15% memory space compared with Ref [2]. Assume under the same condition with encoding process, the comparison result of throughput are listed as bellow :

$$\text{ref [2]}: \quad \frac{f}{(\frac{S \times C}{2} \times 5)+1} \times S \quad \text{Mbps}$$

$$\text{ref [3]}: \quad \frac{f}{(S \times C)+1} \times S \quad \text{Mbps}$$

$$\text{our system}: \quad \frac{f}{(\frac{S \times C}{2})} \times S \quad \text{Mbps}$$

Our architecture's throughput is about five times of Ref [2] and 2 times of Ref [3] respectively.

4. Codec system integration

In the previous section, we discussed the encoding/decoding procedures and hardware architectures. These two systems are then integrated into a complete codec system here. The codec system architecture is shown in Fig. 8. We use a programmable control unit (FSM) [14] to determine the processing steps. It also does the system initialization. Firstly, it will load both relative mapping data into the memory modules and

some parameters into the register files. After that, it can do encoding or decoding process sequentially depending on the coding modes. In addition, we use barrel shifters to transform the input and output data formats. As a result, the I/O interface of the codec system is in parallel format. The codec system can be used for other tree-based data compression and decompression.

It is important to use pipeline technique and parallel architectures in the ALU unit for improving the operating speed [6], [11], [12], [13]. The codec system uses separate memory module and register files to store the different data. Besides, we can also control the inout bandwidth of the memory modules. These scalability will improve the memory efficiency. The codec system can handle two or more different sizes of Huffman codes concurrently through the programmable control unit and separate scalable bandwidth memory modules. It is possible to integrate two or more codec system into a "large" codec system which can be used to process multiple independent bit streams. The overhead is low and the complexity only increase linearly with the throughput improvement.

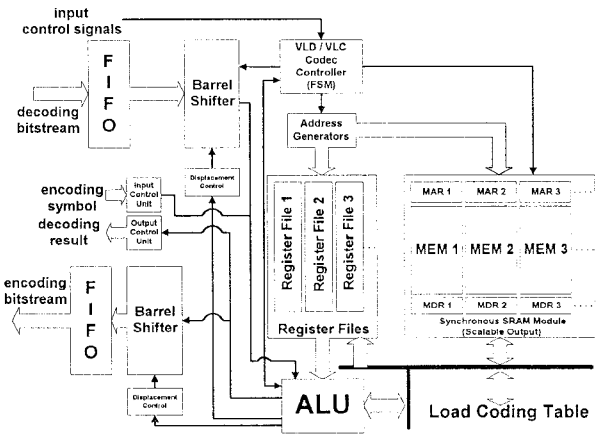

Fig 8 : Codec system archictecture

5. Conclusion

In this paper, we have presented an algorithm and architecture of a very high throughput VLC/VLD codec system. The architecture is based on an efficient scheme of mapping two-bit tree structure onto memory modules separately. It can be used to implement lossless variable-length coding as well as for the image compression standards and tree-base coding. The approach is well suited for large coding tables and high throughput codec system design. The codec system is now under construction by an in-house 3-V cell library and a synchronous SRAM memory with an access time of 2.4ns in 0.35um twin-well CMOS process. The codec system operates at 3V power supply with clock rate up to 300MHz. Assuming 50% compression ratio, simulation results show that our codec architecture can achieve compression and decompression rate of 1.2Gbits/s. This high throughput can meet the need of many high data rate applications.

References

[1] D.A. Huffman, "A method for the construction of minimum-redundancy codes," in *Proc. IRE*, no. 40, Sept 1952, pp. 1098 - 1101

[2] Amar Mukherjee, N. Ranganathan, Jeffrey W. Flieder, and Tinku Acharya, " MARVLE : A VLSI Chip for Data Compression Using Tree-Based Codes," *IEEE Trans. on Very Large Scale Integration (VLSI) System*, pp.203-213, 1993

[3] Heonchul Park and Viktor K. Prasanna, " Area Efficient VLSI Architectures for Huffman Coding," *IEEE Trans. on Circuits and System*, pp.568-575, 1993

[4] Keshab K. Parhi," High-Speed Huffman Decoder Architectures," *IEEE Trans. on Circuits and System*, pp.64-68, 1991

[5] Shih-Fu Chang and David G. Messerschmitt," Designing a High-Throughput VLC Decoder Part I – Concurrent VLSI Architectures," *IEEE Trans. on Circuits and Systems for Video Technology*, pp.187-196, 1992

[6] Horng-Dar Lin and David G. Messerschmitt," Designing a High-Throughput VLC Decoder Part II – Parallel Decoding Methods," *IEEE Trans. on Circuits and Systems for Video Technology*, pp.197-206, 1992

[7] A. Mukherjee, N. Ranganathan, and M. Bassiouni," Efficient VLSI design for data transformations of tree-based codes," *IEEE Trans. on Circuits and Systems*, pp.306-314, 1991

[8] G. Jacobson," Space-efficient static trees and graphs," *IEEE Symp. on Foundations of Computer Science*, pp.549-554, 1989

[9] A. Bassiouni and A. Mukherjee," Multibit and multigroup techniques for data compression," Univ. of Central Florida, Aug. 1992, private communication

[10] A. Mukherjee, H. Bheda, and T. Acharya," Multibit decoding / encoding of binary codes using memory-based architectures," in *Proc. Data Compression Conf.*, Snowbird, UT, pp.352-361, 1991

[11] N. Ranganathan and S. Henriques," A parallel architecture for data compression," in *Proc. IEEE Int. Symp. on Parallel and Distributed Processing*, Dallas, TX, 1990

[12] James A. Storer, John H. Reif, and Tassos Markas," A massively parallel VLSI design for data compression," in *Proc. IEEE Workshop on VLSI Signal Processing*, 1990

[13] M.T. Sun and S.M. Lei," A parallel variable-length-code decoder for advanced television applications," in *Proc. 3 rd Int. Workshop on HDTV*, 1989

[14] M.T. Sun, K.M. Yang, and K.H. Tzou," A high speed programmable VLSI for decoding variable length codes," in *Proc. SPIE, Applications of Digital Image Processing XII*, vol.1153, Aug. 1989