

AN EFFICIENT VLC DECOMPRESSION SCHEME FOR USER-DEFINED CODING TABLES*

Bai-Jue Shie and Chen-Yi Lee

Department of Electronics Engineering, National Chiao Tung University,
1001, Ta Hsueh Road, Hsinchu, 300, Taiwan, R.O.C.

ABSTRACT

With the increase of information and data types, high-throughput and flexible memory-based VLC decoder is required for user-defined coding tables to achieve higher compression ratio. In this paper, we present a memory-based VLC decoder which is quite suitable for the applications with user-defined tables. By parallel loading data into memories, the coding tables can be changed with much less time. The codeword-boundary prediction algorithm breaks the recursive dependency of decoding procedures. As a result, the VLC decoder can be realized on multi-processor architecture and hence the decoding throughput is enhanced significantly. Additionally, the INDEX-OFFSET symbols that can recover all data with pure VLC codeword and smaller table size are presented. Simulation results show that the combination of the proposed VLC decoder and user-defined table can achieve high decompression rate. As a result, it is quite suitable for high data rate applications with user-defined coding tables, such as MPEG-4.

1. INTRODUCTION

Variable-Length-Code (VLC), also called the Huffman code [1], is the most popular data compression technique that has been used in many standards, such as JPEG, MPEG, and H.263. Based on the predetermined weight of each symbol, the idea is to assign shorter codewords to the symbols having higher frequency and the less probability symbols are assigned with longer codewords. It exploits the information redundancy and the result of encoded codeword bit stream is very close to source entropy. However, different data types lead to different symbol probability distributions. As a result, it needs individual VLC coding table for each data type to achieve higher compression ratio.

With the advances of 3D graphics, multimedia and communication technologies, both the amount of information and the variability of data types increase significantly. As a result, user-defined coding tables, which explore more data redundancy, are necessary to reduce memory requirement and release communication bandwidth. Being different with standard-defined tables, the decoding information of user-defined table is transmitted from communication channel or storage device before the corresponding bit stream. To increase the transmission efficiency of codewords, it has to reduce the table size and decoding information for such applications. For physical design, the VLC decoder is a small part of whole system. Because memory modules occupy large area, it is not efficient that all

used coding tables are stored in VLC decoder. Consequently, the decoding information is loaded when they are requested. If the tables are changed for each part of bit stream, the time for loading tables will reduce decoding throughput and system performance. This may not be accepted for high data rate applications. On the other hand, the procedure of data recovery includes codeword decoding to symbol and symbol converting to practical value. To achieve high decompression rate, not only codeword bit stream has to be suitable for high throughput VLC decoder, but also symbol representation needs to be modified to simplify data recovery procedure.

A set of memory-based architectures has been discussed in [2], [3], [4], [5], [6]. They store the table information in on-chip memories. For this reason, user can change the coding table to meet the requirements of various applications and obtain the flexibility. To optimize system clock rate, some of these designs use pipeline technique to divide the decoding procedure. However, they did not break the recursive dependence of codeword bit stream and the decoding throughput does not increase very efficiently. To satisfy diverse applications requirements, the motivation behind our research is to develop a flexible and high-throughput VLC decoder with less loading time for decoding information. Moreover, symbol representation that can represent all data value with smaller table size and simply data recovery procedure is our focus, too. The organization of this paper is as follows. In section 2, the codeword-boundary prediction algorithm is described. The architecture of VLC decoder with dual-processor is presented in this section, too. In section 3, the INDEX-OFFSET symbols and corresponding data recovery method that can restore all data value with simple method and smaller table size are shown. Finally, concluding remarks are made in section 4.

2. VLC DECODER WITH DUAL-PROCESSOR ARCHITECTURE

2.1 Codeword-Boundary Prediction

In decoding procedures, tree structure is formed by various types of branches among parent-node and its child-nodes. The branch type, stored in the parent-node, is the primary information for searching the child-node. VLC decoder depends on the codeword bit stream and the branch type information to go through one tree level in each decoding procedure. As a result, it traverses a unique path from the root to the leaf node that matches the symbol of the codeword. If VLC decoder can learn about the

* WORK SUPPORTED BY THE NATIONAL SCIENCE COUNCIL OF TAIWAN, ROC, UNDER GRANT NSC87-2215-E-009-035

child-nodes condition, such as “Child-nodes are All Terminal nodes” (CAT) or “there are Special terminal nodes” (S), during the period of decoding their parent node, it is able to know how many bits remain to be decoded after this parent node. Consequently, the codeword-boundary and code-length can be predicted.

Traditional branch types consist of one kind of regular node and three kinds of “Special terminal node” [3]. Based on the requirement mentioned above, the “CAT” and “S” information is added into these branch types. Then, the 2-bit tree branch models, which can perform the codeword-boundary prediction, are established as shown in Fig 1. It also presents the bit assignment for each branch model. In addition, several symbols having very small probabilities will be combined into a group to reduce the complexity of encoding procedures. Therefore, we define 2 kinds of “Group branch models” to predict group-codewords in a more efficient way.

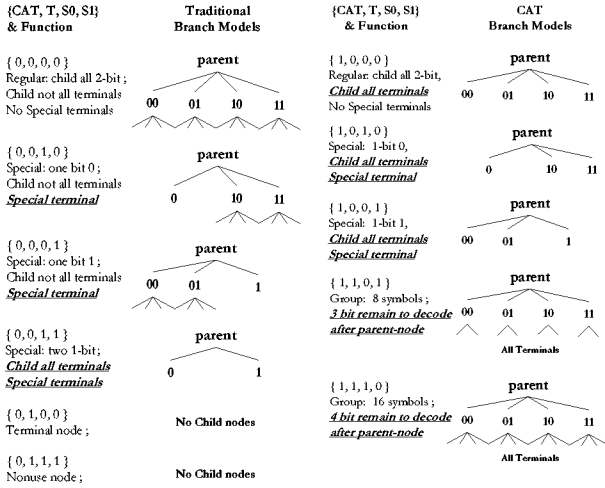


Figure 1: The branch model and bit assignment.

2.2 Memory-Mapping Scheme

Based on a memory-mapping scheme proposed in [9], the child-nodes of the same parent node are merged into a “Loc” node and each “Loc” node stores 4-set bit assignments of branch models to represent the child-nodes conditions. Beside, two additional memories, “T” and “C”, are introduced to calculate the “symbol address” and “next Loc address” respectively. The i -th location of the “T” memory stores the total number of terminal nodes appearing from 0 to $(i-1)$ -th “Loc” node. On the contrary, the i -th value of “C” memory indicates the total number of nodes, which has child-nodes extension from 0 to $(i-1)$ -th “Loc”. Because the “Loc” locations greater than “Cmax” only consist of terminal and nonused nodes, it uses 4-bit data “R” to indicate valid terminal nodes instead of using 4-set bit assignments “Loc”. Besides, the “C” memory can be eliminated since it needs not calculate next Loc address when the terminal node is found.

Since the data structure is different, two memory modules are used for storing decoding information of {LOC, T, C} and {R, T_R } respectively. Moreover, symbols are stored in the third

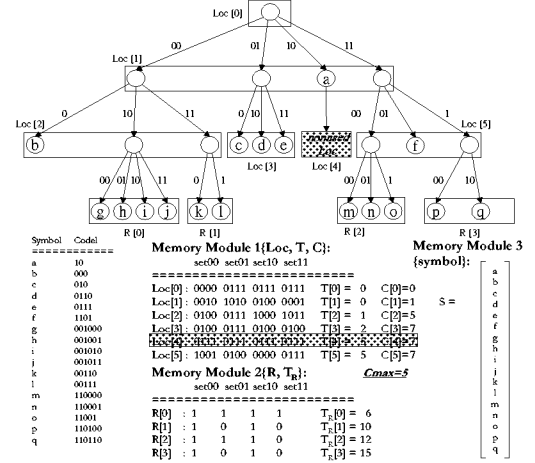


Figure 2: An example of memory-mapping data format.

memory module. By parallel loading, the time for storing table into memories is reduced efficiently. An example of memory-mapping data format is shown in Fig. 2.

2.3 Codeword-Boundary Prediction

The decoding procedures with codeword-boundary prediction can be explained by the following example. Assuming the codeword bit_stream is {11001} of the Huffman tree shown in Fig 2. Because every codeword starts from tree level 1 that only consists of Loc[1], it is stored in individual register and can be accessed directly. At the same time, the 4th Loc, Loc[5], in level 2 is accessed depending on the bit_stream[0:1] = 11, too. Then they are stored in register “dMDR” and “pMDR” respectively. Depending on the bit_stream[0:1] = 11 and bit_stream[2:3] = 00, set11 in Loc[1] and set00 in Loc[5] is selected. According to branch model set11 in Loc[1], bit_stream[0:1] is not terminal and its child nodes are not all terminal nodes, neither. On the other hand, the child-nodes of bit_stream[2:3] are all terminals and there is a special terminal, whose label is 1, based on the branch model set00 in Loc[5]. After comparing the 5th bit with the branch model, it is known that the codeword is the special terminal and 1 bit remains to be decoded after bit_stream[2:3]. Now, the code-length = 5bits is predicted.

Because there is not enough information to find the matching symbol address even though the code-length is known. The decoding procedure described above has to be applied again. The next Loc address having to access is minus 1 of the sum of C[5] and “OFSC”, which is the number of non-terminal nodes before label 00 in Loc[5] stored in dMDR register. Because $(C[5]=7) + (OFSC=1) = 8 > (Cmax=5)$, $R[(8-5)-1]$ is accessed and stored in pMDR in the second cycle. Besides, the Loc data stored in pMDR is shifted into dMDR. Now, we use the branch model set00 of Loc[5] stored in dMDR to decode bit_stream[2:3] = 00. It is not the terminal node, neither. The code-length needs not predict anymore since it has been known in the previous cycle. Finally, R[2] is used to detect the terminal node and symbol address in the third cycle. The symbol address = 14 is minus 1 of the sum of $T_R[2]=12$ and “OFST”=3, which is the number of terminal nodes before label 1 in R[2] stored in dMDR.

2.4 Multi-Processor Architecture

Next codeword bit stream is available when the code-length is known. However, VLC decoding processor has to complete the decoding procedures for each codeword, otherwise it can not find the matching symbol address. To increase the decoding throughput, another decoding processor is used to decode the valid next codeword bit stream. The VLC decoder with dual-processor architecture is shown in Fig 3. Both processors can start the decoding procedures when the correct “codeword bit stream” is ready and the “Start” signal is valid. On the other hand, the controller can learn about the codeword-boundary by the “code-length”, which is transmitted from decoding processor when the “Predict” signal is valid. Because the tree structure data are the same for each processor, the requirement of memory space remains the same by using multi-read-port memory modules. Therefore, the area overhead of multi-processor architecture is acceptable since only decoding processor needs to be duplicated.

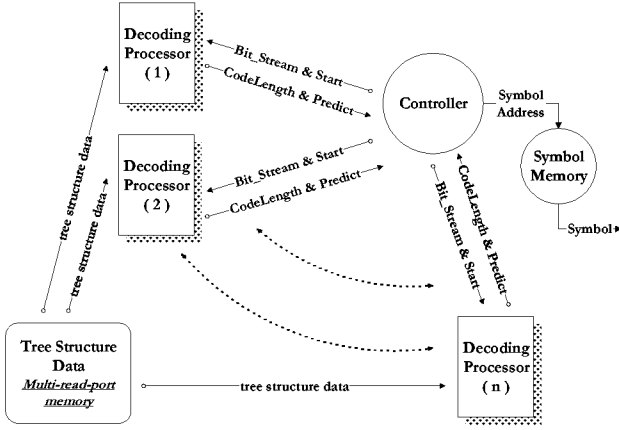


Figure 3: Dual-processor architecture for the proposed VLC decoder.

The detailed VLC decoder with dual-processor architecture is shown in Fig. 4. The main components are listed as follows: 1) the “FSM Controller” controls the operations of whole VLC decoder. 2) “Stream Assignment Unit” (SAU) selects codeword bit stream to decode. 3) The dual “Decoding Processors” execute codeword-boundary prediction and decoding procedures to find code-length and matching symbol address. 4) “2-Read Port Memory {Loc, T, C} & {R, T_R}” and “Initial Loc, T, C Register” store tree structure data described in the previous section. 5) The “Address Reorder Buffer” is used to reorder the symbol address as input codeword. 6) “Symbol memory” stores the symbols, run/length or data value.

3. INDEX/OFFSET Symbols

To meet real-time processing, high throughput decompression scheme and simple data recovery procedure are requested. RUN/LEVEL symbol conjunct with one sign bit is used to encode DCT coefficients as needed in MPEG standard. Because only high frequency symbols are defined in coding table for controlling size, the fixed length RUN/LEVEL following by an ESCAPE code is used to represent all possible RUN value and

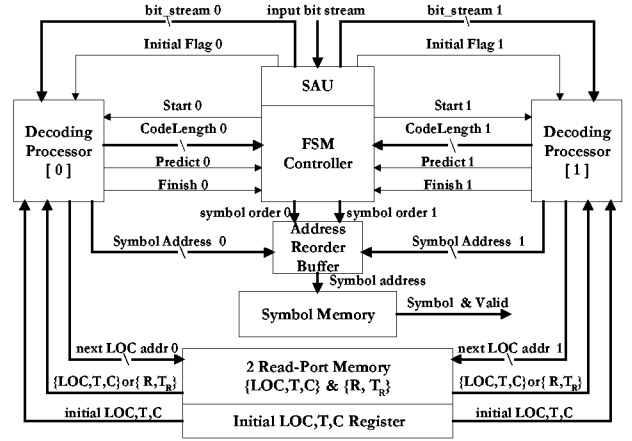


Figure 4: Detail VLC decoder with dual-processor

practical DCT coefficients. However, the compression ratio is reduced in the case of ESCAPE code. To achieve higher compression ratio, RUN/LENGTH symbol conjunct with magnitude is used for static images, such as JPEG. The bit length of magnitude is variable and has to be determined by LENGTH after VLC decoding procedures are completed. Besides, the data is recovered when the variable magnitude extends to fix-length value. As a result, the procedure of data recovery is much more complex and the decompression rate may not meet real-time performance. Additionally, the recursive dependency of decoding procedure can not be broken since the bit-stream, which is generated by RUN/LEVEL or RUN/LENGTH symbol, is not pure VLC codewords. The decoding throughput may not meet the demand of higher-data rate, user-defined table applications where the time for loading table is taken into account.

The combination of RUN and LEVEL is the reason why it needs large table to define all possible symbols. Therefore, we split RUN/LEVEL into 2 kinds of symbols, RUN and non-zero value. Sixteen RUN symbols, one zero to sixteen zero, is defined. As a result, any length of RUN can be represented by a sequential of RUN symbols. The distribution of non-zero value is shown in Fig. 5, which is generated by analyzing twenty different images. Because most of the probabilities are in the range between $-32 \sim +32$, sixty-four OFFSET symbols that exactly represent all non-zero values in the same range are defined. For the value out of this rang, the more complex data recovery method, “INDEX + OFFSET”, is acceptable since the frequency is much lower. The distance between INDEXs is the size of OFFSET range. As the example above, INDEX symbols represent the multiple of 64, such as $\pm 64, \pm 128, \dots$, are defined. One data recovery example of “INDEX + OFFSET” is shown as follow:

$$50 = 64 + (-14) ; \text{ or } -50 = (-64) + 14 ;$$

Additionally, the order of codewords in bit stream is shown below:

INDEX – OFFSET

As a result, it adds the following symbol value when the INDEX symbol value is received. Although, it is the facts that 2 symbols are used to represent one data value that out of the OFFSET range and the compression ratio may be reduced. The penalty is as low as the frequency of this case. However, the decoding

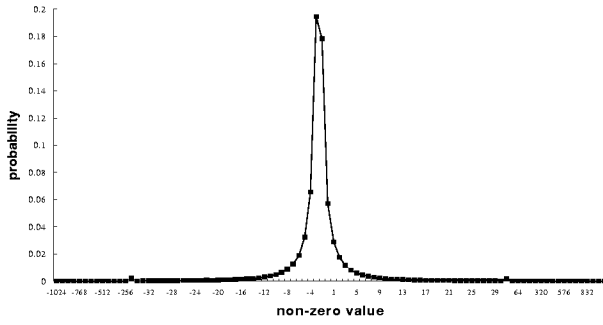


Figure 5: Probability distribution by analyzing 20 different images.

throughput of pure VLC codeword bit stream, which is formed by INDEX-OFFSET symbols, can be increased efficiently by using codeword-boundary prediction algorithm and VLC decoder with multi-processor architecture.

As the method described above, the user-defined table that can recover all values between $-1024 \sim +1024$ is shown in Fig. 6. It includes 16 RUN, 64 OFFSET, 32 INDEX. Comparison of total bit lengths with different symbol representations is shown in Table 1. Simulation results of the decompression rate with the combination of the proposed VLC decoder and INDEX-OFFSET symbols are listed in Table 2.

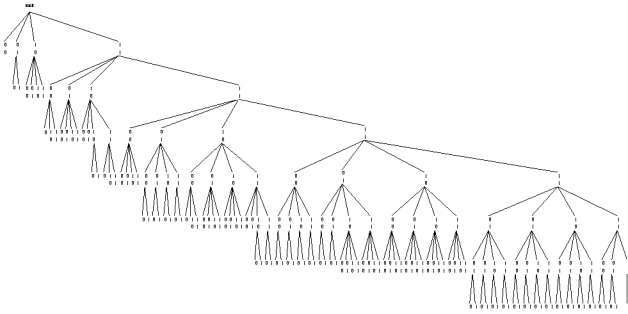


Figure 6: The 2-bit tree structure of user-defined coding table.

4. CONCLUSION

In this paper, we have shown a high-throughput, flexible memory-based VLC decoder design. To reduce the time for loading decoding information, both data structure partitioning and parallel loading are used. The recursive dependency of decoding procedure can be broken by codeword-boundary prediction. With the proposed dual-processor architecture, the decompression rate increases efficiently. Moreover, the INDEX-OFFSET symbol representation is presented to recover data value by pure VLC codeword bit stream and to reduce table size. Simulation result shows that decoding throughput of the combination of the proposed VLC decompression scheme and INDEX-OFFSET symbol representation can, on the average, achieve 780Mbps for 12bit symbols with 100MHz-clock rate. As a result, the proposed VLC decompression scheme with user-defined table is able to meet the requirement of high data rate, real-time applications, such as MPEG-4.

5. REFERENCES

- [1] D.A. Huffman, "A method for the construction of minimum-redundancy codes," in Proc. IRE, vol. 40, pp. 1098 - 1101, Sept. 1952.
- [2] A. Mukherjee, N. Ranganathan, and M. Bassiouni, "Efficient VLSI design for data transformations of tree-based codes," IEEE Trans. on Circuits and Systems, vol.38, No. 3, pp.306-314, Mar.1991.
- [3] 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, Vol.1, No.2, pp.203-213, June.1993.
- [4] Heonchul Park and Viktor K. Prasanna, "Area Efficient VLSI Architectures for Huffman Coding," IEEE Trans. on Circuits and System, Vol.40, No.9, pp.568-575, Sept.1993.
- [5] Liang-Ying Liu, Jhing-Fa Wang and Jau-yien Lee, "Cam-Based VLSI Architecture for Daynamic Huffman Coding", IEEE Trans. on Consumer Electronics, vol.40, No.3, pp.282-289, SeptAugust1994.
- [6] M. K. Rudberg and L. Wanhammar, "Implementation of a fast MPEG-2 Compliant Huffman Decoder, " in Proceedings of EUSIPCO-96, vol.3, pp.1467-1470, Sept.1996.
- [7] Juinn-Ying Wu and Liang-Gee Chen, "A variable length decoder for MPEG-2, " in 1996 HD-Media Technology and Applications Workshop, No.A5, pp.3/13-3/18.
- [8] Yew-San Lee and Chen-Yi Lee, "A Memory-Based Architecture for Very-High-Throughput Variable Length Codec System" in Proc. Of ISCAS'97, vol. 3, pp. 2096 - 2099, June 1997.
- [9] Bai-Jue Shieh, Yew-San Lee and Chen-Yi Lee, "A High Throughput Variable Length Decoder with Modified Memory Based Architecture" in Proc. Of ISCAS'98, WAA14-24, June 1998.

Table 1: Comparison of total bit lengths with different symbol representations.




Images Symbols	 (320×240)	 (512×512)	 (512×512)
RUN/LEVEL (MPEG-2 TB 15)	103503	317764	251343
INDEX-OFFSET (User-Defined)	101575	315341	244079

Table 2: Simulation results of the decompression rate of combining the proposed VLC decoder and INDEX-OFFSET symbols at 100MHz-clock rate.

Images Symbols	Symbol Count	Decoding Cycle	Throughput
Lena	22551	34926	775 Mbps
Scene	70145	106141	793 Mbps
Pepper	53440	83969	764 Mbps
Average			780 Mbps