國立交通大學

電子工程學系　電子研究所碩士班

碩　士　論　文

具抗錯力之以記憶體為基礎的雙模視訊標準可變長度解碼器設計

Design of An Error-robust Memory-based VLC Decoder for Dual-mode Video Decoding

研究生　：　李韋磬

指導教授　：　李鎮宜　教授

中華民國九十七年七月

具抗錯力之以記憶體為基礎的雙模視訊標準可變長度解碼器設計

Design of An Error-robust Memory-based VLC Decoder for Dual-mode Video Decoding

研 究 生：李韋磬　　Student：Wei-Chin Lee

指導教授：李鎮宜　　Advisor：Chen-Yi Lee

國 立 交 通 大 學
電子工程學系 電子研究所 碩士班
碩 士 論 文

A Thesis

Submitted to Department of Electronics Engineering & Institute of Electronics

College of Electrical and Computer Engineering

National Chiao Tung University

In Partial Fulfillment of the Requirements

For the Degree of Master

In

Electronics Engineering

July 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年七月

具抗錯力之以記憶體為基礎的雙模視訊標準可變長度解碼器設計

學生：李韋磬　　　　　　　　　　　指導教授：李鎮宜 教授

國立交通大學

電子工程學系 電子研究所碩士班

摘要

本論文提出在支援雙模(H.264/AVC 和 MPEG-2)視訊壓縮標準解碼器下的熵解碼器中，增進解碼表格在記憶體中的使用效率。此熵解碼器採用多張解碼表格合併演算法並可程式化以包含不同視訊壓縮標準的熵解碼。首先利用單一表格將字碼分類到不同的群組，只把各個群組的最重要的資訊存於記憶體中，解碼的過程只需要用字碼的運算和群組資訊就可以算出符號在記憶體中的位址。在這樣的結構下，可以比傳統的方式節省記憶體的使用量。視訊壓縮標準的表格相當多張，因此把需要使用的表格的群組資料再做合併以更減少表格間的贅餘部份而減少記憶體空間。經由修改演算法可以讓熵解碼器以較少的空間及較高的效率來存放足夠的資訊。

此論文另外針對視訊無線傳輸的應用下，提出了防止錯誤傳遞的方式而且不用額外傳輸資料來輔助，也就是幾乎不會增加頻寬成本。這個方式可以利用通道解碼提供的位元可靠度來決定啟動防止錯誤傳遞的模組，而這個模組裡用記憶體儲存區塊預測資訊。用這些資訊可以預測出區塊邊界而讓可能出錯的區塊的解碼不會影響到接下來的區塊。這些資訊的比對是由以記憶體為基礎的可變長度解碼的方式來決定是否預測成功，在演算上可與傳統的以記憶體為基礎的可變長度解碼器結合，硬體實作上的複雜度不高。這個提出的方法因為限制了錯誤傳遞的範圍，可以大幅改善畫質。

Design of An Error-robust Memory-based VLC Decoder for Dual-mode Video Decoding

Student : Wei-Chin Lee Advisor : Dr. Chen-Yi Lee

Department of Electronics Engineering
Institute of Electronics
National Chiao Tung University

ABSTRACT

This thesis proposed entropy decoder which can support dual-mode video format (H.264/AVC and MPEG-2) decoding and improved memory efficiency. The entropy decoder adopts multi-table merging (MTM) algorithm and is programmable. First, for a coding table, all codewords are separated into different groups and only the most significant information of each group is stored in memory. The decoding can be completed by looking up information needed and arithmetic computation such that the symbol address is known. Under this type of decoding, memory space can be reduced compared to conventional VLC decoder. For multiple tables, the redundancies between group information of each table are further exploited and only different parts are stored in the memory. By the modified MTM algorithm, the proposed can store information needed for the standards with higher memory efficiency and less memory space.

This thesis also proposed a scheme which can stop error propagation without transmission of extra data helping stop error propagation, i.e., no additional bandwidth overhead. In this method, the stopping error propagation module is activated when the bit reliability coming from FEC is low enough. The error resynchronization information is stored in the memory and block boundary prediction is achieved with the information. Once the block boundary, the following block can be correctly decoded and error propagation can be restricted. The searching of information used group-based VLC decoding algorithm to determine if prediction is finished. In the algorithm level, the method is combined with conventional group-based VLC decoding; in the viewpoint of hardware implementation, the complexity is low. Owing to stopping error propagation from this scheme, video quality is improved drastically.

Acknowledgements

I am very grateful to my advisor Dr. Chen-Yi Lee because of his patient guidance and very valuable comment for my research. I have learned much experience for research and thinking from him.

Second, I would like to appreciate NSC for the financial support, the teammates of Si2 multimedia group. My seniors like Tsu-Ming Liu and Yao Li gave a lot of suggestion for my thinking and skill training. I am very thankful to them and Yu-De Wu, Yu-Fan Lai. Besides, graduated seniors such as Jun-Yan Yang, Yi-Hung Huang, Kon-Zeng Hou, Zeng-Chie Hsien, Bin-Chang Lin, Yua-Chih Hong and Da-Jiah Lin also shared much experience with me for my graduate life. Finally, I want to express the gratitude for senior Ching-Che Chung's work on the setting and testing of libraries and technology process and building of simulation environment. Besides, Dr. Ti-Hao Chiang and Dr. Hsie-Chia Chang also gave me some precious comment.

Finally, I would like to appreciate my family and my friends for their assistance and encouragement when I feel stress. I wish all of them are healthy and blest perpetually.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview of H.264/AVC

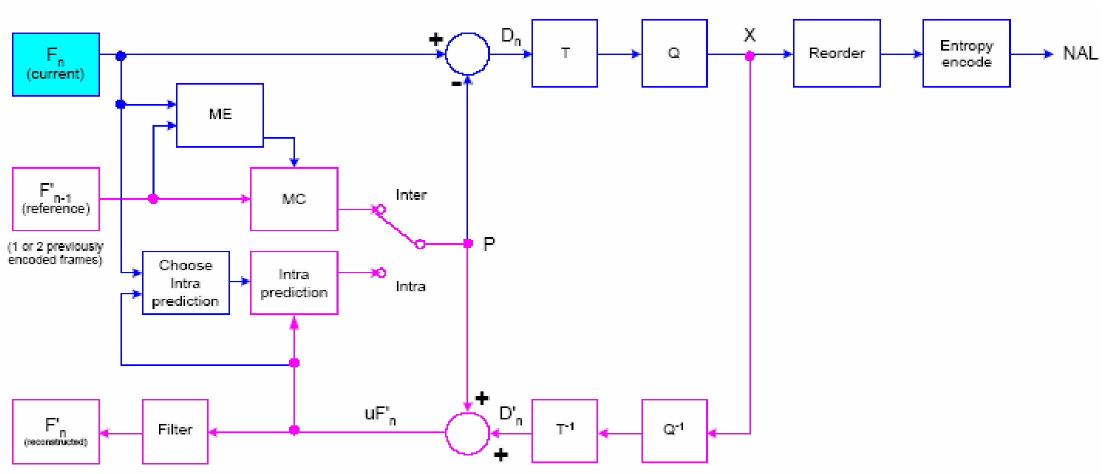


Fig. 1 The block diagram of H.264 encoder

The newest video coding standard is H.264/AVC which was developed by Joint Video Team (JVT). H.264/AVC outperforms the previous video coding standards in the coding efficiency. The block diagram of the H.264 encoder is shown in Fig. 1. The current frame is predicted either by intra prediction or inter prediction. If the frame is an intra frame, all data in the predicted frame come from the current frame. For inter frame, the current frame and reference frame are use to compute motion vectors and compensated by the reference blocks. After prediction, data of current frame subtract that of predicted frame so that only residual data remained. The residual data then passes transformation, quantization, reorder and entropy coding and becomes bitstream. In the backward path, the predicted frame and the residual data are added to form the unfiltered frame for intra prediction. Finally, the reconstructed frame is formed by filtering the uF’n.

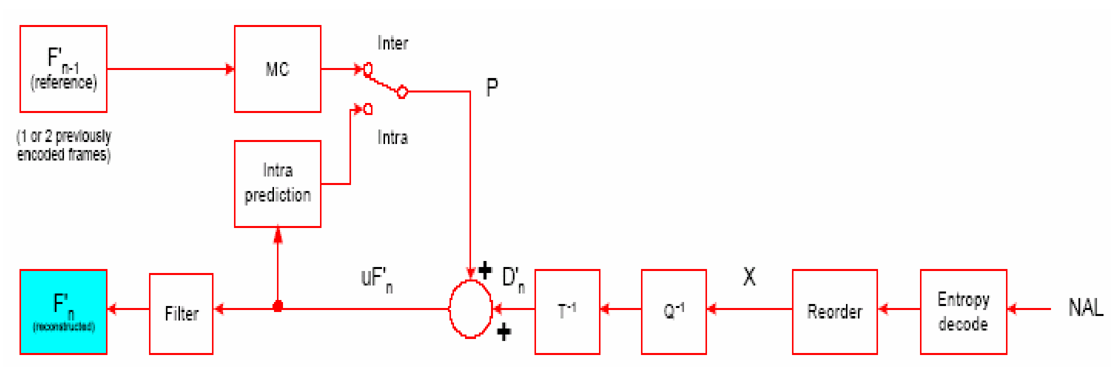


Fig. 2 The block diagram of H.264 decoder

Fig. 2 shows the block diagram of H.264 decoder. The first step of decoding is entropy decoding of bitstream. The output data are sent into inverse quantization and inverse integer discrete cosine transform. Now, the data is the residual coefficients between the current frame and the prediction frame. Next step is to add residual data to prediction of current frame (intra or inter). Lastly, the frame passes loop filter to reduce the blocking effect.

Table 1 H.264/AVC profiles and the corresponding tools

Profile	Baseline	Main	Extended
Tools			
CAVLC	√	√	√
CABAC		√	
FMO	√		√
Slice group and Adaptive Slice Ordering	√		√
I & P Slices	√	√	√
1/4 pixel MC	√	√	√
Loop Filter	√	√	√
Intra Prediction	√	√	√
Multiple Reference Frame	√	√	√
B Slices		√	√
Field Coding		√	√
MB-Aff		√	√
Weighted Prediction		√	√
Data Partitioning			√
SP/SI Slices			√

For different applications, there are different profiles and tools in H.264/AVC which shows as Table 1. Baseline profile is mainly for mobile applications of low bit rate such as portable devices because of its lower computation complexity than other profiles. Extended profile is based on baseline profile and has error resilient tools for video streaming or video on demand (VOD) applications.

The higher level profile based on baseline profile is main profile which is for broadcast application. The computation complexity of main profile is more than that of baseline profile. In addition, H.264/AVC has the high profile, high 10 profile, high 4:2:2 and high 4:4:4 profiles based on main profile for high definition multimedia applications. The high profile supports 8x8 integer transform and high 10 profile contains high profile with extra support of 10-bit sample precision of the decoded pixels. Further, high 4:2:2 profile based on high 10 profile supports 4:2:2 chroma sampling precision and 10-bit per sample. Finally, high 4:4:4 profile supports 4:4:4 chroma sampling and 12-bit per sample.

From table 1, we can see that there are two entropy coding approaches for entropy coding, one is context adaptive variable length coding and the other is context adaptive binary arithmetic coding. Although CABAC has better compression rate than CAVLC, CABAC has extremely more complex structure which limits the throughput of CABAC than CAVLC. Besides, CAVLC is suitable for all profiles in H.264/AVC system and it has more flexibility for different applications. Therefore, we still further discuss CAVLC in the following sections after the overview of MPEG-2.

1.2 Overview of MPEG-2

MPEG-2 is a video standard established by Moving Pictures Experts Group (MPEG) which is a team of International Standards Organization (ISO). There are five profiles in the MPEG-2 system. The simple profile supports 4:2:0 sampling, intra and inter prediction. Main profile contains all tools of simple profile plus bi-direction prediction. In addition, SNR scalable profile and spatially scalable profile provide the base layer and one or more upper layer of coded bitstream for wider and different application conditions. Finally, the high profile contains all previous tools and it is for the applications where there are no constraints on bit rate.

There are four levels specified in MPEG-2: High level, High 1440, Main level, and Low level. Higher level supports higher resolution of video. Main Profile and Main level is the most widely accepted combination for the majority of applications.

1.3 VLC and CAVLC

1.3.1 Huffman Code

The Huffman code can encode one source with variable length code (VLC) based on the probability distribution of the source symbols. For example, a source contains four symbols — { a, b, c, d } and the probability of them are 0.5, 0.25, 0.125 and 0.125, respectively. Therefore, we can trace the tree structure to assign codewords for the symbols as shown in Fig. 3. If we use 2-bits codewords to encode the source, the average length is 2 x 0.5 + 2 x 0.25 + 2 x 0.125 + 2 x 0.125 = 2-bits. However, we can use VLC such that average length is 1 x 0.5 + 2 x 0.25 + 3 x 0.125 + 3 x 0.125 = 1.75-bit. For the decoding process, we can just trace the tree from the root to the leaf and then back to the root to decode the next symbol. In MPEG-2, we can see that the coefficients in the block are encoded by run-level coding. The tables defined the mapping between run-level symbols and codewords.

Symbol	Probability	Codeword
a	0.5	1
b	0.25	01
c	0.125	001
d	0.125	000

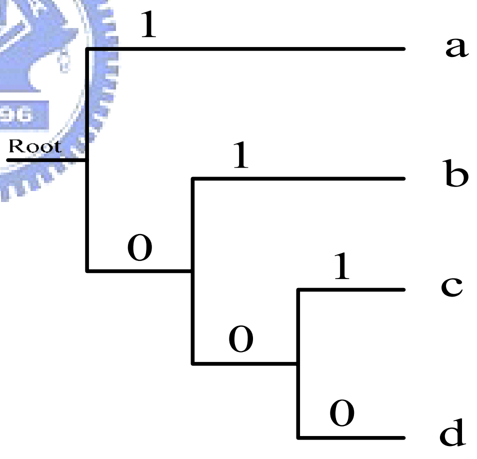


Fig. 3 The distribution of symbols and the Huffman tree.

1.3.2 CAVLC Decoding

The entropy coding of baseline profile in H.264 is extension of VLC because of the context-adaptive property. The so-called context adaptation means to use different probability model under different conditions and assign corresponding VLCs. Therefore, the VLCs separately assigned can achieve better coding efficiency than only use one VLC for all conditions. In CAVLC, there are several tables for one symbol according to the context conditions. The encoding and decoding process is still by look-up tables.

It is important to consider the throughput when design CAVLC decoder. The number of macroblocks must be decoded per second for different resolution shown in Table 2. In addition, application of baseline profile is mainly mobile devices, thus the power consumption issue must also be taken into account.

Table 2 The throughput of decoding CAVLC under different resolution

	30 frames per second (fps)						
	QCIF	CIF	QVGA	VGA	D1	HD 720p	HD 1080
MB/sec	2970	11880	9000	36000	40500	108000	244800

1.4 *Error Robustness*

Nowadays, wireless video transmission is more and more popular in daily life. Over wireless channel, noise interference affects the data correctness and then the source decoder will accept erroneous data. For wireless video transmission, the data are variable in length. Therefore, even only one bit is corrupted, the whole bitstream may lose synchronization which degrades video quality drastically.

In channel coding, there are many coding techniques to protect transmitted data by appending redundancy to achieve error correction. Another method is to detect error and signal the request of re-transmission of video data. These two methods need higher bandwidth. However, in some application, the bandwidth is limited and the less redundancy added on data, the less capability of error correction. Besides, forward error correction code can not promise totally correct all erroneous data and decoder input may still have remained erroneous bits in it.

There are some schemes to improve the error robustness in the decoder side. For example, error concealment replaces the corrupted block by surrounding blocks which is correctly decoded and improves the video quality. Nevertheless, an error detection module must also help to find the location of corrupted blocks.

Soft-input decoding is another method to reduce bit error rate (BER) of the decoder input bitstream. Soft-input decoding uses channel information to find the maximum likelihood or maximum a posteriori path of the trellis diagram. In advance, joint source–channel design (JCSD) can improved BER performance by considering trellis structure and symbols probability of source and channel concurrently.

1.5 Motivation

A video decoder can support multi-standard video format is very significant in today's applications. Most data in the video bitstream is composed of coefficients of blocks. In addition, the first stage of the video decoder is VLC entropy decoder which maps codewords to symbols. As a result, an entropy decoder which is programmable and compatible to different standards with enough throughputs is necessary. Besides, the symbol format of coefficients data is represented by a pair of Run and Level while that of CAVLC is quite different from this.

For the video transmission over wireless environment, we have to find methods to reduce the effect of error propagation. The challenge lies in less information or redundancy remained in the input bitstream of video decoder that can help to detect error and even correct error. Although the whole video transceiver can set as automatically repeat request for the corrupted data, that is, receiver signals a flag to transmitter to re-transmit data. This method results in increasing usage of bandwidth. If soft-input decoding or JSCD is used, high complexity and cost are inappropriate to hardware implementation.

1.6 Organization of the Thesis

The chapter 2 will discuss the previous works about VLC decoder of different implementation and target applications first. Then the previous works of error robust on decoder side only or JSCD will be mentioned. Chapter 3 will show the CAVLC operation and the design of memory-based VLC decoder supporting multiple standards. In the memory-based VLC decoder, multi-table merging algorithm is used and the allocation of memory is considered. Next, we will show the hardware architecture and implementation result in chapter 4. Chapter 5 proposes an algorithm used to find the block boundaries in frames stops error propagation under the condition that channel information is known. Also, the simulation result will be in the chapter 6. Chapter 7 will show the conclusion about the whole design of multi-mode and memory-based VLC decoder with error robustness.

Chapter 2

Previous Work

2.1 CAVLC Decoding Process

There are five syntax elements in CAVLC：*Coefficient_Token* (*Coeff_Token*), *TrailingOnes_Sign* (*T1s_Sign*), *Level_Prefix*, *Level_Suffix*, *Total_Zeros* and *Run_Before*. They are decoded in order defined by the following rules and the block data composed of these syntax elements is shown in Fig. 4

1. The first decoded syntax element is *Coeff_Token*, which includes to symbol: *Total_Coeff* and *TrailingOnes*. *Total_Coeff* represents number of non-zero coefficients in this block and *TrailingOnes* represents number of coefficient with magnitude one and it is 3 at most. The sub-tables are select by n*C* parameter from system. n*C* is positive for luma and -1 for chroma.
2. *TrailingOnes_Sign* is decoded by getting *TrailingOnes* bits from bitstream.
3. *Level_Prefix* is decoded by leading one detector and is equal to number of zeros before the leading one.
4. Then, a parameter called *SuffixLength* is initially set to 0 or 1 if *Total_Coeff* is greater than 10 and *TrailingOnes* is less than 3. *LevelSuffixSize* is set to *SuffixLength* with two except case: 1. *Level_Prefix* is equal to 14 and *SuffixLength* is equal to 0. 2. *Level_Prefix* is equal to 15. *LevelSuffixSize* is set to 4 in case1 and 12 in case2. Next, *Level_Suffix* is decoded by getting *LevelSuffixSize* bits from bitstream and is set as 0 if *LevelSuffixSize* is 0.
5. Select *Total_Zeros* sub-tables according to *Total_Coeff*. If *Total_Zeros* is 0, the decoding process is finished.
6. The *Zeros_Left* is set as *Total_Zeros*. *Run_Before* is subtracted from *Zeros_Left* and the result is assigned to *Zeros_Left* until *Zeros_Left* is 0.

Coeffi_Token	TrailingOnes	LevelPrefix	LevelSuffix	TotalZeros	RunBefore

Fig. 4 Sequential syntax elements decoding in CAVLC

2.2VLC Decoder

There are several ways to implement VLC decoder such as memory-based technique, hardwired implementation. [1] proposed group-based algorithm to classify VLC codewords into different groups such that memory just stored group information. In [1], the symbol addresses are calculated by input bit-stream and group information. Last, the symbol memory stored all symbols are accessed to output decoded symbols. The codec can support a coding table with 256-entry 12-bit symbols and 16-bit codewords. Furthermore, [2] proposed the multi-table-merging algorithm to reduce memory space and codec can support JPEG, MPEG-2 and MPEG-4 coding tables. [3] used cache and table partitioning on the group-based VLC decoder to achieve power reduction for MPEG-2. The decoding method in [4] decodes some short codewords by arithmetic operation and the others are mapped into memory to reduce memory access. But [4] was just for *Coeff_Token* tables and its sequential searching in the memory would lead to low throughput. The scheme proposed by [5] and [6] is that decode short codewords with arithmetic operation while other codewords are decoded by conventional decoding to saving memory access.

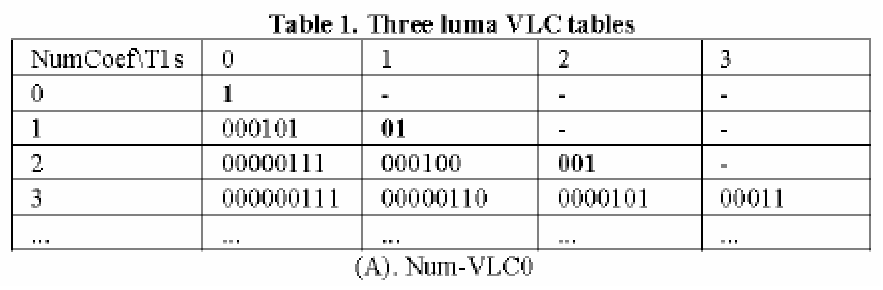


Table 1. Three luma VLC tables

NumCoef\T1s	0	1	2	3
0	**1**	-	-	-
1	000101	**01**	-	-
2	00000111	000100	**001**	-
3	000000111	00000110	0000101	00011
...	...	...	...	...

(A). Num-VLC0

$$NumCoeff = T1s = N0$$

Fig. 5 Examples of short codewords, they can be decoded by arithmetic decoding from the equation in [6].

For the hardware implementation proposed by [7], it was ROM-based and used HLLT (hierarchical logic for look-up table, Fig. 6) to improve speed and PCCF (partial combinational component freezing) to reduce power consumption.

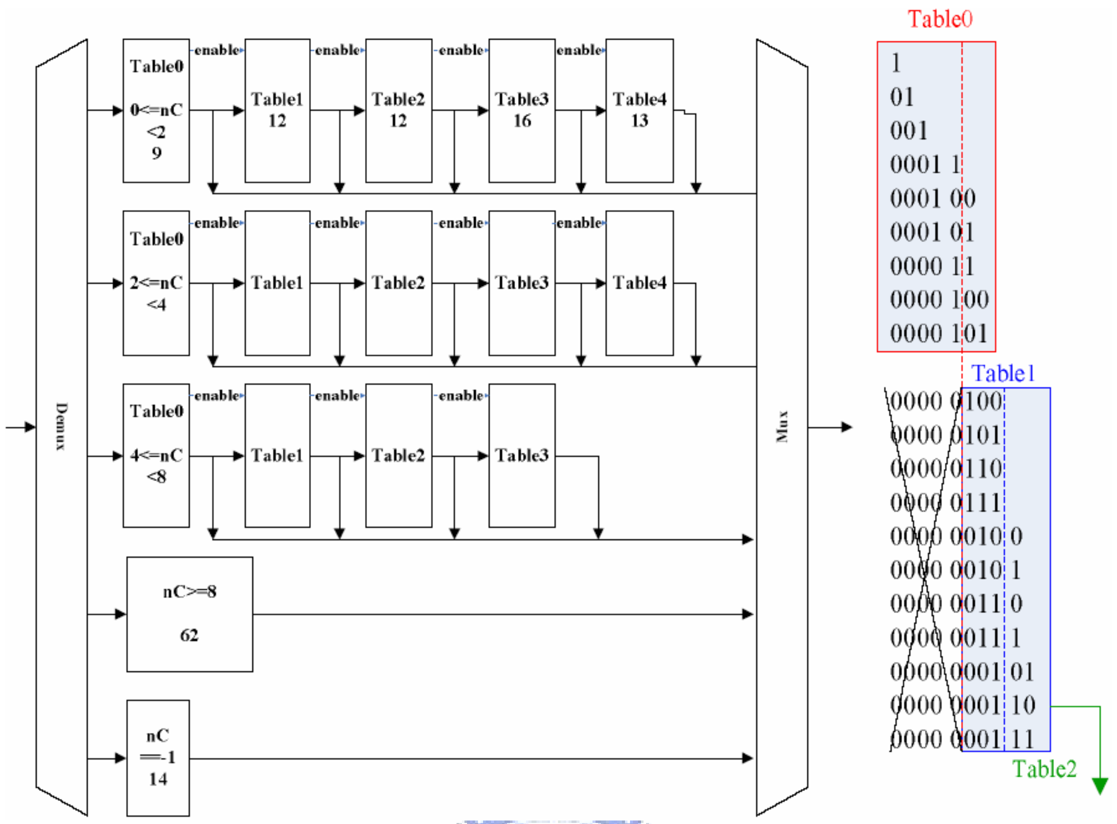


Fig. 6 The implementation of HLLT partitions the original big LUT into many small LUTs in [7].

Design of VLC decoder in [8] was for MPEG-1/2/4 decoding and LUTs are implemented by hardwire. The codewords are separated into groups in several look-up tables and one address generator is used to calculate symbol address.

In [9], the multi-symbol for level decoding in CAVLC is proposed to reduce operation frequency while maintain enough throughput for real-time requirement. [10] proposed a modified *SuffixLength* detector to reduce critical path in level decoding .

2.3 *Error Robustness on Wireless Video Transmission*

Until now, there has been much research on improvement of error robustness to reduce the effect of error propagation in video decoder, compensation of erroneous data and correction error. They can be mainly separated into two sections: source decoder side only and joint source-channel design.

The error robustness mechanism at the source side only includes error detection, error concealment and error resynchronization. Error detection is to find the location of error data or bits in the blocks. The simplest error detection is syntax-based error detection, that is, use some rules that violated regular decoding process. For example, a codeword is not found or the value of a variable overflows. [11] made some rules of syntax-based error detection and analyzed the performance of detection. However, the detection has delay between the correctly detected block and exactly erroneous block as shown in Fig. 7

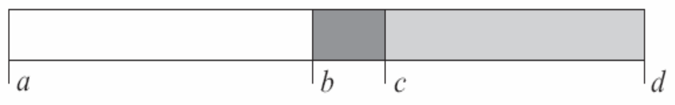


Figure 7. Origination of macroblock level concealment delay

- ***Interval* [*a*,*b*):** The slice is correctly decoded from its begin a up to the error at the position b.
- ***Interval* [*b*,*c*):** The error is undetected until the position $c \geq b$. This part is decoded incorrectly.
- ***Interval* [*c*,*d*]:** Starting from the position c until the end of the slice d concealment is used.

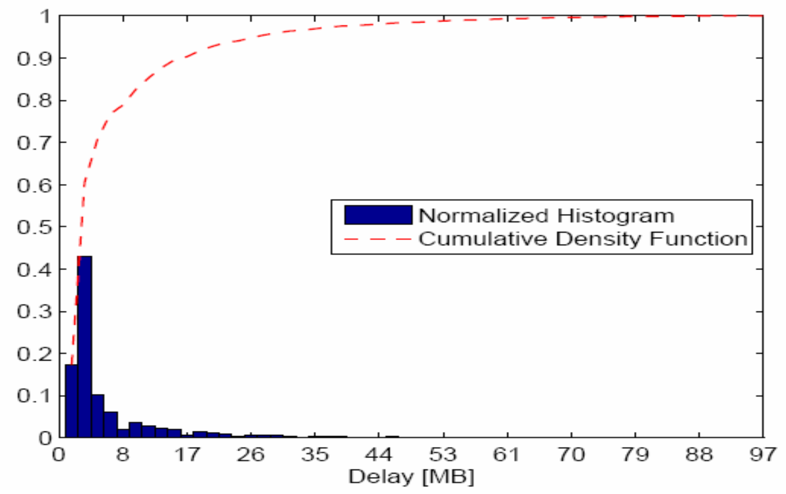
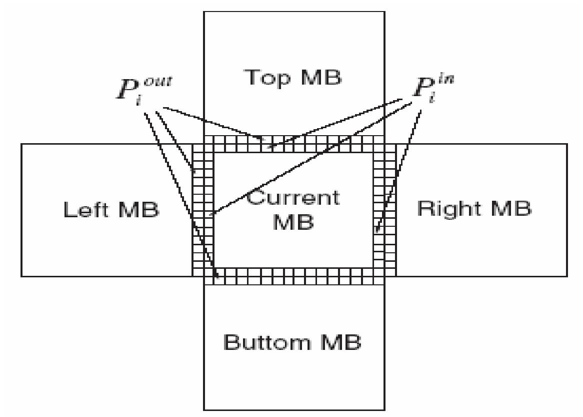


Fig. 7 Organization of macroblock level concealment delay and detection delay in [11]

[12] detected error blocks basically by computing the boundary difference and used threshold as the determination rule. In Fig. 8, L means "Left" and can be replaced by T(TOP), R(Right) or B (Bottom) and K2 is the number of available neighboring blocks of current MB. The temporal boundary checking is shown in Fig. 9. The threshold in this paper is adaptively decided according to the statistics of decoded MBs in a frame



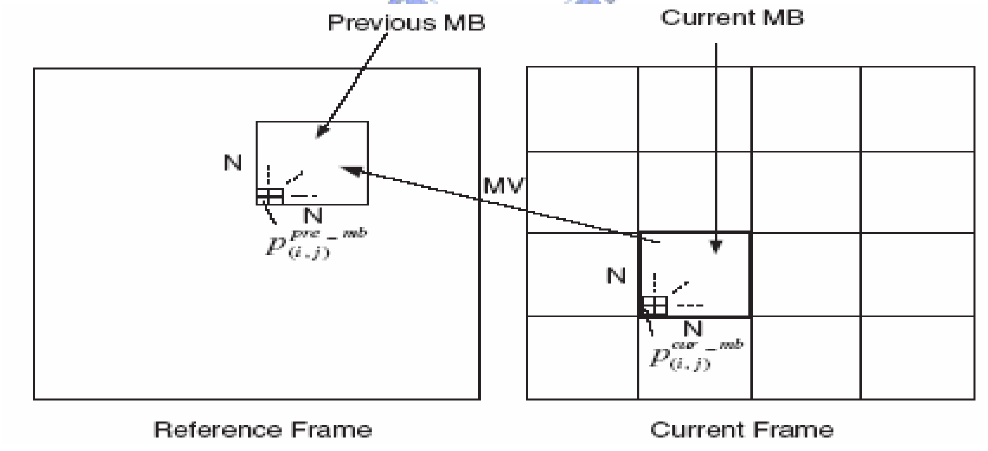
$$\mathrm{AIDB(C:L)} = \begin{cases} 0 & \text{if } \textit{Left MB} \text{ does not exist} \\ \frac{1}{N}\sum_{i=0}^{N-1} \left| P_i^{in} - P_i^{out} \right| & \text{else} \end{cases}$$

(a)

$$\mathrm{AIDB} = \frac{1}{K_2}(\mathrm{AIDB(C:L)} + \mathrm{AIDB(C:R)} + \mathrm{AIDB(C:T)} + \mathrm{AIDB(C:B)}),$$

(b)

Fig. 8 (a) Pixels for average inter-sample difference across boundary (AIDB) calculation and equation in [12] with N=16. (b) Final equation of the AIDB

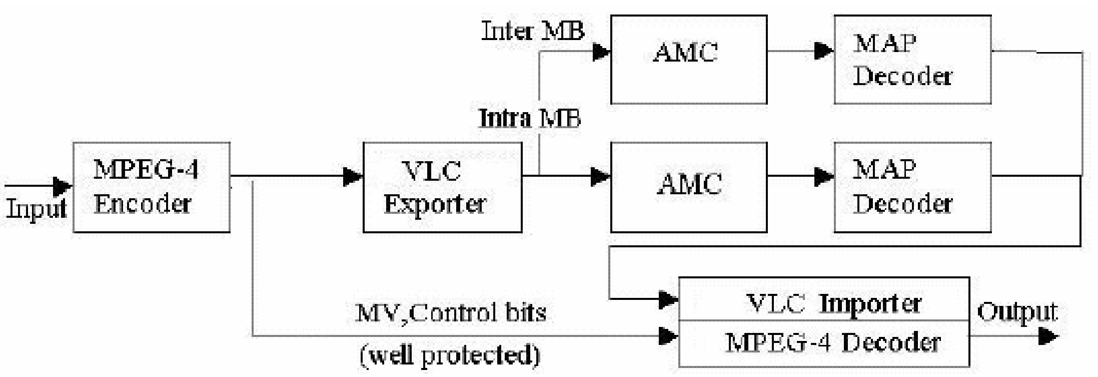


$$\mathrm{ADF} = \frac{1}{N \times N} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left| P_{(i,j)}^{cur\_mb} - P_{(i,j)}^{pre\_mb} \right|$$

Fig. 9 Pixels of Average difference across frames (ADF) calculation and N = 16

After error detection, error concealment can be activated to compensate the corrupted blocks. Error resynchronization can be achieved by inserting markers in the bitstream to know the boundary of the next decoding unit. In H.264/AVC, one frame can separated into slices and the decoding of one slice would not reference data in other slices. Therefore, if one slice data is corrupted, the error can be restricted in that slice thus the resynchronization is achieved. The other method is inserting refreshment frames, slices or macroblocks so that temporal error propagation can be stopped. [13], [14] and [15] are joint-source channel design for MPEG-4 video format. [13] and [14] simulated the performance of using Maximum A Posteriori (MAP) decoder under additive Markov channels (AMC) and the simulation environment are shown in Fig. 10. [15] combined the source state space with the channel state space to one finite state machine (Fig. 11) and the corresponding trellis decoding can be defined. [16] was a JCSD for H.264 motion vectors data to improve video quality.

(a)



(b)

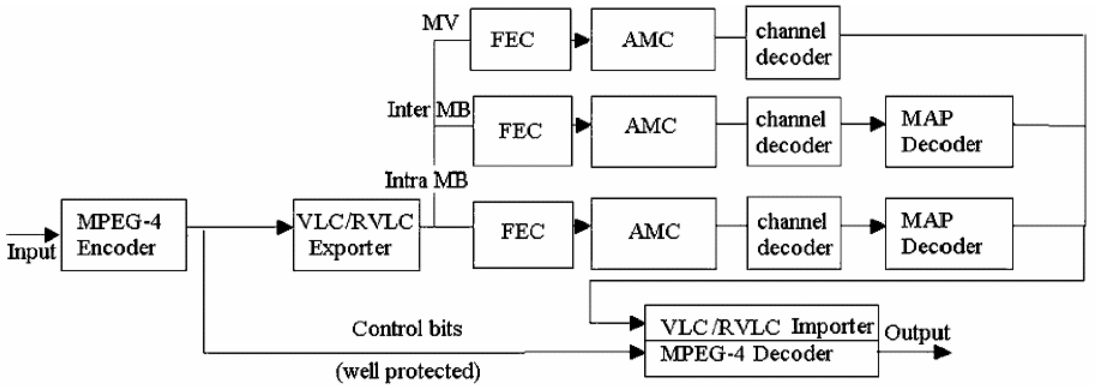


Fig. 10 Experimental Set-up for evaluating the performance of the MAP decoder in (a) [13] , (b) [14]

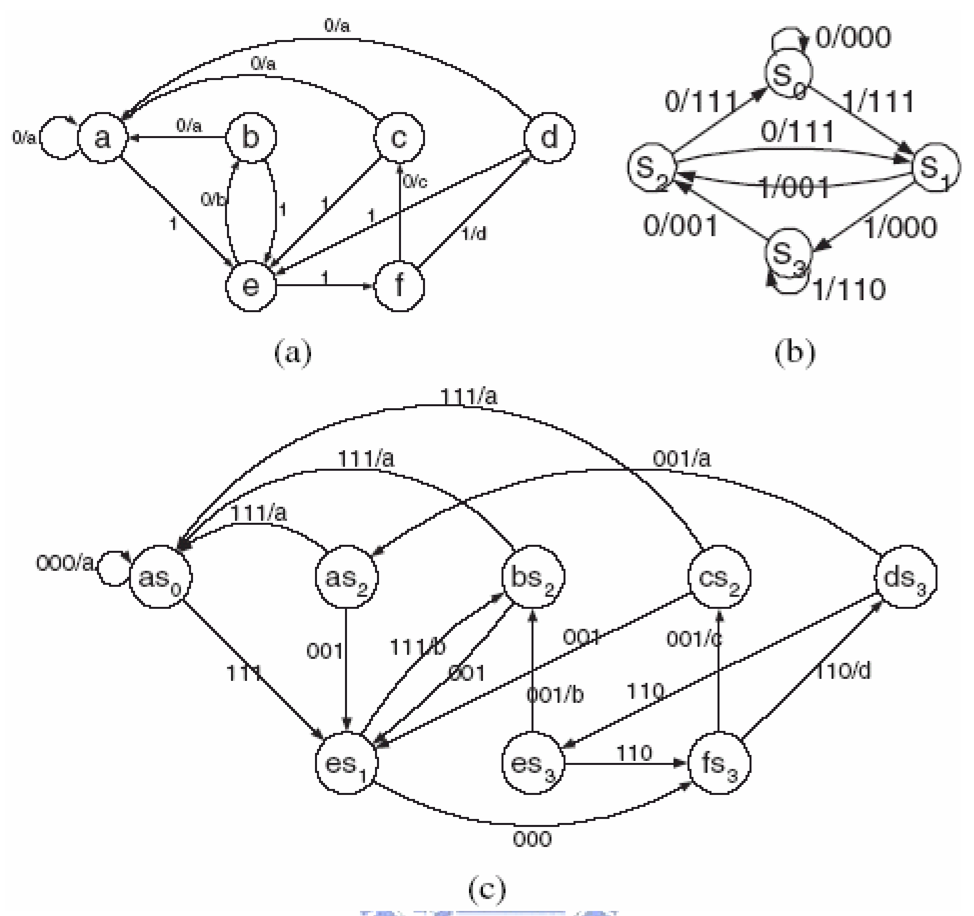


Fig. 11 Combing source and channel state space. (a) source state space. (b) channel state space. (c) integrated state space.

Chapter 3

Algorithm of Memory-based VLC Decoding

3.1 Conventional Group-based VLC Algorithm and Decoding

Flow

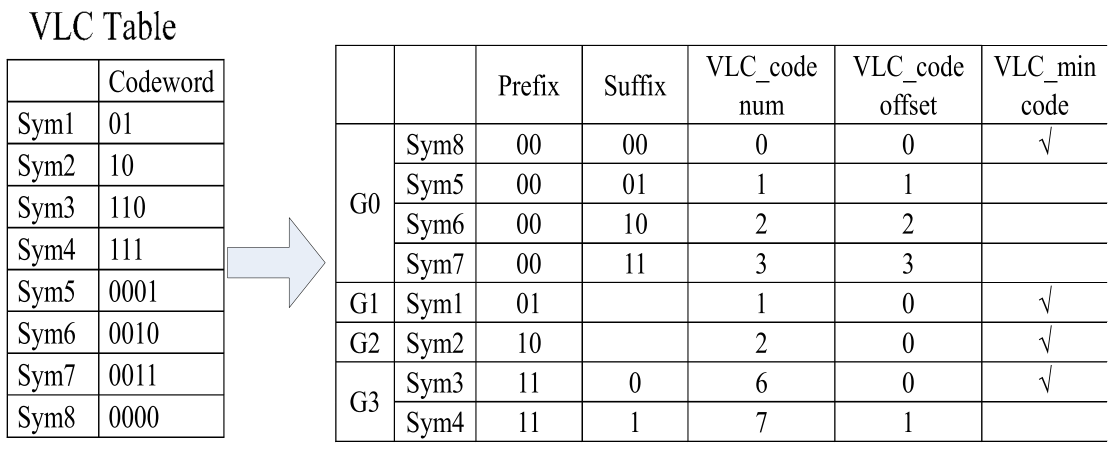
This section was previously developed and verified by Bai-Jue Hsieh in [2]. The intention of this section is to quickly talk about the concept of conventional group-based VLC decoder system and how it works.

3.1.1 Definition of Codeword Groups

For a coding table, we separated codewords into groups. Codewords in a group has the following properties:

1. In a group, the codeword can be treated as a binary number which is codeword length-bit long, called VLC_codenum, since the codeword length is the same.
2. The codeword that has the smallest VLC_codenum in a group is denoted VLC_mincode.
3. A VLC_codeoffset is the offset value between the VLC_mincode and the VLC_codenum.

For the example shown in Fig. 12, the VLC table has 8 codewords and the codewords with the same length and prefix are classified as the same group. The codewords in G0 have 4 bits with 2-bit prefix and 2-bit suffix. Therefore, the VLC_codenum are the 0,1,2,3 thus the VLC_codeoffset of Sym5, Sym6, Sym7 are 1, 2 and 3, respectively. Because 01 and 10 have different prefix so they belong to different groups although they have the same length. The codewords in G3 have 3 bits in length and VLC_codenum are 6 and 7; the VLC_codeoffset are 0 and 1 for Sym3 and Sym4.



VLC Table

	Codeword
Sym1	01
Sym2	10
Sym3	110
Sym4	111
Sym5	0001
Sym6	0010
Sym7	0011
Sym8	0000

		Prefix	Suffix	VLC_code num	VLC_code offset	VLC_min code
G0	Sym8	00	00	0	0	√
	Sym5	00	01	1	1	
	Sym6	00	10	2	2	
	Sym7	00	11	3	3	
G1	Sym1	01		1	0	√
G2	Sym2	10		2	0	√
G3	Sym3	11	0	6	0	√
	Sym4	11	1	7	1	

Fig. 12 Grouping of codewords in the table

3.1.2 Intra-Group Decoding Procedure

In the same group, the codewords have arithmetic relationship from the VLC_codenum, VLC_codeoffset and VLC_mincode. Thus, only the VLC_mincode information of every group is stored in memory and we can find the information about other codewords by means of computation of the offset. In other words, if the symbols of the same group are allocated in the continuous location in the symbol memory and the decoded symbol address can be known by adding offset amount to a base address.

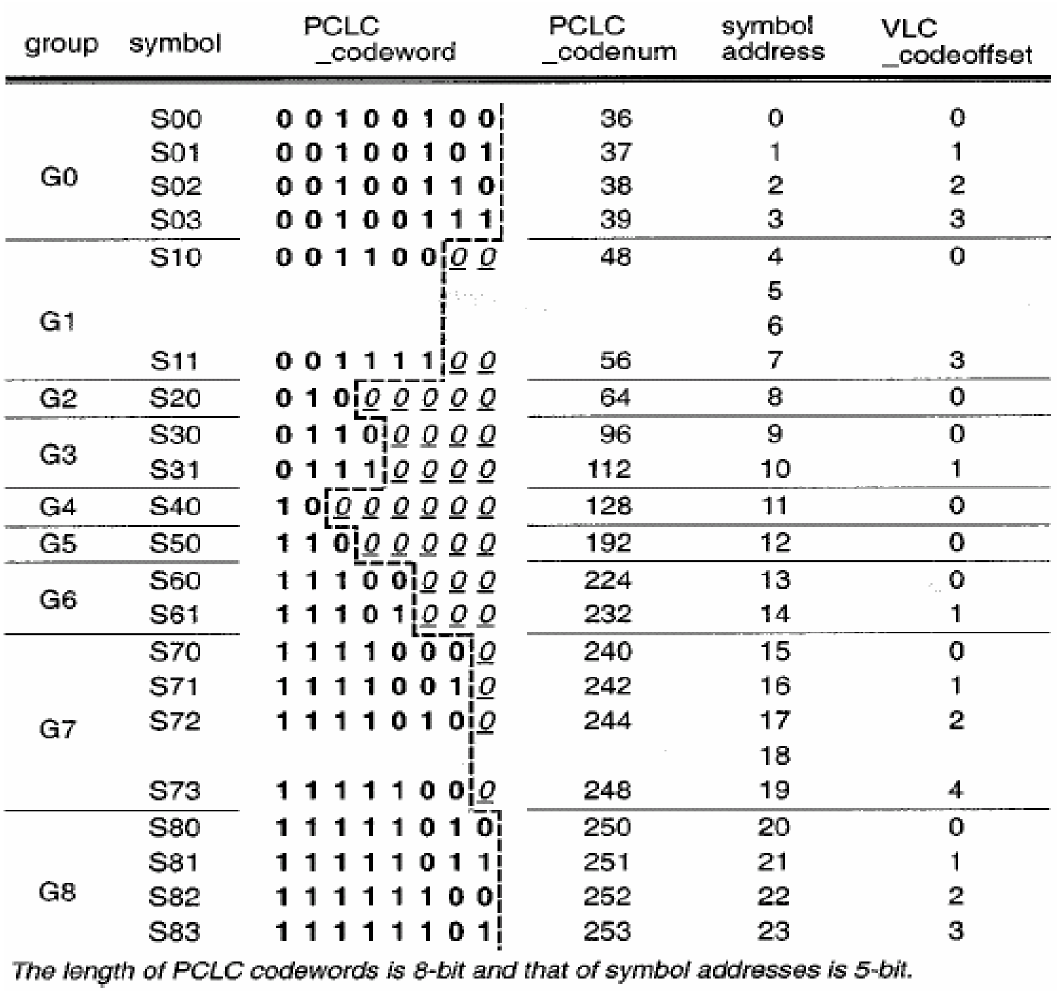
Fig. 13 shows the information within one group. For example, if the 0000011 is received, the offset equals to 3 and thus the symbols address is 3 + 60 = 63 to that S3 is decoded.

Symbol	Prefix	Suffix	VLC_codenum	VLC_codeoffset	Address
S1	0000	000	0	0	60
S2		001	1	1	61
S3		010	2	2	62
S4		011	3	3	63
S5		100	4	4	64
S6		101	5	5	65
S7		110	6	6	66
S8		111	7	7	67

Fig. 13 One group with address assignment

3.1.3 Group Searching Scheme

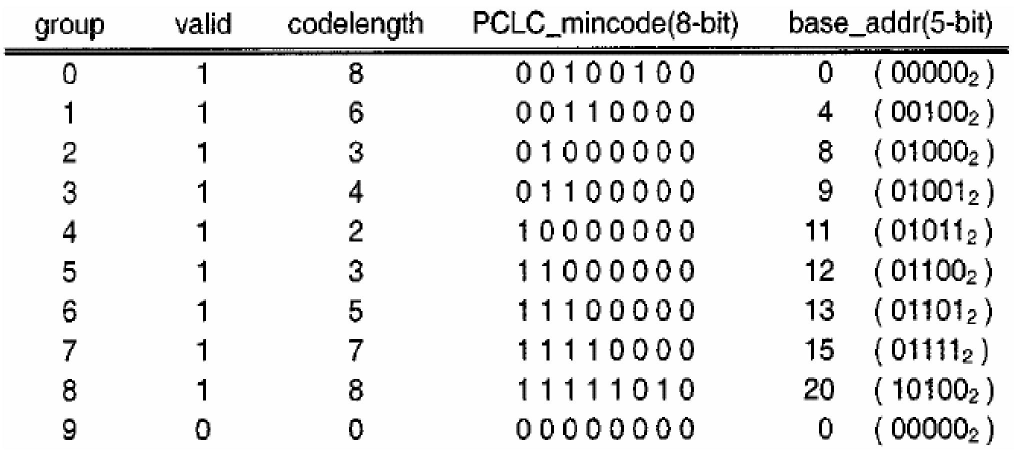
To search the group that the correct symbol locates in, Pseudo Constant Length Codeword (PCLC) is used. In the table, all codewords are extended to the length of the longest codeword by appending 0’s behind the codewords. All PCLCs have the same length and can be view as binary numbers, PCLC_codenum. All PCLCs are organized in ascending order so that PCLC_codenum*0* < PCLC_codenum*1* < PCLC_codenum*2*…PCLC_codenum*n* and thus PCLC_mincode*0* < PCLC_mincode*1* < PCLC_mincode*2*….PCLC_mincode*n*. Next, the base addresses are assigned to PCLC_mincode and base_addr*0* < base_addr*1* < base_addr*2*…..base_addr*n*. The example of the intra-/inter- group symbol memory mapping is shown in Fig. 14 and the group information of the tables is shown in Fig. 15, where the valid bit means whether the table contains this group or not.



group	symbol	PCLC _codeword	PCLC _codenum	symbol address	VLC _codeoffset
G0	S00	0 0 1 0 0 1 0 0	36	0	0
	S01	0 0 1 0 0 1 0 1	37	1	1
	S02	0 0 1 0 0 1 1 0	38	2	2
	S03	0 0 1 0 0 1 1 1	39	3	3
G1	S10	0 0 1 1 0 0 *0 0*	48	4	0
				5	
				6	
	S11	0 0 1 1 1 1 *0 0*	56	7	3
G2	S20	0 1 0 *0 0 0 0 0*	64	8	0
G3	S30	0 1 1 0 *0 0 0 0*	96	9	0
	S31	0 1 1 1 *0 0 0 0*	112	10	1
G4	S40	1 0 *0 0 0 0 0 0*	128	11	0
G5	S50	1 1 0 *0 0 0 0 0*	192	12	0
G6	S60	1 1 1 0 0 *0 0 0*	224	13	0
	S61	1 1 1 0 1 *0 0 0*	232	14	1
G7	S70	1 1 1 1 0 0 0 *0*	240	15	0
	S71	1 1 1 1 0 0 1 *0*	242	16	1
	S72	1 1 1 1 0 1 0 *0*	244	17	2
				18	
	S73	1 1 1 1 1 0 0 *0*	248	19	4
G8	S80	1 1 1 1 1 0 1 0	250	20	0
	S81	1 1 1 1 1 0 1 1	251	21	1
	S82	1 1 1 1 1 1 0 0	252	22	2
	S83	1 1 1 1 1 1 0 1	253	23	3

The length of PCLC codewords is 8-bit and that of symbol addresses is 5-bit.

Fig. 14 PCLC table and intra-/inter- group symbol memory mapping



group	valid	codelength	PCLC_mincode(8-bit)	base_addr(5-bit)
0	1	8	00100100	0 (00000_2)
1	1	6	00110000	4 (00100_2)
2	1	3	01000000	8 (01000_2)
3	1	4	01100000	9 (01001_2)
4	1	2	10000000	11 (01011_2)
5	1	3	11000000	12 (01100_2)
6	1	5	11100000	13 (01101_2)
7	1	7	11110000	15 (01111_2)
8	1	8	11111010	20 (10100_2)
9	0	0	00000000	0 (00000_2)

Fig. 15 Group information of the table in Fig. 14

Like the PCLC_codenum, a segment of bitstream with the same length of PCLC can be treated as a binary number, bitstream_num. The group searching scheme can be achieved by computed the (bitstream_num – PCLC_mincode*i*). The hit condition of the decoded symbol located the group Gn is PCLC_mincode*n* < bitstream_num < PCLC_mincode*n+1*.

The overall decoding process of the group-based algorithm is as follows: Assume the bitstream input is 001111100110…..

1. Do group searching

➔PCLC_mincode1(8’b00110000)<bitstream_num < PCLC_mincode2(8’b01000000)

➔The matching group: G0

2. **Send group information**

➔ code length = 6-bit, PCLC_mincode = 8’b00110000, base_addr(5-bit) = 5’b00100.

3. **Find the valid VLC_codeoffset, which is the code length most significant bits of the result of subtracting the PCLC_mincode from the bitstream_num**

➔Bitstream_num(8’b00111110) – PCLC_mincode(8’b00110000) = 8’b00001110.

➔The valid VLC_codeoffset = 6’b000011= 3.

4. **Extract the VLC_codeoffset operand, which has the same word length as the symbol address**

➔VLC_codeoffset = 5’b00011 = 3.

5. **Calculate the decoded symbol address**

➔symbol_addr = base_addr(5’b00100) + VLC_codeoffset(5’b00011) = 5’b00111= 7.

6. **Fetch the decoded symbol**

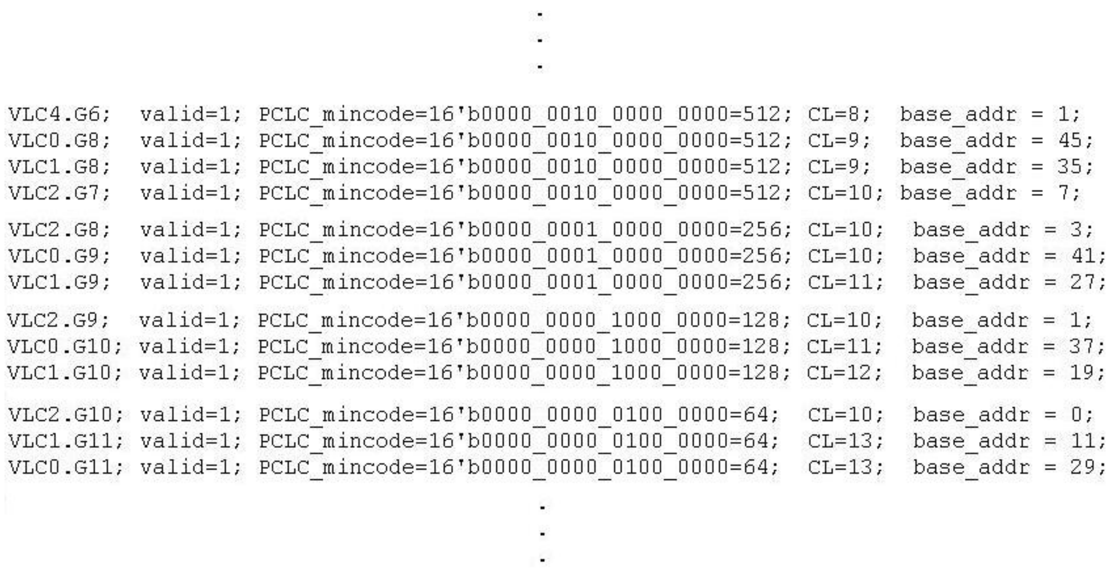
➔ sym_memory[7] = S11.

3.2 Conventional Multi-Table Merging Algorithm and Decoding Flow

This section was previously developed and verified by Bai-Jue Hsieh in [2]. The intention of this section is to quickly talk about the concept of conventional multi-table merged VLC decoder system and how it works

3.2.1 Collection of Group Information of All Coding Tables

According to group-based decoding algorithm, group information of all tables can be known and the PCLCs of groups of a table can be viewed as a codeword in that table. Therefore, all PCLCs are collected in the ascending order as Fig. 16 shows. In this figure, all group information items are ordered according to the magnitude of PCLC_mincode and there are 13 items.

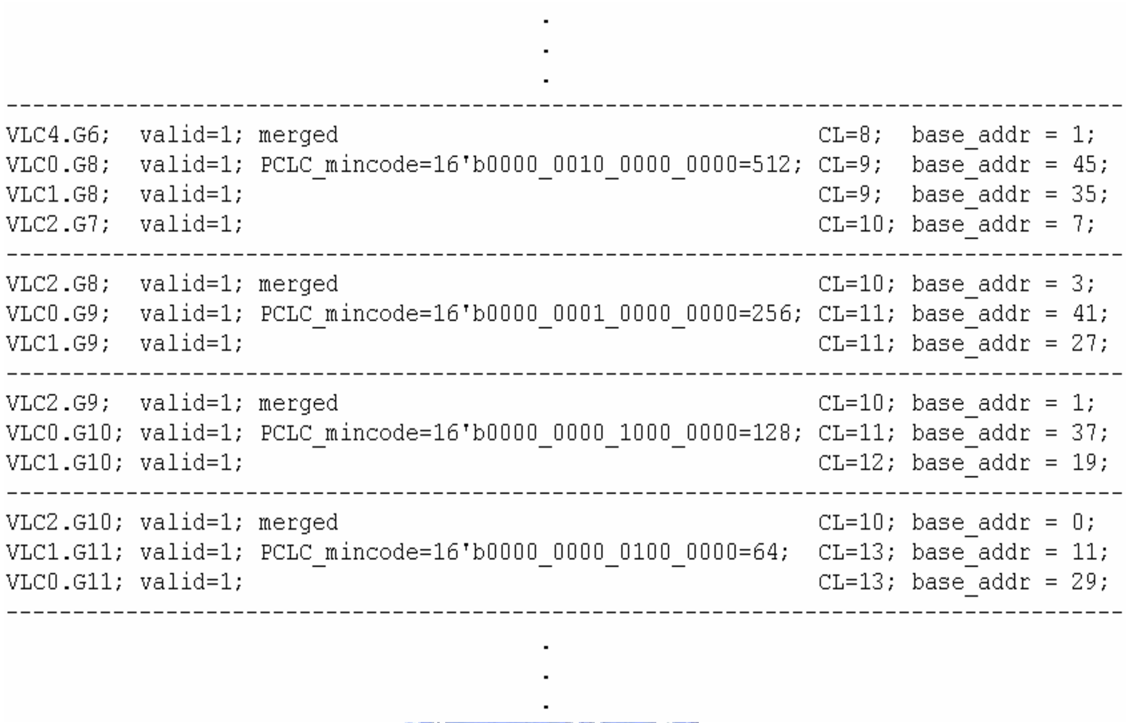


```
                                        .
                                        .
                                        .
VLC4.G6;  valid=1; PCLC_mincode=16'b0000_0010_0000_0000=512; CL=8;  base_addr = 1;
VLC0.G8;  valid=1; PCLC_mincode=16'b0000_0010_0000_0000=512; CL=9;  base_addr = 45;
VLC1.G8;  valid=1; PCLC_mincode=16'b0000_0010_0000_0000=512; CL=9;  base_addr = 35;
VLC2.G7;  valid=1; PCLC_mincode=16'b0000_0010_0000_0000=512; CL=10; base_addr = 7;

VLC2.G8;  valid=1; PCLC_mincode=16'b0000_0001_0000_0000=256; CL=10;  base_addr = 3;
VLC0.G9;  valid=1; PCLC_mincode=16'b0000_0001_0000_0000=256; CL=10;  base_addr = 41;
VLC1.G9;  valid=1; PCLC_mincode=16'b0000_0001_0000_0000=256; CL=11;  base_addr = 27;

VLC2.G9;  valid=1; PCLC_mincode=16'b0000_0000_1000_0000=128; CL=10;  base_addr = 1;
VLC0.G10; valid=1; PCLC_mincode=16'b0000_0000_1000_0000=128; CL=11;  base_addr = 37;
VLC1.G10; valid=1; PCLC_mincode=16'b0000_0000_1000_0000=128; CL=12;  base_addr = 19;

VLC2.G10; valid=1; PCLC_mincode=16'b0000_0000_0100_0000=64;  CL=10;  base_addr = 0;
VLC1.G11; valid=1; PCLC_mincode=16'b0000_0000_0100_0000=64;  CL=13;  base_addr = 11;
VLC0.G11; valid=1; PCLC_mincode=16'b0000_0000_0100_0000=64;  CL=13;  base_addr = 29;
                                        .
                                        .
                                        .
```

Fig. 16 Part of group information of several tables.

3.2.2 Codeword Merging

From Fig. 16, the PCLCs of some groups are identical to others. If all these PCLCs are stored, there is much redundancy. As shown in Fig. 17, we can separate the groups with the same PCLC into the same group and only one PCLC is stored. This reduces storage space. We can see that there are 13 items and after codeword merging, the number of items reduces to 4.



```
                                      .
                                      .
                                      .
--------------------------------------------------------------------------------
VLC4.G6;  valid=1; merged                                 CL=8;  base_addr = 1;
VLC0.G8;  valid=1; PCLC_mincode=16'b0000_0010_0000_0000=512; CL=9;  base_addr = 45;
VLC1.G8;  valid=1;                                        CL=9;  base_addr = 35;
VLC2.G7;  valid=1;                                        CL=10; base_addr = 7;
--------------------------------------------------------------------------------
VLC2.G8;  valid=1; merged                                 CL=10; base_addr = 3;
VLC0.G9;  valid=1; PCLC_mincode=16'b0000_0001_0000_0000=256; CL=11; base_addr = 41;
VLC1.G9;  valid=1;                                        CL=11; base_addr = 27;
--------------------------------------------------------------------------------
VLC2.G9;  valid=1; merged                                 CL=10; base_addr = 1;
VLC0.G10; valid=1; PCLC_mincode=16'b0000_0000_1000_0000=128; CL=11; base_addr = 37;
VLC1.G10; valid=1;                                        CL=12; base_addr = 19;
--------------------------------------------------------------------------------
VLC2.G10; valid=1; merged                                 CL=10; base_addr = 0;
VLC1.G11; valid=1; PCLC_mincode=16'b0000_0000_0100_0000=64;  CL=13; base_addr = 11;
VLC0.G11; valid=1;                                        CL=13; base_addr = 29;
--------------------------------------------------------------------------------
                                      .
                                      .
                                      .
```

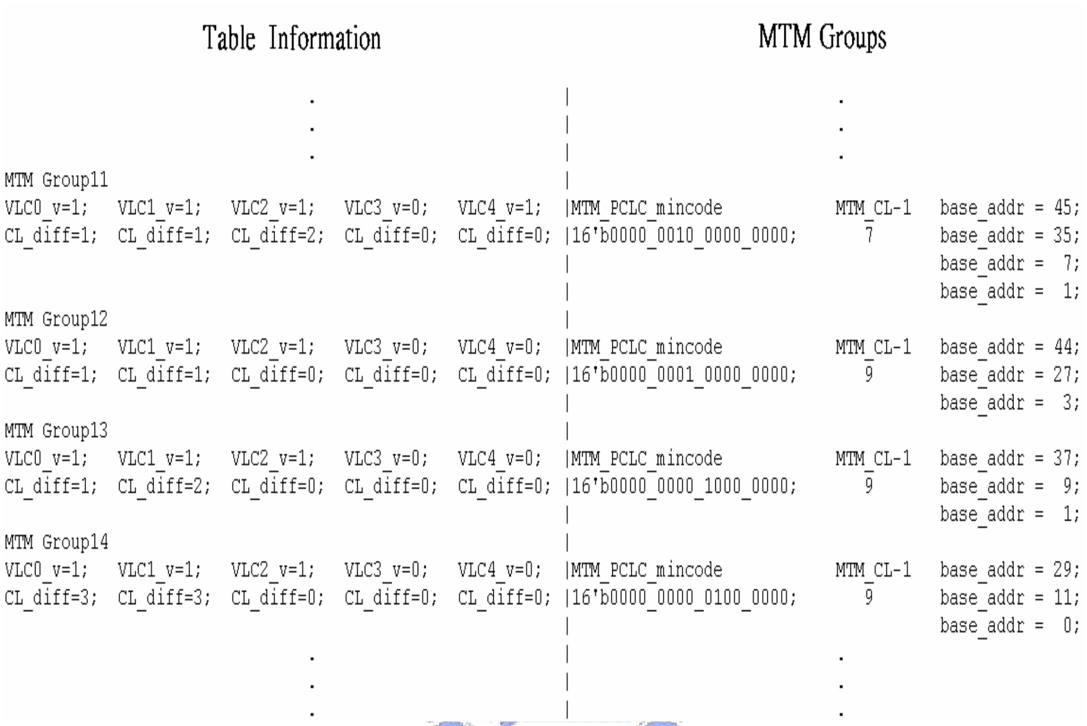
Fig. 17 One portion of the groups after codeword merging process.

3.2.3 Prefix Merging

The prefix merging check any two neighbor groups after codeword merging. When the longest VLC_mincode in a group is the prefix the PCLC_mincode in the adjacent codeword group, they can be merged together to one group. In the case of Fig. 17, there is no prefix merging can be operated.

3.2.4 Set Table Information

After merging process, merged groups and PCLC_mincodes are MTM groups and MTM_PCLC_mincodes, respectively. The table information of a coding table includes the valid-bit and the length of codewords. Because the shortest length of codeword is 1 bit and the length is from 1-bit to 16-bit, we just store (length-1),i.e. 0 ~ 15 in the memory to save memory space. After this shifting operation, the smallest (length-1) in all the groups is defined as MTM_CL-1 and stored in the group information memory. Therefore, the difference between the larger (length-1) and (MTM_CL-1) which is defined as CL_diff is stored in the table information memory. The memory space is further saved because the data redundancy among the lengths in a MTM group is exploited. The table information and group information are shown in the Fig. 18.



```
                         Table Information                                          MTM Groups

                                       .                  |                    .
                                       .                  |                    .
                                       .                  |                    .
MTM Group11                                               |
VLC0_v=1;  VLC1_v=1;  VLC2_v=1;  VLC3_v=0;  VLC4_v=1;  |MTM_PCLC_mincode         MTM_CL-1   base_addr = 45;
CL_diff=1; CL_diff=1; CL_diff=2; CL_diff=0; CL_diff=0; |16'b0000_0010_0000_0000;    7       base_addr = 35;
                                                          |                                    base_addr =  7;
                                                          |                                    base_addr =  1;
MTM Group12                                               |
VLC0_v=1;  VLC1_v=1;  VLC2_v=1;  VLC3_v=0;  VLC4_v=0;  |MTM_PCLC_mincode         MTM_CL-1   base_addr = 44;
CL_diff=1; CL_diff=1; CL_diff=0; CL_diff=0; CL_diff=0; |16'b0000_0001_0000_0000;    9       base_addr = 27;
                                                          |                                    base_addr =  3;
MTM Group13                                               |
VLC0_v=1;  VLC1_v=1;  VLC2_v=1;  VLC3_v=0;  VLC4_v=0;  |MTM_PCLC_mincode         MTM_CL-1   base_addr = 37;
CL_diff=1; CL_diff=2; CL_diff=0; CL_diff=0; CL_diff=0; |16'b0000_0000_1000_0000;    9       base_addr =  9;
                                                          |                                    base_addr =  1;
MTM Group14                                               |
VLC0_v=1;  VLC1_v=1;  VLC2_v=1;  VLC3_v=0;  VLC4_v=0;  |MTM_PCLC_mincode         MTM_CL-1   base_addr = 29;
CL_diff=3; CL_diff=3; CL_diff=0; CL_diff=0; CL_diff=0; |16'b0000_0000_0100_0000;    9       base_addr = 11;
                                                          |                                    base_addr =  0;
                                       .                  |                    .
                                       .                  |                    .
                                       .                  |                    .
```

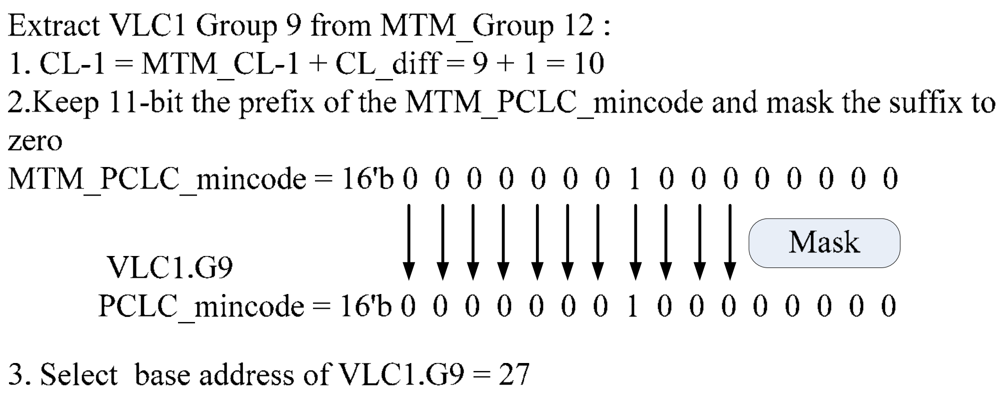
Fig. 18 Table information and group information

3.2.5 Base Address Merging

Although base addresses can be stored for different tables under the given group, the required memory space is large when the number of tables becomes large. [2] proposed a method that classify base address in to categories according to the numbers of table entries. For example, the table1 has 28 entries and table2 has 136 entries, the base addresses of them are classified into two categories: base_addr1 is 5-bit and base_addr2 is 8-bit. With the base address adjustment, different tables with the same category can use common set of base addresses.

3.2.6 Group Information Recovery

According to table information and group information in Fig. 18, the example of group information recovery is shown in Fig. 19. In the first step, (length-1) of VLC_mincode is computed by adding MTM_CL-1 and CL_diff. Second, the most length bits of the MTM_PCLC are assigned to PCLC_mincode while the remained bits are 0s. Finally, the base address is accessed according to base address selection.



Extract VLC1 Group 9 from MTM_Group 12 :

1. CL-1 = MTM_CL-1 + CL_diff = 9 + 1 = 10

2.Keep 11-bit the prefix of the MTM_PCLC_mincode and mask the suffix to zero

MTM_PCLC_mincode = 16'b 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

VLC1.G9

PCLC_mincode = 16'b 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

3. Select base address of VLC1.G9 = 27

Fig. 19 Example of group information recovery

Finally, Table 4and Table 5 shows the number of groups of every table and MTM groups in CAVLC and MPEG-2, respectively. The number of items is reduced greatly in both standards.

Table 3 The number of groups of tables in CAVLC and the number of MTM groups

	# of Group	# of symbols	# of group after MTM
Coeff_Token(0<=nC<2)	17	62	23
Coeff_Token(2<=nC<4)	16	62	
Coeff_Token(4<=nC<8)	11	62	
Coeff_Token(8<=nC)	7	62	
Coeff_Token(nC= -1)	8	14	
Total_Zeros(TC =1)	9	16	
Total_Zeros(TC =2)	8	15	
Total_Zeros(TC =3)	8	14	
Total_Zeros(TC =4)	7	13	
Total_Zeros(TC =5)	7	12	
Total_Zeros(TC =6)	7	11	
Total_Zeros(TC =7)	8	10	
Total_Zeros(TC =8)	7	9	
Total_Zeros(TC =9)	7	8	
Total_Zeros(TC =10)	6	7	

Total_Zeros(TC =11)	5	6	
Total_Zeros(TC =12)	5	5	
Total_Zeros(TC =13)	4	4	
Total_Zeros(TC =14)	3	3	
Total_Zeros(TC =15)	2	2	
Total_Zeros_ch (TC =1)	4	4	
Total_Zeros_ch (TC =2)	3	3	
Total_Zeros_ch (TC =3)	2	2	
Run_Before(ZL = 1)	2	2	
Run_Before(ZL = 2)	3	3	
Run_Before(ZL = 3)	3	4	
Run_Before(ZL = 4)	4	5	
Run_Before(ZL = 5)	4	6	
Run_Before(ZL = 6)	5	7	
Run_Before(ZL > 1)	11	15	

Table 4 The number of groups of tables in MPEG2 and the number of MTM groups
*: There are 9 locations are unused because the VLC_codnum in one groups are not continuously increment.

	# of Group	# of symbols	# of group after MTM
TB14	13	111	21
TB15	19	111*	

3.3 Modified MTM algorithm for Improvement of Memory Efficiency

Based on the basic concept of MTM algorithm, we applied the algorithm for all coding tables in CAVLC to achieve programmability. The tables include *Coeff_Token* (0 <= nC < 2, 2 <= nC < 4, 4 <= nC < 8, 8 <= nC, nC = -1), *Total_Zeros*(4x4), *Total_Zeros* (chroma DC 2x2) and *Run_Before*, up to 30 coding tables and the entry number range is from 2 ~ 62. That is, the conventional MTM algorithm must support 6 categories for 2-entry, 4-entry, 8-entry …64-entry tables. This will increase the cost overhead and critical path of the group detector in hardware. Besides, the base address adjustment will shift the base address to the maximum value of the group within the same category hence increase the unused locations in symbol memory. Take *Coeff_Token* (0 <= nC < 2, 2 <= nC < 4, 4 <= nC < 8, 8 <= nC) tables as a example, these four tables are 62-entry and they should belong to 6-bit address category. Fig. 20 shows that most base addresses are adjusted to meet the requirement and we can see that the total shift amount is 83+59+53+52=247. The base address implies the symbol address, as a result, there are 247 entries in symbol memory are unused after the adjustment procedure. The symbol length of *Coeff_Token* is 7-bit thus there 247 * 7 =1729 bits are unused.

```
   ====== Group0: MTM_CL-1=0        1000_0000_0000_0000 ======
VLC0.G0              61 +4 +4 +4 +5 +5 +7 +23
VLC1.G0              60 +5 +7 +4 +2 +4 +4 +4 +2 +21
VLC2.G0              54 +29 +7 +6 +1 +8 +8
VLC3.G0              30 +56 +9 +5 +5 +8
   ====== Group1: MTM_CL-1=2        0110_0000_0000_0000 ======
VLC1.G1              59 +5 +7 +4 +2 +4 +4 +4 +2
   ====== Group2: MTM_CL-1=1        0100_0000_0000_0000 ======
VLC0.G1              60 +4 +4 +4 +5 +5 +7
VLC1.G2              57 +5 +7 +4 +2 +4 +4 +4 +2
VLC2.G1              46 +29 +7 +6 +1
VLC3.G1              14 +56 +9 +5 +5
   ====== Group3: MTM_CL-1=4        0011_0000_0000_0000 ======
VLC1.G3              55 +5 +7 +4 +2 +4 +4 +4
   ====== Group4: MTM_CL-1=2        0010_0000_0000_0000 ======
VLC0.G2              59 +4 +4 +4 +5 +5
VLC1.G4              51 +5 +7 +4 +2 +4 +4 +4
VLC2.G2              38 +29 +7 +6 +1
VLC3.G2              6   +56 +9 +5 +5
   ====== Group5: MTM_CL-1=4        0001_1000_0000_0000 ======
VLC0.G3              58 +4 +4 +4 +5
   ====== Group6: MTM_CL-1=3        0001_0000_0000_0000 ======
VLC0.G4              56 +4 +4 +4 +5
VLC1.G5              47 +5 +7 +4 +2 +4 +4
VLC2.G3              30 +29 +7 +6 +1
VLC3.G3              3   +56 +9 +5
   ====== Group7: MTM_CL-1=5        0000_1100_0000_0000 ======
VLC0.G5              55 +4 +4 +4
```

```
VLC3.G4         2 +56 +9
  ====== Group8: MTM_CL-1=4       0000_1000_0000_0000 ======
VLC0.G6         53 +4 +4 +4
VLC1.G6         43 +5 +7 +4 +2 +4
VLC2.G4         22 +29 +7 +6 +1
  ====== Group9: MTM_CL-1=5       0000_0100_0000_0000 ======
VLC0.G7         49 +4 +4
VLC1.G7         39 +5 +7 +4 +2
VLC2.G5         14 +29 +7 +6 +1
VLC3.G5         1 +56
  ====== Group10: MTM_CL-1=8      0000_0011_1000_0000 ======
VLC2.G6         13 +29 +7 +6
  ====== Group11: MTM_CL-1=6      0000_0010_0000_0000 ======
VLC0.G8         45 +4
VLC1.G8         35 +5 +7 +4 +2
VLC2.G7         7   +29 +7 +6
  ====== Group12: MTM_CL-1=7      0000_0001_0000_0000 ======
VLC0.G9         41 +4
VLC1.G9         27 +5 +7 +4 +2
VLC2.G8         3   +29 +7 +6
  ====== Group13: MTM_CL-1=8      0000_0000_1000_0000 ======
VLC0.G10        37
VLC1.G10        19 +5 +7 +4 +2
VLC2.G9         1   +29 +7
  ====== Group14: MTM_CL-1=9      0000_0000_0100_0000 ======
VLC0.G11        29
VLC1.G11        11 +5 +7 +4 +2
VLC2.G10        0   +29
  ====== Group15: MTM_CL-1=12     0000_0000_0011_0000 ======
VLC1.G12        9   +5 +7 +4
  ====== Group16: MTM_CL-1=10     0000_0000_0010_0000 ======
VLC0.G12        21
VLC1.G13        5   +5 +7 +4
  ====== Group17: MTM_CL-1=13     0000_0000_0001_0000 ======
VLC0.G13        13
VLC1.G14        1   +5 +7
  ====== Group18: MTM_CL-1=12     0000_0000_0000_1000 ======
```

VLC0.G14	5	
VLC1.G15	0	+5
====== Group19: MTM_CL-1=15	0000_0000_0000_0100 ======	
VLC0.G15	1	
====== Group20: MTM_CL-1=14	0000_0000_0000_0010 ======	
VLC0.G16	0	
====== Group21: MTM_CL-1=0	0000_0000_0000_0000 ======	
VLC3.G6	0	

Fig. 20 Base address adjustment procedure. The first number after VLC*i*. G*n* is base address and the +K is the adjusted amount of base address.

From this reason, base address adjustment is not considered as the method of reducing memory space. Nonetheless, direct record of the base address as MTM group information also makes many unused locations in the group information memory. Because there are many small tables in CAVLC and they have short codewords and a few of entries. Usually, the small tables use just one or two groups and other groups are invalid for them. In other words, many entries of group information are unused if the table is small as shown in Fig. 21. In Fig. 21, "TB" represents large tables which have long codewords and many entries while "tb" represents small tables which almost have short codewords and a few entries. The blank blocks is the filed that is unused by the table, i.e., validbit = 0 for that group of a table. For example, TB*N* does not contain GP0 and valid bit for GP0 is 0 while valid bit for other groups are 1. Also, many fields in the Fig. 21 for tb0~tb*m* are unused.

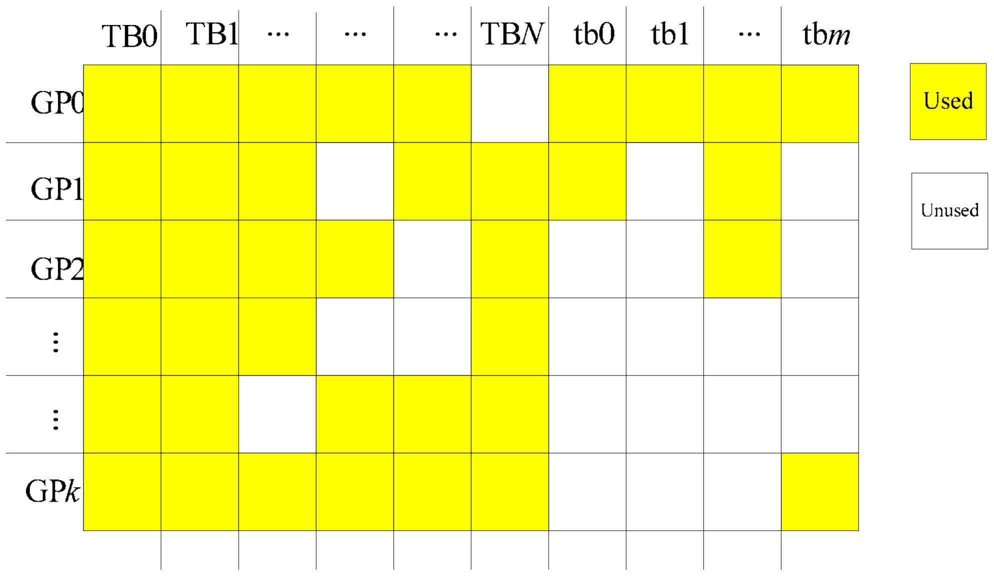


Fig. 21 Unused locations for the information memory.

Due to the above phenomenon, the modified organization of group information and table information is proposed. First, we use the longest codeword of a table as the determination of whether the table is large or small. We use 4 as the threshold. Therefore, there are 14 small tables and 16 large tables among CAVLC. Next, the practical storage unit is separated in two parts: one for small tables and one for large tables. By doing this partitioning, the unused locations can be reduced. Originally, the size of the memory space is (n + m) × k and the total size is reduced to n × k + m × ks, where ks is number of groups used by small tables, k is number of groups used by large tables, m is number of small tables, and n is number of large tables. In addition, the PCLC of small tables is shorter than the longest PCLC among all tables. Therefore, size of group information can also be reduced. Table 5 shows the comparison of size for conventional MTM without base address adjustment and the modified memory allocation.

Table 5 Space of the conventional MTM without base address adjustment and the modified memory allocation

	Group information size	Table information size	Total
Conventional	23×(16+4+30×8) = 5980 bits	23×30×(1+2) = 2070 bits	8050 bits
Modified	23×(16+4+16×8) + 5×(4+2+14×8) = 3994 bits	23×(4×16) + 5×(14×3) = 1620 bits	5676 bits

3.4 Symbol Memory Allocation

Generally, the symbols are store in the symbol memory and symbol lengths of different tables are different. If only one symbol memory is used, the word length must be the length of the longest symbols among all tables. This allocation leads to some wasted space for shorter symbols. In [2], there are several symbol memories with different kinds of word length in order to save the space. The symbol length is either 7-bit (*Coeff_Token*) or 4-bit (*Total_Zeros* or *Run_Before*) in CAVLC tables and that of MPEG-2(TB14 and TB15) is 11-bit. As a result, 256 × 11 + 256 × 7 + 256 × 4 = 5632 bits are used in symbol memory. In the proposed allocation, we can use only a symbol memory with 256x11 bits to store all symbols in CAVLC, TB14 and TB15.

That is, the symbols in *Coeff_Token* and *Total_Zeros/Run_Before* can be concatenated into 11-bit words and stored in the 256 × 11 symbol memory. The overhead includes a mask and the multiplexer to choose the format of the symbol according to the standards and decoding tables as shown in Fig. 22. Besides, the start positions of tables are stored in a small register files. As CAVLC is used, we can select the most significant 7 bits of the memory output for *Coeff_Token* symbols or the least significant 4 bits for *Total_Zeros/Run_Before* symbols; the whole word is assigned to the symbols for MPEG2.

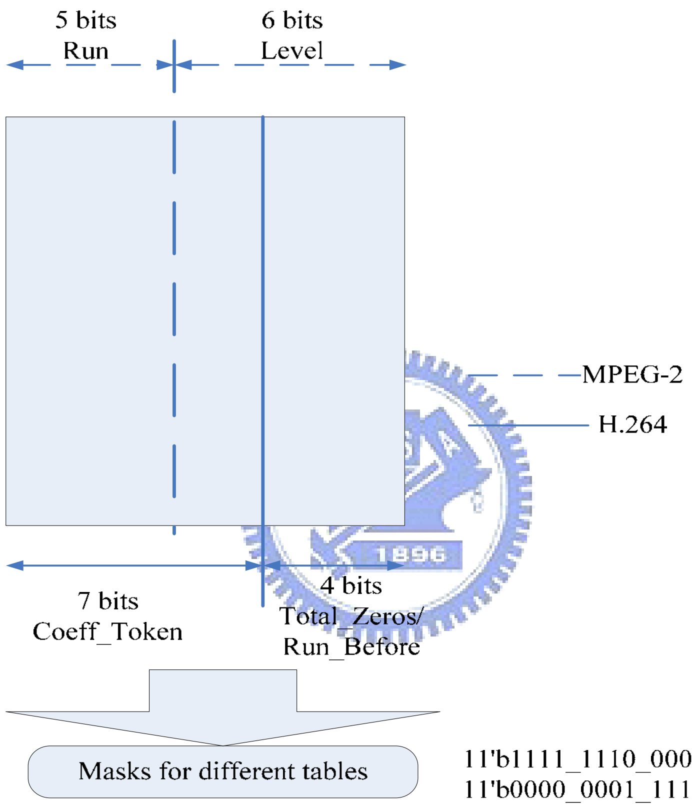


Fig. 22   256-entry symbol memory allocation in VLC decoder for MPEG-2 and H.264 standards.s

Chapter 4

Error Resynchronization

4.1 Concept of Error Resynchronization

In H.264/AVC error resilient tools, the most important feature is slicing. A slice consists of a sequence of macroblocks (Fig. 23) and the intra-prediction in one slice does not refer to data belongs to the other slices in one frame. If one slice is corrupted during the transmission, the error would not propagate to other slice regions so that the corrupted data is restricted within that slice. Insertion of refreshment frames, slices or macroblocks is a method to stop error propagation in the temporal domain. In addition, [17] mentioned that the insertion of additional markers in the bitstream can achieve VLC resynchronization. From these three methods, we can find that the error propagation can be stopped in a certain region and the following bitstream can be correctly decoded. That is, error resynchronization is achieved. However, these methods have bitstream overhead and increase needed data bandwidth.

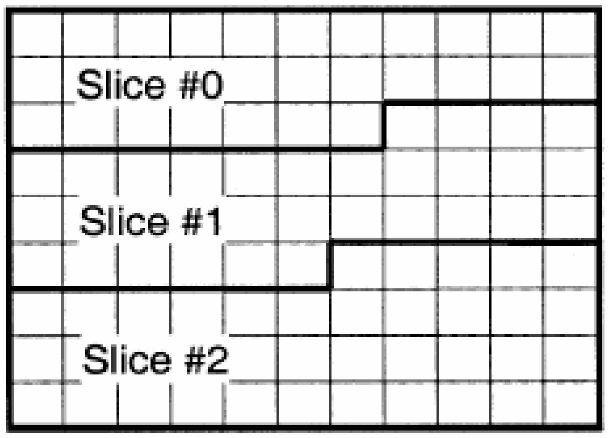


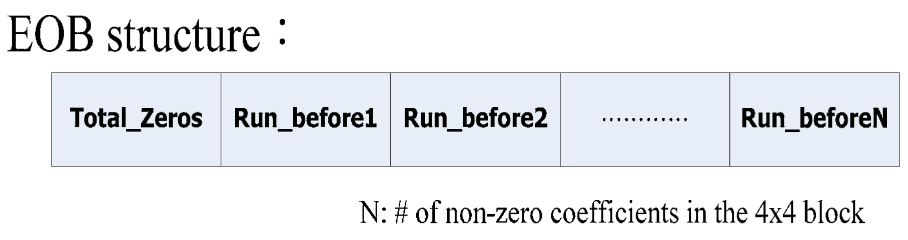
Fig. 23 Division of a picture into several slices.

To achieve resynchronization in block-level without bitstream overhead, information remained in bitstream after compression is needed to help find the resynchronization point. However, less information left in H.264 bitstream can provide error detection ability or resynchronization because of its high compression performance. To accomplish error resynchronization, the position of the next decoding unit is necessary so that error will be stopped until the current decoding units. In the following sections, we focus on the I-frame error resynchronization because I-frames are much important than P-frame and I-frame is reference of P-frame.

4.2 Proposed Scheme for Error Resynchronization

The smallest decoding unit in H.264/AVC is 4x4 block and the position of the block can only be known by end of block symbol. Unlike the previous video standards, there is no end-of-block symbol in CAVLC. Take MPEG-2 as an example, TB-14 and TB-15 define the end of block symbol with codeword “10” and “0110”, respectively. As a result, we can try to find the block boundary by searching “10” or “0110” in the bitstream. A further insight into the tables makes these two codewords mistakenly predict the block boundaries because other codewords also contain “10” or “0110”. For instance, codewords of (run, level) = (4, 1), (7, 1) and (2, 2) contain “10” in TB-14; codewords of (run, level) = (4, 1), (1, 2) and (16, 1) contain “0110” in TB-15.

From the conventional CAVLC decoding process, a block with non-zero coefficients has the following steps to complete the decoding: 1. decoding of total coefficients and trailing ones, 2. decoding of sign of trailing ones, 3. decoding of levels, 4. decoding of total zeros, 5. decoding of runs before every non-zero coefficients. The last two steps are about total zeros and runs. Therefore, we can say that end of block consists of these two syntax elements. In bitstream, all combinations of the codewords of theses two tables that meet the rule of CAVLC decoding can be viewed as the EOBs. Since the constructed EOBs have many possibilities, we can choose those ones which are not one segment of combinations of other codewords in the other tables. Once the EOBs are known, the block boundaries can be predicted by them and thus the error can be limited until the current block. Then, the decoding of the next block is resynchronized and correct. The EOB format is shown in Fig. 24.


EOB structure：

Total_Zeros	Run_before1	Run_before2	…………	Run_beforeN

N: # of non-zero coefficients in the 4x4 block

Fig. 24 The format of EOB

4.3 EOB Construction

To know all combinational series of codewords from *Total_Zeros(T.Z.)* and *Run_Before(R.B.)* tables, we must know all kinds of distribution of coefficients in one 4x4 block. For different distribution of non-zero coefficients, the combinations of the *T.Z.* codeword and the *R.B.* codewords are different. The total number of combination can be computed by the following equation (1):

$$Total = \sum_{k=1}^{16} C_k^{16} \text{ , where } k \text{ is the number of coefficients in 4x4 block} \qquad (1)$$

Therefore, there are totally 65535 kinds of EOB and histogram of the distribution is shown in Fig. 25.

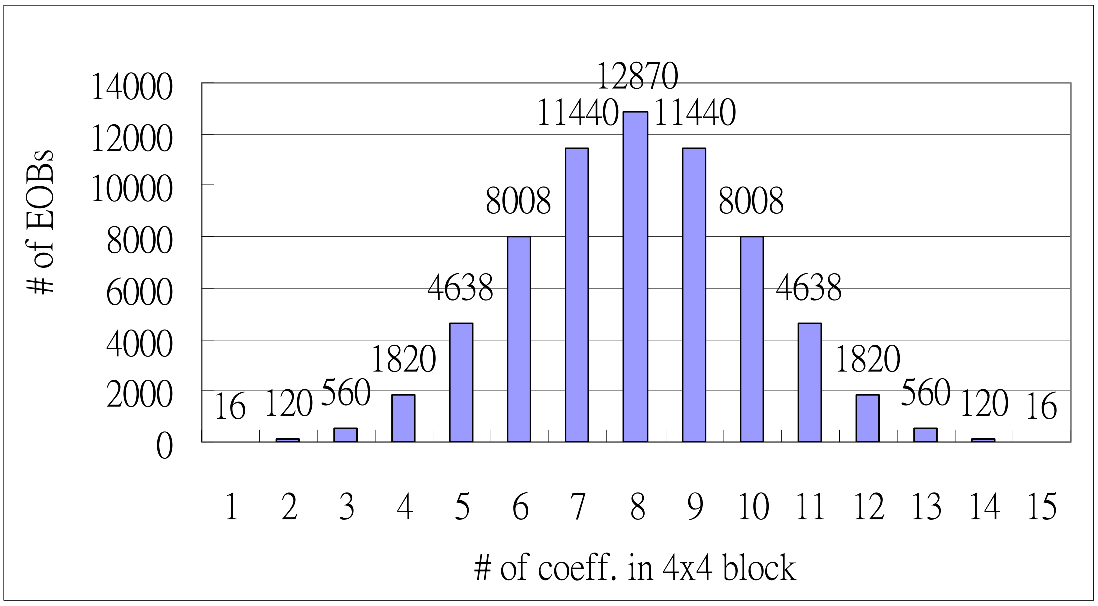


Fig. 25 Distribution of EOB number at different numb of coefficients

The next step verifies the EOBs that are not segments of combination of other codewords. The EOB is viewed as a sliding window through the whole bitstream and collect those EOBs that only occur at the exact positions. However, the number of total EOBs is large and the total bit of the bitstream is much larger, thus the simulation time is too long. Therefore, we choose another method to reduce the simulation time.

In this method, 300 I-frames are chosen to generate the all of the EOB’s of each frame individually. Next, the number of EOBs in one frame is reduced by eliminating short EOBs because short EOBs are more easily found in other places in the bitstream. With EOB positions which are known, the reduced EOBs set is checked in a certain range to determine whether one EOB is unique within the given range. As Fig. 4 shows, the EOB is viewed as a sliding window and check the correlation between the EOB and bitstream. The range is set because in real cases, EOB is not necessary to be unique globally thus only local uniqueness is checked.

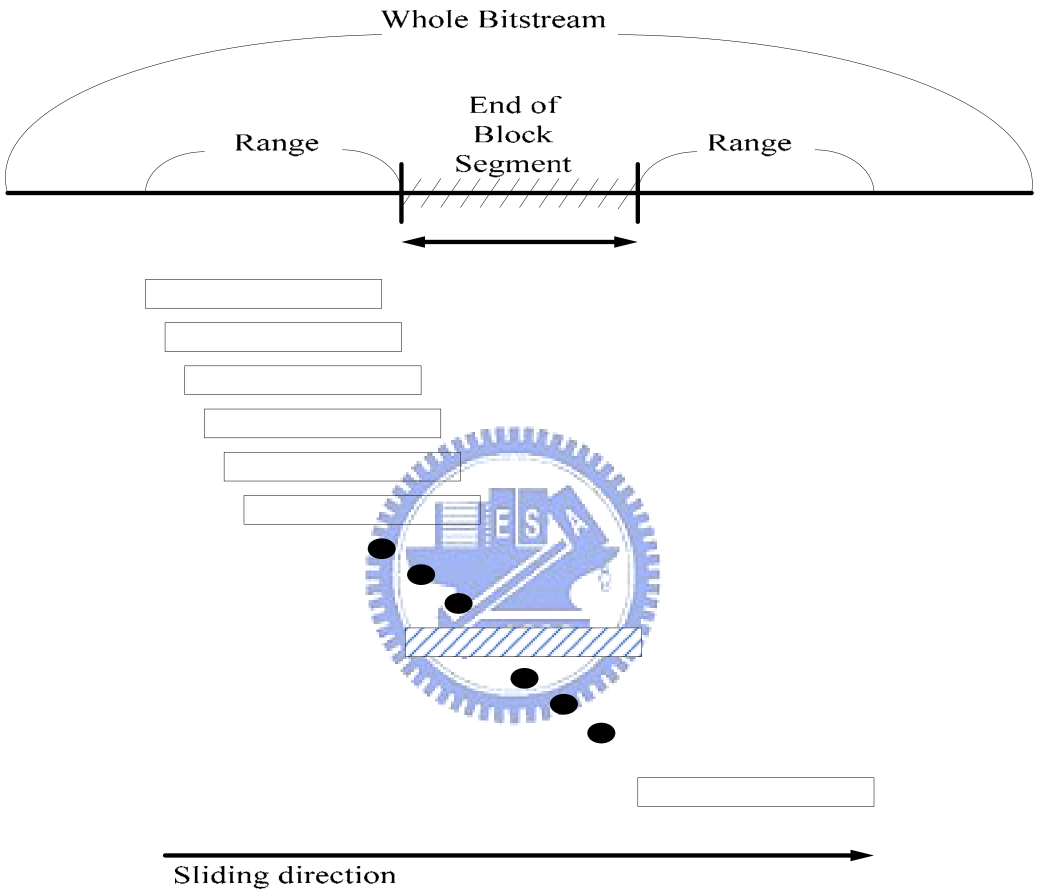


Fig. 26 Checking whether the EOB is unique within the given range or not

After all EOBs and the corresponding bitstream are checked, the EOBs are reduced because the EOBs that occurred multiple times are removed as shown in Fig. 27. The remained EOBs are defined as intra-set. To ensure EOBs can survive in other frames, they are also checked their uniqueness in other frames. The further reduced set is called the inter-set as shown in Fig. 28. After inter-checking, the remained set can be the error resynchronization information and stored in the memory.

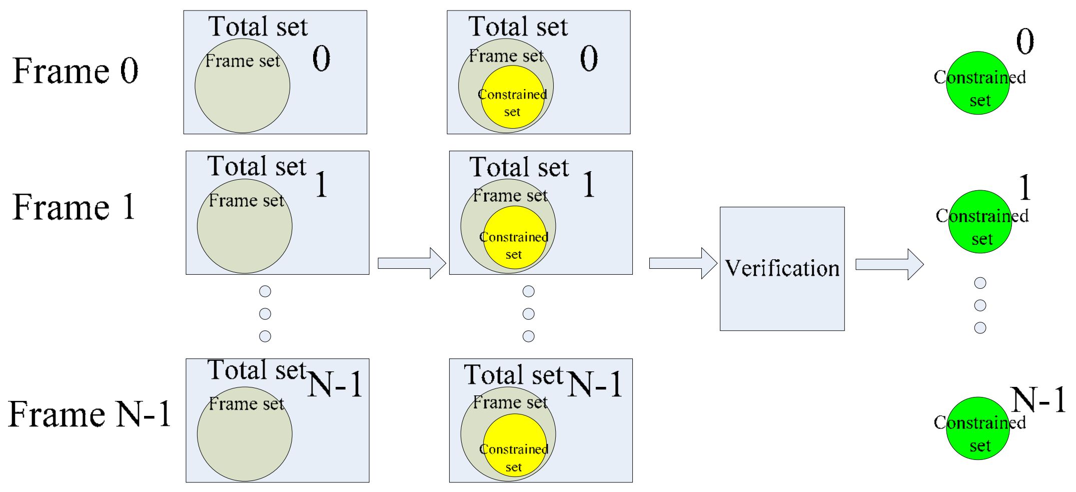


Fig. 27 Reduction phase of intra-checking. The number of EOBs generated from each frame is reduced.

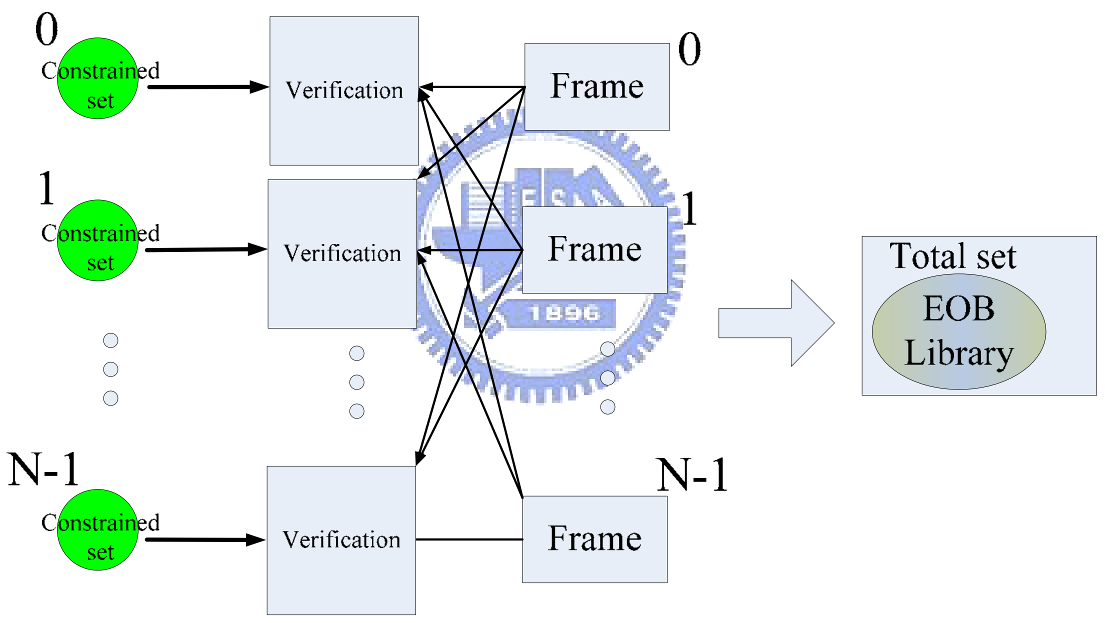


Fig. 28 Reduction phase of inter-checking. After this reduction the remained EOBs is the EOB library stored in memory.

4.4 EOB Storage Using Group-based Scheme

We can treat all EOBs stored in memory as the codewords of a virtual table. From the group-based algorithm we know that the symbol of the decoded codeword can be accessed by calculation. Therefore, the symbol for EOBs codewords is composed of 1-bit hit flag. Thus, a large amount of searching EOBs can be done in a fast process. For instance, based on the Fig. 29, if the bitstream is 0000_0000_0110_0110_0000_0000_0000_0 1011_0100…..

1. **Do group searching**

➔PCLC_mincode1(29’b0000_0000_0101_1111_0000_0000_0000_0) < bitstream_num < PCLC_mincode2(29’b0000_0000_0110_1011_0000_0000_0000_0)

➔The matching group: G1

2. **Send group information**

➔ code length = 15-bit, PCLC_mincode = 29’b0000_0000_0110_0100_0000_0000_0000_0, base_addr(6-bit) =6’b1000_00.

3. **Find the valid VLC_codeoffset, which is the code length most significant bits of the result of subtracting the PCLC_mincode from the bitstream_num**

➔Bitstream_num(29’b0000_0000_0110_0110_0000_0000_0000_0) – PCLC_mincode(29’b0000_0000_0110_0100_0000_0000_0000_0) = 29’b0000_0000_0000_0010_0000_0000_0000_0.

➔The valid VLC_codeoffset = 15’b0000_0000_0000_001= 1.

4. **Extract the VLC_codeoffset operand, which has the same word length as the symbol address**

➔VLC_codeoffset = 6’b0000_01= 1.

5. **Calculate the decoded symbol address**

➔symbol_addr = base_addr(6’b1000_00) + VLC_codeoffset(6’b0000_01) = 6’b1000_01= 33.

6. **Fetch the decoded symbol**

➔ sym_memory[33] = S33.

As a consequence, we can store flag as the S33 in the location of address = 33

The flag means if hit or miss for the EOB checking.

PCLC	Length-1	Address
---------------G0---------------		
0000_0000_0110_1111_0000_0000_0000_0	15	[40]
0000_0000_0110_1100_0000_0000_0000_0	15	[37]
0000_0000_0110_1011_0000_0000_0000_0	15	[36]
---------------G1---------------		
0000_0000_0110_1010_0000_0000_0000_0	14	[35]
0000_0000_0110_1000_0000_0000_0000_0	14	[34]
0000_0000_0110_0110_0000_0000_0000_0	14	[33]
0000_0000_0110_0100_0000_0000_0000_0	14	[32]
---------------G2---------------		
0000_0000_0101_1111_0000_0000_0000_0	15	[31]
---------------G3---------------		
0000_0000_0101_1110_0000_0000_0000_0	14	[30]
0000_0000_0101_1100_0000_0000_0000_0	14	[29]
---------------G4---------------		
0000_0000_0011_1110_0000_0000_0000_0	14	[28]
0000_0000_0011_1100_0000_0000_0000_0	14	[27]
0000_0000_0011_1010_0000_0000_0000_0	14	[26]
0000_0000_0011_1000_0000_0000_0000_0	14	[25]
0000_0000_0011_0110_0000_0000_0000_0	14	[24]
0000_0000_0011_0100_0000_0000_0000_0	14	[23]
0000_0000_0011_0000_0000_0000_0000_0	14	[21]
0000_0000_0010_1110_0000_0000_0000_0	14	[20]
0000_0000_0010_1100_0000_0000_0000_0	14	[19]

Fig. 29 A portion of groups of EOBs

4.5 Joint with Channel and VLC Source Decoder

The bock diagram is shown in Fig. 30. In this error resynchronization scheme, we assume that channel information is available. The system includes the channel side and source side. The source decoder can use the bit reliability which is the information comes from channel to determine when to check EOB. Once the bit reliability is lower than a threshold, we activate the searching EOB by the method in the last section for the following bit-stream. If the hit flag is accessed from memory, we think it as the EOB of a 4x4 block and thus we can decode the next block from the actually bit position. Besides, the method is mainly for random error type.

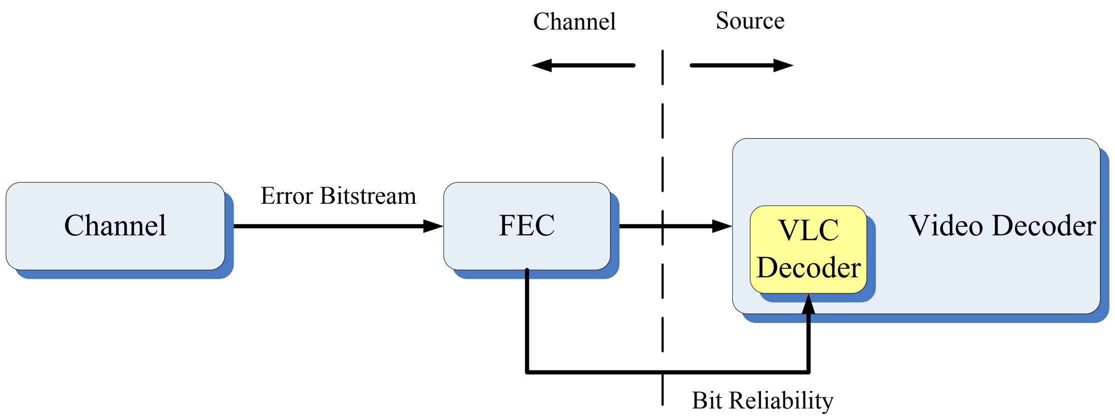


Fig. 30 Block diagram of the decoder in the wireless transmission environment.

Chapter 5

Simulation Result

From the proposed algorithm, block boundary prediction by EOBs is a probability issue and EOBs are also VLC, therefore, the length of EOBs has effect on the probability of prediction. On one hand, longer EOB codewords has lower occurrence probability. On the other hand, longer EOB codewords are removed more hardly and have more probability of being in the EOB library finally.

The probability of EOBs that meet length constraints are shown in Fig. 31 and the calculation equation (2) is as follows:

$$\text{occurred probability} = \frac{\#\text{ of EOBs(length} > \text{L)}}{\text{Total }\#\text{ of EOBs}} \tag{2}$$

All testing frames are QCIF format and the first 100 frames is the first 100 frames from akiyo sequence, 200~299 frames is the first 100 frames from foreman sequence and the last 100 frames is the last 100 frames from foreman sequence. Fig. 32 shows the probability of correctly found EOBs under three different length constraints. The simulation shows that probability is very close when length is 8 and 10 bits at least and the probability of EOBs (L > 14) is lower.

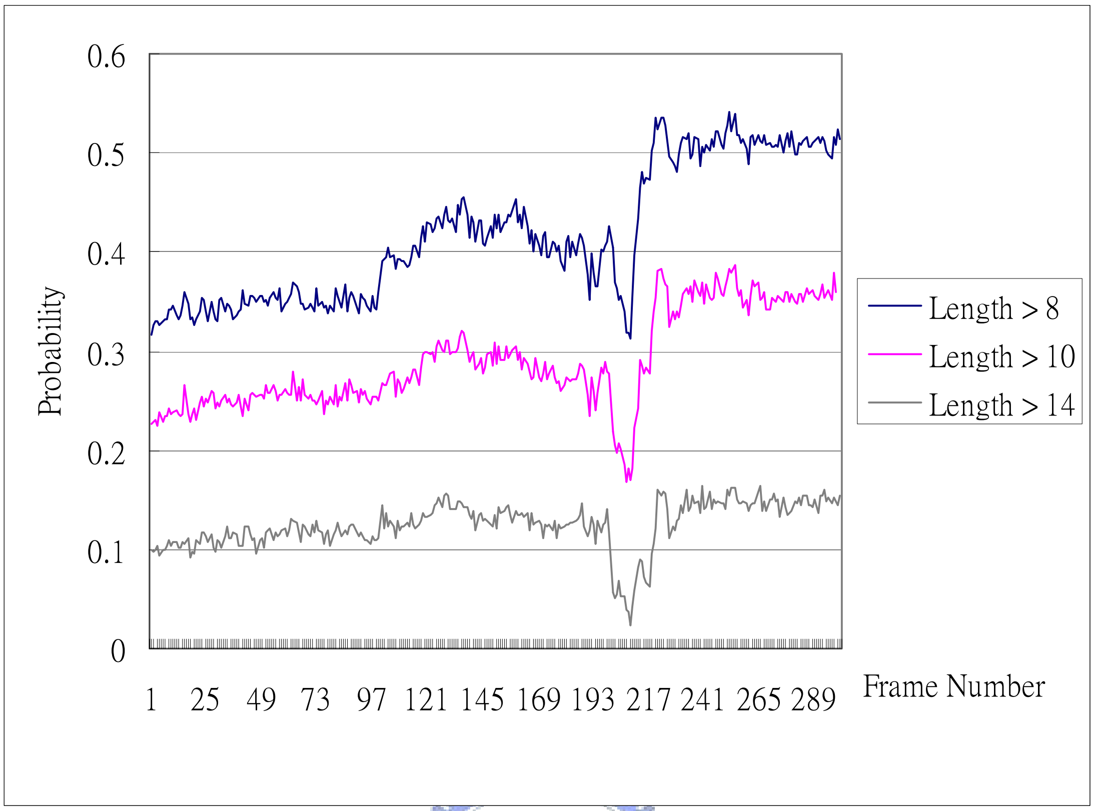


Fig. 31 Probability of EOBs with length constraints.

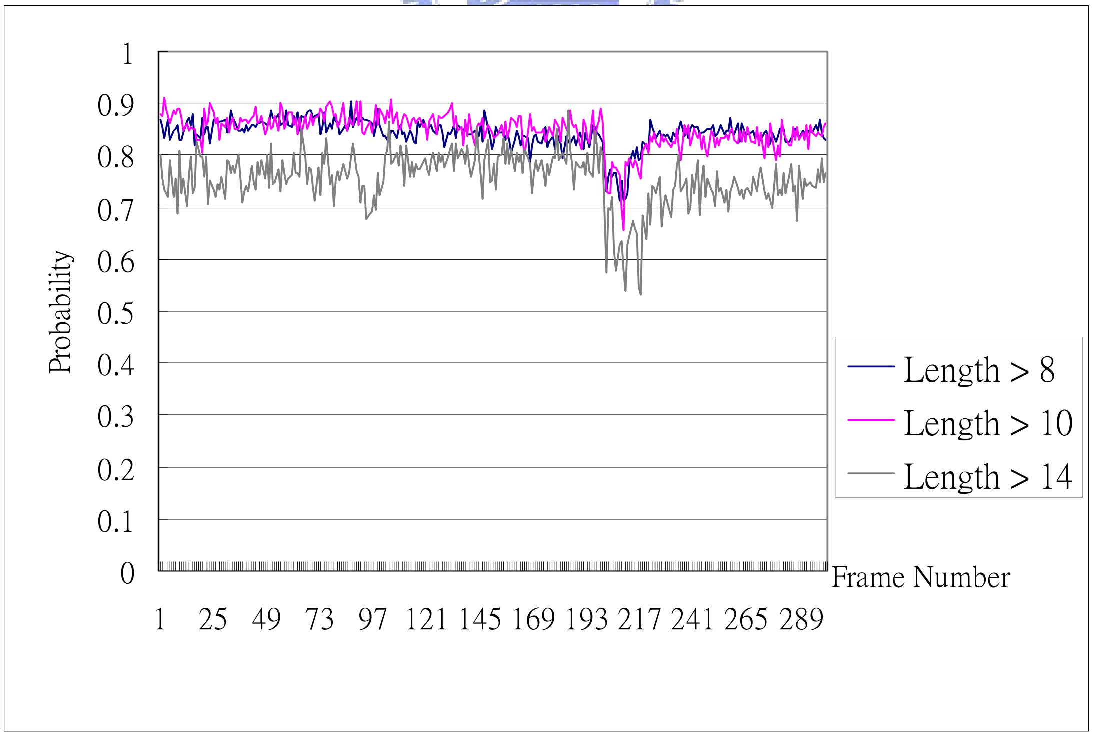


Fig. 32 Probability of EOBs that are correctly found in the bitstream.

Table 6 and Table 7 show the memory space for different sets of EOBs. Different grouping strategy makes memory space of group information and symbol quite different. If we separate EOBs into groups by position of leading one (prefix), number of groups is small while the maximum symbol address is large. In contrary, we separate EOBs into groups by prefix and length condition, number of group is large while maximum symbol address is much smaller than the previous one.

Table 6 Estimation of memory usage for different sets of EOBs
Grouping by prefix only

.	Length		
	8	10	14
number of group	17	17	17
max. symbol address	160975	282645	639418
PCLC	29	29	29
symbol memory	1024*256	2048*256	8192*128
base address	10	11	13
group information memory	17*(29+5+10)	17*(29+5+11)	17*(29+4+13)
Total	262892	525053	1049358

Table 7 Estimation of memory usage for different sets of EOBs.
Grouping by prefix and length

	Length		
	8	10	14
number of group	13323	13034	9454
max. symbol address	20760	20678	23640
PCLC (bits)	29	29	29
symbol memory(bits)	1024*32	1024*32	1024*32
base address(bits)	10	10	10
group information memory(bits)	13323*(29+5+10)	13034*(29+5+10)	9454*(29+4+10)
Total(bits)	618980	606264	439290

Chapter 6

Hardware Architecture and implementation

6.1 Overview of Hardware Architecture

Fig. 33 shows the block diagram of proposed dual-mode memory-based VLC decoder. There are mainly five components: 1) Controller, 2) Input Shift Buffer, 3) Memory-based VLC Decoder, 4) Coefficient Buffer and 5) Level Decoder. The controller assigns the control signals for each syntax element according to *nC*, maxnumcoeff and enable signal from syntax parser. The controller is implemented by a finite state machine (FSM). The memory-based VLC decoder can support CAVLC and MPEG-2 coefficient decoding. Several control signal are needed to control internal memory and these control signal are directly from chip I/O ports so that the content can be loaded into memory for different video standards. The level decoder is mainly composed by level prefix decoder, level suffix decoder and suffix length. Level prefix is composed of a leading 1 detector and level suffix is decoded by getting bits from bitstream buffer depending on level suffix size. The coefficients buffer consists of 4x4 register array and is controlled by run index and level index such that the decoded runs and levels can be put into buffer in order. The input bitstream buffer is integrated into higher hierarchical module in the H.264/MPEG2 video decoder, which is proposed in [18]. Also, the design is also integrated into the H.264/MPEG2 video decoder in the previous video decoder [18].

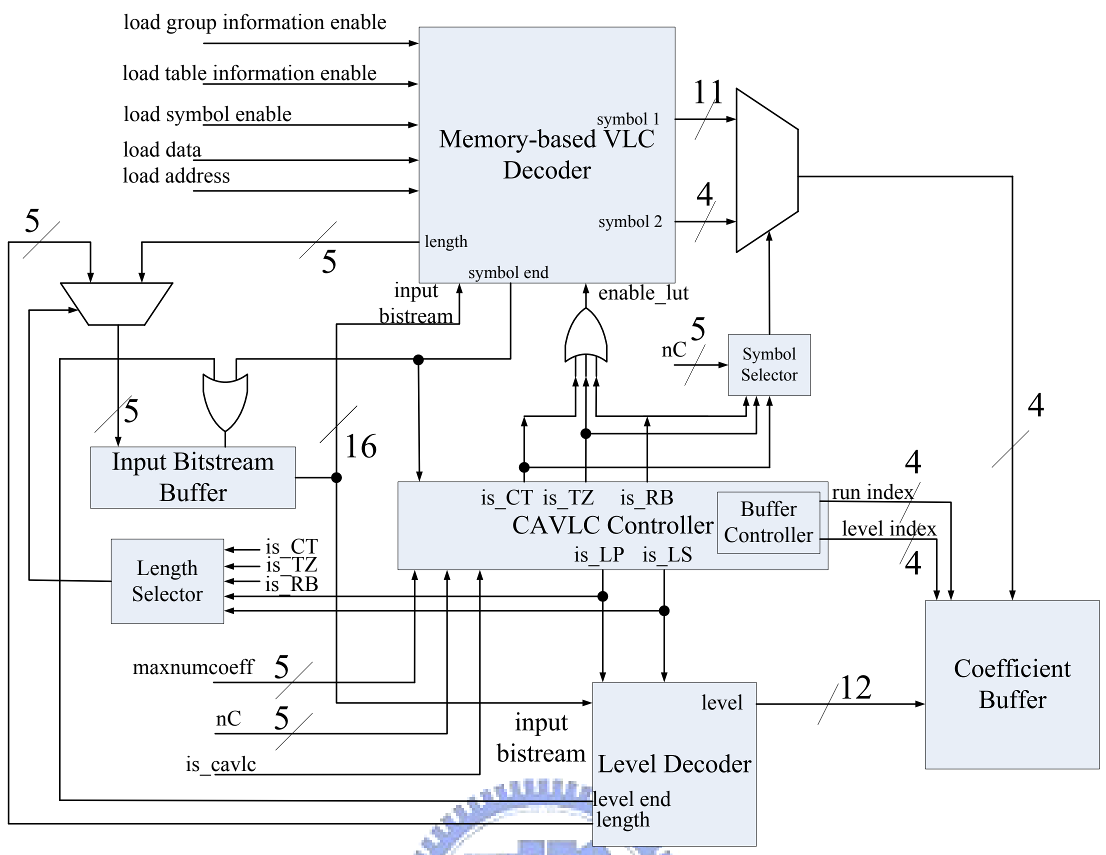


Fig. 33 Block diagram: controller, bitstream buffer, Memory-based VCL Decoder, Level decoder , coefficient buffer

6.2 *Memory-based VLC Decoder*

From the modified MTM algorithm, the VLC decoder has two categories for storing group information and table information. They are selected by table index. Each group information is extracted to compute offset. The final offset is determined by the enable signal from group detectors. The tri-state buffer can viewed as the gate for each group. One group has three data items: offset, base address and (length-1). For each decoding time, only one set of data among all groups is passed by the tri-state buffer while others are floated. After the set of data are passed, (length -1) is returned to bitstream shift buffer in the other module and base address is added to offset to compute symbol address. Finally, symbol memory is accessed to output decoded symbol.

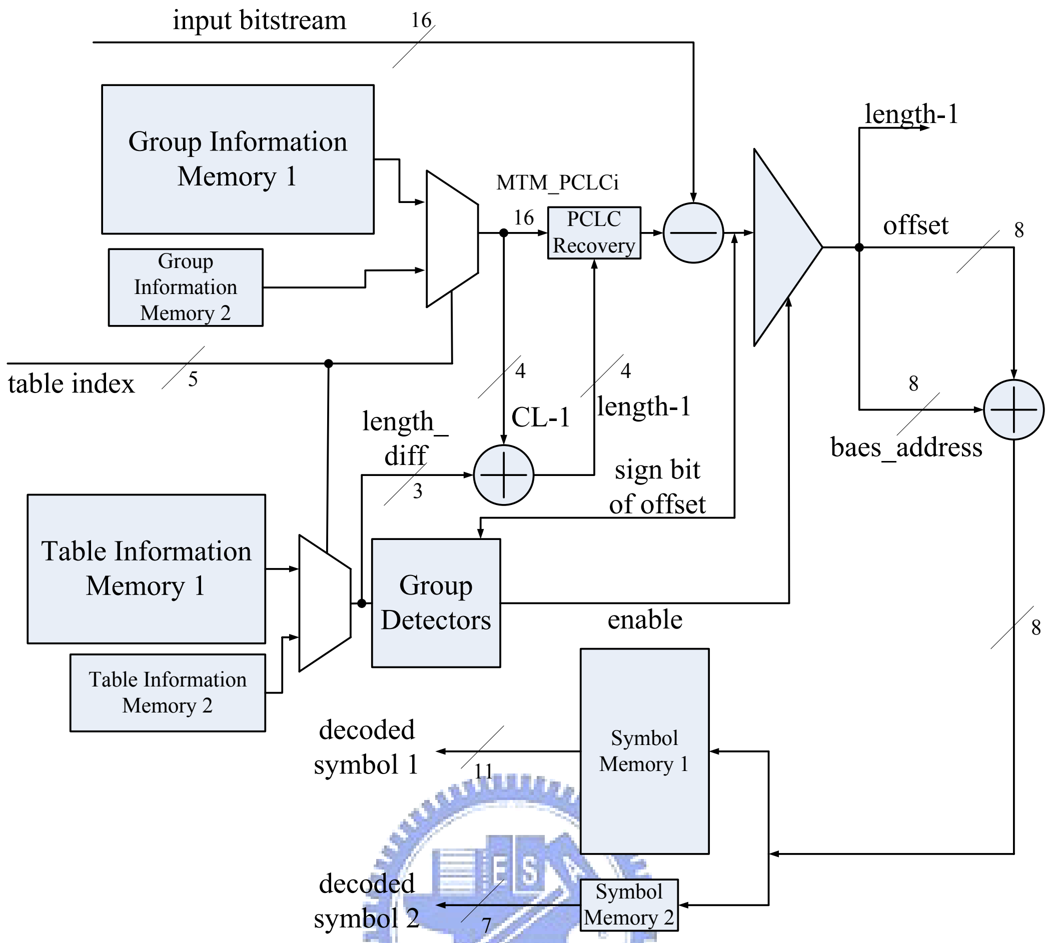


Fig. 34 Total Block Diagram of memory-based VLC decoder

The stage partition of memory-based VLC decoder is briefly shown in Fig. 35 and cycle time of important signals are also shown. The register file generated by memory compiler needs one cycle to read data thus the access of group information and table information memory is viewed as the first stage. After the first stage, the symbol address is known thus the access of symbol memory is the second stage.

The group information and table information memory can be accessed when the table used to decode is known. The example of cycle time is shown as in Fig. 35. The first cycle is reading of memory and address computation and the decoded symbol is outputted in the second cycle. The *in_valid* signal represents the input bitstream and *table idx* is valid. The symbol is valid until the second cycle. In the third cycle, next valid data is sent after the controller received the *Symbol_valid* signal.

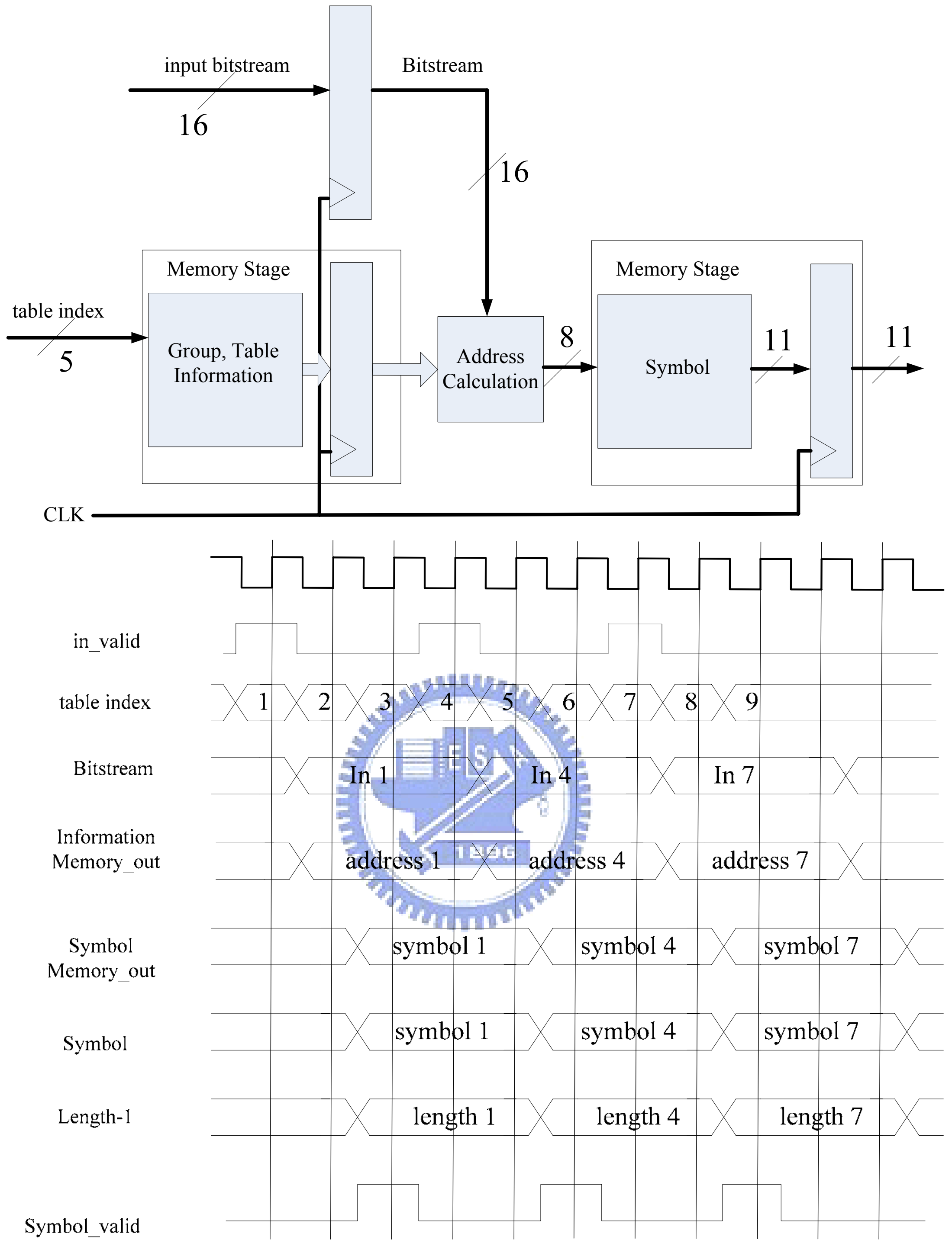


Fig. 35 Stage and timing diagram

6.3 *Improvement of throughput*

According to the VLC property, shorter codewords occurred more frequently than long codewords. According to observation of CAVLC decoding, decoding of

Run_Before consumes many cycles for a block especially in low-QP video and most codewords of *Run_Before* tables are short. Therefore, we assume that the short codeword stores information of *Run_Before* such as length of codewords, symbols and table index.

Fig. 36 shows the block diagrams of memory-based VLC decoder with cache. Only some important signals are annotated for simplicity. Again, the controller send the *read_en* and *write_en* signal to activate reading or writing of the cache. If cache hit occurs, the memory-based VLC decoder is disabled and the decoded symbol and length is from the cache. If cache missed, conventional VLC decoder is enabled and the symbol and length are stored in the cache. Looking up cache can be done in one cycle while decoding of memory-based VLC decoder need three cycles. As a result, adding cache is a good method to improve the throughput of the whole decoding.

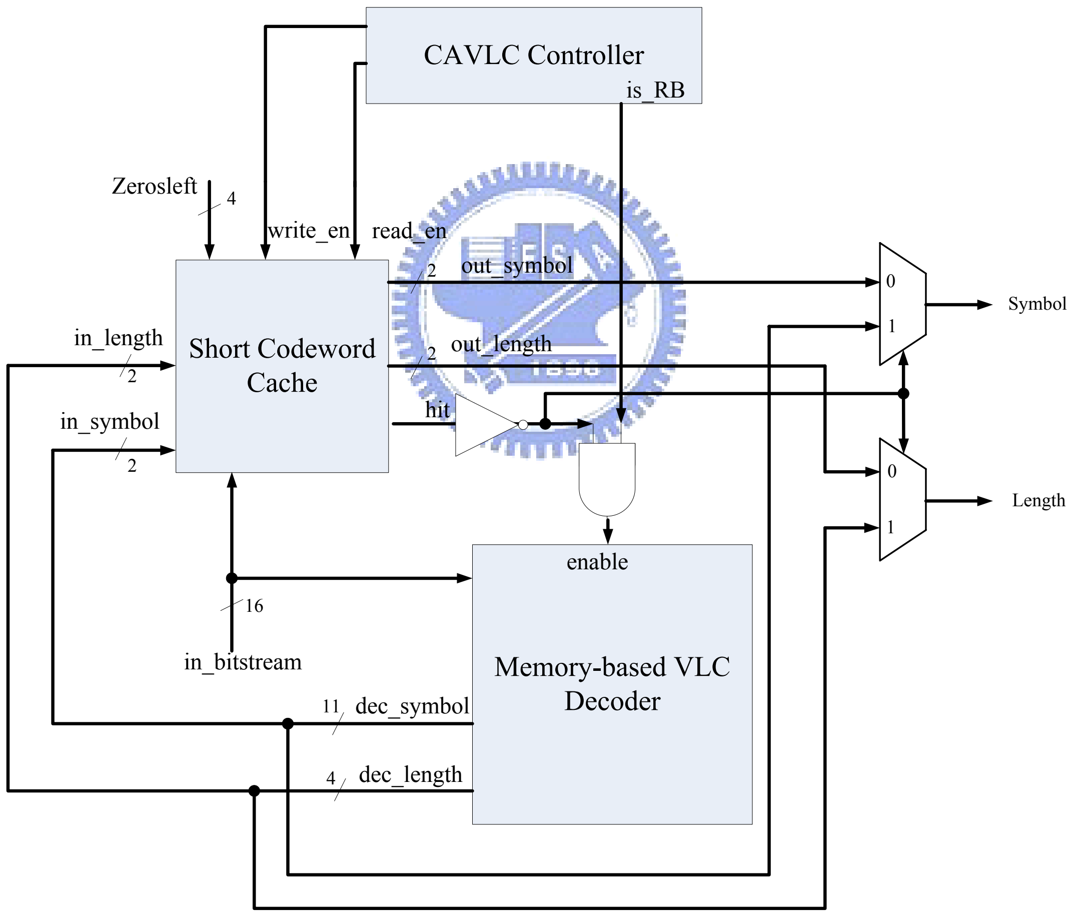


Fig. 36 Small Cache for Run (short codewords)

In addition to adding cache, one method is also proposed to improve throughput further. In Fig. 35, the *table index* is known after the FSM is in *Coeff_Token* and *Total_Zeros* states. From the conditions of table transition, the next table used to decode can be known earlier than these two states. As a consequence, group information and table information can be accessed once the condition is known and the conventional three-cycle stage is reduced to two-cycle stage as shown in Fig. 37. In this figure, current *Information Memory_out* is known before *in_valid* is high or in the same cycle as *in_valid* is high. Therefore, the pre-fetching mechanism can achieve one symbol decoding every two cycles. This method can further improve throughput compared to the original design in this thesis.

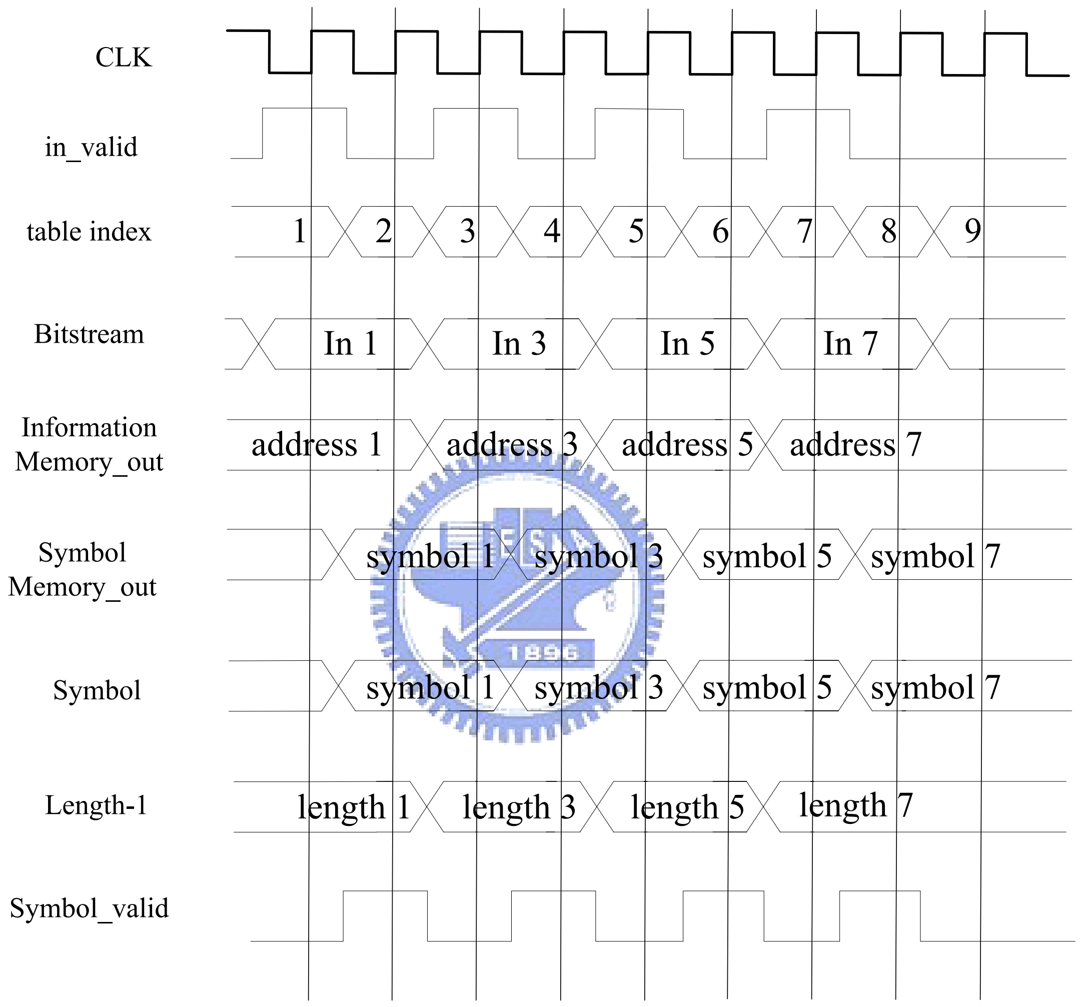


Fig. 37 Cycle time of the pre-fetching method

6.4 Implementation Result

In hardware implementation, the group information for MTM_PCLC and (MTM_CL-1) are stored in registers because they are used in the same time. Therefore, this leads to more gate count and less memory space. In summary, the memory space is shown in Table 8.

Table 8 Memory space for VLC decoder

	Storage content	Number of bits	Physical space
Table information memory	Valid bit and CL_diff	16 x 23 x 4+ 14 x 5 x 3	16 x 23 x 4+ 16 x 5 x 3
Group information memory	Base address	16 x 23 x 8 + 14 x 5 x 8	16 x 23 x 8 + 16 x 5 x 8
Symbol memory	C.T and TZ/RB Run-Level	256 x 11	256 x 11
	C.T for CAVLC	16 x 7	16 x 7
Total	8114 bits		8224 bits

The gate count and power consumption of the designs are shown in Table 9. From this table, the gate count of these design are similar. However, the power consumption of memory-based VLC decoder with cache only is lower than that of memory-based VLC decoder without cache. This shows that the cache storing frequent codewords without replacement can achieve power reduction and improve throughput. The throughput of foreman and mobile sequences under different QP are shown in Fig. 38 and Fig. 39, respectively. From these two figures, the proposed design can achieve HD 720p even for very low QPs under 100MHz. The design can also meet requirement of HD1080p when operation frequency is 200MHz as shown in Fig. 40 and Fig. 41.

Table 9 Gate count and power of different designs.

	Gate Count	Power Consumption(mW)
Memory-based VLC decoder	15.4 k	1.132
Memory-based VLC decoder + cache	17.2 k	1.078
Memory-based VLC decoder + pre-fetch	17.2k	1.017

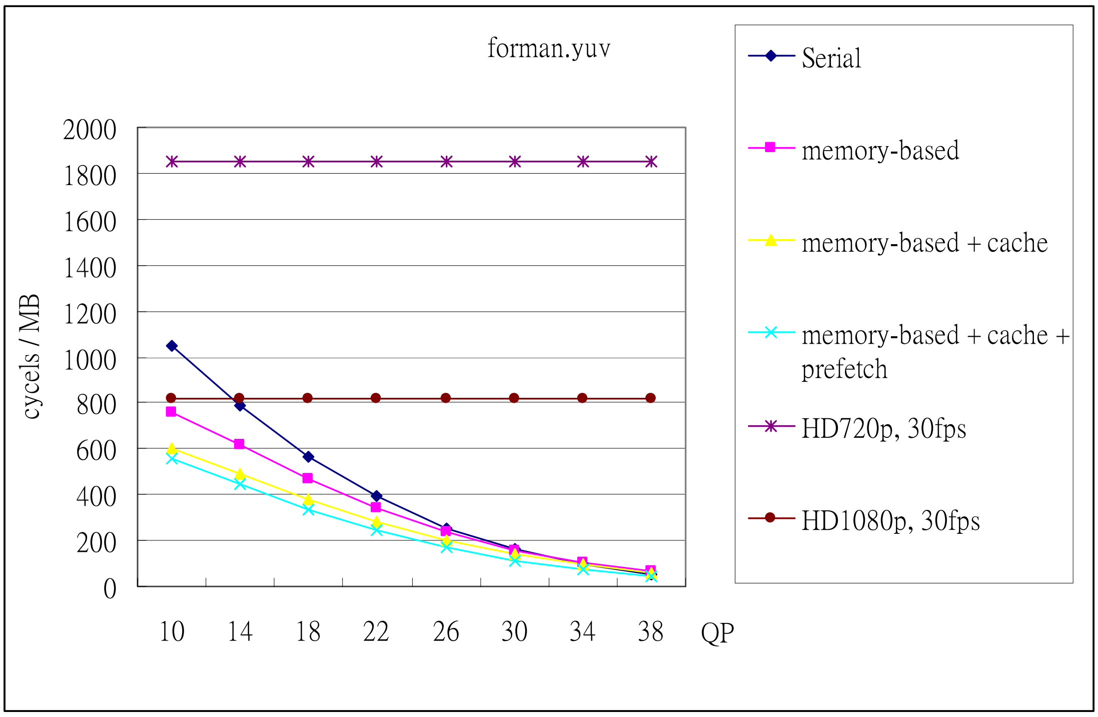


Fig. 38 Throughput of decoding under Foreman sequence under 100MHz.

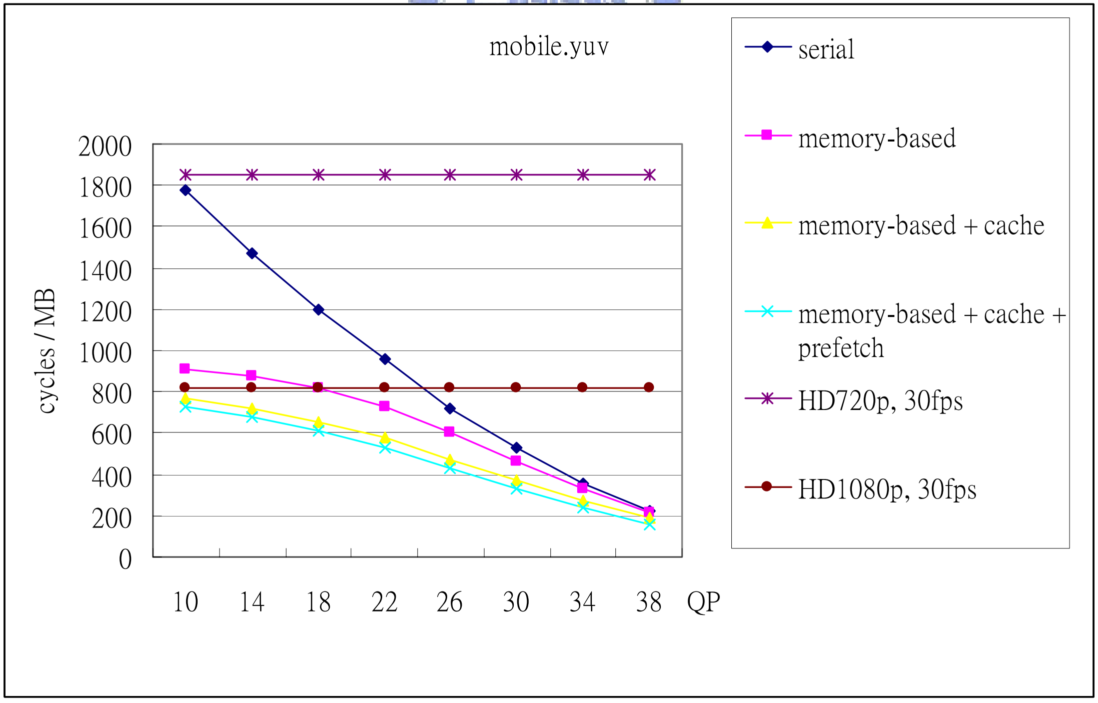


Fig. 39 Throughput of decoding under Mobile sequence under 100MHz.

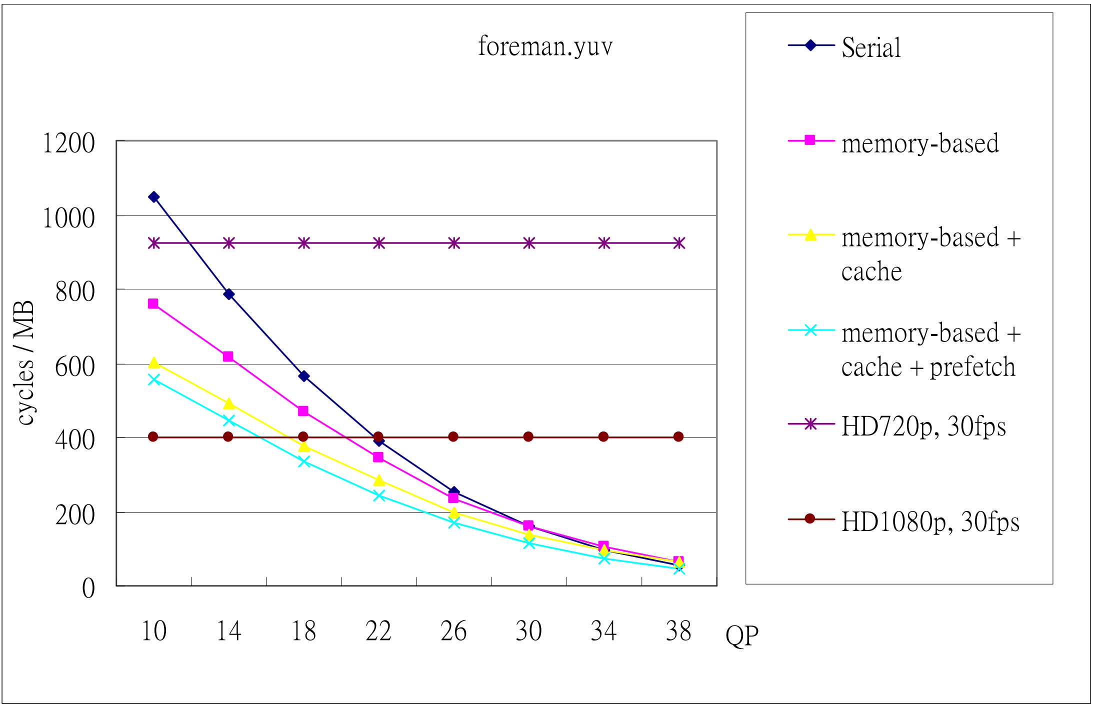


Fig. 40 Throughput of decoding for Foreman sequence under 200MHz.

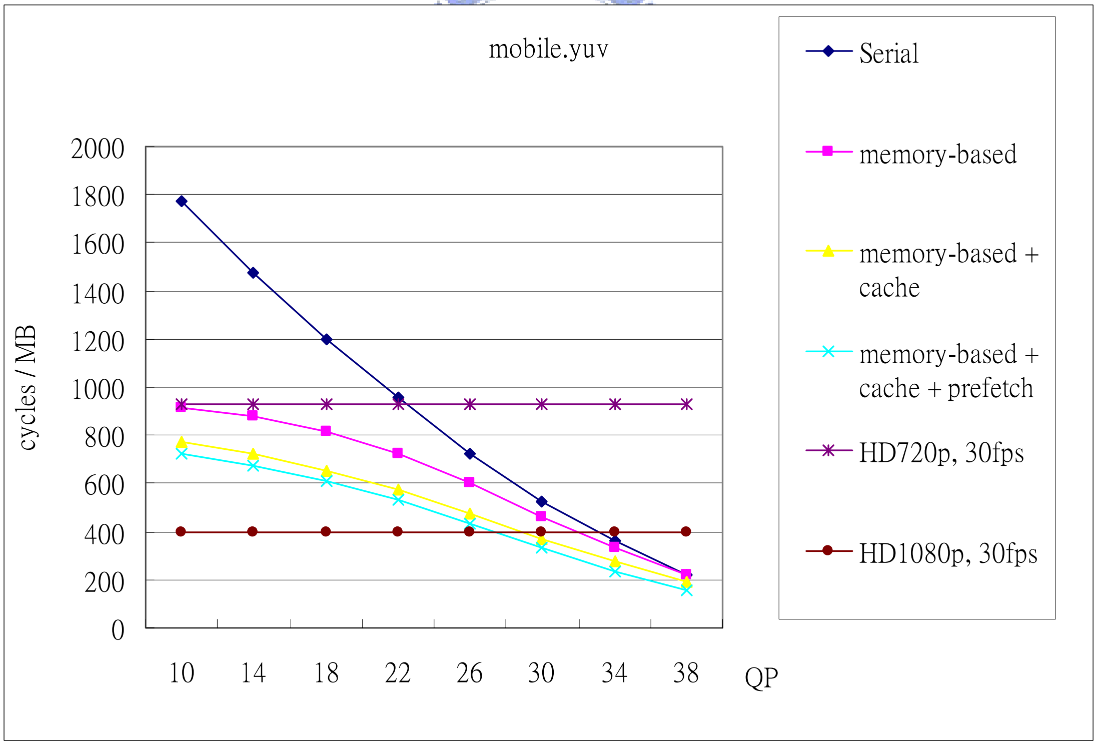
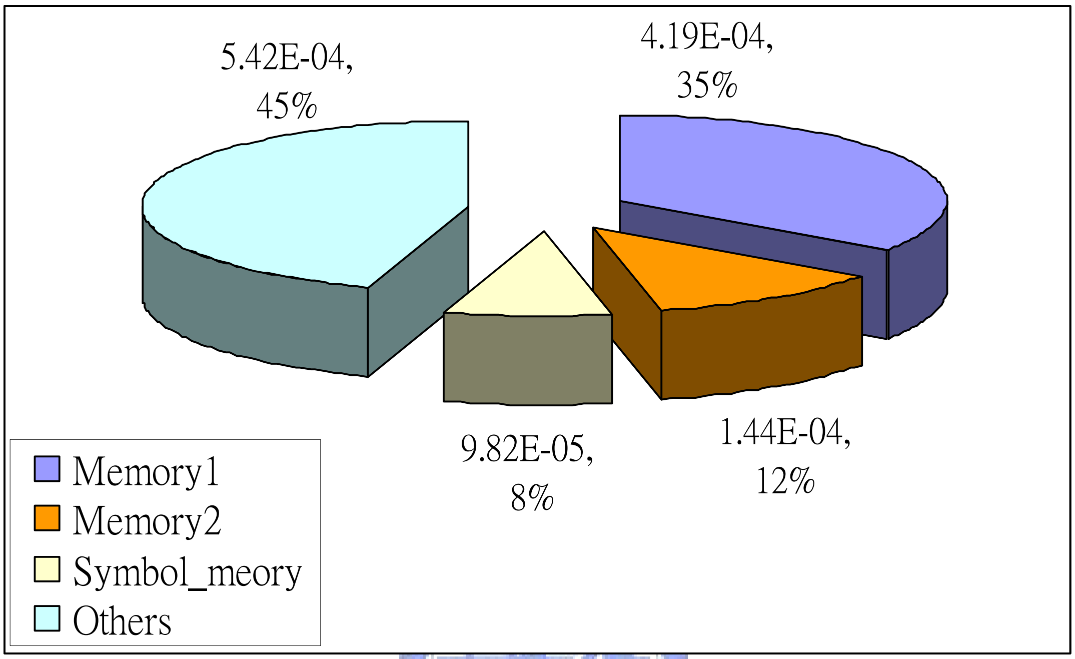
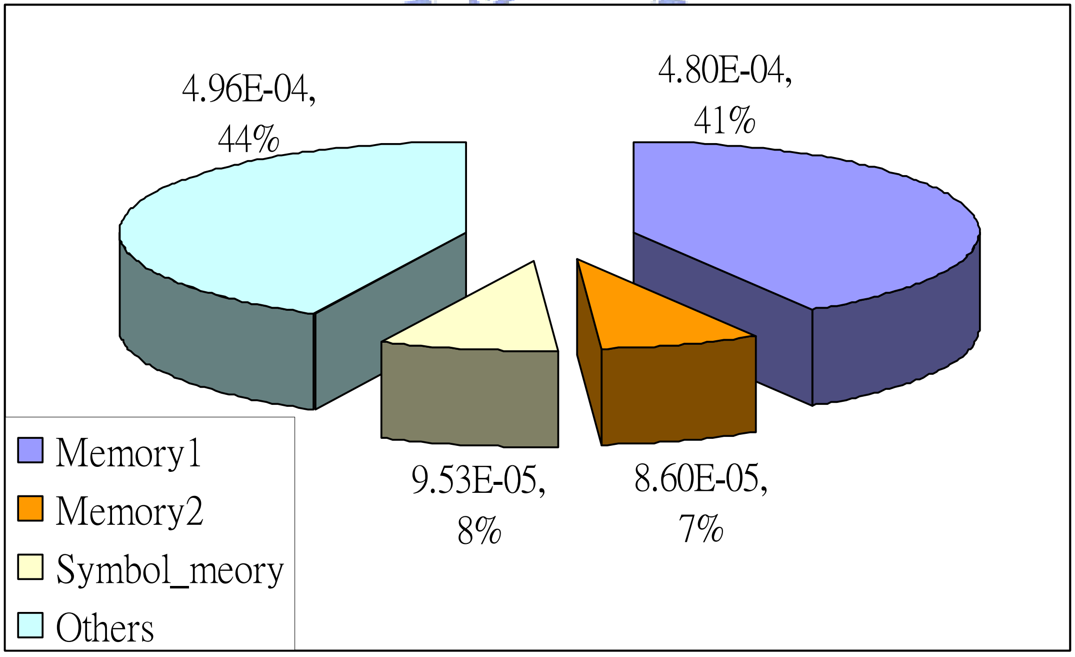


Fig. 41 Throughput of decoding for Mobile sequence under 200MHz.

Fig. 42 and Fig. 43 show power distribution of each main module for test pattern of mobile and akiyo, respectively. The operating frequency is 100MHz. Memory1 stored the base addresses and table information for large tables and Memory2 stored the base addresses and table information for small tables.

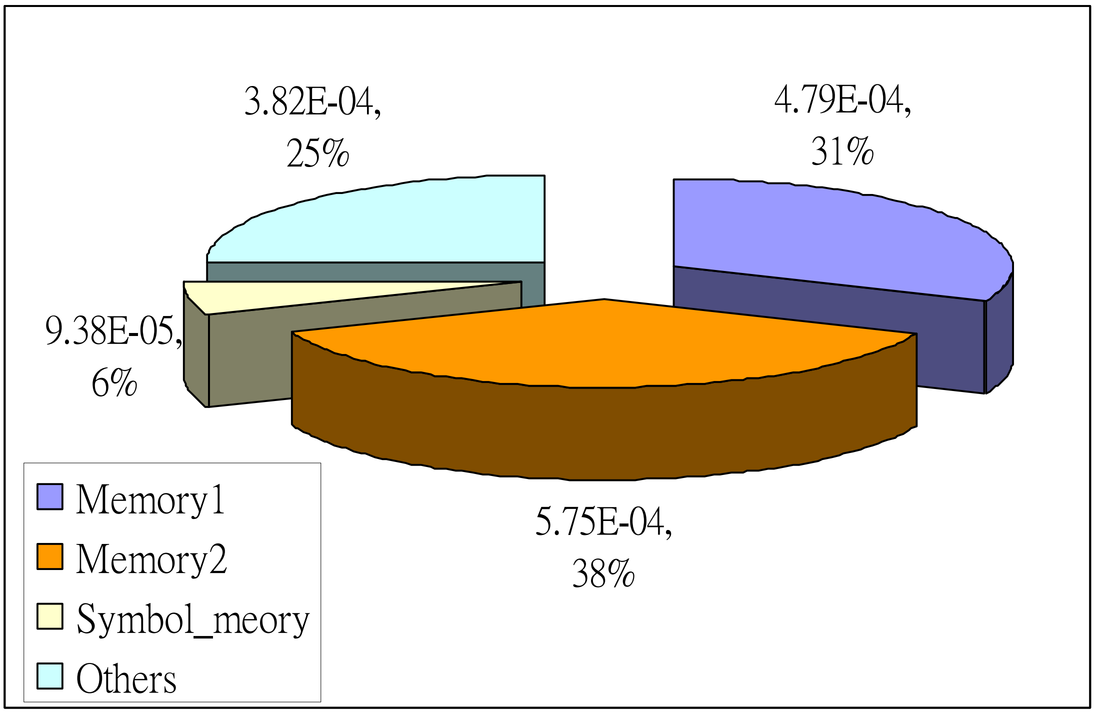


(a) QP = 16 for mobile

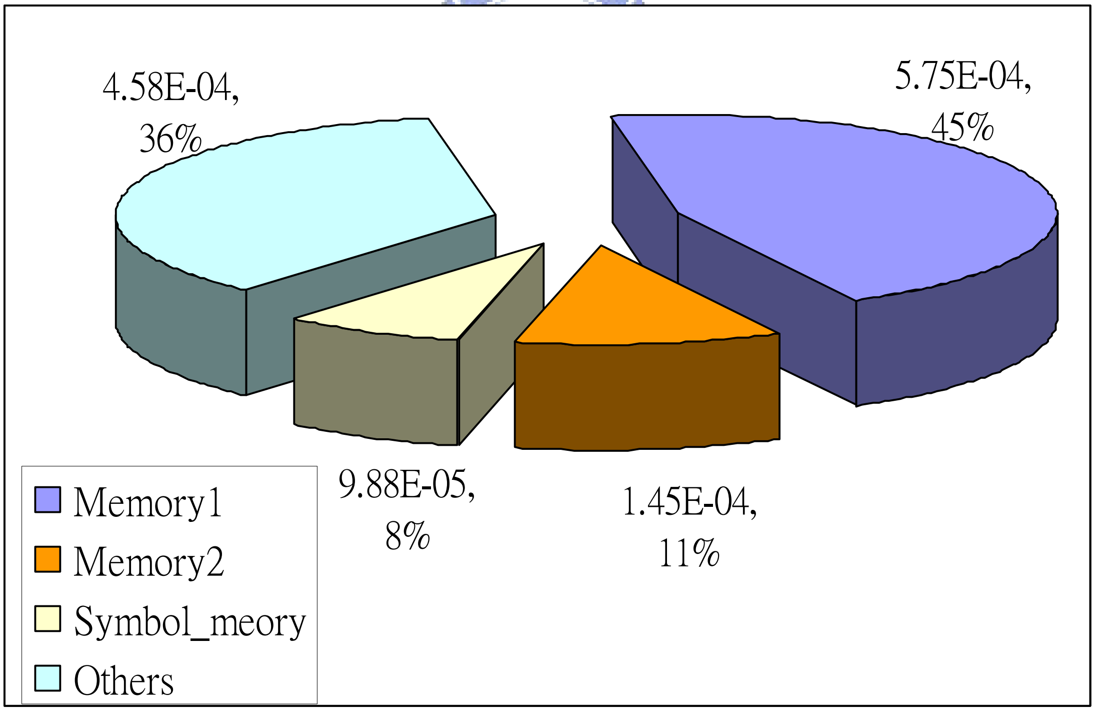


(b) QP = 34 for mobile

Fig. 42 Power chart of different module in mobile test frame for (a) QP=16, (b) QP=34



(a) QP = 16 for akiyo



(b) QP = 34 for akiyo

Fig. 43 Power chart of different module in akiyo test frame for (a) QP=16, (b) QP=34

Table 10 shows the comparison of the proposed design. We can see that the proposed design support two different entropy decoding, i.e. MPEG2 and H.264. Besides, the proposed design has error resilience feature for application of wireless video transmission. These two features are quite different from the other designs.

Table 10 Comparison of other designs and proposed design

	[1] NCCU TCSVT'06	[2] NCU ASSCC'07	[3] NCKU Trans. on Multimedia'08	Proposed
Process	0.18um	0.18um	0.18um	0.09um
Technique	Hardwired	Hardwired	Hardwired	**Memory-based**
Features	Parallel LUT	Multi-symbol for level	Modified level detector	**Error resilience**
Max Frequency	N/A	102MHz	213MHz	200MHz
Gate Count	13.1K	13.2K	6.7K	17.2K
Memory (bits)	N/A	N/A	N/A	8114 bits
Target Format	HD1080, 30fps	HD1080, 30fps	HD1080, 30fps	HD1080,30fps
Multi-mode	MPEG-1/2/4	H.264 (CAVLC)	H.264 (CAVLC)	**MPEG-2, H.264**

Chapter 7

Conclusion and Future Work

As we know, entropy decoder of MPEG-2 and H.264/AVC are very different from each other, such as decoding flow, symbol format and table transition. A memory-based VLC decoder which support dual-mode vide format (H.264/AVC) and MPEG-2 with error robustness is proposed in the thesis. The thesis focuses on improvement of memory efficiency for conventional VLC decoder first. Although MPEG-2 part is not yet exactly implemented in the decoder, only little overhead like multiplexers and additional coefficients buffers are needed when MPEG-2 is required because the memory utilization and size are considered in this design. For CAVLC decoding, throughput is limited by dependency between syntax elements, hence, pipeline stage is not adequate for this decoding. However, the decoding of MPEG2 can be pipelined because there symbols are independent. The VLC decoder is synthesized under 100MHz and can be promised to support HD720p even under low QPs. The design can also meet requirement of HD1080p when operation frequency is 200MHz.

In addition, a novel error resynchronization is proposed in the thesis. This method can be combined with conventional memory-based VLC decoding without extra bandwidth overhead. In this scheme, the EOBs are constructed with length constraint. The flow of EOB construction is proposed to reduce off-line simulation time and the analysis of the EOB probability is also presented. After EOB library is set, group-based decoding of VLC is applied to determine if EOB are found. Compared to trellis-based JSCD or soft-input VLC decoding, this method makes hardware implementation can be done and much less complexity.

There is some works can be developed further for this design. First, integrate MPEG-2 decoding circuit into current design to achieve fully scalability. Second, some other power reduction schemes are necessary for mobile applications. Third, from H.264 CAVLC Run_Before tables, we can see that many codewords is composed of 1's string, thus, we can use this characteristic to improve boundary prediction probability. Beside, this can also help to reduce size of EOB library thus the memory size for error resynchronization information. Finally, the design is appended channel model to build the whole wireless transmission system.

References

[1] B. J. Shieh, Y. S. Lee, and C. Y. Lee, "A New Approach of Group-based VLC Codec System with Full Table Programmable," IEEE Trans. Circuits and System for Video Technology, Vol. 11, no. 2, pp.210-221, Feb. 2001.

[2] B. J. Shieh, "Area Efficient and High Throughput Memory-Based VLC Codec Designs", Ph. D. dissertation, 2001.

[3] C. H. Liu, B. J. Shieh, and C.Y. Lee, "A Low-Power Group-Based VLD Design," Proceedings of the 2004 International Symposium on Circuits and Systems, vol.2, pp.II 337- II 340, May 2004.

[4] Y. H. Moon, "A New Coeff_Token Decoding Method with Efficient Memory Access in H.264/AVC Video Coding Standard," IEEE Trans. Circuit and System for Video Technology, Vol. 17, no. 6, pp.729-736, June 2007.

[5] Y. H. Moon, G. Y. Kim, and J. H. Kim, " An Efficient Decoding of CAVLC in H.264/AVC Video Coding Standard," IEEE Trans. Consumer Electron. , Vol. 51, no. 3, pp. 933–938, Aug. 2005.

[6] Chen, Yanling; Cao, Xixin; Peng, Xiaoming; Peng, Chungan; Yu, Dunshan; Zhang, Xing; "A Memory-Efficient CAVLC Decoding Scheme for H.264/AVC" 10th International Conference Advanced Communication Technology, Vol. 2, 17-20 Page(s):1135 – 1138, Feb. 2008.

[7] H. C. Chang, C. C. Lin, and J. I. Guo, " A Novel Low-cost High-performance VLSI Architecture for MPEG-4 AVC/H.264 CAVLC Decoding," in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), pp. 6110–6113, 2005.

[8] C. D. Chien, K. P. Lu, Y. M. Chen, J. I. Guo, Y. S. Chu and C. L. Su, "An area-efficient variable length decoder IP core design for MPEG-1/2/4 video coding applications," IEEE Trans. Circuits and Systems for Video Technology, Vol. 16, issue 9, pp. 1172-1178, Sep. 2006.

[9] Tsung-Han Tsai and De-Lung Fang ,"A Novel Design of CAVLC Decoder with Low Power Consideration, " IEEEE,ASSCC, Nov, 2007.

[10] Heng-Yao Lin, Ying-Hong Lu, Bin-Da Liu, Jar-Ferr Yang, “A Highly Efficient VLSI Architecture for H.264/AVC CAVLC Decoder, ” IEEE Transaction on Multimedia, Vol. 10, No. 1, Jan, 2008.

[11] Superiori, L., Nemethova, O. and Rupp, M. “Performance of a H.264/AVC Error Detection Algorithm Based on Syntax Analysis, ”the Fourth International Conference on Advances in Mobile Computing and Multimedia (2006).

[12] Wu, Guan-Lin; Chien, Shao-Yi, “Spatial-Temporal Error Detection Scheme for Video Transmission over Noisy Channels, ” 9th IEEE International Symposium on Multimedia, Page(s):78 - 85, Dec. 2007.

[13] Q. Chen and K. P. Subbalakshmi, “Trellis decoding for MPEG-4 streams over wireless channels,” in Proc. SPIE Electronic Imaging: Image and Video Communications and Processing, pp. 810–819, Jan. 2003.

[14] Qingyu Chen and K. P. Subbalakshmi, “Joint Source-Channel Decoding for MPEG-4 Video Transmissioan Over Wireless Channels, ” IEEE Journal on Selected Areas in Communications, Vol. 21, No. 10, Dec 2003.

[15] Qingyu Chen; Subbalakshmi, K.P, “An Integrated Joint Source-Channel Decoder for MPEG-4 Coded Video, “ Vehicular Technology Conference, Vol. 1, Page(s):347 - 351 , 6-9 Oct. 2003.

[16] Yue Wang; Songyu Yu, “Joint Source-Channel Decoding for H.264 Coded Video Stream,” Consumer Electronics, IEEE Transactions, Vol. 51, Issue 4, Page(s):1273 – 1276, Nov. 2005.

[17] O. Nemethova, J. Canadas Rodriguez, M. Rupp, “Improved Detection for H.264 Encoded Video Sequences over Mobile Networks”, Proc. of Eight International Symposium on Communications Theory and Applications, Ambleside, Lake District, UK, July 2005.

[18] Tsu-Ming Liu, Ting-An Lin, Sheng-Zen Wang, and Chen-Yi Lee,“ A 125-μW, Fully Scalable MPEG-2 and H.264/AVC Video Decoder for Mobile Applications”, to appear in IEEE Journal of Solid-State Circuits, Jan. 2007. (EI/SCI, NSC94-2215-E-009-045).

簡　歷

姓名：	李韋磬
出生地：	台灣，台北市
出生日期：	民國七十三年五月七日

學歷：

- 2006年9月~2008年7月　國立交通大學電子研究所系統組碩士班
- 2002年9月~2006年6月　國立交通大學電子工程學系
- 1999年9月~2002年6月　台北市立大同高中