國立交通大學

電機學院 IC 設計產業研發碩士班

碩　士　論　文

以節省記憶體為基礎之可程式化可變長度解碼器設計

A Memory-Efficient VLC Decoder Design

學生 ： 孫紹銘

指導教授 ： 李鎮宜 教授

中華民國九十六年一月

以節省記憶體為基礎之可程式化可變長度解碼器設計

A Memory-Efficient VLC Decoder Design

研　究　生：孫紹銘　　　　　Student：Shao-Ming Sun

指導教授：李鎮宜　　　　　Advisor：Dr. Chen-Yi Lee

國　立　交　通　大　學
電機學院 IC 設計產業研發碩士班
碩　士　論　文

A Thesis
Submitted to College of Electrical and Computer Engineering
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master
in

Industrial Technology R & D Master Program on
IC Design

January 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年一月

以節省記憶體為基礎之可程式化可變長度解碼器設計

學生：孫紹銘　　　　　　　　　　　　　　指導教授：李鎮宜 教授

國立交通大學

電國立交通大學電機學院產業研發碩士班

摘要

H.264/AVC 是最新的視訊壓縮標準，與 MPEG-2，H.261 及 H.263 相比 H.264 提供了更好的壓縮效率與壓縮品質。而在這些視訊編碼中最重要的可變長度編碼是一種無失真的高效率編碼，其原理是給比較常發生的資訊較短的編碼長度，反之就給較長的編碼長度，因此在長度不定的狀況下要相對應到固定長度之記憶體位址勢必要花費許多不必要的記憶體空間，以及現今多種視訊標準中都採用了霍夫曼可變長度編碼，其中若要符合多種視訊標準，其多個霍夫曼編碼表將是浪費記憶的瓶頸。

從系統設計的角度，這篇論文提出了用記憶體程式化的方法來符合 MPEG-2 與 H.264 之雙模的視訊標準。在演算法方面，我們從霍夫曼編碼方法中著手，採用了霍夫曼編碼中自已本身就有分群組的資訊，來分每一張霍夫曼表的群組，以降低所需記憶體的需求量，再採用在霍夫曼樹叢中反向提供定址位址的概念來把浪費的符號記憶

體需求量最小化。在硬體架構設計方面，為了符合多種視訊標準與使用單一 SRAM 記憶體來降低多張霍夫曼表要分多塊記憶體的額外負擔，我們採用了在記憶體位址中分群組之方法來達成僅使用單一 SRAM 記憶體即可代表多張霍夫曼表的方法。

最後本論文利用 C++語言証實了此演算法可使 MPEG-2 與 H.264 所有的可變長度編碼其所有的符號記憶體可達到 90.22%的平均利用率，而且利用了 UMC 0.18 製程合成 MPEG-2 與 H.264 雙模的可變長度解碼器在操作頻率 200MHz 下，邏輯閘與記憶體總數 31.12K，不僅可以滿足大部份的應用需求，而且此演算法與主要的硬體架構也可以應用於未來的可變長度解碼器系統之中。

A Memory-Efficient VLC Decoder Design

Student: Shao-Ming Sun Advisor: Dr. Chen-Yi Lee

Industrial Technology R & D Master Program of
Electrical and Computer Engineering College
National Chiao Tung University

ABSTRACT

H.264/AVC is the newest video coding standard. Compared to MPEG-2, H.261 and H.263, it provides more efficient compression ratio and quality. However, one of the most important variable length coding is the high efficient lossless coding. The principle is to assign high frequency information to shorter coding length, and vice versa. Therefore, in the situation of uncertain length, it must waste more unnecessary memory space for assigning to fixed length memory address. Furthermore, in the existing various video coding standards, they all adopt Huffman variable length code. It will be a bottleneck in memory consuming to conform to various video standards.

In the system point of view, this thesis provides a programmable method to conform MPEG-2 and H.264 dual mode video standards. In the aspect of algorithm, we undertake the method of Huffman coding and adopt the grouping information that naturally exists in its Huffman coding. It groups every Huffman coding tables and reduces the memory requirement. Moreover, a reversed addressing concept in the Huffman cluster is adopted to minimize the wasted symbol memory. Furthermore, in the aspect of architecture design, a single SRAM memory is used to reduce the overhead of memories. A memory address grouping method is adopted to achieve that using single memory represents many Huffman tables.

Finally, this thesis proves that the proposed algorithm can achieve 90.22% symbol memory utilization of MPEG-2 and H.264 via C++ tools. Moreover, the MPEG-2 and H.264 dual mode design is synthesized using UMC 0.18 process under operation frequency of 200 MHz. The logic and memory gate counts are 31.12K. The algorithm and main architecture not only satisfies the most of requirements and can be adopted for advanced applications.

誌　　　謝

二年的時間過得非常快，好似昨天才剛入學。但是隨著此論文漸漸成形，感覺我所學到的東西與獲得的經驗，與昨天的我相比卻是滿載而歸，而我要感謝的人真的是太多了。

首先要感謝的是 李鎮宜 教授。在這兩年中能夠成為他的學生，真的是一件非常辛運的事情，在老師不厭其煩地在研究上給予建議與指導，除了獲得真正的研究方法與知識外，還在老身上學到一個真正的好老師有的特質：謙虛、和諧、待人和藹。這是除了獲得研究方面知識外的最大收獲了，對我待人處事的人生觀又多了一個好的範例。我想，這也是 SI2 研究群日漸壯大的主要原因之一。當一個研究生能進對實驗室，做對的研究，更重要的是跟對的老師，應該沒有什麼比這還幸運了。

不可或缺的，要感謝我的家人，爸爸，媽媽，哥哥皆能體諒且支持我再多花兩年來拿 IC 設計方面的學位，若沒有你們的支持，相信這兩年也不可能那麼順利完成學業。尤其最感謝媽媽的辛勞，不管任何事總是以我為優先，我想我這輩子再怎麼報答也報答不完的。

接著要感謝的是 SI2 研究群的大家，能進來 SI2 的同學都是非常優秀的學生，因此駑鈍的我有了許多的學習榜樣，從大家身上學到許許多多的方法與技巧以及努力的態度，跟大家切磋了兩年後，一塊不起眼的璞玉也能漸漸地有他的光澤，以至這兩年真的是覺得獲益良多。

再感謝由 mingle 帶領的 SI2 的多媒體研究群，由老師提供充足的研究軟硬體，與 mingle 提供充足的研究材料，以及每次 meeting 時大家充分地討論，在多媒體相關的領域知識才有更進一步的成長，研究也才能有突破、創新。

最後，也是同等重要的，要感謝所有 SI2 研究群的學長姊、同學、學弟妹，不時提供他們的經驗，使我寫些論文時順利不少。以上如有被我粗心遺漏的人，希望你們

見諒，然而此論文是在我有限的知識與時間下完成的，以青澀的步伐所走出的第一步，所以難免有不週之處，希望有閱讀到此論文的讀者可以給予批評指教，這將是我進步的源動力。

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Motivation

In recent years, various video standards appear in the world, such as MPEG-1/2/4, H.261, H.263, H.264 etc. Therefore, the integration is more and more important to conform to various standards. It is fortunate that entropy coding is always used in the various different standards. Furthermore, the Huffman coding skill is widely used in the entropy coding. Hence, how to conform different video standards without redesigning the original architecture by the Huffman coding properties is an important issue.

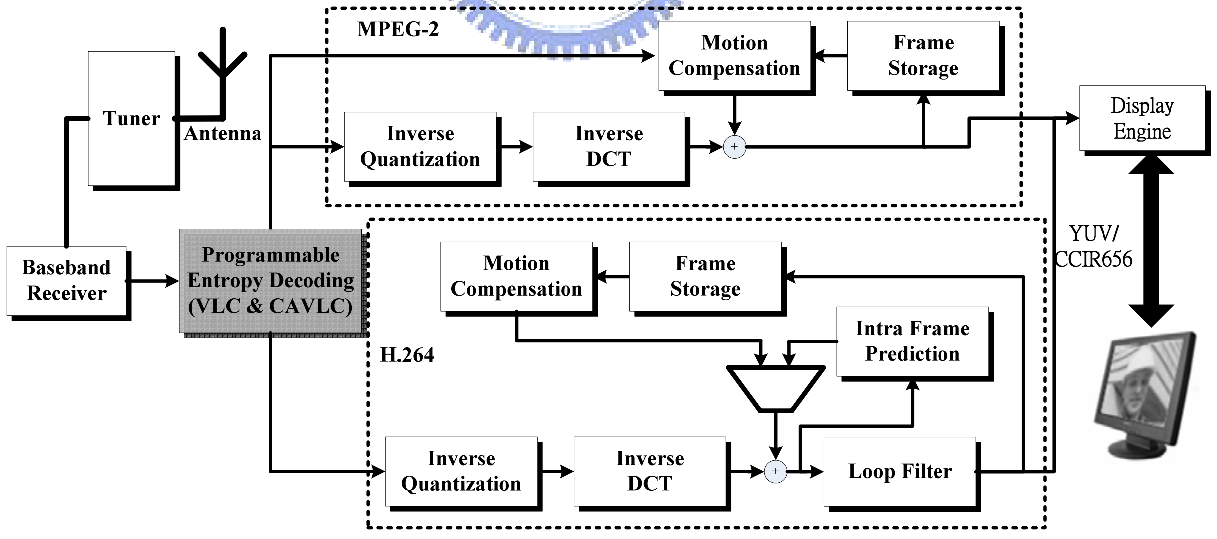


Fig.1.1 System view of programmable VLD

In the system point of view in Fig 1.1, this thesis will focus on the entropy decoding system in various video standards, such as VLC and CAVLC for MPEG-2 and H.264. We hope to design an architecture that can decode the bit streams from baseband receiver and conform to two video standards by the programmable the VLC tables existing in MPEG-2 and H.264. Further, the hardware overhead also hopes to be reduced.

For the purposes, some algorithms are developed to conform to different video entropy coding standards. Besides the algorithms can reduce the hardware overhead, the proposed architecture design also can simplify the decoding flow, and increase the memory utilization efficiency.

1.2 Overview of VLC and CAVLC

Huffman encoding skill is a well known lossless coding algorithm, whose principle is to assign the shorter codeword to more frequency information, and vice versa. Its coding efficiency is to be close to the entropy coding. The probability of the formula is described in equation (1).

$$\sum_{n=L_{min}}^{L_{max}} P(n) = 1 \quad \text{, for } n \in N, L_{min} \leq n \leq L_{max}$$

$$, \text{ and } \; P(L_{max}) \leq P(n) \leq P(L_{min}) \qquad (1)$$

$P(n)$: Using probability in codeword length n.

Therefore, it is used in many video standards, such as VLC and CAVLC in MPEG-2 and H.264. This section will introduce the principles of VLC and CAVLC.

1.2.1 The VLC algorithm of MPEG-2 standard

In the MPEG-2 standard, the entropy function block i.e. variable length coding is designed by the Huffman coding. There are fifteen tables, and are divided into two main parts.

The first part is used in macroblock header, it includes macroblock addressing, microblock type I-picture, microblock type P-picture, microblock type B-picture, microblock pattern and motion code. The second part is used in Run/Level symbols, intra DC luminance, intra DC chrominance, intra AC and non-intra AC is included.

In the tables of MPEG-2 mentioned in above, they are all established according to the statistics theory and use Huffman coding algorithm to reduce the redundant information. However, the decoding processes look up the table combined with source symbols and variable length codewords.

For example, Table 1.1 is a part of MPEG-2 table B-15, the decoding processes of VLC are that if received bitstream is 00111, it will map to the run and level with 5 and 1 respectively. Therefore, the decoding result is 5 and 1 [1].

Table 1.1: A part of MPEG-2 table B-15

Variable length code	run	level
0001 11	5	1
0001 10	4	1
0001 01	0	6
0001 00	0	7
0000 111	2	2
0000 110	6	1
0000 101	8	1
0000 100	7	1

1.2.2 The CAVLC algorithm of H.264 standard

In the H.264 standard, the entropy decoding process is different from the VLC of MPEG-2. It is not purely to look up tables. Further, it uses adaptive concept, and hence the entropy coding is named CAVLC (Context Adaptive Variable Length Coding). The CAVLC is mainly composed of five functions: coeff_token decoder, TrailingOnes decoder, level decoder, total_zero decoder and run_before decoder. The basic concept of coeff_token, total_zero and tun_before functions are to look up VLC tables. The meaning of every module and derivatives are described as follows:

1. **Coeff_token:** getting the coefficients of 4X4 macroblock, it includes the TotalCoeff and TrailingOnes.
2. **TotalCoeff:** the total coefficients of 4X4 residual macroblock, zeros are excluded.
3. **TrailingOnes:** the tail remainder of positive or negative one in 4X4 residual macroblock after zigzag scanning.
4. **Total_zero:** the number of total zeros in 4X4 macroblock.
5. **Run_before:** the number of zeros before every coefficient.

The decoding flow of CAVLC mainly includes several parts and is introduced by the Fig. 1.2:

1. Decoding the TotalCoeff and TrailingOnes: TotalCoeff = 5, TrailingOnes = 3
2. Decoding the sign of TrailingOnes: TrailingOnes sign = -1, -1, 1
3. Decoding the levels of the remaining non-zero coefficients: level = 3,1
4. Decoding the total number of zeros before the last coefficient: total_zeros = 3
5. Decoding each run_before of zeros: 3,0,1,-1,-1,0,1

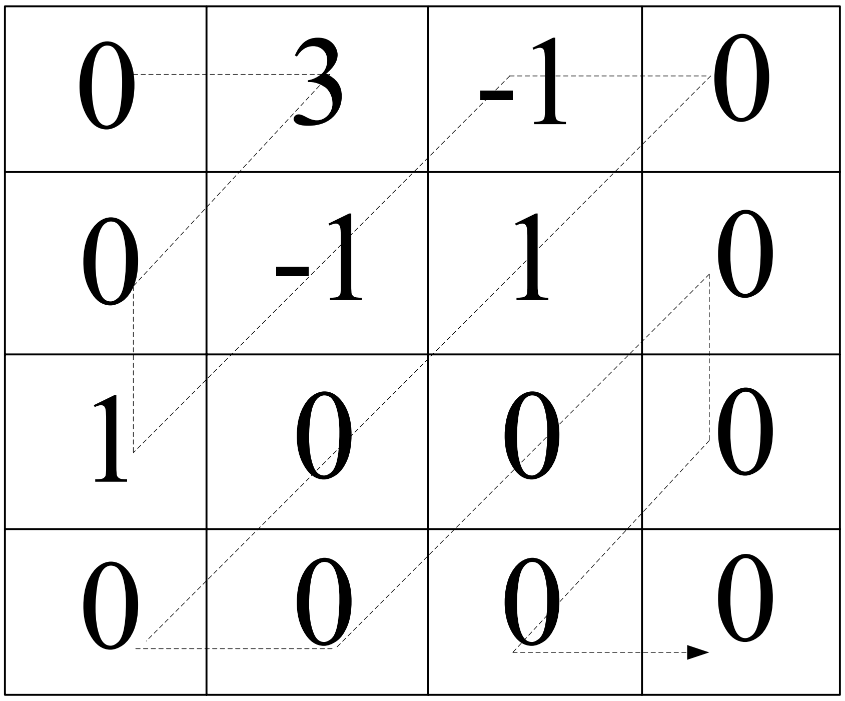


Fig.1.2 4X4 residual macroblock of H.264

The decoding flows of 1, 4 and 5 processes look up the Huffman coding-based tables such as Table 1.2. For example, if the received bitstream is 001111, the decoded symbol values TrailingOnes and TotalCoeff are 0 and 1 [2], respectively.

Table 1.2: A part of H.264 table 9-5

TrailingOnes(coeff_token)	TotalCoeff(coeff_token)	4<=nC<8
0	1	0011 11
1	6	0011 10
2	6	0011 01
3	9	0011 00

1.3 *Review of Prior Works*

In this section, some literatures are discussed and divided into three parts. The first is to review the VLC buffer and existing codeword prediction algorithm. The second is to introduce existing memory-based VLC algorithm and architectures. Furthermore, the newest memory-based CAVLC is introduced in the third part.

1.3.1 Codeword Boundary Prediction and Buffer

In the existing parallel decoding algorithms and architectures, the boundary of variable length code is difficult to divide because it doesn't have clear and defined demarcation line. But determining the variable length code boundary is a key point to increase the decoding speed. In Fig.1.2, the traditional methods are that the boundary of codeword is identified after the symbol is decoded but the traditional method is not suitable for high speed application because there is a feedback loop to align the next bitstream [3][4].

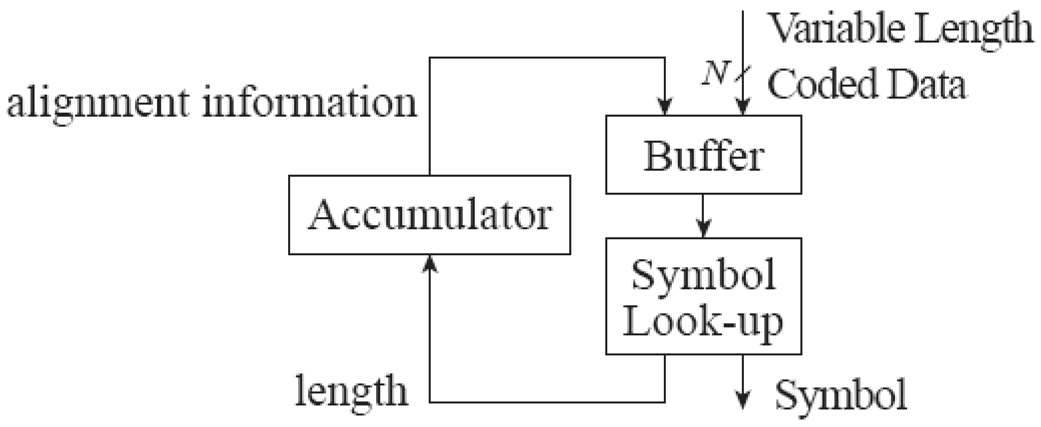


Fig.1.2 Traditional parallel variable length decoding buffer

Instead of feedback loop, Rudberg and Wanhammar replaced the barrel shifter with a shift register, where the decoder resembles parallel pipelined decoders [5]-[7]. It is illustrated in Fig. 1.3. The architecture makes it possible for the loop-free VLC decoder. It can send the bounded bitstream in parallel fashion to the symbol look-up unit. The principle is that pipelined length decoder decodes the input bitstream length before sending bitstream to the symbol look-up unit.

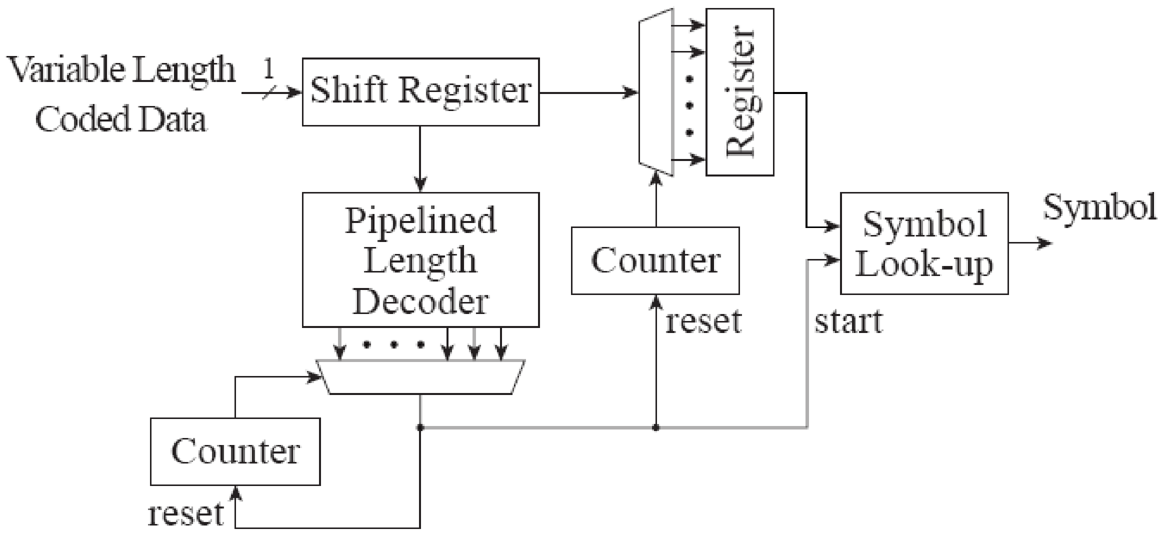


Fig.1.3 The loop-free architecture for VLC input buffer

Besides the above architecture and method, Shieh et al. propose codeword boundary prediction algorithm, where the input codeword boundary can be predicted before decoding the final symbol [8]-[10]. The principle is to develop branch models to prediction as shown in Fig.1.4.

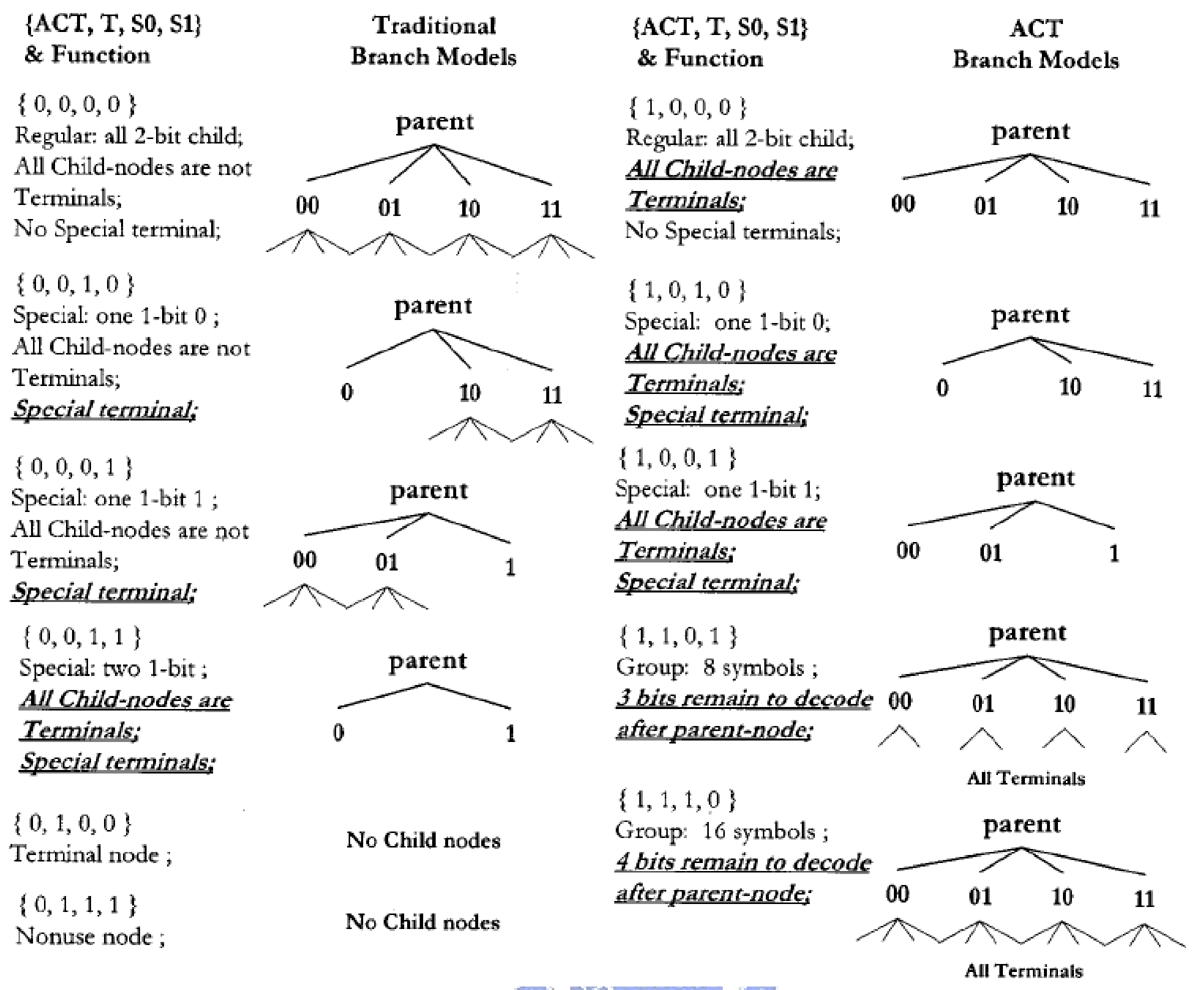
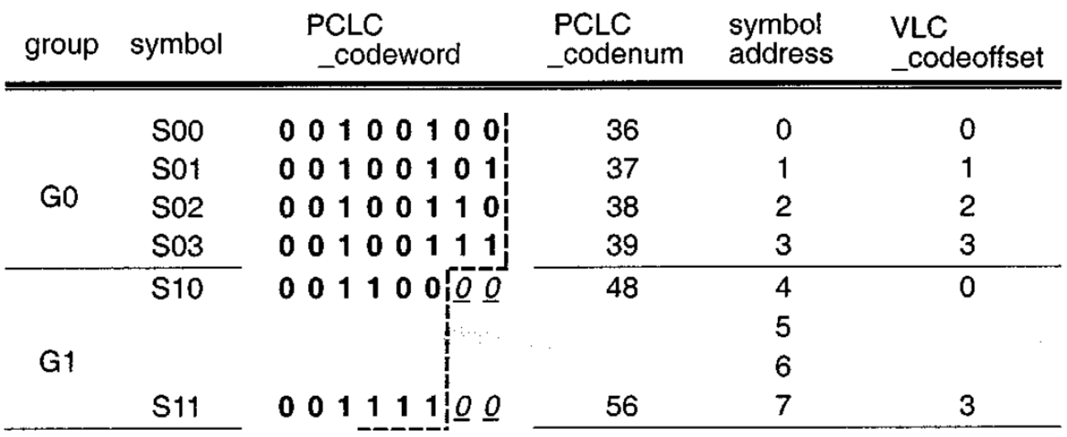


Fig.1.4 The traditional branch models and the proposal of Shieh et al.

1.3.2 Prior Works for VLC

In existing designs, several categories of VLC decoders have been proposed, such as CAM-, RAM-based designs [11]-[17]. However, the CAM-based designs require high cost to store all possible patterns while the RAM-based design is suitable for programmability. Shieh at al. propose memory-based VLC to decode and encode the bitstream and symbol. Moreover, the decoding flow is described as follows based on Fig 1.5:



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

Fig.1.5 An example of prior works [17]

1. Received bitstream : 0011_1100
2. Do group search: G1
3. Get minicode: 0011_0000, codelength: 6, base address: 4
4. Calculate codeoffset=0011_1100-0011_0000=0000_1100 $\rightarrow$ 00_0011 (because codelength:6)
5. Calculate symbol address: base address + codeoffset = 4 + 3 = 7

Besides, Chien et al. propose an improved grouping algorithm, and is described and summarized as follows based on Fig 1.6 [18]:

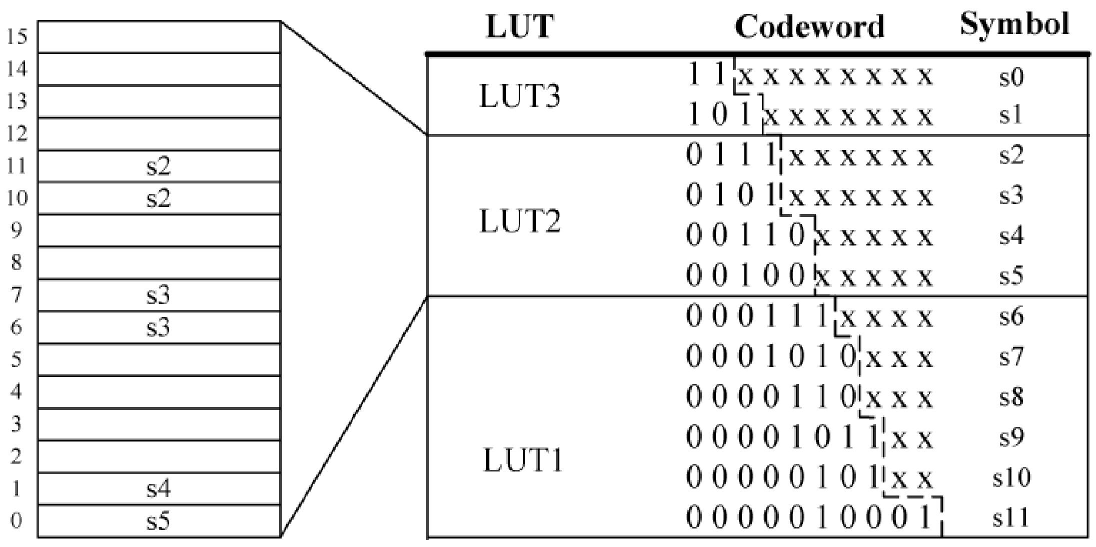


Fig.1.6 An example of prior works [18]

1. Received bitstream: 0_1011
2. Do group search: LUT2
3. Get m: 10, S: 5, minicode: 0_0100
4. Calculate symbol address: 0101_1xxx_xx >> 5 – 4 = 0101_1- 0010_0 = 0_0111 = $7_{(10)}$

1.3.3 Prior Works for CAVLC

In existing CAVLC design, it is rarely designed for programmable applications. The existent designs almost adopt hardwire method by the table property [19]-[22]. Yanmmei Qu and Yun use memory-based design method, but the memory is designed to fit the properties of tables in CAVLC. The other properties of VLC tables are not included, such as the VLC and CAVLC of MPEG-2 and H.264. Therefore it is not suitable for programmability applications [23]. The method of Qu et al. is introduced by the example of coeff_token module:

For the FLC table, the equation (2) is used.

$$(T1, TotalCoeff\_t) = \begin{cases} (0,0), \ when\ data[5:0]=3 \\ (data[1:0], data[5:2]), \ otherwise \end{cases} \qquad (2)$$

The equation (2) is based on the table of FLC properties and is hardwired architecture.

For the other VLC tables, GSGEM algorithm is proposed to use the main leading zero property of VLC tables and the following is the grouping rules:

1. If the length of the codewords with same leading zeros is the same, they are assigned to the same group. The group number is the number of leading zeros.
2. If the length of the codewords with same leading zeros is different, the codewords are grouped according their prefixes and the codewords with same length are assigned to the same group.
3. To reduce the symbol memory, they merge the symbols based on the range of TrailingOnes which is 0~3.
4. Because TotalCoeff ranging from 0 to 16 and few states of TotalCoeff equal to 0, they use a flag to show these states and store TotalCoeff_t of 4 bits instead of TotalCoeff of 5 bits to reduce one bit per memory unit of the symbol memory.

1.4 Thesis Organization

This thesis is organized as follows. At first, the Self-grouping and Reversed Cluster addressing algorithms are described in Chapter 2. Furthermore, the architectures of Self-grouping and Reversed Cluster algorithm are proposed in Chapter 3. Chapter 4 exhibits the simulation results to prove that the algorithms are efficient. Some comparison and implementation results are shown in Chapter 5. Finally, the contributions of this thesis and other issues for further research are highlighted in Chapter 6.

Chapter 2
Self-Grouping and Reversed Cluster Addressing Algorithm

To improve the efficiency of prior works, some algorithms and architectures are developed. In this chapter, the Self-grouping and Reversed Cluster Addressing algorithm are proposed. Before introducing the algorithms, some terminologies are introduced first. Furthermore, an example including Self-grouping and Reversed Cluster addressing is described in the last section.

2.1 Define Terminologies of Proposed Algorithm

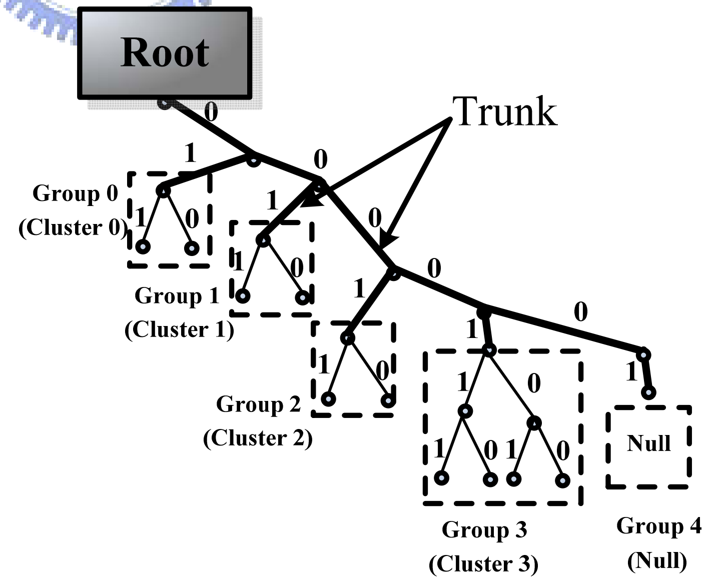
In this section, three terminologies of variable length code are defined [24][25]. As shown in Fig. 1.7, (a) are variable length code and (b) is the corresponding Huffman tree structure. Moreover, the terminologies are defined using leading 0's case. The leading 1's case is similar to the leading 0. They are defined as follows:

1. **Root:**
    1) Root is the starting point of the Huffman tree as shown in Fig. 1.7 (b).
2. **Trunk:**
    1) Trunk is the group information or the stop signal.
    2) In Fig. 1.7 (a), Trunk is the front zeros or one of variable length code.
    3) In Fig. 1.7 (b), Trunk is the thick line.

3. **Cluster:**
   1) Cluster is the set behind Trunk, i.e. Cluster 0, Cluster 1, Cluster 2 and Cluster 3.
4. **Null Cluster:**
   1) The Cluster 4 is the Null Cluster because that behind the Trunk, there is null set.
5. **Group:**
   1) In the Trunk of the variable length code, the same group means have the same zero numbers in front of Cluster such as group 0, group 1, group 2, group 3 and group 4 in Fig1.7.
   2) Group number and Cluster number are the same in the identify Huffman tree system, i.e. Group 0 = Cluster 0, Group 1= Cluster 1, Group 2= Cluster 2, Group 3= Cluster 3 and Group 4= Null Cluster.

Group	Variable length code		
0	01	1	Cluster 0
0	01	0	
1	001	1	Cluster 1
1	001	0	
2	0001	1	Cluster 2
2	0001	0	
3	00001	11	Cluster 3
3	00001	10	
3	00001	01	
3	00001	00	
4	000001		Null
	Trunk	Cluster	

(a)



(b)

Fig.2.1 (a) Variable length code and (b) The Huffman tree of variable length code

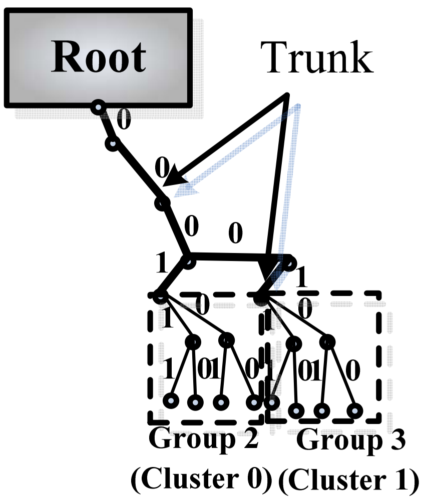
2.1.1 The table and corresponding Huffman tree of MPEG-2

In the Fig. 2.2, it is a part of table B-15 of MPEG-1 and the expanded Huffman tree. The terminologies of the Fig. 2.2 are the same as previous introduction. Further, every symbol, run and level are corresponding to unique variable length code (codeword). That is to say the codeword is one to one and mapping in the same table of MPEG-2.

In the Fig. 2.2 (b), the Trunk is thick line and there are two Clusters drawn by the dotted line i.e. Cluster 0 and Cluster 1 that they are corresponding to group 2 and 3 respectively.

Variable length code			run	level
0001	11	Cluster 0	5	1
0001	10		4	1
0001	01		0	6
0001	00		0	7
00001	11	Cluster 1	2	2
00001	10		6	1
00001	01		8	1
00001	00		7	1
Trunk	Cluster			

(a)



(b)

Fig.2.2 (a) A part of table B-15 in MPEG-2 standard and (b) Its Huffman tree

2.1.2 The table and corresponding Huffman tree of H.264

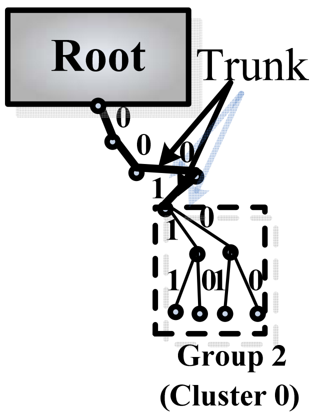
As shown in Fig 2.3, there is a part of table 9-5 of H.264. Similarly, every variable length code (codeword) is corresponding to unique symbol that is TrailingOnes and TotalCoeff. Fig 2.3 (b) is the expanded Huffman tree. The Trunk and Cluster are corresponding to Fig 2.3 (a).

The principle of CAVLC is also based on looking up table. Hence it must be one to one and mapping. Every leaves of the Huffman tree in Fig 2.3 are corresponding to one symbol.

TrailingOnes(coeff_token)	TotalCoeff(coeff_token)	2<=nC<4	
0	2	0001	11
1	4	0001	10
2	4	0001	01
3	7	0001	00
		Trunk	Cluster

Cluster 0

(a)



(b)

Fig.2.3 (a) A part of table 9-5 in H.264 standard and (b) Its Huffman tree

2.1.3 The leading 0 and leading 1 topology of Huffman tree

Fig. 2.4 is the Huffman tree of leading 1 and leading 0. The prior introduction is focusing on leading 0. However, the Huffman tree of leading 1 is shown in Fig.2.4 (a) existing widely in every table of VLC and CAVLC.

The difference between leading 0 and leading 1 is the Trunk. The Trunk begins in “1” for leading 1’s codeword and stop in 0. The leading 0’s Huffman tree is opposite. However, the Cluster have not the above property, the property of leading 1 and leading 0 codewords are the same.

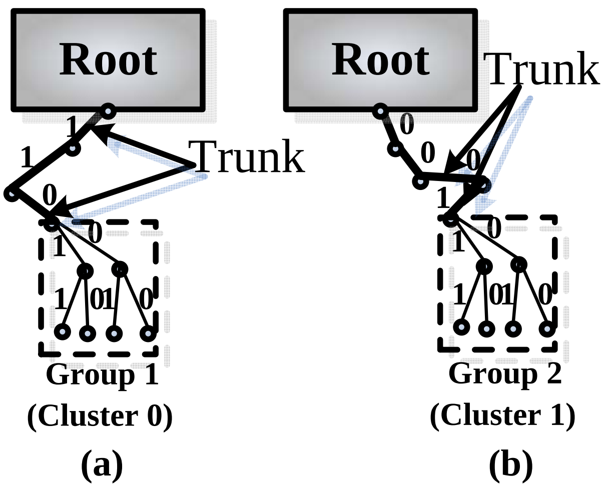


Fig.2.4 (a) A leading 1 Huffman tree (b) A leading 0 Huffman tree

2.2 *Self-grouping Algorithm*

In this section, Self-grouping algorithm is introduced. The meaning of Self-grouping is that the grouping information exists naturally in the Huffman tree and the grouping method is that using self-information and it results the name of algorithm is Self-grouping algorithm. As shown in Fig.2.5, there is information. The first is Cluster information, the second is stop signal information and the third is grouping information. In this section, the grouping information is discussed.

From the Fig. 2.5, we define the numbers of zero as groups. For example, if there is one zero in the front of codeword, the group 0 is allocated and if there are seven zeros, the group 6 is allocated and so on.

After allocating the group, there is a signal opposite to the front group information and the signal is named as stop signal that needn't to store the information. Hence, after grouping the codeword there is not needed to store any information in the decoder. The need is to identify theses information.

In the Fig. 2.5, the original bit needed to be stored is 76 bits, but after applying Self-grouping algorithm, there are only 23 bits needed to be stored for example. Hence, 69.74 % codeword size is reduced by applying the Self-grouping algorithm.

.

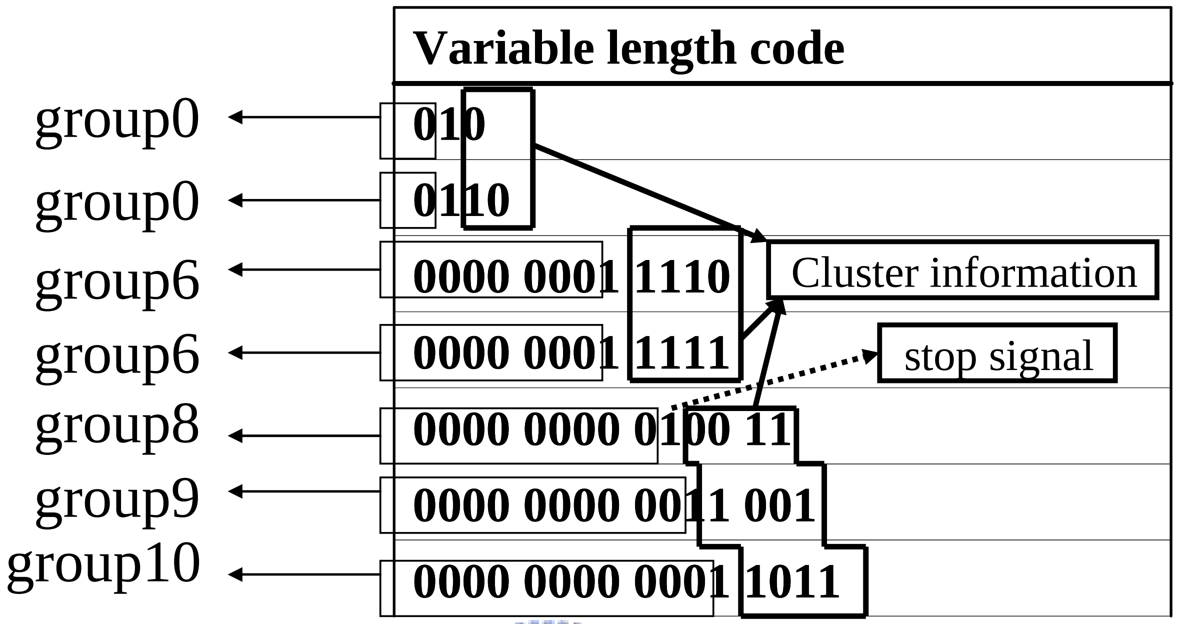


Fig.2.5 The variable length code and its self-grouping information

2.3 *Reversed Cluster Addressing Algorithm*

In this section, Reversed Cluster addressing algorithm is described and some issues are discussed for memory applications.

The meaning of Reversed Cluster addressing is that the addressing method is to use reversed Cluster bitstream. In the Fig.2.6, there are some differences between reversed and non-reversed Cluster addressing. If Cluster information is used to address the memory location, the reversed Cluster information is better than non-reversed one because the needed least memory space is less than non-reversed Cluster addressing.

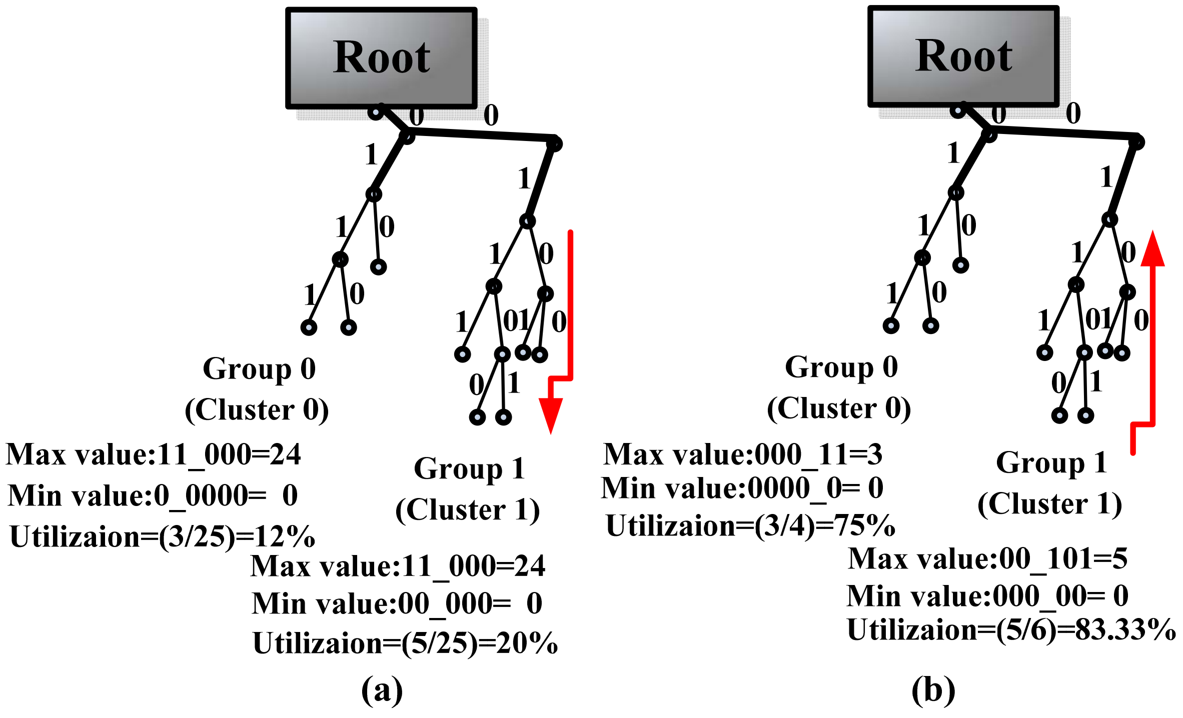


Fig.2.6 (a) Normal Cluster addressing (b) Reversed Cluster addressing

For example, in Fig. 2.6 (a), if the non-reversed Cluster is used, the corresponding symbol memory utilization is shown as follows by equation (3):

$$\begin{cases} Group0\_Utilization = \dfrac{(Group(0)\_Symbol\_number)}{(Group(0)\_Max\_value+1)} = \dfrac{3}{25} = 12\% \\ Group1\_Utilization = \dfrac{(Group(1)\_Symbol\_number)}{(Group(1)\_Max\_value+1)} = \dfrac{5}{25} = 20\% \\ \\ TableN\_Utilization \\ = \dfrac{\sum_{i=0}^{1}\sum_{j=0}^{l_i}(Leading(i)\_Group(j)\_Symbol\_number)}{\sum_{i=0}^{1}\left\{\sum_{j=0}^{l_i}(Leading(i)\_Group(j)\_Max\_value)+l_i\right\}} = \dfrac{8}{50} = 16\% \\ \\ Where\ N\ is\ the\ table\ of\ VLC\ and\ CAVLC, \\ l_i\ is\ the\ maximum\ group\ number\ of\ tableN\ in\ leading\ 0\ or\ leading\ 1 \end{cases} \quad (3)$$

The group0_utilization is 12% and group1_utilization is 20% but the table utilization is 16% calculated by the equation (3).

If the Reversed-Cluster algorithm is used, the addressing mode will follow the formula (4).

$$\begin{cases} Group0\_Utilization = \dfrac{(Group(0)\_Symbol\_number)}{(Reversed\_Group(0)\_Max\_value+1)} = \dfrac{3}{4} = 75\% \\ Group1\_Utilization = \dfrac{(Group(1)\_Symbol\_number)}{(Reversed\_Group(1)\_Max\_value+1)} = \dfrac{5}{6} = 83.33\% \\ TableN\_Utilization \\ = \dfrac{\sum_{i=0}^{1}\sum_{j=0}^{l_i}(Leading(i)\_Group(j)\_Symbol\_number)}{\sum_{i=0}^{1}\left\{\sum_{j=0}^{l_i}(Reversed\_Leading(i)\_Group(j)\_Max\_value)+l_i\right\}} = \dfrac{8}{10} = 80\% \\ Where\ N\ is\ the\ table\ of\ VLC\ and\ CAVLC, \\ l_i\ is\ the\ maximum\ group\ number\ of\ tableN\ in\ leading\ 0\ or\ leading\ 1 \end{cases} \tag{4}$$

If Reversed Cluster addressing is used, 64% symbol memory utilization can be improved for the case of Fig.2.6.

2.4 *An example of the Proposed Algorithm*

In Fig.2.7, an example of proposed algorithm is introduced and is simplified several steps shown as follows:

1. Received the bitstream of table 0: 0000_0110
2. Group detect: group 4
3. Codeword address: $4_{(10)}$+{Table_select=0, Bitstream [15], 4'd 0}=00_0100
4. Cluster: 10_000: reversed Cluster: 000_01
5. Symbol address:

base_address [codeword_address]+reversed Cluster=0000_0100+000_01=5$_{(10)}$

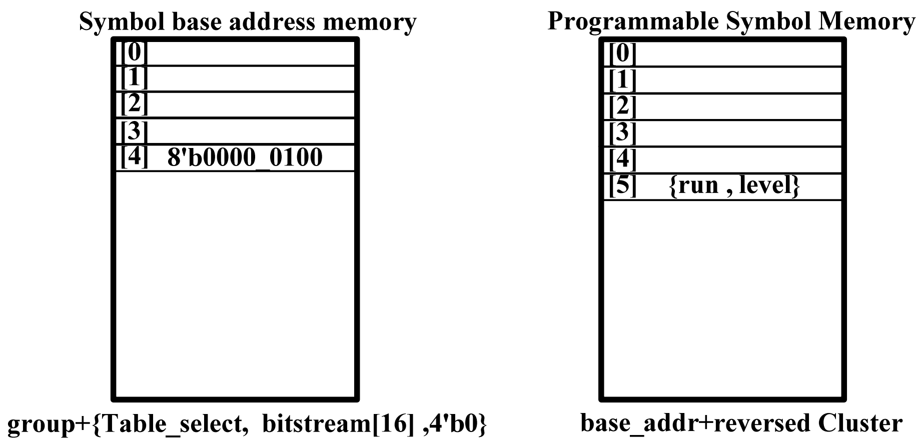

Symbol base address memory
[0]
[1]
[2]
[3]
[4] 8'b0000_0100
group+{Table_select, bitstream[16] ,4'b0}
Programmable Symbol Memory
[0]
[1]
[2]
[3]
[4]
[5] {run , level}
base_addr+reversed Cluster

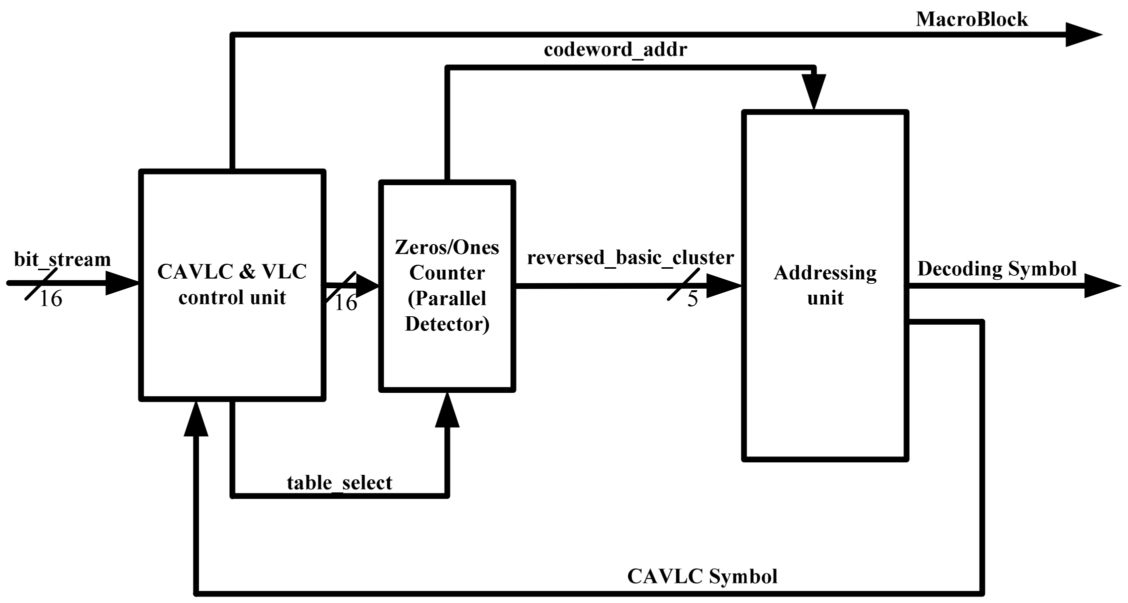
Fig.2.7 An example of the proposed algorithm

Chapter 3
Proposed Architecture Of Self-Grouping and Reversed Cluster Addressing Algorithm

In this Chapter, the proposed architectures are presented. Moreover, the hardware block diagram of VLC and CAVLC is shown in Fig.3.1. The input bit_stream through the CAVLC & VLC control unit and Zeros/Ones counter, the Zeros/Ones Counter extracts codeword_address and reversed_basic_cluster signals to the addressing unit.

Furthermore, if it is in the VLC decoding action, the decoding symbol is sent to other function block such as inverse quantization or motion compensation. However, if it is in the CAVLC decoding flow, there is a feedback to the control logic to fit the CAVLC FSM decoding action.

Hence, in this Chapter, two main parts are classified and introduced. The first is the implementation of Self-grouping algorithm i.e. Zeros/Ones Counter and introduced in section 3.1. The second is addressing unit, it is the main part to address the symbol memory and composed of memories and introduced in section 3.2. Finally, some examples of the hardware action are described in section 3.3.



.

Fig. 3.1 The implementation hardware block diagram of VLC and CAVLC

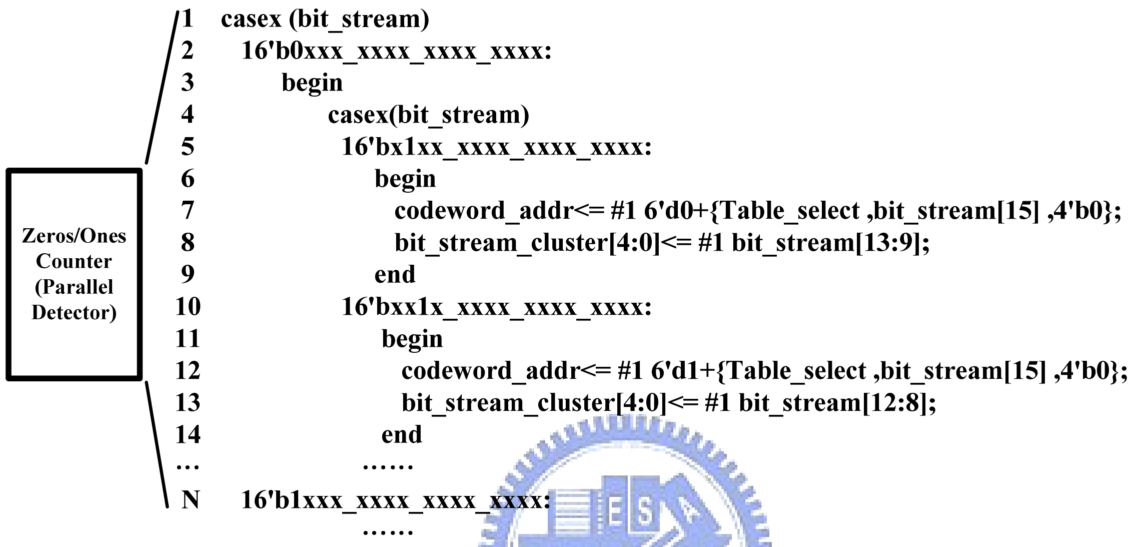
3.1 *The Architecture of Self-grouping Algorithm*

The Architecture of Self-grouping algorithm is described in HDL code, as shown in Fig.3.2. Although the name is Zeros/Ones counter, it is implemented using parallel detector skill. The principle of the hardware is described as follows:

1. The line 1 and line N identify the leading 0 or leading 1
2. The line 5 and line 10 extract the codeword address and bit stream cluster to the addressing unit.

The codeword address is composed of Table_select, bitstream [15] and group. The bitstream[15] is the location of symbol base address memory, its intervals are 16. Therefore, it must be in the front of group and behind the Table_select. Moreover, to integrate symbol base address memory to single memory in different tables in VLC and CAVLC, the

Table_select is in the location of MSB of codeword_addr. Hence, it means the interval of every table in symbol base address memory is 32. Although there is some waste of symbol base address memory, it is worth doing this arrangement instead of many symbol base address memories having larger overheard.



```
casex (bit_stream)
  16'b0xxx_xxxx_xxxx_xxxx:
      begin
          casex(bit_stream)
            16'bx1xx_xxxx_xxxx_xxxx:
               begin
                 codeword_addr<= #1 6'd0+{Table_select ,bit_stream[15] ,4'b0};
                 bit_stream_cluster[4:0]<= #1 bit_stream[13:9];
               end
            16'bxx1x_xxxx_xxxx_xxxx:
                begin
                  codeword_addr<= #1 6'd1+{Table_select ,bit_stream[15] ,4'b0};
                  bit_stream_cluster[4:0]<= #1 bit_stream[12:8];
                end
              ……
  16'b1xxx_xxxx_xxxx_xxxx:
              ……
```

Fig. 3.2 The HDL code of parallel detector of Zeros/Ones counter

3.2 *The Architecture of Reversed Cluster Addressing Algorithm*

The architecture of reversed Cluster addressing is addressing unit in Fig. 3.1, it is composed of two memories. The first is symbol base address memory and the second is symbol memory.

Symbol address memory stores the base address of symbol, and the stored information is an addressing accumulation by the dispersed sequence equation (5) as shown in follows:

$$\begin{cases} \sum_{i=0}^{1}\left\{ Leading(i)\_Reversed\_Group(j)\_Max\_value + \sum_{j=0}^{s-1} Leading(i)\_Reversed\_Group(j)\_Max\_value \right\}_{j=0}^{l_i} \\ l_i \text{ is the maximum group number of tableN in leading 0 and leading 1} \\ s \text{ is the (present group number -1)} \end{cases} \quad (5)$$

Fig. 3.3 shows the addressing unit, first the codeword address sends address to symbol base address memory from Zeros/Ones counter to get the base address of symbol. Meanwhile, the reversed Cluster addressing signal is sent to the adder shown in Fig.3.3, the sum of symbol base address and {3'b0, bit_stream_cluster [0:4]} is the correct symbol address.

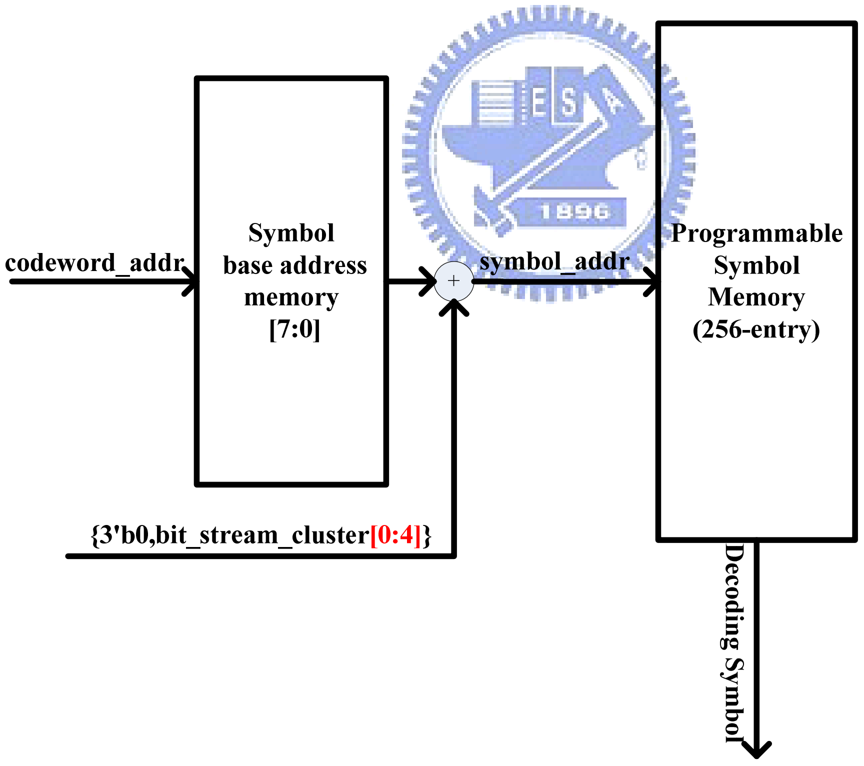

codeword_addr
Symbol base address memory [7:0]
+
symbol_addr
Programmable Symbol Memory (256-entry)
{3'b0,bit_stream_cluster[0:4]}
Decoding Symbol

Fig. 3.3 The architecture of addressing unit

The design concept of bit_stream_cluster is five bits width because the highest Cluster existing in VLC and CAVLC is five bits high as shown in Fig.3.4.

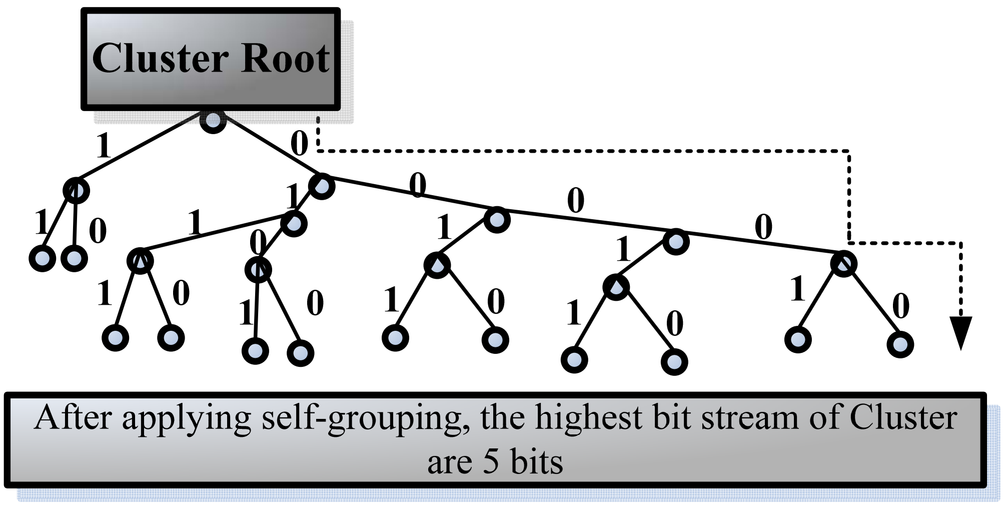

Cluster Root
1
0
1
0
1
0
0
0
0
1
1
1
1
0
1
0
1
0
1
0
1
0
1
0
After applying self-grouping, the highest bit stream of Cluster are 5 bits

Fig. 3.4 The highest in Basic Cluster of leading0 and 1's case

3.3 *Some examples of hardware decoding flow*

Based on Fig. 3.5, some decoding flows are presented to help understand the hardware action.

<u>Example 1</u>, if the bitstream 0110 of Table 0 is received:

1. 0110 through Zeros/Ones counter and extract the codeword address

   Ex:

   1) codeword_addr

      = group + {Table_select, bitstream [15], 4'b0}

      = $0_{(10)}$ + {00, 0, 0000}

      = 000_0000

      = 0

2. The symbol base address is obtained as follows

   Ex:

   1) symbol_base_addr [codeword_addr]

= symbol_base_addr [0]

= 8’b0000_0000

3. At the same time, the reversed Cluster is also extracted

   Ex:

   1) reversed_cluster

      = {3’b0, bit_stream_cluster [0:4]}

      = 8’b0000_0001

4. The sum of symbol_base_addr and reversed_cluster is the symbol address

   Ex:

   1) symbol_addr

      = symbol_base_addr + reversed_cluster

      = 8’b0000_0000 + 8’b0000_0001

      = 8’b0000_0001

      = $1_{(10)}$

   2) {run, level}

      = programmable symbol memory [symbol_addr]

      = programmable symbol memory [1]

<u>Example 2</u>, if the bitstream 1111_1010 of Table 0 is received:

1. 1111_1010 through Zeros/Ones counter and extract the codeword address

   Ex:

   1) codeword_addr

      = group + {Table_select, bitstream [15], 4’b0}

      = $4_{(10)}$ + {00, 1, 0000}

      = 001_0100

= $20_{(10)}$

2. The symbol base address is obtained as follows

   Ex:

   1) symbol_base_addr [codeword_addr]

      = symbol_base_addr [20]

      = 8’b0110_0100

3. At the same time, the reversed Cluster is also extracted

   Ex:

   1) reversed_cluster

      = {3’b0, bit_stream_cluster [0:4]}

      = 8’b0000_0001

4. The sum of symbol_base_addr and reversed_cluster is the symbol address

   Ex:

   1) symbol_addr

      = symbol_base_addr + reversed_cluster

      = 8’b0110_0100 + 8’b0000_0001

      = 8’b0110_0101

      = $101_{(10)}$

   2) {run, level}

      = programmable symbol memory [symbol_addr]

      = programmable symbol memory [101]

Example 3, if the bitstream 0001_00 of Table 3 received:

1. 1111_1010 through Zeros/Ones counter and extract the codeword address

   Ex:

   1) codeword_addr

      = group + {Table_select, bitstream [15], 4’b0}

      = $2_{(10)}$ + {11, 0, 0000}

      = 110_0010

      = $98_{(10)}$

2. The symbol base address is obtained as follows

   Ex:

   1) symbol_base_addr [codeword_addr]

      = symbol_base_addr [98]

      = 8’b1111_0100

3. At the same time, the reversed Cluster is also extracted

   Ex:

   1) reversed_cluster

      = {3’b0, bit_stream_cluster [0:4]}

      = 8’b0000_0000

4. The sum of symbol_base_addr and reversed_cluster is the symbol address

   Ex:

   1) symbol_addr

      = symbol_base_addr + reversed_cluster

= 8’b1111_0100 + 8’b0000_0000

= 8’b1111_0100

= $244_{(10)}$

2) {run, level}

= programmable symbol memory [symbol_addr]

= programmable symbol memory [244]

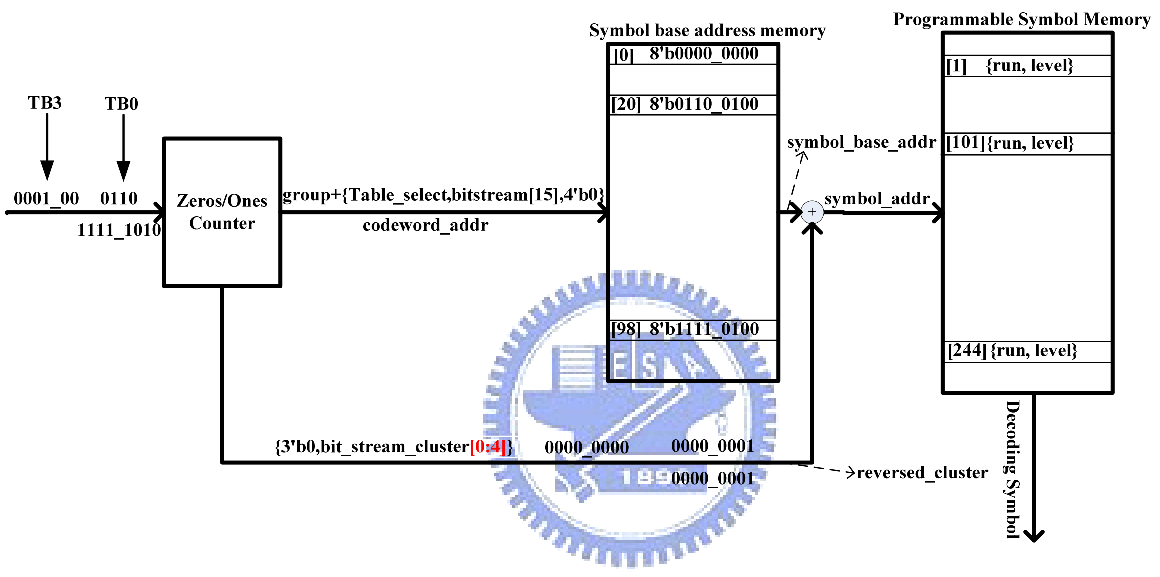


Fig. 3.5 Some examples of decoding flow

Chapter 4
Simulation Results

In this chapter the simulation results are presented, it includes the symbol memory utilization in fix length code and variable length code. Furthermore, the simulation results prove the proposed algorithm is efficient.

4.1 Variable Length Code Simulation Results

Fig. 4.1 is the needed entries of symbol memory in non-reversed Cluster addressing, reversed Cluster addressing and the original symbol number. Moreover, the table number is assigned to sequence number. Table 0~ 10 are MPEG-2 non-scalable and Tables 50~53 are for scalable application. Further, Tables 11~49 are CAVLC tables 9-5~ 9-10.

We can find the least needed entries number of reversed Cluster addressing drawn in tetragon is less than the non-reversed Cluster addressing drawn in rhombus in Fig 4.1. Further, the reversed Cluster addressing is closely to the original symbol number drawn in triangle. That is why a reversed Cluster addressing algorithm is adopted.

There are several important tables that are worth to discuss. Table 9 and Table 10 of Fig 4.1 are the tables 14 and 15 in MPEG-2 that is the maximum table size in VLC and CAVLC. If non-reversed Cluster addressing algorithm is adopted, it needs 279 and 369 entries at least but reversed Cluster addressing is adopted, 131 and 144 entries is needed. The 133 and 144 is closely to the 114 and 113 of original symbol number.

Moreover, the symbol memory utilization that excludes VLC of MPEG-2 from the Fig.

4.1 is almost 100%. Furthermore, there don't have any non-reversed Cluster addressing that is larger than reversed Cluster addressing symbol memory utilization.

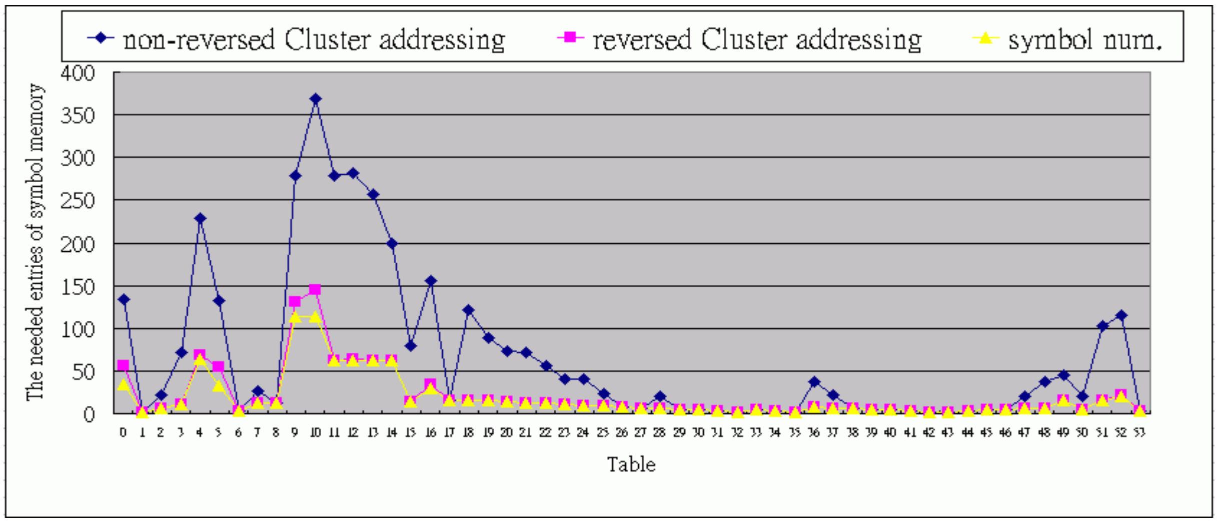


Fig. 4.1 The needed entries of symbol memory (MPEG2 (TB0~10, 50~53) and H.264 (TB11~49))

To be more clear that Fig. 4.2 is presented to show the symbol memory utilization. Although there are also 100% symbol memory utilization, the non-reversed symbol memory utilization is about 20%. However, if reversed Cluster addressing is used, the utilization lies in 80% ~ 100%. In Fig. 4.2, it is easily to understand the advantage of reversed Cluster addressing.

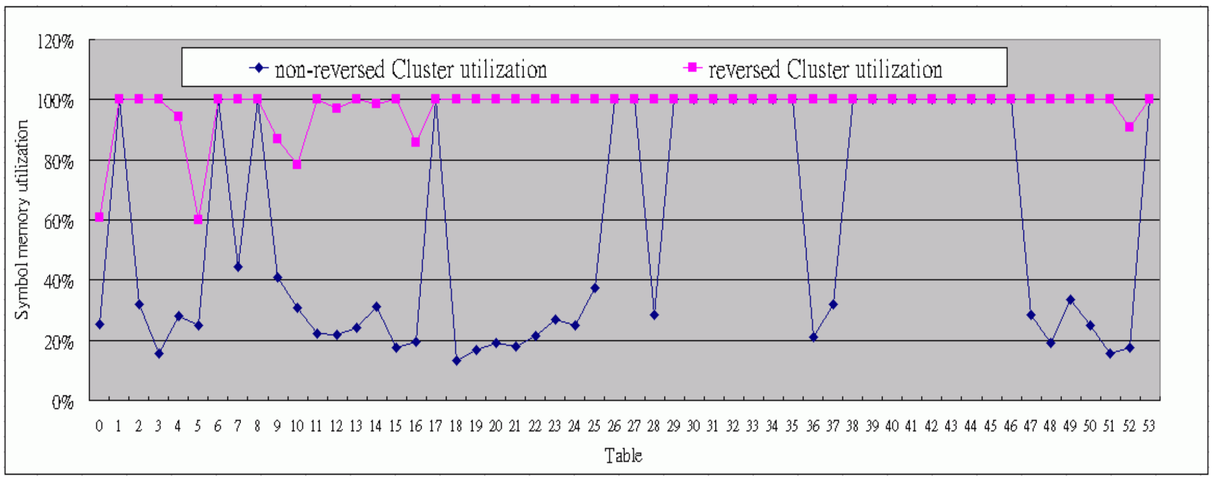


Fig. 4.2 The symbol memory utilization (MPEG2 (TB0~10, 50~53) and H.264 (TB11~49))

Fig. 4.3 is the whole comparison of VLC and CAVLC. The total entries of symbol memories are 1084 if reversed Cluster addressing algorithm is used. However, there are 3563 entries needed to store the total symbols if non-reversed Cluster addressing algorithm is used. The range is 67.40% when using two different algorithms. Further, the original symbol numbers are 978 entries, it is closely to 1084 and that only 106 entries are wasted. But the wasted entries are suitable for various programmable applications.

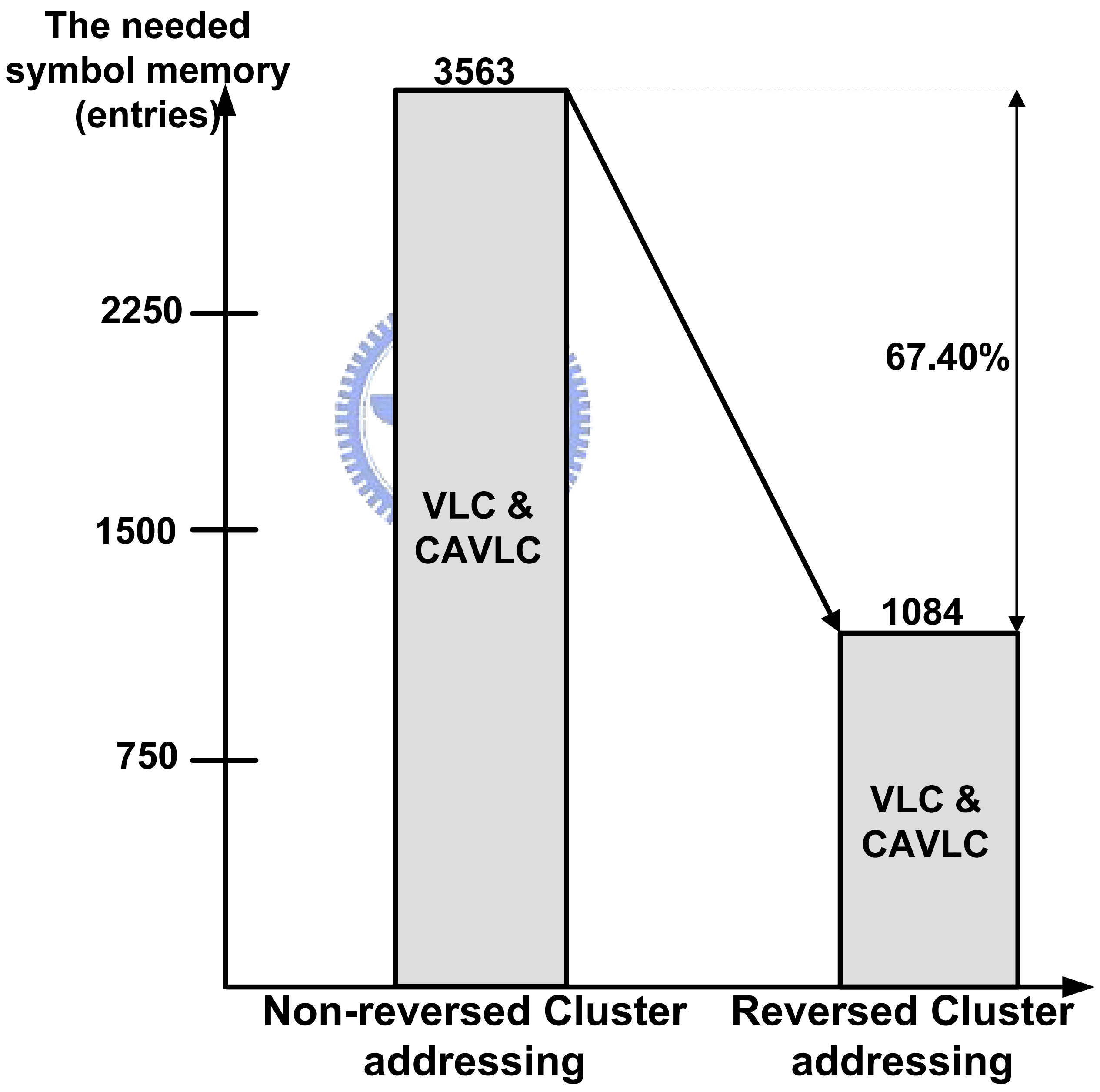


Fig. 4.2 The entries of needed symbol memory

Fig. 4.3 uses another point of view to understand the advantage of using reversed Cluster addressing. In the total simulation of VLC and CAVLC, if non-reversed Cluster addressing is used, only 27.42% symbol memory is utilized. The wasting of symbol memory is very huge. Hence, not only power consumption but also area is suffered very huge challenge. However, if reversed Cluster addressing algorithm is used, the challenge promotes to 90.22% and that 62.80% is improved.

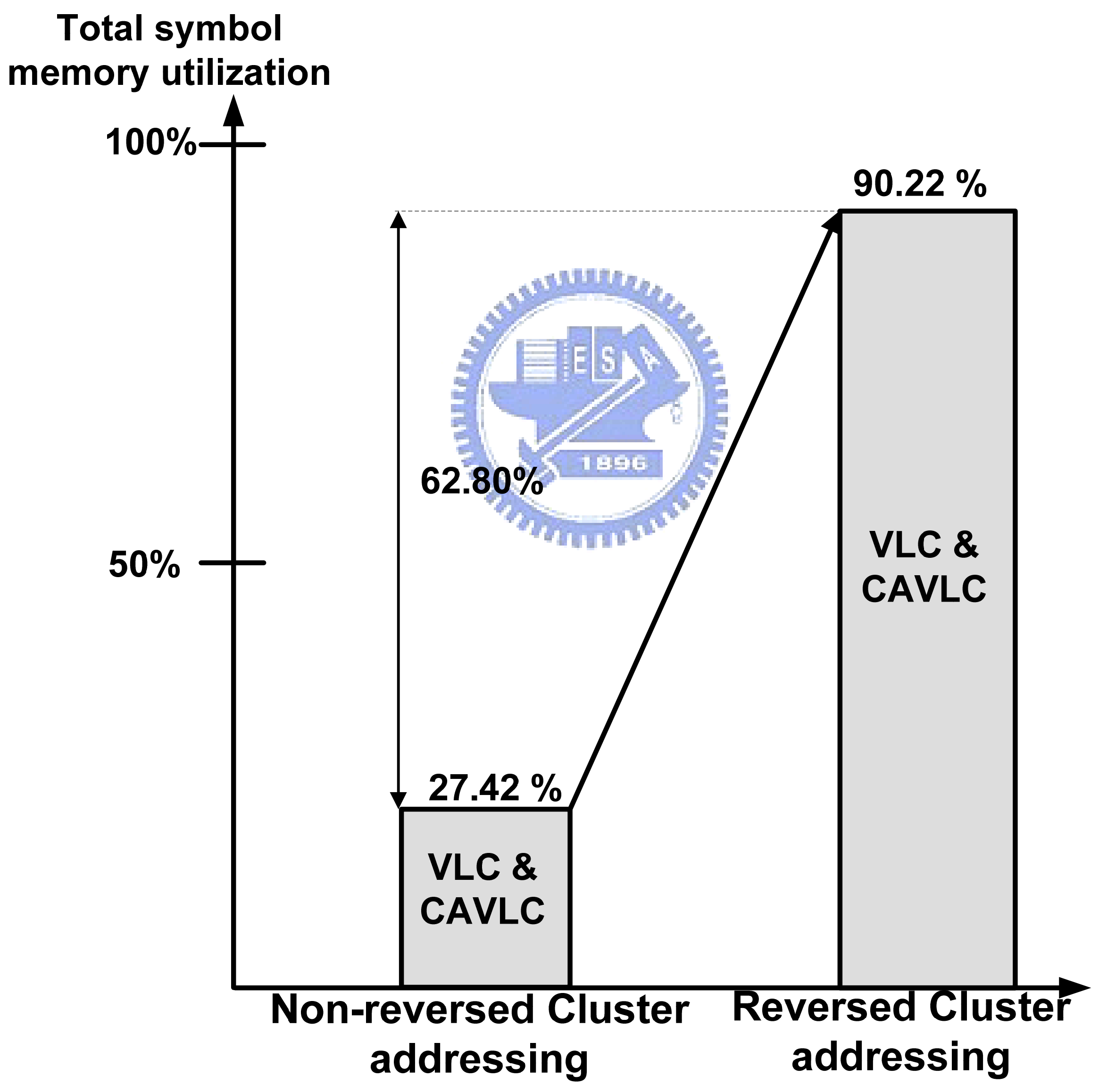


Fig. 4.3 The symbol utilization (MPEG2 and H.264)

4.2 Fixed Length Code Simulation Results

This section provides a very interesting phenomenon as shown in Fig 4.4. It contains different data: the first is the original symbol number, and the second is using fix length code to addressing symbol memory and finally the proposal is using Self-grouping and reversed Cluster addressing algorithm to address the symbol memory.

Because CAVLC table 9-5, 8<=nC is a fixed length codeword, we adopt it to have a experiment. We can find the original symbol numbers are 62 entries. If using fix length addressing the symbol memory, there must have 63 entries symbol memory to fit the all symbol memory for table 9-5, 8<=nC. However, if proposed algorithm is used to address the total symbol memory, it also needs 63 entries to addressing all symbol memory. The efficiency of fix length addressing and the proposed algorithm remains the same. Thus it can be seen the proposed algorithm not only suit variable length code but also fixed length code.

There are advantages of the results mentioned in above. First, that means not only fixed length but also variable length code can share the same hardware. The second is that the implementation is more easy because it doesn't need other logic to process fixed length code and variable length code.

Beside the results of above, there are sequence codewords i.e. 0000, 0001, 0010, … 1111. The simulation results show that the symbol memory utilization is 100% that is also the same as fixed length code addressing.

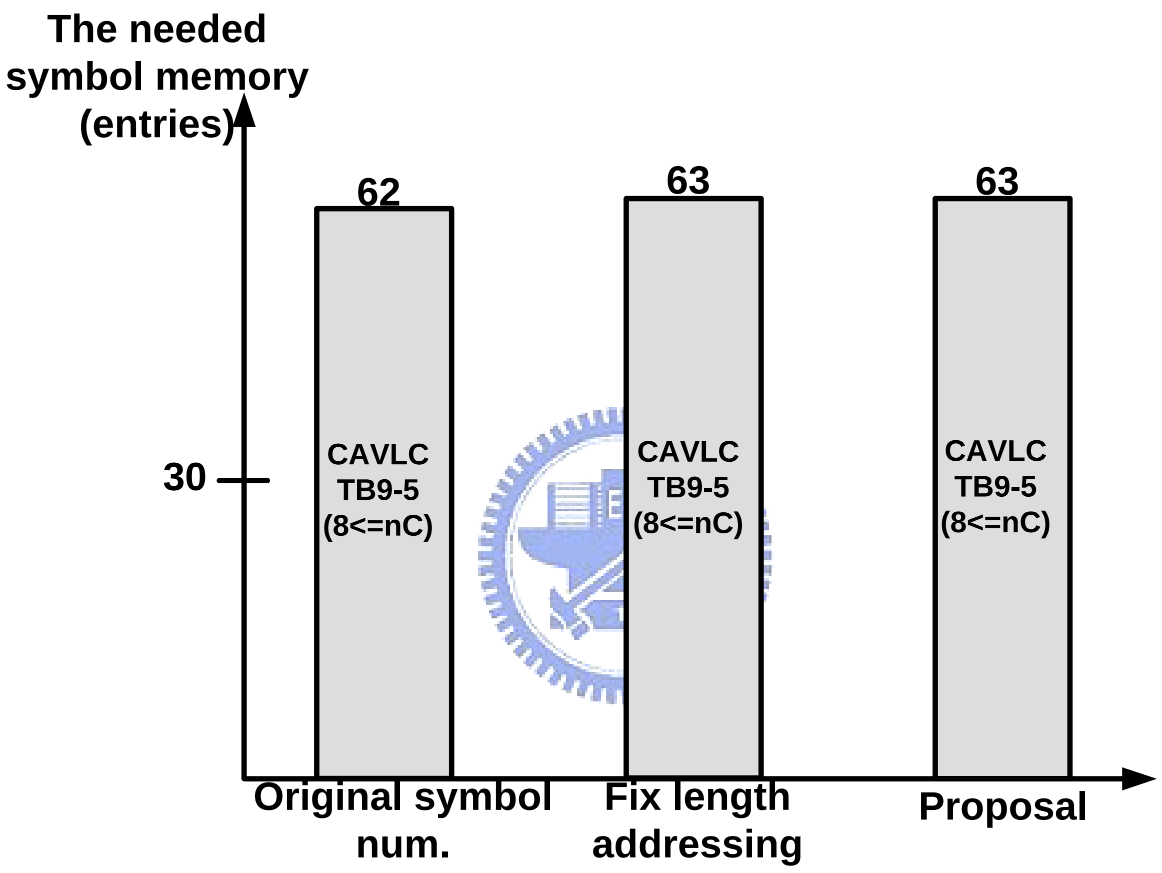


Fig. 4.4 The symbol utilization of fix length code and proposal

Chapter 5
Comparison of Implementation Results

5.1 Comparison of VLC

The comparison of VLC is shown in Table 5.1. References [17] and [18] are some comparisons by using MPEG-2 standard. However, our proposal is using merged MPEG-2@MP and H.264@BP to have a comparison. Further, the codeword memories of [17] are 928 bit registers but [18] and our proposal use SRAM. Furthermore, under 0.6 um and 0.18 um process, the gate counts are 110K, 7.8K and 16.34K.

Comparing to [17] and [18] and our proposal, both references [17] and [18] must have two groups to be codeword memory, but our proposal can be merged as single SRAM to be a codeword memory. The “VLC TB14 & TB15 (non-merge)” means that the tables B-14 and B-15 of MPEG-2 have the same codeword and symbol and that we don’t merge them. However, our algorithm and architecture can merge the codewords that has same symbols, but [17] can’t do it. Further, if we merge both tables B-14 and B-15 by the same codewords having the same symbols, reference [18] needs 4096 bits for two tables, and 2264 bits are needed for our proposal under the same merged method.

There is something worth having a comparison that the grouping method of reference [17] and [18] are searching method but our proposal uses detection method. It reduces the overhead of grouping too many groups by using SRAM.

Table 5.1 Comparison of VLC

Design	[17]	[18]	Proposal
Standard	MPEG2	MPEG2	(MPEG2@MP ／H.264@BP)
Leading 0/1 constraint	No	No	No
Codeword memory (VLC TB14 & TB15)	928 bit Registers (Group information X 2)	(LUT memory X2)	320 bits SRAM (Base address memory X 1)
VLC TB14 & TB15 (non-merge)	4000 bits	NA	3668 bits
VLC TB14 & TB15 (codeword & symbol merging)	Can't	4096 bits	2264 bits
Process	0.6 um	0.18um	0.18 um
Grouping algorithm	Search	Search	Detect
Clock rate	100MHz	125MHz	125MHz (Max: 200MHz)
Gate Counts	110K (enc/dec) (with mem.)	7.8K (decoder) (with mem.)	16.34K (decoder) (with mem.)

5.2 *Comparison of CAVLC*

Table 5.2 is the comparison of CAVLC. The design of reference [11] is for VLC 0, VLC 1 and VLC 2 tables i.e. table 9-5 0 <= nC < 8. Hence, the needed codewords and symbol memory are about 69 and 121 bytes. Because the design is to consider the properties of VLC 0, VLC 1 and VLC 2, it is not programmable. However, if we also design the hardware module by the properties of VLC 0, VLC 1 and VLC 2 and the proposed algorithms also are used i.e. (35+7) X 8 for codeword memory and (62+64+62) X 6 for symbol memory. The symbol memory of 6 bits is used because we use a flag for the TotalCoeff is zero instead of 5 bits TotalCoeff. It results the TotalCoeff is 4 bits. Hence, the TrailingOnes and TotalCoeff are totally 6 bits width. Moreover, the total memory size is 1464 bits and it is smaller than reference [11].

Table 5.2 Comparison of CAVLC

Design	[11]	Proposal
Standard	H.264 (VLC0, VLC1, VLC2)	H.264 (VLC0, VLC1, VLC2)
Leading 0/1 constraint	No	No
Codeword memory (VLC0, VLC1, VLC2)	about 69 bytes (552 bits)	336 bits SRAM (Base address memory X 1)
Symbol memory	about 121 bytes (968 bits)	1128 bits
Total memory	1520 bits	1464 bits
Programmable?	No	No

Chapter 6
Conclusion and Future Work

6.1 Conclusion and contribution

In this thesis, we propose an efficient and programmable VLC decoder to reduce the complexity of hardware and the needed memory size.

In general, the VLC decoder grouping method is to use search method, but we use detection method to increase the decoding speed because we use the skill that grouping information existing naturally in the Huffman tree. Specially, we simplify the memory addressing algorithm by reversed Cluster addressing action to achieve 90.22% symbol memory utilization. Further, 62.80% symbol memory utilization is increased compared to non-reversed Cluster addressing. Hence, the memory size of 43.40% and 44.73% are reduced for MPEG-2 table B-14 and table B-15 compared to reference [17] and [18], respectively. However, if non-programmable architecture is used, 3.69 % memory size is reduced compared to [11] for VLC 0, VLC 1 and VLC 2 of CAVLC. To reduce the architecture overhead, the single memory architecture is designed. That means our proposal becomes possible to merge different tables used in the same standard or different standards. It can reduce the overhead of too many symbol or codeword memories.

6.2 Future Work

In the implementation results, several improvements can be done to integrate this design into the system. First, the input buffer should be implemented according to references [5]-[10] to achieve high speed decoding in parallel. Beside, the memory should be initialized by external memory or software to be integrated and can work in the system. In this thesis, the initialization of memory is in the verilog HDL code by reading programmed text file.

Bibliography

[1] ISO/IEC 13181-2, "Information Technology – Generic Coding of Moving Pictures and Associated Audio Information: Video," 1995

[2] Draft ITU-T Recommendation and Final draft International Standard of Joint Video specification (ITU-T Rec. H.264/ISO/IEC 14496-10 AVC), Mar. 2003

[3] S. M. Lei and M. T. Sun, "An entropy coding systen for digital HDTV applications," IEEE Transactions on Circuits and Systems for Video Technology, vol. 1, no. 1, pp. 147-155, March 1991.

[4] J. Nikara "Application-Specific Parallel Structures for Discrete Cosine Transform and Variable Length Decoding," Thesis for the degree of Doctor of Technology at Tampere University of Technology, on the 18 th of June 2004, at 12 noon.

[5] M. K. Ruderg and L. Wanhammar, "New Approaches to High Speed Huffman Decoding," IEEE ISCAS vol.2, pp.149-152, Mar 1996.

[6] M. K. Rudberg and L. Wanhammar, "Implementation of a Fast MPEG-2 Compliant Huffman Decoder," EUSIPCO'96, Trieste, Italy, September 1996.

[7] M. K. Ruderg and L. Wanhammar, "High speed pipelined parallel Huffman decoding," IEEE ISCAS vol.3, pp2080-2083, June 1997.

[8] B. J. Shieh, Y. S. Lee and C. Y. Lee, "A High-Throughput Memory-Based VLC Decoder with Codeword Boundary Prediction," IEEE Trans. Issue 8, vol. 10, pp.1514-1521, December 2000.

[9] Y. S. Lee, B. J. Shieh and C. Y. Lee, "A Generalized Prediction Method for Modified Memory-Based High Throughput VLC Decoder Design," IEEE Trans. Issue 6, vol. 46, pp.742-754, June 1999.

[10] B. J. Shieh and C. Y. Lee, “An Efficient VLC Decompression Scheme for User-defined Coding Tables,” IEEE ICASSP, vol. 4, pp.1961-1964, March 1999.

[11] A. Mukherjee, N. Ranganathan, and M. Bassiouni, “Efficient VLSI design for data transformations of tree-based codes,” IEEE Trans. Circuits Sys., vol. 38, pp. 306-314, March 1991.

[12] P. A. Ruetz, P. Tong, D. Luthi, and P. H. Ang, “A video-rate JPEG chip set,” J. VLSI Signal Processing, vol. 5, pp. 141-150, 1993.

[13] H. Park and V. K. Prasanna, “Area efficient VLSI architectures for Huffman coding,” IEEE Trans. Circuits Sys., vol. 40, pp. 568-575, September 1993.

[14] Y. Ooi, A. Taniguchi, and S. Demura, “A 162Mbit/s variable length decoding circuit using an adaptive tree search technique,” in Proc. IEEE 1994 Custom Integrated Circuits Conf., pp107-110, May 1994.

[15] R. Hashemian, “Design and hardware implementation of a memory efficient Huffman decoding,” IEEE Trans. Consumer Electron., vol. 40, pp 345-352, August 1994.

[16] C. –T. Hsieh and S. P. Kim, “A concurrent memory-efficient VLC decoder for MPEG applications,” IEEE Trans. Consumer Electron, vol. 42, pp 439- 446, August 1996.

[17] B. J. Shieh, Y. S. Lee and C. Y. Lee, “A new approach of group-based VLC codec system with full table programmability,” IEEE Trans. Circuits Sys. for Video Technology, vol. 11, pp210-221, February 2001.

[18] 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 Sys. for Video Technology Issue 9, vol.16, pp.1172-1178, September 2006.

[19] Y. H. Moon, G. Y. Kim and J. H. Kim ”An Efficient Decoding of CAVLC in H.264/AVC Video Coding Standard,” IEEE Trans. Issue 3, Vol. 51, pp.933-938, August 2005.

[20] 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,” IEEE Circuits Sys. ISCAS Vol. 6, pp.6110-6113, May 2005.

[21] W. Di, G. Wen, H. Mingzeng and J. Zhenzhou, ”A VLSI Architecture Design of CAVLC Decoder,” IEEE ASIC, Vol. 2, pp.962-965, October 2003.

[22] C. C. Lin, J. W. Chen, H. C. Chang, Y. C. Yang, Y. H. Ou Yang, M. C. Tsai, J. I. Guo and J. S. Wang, ”A 160K Gates/4.5 KB SRAM H.264 Video Decoder for HDTV Applications,” IEEE Journal of Solid-State Circuits. No. 1, Vol. 42, January 2007.

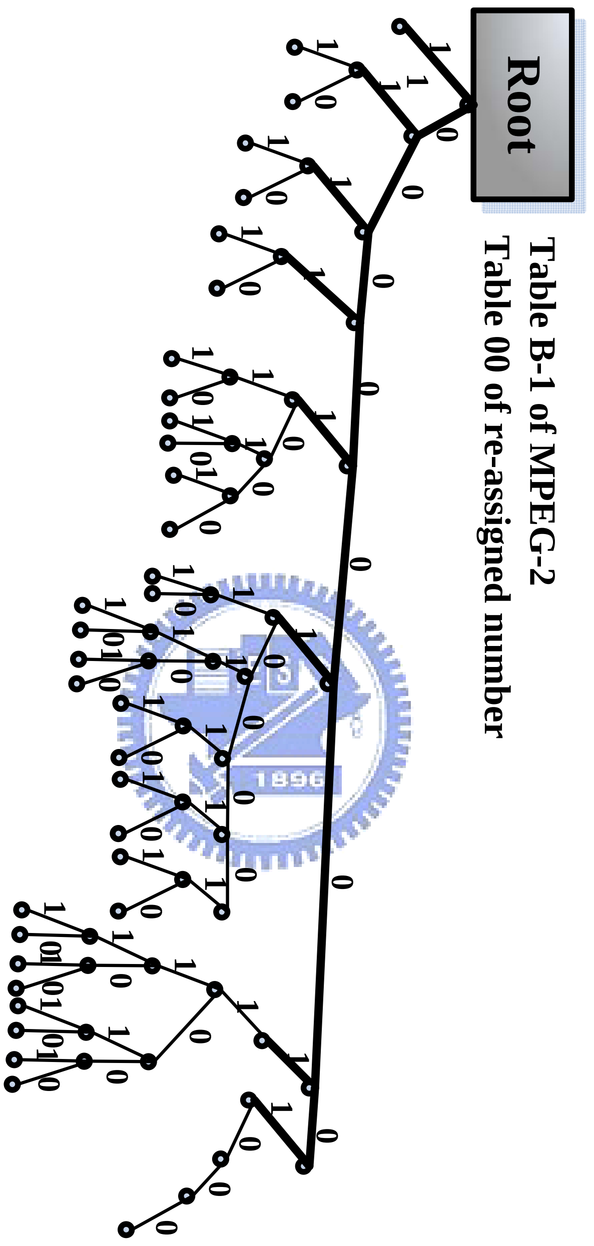
[23] Y. Qu and Y. He, “A Novel Memory Efficient and Cost Effective VLSI Architecture of CAVLC Decoder for H.264/AVC,” PCS 2006

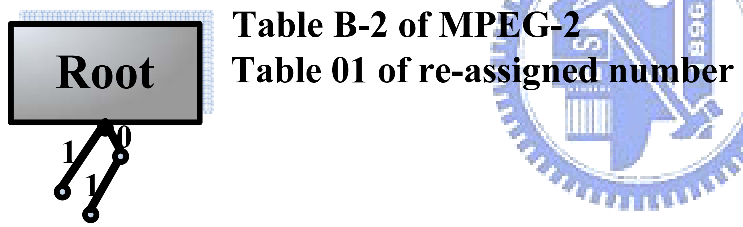
[24] S. M. Sun, T. M. Liu, C. Y. Lee, "Self-Grouping and Table-Merging Algorithm for VLC-Based Video Decoding System" in Proceedings of the 17th VLSI/CAD Symposium, pp. 481-484, August 2006.

[25] S. M. Sun, T. M. Liu, C. Y. Lee, "A Self-Grouping and Table-Merging Algorithm for VLC-Based Video Decoding System" IEEE Asia-Pacific Conference on Circuits and Systems (APCCAS'06), Singapore, December 4-7, 2006.

[26] S. M. Sun, T. M. Liu, C. Y. Lee, "Self-Grouping and Table-Merging Algorithm for VLC-Based Video Decoding System with Full Table Programmability" submitted to International Journal of Electrical Engineering, 2007.

Appendix A　The Huffman Tree of MPEG-2 VLC Tables




Root
1
0
1
Table B-2 of MPEG-2
Table 01 of re-assigned number

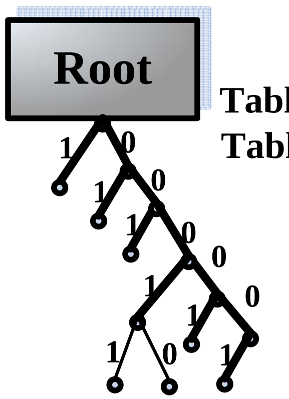


Table B-3 of MPEG-2
Table 02 of re-assigned number

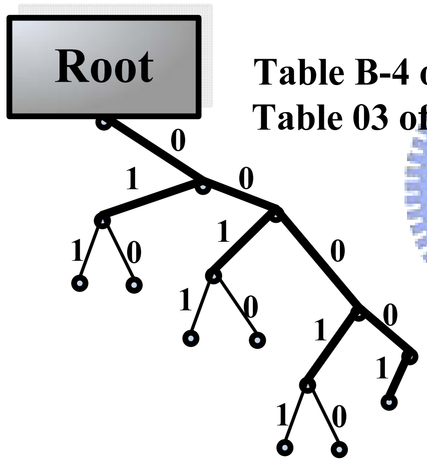


Table B-4 of MPEG-2
Table 03 of re-assigned number

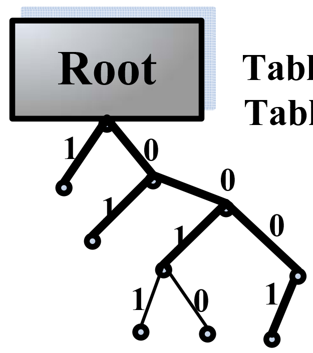


Table B-5 of MPEG-2
Table 50 of re-assigned number for scalable

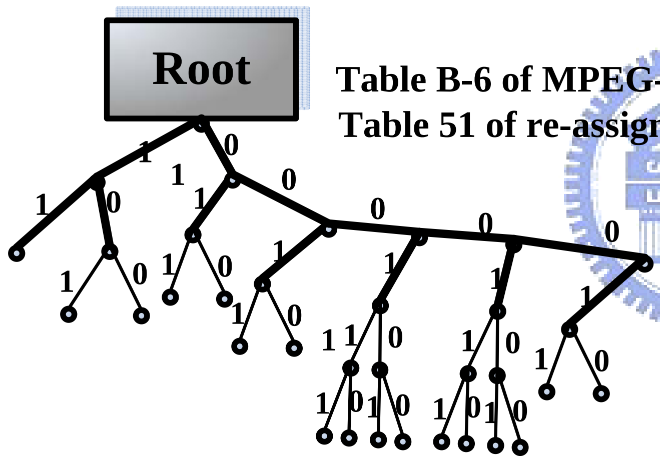
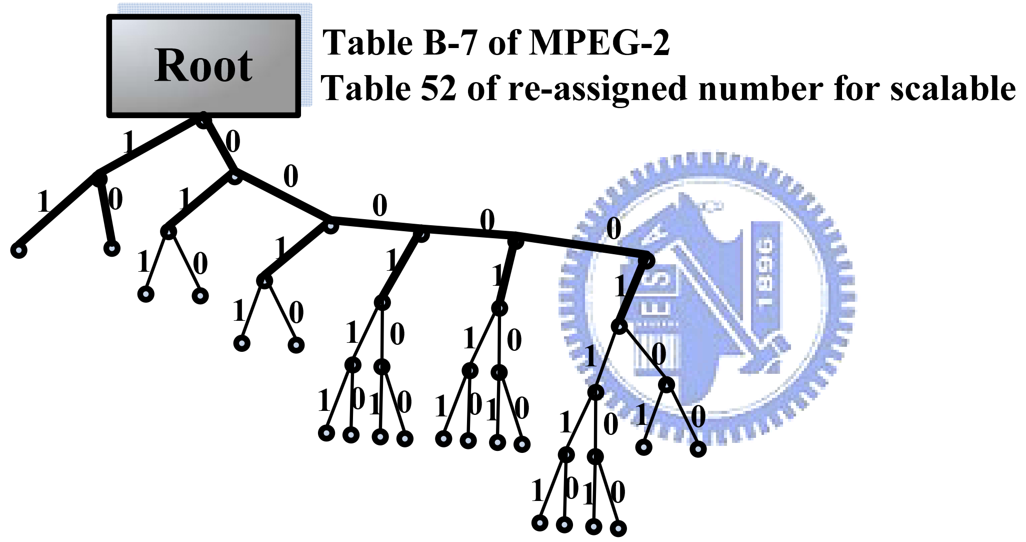


Table B-6 of MPEG-2
Table 51 of re-assigned number for scalable


Root
Table B-7 of MPEG-2
Table 52 of re-assigned number for scalable

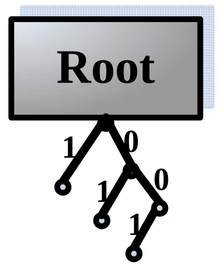


Table B-8 of MPEG-2
Table 53 of re-assigned number for scalable

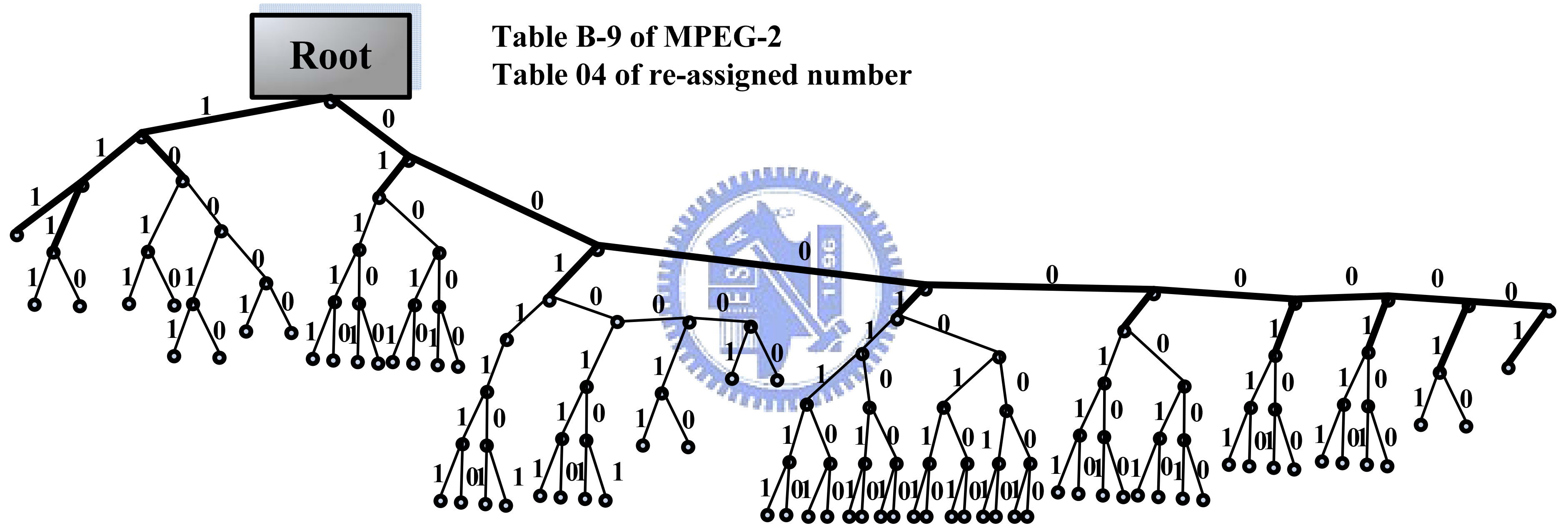
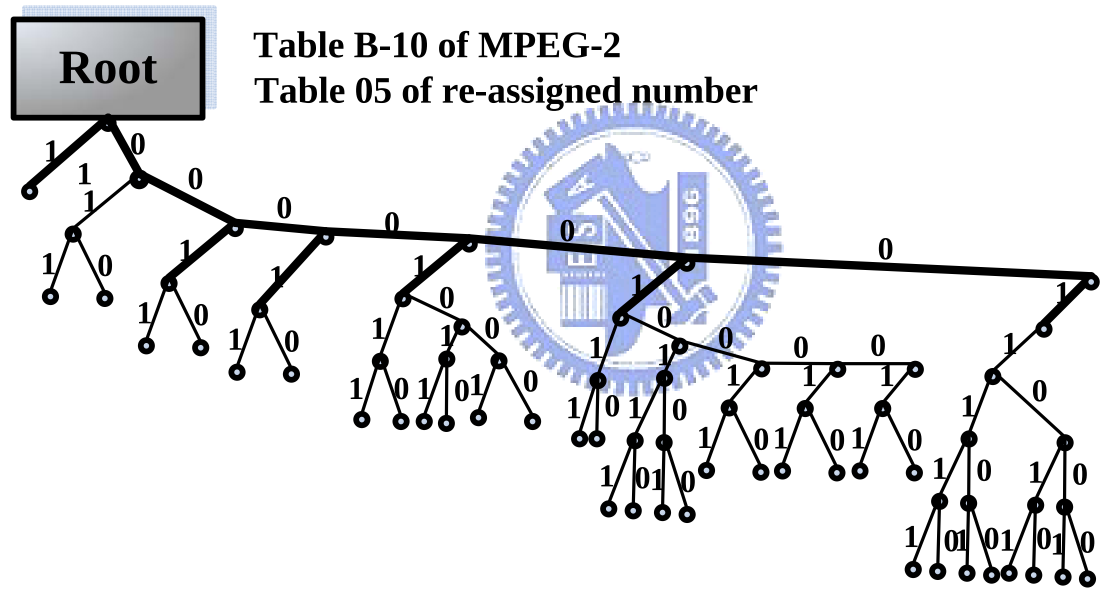


Table B-9 of MPEG-2
Table 04 of re-assigned number

Table B-10 of MPEG-2

Table 05 of re-assigned number



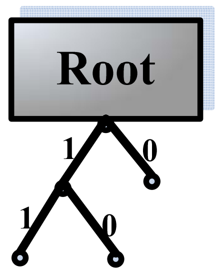


Table B-11 of MPEG-2
Table 06 of re-assigned number

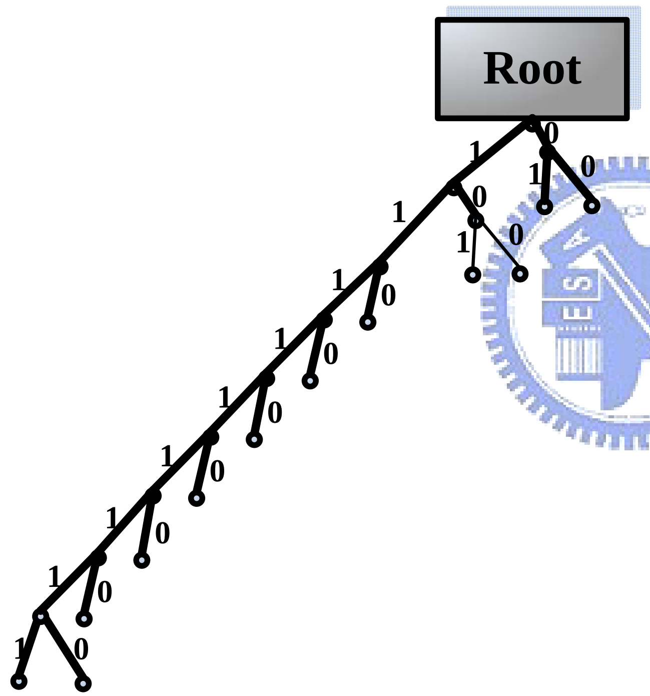


Table B-12 of MPEG-2
Table 07 of re-assigned number

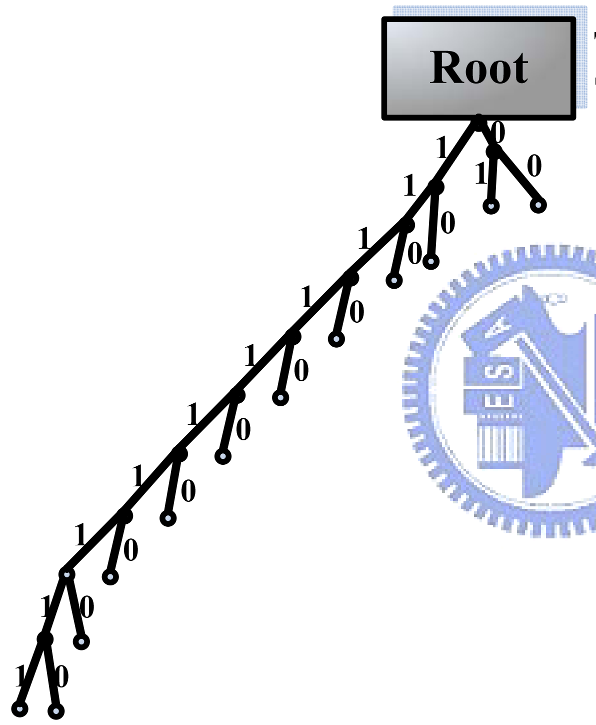


Table B-13 of MPEG-2
Table 08 of re-assigned number

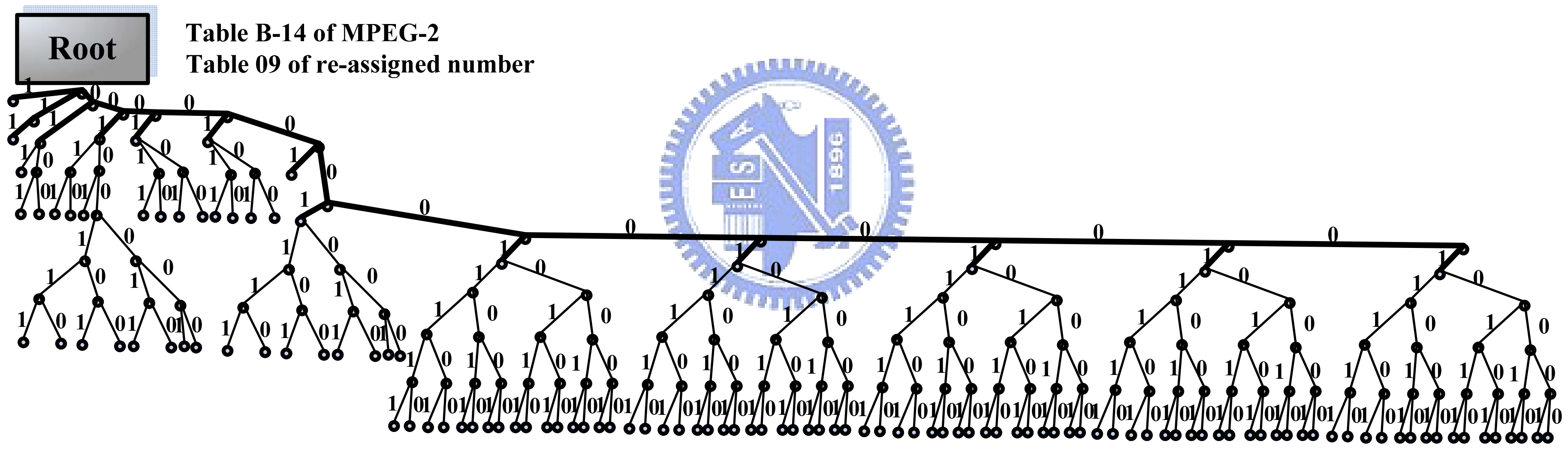
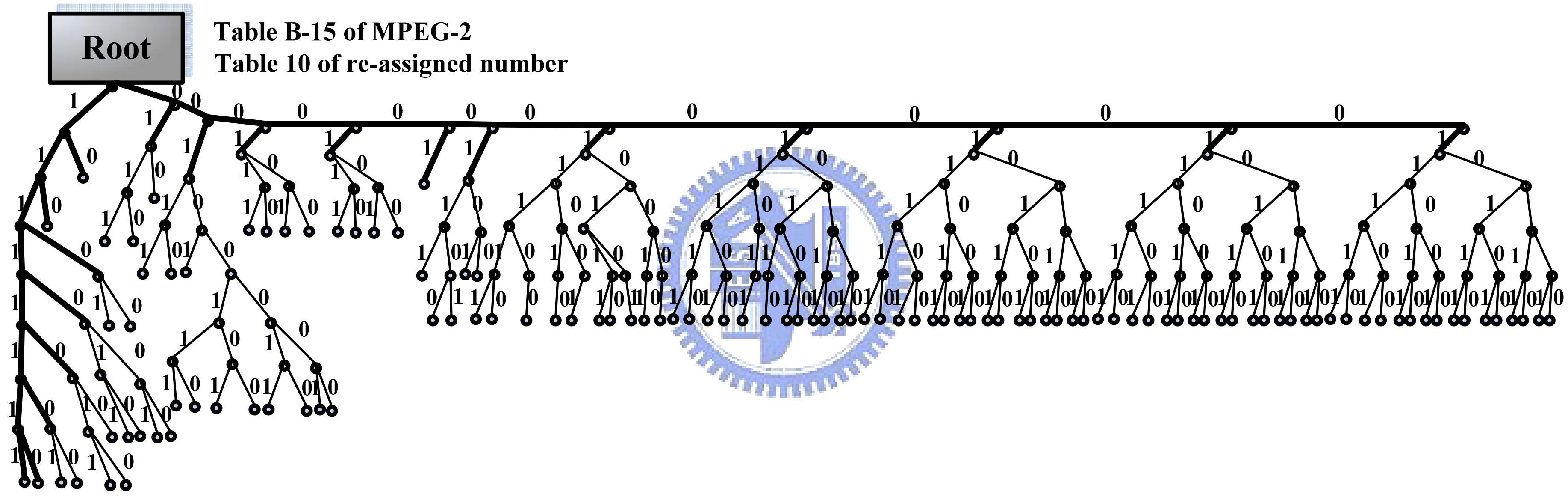


Table B-14 of MPEG-2
Table 09 of re-assigned number

Table B-15 of MPEG-2
Table 10 of re-assigned number



Appendix B The Huffman Tree of H.264 CAVLC Tables

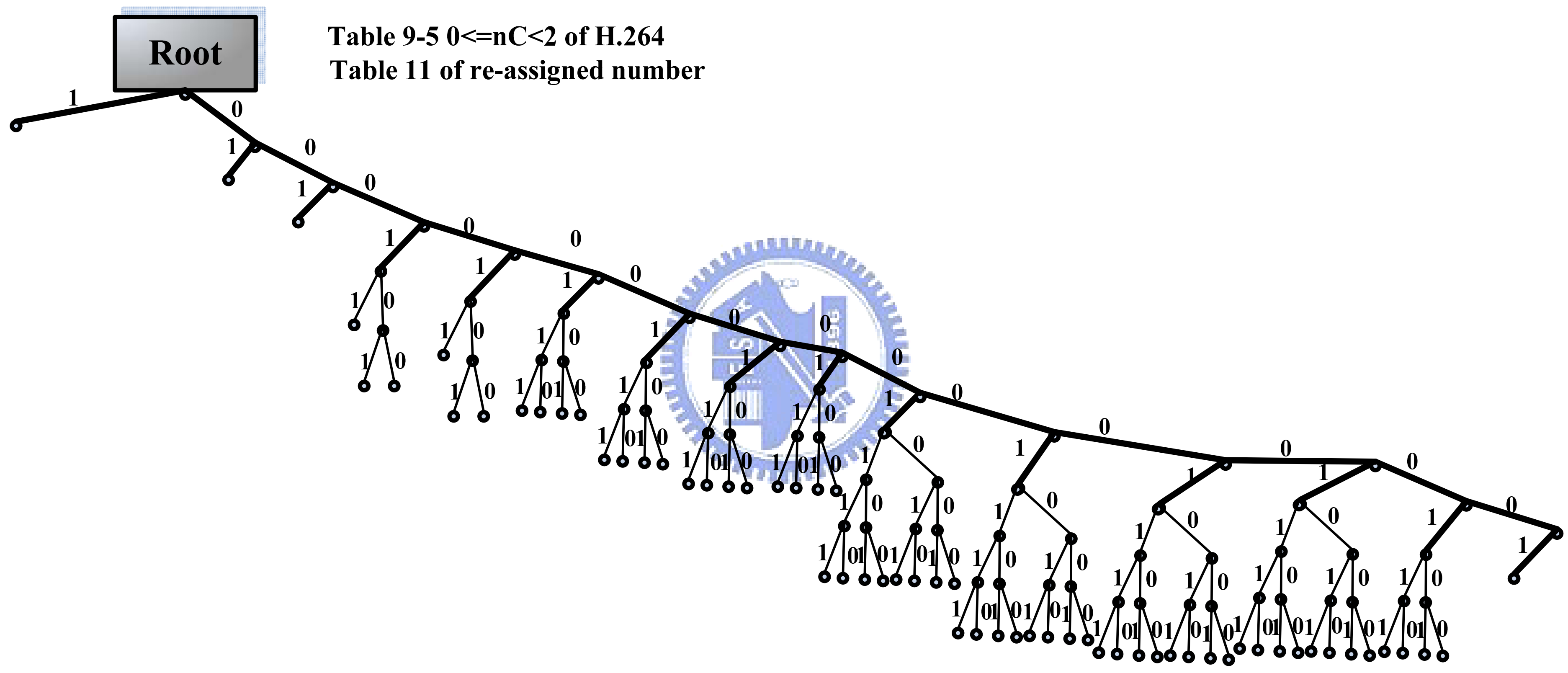


Table 9-5 0<=nC<2 of H.264
Table 11 of re-assigned number

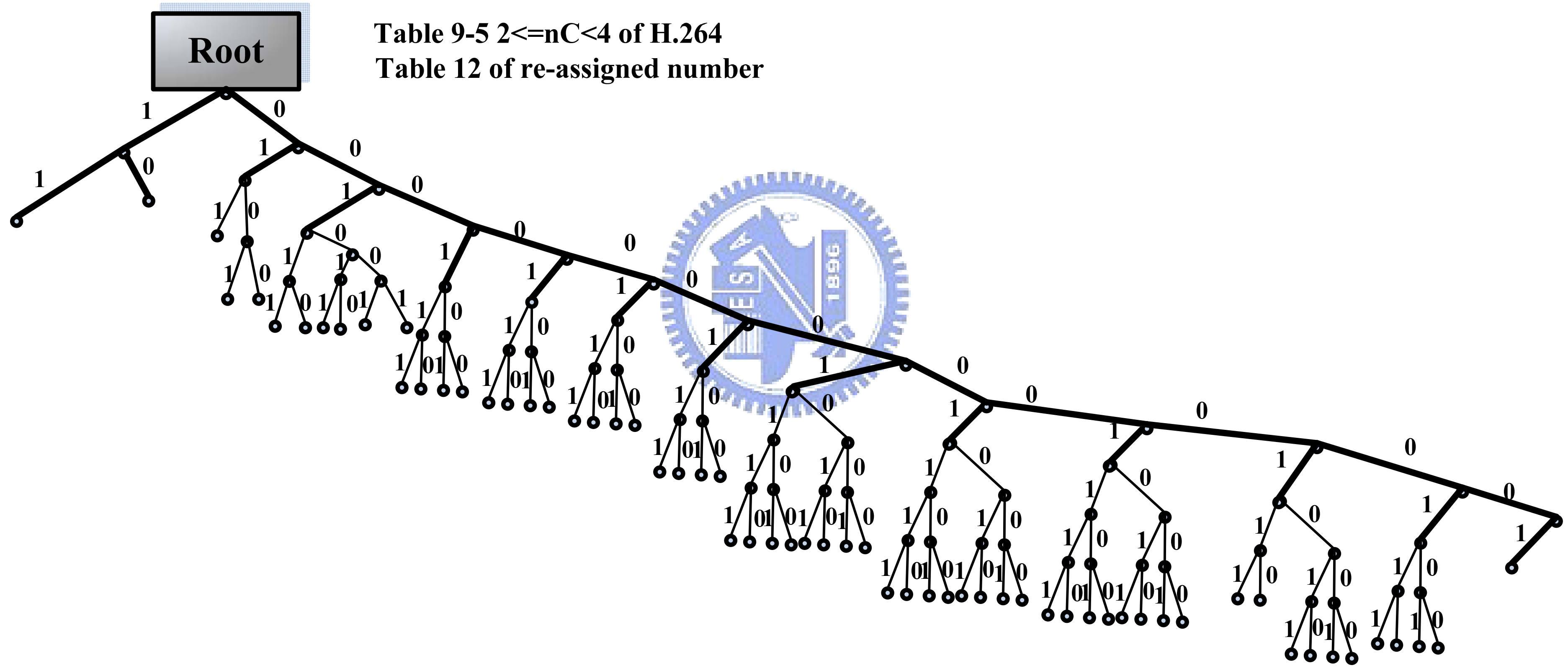


Table 9-5 2<=nC<4 of H.264
Table 12 of re-assigned number

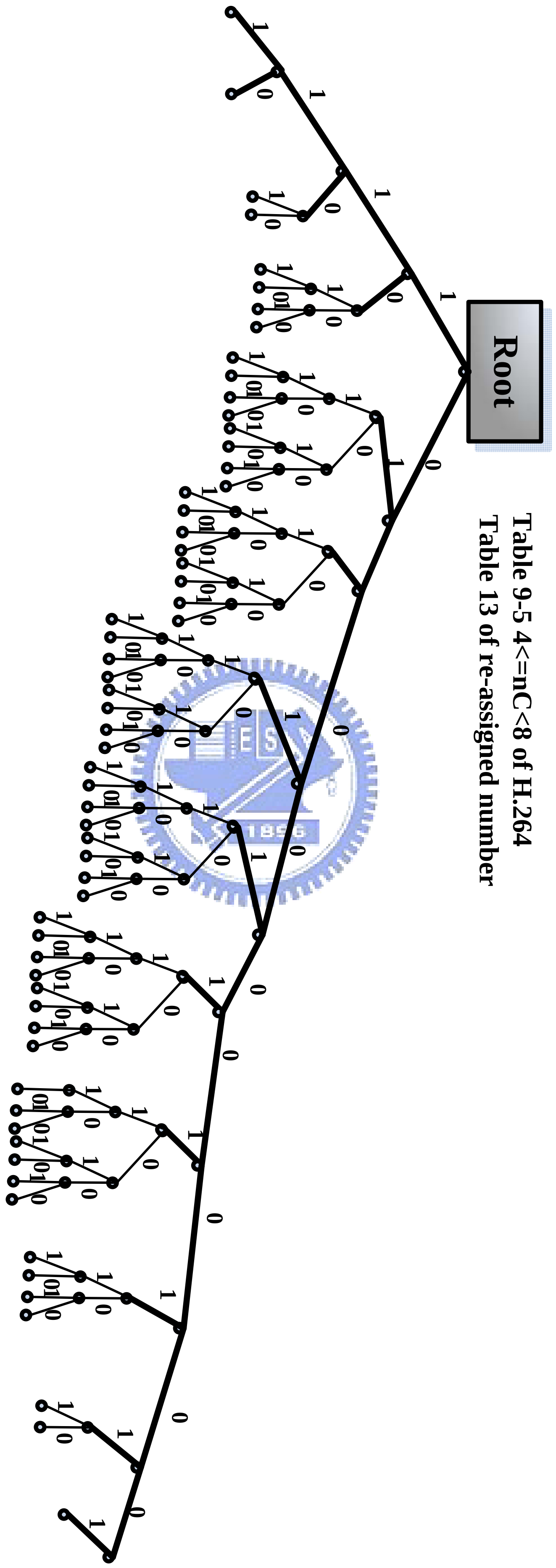


Table 9-5 4<=nC<8 of H.264
Table 13 of re-assigned number

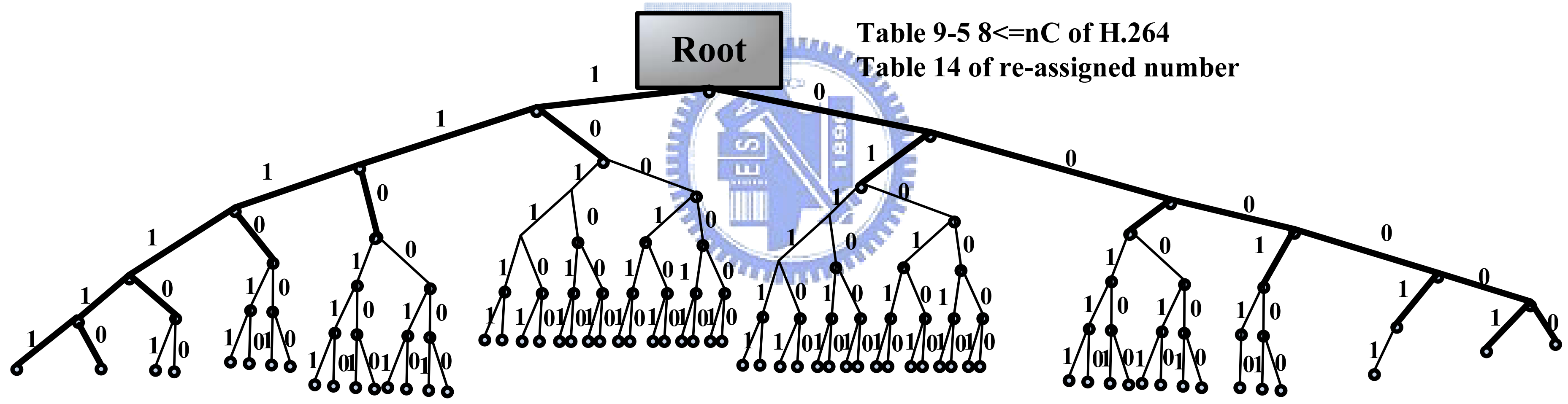


Table 9-5 8<=nC of H.264
Table 14 of re-assigned number

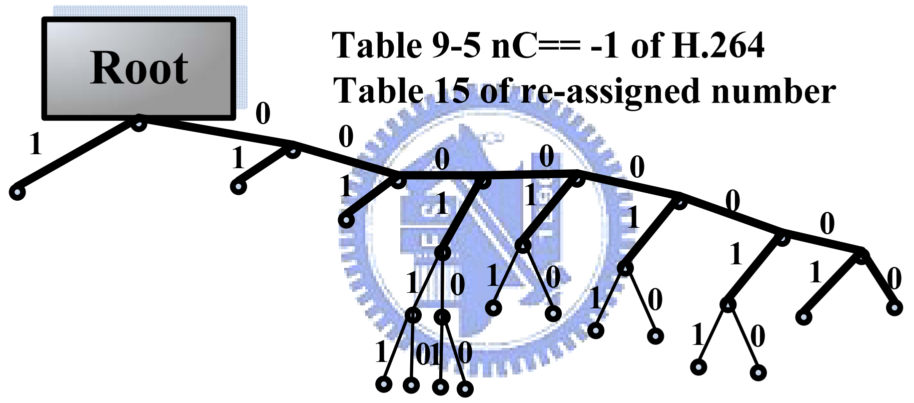


Table 9-5 nC== -1 of H.264
Table 15 of re-assigned number

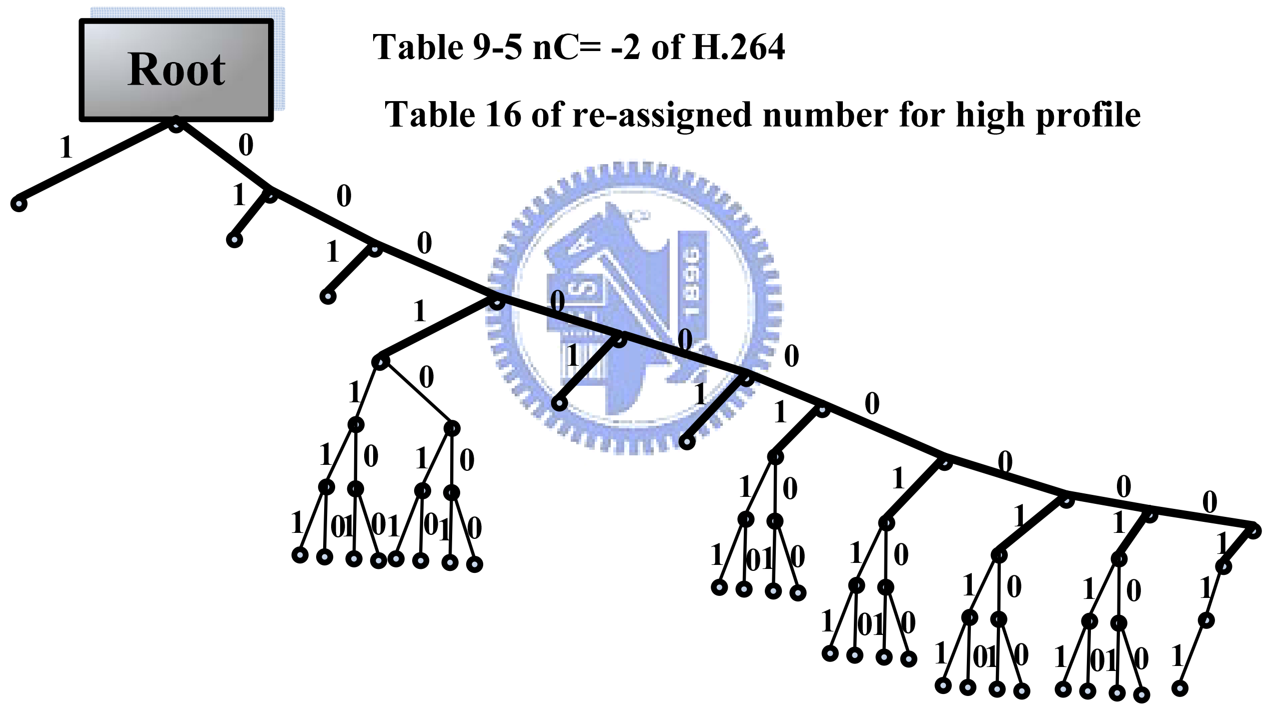
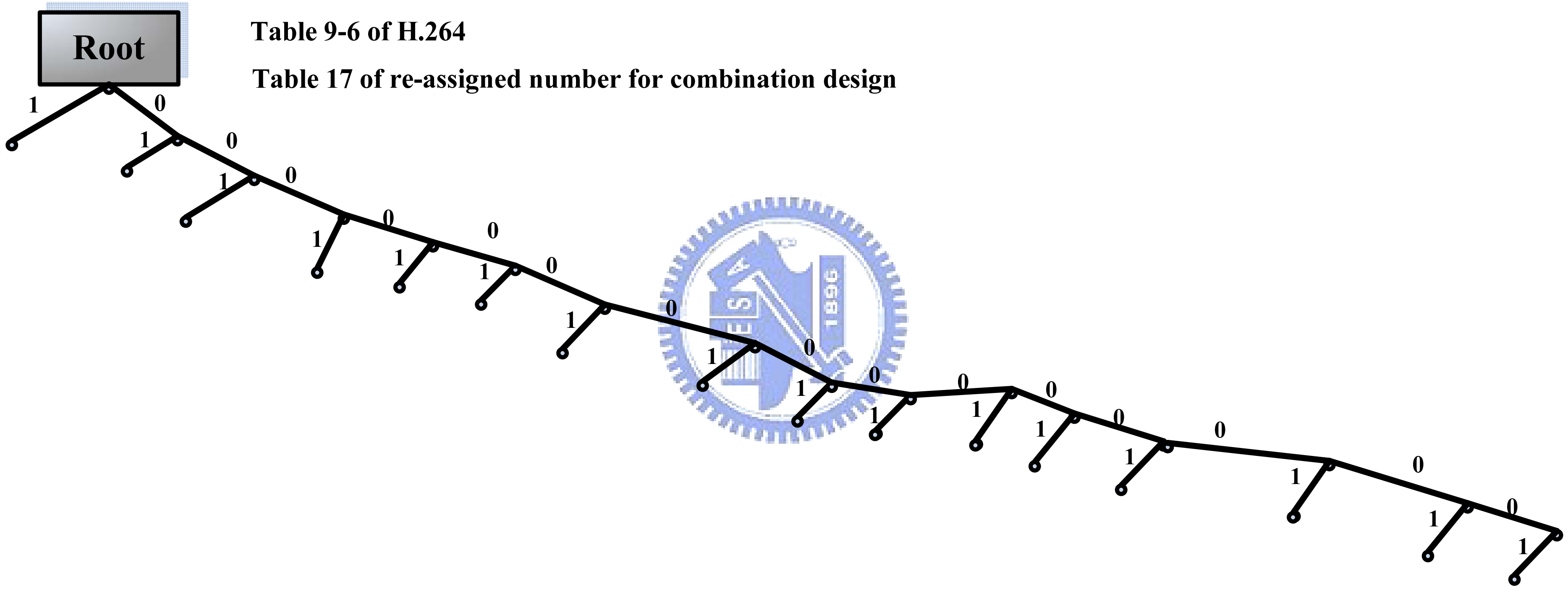


Table 9-5 nC= -2 of H.264

Table 16 of re-assigned number for high profile

Table 9-6 of H.264

Table 17 of re-assigned number for combination design



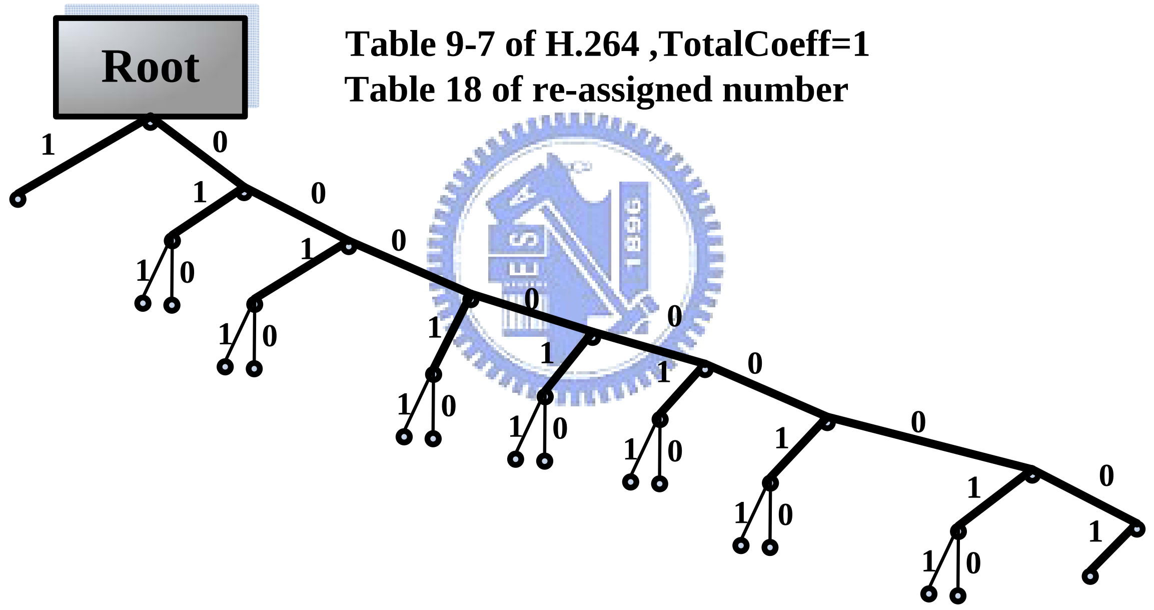


Table 9-7 of H.264 ,TotalCoeff=1
Table 18 of re-assigned number

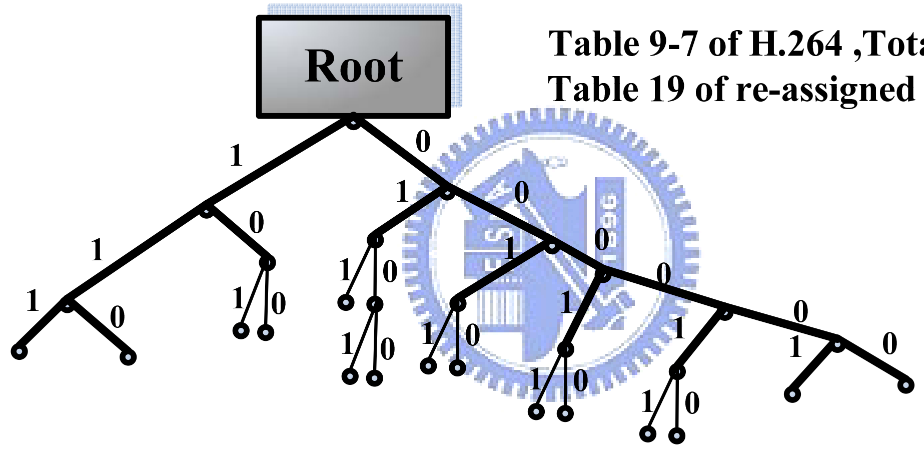


Table 9-7 of H.264 ,TotalCoeff=2
Table 19 of re-assigned number

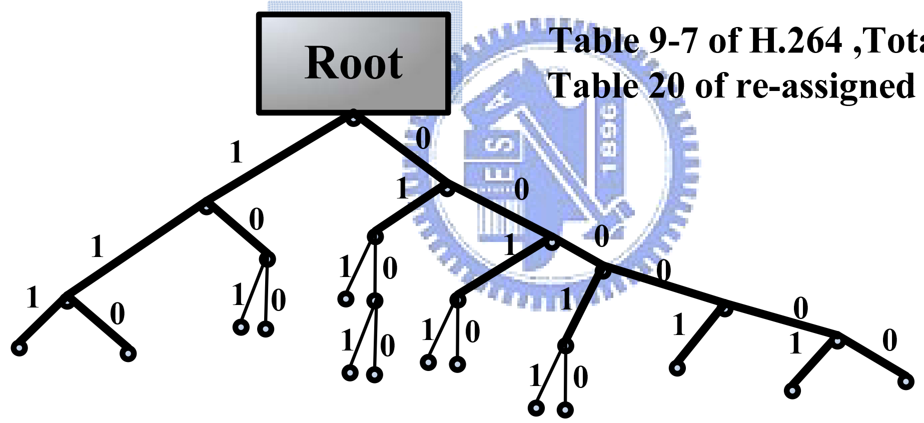


Table 9-7 of H.264 ,TotalCoeff=3
Table 20 of re-assigned number

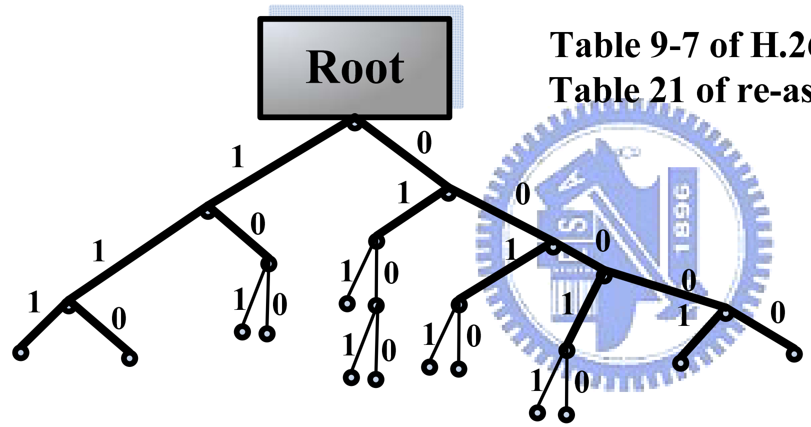


Table 9-7 of H.264 ,TotalCoeff=4
Table 21 of re-assigned number

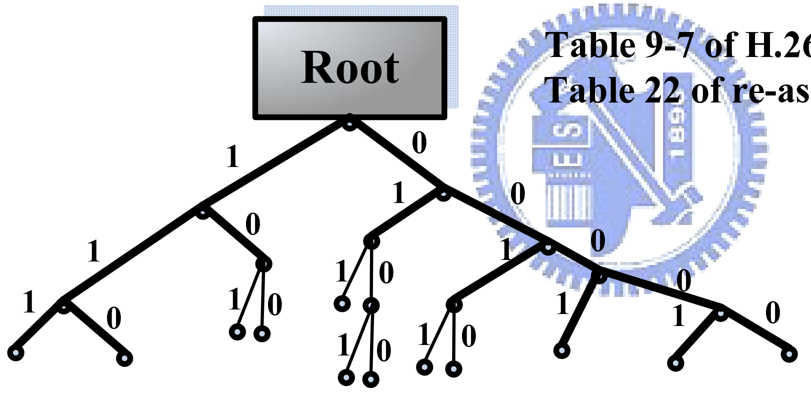


Table 9-7 of H.264 ,TotalCoeff=5
Table 22 of re-assigned number

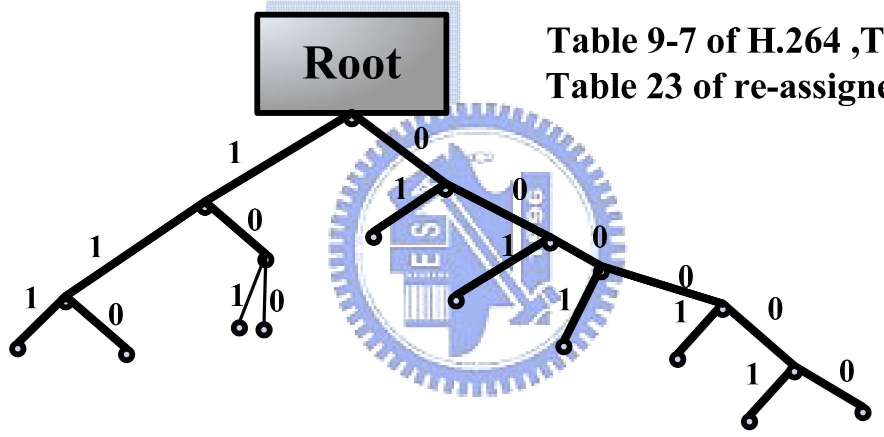


Table 9-7 of H.264 ,TotalCoeff=6
Table 23 of re-assigned number

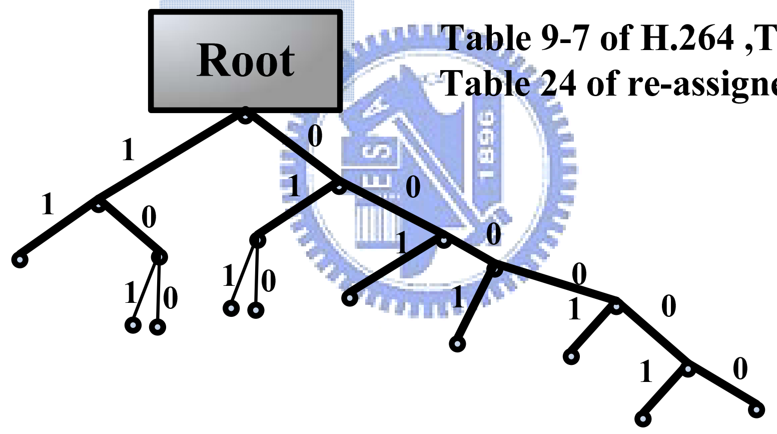


Table 9-7 of H.264 ,TotalCoeff=7
Table 24 of re-assigned number

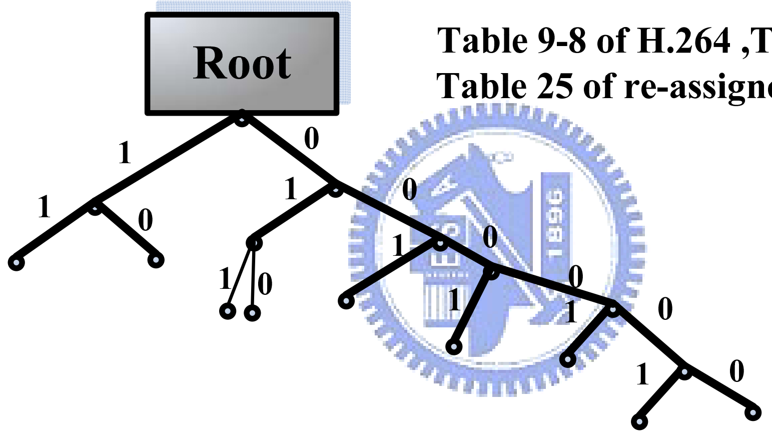


Table 9-8 of H.264 ,TotalCoeff=8
Table 25 of re-assigned number

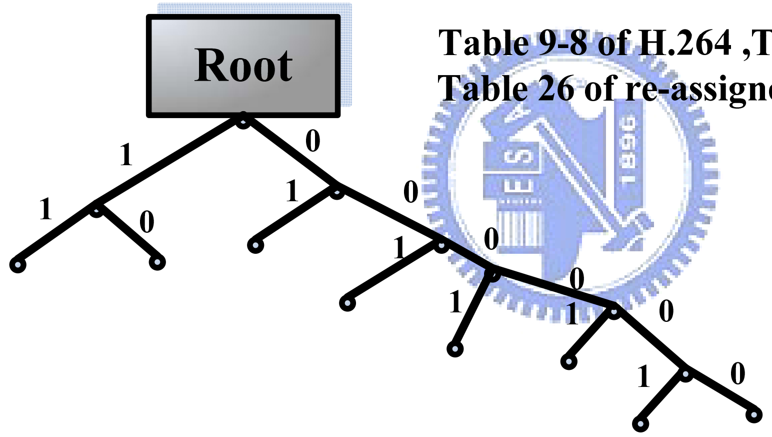


Table 9-8 of H.264 ,TotalCoeff=9
Table 26 of re-assigned number

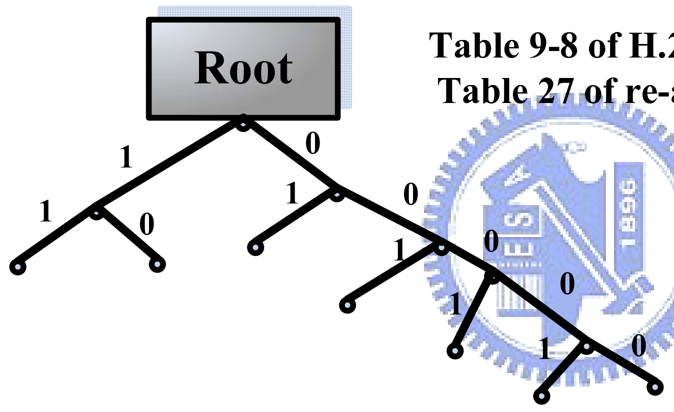


Table 9-8 of H.264 ,TotalCoeff=10
Table 27 of re-assigned number

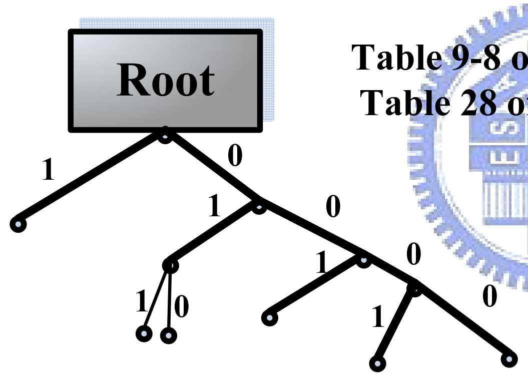


Table 9-8 of H.264 ,TotalCoeff=11
Table 28 of re-assigned number

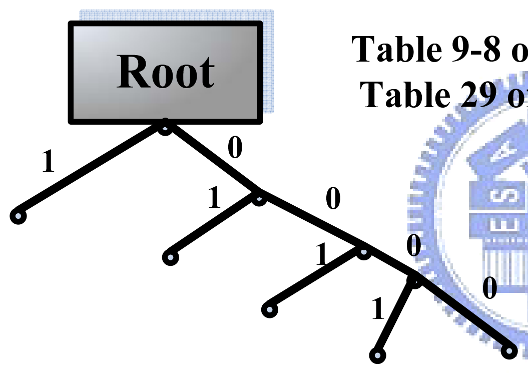


Table 9-8 of H.264 ,TotalCoeff=12
Table 29 of re-assigned number

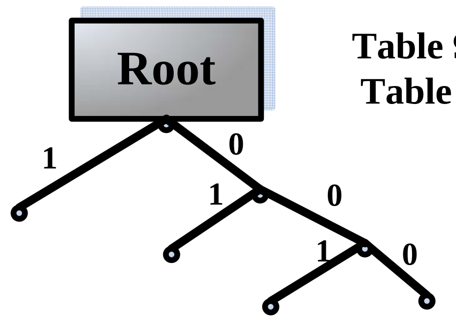


Table 9-8 of H.264 ,TotalCoeff=13
Table 30 of re-assigned number

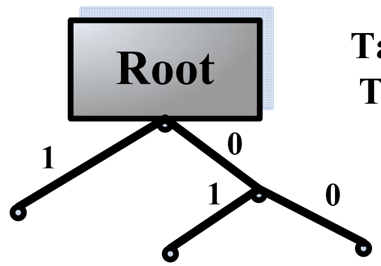


Table 9-8 of H.264 ,TotalCoeff=14
Table 31 of re-assigned number

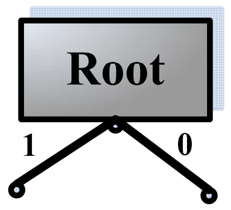


Table 9-8 of H.264 ,TotalCoeff=15
Table 32 of re-assigned number

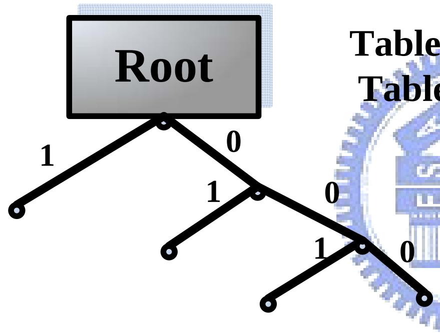


Table 9-9a of H.264 ,TotalCoeff=1
Table 33 of re-assigned number

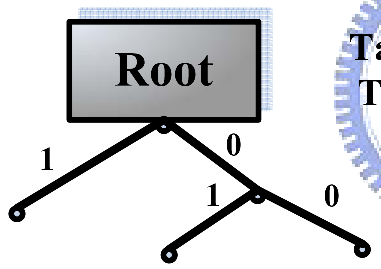


Table 9-9a of H.264 ,TotalCoeff=2
Table 34 of re-assigned number

Table 9-9a of H.264 ,TotalCoeff=3
Table 35 of re-assigned number

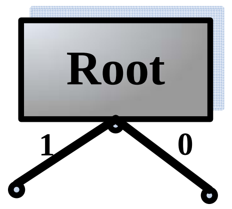
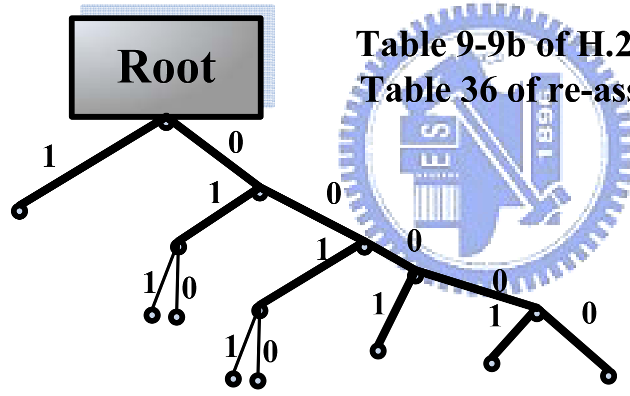


Table 9-9b of H.264 ,TotalCoeff=1

Table 36 of re-assigned number



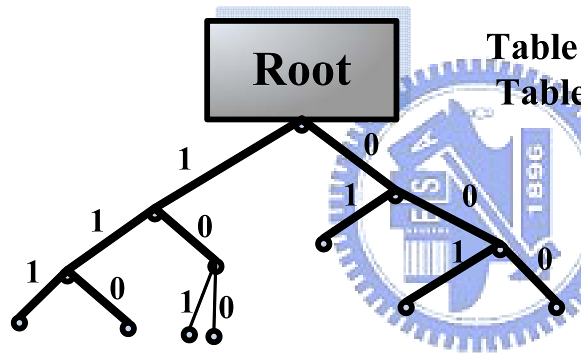


Table 9-9b of H.264 ,TotalCoeff=2
Table 37 of re-assigned number

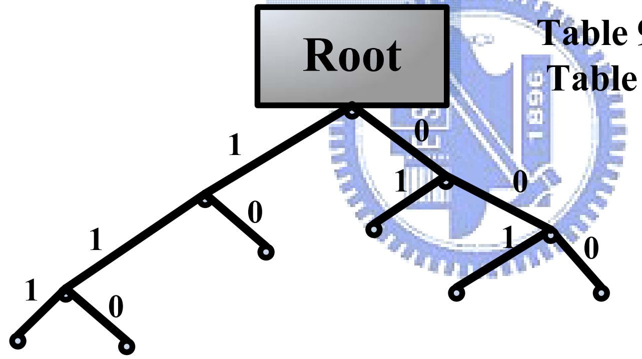


Table 9-9b of H.264 ,TotalCoeff=3
Table 38 of re-assigned number

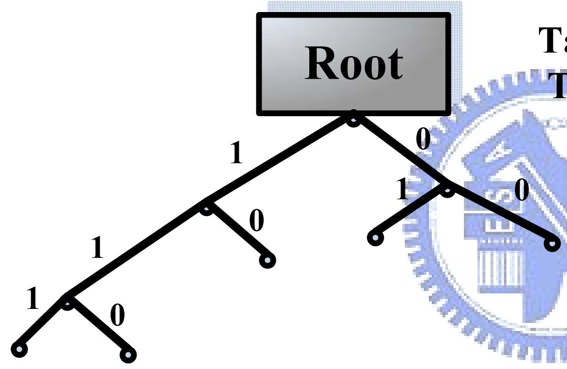


Table 9-9b of H.264 ,TotalCoeff=4
Table 39 of re-assigned number

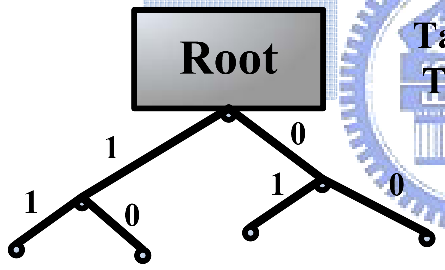


Table 9-9b of H.264 ,TotalCoeff=5
Table 40 of re-assigned number

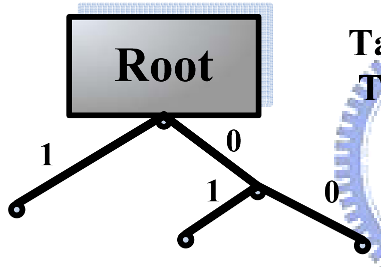


Table 9-9b of H.264 ,TotalCoeff=6
Table 41 of re-assigned number

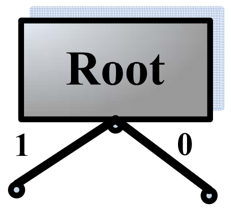


Table 9-9b of H.264 ,TotalCoeff=7
Table 42 of re-assigned number

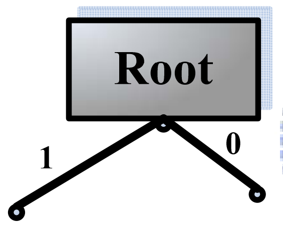


Table 9-10 of H.264 ,zeroLeft=1
Table 43 of re-assigned number

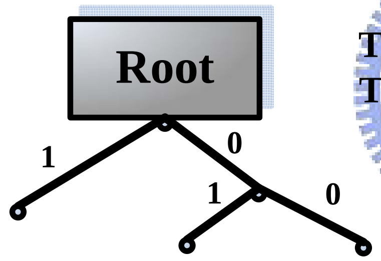


Table 9-10 of H.264 ,zeroLeft=2
Table 44 of re-assigned number

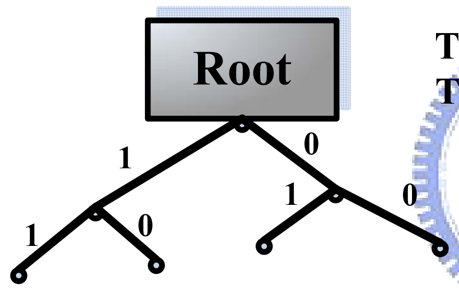


Table 9-10 of H.264 ,zeroLeft=3
Table 45 of re-assigned number

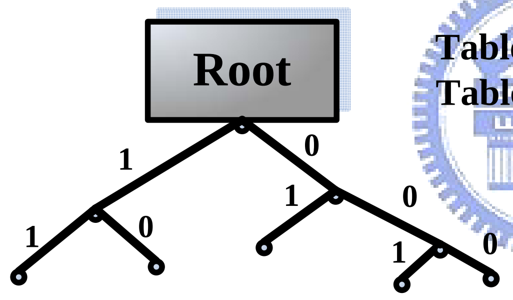


Table 9-10 of H.264 ,zeroLeft=4
Table 46 of re-assigned number

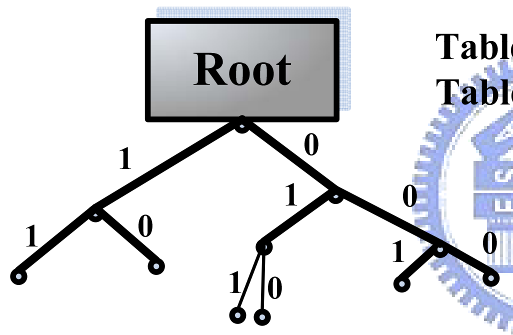


Table 9-10 of H.264 ,zeroLeft=5
Table 47 of re-assigned number

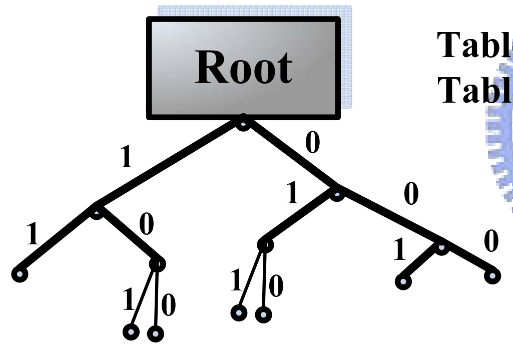


Table 9-10 of H.264 ,zeroLeft=6
Table 48 of re-assigned number

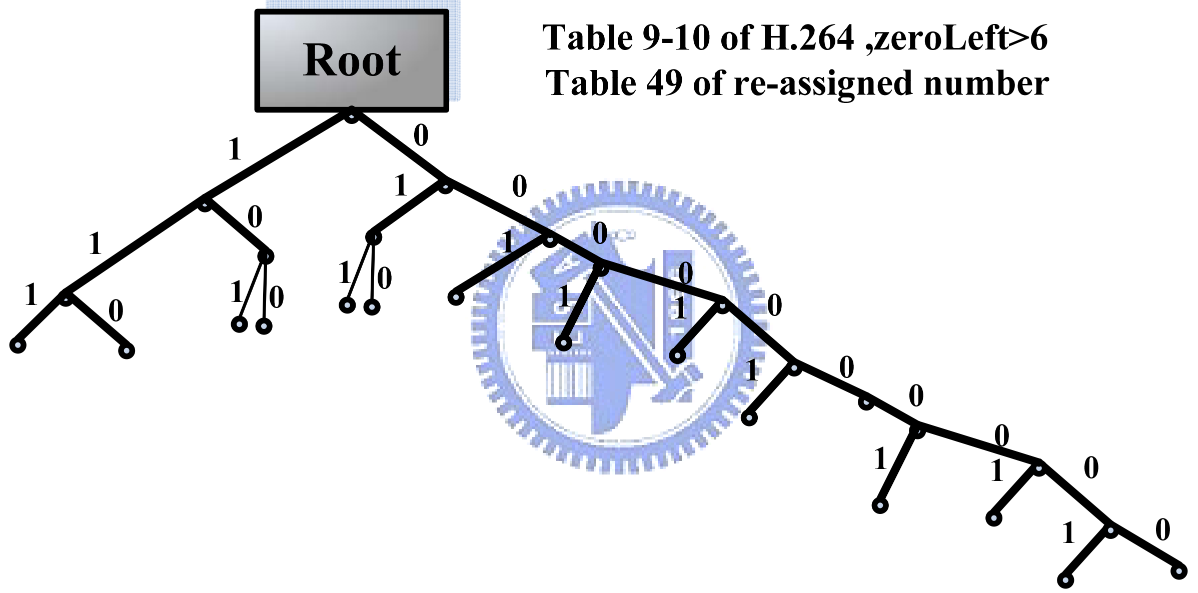


Table 9-10 of H.264 ,zeroLeft>6
Table 49 of re-assigned number

作　者　簡　歷

姓名　　：孫紹銘

戶籍地　：台灣省台中市

出生日期：1979. 12. 31

學歷：

	1995. 9 ~ 1998. 6	台中市 國立台中第二高級中學
	1998. 9 ~ 2002. 6	逢甲大學 電機工程學系
	2002. 9 ~ 2005. 1	國立中興大學 精密工程研究所
	2005. 2 ~ 2007. 1	國立交通大學 IC 設計產業研發碩士班

得　獎/榮　譽

第二學期 凌陽科技產業研發碩士班優秀學生獎學金 第一名

第三學期 凌陽科技產業研發碩士班優秀學生獎學金 第二名

第四學期 凌陽科技產業研發碩士班優秀學生獎學金 入選第三名

International Journal of Electrical Engineering：Invited for a special issue

The 17th VLSI/CAD Symposium：The second paper on signal processing ICs section

IEEE Asia Pacific Conference on Circuit and system：The second paper on video signal processing section

發　表　論　文

國內會議論文：

- **Shao-Ming Sun**, Tsu-Ming Liu, Chen-Yi Lee, **"Self-Grouping and Table-Merging Algorithm for VLC-Based Video Decoding System"** in Proceedings of the 17th VLSI/CAD Symposium, pp. 481-484, August 2006.

國際會議論文：

- **Shao-Ming Sun**, Tsu-Ming Liu, Chen-Yi Lee, **"A Self-Grouping and Table-Merging Algorithm for VLC-Based Video Decoding System"** IEEE Asia-Pacific Conference on Circuits and Systems (APCCAS'06), Singapore, December 4-7, 2006.

國際期刊論文：

- **Shao-Ming Sun**, Tsu-Ming Liu, Chen-Yi Lee, **"Self-Grouping and Table-Merging Algorithm for VLC-Based Video Decoding System with Full Table Programmability"** invited paper, accepted by International Journal of Electrical Engineering, 2007.