國立交通大學

電子工程學系　電子研究所碩士班

碩　士　論　文

應用於行動式視訊裝置之嵌入式壓縮器解壓縮器設計

Design of An Embedded Compressor/Decompressor for Mobile Video Applications

學生　：　吳昱德

指導教授　：　李鎮宜 教授

中華民國九十七年七月


E
S
A
1896

應用於行動式視訊裝置之嵌入式壓縮器解壓縮器設計

Design of An Embedded Compressor/Decompressor for Mobile Video Applications

研 究 生：吳昱德　　Student：Yu-De Wu

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 Institute of Electronics

College of Electrical Engineering and Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Electronics Engineering

July 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年七月


E
S
A
1896

應用於行動式視訊裝置之嵌入式壓縮器解壓縮器設計

學生：吳昱德　　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

本論文提出適合嵌入於行動式視訊裝置上的有失真嵌入式壓縮器/解壓縮器設計。藉由有失真資料壓縮來減少晶片與外部記憶體間所需要的資料傳輸量，損失些微的視訊品質，來達到縮小外部記憶體空間需求、減少頻寬使用以及降低能量消耗等多種目的。

所提出的演算法是以二維離散餘弦轉換搭配簡約位元平面區域編碼所構成。在壓縮率為二的前提之下，將一個四乘以四的像素矩陣壓縮為六十四位元的壓縮封包。首先將四乘以四像素矩陣以二維離散餘弦轉換為十六個不同頻率之係數分量，再使用簡約位元平面區域編碼將係數予以編碼封存後送到外部記憶體。解壓縮過程中並提出一個簡單的補償方式來彌補失真壓縮所造成的資料遺失。

所提出的硬體架構可以嵌入在視訊解碼器上以 100MHz 的操作頻率支援每秒三十張的高畫質電視規格(HD1080)。由於將壓縮率固定為兩倍，所以壓縮後的封包大小固定，記憶體地址轉換十分簡單並且可以支援動作補償單元 (Motion Compensation)的亂數存取。在 UMC 90 奈米製程技術下，所提出的硬體使用了

30k 個邏輯閘數目。壓縮一個巨型區塊(MB)需要 72 個週期，解壓縮一個巨型區塊(MB)則僅需要 34 個週期。整體系統對於記憶體的存取次數則節省了原本的百分之四十。相較於所消耗的功率，節省的功率相當的可觀。

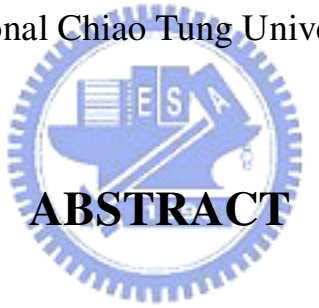
Design of An Embedded Compressor/Decompressor for Mobile Video Applications

Student : Yu-De Wu Advisor : Dr. Chen-Yi Lee

Department of Electronics Engineering

Institute of Electronics

National Chiao Tung University



ABSTRACT

This thesis proposes an embedded compressor/decompressor for mobile video applications. It uses lossy compression scheme to reduce the amount of data transferring between chip and external memory. This lossy compression can maintain acceptable video quality while reduces the required size of external memory, the bandwidth requirement and the power consumption on memory access.

Proposed algorithm is composed by discrete cosine transform (DCT) with coarse grain bit-plane zonal coding (CGBPZ). The compression ratio is two. It compresses a 4x4 pixel-array into a 64 bits segment. First, the two dimensions discrete cosine transform converts 16 pixels into 16 elementary frequency components. Coarse grain bit-plane zonal coding packets the coefficients and then sends to external memory. A

compensation scheme is also proposed for decoding.

Hardware architecture of the proposed algorithm is able to be embedded into video decoder and support HD1080@100MHz, 30 frames per second. Since the compression ratio is fixed at two, the coded segments have fixed size and can be randomly accessed by motion compensation unit. The gate counts are 30K synthesized by UMC 90 nm CMOS technology. It costs 72 cycles to encode a MB and 34 cycles to decode a MB. Overall reduction ratio on memory access is 40%. Comparing with the power consumed of proposed design, the amount of power saving is large.

誌　　謝

在 SI2 Lab 的兩年裡，是我人生中珍貴的日子。首先，我要對我的指導教授李鎮宜博士表達我最深的感謝。老師總是熱心且耐心的指導我，並適時的給予鼓勵，使我在碩班兩年裡獲益良多。在此誠摯的給予老師最深的祝福。

其次我要感謝的是 multimedia group 的博班學長，劉子明和李曜。學長們對我的訓練和不厭其煩的指導，為我的研究打下了扎實的基礎。阿龍、義閔學長在專業的領域上也給予我很多幫助。特別要感謝的是蔣迪豪教授和鍾菁哲博士，除了研究上犀利的建議之外，也給我很多就業與職場上的詳盡分析。同屆的夥伴們，韋磬、Amos、bluer、俊廷、琇茹、清峰、建螢、點子、茗智、學弟昱帆和其他每一位 SI2 成員們，有你們一起思考、互相幫忙、適時搞笑，讓研究生活充實而不枯燥。此外，我也要感謝我的室友們，宗學、良諺、碩宇、振祐、育瑋、文炫，讓我的宿舍生活充滿了歡樂。

最後，我要感謝我的家人和我的朋友們，因為有你們的支持、付出與鼓勵，我才能全心全意的向前邁進。願你們永遠健康、快樂、順心。

Index

Figure Index

Table Index

Chapter 1
Introduction

1.1 Motivation

To improve the video coding efficiency, eliminating temporal redundancy between frames is a useful technique. This technique is widely used in nowadays video coding standards such as MPEG-1/2/4, H.263 and H.264. But to accomplish this method when encoding or decoding, at least one previous frame must be stored in frame memory as reference. However, the accesses between external memory and decoder chip may consume a lot of power. The rapid data accesses of motion compensation dominate the power consumption of whole system.

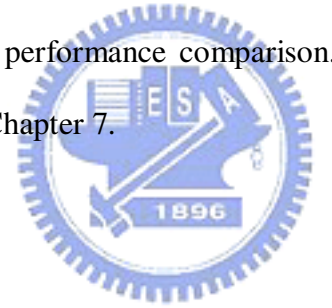
For a mobile device, power is always the critical issue that people do care about. Although the power consumed on chip can be reduced by many low power techniques, data transferring still consumes a lot of power. Therefore, minimization of memory access operations is a key consideration in hardware design of mobile video devices.

Embedded compression is a technique to reduce the transferring of data and the size of off-chip frame memory. Since mobile video devices are suffered from limited battery life and the visual quality criterion is not so strict due to the small display screen, we hope to reduce the bandwidth requirement while maintain the acceptable visual quality.

Nowadays, the mobile devices become more and more powerful by their various functions. Reduce the bandwidth and resource requirement of each hardware accelerator is definitely an important topic.

1.2Thesis Organization

This thesis is organized as follows. First, the basic introduction of compression scheme and the reviews of prior works are described in Chapter 2. The proposed lossy embedded compression algorithm is proposed in Chapter 3. To integrate with H.264/AVC decoder, there are some constraints needed to be specified and the proposed algorithm must be modified to fit in those constraints in hardware design. The modified algorithm and hardware architecture is presented in Chapter 4. Moreover, the simulation results about proposed algorithm integrated with H.264/AVC HDTV decoder are also presented in this chapter. The design implementation, integration and verification are shown in Chapter 5. Chapter 6 shows the experimental results and performance comparison. Finally, the conclusions and future work will be given in Chapter 7.



Chapter 2
Previous Works

Basically, compression techniques can be divided into two types: lossless compression and lossy compression. In this chapter, we will simply introduce the algorithms that have been proposed before. Also, the bit-plane coding is introduced in chapter 2.3. Bit-plane coding can be used as lossy or lossless coding. The concept of bit plane coding is used in our proposed methods.

2.1 Lossless Embedded Compression Schemes

A lot of lossless compression methods have been proposed. The benefit of lossless compression is obviously: it can maintain the information while cutting down the data size. To embed a lossless compression mechanism into a video system is quite acceptable, since it would not cause the drifting effect no matter in encoder system or in decoder system.

However, behind those advantages mentioned above, it does suffer from the variable data amount after lossless compression. By mathematical theory, even for ideal lossless compression, the information of source data still controls the compression ratio. That means, the more the information of the source data contained, the longer the coded data is. This unstable factor becomes the fatal wound of lossless embedded compressions. Embedded compression schemes are born to reduce the data access times between the external memories and reduce the size of external memory. However, Variable data amount after lossless compression can not guarantee the reduction ratio nether on the size of frame memory since the memory must be well

prepared for the worst case nor the bandwidth reduction since the compressed data amount is unknown. A research of lossless compression is shown in [2].

2.2 Lossy Embedded Compression Scheme

Lossy compressions with fixed compression ratio are suitable to reduce the size of frame memory and the bandwidth since the predictable amount of compressed data can guarantee the reduction. Therefore, lossy embedded compressions are more popular in comparing with lossless embedded compressions on solving this bandwidth reduction problem. [3] – [14] are the previous works of lossy compression.

2.2.1 Transform-Based Lossy Embedded Compression

Transform-based lossy embedded compression is a popular way to compose lossy compressions. It converts a signal into elementary frequency components. With the characteristic of human visual system, lower frequency component is more noticeable than higher frequency component. Thus implying quantization and data collection on each component by their visual priority could be an efficient way to collect data within limited data budget. The research uses the Hadamard Transform and quantizes the coefficients by their priority, and then encodes quantized coefficients by Golomb-Rice Coding is in [3]. Golomb-Rice coding is an efficient coding method, and it can nearly reach the coding ability of Huffman coding by selecting the suitable K factor. However in this paper, it pursuit low complexity, therefore it chose fixed K values according to simulation. It can operate on 100 MHz and the cycle usages of encoding/decoding a MB are both 33 cycles. It is a work of high speed.

2.2.2 Delta Pulse Code Modulation Lossy Embedded Compression

Delta Pulse Code Modulation (DPCM) is another popular way to compose the lossy compression. Since the neighbor data has relatively small difference, the information of data after DPCM can be efficiently reduced by comparing with the source data. It does help on reduction of source information.

[4] uses DPCM as base coding method and takes the intra prediction mode from H.264 video coding standard to find the best direction to perform DPCM. This smart idea makes this algorithm more adaptive in each video pattern and achieves the satisfied quality than [3].

However, the satisfied performance of DPCM method costs a lot. DPCM method needs to collect every difference into limited budget, but those differences are not always as small as we wish. To derive best quantization level and fit every difference into limited budget, this DPCM-based method needs several iterations to get the best performance. This situation causes this algorithm not to be able to use pipeline scheme. And to avoid large gate counts, it is more acceptable to deal with subtractions clock by clock instead of parallel architecture. However it leads to longer coding cycles and becomes a heavy load of original system on timing issue. In the view point of system integration, it needs to increase the operation frequency or slow down the system throughputs to perform this DPCM-based embedded compression scheme.

2.2.3 Other Embedded Lossy Compression

There are still many approaches about lossy compression such as adaptive vector quantize (VQ)[11], down-sampling based compression algorithm [12] and adaptive DPCM in [13]. [15] provides two compression schemes and uses a pre-determining

mechanism to choose with methods to use. It claims that this mechanism can achieve better performance by choosing adaptive algorithm to fit the different feature of video sequence. DWT with SPIHT in [14] is also another transform approach. And the algorithm used in [14] makes it be able to perform lossy and lossless with the same architecture.

We can see that lossy embedded compression scheme is truly the mainstream. However it suffers from the loss of quality and the drift effect. Therefore, how to organize the lossy coding methods is very important. To cover information as much as possible within limited budget is the main challenge of lossy compression.

2.3 Bit-Plane Coding

Bit-plane zonal coding is a well known coding method and widely used in many compression algorithms. It uses bit-plane as its basic unit to encode a group of number instead of individual number. It can be combined into a lossy or lossless compression scheme by adjusting the budget of bit storage. It can fully represent the group of number with sufficient bit budget. On the other hand, with un-sufficient budget it may loss some information at lower bits and thus becomes a lossy compression. The details of bit-plane zonal coding will be shown in the following sections.

2.3.1 Bit-Plane Truncation Coding (BPT)

Before introducing proposed bit-plane zonal coding, we would like to introduce the basic concept first. Bit-plane truncation coding is the prototype of bit-plane zonal coding. It can be shown in Fig. 1 as an example. Fig. 1 is the coefficients after 4x4

DCT. We can simply classify those coefficients into one DC coefficient and 15 AC coefficients. The idea of bit-plane coding is to collect data in bit-plane (that is, to take the N-th bit out of each coefficients as a union) rather in individual coefficient. Sometimes, we want to further analyze a group of numbers and to cut them into several parts by their importance, separating them into bit-planes is a good idea. Moreover, for a group of coefficients, the upper bit-planes are zero most of the time. Therefore to record start plane is the smart way to improve the coding efficiency. For a group of 4x4, N bits coefficient, about cell function ($\log_2$ N) bits is needed for recording start plane, but it can represent 15 zero bits for each skipped bit-plane. After the bit-plane truncation coding, the coded format is shown in Fig. 2.

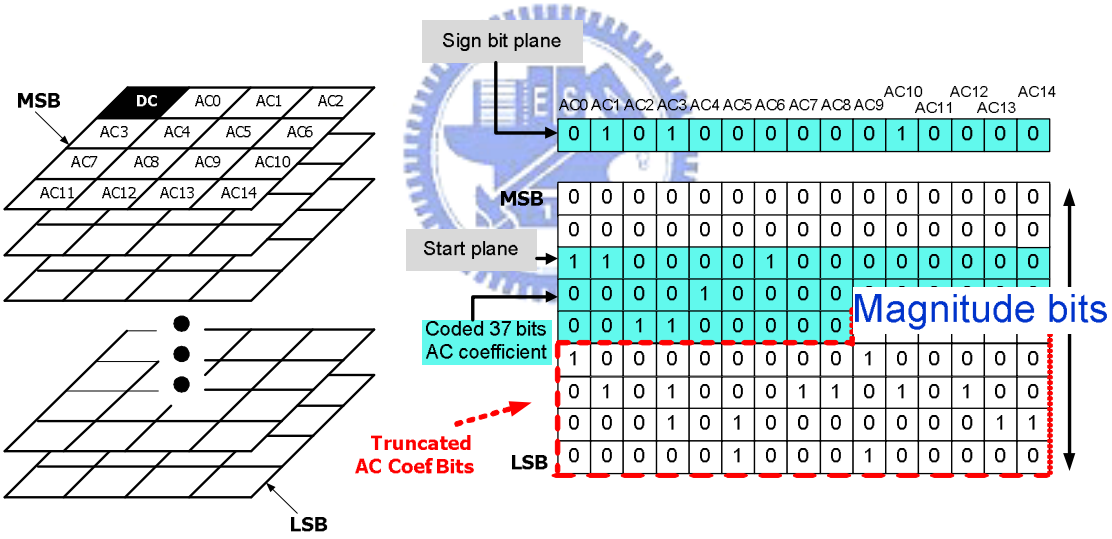


Fig. 1 Bit-plane truncation: AC coefficients are packed from the start plane. Due to the limitation of packing budget, coefficient bits of lower digit plane surrounded by dash line will be truncated.

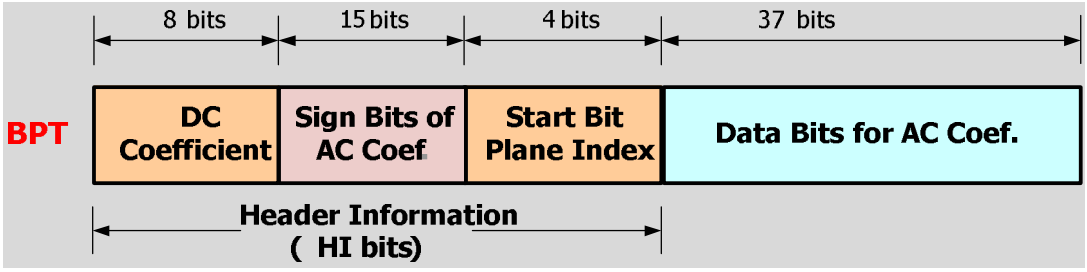


Fig. 2 Coding format for bit-plane truncation coding (BPT).

2.3.2 Bit-Plane Zonal Coding (BPZ)

However, BPT has poor performance and image quality must be enhanced by other approach to reduce energy loss of DCT coefficients. In this section, an improved coding algorithm named bit-plane zonal coding (BPZ) [18] will be described in detail. Familiar with BPT, BPZ packets DCT coefficients bit-plane by bit-plane, but the packing scheme is quite different from BPT. We will show that the packing efficiency of BTZ is much better than BPT.

The word “zonal” is the idea to encode a bit-plane with its zonal characteristic. Fig. 3 is a possible outcome of a bit-plane. The coefficients with larger magnitude tend to be gathered at up-left corner (lower horizontal or vertical frequencies) by DCT. Also, the bits at down-right corner tend to be zero in the same bit-plane. Furthermore, the data for individual DCT blocks often has a bias for either the horizontal or vertical direction. Besides, by describing the maximum row and column number of valid data in this scan zone, named RMAX and CMAX respectively, we have large probability to represent the information of a bit plane within less than 15 bits. Therefore, a signal-dependent rectangular scan zone starts from the upper-left corner will perform a more efficient coding of the coefficients [12].

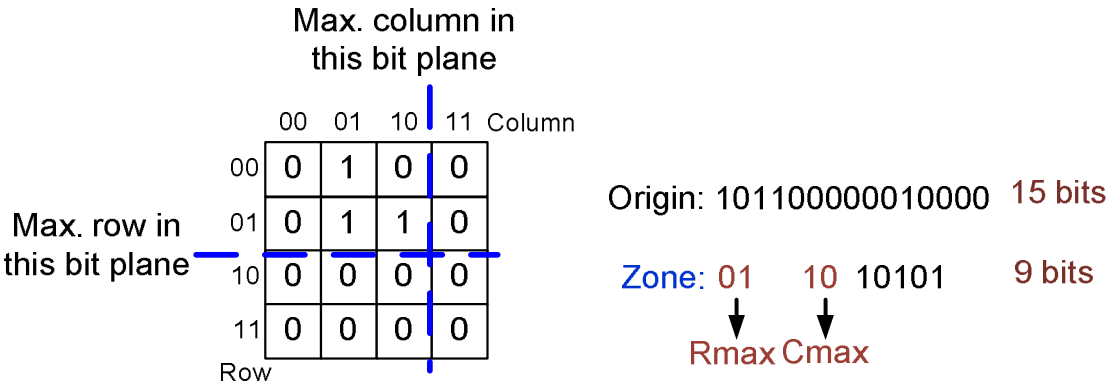


Fig. 3 The concept of bit-plane

Two classes of coefficients namely significant and in-significant coefficients are defined respectively. In the encoding flow, significant coefficient will have a 1 in any of the higher coded bit-planes. In the contrary, in-significant coefficient always have all 0's on the higher bit-planes.

Sometimes, zone represented by RMAX/CMAX will be very similar between the neighboring bit-planes. This feature allows us to use this data-similarity to develop more efficient coding mechanism.

The detail coding flow is described as follow: For DCT coefficient blocks, we can divide the process into DC and AC flows. In DC flow, the DC coefficient is completely packed for avoiding significant degradation in quality as BPT. In AC flow, the procedure of this algorithm is shown as Fig. 4. Initially, all AC coefficients are marked as insignificant. Then, we start from the most significant plane (MSP) to encode the subsequent bit-planes. The first plane which contains nonzero bit is defined as start plane, and the nonzero bits in start plane are the newly significant coefficients. Thus, sign bits are inserted behind each nonzero bit. For the subsequent bit-plane, there is only one question. If the following bit-plane has a newly significant bit, a bit "1" is packed first to represent the newly significant bit is founded and then the RMAX/CMAX must be also updated. The newly significant bits are followed by

corresponding sign bits. Those significant bits and in-significant bits are no need to be followed by sign bits since the sign bits of significant bits are already packed and the sign bit of in-significant bit are useless so far. Notice that unlike the fully packed sign bit in BPT, the sign bit packed in BPZ is on demand.

If no newly significant appeared in current bit-plane, a bit "0" is inserted to represent that the RMAX/CMAX of current bit-plane is the same as previous bit-plane and only the bits in the position of significant coefficient needed to be packed. BPZ repeat this procedure until all bit-planes have been packed. For the category on packing sign bits and the no newly significant bit-plane, we can see the efficiency of BPZ and that is why BPZ can achieve better performance than BPT.

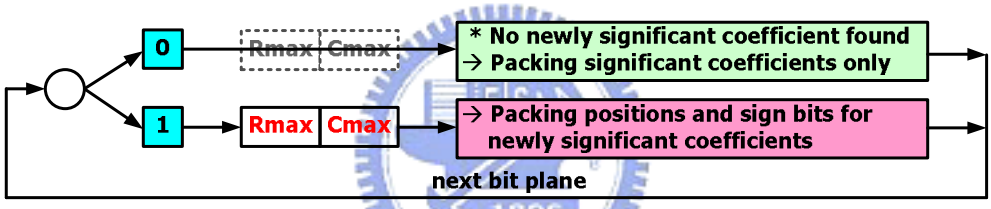


Fig. 4 Coding procedure of BPZ algorithm

An example for bit-plane classification is illustrated in Fig. 5. The same as BPT, the start plane of DCT coefficients is also packed as a part of header information. Sign bits of a DCT coefficient block are not a part of header information any more. They are dispersed and accompanied with newly significant coefficients found in certain bit-planes. Header information is shortened and more AC coefficient packing budget is reserved. New packing data format is shown in Fig. 6.

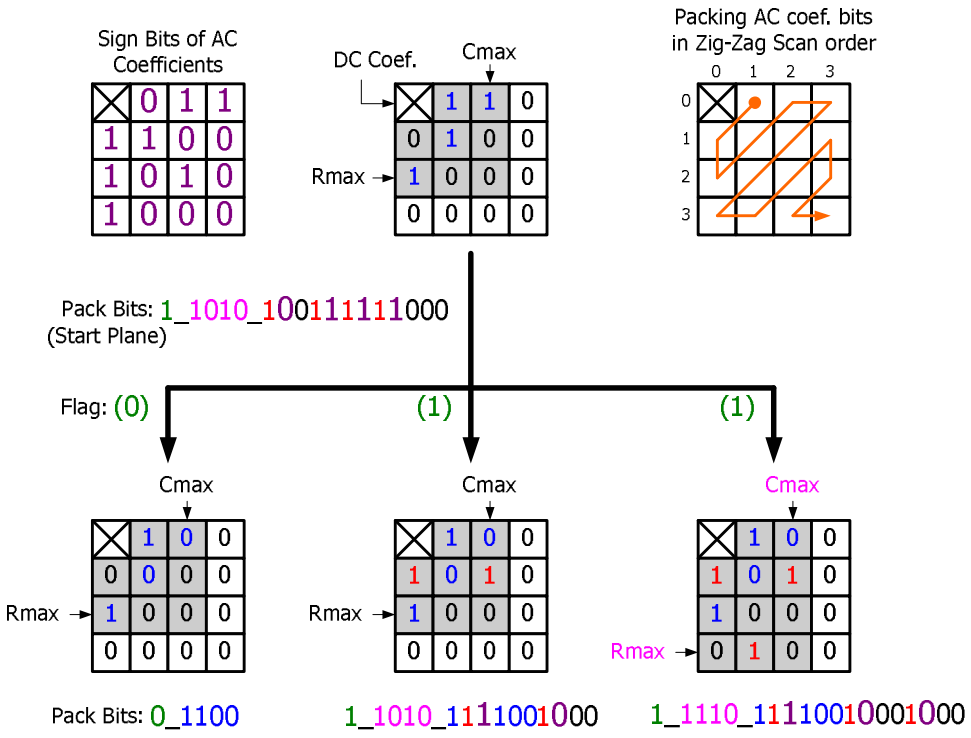


Fig. 5 An example of BPZ coding

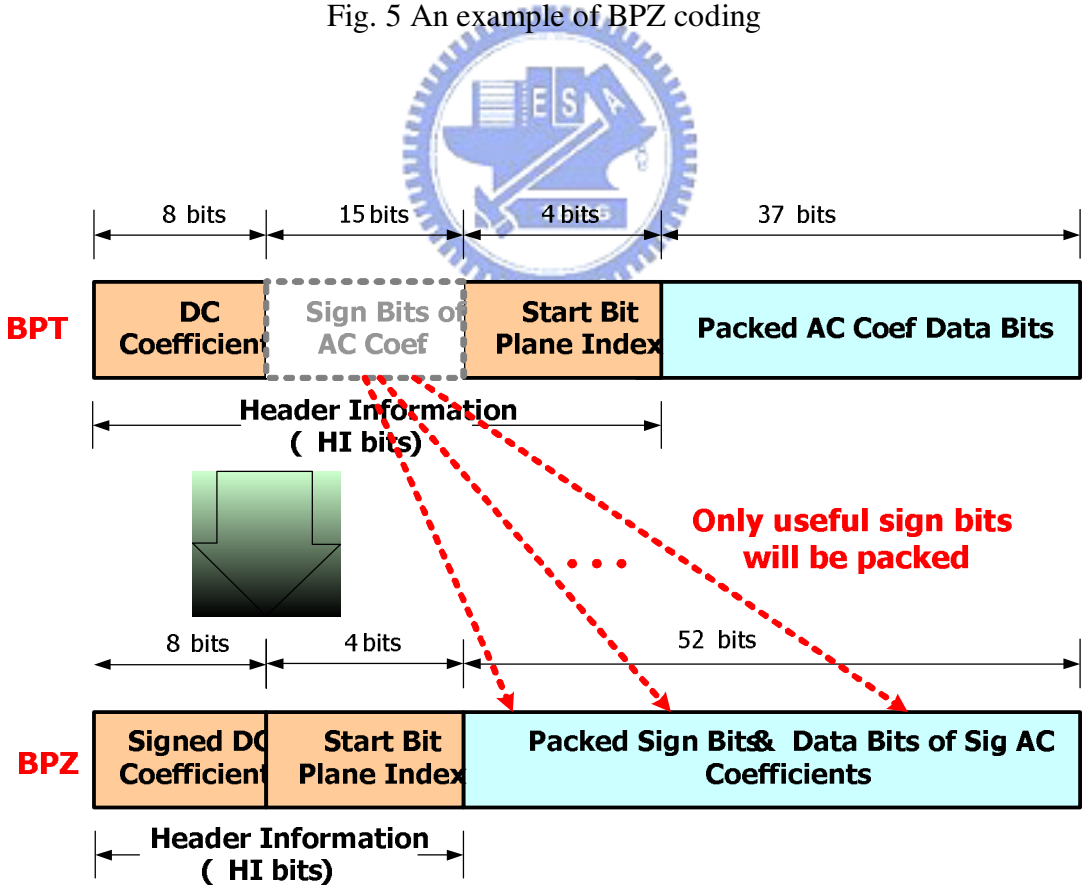


Fig. 6 New packing data format (BPZ) versus BPT

2.3.3 Modified Bit Plane Zonal Coding

If we take more look to the BPZ algorithm from the example shown in Fig. 4, we will discover that the original BPZ algorithm can be further improved. For software application, adding a little complexity can achieve more coding efficiency. A mechanism within good trade off between complexity and coding efficiency is proposed in [20].

The starting point is to use the limit budget in more efficient way. Carefully looking at the coding type of bit-plane zonal coding (BPZ), we can find that there is an annoying format to deal with the occurrences of newly significant coefficient because of the longest header information. Every time we found a newly significant bit, we need to packet 4 bits for RMAX/CMAX and one bit to distinguish coding format. However, the four bits of RMAX/CMAX is not really necessary since the RMAX/CMAX may be the same with the previous bit-plane. Therefore, [20] proposes a new coding format to deal with this situation. The new coding format is adopted when "newly significant bit is found, but the RMAX/CMAX of current bit-plane is the same with the previous bit-plane" and overall coding types shown in Table 1. The drawback is that we need one more bit to distinguish from original type B with new proposed type C. However the advantage is saving four bits comparing with original coding format. Fig. 7 shows the coding flow of modified bit-plane zonal coding proposed in [20].

Table 1 Coding types of bit-plane proposed in [20]

Type	Newly Sig. Coef. Found	Rmax/Cmax Changed	Flag	Bits for Rmax/Cmax	Bits for Flag(s) and Rmax/Cmax
A	Yes	Yes	None	4	4
B	No	No	00	None	2
C	Yes	No	01	None	2
D	Yes	Yes	1	4	5

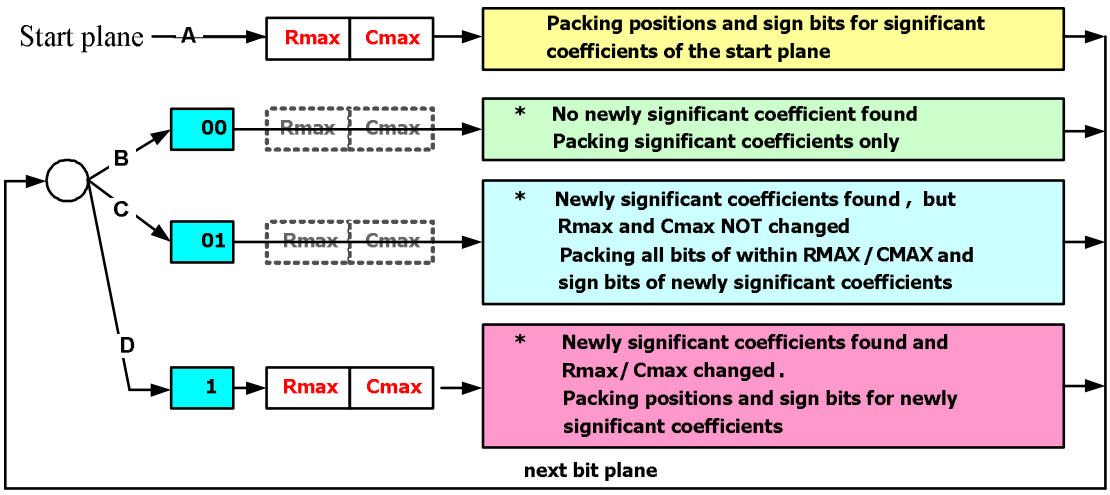


Fig. 7 Coding procedure of MBPZ algorithm

An example of the modified bit-plane zonal coding (MBPZ) proposed in [20] is given in Fig. 8. The bit streams in the bottom of figure are coded by original BPZ and modified BPZ (MBPZ) respectively. Through this compare we can clearly figure out the benefit brought by MBPZ. There is a small technique here. When packing a bit-plane of AC coefficients, we use zigzag scan order to collect bits. Since human visual system is more sensitive on low frequency signal elements, when we are running out of packing budget, zigzag scan order can store the relative important signal and bring us better visual quality within the same packing budget.

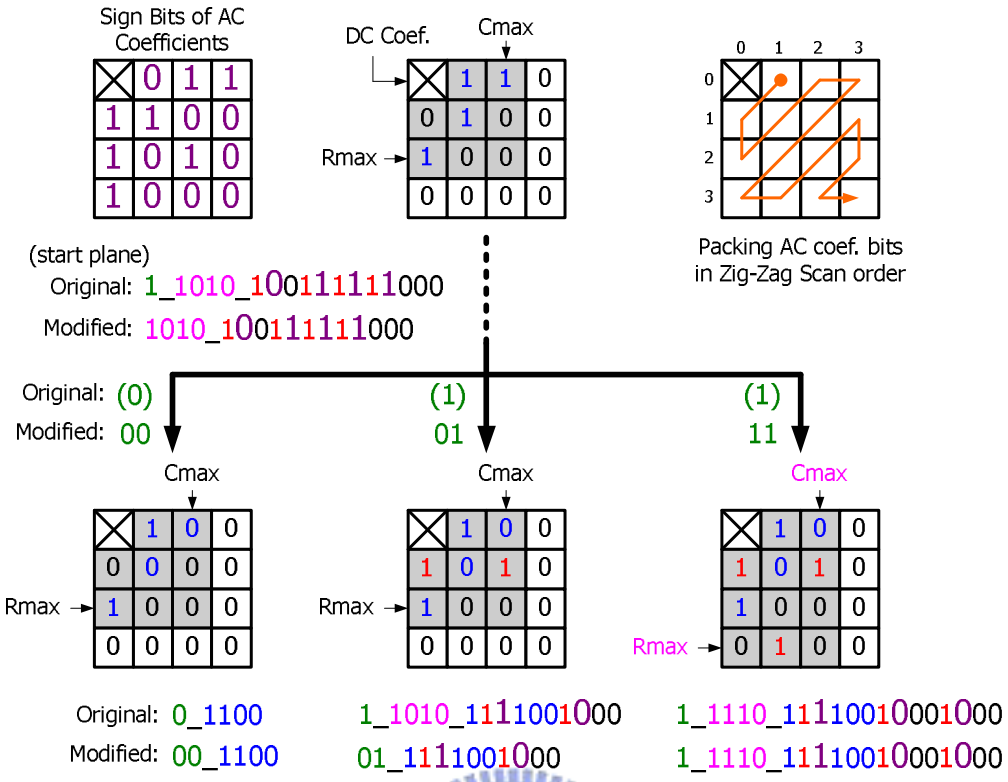


Fig. 8 An example for MBPZ coding

Using MBPZ to encode AC coefficients within limited budget, quality loss is inevitable. To slightly compensate for the truncated data bits, [20] also propose a method to raise the quality. First, if the value of this coefficient is large or equal than 4, scan the decoded AC coefficients from LSB to find the first non-zero bit, and then paste a “1” to its lower-two digit. If the value of this coefficient is less than 4, nothing will be changed on it. Finally, recover the coefficients by the corresponding sign bits (do two’s compliment or not). The compensation procedure is illustrated as Fig. 9.

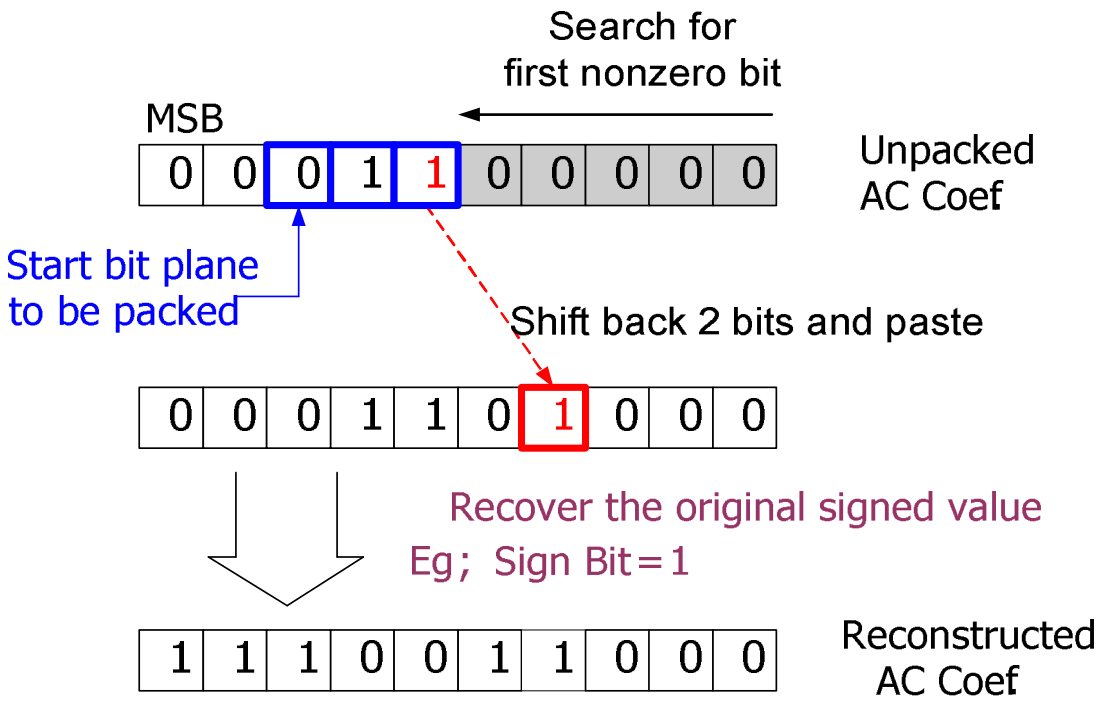


Fig. 9 Compensation for a bit-truncated AC coefficient.

2.4 Summary

From the introduction and discussion above, we classify the existing algorithm into two basic types and briefly introduce the pros. And cons. We can find that lossy compression is the popular way to implement embedded compressor by the advantage on fixed compression ratio and fixed amount of coded data. However, good performance usually comes with time consuming while low complexity usually brings worse quality. The former kind of methods derives better performance but the large buffer may be required, and longer processing cycles will enlarge the loading of the system and the barrier to embed this extra function into system. Although to slow down the system or to increase the operation frequency can fix this problem, but the former methods will decrease the coding throughput and the later methods will increase the power consumption. Each drawback is not what we want. Some lossy compression schemes are low complexity and high speed and easy to be embedded

into a decoder system as far as hardware is concerned, but at the same time, those schemes often suffer from unsatisfied quality.

For the real time, low power HDTV H.264/AVC decoder, low latency is the basic requirement. Not to increase the loading of original system is also another target. Therefore, our design challenge on embedded compressor is to find the optimal trade off between low latency, low complexity and high performance.

Chapter 3
Proposed Embedded Compression Algorithm

3.1 Overview

Researches about data compression have been developed for a long time. Those developed algorithms show us that enhancing the complexity can reach better performance. However, the problem is to find a suitable compression category to combine with H.264 system but not to affect the performance of overall system. The discussions in chapter 2 have shown us that the threshold of embedding an extra function may arise with higher complexity coding scheme. In this chapter, further discussion will be presented.

In practice, block-based schemes are the most convenient schemes because they match the block-oriented structure of the incoming bit-stream in H.264 system and allow on-the-fly process. However, another problem exists: the overhead. The overhead can be defined as the ratio between the number of pixels that are actually accessed during the motion compensation of a block and the number of pixels that are really useful in the reference block. In original system, the ratio is 1 since every accessed pixel is on demand. After embedding block-based algorithm adopted, this ratio will always superior to 1 because of the nature of block-based embedded compression algorithm. Fig. 10 shows the concept between block-based and pixel-based. The left of Fig. 10 is pixel-based, represents the data without EC. The right of Fig. 10 is block-based since the characteristic of EC. Fig. 11 is an example to show how overhead occurs.

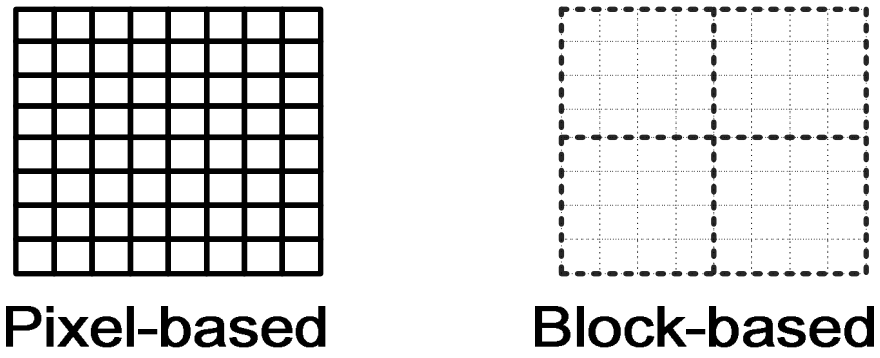


Fig. 10 Pixel-based (left) versus block-based (right)

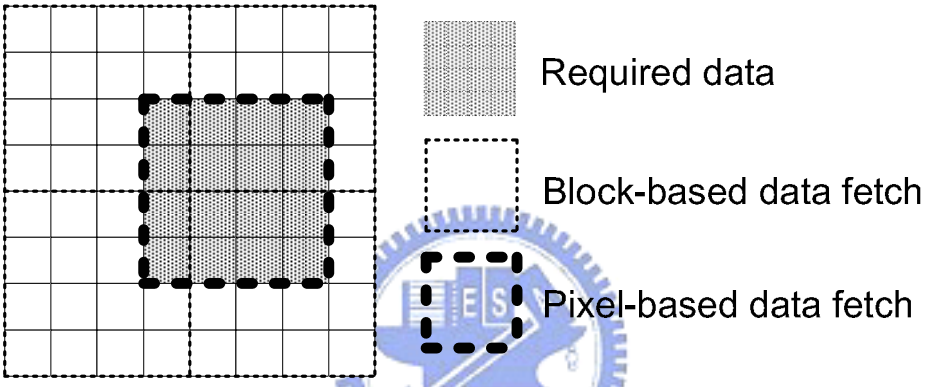


Fig. 11 An example of overhead problem

According to the standard of H.264, a 16x16 macro block can be divided into 8x8, 8x16 or 16x8 blocks during the process of motion compensation (MC). Further more, an 8x8 block can then be sub-divided into 8x4, 4x8 or 4x4 sub blocks. If the compensated block is not aligned with the coded block grid, the overhead will be occurred like depicted in Fig. 11. Four coded blocks have to be loaded and decoded to get the required pixels. If the EC scheme is 8x8 block-based and the compensated block is 4x4 block, we need to load and decode 256 pixels to derive 16 useful pixels. The overhead in this case is 16. Because of the overhead problem, the relation between the compression ratio of EC and the gain in memory transfer is not direct.

There is a statistic material about the phenomenon of overhead provided by [15]. Fig. 12 shows the relation between overhead and encoding bit-rate simulating with

Stefan sequence. Three kinds of EC block-grid are presented. Since H.264 encoder allows macroblock (MB) partitioning and larger motion vectors at high rate (which also means the small quantization step and better quality) and favors the null vectors with 16x16 partition at low rate, the overhead increases while the bit rate increases.

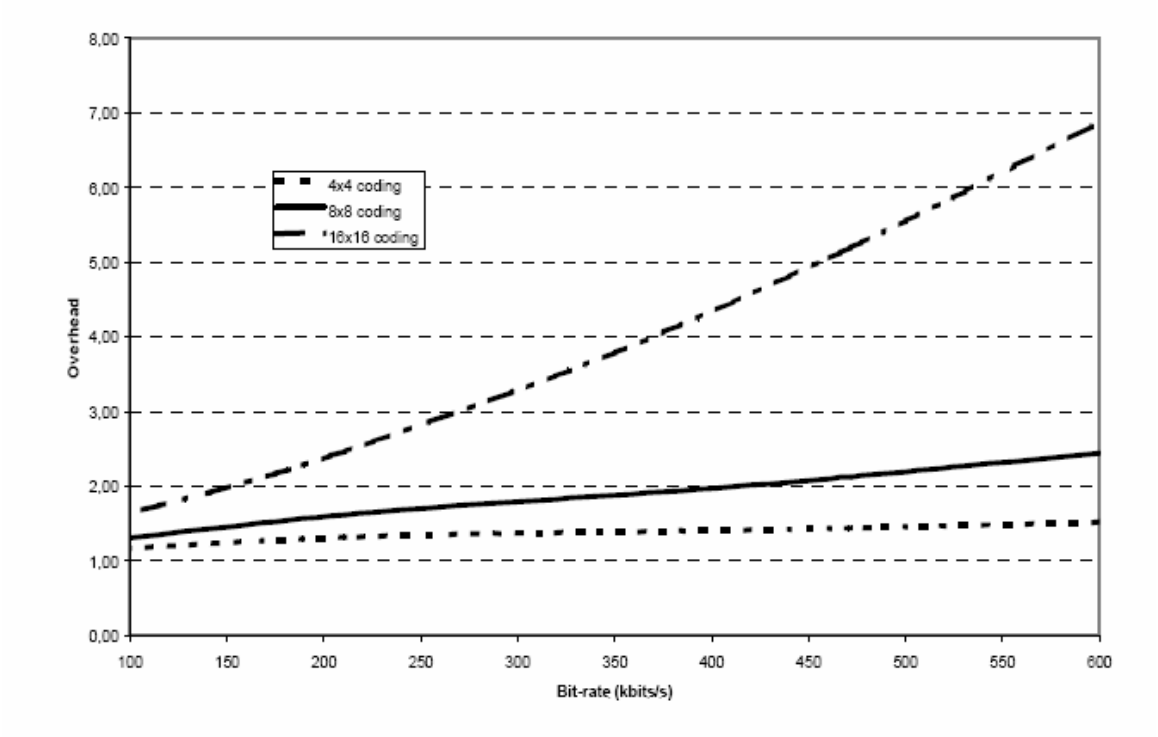


Fig. 12 The correlation between bit-rate and overhead (Stefan sequence) simulated with 4x4, 8x8 and 16x16 block grid

Table 2 [15] is the summary of the statistical analysis simulated with six sequences. In this table, we can see that the relatively still sequences (News, Weather) generate smaller overhead since the motion vector is often equal to zero while the fast motion sequence such as Stefan generates more overhead. Finally, an important conclusion is that the smaller block-grid gets the better of larger block-grids and derives the smallest overhead.

Table 2 Overhead with EC block grid for each sequence

Sequence	4x4 block grid	8x8 block grid	16x16 block grid
Foreman	**1.31**	**1.77**	**3.69**
Flower	**1.30**	**1.74**	**3.77**
News	**1.14**	**1.51**	**2.78**
Silent	**1.17**	**1.50**	**3.22**
Stefan	**1.51**	**2.44**	**6.95**
Weather	**1.17**	**1.49**	**3.18**
All	**1.27**	**1.73**	**3.93**

3.2 Algorithm of Embedded Compressor

We adopt transform-based and 4x4 block-grid as our coding algorithm. First reason is the smallest overhead according to the statistical result that we presented in previous section. Actually it is a trade off between coding efficiency and overhead. We know that as far as the transform algorithm is concerned, the bigger the block-grid, the better coding efficiency it can achieve. Since we want to increase the coding efficiency with less overhead, the 4x4 block-grid is our best choice.

The basic concept of proposed algorithm is the combination of DCT with bit-plane zonal coding. DCT is a well known technique so we just simply introduce it. The two proposed bit-plane zonal coding are the main characters. Fine grain bit-plane zonal coding (FGBPZ) is quite efficiency and is suitable to be used in software application. Coarse grain bit-plane zonal coding is relatively simple and is suitable for hardware implementation. Fig. 13 is the coding flow of proposed DCT-FGBPZ/CGBPZ algorithm. This is a one way open-loop coding scheme and no iteration is needed. The discrete cosine transform (DCT) is divided into two one dimension DCT. The coefficients of DCT are packed by fine grain bit-plane zonal coding (FGBPZ) or coarse grain bit-plane zonal coding (CGBPZ) we proposed. The detail of each part will be introduced in the following sections.

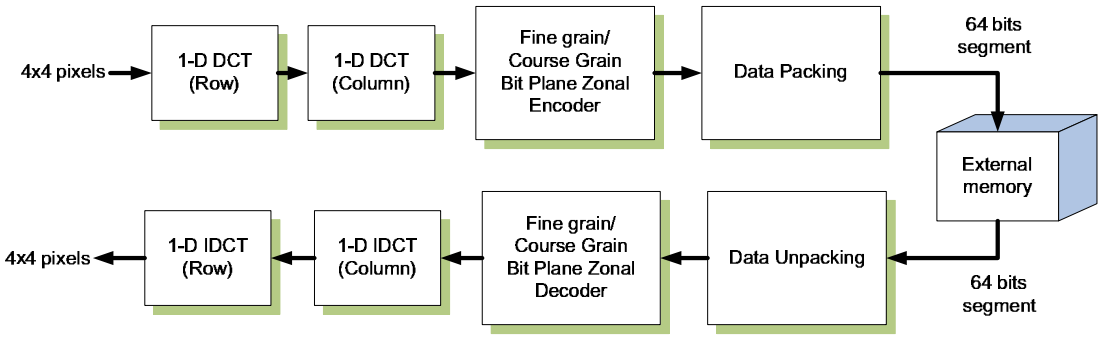


Fig. 13 The flow chart of proposed DCT-FGBPZ/CGBPZ embedded compression

3.2.1 Discrete Cosine Transform

Discrete cosine transforms (DCT) is a powerful technique for converting a signal into elementary frequency components. It is widely used in image compression and JPEG is the well-known example.

For human visual system, human eyes are more sensitive on low frequency component of a picture and less sensitive on high frequency component. Therefore, the quality loss in high frequency component is relatively unnoticeable. The DCT can generate the relatively important low frequency component on up left corner, and the most high frequency in down right corner. Thus the DCT combines with bit-plane zonal coding with original point at up left corner can efficiently collect the information.

But the biggest disadvantage of DCT is its complexity on hardware design. Here we make our coding unit in 4x4 block grid, the complexity of 4 point DCT is minor and still can take the advantage of the transform. The complexity of different size of DCT can be evaluated in Table 3. Two designs are shown in Table 3. A design is reference from [16] and B design is reference from [17]. B design is focus on reducing multiplications by increasing additions. We can see that in both designs, the complexity of 4 points DCT is much simpler than 8 points and 16 points. 4 points DCT can be considered as most economical type of DCT. Notice that $N = 2^m$.

Table 3 The complexity of N-point DCT

m	N	Number of Multiplications		Number of Additions	
		A	B	A	B
2	4	2	4	6	9
3	8	16	12	26	29
4	16	116	80	194	209

3.2.2 Proposed Fine Grain Bit Plane Zonal Coding (FGBPZ)

Base on the modified bit-plane zonal coding proposed in [20], the coding efficient is quite good. But we are not satisfied yet. To further improve the coding efficiency, we introduce a pre-determined variable length coding here with a small code book.

3.2.2.1 VLC Codebook

Before further change the MBPZ in [20], we make simulation here to evaluation the occurrences of each MBPZ types and Fig. 14 is the simulation result. The naming of each type A, B, C and D is referred from [20] (see Fig. 7). We can see that the appearance probability of type B and type C are relatively small although they have better coding efficiency. Type D is the dominate type but the bits recording header information are 5 bits including one bit for distinguishing between types and 4 bits for RMAX/CMAX. Therefore, we want to improve the efficiency by adding a small variable length code (VLC) codebook on type D.

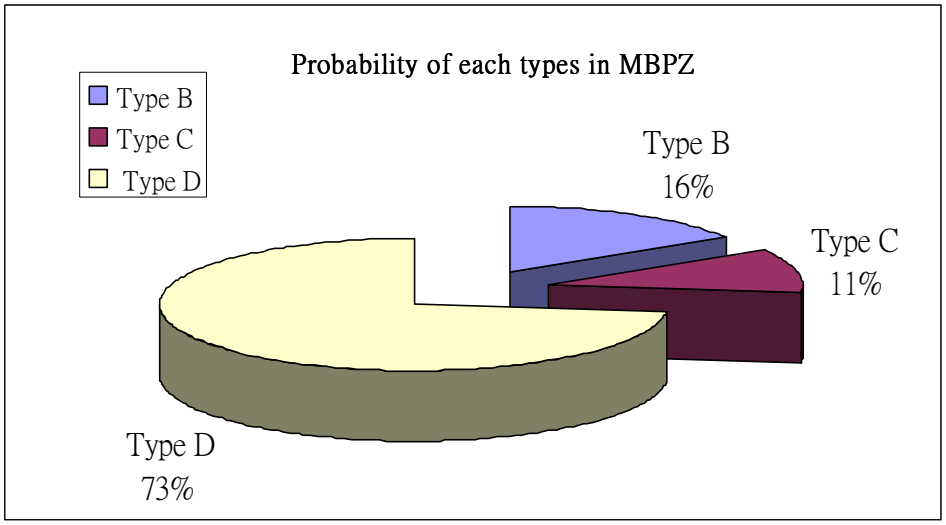


Fig. 14 the occurrence probability of each types in MBPZ

According to the modified bit-plane zonal coding proposed in [20], the RMAX/CMAX of each bit-plane is accumulated bit-plane by bit-plane and is always large or equal to the RMAX/CMAX of previous plane. Recall that type D is applied when RMAX/CMAX is changed. Therefore, when type D is applied, the possible outcomes of the RMAX/CMAX in next bit-plane are limited: they must larger than the RMAX/CMAX of previous plane.

For example, if RMAX/CMAX of current plane is 2/2 and next plane is coded by type D, the possible outcomes of next plane RMAX/CMAX must be 3/2, 2/3 or 3/3. Notice that 2/2 is also possible to be the RMAX/CMAX of next bit-plane, but type D only deals with the situation that RMAX/CMAX is different from previous bit-plane. Those 3 possible outcomes can be fully presented by 1~2 bits instead of original 4 bits. The description above explains the chance of reducing the codeword length in type D. Fig. 15 shows the coding flow of FGBPZ with VLC codebook. This method can save up to four bits when type D is applied.

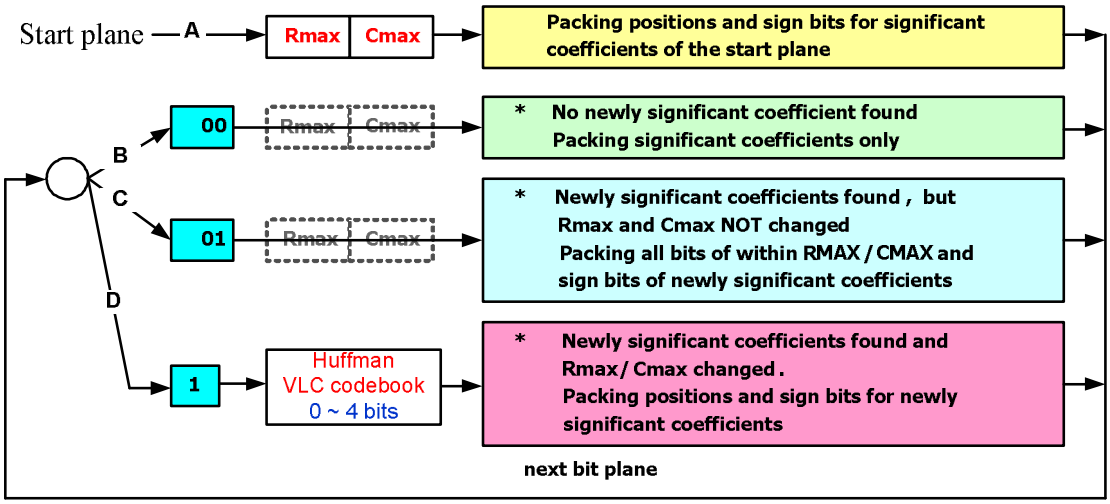


Fig. 15 Coding flow of FGBPZ with VLC codebook. Recall that type A, B, C and D is referred from [20].

We generate those codes by Huffman coding methods and the probabilities of next possible RMAX/CMAX ($P_{\text{current RMAX/CMAX}}$ [next RMAX/CMAX]) are derived from simulation on over 3000 frames. The code words in this codebook are fixed.

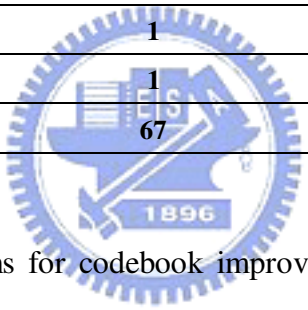
To cover all possible CMAX/R/MAX of next bit-plane according to current plane, the needed codebook entry and their related RMAX/CMAX is shown in Table 4. The number of possible outcomes of next RMAX/CMAX is shown in (1). For a 4x4 bit-plane, the row/column are mark as 0, 1, 2, 3. When type D is applied, at least one of row or column is changed. Therefore, this equation is to calculate the outcomes which are large than or equal to current RMAX/CMAX and then minus one outcome that RMAX and CMAX are both equal to current bit-plane.

Next possible outcomes =

$$(4 - Current\_RMAX) \times (4 - Current\_CMAX) - 1 \quad (1)$$

Table 4 The needed codebook entries and their related RMAX/CMAX

Current RMAX/CMAX	The number of next Possible RMAX/CMAX outcomes	Huffman Code length (bits)
(0, 1)	**11 (4*3-1)**	**3~4**
(1, 0)	**11 (3*4-1)**	**3~4**
(1, 1)	**8 (3*3-1)**	**2~4**
(2, 0)	**7 (2*4-1)**	**2~4**
(0, 2)	**7 (4*2-1)**	**2~4**
(2, 1)	**5 (2*3-1)**	**2~3**
(1, 2)	**5 (3*2-1)**	**2~3**
(2, 2)	**3 (2*2-1)**	**1~2**
(3, 0)	**3 (1*4-1)**	**1~2**
(0, 3)	**3 (4*1-1)**	**1~2**
(3, 1)	**2 (1*3-1)**	**1**
(1, 3)	**2 (3*1-1)**	**1**
(3, 2)	**1**	**0**
(2, 3)	**1**	**0**
Summary	**67**	**0~4**



But there are still rooms for codebook improvement. Consider the following two cases: case 1), current RMAX/CMAX is 2/3; next RMAX/CMAX is 3/4. Case 2), current RMAX/CMAX is 3/2; next RMAX/CMAX is 4/3. With the original codebook, the codebook index for case 1 is {(2, 3), (3, 4)} and case 2 is {(3, 2), (4, 3)}. Actually, the mainly different of case 1 and case 2 is the direction of row and column. Both cases are similar even on the probability distribution of each possible "next RMAX/CMAX". If we switch the row to the column, we can find that those two cases are undergoing the same changes. According to this idea, we introduce our symmetric VLC codebook. By eliminating the bias of Row and Column in codebook, the symmetric cases can share the same codeword. We can reduce the 67 entries codebook into 40 entries by this idea.

Actually, this idea does not reduce the number of comparisons. But it reduces the

codebook size by 42%. The timing wasted on codebook searching is also reduced.

And then we will show how to use our symmetric VLC codebook. We represent current RMAX/CMAX as Cm_cur, Rm_cur, previous RMAX/CMAX as Cm_pre, Rm_pre. The action of table look up can be described as follow:

If ($Cm\_pre \geq Rm\_pre$)
  Codeword at index {(Cm_pre, Rm_pre) (Cm_cur, Rm_cur)} is applied.
Else
  Codeword at index {(Rm_pre, Cm_pre) (Rm_cur, Cm_cur)} is applied.

Therefore, 40 codeword is enough.

And then we want to explain the decoding procedure of symmetric VLC codebook. After start plane is decoded, the RMAX/CMAX of start plane is known and can be used as reference. Decoding procedure for the subsequent bit-planes can be illustrated in (2).

If ($Cm\_pre \geq Rm\_pre$)
  Codeword in block {(Cm_pre, Rm_pre)} is searched;
  And the result is in {(Cm_cur, Rm_cur)} order.
Else (2)
  Codeword in block {(Rm_pre, Cm_pre)} is searched;
  And the result is in {(Rm_cur, Cm_cur)} order.

These switch actions between RMAX and CMAX in encoding procedure need not to be recorded since they can be derived from the decoding procedure. The final VLC codebook is shown in Table 5 and is formed by eliminating the symmetric entry in Table 4. The coding example for FGBPZ is shown in Fig. 16. Table 6 is our detail codebook with code words.

Table 5 The final 40 entries VLC codebook

Current RMAX/CMAX	The number of next Possible RMAX/CMAX outcomes	Huffman Code length (bits)
(1, 0)	**11 (3*4-1)**	**3~4**
(1, 1)	**8 (3*3-1)**	**2~4**
(2, 0)	**7 (2*4-1)**	**2~4**
(2, 1)	**5 (2*3-1)**	**2~3**
(2, 2)	**3 (2*2-1)**	**1~2**
(3, 0)	**3 (1*4-1)**	**1~2**
(3, 1)	**2 (1*3-1)**	**1**
(3, 2)	**1**	**0**
Summary	**40**	**0~4**

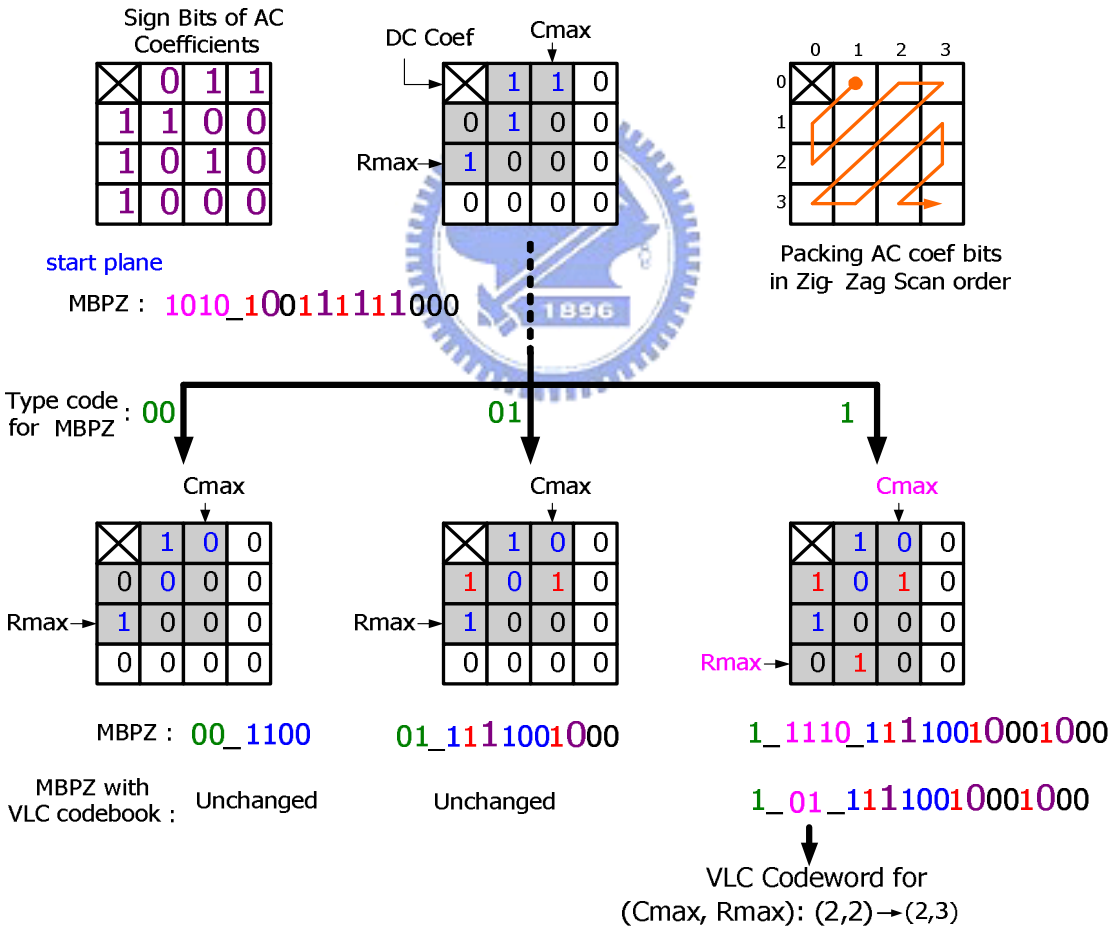


Fig. 16 A coding example for FGBPZ

Table 6 The overall codeword in VLC codebook

Current RMAX/CMAX	Next RMAX/CMAX	Codeword	Code length (bits)
(1, 0)	**(1, 1)**	**000**	**3**
	(2, 0)	**001**	**3**
	(2, 1)	**010**	**3**
	(3, 0)	**011**	**3**
	(2, 2)	**100**	**3**
	(1, 2)	**1010**	**4**
	(3, 1)	**1011**	**4**
	(3, 2)	**1100**	**4**
	(3, 3)	**1101**	**4**
	(2, 3)	**1110**	**4**
	(1, 3)	**1111**	**4**
(1, 1)	**(2, 2)**	**00**	**2**
	(2, 1)	**100**	**3**
	(1, 2)	**101**	**3**
	(3, 3)	**110**	**3**
	(3, 2)	**111**	**3**
	(2, 3)	**010**	**3**
	(3, 1)	**0110**	**4**
	(1, 3)	**0111**	**4**
(2, 0)	**(2, 1)**	**00**	**2**
	(3, 0)	**01**	**2**
	(3, 1)	**100**	**3**
	(2, 2)	**101**	**3**
	(3, 2)	**110**	**3**
	(3, 3)	**1110**	**4**
	(2, 3)	**1111**	**4**
(2, 1)	**(3, 2)**	**01**	**2**
	(2, 2)	**00**	**2**
	(3, 3)	**10**	**2**
	(3, 1)	**110**	**3**
	(2, 3)	**111**	**3**
(2, 2)	**(2, 3)**	**00**	**2**
	(3, 2)	**01**	**2**

	(3, 3)	**1**	**1**
(3, 0)	**(3, 1)**	**0**	**1**
	(3, 2)	**10**	**2**
	(3, 3)	**11**	**2**
(3, 1)	**(3, 2)**	**0**	**1**
	(3, 3)	**1**	**1**

3.2.2.2 Data Packing

Since our compression ratio is fixed at two, the budget of coded data is 64 bits. After DCT and bit-plane zonal coding, we need to packet coded data into 64 bits segment before sending to external memory. First we reserve for the DC coefficient because of its importance in transform. Second, we use 4 bits to packet the start plane. The rest of budget, that is to say, 52 bits, is used for storing AC coefficients. With the help of the fine grain bit-plane zonal coding, AC coefficient are divided into bit-planes and presented by the coding format in Fig. 15 Coding flow of FGBPZ with VLC codebook. Recall that type A, B, C and D is referred from [20].. The procedure is keep packing bit-plane by bit-plane until the end of bit-planes or running out of bit budget.

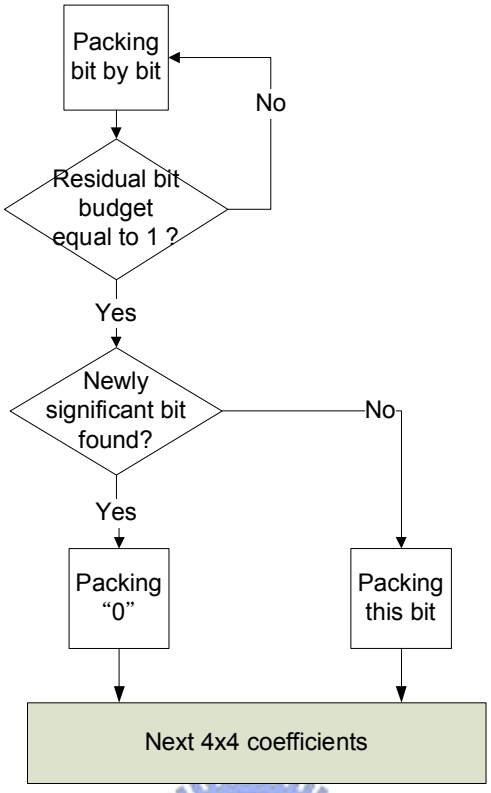


Fig. 17 Protecting mechanism for unknown sign bit

When running out of budget, unpacked information will be loss. Recall that the newly significant coefficient must be followed by its sign bit. If newly significant bit is packed while its sign bit is cut, this coefficient will be wrong after decoded. We make a mechanism to avoid this situation and show in Fig. 17. If next packing bit is newly significant bit and the rest of the budget is less than two bits, we will abort packing this newly significant bit.

The final encoding flow chart is shown in Fig. 18. Each bold line in Fig. 18 represents a check point checking whether if we run out of the budget.

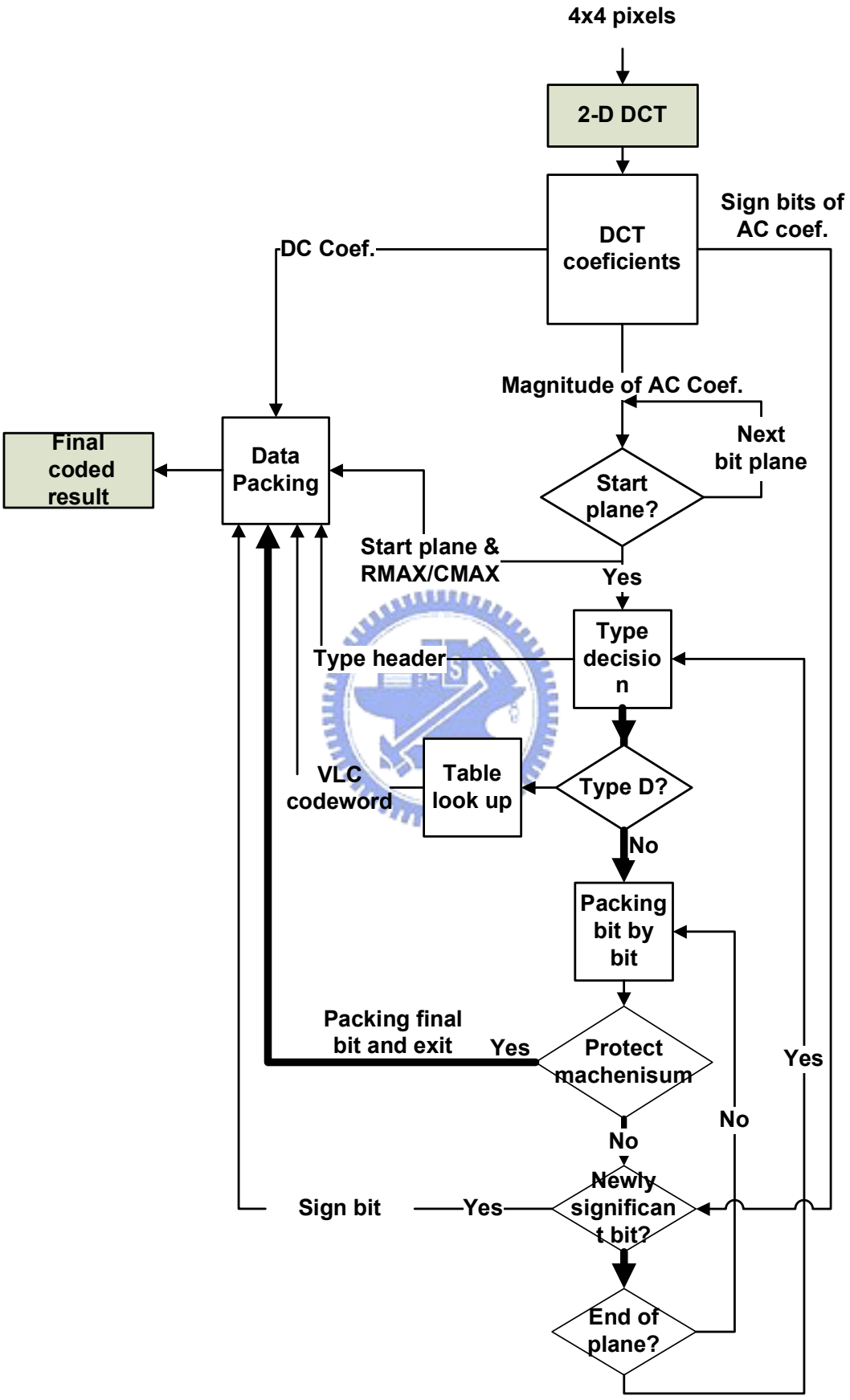


Fig. 18 Final encoding flow chart

3.3 Coarse Grain Bit-Plane Zonal Coding (CGBPZ)

FGBPZ introduced in section 3.2.2 is simple and efficiency. This algorithm encodes the coefficients on "bit" level. But its encoding procedure may cost more than 30 cycles and decoding procedure may cost more than 10 cycles under our estimation. So FGBPZ is more suitable embedded into software or hardware/software co-design system. To implement the algorithm as hardware accelerator, the algorithm must be further modified into simpler version.

By the discussion in chapter 6.1, we will see the critical problems of embedding a compressor into system. Taking all these problems into consideration, we propose coarse grain bit-plane zonal coding (CGBPZ). CGBPZ is a trade off between short cycles, ability of parallelism and quality. The details will be presented in this section.

Fig. 19 is the coding formats of CGBPZ. All magnitude bit-planes of AC coefficients are coded in uniform format. For each bit-plane, we record the RMAX/CMAX (4 bits), and then pack the bits which are enclosed by RMAX and CMAX. 4 bits are used to record RMAX/CMAX of each plane. The dependencies between bit-planes are not used in CGBPZ.

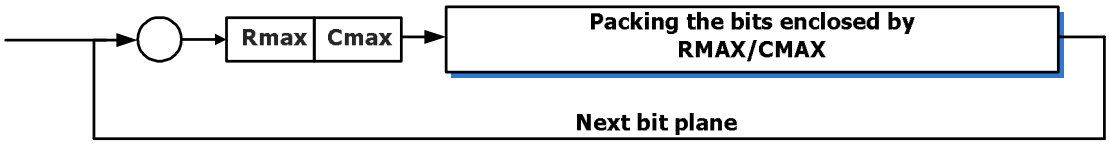


Fig. 19 CGBPZ coding format for the magnitude of AC coefficients

In CGBPZ, we introduced the concept of sign bit-plane. Sign bit-plane can be considered as union of sign bits for each coefficient. We only packet those used sign bits. Because we have only 64 bits budget for each 4x4 unit, the situation of unable to

pack all the information may be happened frequently. Since not every coefficient can be packed, packing whole sign bit-plane may become a waste. So we take the maximum value of RMAX and CMAX from packed bit-plane (from start plane to end plane) and packing sign bit-plane by those two boundaries. Under this method we will waste least bits to pack unused sign bits. The RMAX/CMAX of sign bit-plane needs not to be packed when encoding, because they can be derived from those coded bit-plane. Fig. 20 illustrates the idea of how we derive the RMAX/CMAX of sign bit-plane.

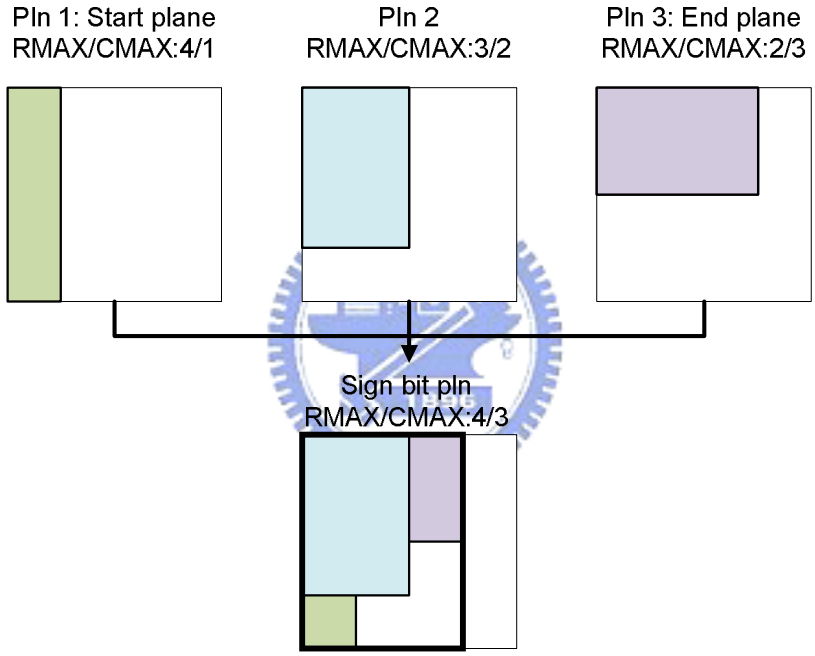


Fig. 20 The concept of how to derive the RMAX/CMAX of sign bit-plane from coded bit plane.

Finally, in CGBPZ, the end plane needs to be estimated and packed to fulfill the decoding procedure. Fig. 21 shows the simple concept of end plane decision. From MSB plane to LSB plane, the calculator estimates the total bits usage accumulated from most significant plane (MSP) to current plane. If total bits usage is more than 64 bits when accumulates to Nth bit-plane, (N+1)th bit-plane is the end plane.

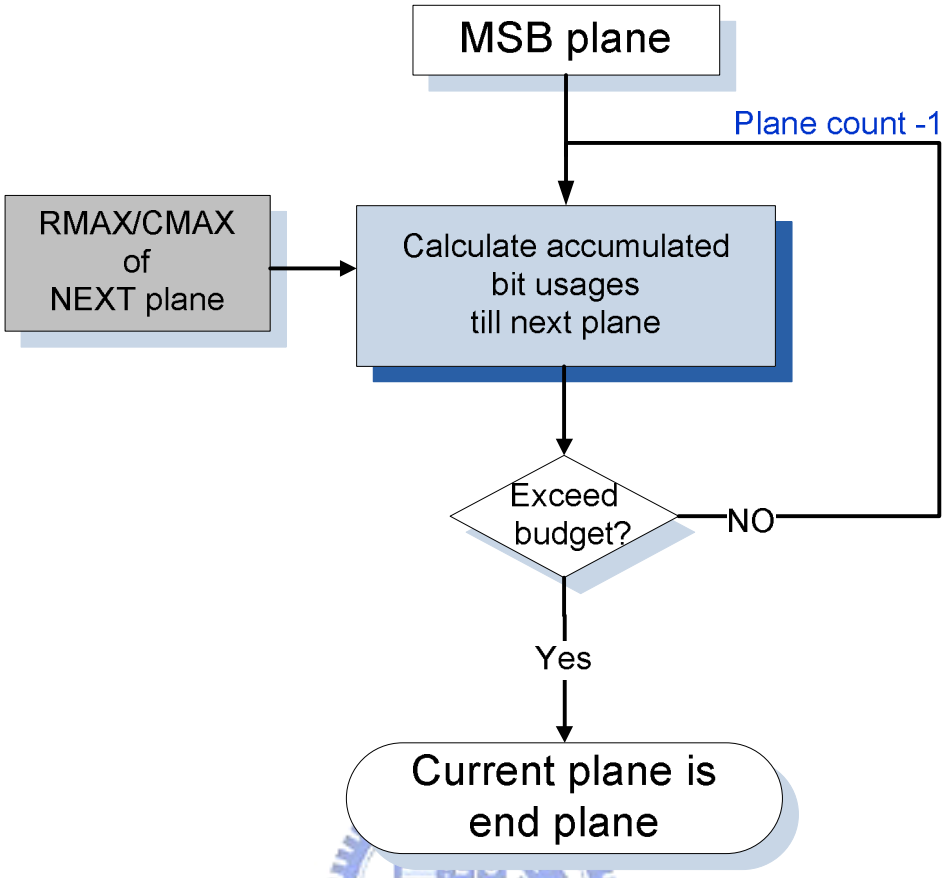


Fig. 21 End plane decision

The overall encoding flow can be shown in Fig. 22. Finally, there is one small skill. According to the description above, the bits usage accumulated to end plane is less than bit budget. Therefore, there are few bits unused. To well use those bit budgets, we keep putting the information into those unused budgets within the RMAX/CMAX of sign bit-plane.

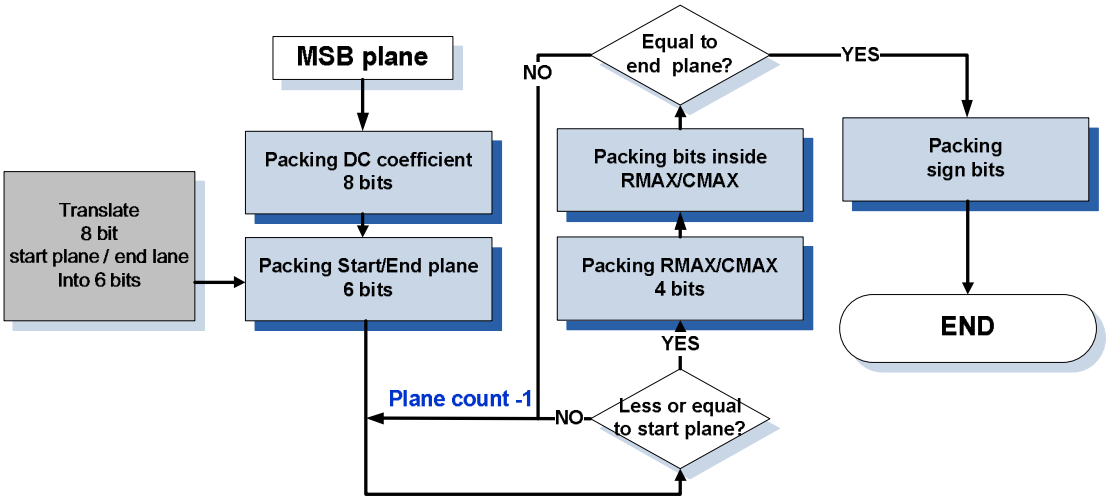


Fig. 22 Overall encoding flow of CGBPZ

3.4 Decoding Process and the Compensation

Roughly, the decoding process can be thinking as the inverse process of encoding. We take the coded data segments and divide them into DC coefficient and AC coefficients.

Since the algorithm we proposed is a lossy compression and the lower bit-planes of AC coefficients are often truncated due to limited budget, we apply a simple compensation here. The basic concept is shown in Fig. 23. The compensation is applied when the coefficient is nonzero and the end plane is larger than least bit-plane. This compensation technique can be considered as adding a median number of lost bit-plane. It leads to a satisfied quality improvement. Notice that this compensation is slightly different with [20] and has better quality improvement.

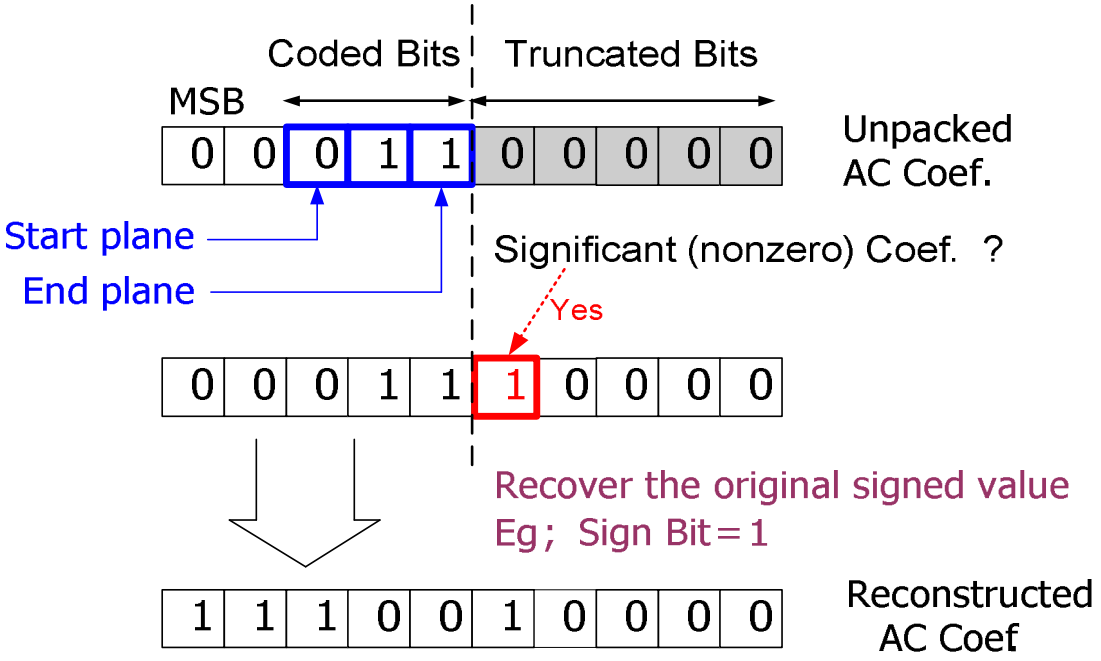


Fig. 23 Proposed compensation technique

3.5 Embedded Result on Software Simulation

Before all the discussion, we want to define the formula of PSNR calculation first. All the PSNR values in this section are the PSNR between compressed sequences versus the original sequence. The reason why we choose original sequence as reference is to establish an absolute quality level. The equation of PSNR is given in (3):

$$PSNR = 10 \times \log \frac{255 \times 255}{\frac{1}{R \times C} \times \sum_{r=0}^{R-1} \sum_{c=0}^{C-1} (P_{origin}(r,c) - P_{compressed}(r,c))^2} \quad (3)$$

3.5.1 FGBPZ versus CGBPZ

In this section, we focus on comparing the coding efficiency between fine grain

bit-plane zonal coding (FGBPZ) and coarse bit-plane zonal coding (CGBPZ). We want to show the result of trade off between FGBPZ and CGBPZ. Fig. 24 shows the embedded result on Foreman sequences with group of picture (GOP) 20. We can see the PSNR value decades along the P frame number. This is because each P frame is formed by referring the blocks in previous frame. Since every reference frames are compressed by our lossy EC algorithm, the errors will be propagated and accumulated through P frames. This phenomenon is also called drift effects. Fig. 25 shows the drift effect but the experimented sequence is Mobile Calendar. Mobile Calendar is famous by its complex components and fast motion. Those features make Mobile sequence difficult to be compressed and the loss on quality may larger than slow motion sequences.

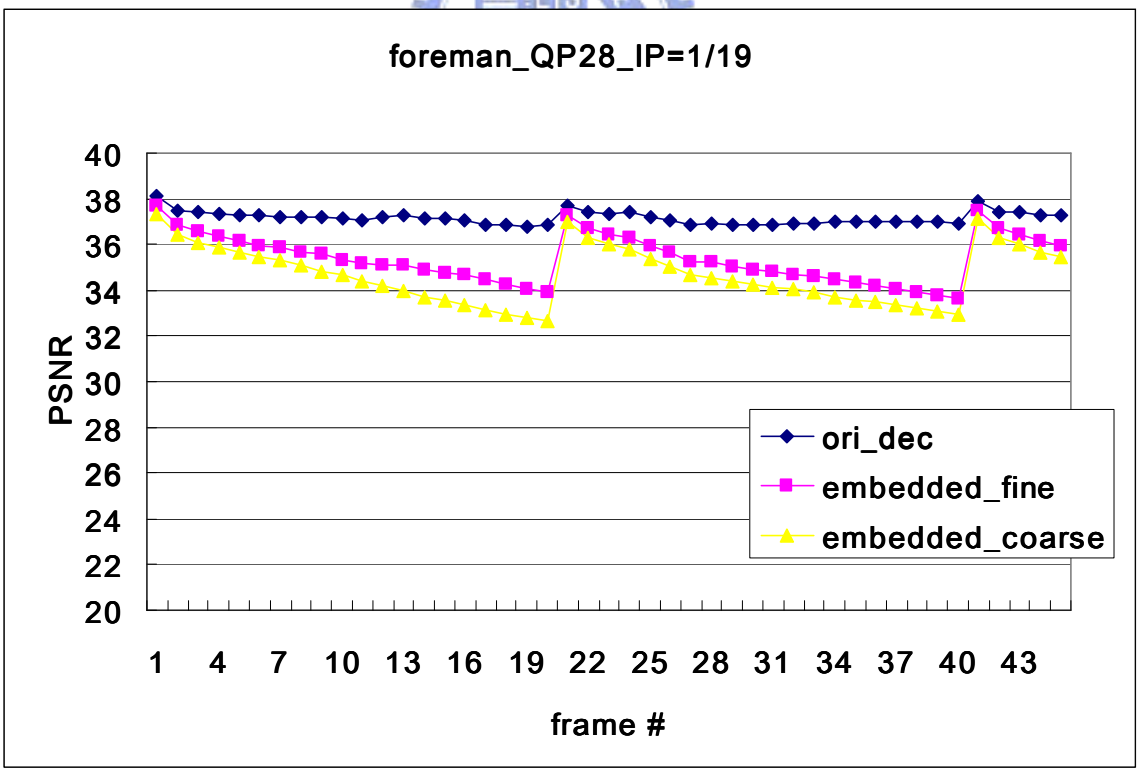


Fig. 24 Drift effects on Forman_QP28_GOP20

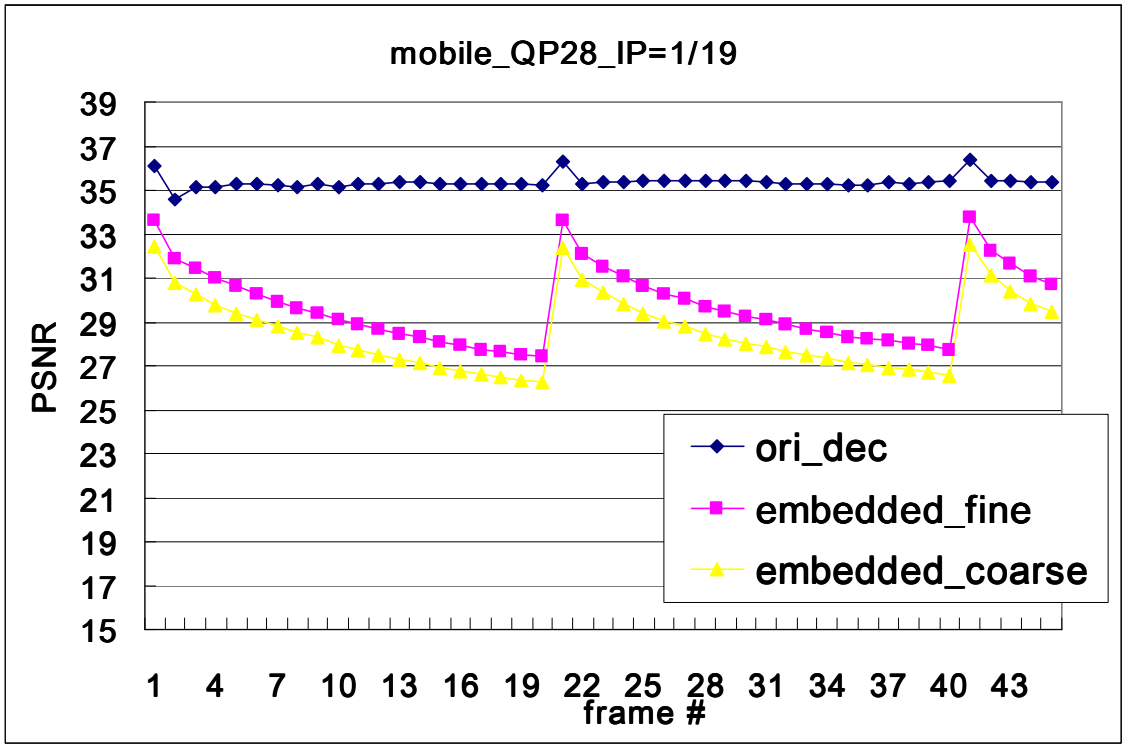


Fig. 25 Drift effects on Mobile_QP28_GOP20

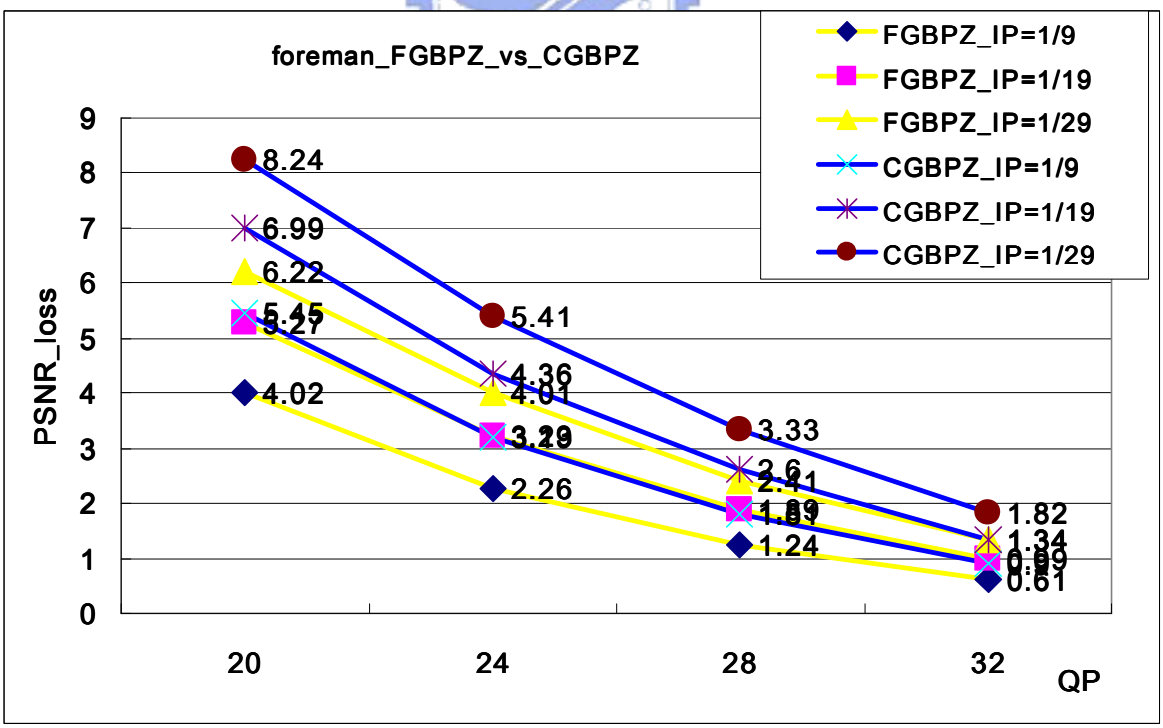


Fig. 26 PSNR loss considering different QP and different GOP (Foreman)

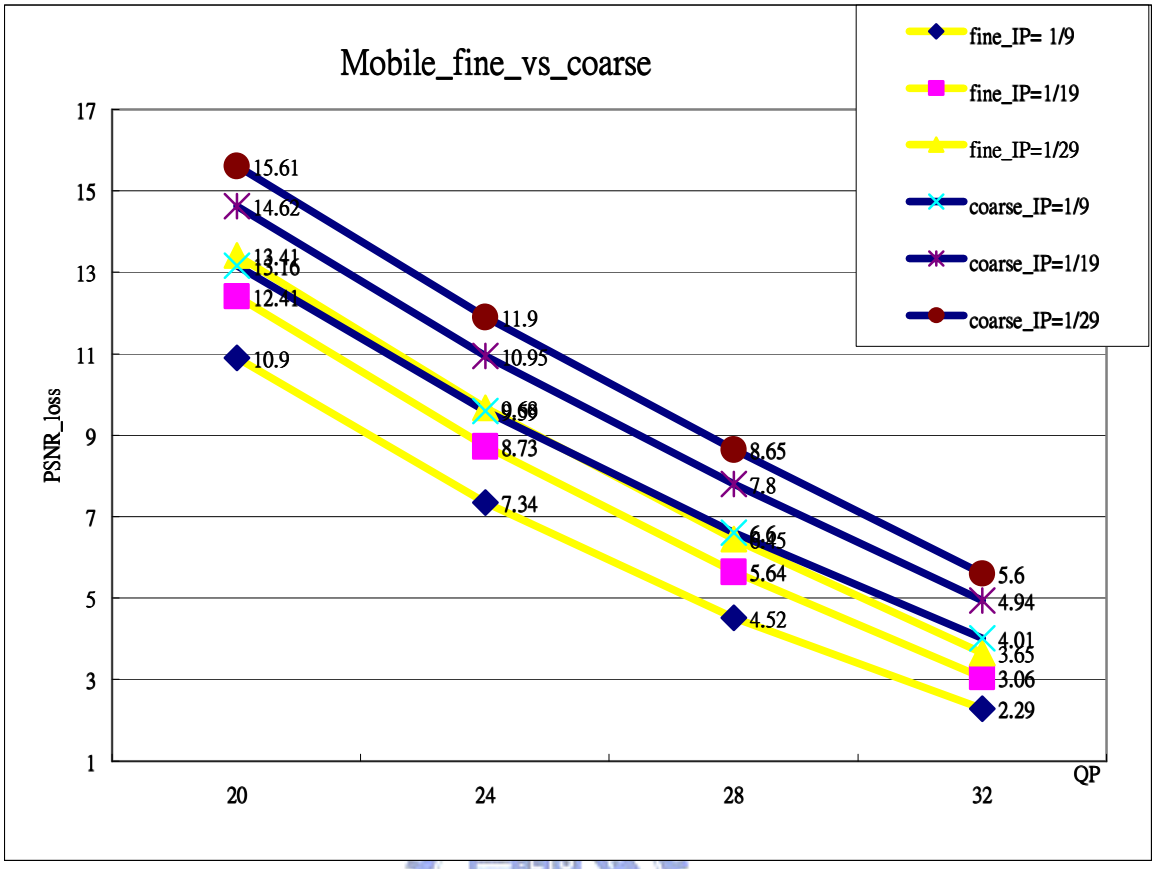


Fig. 27 PSNR loss results different QP and different GOP (Mobile)

Fig. 26 and Fig. 27 show the results of PSNR loss considering different QP and different GOP. We can see that the PSNR loss increases with the increasing GOP while tail off at higher QP values.

According to our simulation results over sequences Akiyo, Foreman, Mobile, Stefan, GOP 10, 20, 30, and QP 20, 24, 28, 32, the average difference in quality between using CGBPZ and FGBPZ is 1.5 dB. This number shows that CGBPZ is a good trade off between complexity and quality. 1.5dB PSNR drop enables the fast encoding procedure form over 30 cycles (FGBPZ) into 2 cycles (CGBPZ).

3.5.2 CGBPZ versus MHT

In this section, we focus on the performance between coarse bit-plane zonal

coding (CGBPZ) and modified Hadamard transform (MHT). CGBPZ is what we use as hardware implementation and system integration. Considering the requirement of high speed processing, we compare CGBPZ with MHT work. Fig. 28 shows the embedded result on Foreman with group of picture (GOP) as 20. The proposed DCT with CGBPZ has better performance and can efficiently slow down the speed of decade compared with MHT work. Fig. 29 also shows the drift effect but the experimented sequence is Mobile Calendar.

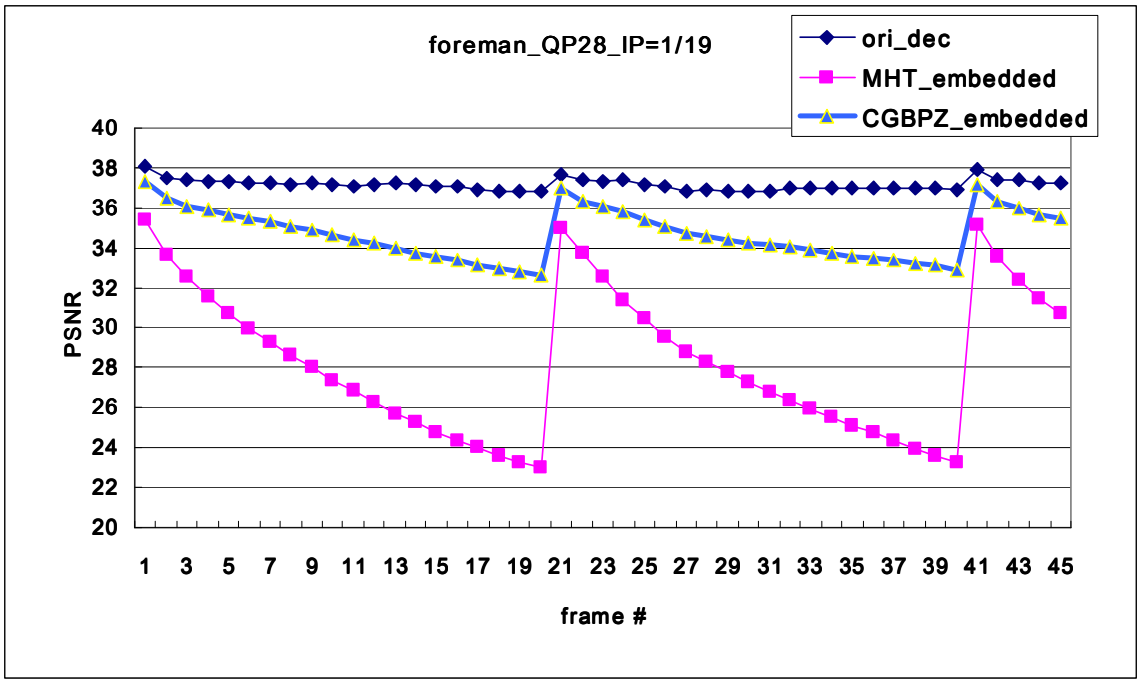


Fig. 28 Drift effects on Foreman_QP28_GOP20

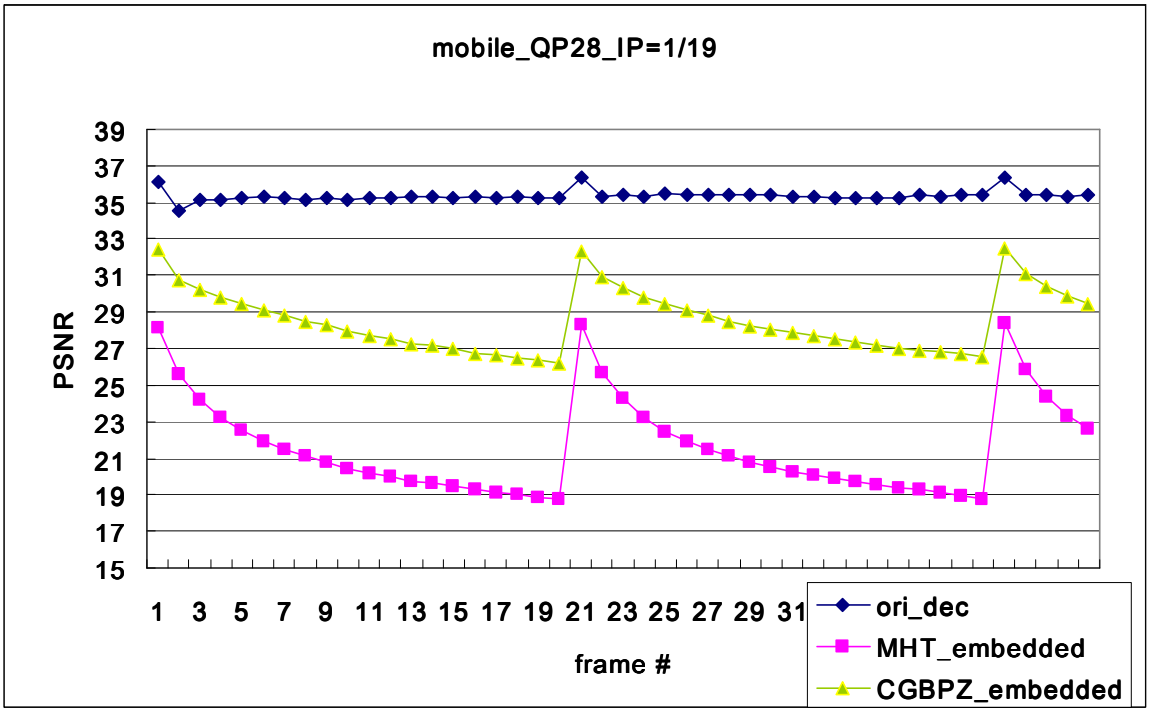


Fig. 29 Drift effects on Mobile_QP28_GOP20

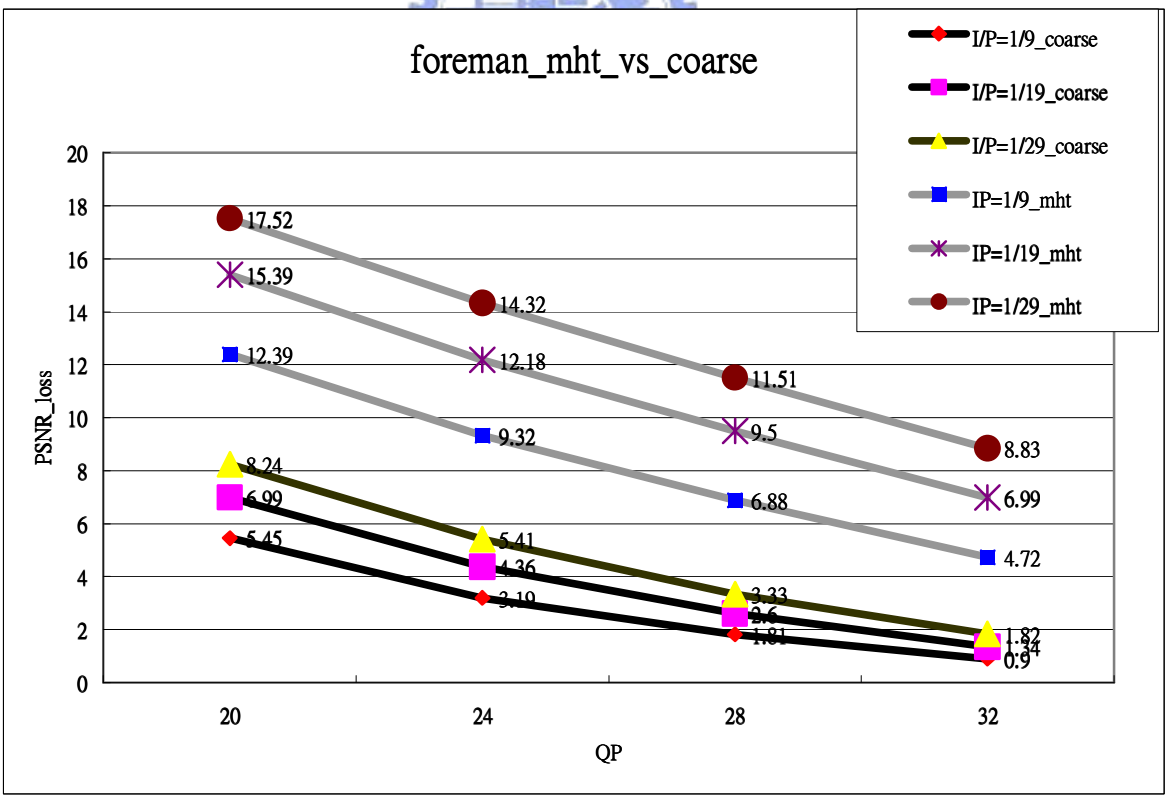


Fig. 30 PSNR loss results different QP and different GOP (Foreman)

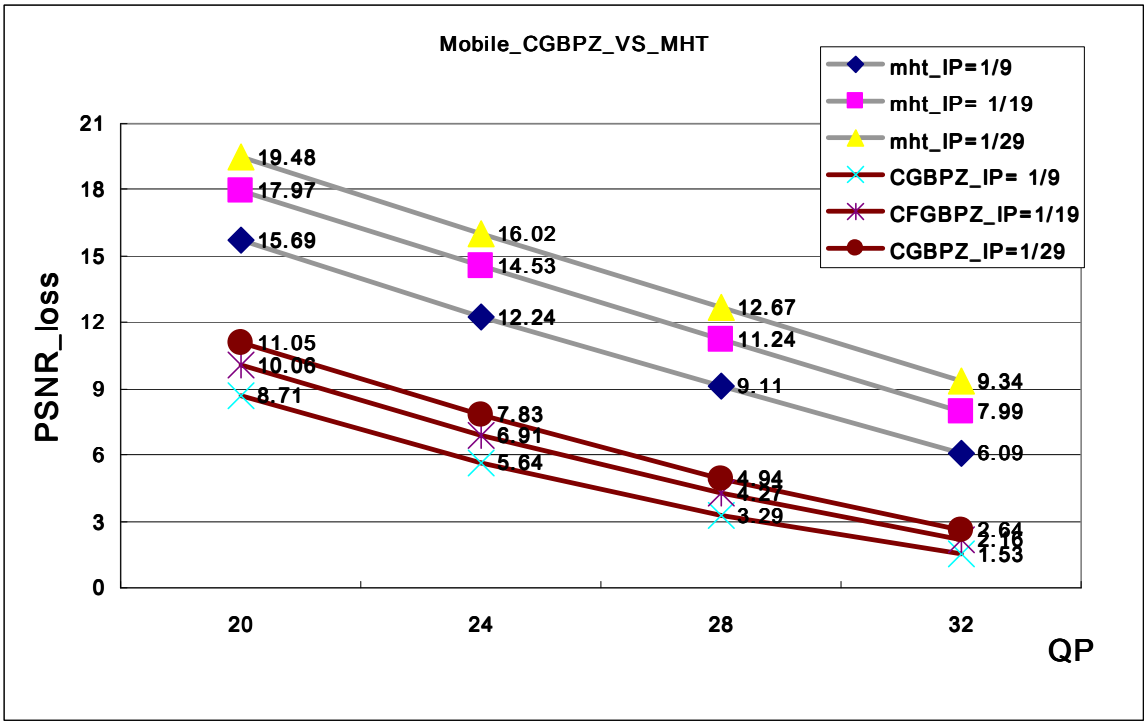


Fig. 31 PSNR loss results different QP and different GOP (Mobile)

Fig. 30 and Fig. 31 show the results of PSNR drop considering different QP and different GOP. According to our simulation results over sequences Akiyo, Foreman, Mobile, Stefan, GOP 10, 20, 30 and QP 20, 24, 28, 32, the average quality difference between DCT plus CGBPZ and MHT is 7.12 dB. This number shows the coding efficiency of proposed algorithm is much better than MHT.

Chapter 4
Proposed Embedded Compressor/Decompressor Architecture

In section 4.1 and 4.2, we will introduce our hardware design of proposed embedded compressor and decompressor respectively. The architectures are designed to fit the specification in chapter 6.1.

4.1 Architecture of Encoder Design

Overall block diagram of embedded compressor is shown in Fig. 32.

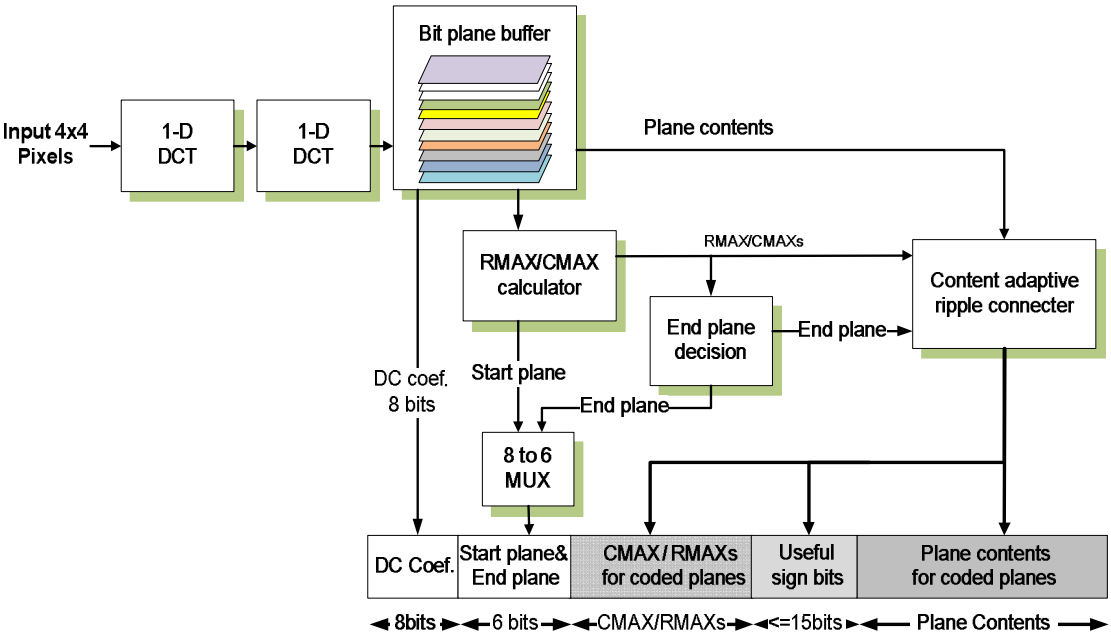


Fig. 32 Overall block diagram of embedded compressor

4.1.1 The Architecture of Two Dimensions Discrete Cosine Transform

The hardware design of DCT is referred from Lee's architecture [16]. This architecture can maintain the same performance with original DCT while reduced the number of multiplications to about half of those required by the existing efficient algorithms. This design allows us to take the advantage of DCT while not suffering from its hardware complexity. Notice that in Table 3, [16] uses more multiplications than [17] when applying 4 points DCT. However, the two inputs of multiplications in [16] are formed by one constant and one variable number while the inputs of multiplications in [17] are formed by two variable numbers. According to our experience, the synthesis area of multiplications which has one constant input is about 1/3 comparing to the synthesis area of multiplications which has two variable numbers. Therefore, design [16] we referred still gets the better of design [17] when applying 4 points DCT.

4.1.2 The Architecture of Coarse Grain Bit-Plane Zonal Encoding and Data Packing

There is a combinational block dealing with coefficients to derive the RMAX/CMAX and plane content of each plane. To serialize the plane information in one cycle, we propose the content adaptive ripple connector to solve the problem. The basic concept is shown in Fig. 33. The 10 lines at left represent the 9 plane contents pulsing 1 sign bit-plane content. Each connecter represents a shifted-outcome generator and a 16 to 1 MUX controlled by 4 bits RMAX/CMAX. It is shown in Fig.

34. By the ripple behavior, the wire at the end of the flow is the connected result. Notice that we embed this embedded compressor into our 100MHz decoder, thus one cycle is enough to finish our ripple processing.

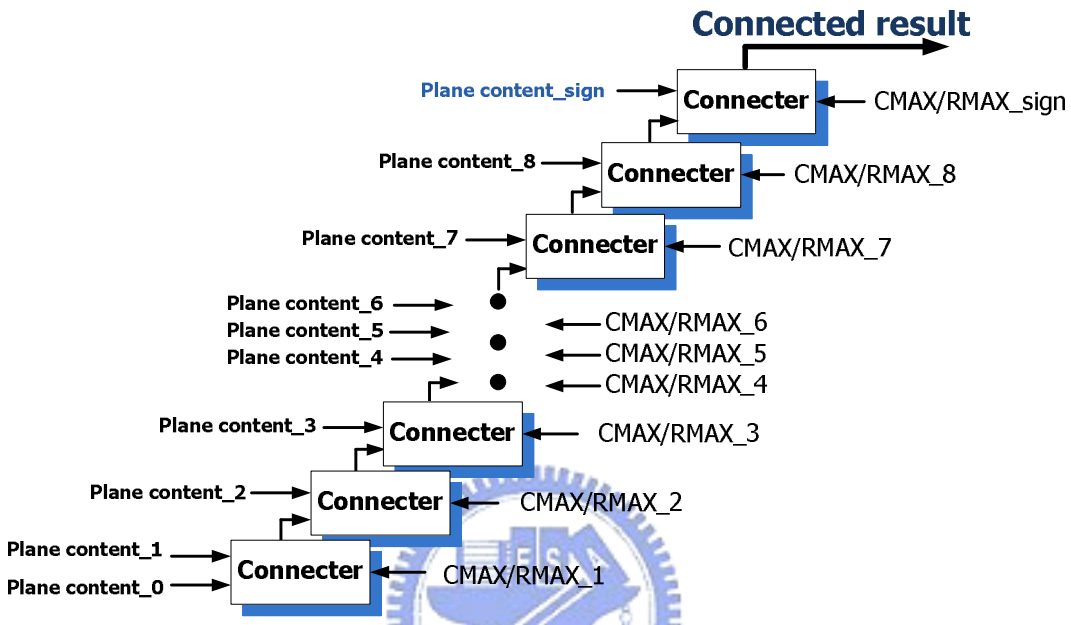


Fig. 33 Content adaptive ripple connecter

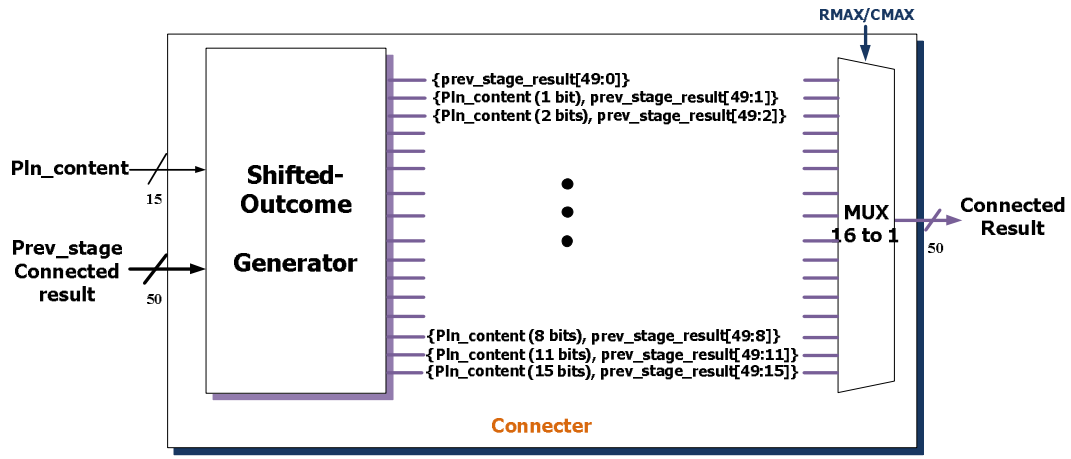


Fig. 34 The architecture of a single connecter in Fig. 33

4.1.3 The Architecture of End Plane Calculation

In CGBPZ, we separate the AC coefficients into sign bit-plane and magnitude

bit-planes. The end plane is important since it dominates the data packing process. We propose the architecture in Fig. 35. We unfold whole loop shown in Fig. 21. This architecture may increase the gate count but enable us to finish end plane decision in one cycle. It can be considered as a 6 bits ripple-adder plus a comparator.

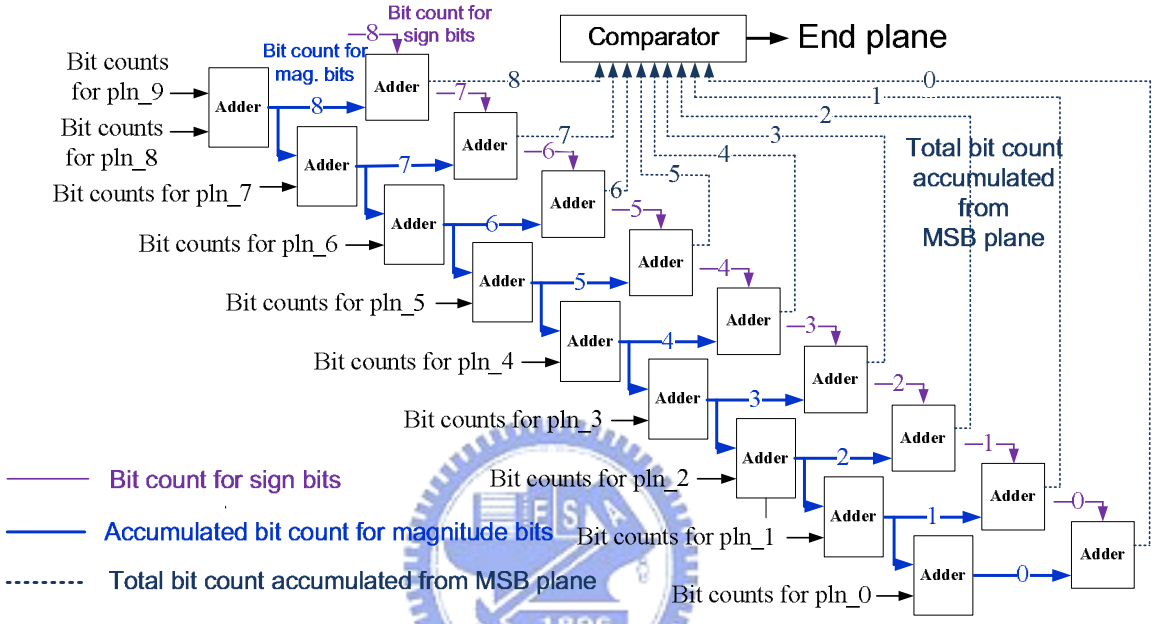


Fig. 35 The architecture of end plane calculation

4.1.4 Overall Encoder Design

Fig. 36 shows the pipeline architecture of compressor design. Since compressor has more time to handle the encoding process, we use three stages here and each stage has 4 cycles. Since the deblocking filter output 4 pixels per cycle and the 2 stages, 8 cycles are used for calculating a 4x4 DCT, one 1-D, 4 points DCT functional block for each pipeline stage is enough.

Under this design, a MB needs 72 cycles to encode. First 4x4 block takes 12 cycles and rest of 15 blocks take 4 cycles.

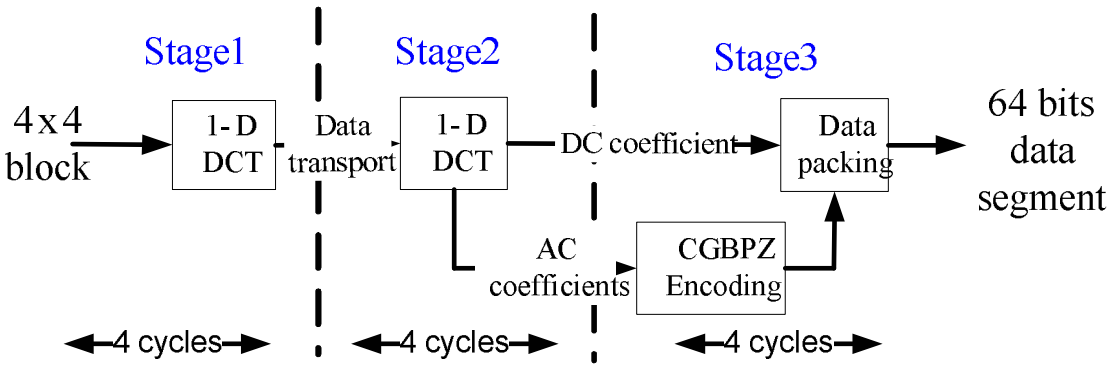


Fig. 36 Overall encoder design

4.2 Architecture of Decoder Design

Overall block diagram of embedded decompressor is shown in Fig. 37.

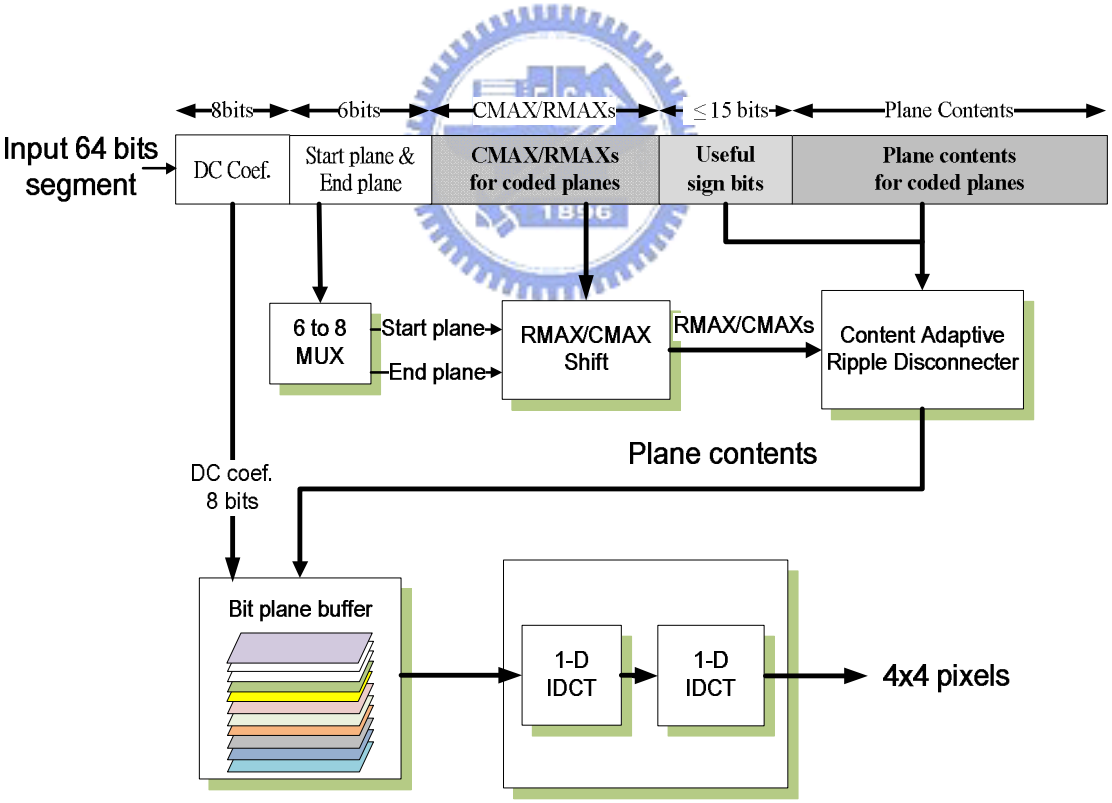


Fig. 37 Overall block diagram of embedded decompressor

4.2.1 Architecture of Data Unpacking, Bit-Plane Zonal Decoding and Compensation

Data unpacking can be considered as a reverse process of encoding. The mainly different is that the information is ready and less calculation is needed at decoder, so the decoder just needs to put the data back into correct positions. Therefore, decoder leads to smaller gate count comparing to encoder.

According to the coding format, the DC coefficient, start plane and end plane are fixed at the beginning of the coded segment and are easy to decode. With the help of start plane and end plane, we can split the union of RMAX/CMAX by a simple MUX. Again, we use adaptive ripple architecture. With the help of RMAX/CMAX of each bit-plane, the content adaptive ripple dis-connector can be applied and the AC coefficients can be pieced together.

After coefficients reconstructed, compensation is applied. There is a MUX controlled by the end plane flag. If Nth plane is the end plane, the (N-1)th plane will be filled with 1 if the magnitude of the position is not zero.

4.2.2 Architecture of Two Dimensions Discrete Cosine Transform

The hardware design of two dimensions DCT in decoder is the same as in encoder. Notice that the DCT unit deals with 4 pixels per time. In decoder design the timing is very critical, therefore four sets of 1-D, 4 points DCT units are needed.

4.2.3 Overall Decoder Design

Fig. 38 shows the pipeline stage of decompressor. To fast provide data for motion

compensation unit, the decompressor must support higher throughput. The decompressor is divided into two stages and each stage needs 2 cycles. A 4x4 block needs 4 cycles to be decoded. Decoding a MB just needs 34 cycles.

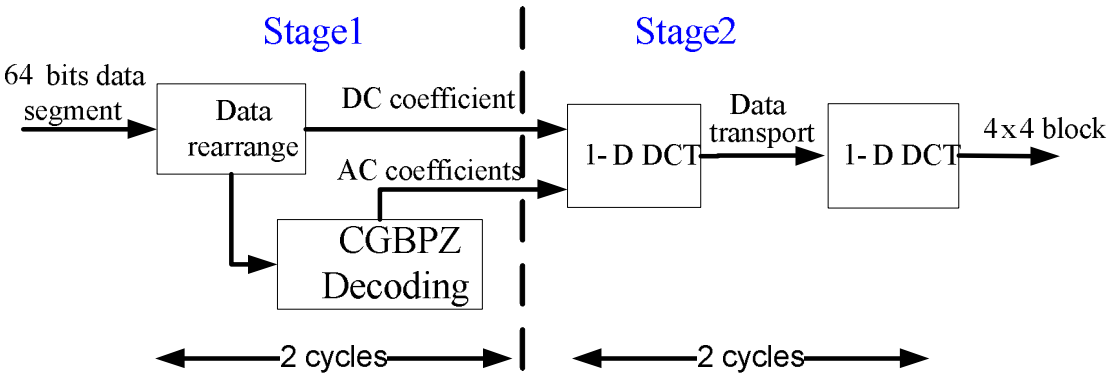


Fig. 38 Overall decoder design

Chapter 5
Design Implementation and Verification

5.1 Design Implementation

In this thesis, we proposed a flexible algorithm which achieves good coding efficiency and is suitable to integrate with any video decoder. The proposed architecture is synthesized with UMC 90-nm CMOS standard-cell library. The operation frequency is 100 MHz. The gate counts of proposed algorithm for compressor/decompressor are 15.8K/14.2K respectively.

Embedded encoder is divided into 3 pipeline stages and each stage cost 4 cycles. Although the pipeline stage of encoder can be shorten to 2 cycles, but considering that the de-blocking filter needs 4 cycles to completely output a 4x4 pixel block. To integrate with the original system with out any extra buffer, four cycles per stage is a better choice. At the same time, longer cycle per stage can reduce the number of 1-D, 4-points DCT functional blocks and can decrease the area.

Embedded decompressor is divided into 2 pipeline stages and each stage costs 2 cycles. The minimum pipeline stage design is one cycle per stage, 3 stages total (2 for 2-D DCT, one for CGBPZ decoder). But this minimum design requires 64 bits bus bandwidth. Therefore 2 stages, 2 cycles each is the fast design in system with 32 bits bus bandwidth.

A summary of hardware design is given at Table 7.

Table 7 Summary of hardware design

	Proposed EC	
Function part	Compressor	Decompressor
Synthesis process	UMC 90nm	
Operate Frequency	CIF@5MHz HD1080@100MHz	
Latency/MB	72 cycles	34 cycles
Gate Counts	15.8K	14.2K
Power	2.78mW	1.66mW

5.2 Design Verification

The flow of design verification is shown in Fig. 39. The verification can be considered as two parts. One is software and the other is hardware. Patterns are generated by software and applied as the input of hardware. At the same time, the software calculates the correct answer and compares the result with hardware's result. And then, the result is stored in memory. Again the coded data is accessed by software decompressor and hardware decompressor. The decoded data is checked to confirm the result is met on software and hardware.

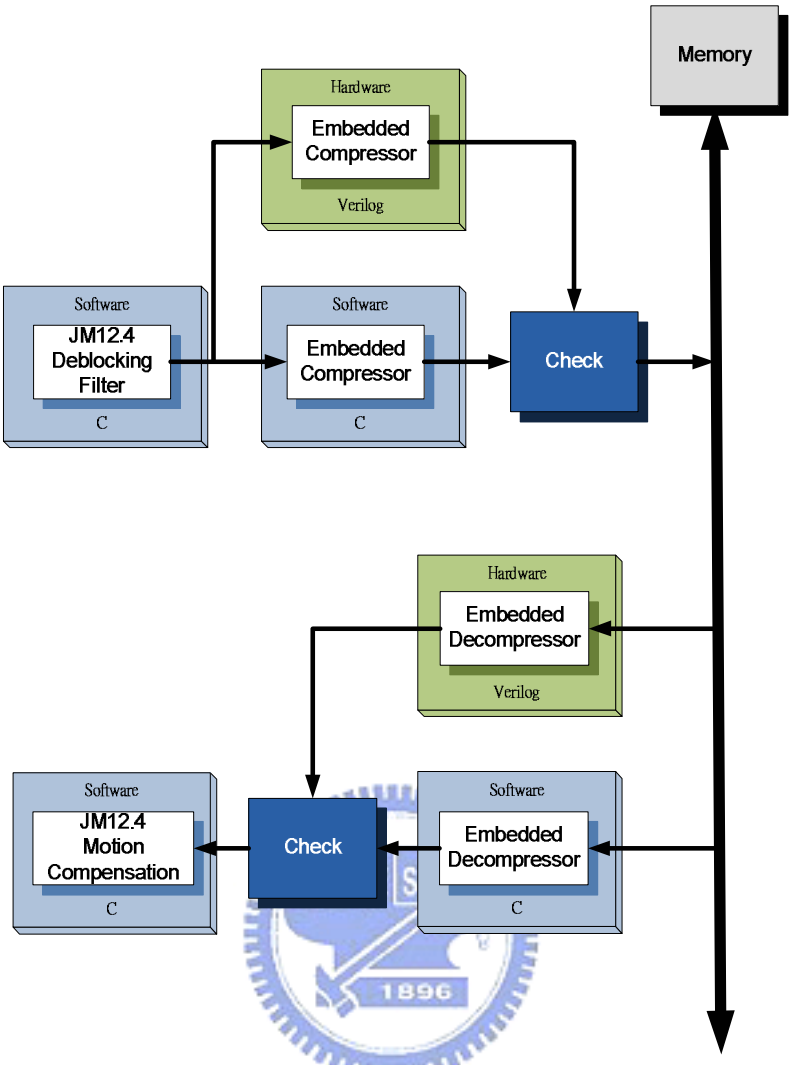


Fig. 39 The flow of design verification

Chapter 6
System Integration and Experimental Results

In section 6.1, we will first introduce the specification of SI2 low power H.264 decoder. The problems occurred during integration will also be discussed. The detail analysis will be given in section 6.2.

6.1 System Analysis

The overall system block diagram is shown in Fig. 40. Our H.264 decoder works at 100 MHz, performing HD1080 at 30frames/per second. The embedded compressor compresses the data from deblocking filter. 4x4 blocks will become 64 bits segments and then stored into off-chip memory. The embedded decompressor decompresses coded segments from external memory and sends to motion compensation unit (MC). The system bus bandwidth is as 32 bits and the external memory is 32 bits per entry.

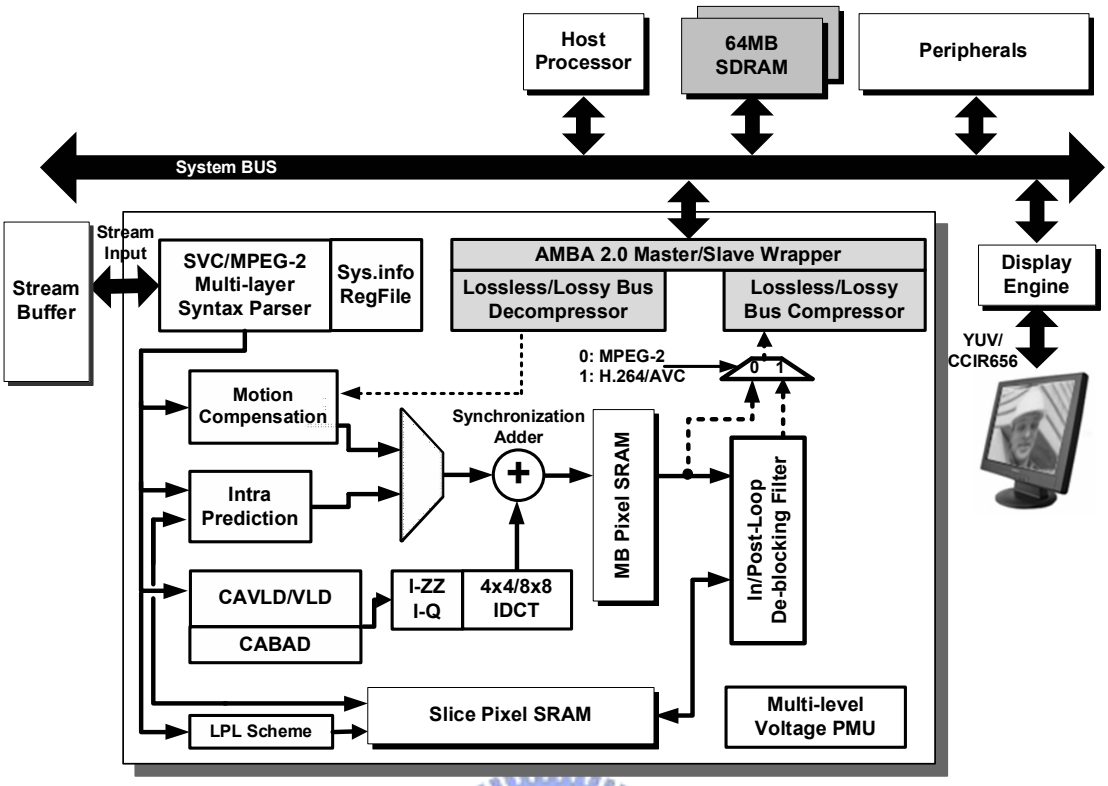


Fig. 40 The overall system block diagram

6.1.1 Interface

The embedded compressor can be considered as an interface between the chip and the external memory. Fig. 41 is the system interface design for embedded codec.

The output speed of deblocking filter is 4 pixels per clock, thus the best processing clocks for each pipeline stage of embedded compressor must less or equal to 4 cycles to avoid the traffic jam at the input of embedded compressor. Another interface issue occurs at the input of motion compensation (MC). The data provider of MC switch form external memory to the proposed embedded decompressor. The input bandwidth of MC in original system is 4 pixels per cycle, so the basic requirement is that the embedded decompressor must output at least 4 pixels per cycle.

Finally, an address converter will be needed. Since the compression ratio is fixed at two, the address converter is easy to implement.

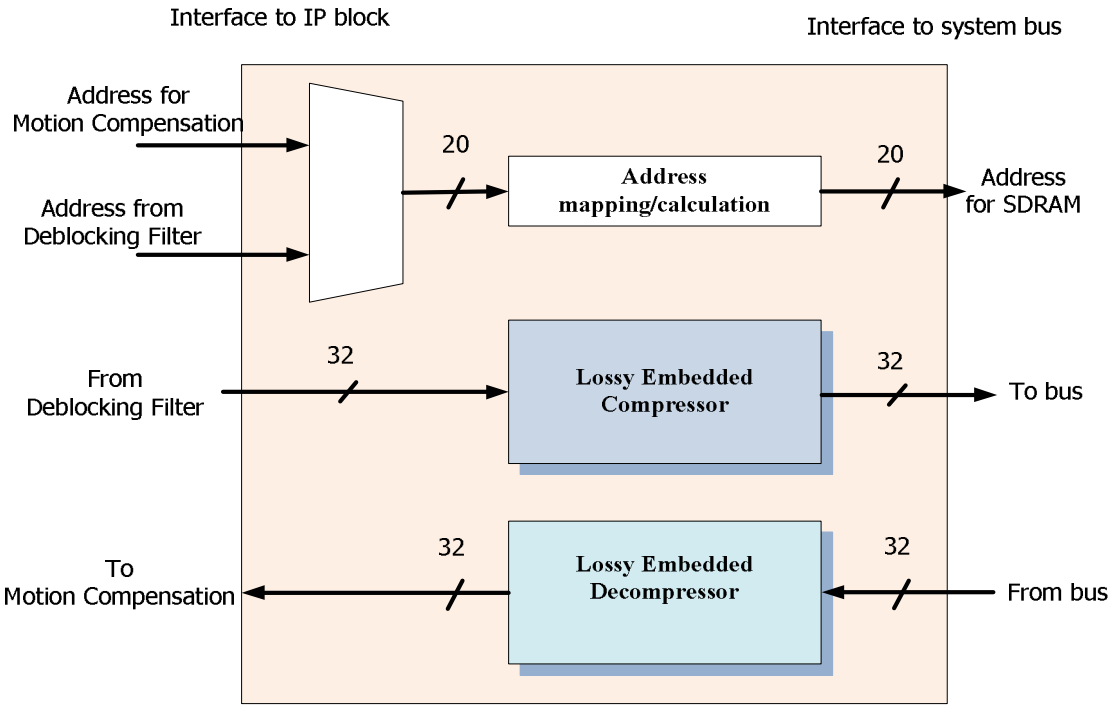


Fig. 41 System interface design for embedded codec.

6.1.2 Overhead Problem

As we introduced in chap 3.1, embedded compressor suffered from overhead problem and the overhead ratio directly links to the coding unit. However, the overhead problem in our system is different with chap 3.1 since our system is 1x4 pixels array-based not pixel based. The access behavior of motion compensation with/without embedded compressor can be analyzed as follows. Here we simply analysis two cases: best case and worst case.

If the requested 4x4 blocks are perfectly aligned with the coded 4x4 blocks, only 2 cycles are needed to fetch the 4x4 block while the original system needs 4 cycles to fetch. This situation is illustrated in Fig. 42

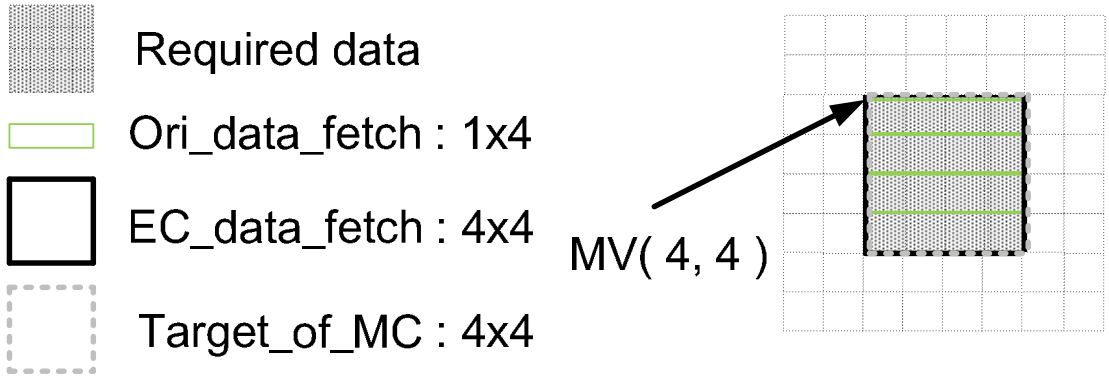


Fig. 42 Best case on data fetching

The worse situation is the sub pixel case. For motion vector (x, y), both x and y are not integers. Therefore a 4x4 block needs a 9x9 pixels block to finish the motion compensation. 18 cycles is needed for embedded compressor while original system needs 27 cycles. Fig. 43 shows the analysis above. Full case analysis will be given in section 6.2.1. In chapter 6.2, we can see that H.264 decoder with an embedded compressor does reduce the access times and can efficiently reduce the access power consumption.

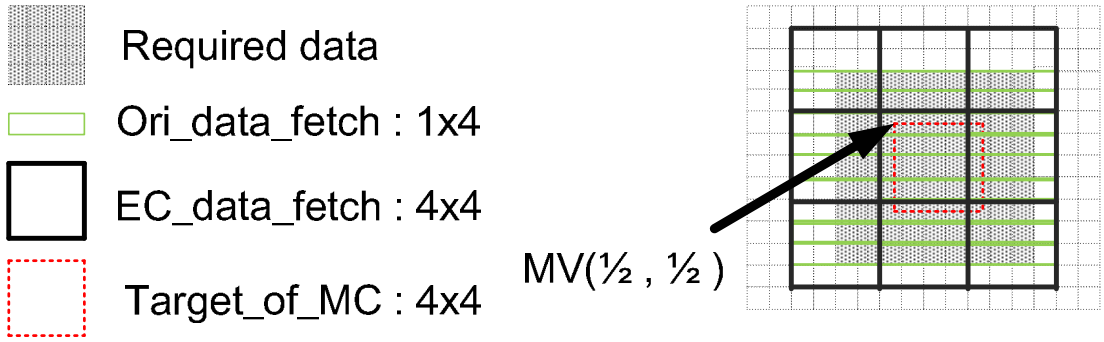


Fig. 43 Worse case: sub pixel case

6.1.3 Processing Cycles Problem

The third part of system analysis is the problem of processing cycle. The existence of this problem is due to the tight processing cycles of our low power H.264

decoder. Our H.264 decoder works at 100 MHz, performing the HD1080i@30fps. By a simple division, we can find that it is only 25 cycles for motion compensation to deal with a 4x4 block. Therefore we need a short-cycle design to turn down the loading on cycles of the embedded compressor. Detail analysis will be given in section 6.2.2.

6.2 System Integration

6.2.1 Access Reduction

Recall our motivation, we try to scarify some quality while achieve power reduction. In following section we will introduce how we reduced the access.

The correlated accesses of EC can be separated into two parts. One is the write accesses from deblocking filter which writing data to external SDRAM. Another part is the read accesses from motion compensation (MC) unit. First part is easy to be analyzed because the write accesses are formed by writing frames into SDRAM. The access times after adding EC (4x4 pixel-unit and CR=2.0) are always half of the original system (1x4 pixel array).

The read accesses requested from MC are much more complicate. Motion compensation unit requests data based on motion vector (MV). For further discussion, the value of x and y in motion vector (x, y) can be classified into 3 types: align, not align and sub pixel case.

1) **Align: the value is a quadruple. It can fit with the 4x4 coded block grid.**
2) **Not align: the value is not a quadruple but an integer. Needed 4 pixels may span two 4x4 block grids.**

3) Sub pixel case: the value is not an integer but accurate to ½ or ¼. It needs 9 pixels to be interpolated into 4 pixels.

In section 6.1.2, we already explain several cases of the access behavior between system with and without EC. Here we give the analysis of overall cases in Table 8. Notice that in all cases the access times with EC are always less than or equal to the access times of original system.

Table 8 Overall cases of read access requested by MC with/without EC

Case of MV (x, y)	Access for Original system	Access times for system with EC	Access times Reduced?	Probability of Each case (%)
(align, align)	**4**	**2**	**yes**	**33**
(align, not align)	**4**	**4**	**equal**	**0.4**
(align, sub)	**9**	**6**	**yes**	**5.1**
(not align, align)	**8**	**4**	**yes**	**4.5**
(not align, not align)	**8**	**8**	**equal**	**0.4**
(not align, sub)	**18**	**12**	**yes**	**5.4**
(sub, align)	**12**	**6**	**yes**	**23.5**
(sub, not align)	**12**	**12**	**equal**	**1.81**
(sub, sub)	**27**	**18**	**yes**	**25.8**

The probabilities of each case are derived by simulation over 4 sequences (Akiyo, Foreman, Stefan, Mobile Calendar), 300 frames each. These sequences are formed by GOP 30.

According to the probabilities, the average reduction achieved on read accesses is 40% of original accesses.

6.2.2 *Processing Cycles Problem*

In section 6.1.3, we had described the problem of processing cycles. In this section, we will show the feasibility of CGBPZ to integrate with our system.

Based on not to change our original system, we have two constrains here. First, the original system specification is HD1080@100MHz, 30 frames per cycles. This means the available cycles for each 4x4 block unit are 25 cycles. We hope to finish MC with proposed EC in 25 cycles. Second, we hope that after embedding EC, we won't change the data input mechanism of MC (sending data into MC continually).

The solution we used to solve constrain 2 is to add a new state into original states and to insert buffer between embedded decompressor and MC. The signal "MC read data enable" is putting off till the buffer has enough data to continually feed into MC.

However, this new state cost times. Now the required cycles of "MC data read states" is equal to the original process cycles of MC plus the new state "EC decode" like in (4). The full cases discussions of "EC decode" cycles plus original "MC data read" cycles are in Table 9.

$$process\_time(MC\_with\_EC) = delay(EC\_decode) + process\_time(Original\_MC) \quad (4)$$

Table 9 Full cases of “EC decode” cycles plus original “MC data read” cycles

Case of MV (x, y)	Num of Ref. block	Delay for EC decode	Processing time for Ori. MC	Total MC cycles for EC	Probability of Each case (%)
(align, align)	1	**4**	**4**	**8**	33
(align, not align)	2	**5**	**4**	**9**	0.4
(align, sub)	3	**5**	**9**	**14**	5.1
(not align, align)	2	**5**	**8**	**13**	4.5
(not align, not align)	4	**7**	**8**	**15**	0.4
(not align, sub)	6	**7**	**18**	**25**	5.4
(sub, align)	3	**6**	**12**	**18**	23.5
(sub, not align)	6	**9**	**12**	**21**	1.81
(sub, sub)	9	**9**	**27**	**36**	25.8

We can see that the new processing cycles in Table 9 of all case MC+EC are much less than 25 cycles except the (sub, sub) case and (not align, sub) case. By using the probabilities, we can calculate the average cycles used for MC+EC. The average cycles are 19.3 cycles. That means, although the (sub, sub) case uses more than 25 cycles, it is still fit the system timing constraint since there are available cycles from other modes. So, embedding proposed codec into original system is feasible.

6.2.3 Access Reduction Ratio

The access ratio of system with EC vs. original system is defined as (5):

$$\text{Access ratio} = \frac{\text{Mem\_read\_EC} + \text{Mem\_write\_EC}}{\text{Mem\_read\_Ori.} + \text{Mem\_write\_Ori.}} \quad (5)$$

According to our simulation, the ratio of read accesses with/without EC is 0.625,

and the ratio of write accesses with/without EC is fixed at 0.5. Also, we obtain the average access ratio of read/write in original system is about 3.51. The overall access ratio (with/without EC) can be calculated below (6):

$$\text{Overall access ratio} = \frac{0.625 \text{ x (Mem\_read\_Ori.)} + 0.5 * \text{(Mem\_write\_Ori.)}}{\text{Mem\_read\_Ori.} + \text{Mem\_write\_Ori.}} = \frac{0.625 \text{ x } (3.51) + 0.5 \text{ x } (1)}{3.51 + 1} = 0.596 \quad (6)$$

Therefore, the average reduction ratio on memory access is:

$$AVG \quad reduction \quad ratio = 1 - overall \quad access \quad ratio$$

$$= 1 - 0.596 = 40.4\%$$

6.2.4 Simulation Result on SDRAM Power Reduction

We choose the system-power calculator [21] as external memory power model and the parameter setting is according to [22]. We simulated the memory using on CIF@50MHz and HD1080@100MHz. The results are shown in Fig. 44 and Fig. 45. Each figure includes the core power of H.264 decoder, SDRAM background power and SDRAM access power (R/W) operated on different frequencies. The power saving on performing CIF is 7.6mW while the power saving on performing HD1080 is 154.8mW. It does make sense since the average available cycles for a 4x4 block on both video formats are the same and the access ratio on R/W is slightly different due to different test sequence. Therefore it is reasonable that the power reduction is almost directly proportional to the frame size.

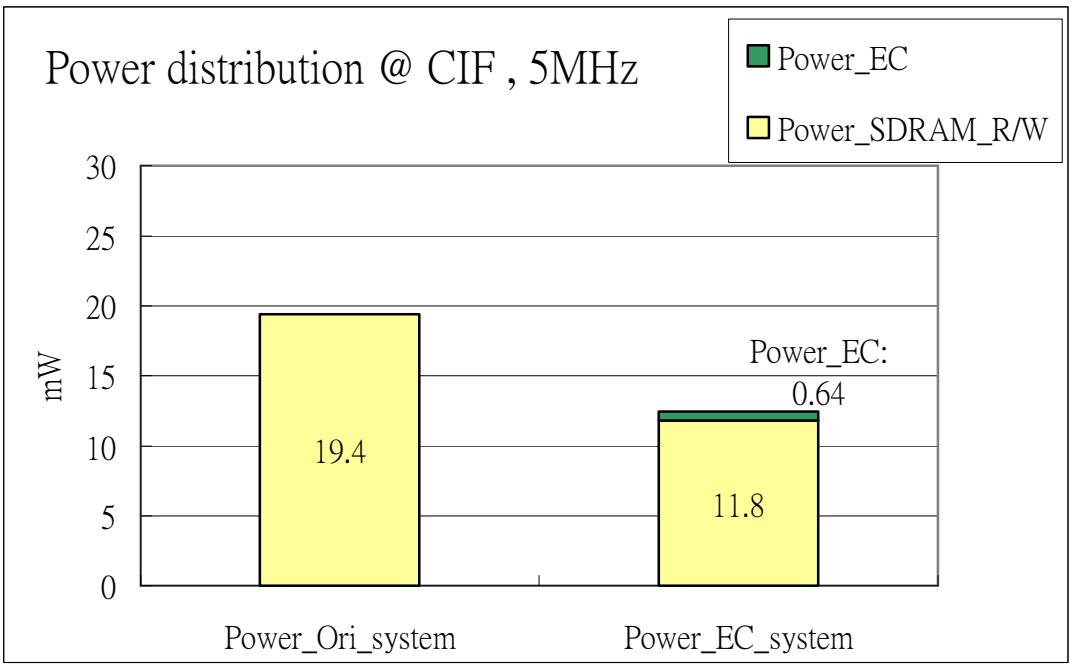


Fig. 44 Power analysis on CIF @ 5.3MHz

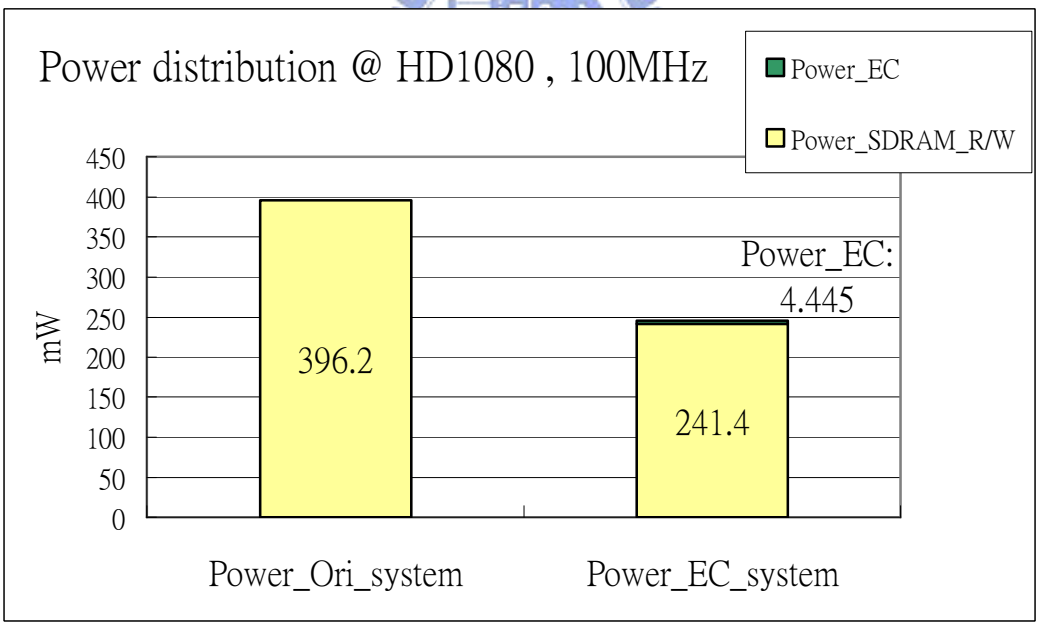


Fig. 45 Power analysis on HD1080 @ 100MHz

Chapter 7
Conclusion and Future Work

7.1 Conclusions

In this thesis, we proposed a flexible algorithm which has good coding efficiency and is suitable to integrate with any video decoder. With the help of this recompression engine, we can reduce the bandwidth requirement and the external frame memory and reduce the data access times to achieve the goal of power saving. The fixed compression ratio makes this extra function easily be integrated with a system by adding a simple address controller. The proposed architecture is synthesized with 90-nm CMOS standard-cell library. The operation frequency is 100 MHz. The gate counts of proposed algorithm for compressor/decompressor are 15.8K/14.2K respectively. The proposed architecture costs 30K gate counts and deals with a 4x4 block unit while previous MHT work costs 20K gate counts in dealing with a 1x8 pixels array. The proposed algorithm not only gains 7.12dB in the quality but also achieves an area-efficient hardware implementation. The peak power consumption of proposed embedded codec @ 100MHz is 4.445mW.

7.2 Future Work

Future works are formed by three parts. First is about the coding efficiency. From proposed FGBPZ to CGBPZ, we made a trade off between the encoding/decoding cycles and the visual quality. CGBPZ achieves fast coding speed and acceptable

visual quality. However, error propagation delivered through 29 P frames is still noticeable by human eyes. To embed a compressor/decompressor in video decoder, the only way to get better visual quality is to reduce the quality loss of each referred frame. Therefore, refine coding scheme to reduce quality loss is very importance.

Second part is to develop adaptive compensation modes according to different characteristics of video sequences. The compensation method we proposed is based on universal behaviors of our testing data base. Proposed compensation method reaches minimum average PSNR loss over our data base. But we also found other kinds of methods have better performance on compensating certain video sequences while having poor performance on the others. That is why we wish to develop adaptive compensation modes. To compensate sequences according to their characteristics can optimal the individual visual quality and is also a direction of quality improvement. Since the power consumption of our embedded codec is much less than the power we reduced on access reduction, to increase reasonable complexity to get better performance is a good idea and is worth us to try.

The final part is about the memory power model. The memory power model we use is to estimate the memory power consumption according to the average behavior of data access ratio [21]. For a detail analysis, there are some factors needed to be specified such as page mode design and burst length. The burst length determines the efficiency of memory data read accesses and page mode determinates the hit rate. Power consumption on memory is related to those factors. If we can build a power model taking those factors into account, we can analyze our memory power consumption in a more accurate way.

References

[1] M. Li, R. Wang and W. Wu, "The High Throughput and Low Memory Access Design of Sub-pixel Interpolation for H.264/AVC HDTV Decoder," IEEE SIPS'05, pp. 296-301, May 2005.

[2] R. Manniesing, R. Kleihorst1, R. V. Vleuten1, and E. Hendriks, “Implementation of lossless coding for embedded compression,” IEEE ProRISC, 1998.

[3] T. Y. Lee, “A New Frame-Recompression Algorithm and its Hardware Design for MPEG-2 Video Decoders,” IEEE Trans. CSVT, vol. 13, no. 6, pp. 529-534, June 2003.

[4] Yongie Lee, et al, "A New Frame Recompression Algorithm Integrated with H.264 Video Compression," IEEE Circuits Sys. ISCAS Vol. 6, pp.6110-6113, May 2007.

[5] H. Shim, N. Chang, and M. Pedram, “A compressed frame buffer to reduce display power consumption in mobile systems,” in Proceedings of the Asia and South Pacific Design Automation Conference, pp. 819–824, 2004.

[6] U. Bayazit, L. Chen, and R. Rozploch, “A novel memory compression system for MPEG2 decoders,” in IEEE International Conference of Consumer Electronics, pp. 56–57, 1998.

[7] M. Schaar-Mitrea and P. With, “Near-lossless embedded compression algorithm for cost reduction in DTV receivers,” in IEEE International Conference of Consumer Electronics, pp. 112–113, 1999.

[8] S. Lei, “A quad-tree embedded compression algorithm for memory-saving DTV decoders,” in IEEE International Conference of Consumer Electronics, pp. 120–121,1999.

[9] E. G. T. Jaspers and P. H. N. With, “Embedded compression for memory resource reduction in MPEG systems,” in IEEE Benelux Signal Processing Symposium, 2002.

[10] G. M. Callico, A. Nunez, R. P. Llopis, and R. Sethuraman, “Low-cost and real-time super-resolution over a video encoder ip,” IEEE, 2003.

[11] R. Bruni et al., “A novel adaptive vector quantization method for memory reduction in MPEG-2 HDTV decoders,” in Proc. Int. Conf. Consumer Electronics, pp. 58-59, 1998.

[12] R. Dugad and N. Ahuja, “A Fast Scheme for Image Size Change in the Compressed Domain,” IEEE Trans. CSVT, vol. 11, no. 4, pp. 461-474, April 2001.

[13] D. Pau et al., “MPEG-2 Decoding with a Reduced RAM Requisite by ADPCM Recompression before Storing MPEG Decompressed Data,” U.S. patent 5838597, Nov. 1998.

[14] C. C. Cheng, P. C. Tseng, C. T. Huang, and L. G. Chen, "Multi-Mode Embedded Compression Codec Engine for Power-Aware Video Coding System," in IEEE, SIPS 2005.

[15] A. Bourge and J. Jung, “Low-Power H.264 Video Decoder with Graceful Degradation,” *SPIE Proc. Visual Communications and Image Processing*, vol. 5308, pp. 372-383, Jan. 2004

[16] Byeong Lee, “A new algorithm to compute the discrete cosine Transform” Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on Volume 32, Issue 6, Page(s):1243 – 1245 Dec 1984

[17] Wen-Hsiung Chen et al., “A Fast Computational Algorithm for the Discrete Cosine Transform”, Communications, IEEE Transactions on [legacy, pre - 1988], Volume 25, Issue 9, Page(s):1004 – 1009, Sep 1977

[18] R. J van der Vleuten *et al*, “Low-complexity scalable DCT image compression”, *IEEE Proc. Image Processing*, vol.3, pp. 837-840, Sep. 2000

[19] Yao-Chun Fang; Chun-Yi Lee; Yin-Ming Wang; Chung-Neng Wang; Tihao Chiang; “Low complexity lossless video compression”, ICIP 2004 on Volume 4, 24-27 Page(s):2519 - 2522 Vol. 4 Oct. 2004

[20] Chieh-Hsien Cheng, “An Embedded Compressor/De-compressor for Video Decoder Using VQ/DCT Hybrid Coding”, master thesis, 2007

[21] Micron® Technology Inc. The Micron® System-Power Calculator: SDRAM. [Online Available]: http://www.micron.com/products/dram/syscalc.html

[22] Micron® Technology Inc. MT48LC2M32B2 64Mb SDRAM. [Online Available]: http://www.micron.com/products/dram/

作　者　簡　歷

姓名　　：吳昱德

戶籍地　：台灣省台中市

出生日期：1984. 02. 01

學歷：　1999. 09 ~ 2002. 06　國立台中第一高級中學

2002. 09 ~ 2006. 06　國立交通大學　電子工程學系

2006. 09 ~ 2008. 07　國立交通大學　電子工程研究所碩士班