

國立交通大學

電子工程學系 電子研究所碩士班

碩 士 論 文

適用於 H.264 HDTV 解碼器之有效率動作補償記憶體架構



**A Bandwidth-Efficient Motion Compensation Memory**

**Organization for H.264 HDTV Decoder**

學生：侯康正

指導教授：李鎮宜教授

中華民國九十五年七月

適用於 H. 264 HDTV 解碼器之有效率動作補償記憶體架構

## A Bandwidth-Efficient Motion Compensation Memory

### Organization for H.264 HDTV Decoder

研究生：侯康正

Student : Kang-Cheng Hou

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



Submitted to Department of Electronics Engineering & Institute of Electronics  
College of Electrical and Computer Engineering  
National Chiao Tung University  
in Partial Fulfillment of the Requirements  
for the Degree of  
Master of Science  
in

Electronics Engineering

July 2006

Hsinchu, Taiwan, Republic of China

中華民國 九十五年 七月

# 適用於 H.264 HDTV 視訊標準之動作補償記憶體架構


學生：侯康正

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



在視訊解碼的領域中，H.264/AVC 是由 ITU-T/ISO 所提出新的視訊標準。由於 H.264/AVC 高壓縮率以及視訊品質的提昇，使得 H.264/AVC 是目前最受歡迎的標準。由於數位電視的風行，而主要著重於視訊資料品質的 H.264 Main Profile 規格也漸漸被重視。因此對於高畫面的解析度上高速解碼及畫面品質的提昇是最重要的挑戰。而視訊解碼標準中，動作補償（Motion Compensation）架構一向是最重要的部份也是整個系統的核心所在。在 Main Profile 規格裡面爲了加強視訊品質而加入新的功能，例如：雙向預測、加權預測、直接預測編碼。在本論文中，針對支援 Main Profile 規格設計動態補償架構以符合高速的規格以及高品質的解析度。對於 H.264 解碼器中我們提出了有效的動作補償模組來提昇速度。此外，對於動作補償中小數點內插器的架構，我們提出了 2D 擴充行列式方法（Extended-2D Column Major Approach）以及新的架構來有效率的降低解碼器所需要的頻寬以及降低整個小數點內插器的複雜度。新的架構也合併了兩種 block 格式(Luma 以及 Chroma) 的小數點內插器，使得這兩種格式都可以使用相同的架構。我們所提出的小數點內插器架構比起其他的架構可以有效降低複雜度 20%，而且所需要的頻寬可以降低約 50%~60%來提昇動作補償的速度。

此外，動作補償系統需要從外部記憶體 SDRAM 存取大量資料，因此對於外部記憶體的存取是影響整個解碼器系統的關鍵。增加外部 BUS 上的頻寬將會提昇動作補償的效能以及支援更大畫面的解析度。一般的記憶體控制器並沒有對於多媒體方面的各個模組作設計。在本篇論文中將會對於存取外部記憶體提出適用於 HDTV H.264 解碼器有效率的記憶體控制器。將可以支援解碼器中的模組對於外部存取的需求，例如：動態補償及去方塊效應濾波器(De-blocking filter)。實現彈性、高速的動態補償系統架構設計。另一方面，我們所提出的記憶體控制器可以支援 H.264 多重參考圖片的技術，並且可以有效率的利用單一記憶體即可存取所需要的參考圖片資料。我們所提出的記憶體架構可以提昇 BUS 利用率(Bus Utilization) 至 90%以上。透過我們所提出的動作補償架構以及記憶體架構，整個系統對於記憶體讀取的頻寬將會改善 40% ~ 50%。最後，我們解碼器系統 Throughput 在高位元率的壓縮比下可以也可以符合標準所訂定的 HDTV 規格。




# **A Bandwidth-Efficient Motion Compensation Memory Organization for H.264 HDTV Decoder**

Student : Kang-Cheng Hou

Advisor : Dr. Chen-Yi Lee

Department of Electronics Engineering  
Institute of Electronics  
National Chiao Tung University

## **ABSTRACT**

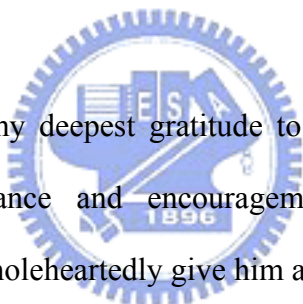
The logo of National Chiao Tung University is a circular emblem. It features a central shield with a book and a torch, surrounded by a gear-like border. The year '1896' is inscribed at the bottom of the shield.

H.264/AVC is the new video coding standard of ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MEPG). H.264 is most popular video standard due to high compress rate and better quality. In particular, the baseline profile of H.264/AVC has been accomplished progressively. In recently year, digital TV is widely adopted so that H.264's Main Profile focus on quality of video will be attended gradually. Therefore, the improvement of resolution and quality for large frame will become important issue. Motion compensation always is important module and kernel of system in video standard. For enhancing quality of video, H.264's main profile adopts new features such as Bi-prediction, weighted prediction and direct mode coding. In this thesis, a bandwidth-efficient motion compensation system is proposed for high definition resolution supported by main profile in H.264/AVC. Presently, we provide a novel structure of motion compensation system in main profile to improve system throughput. Furthermore, we propose Combined Luma/Chroma interpolator architecture in motion compensation and a novel data-reuse technique: Ectended-2D Column Major Approach. Both Luma and Chroma MB

can be interpolated by combined Luma/Chroma interpolator. A combined Luma/Chroma interpolator is proposed in order to save area, which achieves approximately 44% cost reduction. Additionally, an Extend-2D column major approach is presented, which improves 50% ~ 60% required bandwidth within decoder.

The video decoder should deal with large amount of data from external memory due to a real-time high-quality decoding demand. Therefore, both limited access time and bandwidth of memory access on BUS is bottleneck of entire video decoder. However, general memory controller may be not design for multimedia applications. In this thesis, the bandwidth-efficient memory controller architecture is proposed for H.264 decoder to increase limited bandwidth over external bus. The memory controller can support all module of H.264 decoder such as motion compensation and de-blocking filter, etc. Besides, the multiple reference pictures technique can be supported by our proposed memory controller, and can employ unique memory to store all required data for video decoder. About simulation results, the bus utilization can be improved up to 90% for our proposed memory controller. The bandwidth of memory access may be improved to 50% ~ 60% for entire video decoder adopting our proposed bandwidth-efficient motion compensation memory organization. Finally, the system throughput that is proposed by our proposed architecture can meet with specification with HDTV standard at high bit-rate.

# *Acknowledgements*

The logo of National Central University (NCTU) is a circular emblem with a gear-like border. Inside the circle, there is a stylized building and the year '1996' at the bottom. The letters 'NCTU' are prominently displayed in the center.

I would like to express my deepest gratitude to my advisor Dr. Chen-Yi Lee for his sophomore enthusiastic guidance and encouragement to overcome many difficulties throughout the research, and wholeheartedly give him and his family my best wishes.

Moreover, I would like to appreciate NSC for their financial support, my senior Si2 multimedia group mates for their discussions and comments during my research, especially for Yi-Hong Huang, and Jiun-Yan Yang. In addition, I would like to thank all members of Si2 group of NCTU for plenty of fruitful assistance and all engineers of CIC for their CAD supporting.

Finally, I give the greatest respect and love to my family and all my friends for their support and encouragement, and I want to express my highest appreciation. Sincerely, I hope them happy and happy forever.

# **Contents**

---

<b>Chapter 1</b>	<b>Introduction.....</b>	<b>1</b>
1.1	Motivation.....	1
1.2	Thesis Organization .....	2
<b>Chapter 2</b>	<b>Motion Compensation Algorithm of</b>	
	<b>H.264/AVC's Main Profile .....</b>	<b>4</b>
2.1	Profiling .....	4
2.2	Motion Compensation Process Flow .....	5
2.3	Inter Prediction Algorithm for H.264/AVC's Main Profile	
	.....	7
2.3.1	<i>Bi-directional Prediction</i> .....	8
2.3.2	<i>Multiple Reference Frames</i> .....	10
2.4	Motion Vector Prediction .....	12
2.4.1	<i>Traditional MV Prediction</i> .....	12
2.4.2	<i>Direct Mode Coding</i> .....	14
2.5	Fractional Interpolation.....	15
2.6	Weighted Prediction.....	17



2.6.1	<i>Explicit Mode</i> .....	18
2.6.2	<i>Implicit Mode</i> .....	20
2.7	Analysis.....	22
2.8	Comparison for MC of Previous Standards .....	26
2.9	Summary .....	27

## **Chapter 3 A Bandwidth-efficient Motion Compensation**

### **Architecture Design ..... 29**

3.1	Motion Compensation Engine for H.264/AVC's Main Profile.....	30
3.2	Motion Vector Predictor Design.....	32
3.2.1	<i>MVp Prediction Module</i> .....	33
3.2.2	<i>Direct Mode Coding Design</i> .....	36
3.3	Bandwidth-Efficient Fractional Interpolator Design.....	38
3.3.1	<i>Data Reuse Technique</i> .....	39
3.3.2	<i>Combined Luma/Chroma Interpolation Architecture</i> .....	44
3.3.3	<i>Simulation Results</i> .....	53
3.4	Weighted Prediction.....	55
3.5	Summary .....	56

<b>Chapter 4</b>	<b>Bandwidth-Efficient SDRAM Memory</b>	
	<b>Controller.....</b>	<b>58</b>
4.1	SDRAM Module Characteristics .....	59
4.1.1	<i>Basic Concept of SDRAM</i> .....	59
4.1.2	<i>Access Latency Analysis</i> .....	64
4.2	Memory Controller Organization.....	67
4.2.1	<i>Memory Access Scheduling</i> .....	68
4.2.2	<i>Memory Arrangement</i> .....	70
4.2.3	<i>Multiple Reference Prediction</i> .....	74
4.2.4	<i>Architecture of bandwidth-efficient Memory Controller</i> .....	76
4.3	Simulation Results .....	82
4.4	Summary .....	89
<b>Chapter 5</b>	<b>Chip Implementation.....</b>	<b>90</b>
5.1	Chip Specification.....	90
<b>Chapter 6</b>	<b>Conclusion .....</b>	<b>93</b>
<b>Bibliography</b> .....		<b>95</b>



# List of Figures

---

Figure 2.1 H.264 software (JM 9.2) profiling on ARM 7 processor .....	5
Figure 2.2 The general score of motion compensation for H.264/AVC's main profile.....	5
Figure 2.3 General structure of H.264 encoder. ....	6
Figure 2.4 General structure of H.264 decoder .....	7
Figure 2.5 Macroblock partitions and sub-macroblock partitions.....	8
Figure 2.6 Example using Bi-prediction: (a) previous/future (b) previous (c) future .....	9
Figure 2.7 The current block is predicted by $MV_{LO}$ and $MV_{LI}$ motion vector using Bi-prediction .....	10
Figure 2.8 Bi-prediction with multiple reference pictures .....	11
Figure 2.9 (a) directional prediction for 8 x 16 block size, (b) directional prediction for 16 x 8 block size, (c) median prediction .....	13
Figure 2.10 Direct mode prediction for B slices .....	14
Figure 2.11 (a) luma half sample with 6-tap FIR, (b) luma horizontal/vertical quarter sample with bilinear filter, (c) luma diagonal quarter sample with bilinear filter, (d) chroma sample with bilinear filter. Upper-case letters indicate the full samples and lower-case letter indicates the interpolated fractional samples ....	17
Figure 2.12 Bit rate value between baseline and main profile .....	23
Figure 2.13 PSNR between Baseline and Main profile.....	24
Figure 2.14 The proportion of integer/fraction motion vector for luma component in H.264/AVC main profile .....	25
Figure 2.15 The proportion of integer/fraction motion vector for chroma component in H.264/AVC main profile .....	25
Figure 3.1 The block diagram of H.264/AVC main profile decoder system.....	30

Figure 3.2 Motion compensation architecture for HDTV H.264/AVC main profile decoder .....	31
Figure 3.3 MV in shaded and oblique line region must be stored in row-FIFO. ....	32
Figure 3.4 Motion vector generator .....	33
Figure 3.5 Neighboring motion vectors required for decoding all motion vectors in current macroblock.....	34
Figure 3.6 (a) block size position index, (b) directional prediction table (16x8, 8x16), (c) median prediction table (16x16, 8x8), (d) median prediction table (4x4).....	35
Figure 3.7 Pre-scalefactor generator design .....	37
Figure 3.8 (a) Division free replacement (b) Multiplication-free replacement .....	37
Figure 3.9 (a) 4x4-block and 9x9 interpolation search windows for luma component interpolation (b) overlap region between neighboring blocks .....	38
Figure 3.10 (a) 2x2-block and 3x3 interpolation search windows for chroma component interpolation (b) overlap region between neighboring blocks .....	38
Figure 3.11 (a) row-major interpolating order (b) column-major interpolating order .....	39
Figure 3.12: Luma component interpolation: (a) Interpolating block 4 and (b) Interpolating block 3 (c) Interpolating block 6.....	42
Figure 3.13 Chroma component interpolation: (a) Interpolating 2x2-block 0 (b) Interpolation 2x2-block 1 .....	42
Figure 3.14 (a) Adder-chain based [23] (b) Adder-tree based [24] 1-D linear interpolator design .....	45
Figure 3.15 Separate 1-D interpolator design (no parallel).....	46
Figure 3.16 4-parallel separate 1-D luma interpolator with content buffer.....	48
Figure 3.17 Combined luma/chroma interpolator architecture .....	50
Figure 3.18 Combined luma/chroma interpolator unit .....	51
Figure 3.19 Combined luma/chroma FIR.....	52
Figure 3.20 (a)Process path for luma component interpolation (b) Process path for chroma component interpolation .....	52
Figure 3.21 Required bandwidth for different data-reuse approach.....	54

Figure 3.22 A weighted prediction block diagram .....	56
Figure 4.1 Modern SDRAM architecture .....	59
Figure 4.2 SDRAM resource utilization of several commands: (a) PRECHARGE (b) ACTIVE (c) WRITE (d) READ .....	62
Figure 4.3 Simplified bank state diagram.....	62
Figure 4.4 Burst read operation with CasLatency=3 and BurstLength=4.....	63
Figure 4.5 Burst write operation with CasLatency=3 and BurstLength=4. ....	64
Figure 4.6 Access latency for CL=2 (a) read access latency, (b) write access latency.....	65
Figure 4.7 (a) READ command with auto precharge, in the precharge period (tRP), SDRAM cannot issue another command in the same bank (ex: bank 0). (b) READ command without auto precharge, another command can be issued until the tRP is met. ....	66
Figure 4.8 Two un-scheduling and scheduling read memory accesses for bank-miss and row-hit.....	69
Figure 4.9 Luma block (a) one 16 x 16 block arrangement (b) one 4 x 4 block-0 arrangement for MB-0 .....	71
Figure 4.10 Chroma block (a) one 8 x 8 block arrangement (b) one 4 x 4 block consist of block 0-3 arrangement .....	71
Figure 4.11 (a) Co-located motion vector allocation in frame (b) corresponding bank for each 8x8 sub block.....	72
Figure 4.12 The pixels arrangement of one frame are stored of SDRAM. The arrangement of other banks is the same as bank0.....	72
Figure 4.13 The motion vector arrangement of one frame are stored of SDRAM. The arrangement of other banks is the same as bank0.....	73
Figure 4.14 Data amount of one decoding frame for different picture format.....	75
Figure 4.15 Architecture of bandwidth-efficient memory controller for H.264/AVC.....	76
Figure 4.16 Multi-channel address generator .....	77
Figure 4.17 Command and address queue and access status detection.....	79
Figure 4.18 The structure of bank controllers, master bank controller and timing unit.....	80

Figure 4.19 System level analysis relation .....	82
Figure 4.20 Fetching windows of 4x4 block between different burst length .....	84
Figure 4.21 Unscheduled Bus utilization, Data usage and Data utilization for different burst length in memory .....	84
Figure 4.22 Scheduled Bus utilization, Data usage and Data utilization for different burst length in memory. ....	85
Figure 4.23 The data utilization between un-scheduling and scheduling.....	85
Figure 4.24 Average access cycles per MB between different burst length for access under BUS.....	86
Figure 4.25 The bandwidth of memory access under external BUS among different bit-rate .....	88
Figure 4.26 The throughput of motion compensation for different data-reuse approach when operating frequency is 100Mhz. ....	88
Figure 5.1 CHIP photo for H.264/AVC main profile decoder.....	92



## ***List of Tables***

---

Table 2.1 comparison with different standard .....	27
Table 3.1 Analysis for different interpolating approach .....	44
Table 3.2 Comparison of execution cycles for different architectures .....	46
Table 3.3 Simulation results of required memory bandwidth (MByte/s) per MB by using extend-2D column-major approach. ....	53
Table 3.4 Comparison of interpolator architecture with other designs.....	55
Table 4.1 CAS latency .....	65
Table 4.2 Characteristics of READ/WRITE access .....	70
Table 4.3 Required memory size for different picture format supporting multiple reference pictures set 16. ....	75
Table 5.1 H.264/AVC main profile decoder specification for motion compensation .....	90
Table 5.2 Synthesis results of H.264/AVC's main profile decoder including SRAM.....	91
Table 5.3 On/Off-Chip memory size for different module in H.264 main profile decoder..	92

# ***Chapter 1***

## ***Introduction***

---

### ***1.1 Motivation***

The early video technology such as MPEG-1, mainly approach targets on CD-ROM based video storage. Afterward, MPEG-2 standard is published, which can be backward compatible with MPEG-1, serves a wider range of application including video-on-demand (VOD), DVD and high definition TV. Up to now, H.264/AVC [1] is the newest generation video coding standard developed by the Joint Video Team (JVT), which consists of experts from ITU-T VCEG and ISO/IEC MPEG. The H.264/AVC can save about 25-45% bit-rate compared to MPEG-4 Advanced Simple Profile (ASP). Recently, digital video processing technologies have been widely applied in the many video systems, such as videophone, digital TV and VCR, multimedia, etc. In the future, a high-quality HDTV system would integrate the functions of a computer, the internet, and entertainment, so it should become a popular product in the market. Furthermore, digital TV is widely adopted by the next-generation digital video broadcasting (DVB) technology. However, the amount of video data is very huge for these applications. For example, high-quality HDTV system with 1080HD format produces 1,504Mbits/s when the frame rate is 30Hz at level 4 in a real-time system. Therefore, H.264/AVC provides Main profile which supports many efficient coding tools to obtain enormous compression rate. The ultra high coding efficiency comes from many new features, including sub-pixel inter prediction with variable block size (VBS) and multiple reference



frames, intra prediction, bi-prediction, weighted prediction, and entropy coding—CAVLC and CABAC. According to the runtime analysis of H.264/AVC decoder software, the motion compensation can use up to 55% of total decoding time. Thus, motion compensation can dominate performance of entire H.264/AVC decoder. Furthermore, the bandwidth requirement of decoder is extremely high and a bandwidth-efficient design is necessary to achieve high-quality real-time decoding processes for high definition approach.

For motion compensation, we need to refer the previous frame data from memory for motion compensations. Generally, the coding performance becomes better using more temporal information by motion compensation. High definition TV requires enormous data transmission particular in frame memory, and the memory overhead becomes high over bus. For real-time operation, the memory data must be accessed during a limited processing time. The memory design and its addressing become a bottleneck for entire video decoder. Because the multiple reference pictures is supported by H.264/AVC's main profile, the block data controlling and addressing become more complex. How to access the frame memory for real-time operation is an important issue, particular for HDTV systems. Thus, a memory access controller that efficiently communicates with external memory is essentially provided over the entire video decoder to manage data transfer and access conflict.

## *1.2 Thesis Organization*

The thesis is organized as follows. The algorithm description and analysis of H.264/AVC's main profile is introduced and discussed in Chapter 2. In Chapter 3, the proposed bandwidth-efficient motion compensation architecture for H.264/AVC video decoder is described first. Then, the motion compensation engine for supporting H.264/AVC's main profile specification is illustrated. We also propose the novel data-reuse technique to reduce the required bandwidth particularly in H.264/AVC fractional motion compensation.

Chapter 4 presents frame and motion vector memory organization including memory access controller for external SDRAM. We apply a memory scheduling technique to reduce the access latency under external BUS and provide a flexible data arrangement method to improve data hit rate. The CHIP implementation is given in Chapter 5. Finally, conclusion is shown in Chapter 6.



# **Chapter 2**

## **Motion Compensation Algorithm of H.264/AVC's Main Profile**

---

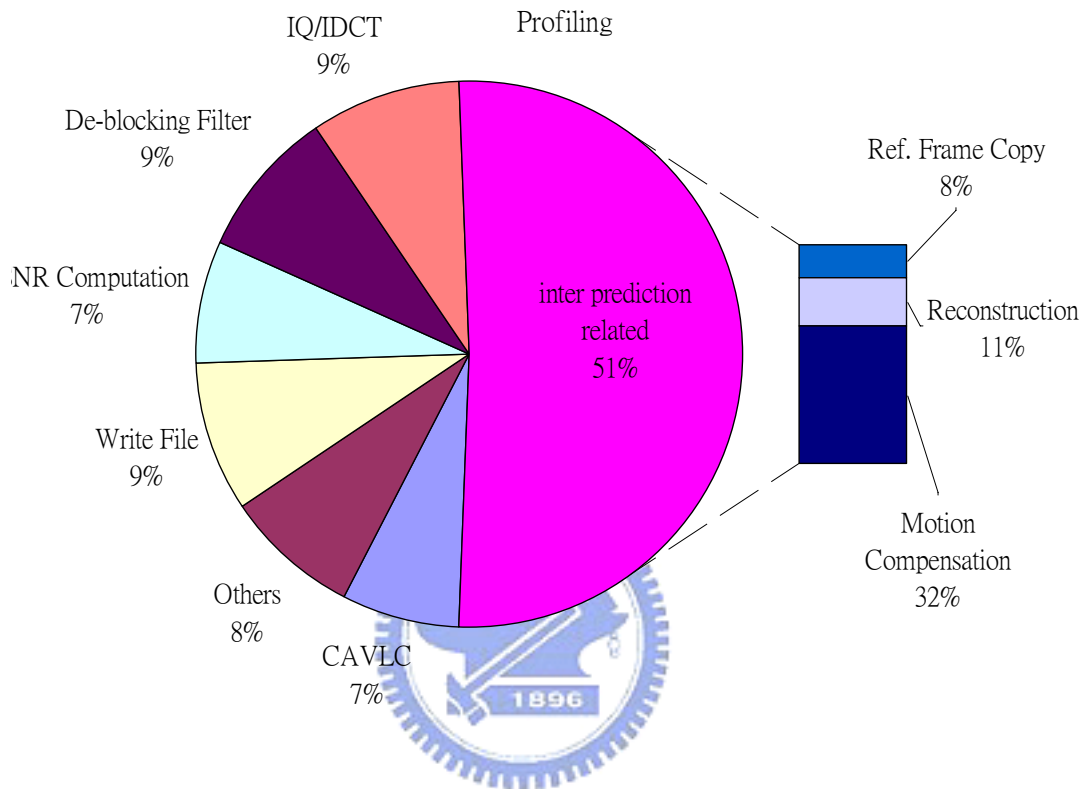
Similar to previous video standard, motion compensation is an important part in a video decoder system. The feature is that the current picture is predicted by previous decoded pictures without requiring extra bit-streams. Thus, the transmission bandwidth can be reduced efficiently without degrading visual quality. Hence, H.264/AVC is used in a wide range of applications due to its better coding efficiency.

In this Chapter, we will introduce a basic structure and concept of H.264/AVC coding standard in Section 2.2. In H.264/AVC, The main profile is almost a superset of the baseline profile. Specifically, additional tools provided by main profile are Bi-directional predictions, direct mode coding, multiple reference frames and weighted prediction for motion compensation part. The detailed algorithms of features related to motion compensation are described in the following sections. Finally, we will list differences among video coding standards such as MPEG-2, MPEG-4, etc in Section 2.6.

### **2.1 Profiling**

Figure.2.1 shows the profiling of H.264/AVC's main profile on ARM-7 processor. The reference software we adopt is JM 9.2 [3]. Specifically, inter prediction related modules, which occupy 51 % of the entire video decoder, include motion compensation, reconstruction,

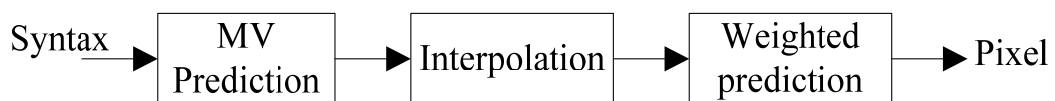
and reference frame copy. If we improve this part efficiently, total performance of the decoder system will be increased as well. This dominated part can be greatly reduced by parallel processing, data-reuse scheme, or pipeline processing on the ASIC design.



**Figure 2.1 H.264 software (JM 9.2) profiling on ARM 7 processor**

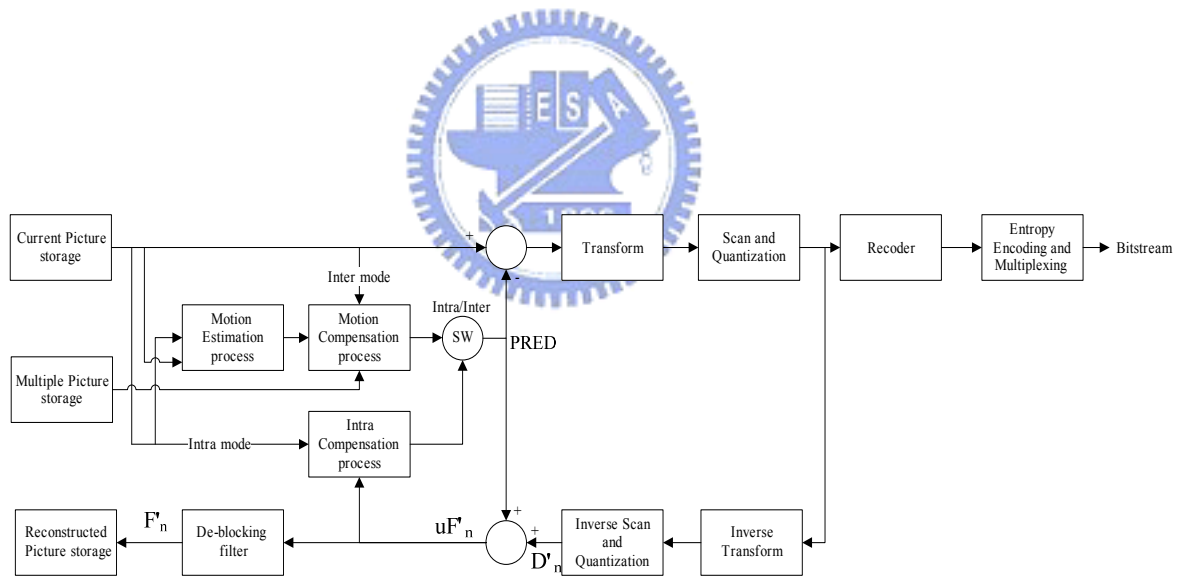
## 2.2 Motion Compensation Process Flow

The score of motion compensation process flow has been explained as Figure 2.2. Data relating to inter prediction are received from syntax parser. It is processed to pixels through several functional units consist of MV prediction, Interpolation and Weighted Prediction.

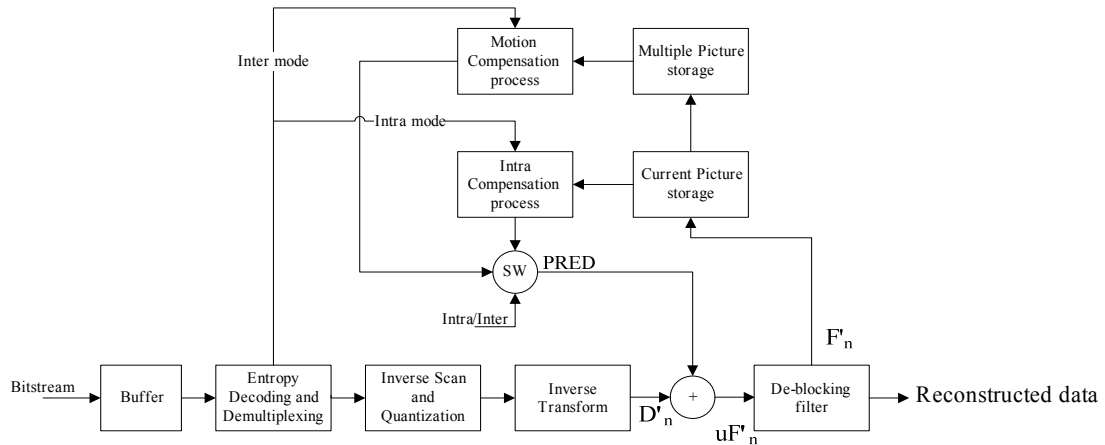


**Figure 2.2 The general score of motion compensation for H.264/AVC's main profile**

Figure 2.3 shows the basic block diagram of H.264/AVC encoding block diagram. The block diagram of decoder is shown in Figure 2.4. With the exception of the de-blocking filter, we can find that most of the basic functional components (prediction, transform, quantization, entropy coding, etc) exist in previous standards such as MPEG-1, MPEG-2, MPEG-4, H.263 but important changes of H.264 occur in the details of each functional block. Because the decoder is our research topic, we will focus on decoder process flow. The decoder receives a compressed bitstream from channel receiver side and thereby entropy decodes the data elements to produce a set of quantized coefficients  $X$ . These coefficients are scaled and inverse transformed to  $D'_n$ . The motion compensation (MC block) reconstructs the PRED according to previous decoded data. The PRED adds  $D'_n$  to produce  $uF'_n$  prior to the deblocking filter.



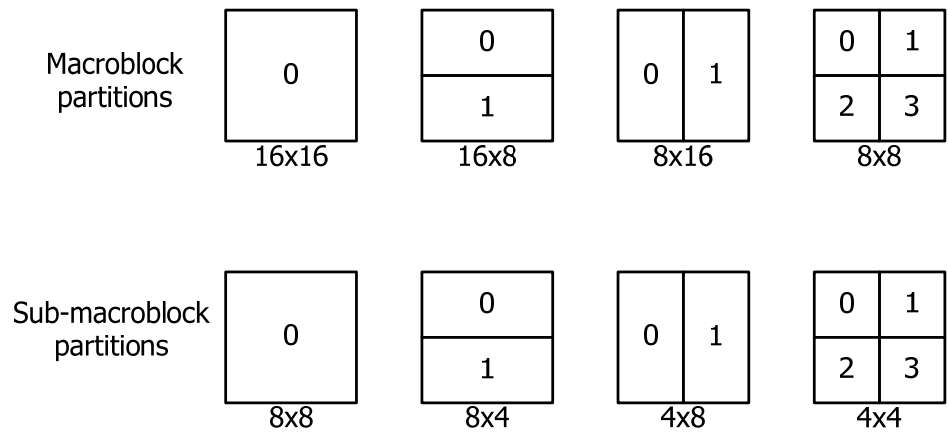
**Figure 2.3 General structure of H.264 encoder.**



**Figure 2.4 General structure of H.264 decoder**

### 2.3 *Inter Prediction Algorithm for H.264/AVC's Main Profile*

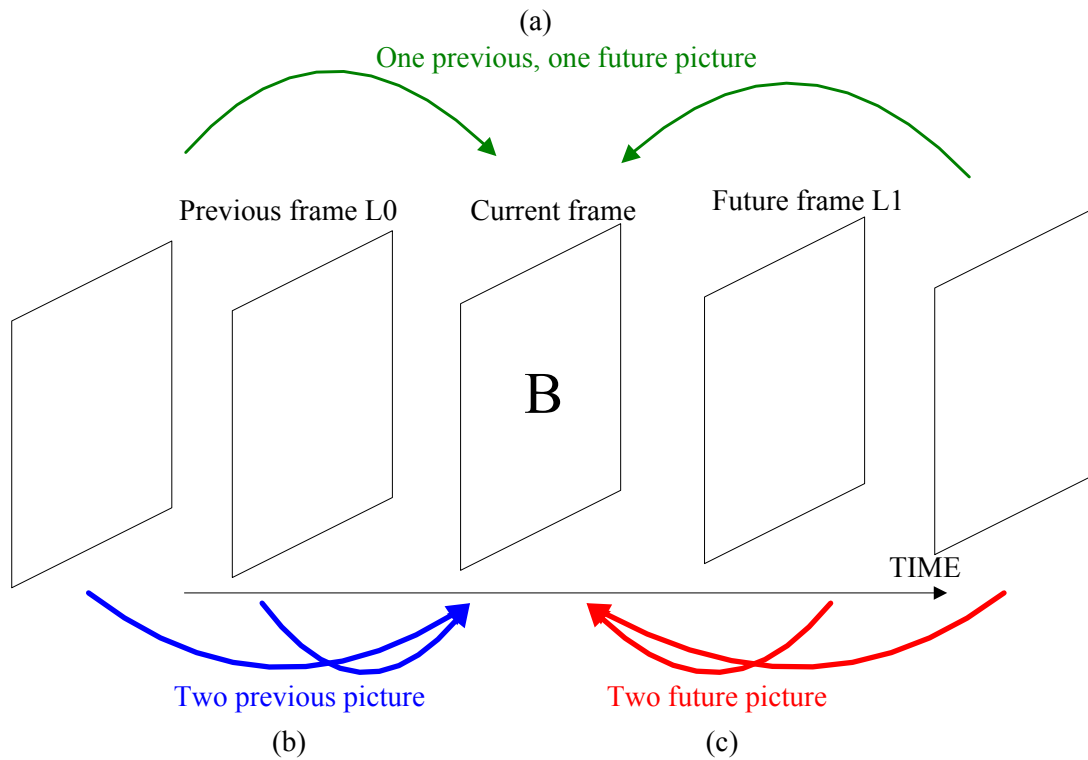
The inter prediction of H.264/AVC's main profile includes tree-structured hierarchical macroblock partitions and more flexible block size selection called as *variable block size (VBS)* compared with previous standards [1][2][4]. In case of motion compensated prediction, macroblocks are predicted from the image signal of transmitted reference images. For this purpose, each macroblock can be divided into smaller partitions such as 16x16, 16x8, and 8x8. The corresponding 8x8 sub-macroblock is further divided into partitions with block sizes of 8x4, 4x8 or 4x4. For each sub-macroblock partition, a motion vector may be independently selected and coded, but the reference picture index and prediction type of the sub-macroblock is used for all sub-macroblock partitions. Chroma components use the same partition as luma components. The smallest block size selection could reach as small as 4x4 and 2x2 for luma and chroma component respectively. For each macroblock partition, a reference picture index, prediction type (list-0, list-1, bi-pred), and a motion vector may be independently selected and coded. Figure 2.5 illustrates all types of partitions.



**Figure 2.5 Macroblock partitions and sub-macroblock partitions**

### 2.3.1 Bi-directional Prediction

A bi-directional prediction is main feature provided by H.264/AVC main profile. Bi-prediction uses two lists of previously decoded reference pictures, list-1 and list-0. The reference picture is previous or future decoded pictures for B-slices. Each macroblock of B slices may be predicted from previous reference picture (list-1) and future reference picture (list-0). In P slices, only single directional prediction is used, and the allowable reference pictures are list-0. In B slices, list-0 and list-1 of reference pictures are considered. For B-slices, single directional prediction using either list 0 or list 1 is allowed, or bi-prediction using both list 0 and list 1 is allowed. Figure 2.6 gives three examples to illustrate Bi-prediction: (a) one previous and one future reference (similar to B-picture prediction in previous MPEG video standard), (b) two past references and (c) two future references.



**Figure 2.6 Example using Bi-prediction: (a) previous/future (b) previous (c) future**

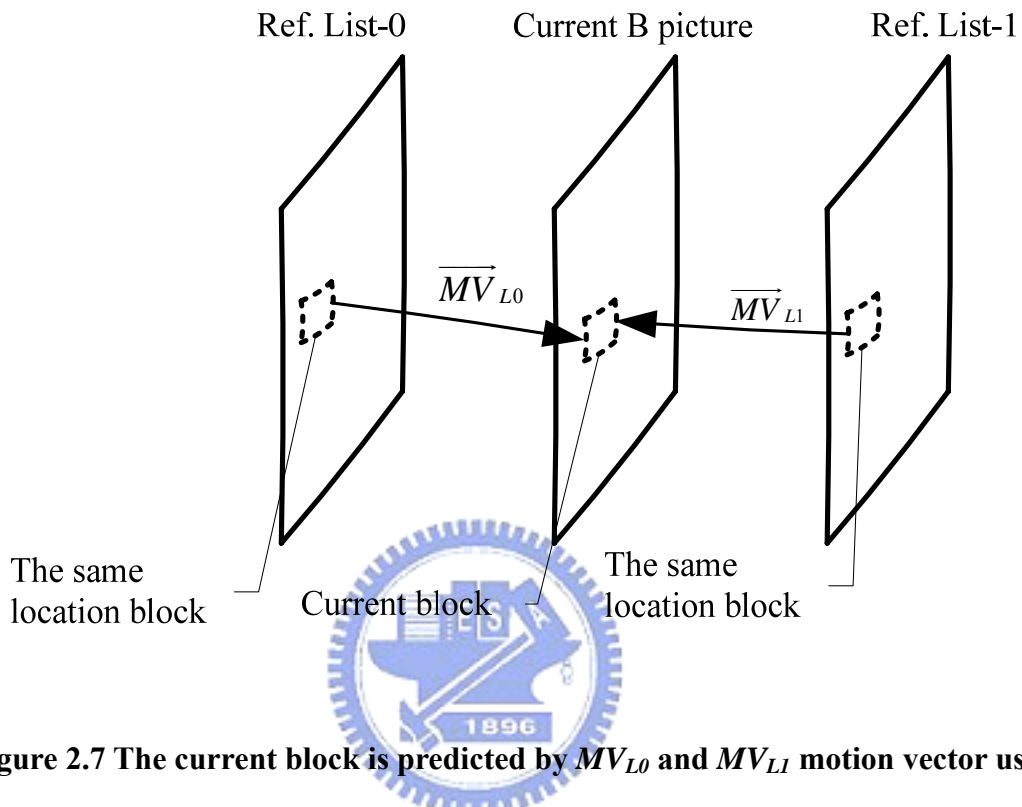
In the bi-prediction, a reference block is created from list-0 and list-1 reference pictures. Two motion compensated reference areas are obtained from a list-0 and list-1 picture respectively, and two separate motion vectors are required. Each sample of the prediction block is calculated as an average of motion vector of the list-0 and list-1 prediction sample. Except when using Weighted Prediction, the following equation is used:

$$Pred(i, j) = (Pred0(i, j) + Pred1(i, j) + 1) \gg 1 \quad (2.2)$$

Where  $Pred0(i, j)$  and  $Pred1(i, j)$  are prediction samples derived from the list-0 and list-1 reference pictures and  $Pred(i, j)$  is a bi-predictive sample. After calculating each prediction sample, the reconstructed samples are a summation of residual and predicted data that is decoded by entropy decoding and intra/inter prediction respectively. The list-0 and list-1 motion vectors in bi-predictive macroblocks or blocks are predicted from neighboring motion vectors that have the same temporal direction. For instance, a motion vector for the



current macroblock pointing to a previous picture is predicted from other neighboring motion vectors that also point to previous pictures. It is illustrated as Figure 2.7. The prediction of motion vector is introduced as next section.

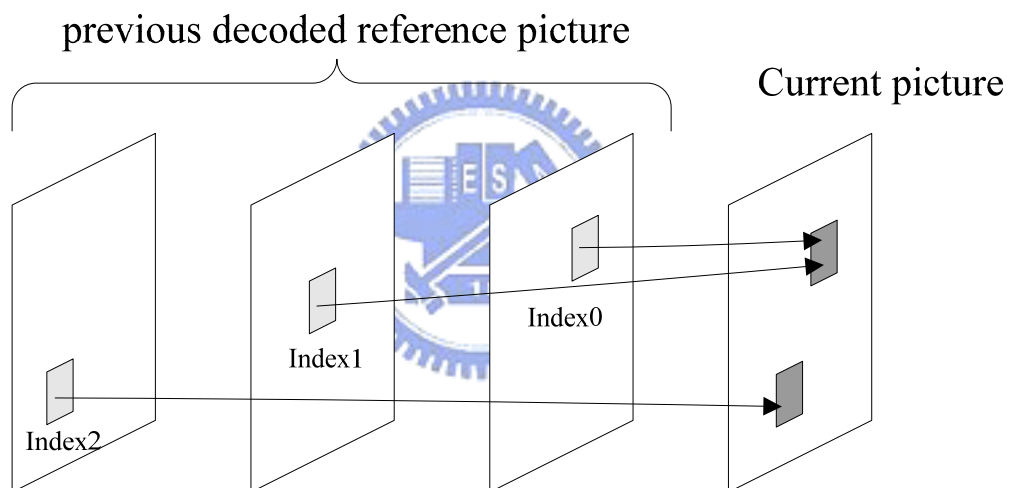


**Figure 2.7 The current block is predicted by  $MV_{L0}$  and  $MV_{L1}$  motion vector using Bi-prediction**

### 2.3.2 Multiple Reference Frames

In H.264/AVC, multiple reference frames may be used for inter-prediction [4], with a reference frame index coded to indicate which multiple reference frames are used. When bi-prediction is used by applying weighted prediction, the list 0 and the list 1 predictors are averaged together to form a final predictor. For each sub-macroblock partition, a motion vector may be independently selected and coded, but the reference frame index and prediction type of the sub-macroblock is used for all of the sub-macroblock partitions. Figure 2.8 shows the bi-prediction with multiple reference frames. An index is a reference frame parameter. An additional picture reference parameter has to be transmitted together with the motion vector in

bitstream. H.264 uses picture order count (POC) to indicate relative distances between coded pictures and reference pictures. POC is used for scaling motion vectors in direct modes, and for weighting factor derivation in WP implicit mode that will be introduced in the following sections. Adopting multiple reference frames increases the access frequency according to a linear model: 25% complexity increase for each added frame. A negligible gain (less than 2%) in bit rate is observed for low and medium bit rates, but more significant savings can be achieved for high bit rate sequences (up to 14%) [4].



**Figure 2.8 Bi-prediction with multiple reference pictures**

Up to five different reference frames can be used for inter-picture coding resulting in better subjective video quality and more efficient coding. Providing multiple reference frames can also help make the H.264 bitstream more error resilient. The error resilient tools are supported by extended profile in H.264/AVC, which will not be discussed in this thesis. Note that this feature leads to increased memory requirement for both the encoder and the decoder since previously decoded and reconstructed multiple reference frames must be maintained in

memory. For storing large pixels of several reconstructed reference frames, the huge memory size is required such as SDRAM. Therefore, we will propose an efficient memory allocation method and SDRAM controller architecture so that remained decoded pictures can be efficiently stored in single external memory. The related concept will be introduced in Chapter 4.

## 2.4 *Motion Vector Prediction*

The prediction for the decoded macroblock is determined by the set of motion vectors (MV) that are associated with that macroblock. The motion vectors indicate the position within the set of previously decoded frames from which each block of pixels will be predicted. A motion vector is generated by motion vector prediction. In baseline profile, motion vector is only generated by traditional MV prediction that includes median and directional prediction. Motion vector prediction of H.264/AVC's main profile supports new predictable method in Bi-predictive slices: direct mode, which except for traditional MV prediction. We introduce these motion vector generations in the following sub-sections.

### 2.4.1 **Traditional MV Prediction**

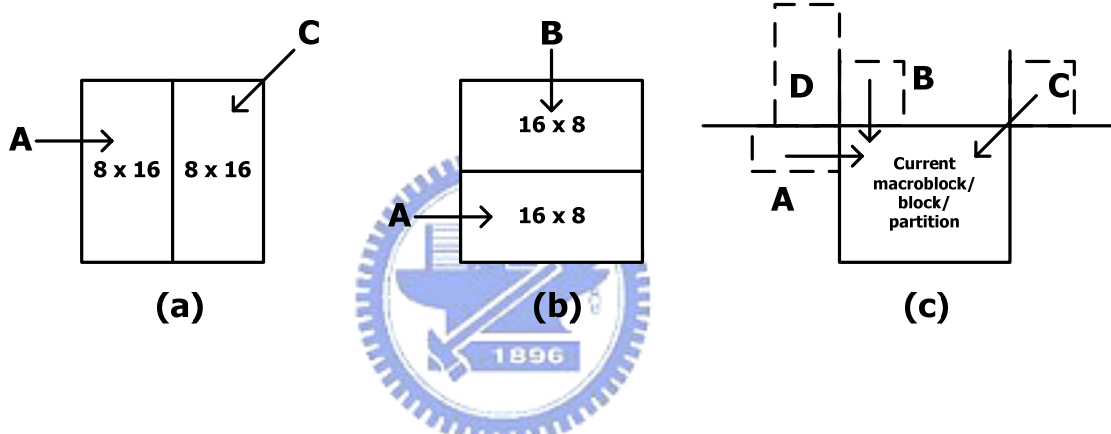
The Motion vector is generated from motion vector difference (MVD) and motion vector prediction (MVP). The associated equations are expressed by (2. 1).

$$\begin{aligned} MV_x &= MVD_x + MVP_x \\ MV_y &= MVD_y + MVP_y \end{aligned} \tag{2.1}$$

MVD is decoded from universal variable length decoder (UVLD) and MVP is predicted according to neighboring motion vectors. MVP algorithm, of which concept is similar to that for MPEG-4, contains directional prediction for 16 x 8 or 8 x 16 block size and median prediction for other block sizes. The detail of MVP decision is shown in Figure 2.9. Equation

of median prediction is expressed by (2. 2). The location of MVA, MVB, MVC, MVD which neighboring current block is depends on different block sizes. For example, MVA is a left neighboring block and MVC is a right-upper neighboring block when block size is 8x16 as Figure 2.9 (a) shows. The definition of neighboring motion vector is illustrated as Figure 2.9 for different block sizes. In addition, some boundary conditions or exceptions have to be handled carefully. For instance, when MVC is not available, its value is replaced by MVD. We do not go into details of those trivial boundary conditions over here.

$$MVP = median(MVA, MVB, MVC) \quad (2.2)$$

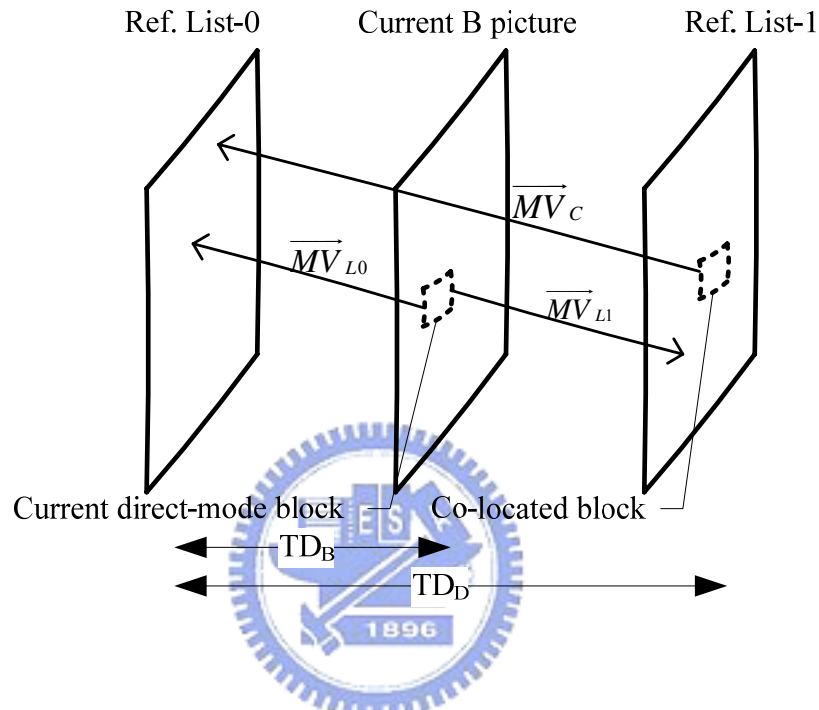


**Figure 2.9 (a) directional prediction for 8 x 16 block size, (b) directional prediction for 16 x 8 block size, (c) median prediction**

In addition to the motion-compensated block size described in Figure 2.5, a P macroblock can also be coded to P\_SKIP mode. For this coding mode, neither residual signal nor motion information is transmitted. That is, motion vectors are only decided according to MVP. The reconstructed data is obtained similar to that of macroblock type P\_16x16. Macroblocks coded in P\_SKIP are often located in large area with no scene change or slow motion. Besides the above techniques, H.264/AVC also supports multiple reference frames, weighted prediction and direct mode for B slice. These tools greatly improve coding efficiency. Application of de-blocking filter is a well-known method to improve image quality

by alleviating blocking artifacts. The de-blocking design in H.264/AVC is brought within motion-compensated prediction loop and the improvement in quality becomes more conspicuous.

## 2.4.2 Direct Mode Coding



**Figure 2.10 Direct mode prediction for B slices**

Direct mode is another method for motion vector prediction. The direct-mode macroblock does not require such side information but derives reference frame, block size, and motion vector data from the subsequent inter pictures. Figure 2.10 is shown to illustrate the process of direct mode coding. This mode superimposes two prediction signals. One prediction signal is derived from the future inter picture and the other comes from a previous picture. The direct mode uses bidirectional prediction and allows residual coding of the prediction error. The forward and backward motion vectors  $\overline{MV}_{L0}$  and  $\overline{MV}_{L1}$  of this mode are derived from the motion vectors  $\overline{MV}_C$  used in the co-located macroblock of the future picture Ref. list-1. Note that the direct-mode macroblock uses the same partition as the co-located

macroblock. The prediction signal is calculated by a linear combination of two blocks that are determined by the forward and backward motion vectors pointing to two reference pictures list-0 and list-1. When using multiple reference picture prediction, the forward reference picture for the direct mode Ref. list-1 is chosen to be the future inter picture with the co-located macroblock. The forward and backward motion vectors for direct-mode blocks are calculated as following equation:

$$X = \frac{\left(16384 + \text{abs}\left(\frac{TD_D}{2}\right)\right)}{TD_D} \quad (2.3)$$

$$ScaleFactor = \text{Clip}(-1024, 1023, (TD_B \times X + 32) \gg 6) \quad (2.4)$$

$$\overline{MV}_{L0} = (ScaleFactor \times \overline{MV} + 128) \gg 8 \quad (2.5)$$

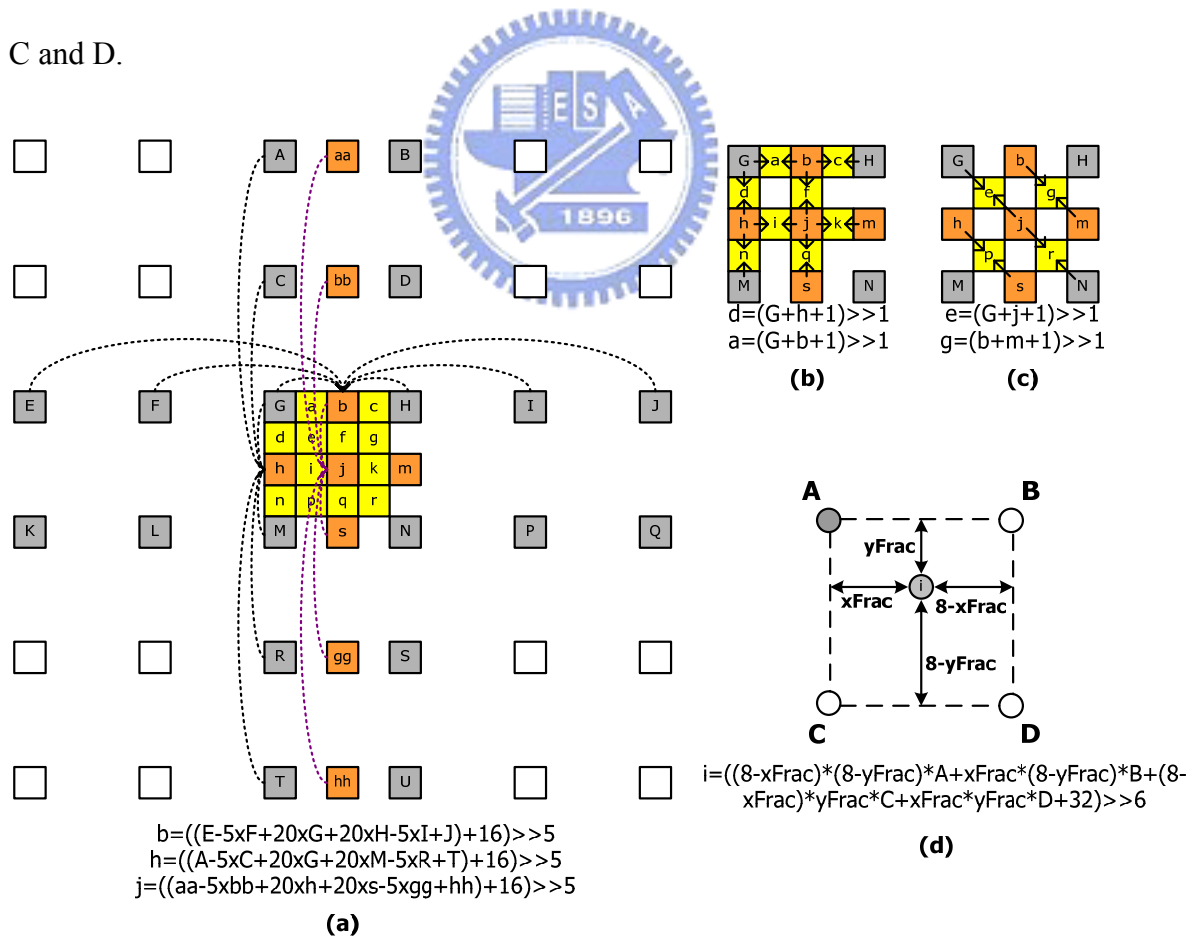
$$\overline{MV}_{L1} = \overline{MV}_{L0} - \overline{MV} \quad (2.6)$$

Where  $\overline{MV}_{L0}$  is the forward motion vector,  $\overline{MV}_{L1}$  is the backward motion vector, and  $\overline{MV}_c$  represents the motion vector of the co-located block in the future inter picture. For B pictures,  $TD_D$  is the temporal distance between the previous and the future inter picture, and  $TD_B$  is the distance between the current B picture and the previous inter picture. In that case, the actual reference picture Ref. list-0 (which is also a reference picture for the co-located macroblock of the following picture) is used for the calculation of the temporal distances  $TD_D$  and  $TD_B$ . And when both the current macroblock and its co-located are in frame mode,  $TD_B$  is the temporal distance between the current B frame and the reference frame Ref. list-0, and  $TD_D$  is the temporal distance between the future reference frame Ref. list-0 and Ref. list-1.

## 2.5 Fractional Interpolation

H.264/AVC main profile standard also supports high motion resolution that reaches quarter motion accuracy for luma component and eighth one for chroma component. This can


be found firstly in advances profile of MPEG-4 Visual standard; however, H.264/AVC reduces the complexity of interpolation processing comparison with MPEG-4 standard. Luma half sample interpolation is generated from integer-position samples using a 6-tap symmetrical Finite Impulse Response (FIR) filter with weights (1, -5, 20, 20, -5, 1). Once all the half-pel samples are available, the quarter samples are produced by linear interpolation using bilinear filter. Luma samples interpolation is shown in Figure 2.11(a)-(c). Quarter-pel resolution motion vectors in the luma component require eighth-sample resolution vectors in the chroma component assuming 4:2:0 chrominance format. Interpolated samples at eighth-sample intervals in chroma component are generated using bilinear interpolator illustrated in Figure 2.10 (d), and the displacement can achieve one-eighth accuracy. Each sub-sample position  $i$  is a linear combination of the neighboring integer sample positions A, B, C and D.



**Figure 2.11 (a) luma half sample with 6-tap FIR, (b) luma horizontal/vertical quarter sample with bilinear filter, (c) luma diagonal quarter sample with bilinear filter, (d) chroma sample with bilinear filter. Upper-case letters indicate the full samples and lower-case letter indicates the interpolated fractional samples**

From mathematical equations, they are both 2-D interpolation for luma and chroma interpolation. However, based on hardware implementation, these equations can be separated into two 1-D to reduce hardware cost, namely, horizontal filter first and then vertical one, or vice versa. In chapter 3, we will propose a novel architecture of interpolation to combine luma and chroma interpolation so that cost and complexity can be improved in ASIC design.

## 2.6 *Weighted Prediction*



The weighted prediction (WP) tool has been adopted in the H.264/AVC Main and Extended profiles to improve coding efficiency by applying a multiplicative weighting factor and an additive offset to the motion compensated prediction [5] [6]. While the concept of applying a weighting factor to a reference picture prediction is not new, the inclusion of the WP tool in the H.264 standard marks the first time such a feature has been incorporated into an international video compression standard. Weighted prediction also compensates the brightness difference so that the reference frame is more strongly correlated to the current frame. The WP allows arbitrary multiplicative weighting factors and additive offsets to be applied to reference picture predictions in both P and B pictures. The WP tool is particularly effective for coding fading sequences. When applying to a single prediction, as in P pictures, WP is similar to a leaky prediction, which has been previously proposed for error resiliency. Leaky prediction becomes a special case of WP, with the scaling factor limited to the range  $0 \leq a \leq 1$ . The WP also allows negative scaling factors, and scaling factors greater than



one. A key difference of H.264's WP tool from previous proposals involving weighted prediction for compression efficiency is the association of the reference picture index with the weighting factor parameters, which allows for efficient signaling of these parameters.

Use of weighted prediction is indicated in the sequence parameter set for P slices using the **weighted\_pred\_flag** field, and for B slices using the **weighted\_bipred\_idc** field. There are two WP modes -- explicit mode, which is supported in P and B slices, and implicit mode, which is supported in B slices only. A single weighting factor and offset are associated with each reference picture index for each color component in each slice. In explicit mode, these WP parameters may be coded in the slice header. In implicit mode, these parameters are derived based on relative distance of the current picture and its reference pictures. For each macroblock or macroblock partition, the weighting parameters are based on the reference picture index (or indices in the case of bi-prediction) of the current macroblock or macroblock partition. The reference picture indices are either coded in the bitstream or may be derived, e.g., for skipped or direct mode macroblocks. The use of the reference picture index to signal which weighting parameters to apply is bit-rate efficient, as compared to requiring a weighting parameter index in the bitstream, because the reference index is already available based on other required bitstream fields.

### 2.6.1 Explicit Mode

Use of explicit mode WP is indicated by **weighted\_pred\_flag** equal to 1 in the picture parameter set of P slices, or by **weighted\_bipred\_idc** equal to 1 in B slices. In explicit mode, the WP parameters are coded in the slice header for each coded slice. A multiplicative weighting factor and additive offset for each color component may be coded for each of the allowable reference picture in list 0 for P slices and B slices. The number of allowable reference pictures in list 0 is indicated in the picture parameter set by **num\_ref\_idx\_l0\_active\_minus1**, and for list 1 for B slices is indicated by

**num\_ref\_idx\_l1\_active\_minus1**. The weighting factors and offsets used in a particular slice are included in the slice header when explicit mode WP is used. The allowable range of parameter values is constrained to 16-bit arithmetic operations in the inter prediction process. The dynamic range and precision of the weighting factors can be adjusted using the **luma\_log\_weight\_denom** and **chroma\_log\_weight\_denom** fields, which are the binary logarithm of the denominator of the luma and chroma weighting factors, respectively. Higher values of the log weight denominator allow more fine-grained weighting factors but require additional bits for coding the weighting factors and limit the range of the effective scaling. For each allowable reference picture index in list 0, and for B slices also in list 1, flags are coded which indicate whether or not weighting parameters are present in the slice header for that reference picture index, separately for the luma and chroma components. If the weighting parameters are not present in the slice header for a given reference picture index and color component, a default weighting factor equivalent to a scaling factor of 1 and a zero offset are used. The multiplicative weighting factors are coded as **luma\_weight\_10**, **luma\_weight\_11**, **chroma\_weight\_10**, and **chroma\_weight\_11**. The additive offsets are coded as **luma\_offset\_10**, **luma\_offset\_11**, **chroma\_offset\_10**, and **chroma\_offset\_11**, respectively. For fades that are uniformly applied across the entire picture, a single weighting factor and offset are sufficient to efficiently code all macroblocks in a picture that are predicted from the same reference picture. When multiple reference pictures are used, the best weighting factor and offsets generally differ during a fade for the different reference pictures, as brightness levels are more different for more temporally distant pictures. Use of the reference picture index in the selection of the weighting parameters allows the coding efficiency gain of multiple reference picture prediction to be added to the coding efficiency gain of weighted prediction. For fades that are non-uniformly applied spatially across an image sequence, e.g. for lighting changes or camera flashes, more than one reference picture index can be associated with a particular reference picture are stored by using reference picture reordering commands. This

allows different macroblocks in the same picture to use different weighting factors even when predicted from the same reference picture store. In explicit mode, the same weighting parameters that are used for single prediction are used together for bi-prediction. The final inter prediction is formed for the pixels of each macroblock or macroblock partition, based on the prediction type used as follows.

Single directional prediction from list-0:

$$\text{SampleP} = \text{Clip1}(((\text{SampleP0} \times W_0 + 2^{LWD-1}) \gg LWD) + O_0) \quad (2.7)$$

Single directional prediction from list-1:

$$\text{SampleP} = \text{Clip1}(((\text{SampleP1} \times W_1 + 2^{LWD-1}) \gg LWD) + O_1) \quad (2.8)$$

Bi-prediction from list-0 and list-1:

$$\text{SampleP} = \text{Clip1}(((\text{SampleP0} \times W_0 + \text{SampleP1} \times W_1 + 2^{LWD}) \gg (LWD+1)) + (O_0 + O_1 + 1) \gg 1) \quad (2.9)$$

Where Clip1 operation is an operator that clips to the range [0, 255],  $W_0$  and  $O_0$  are the list 0 reference picture weighting factor and offset, and  $W_1$  and  $O_1$  are the list 1 reference picture weighting factor and offset, and  $LWD$  is the log weight denominator rounding factor.  $\text{SampleP0}$  and  $\text{SampleP1}$  are the list 0 and list 1 initial predictors, and  $\text{SampleP}$  is the weighted predictor.

## 2.6.2 Implicit Mode

Use of implicit mode is indicated by **weighted\_bipred\_idc** equal to 2 in B slices. In WP implicit mode, weighting factors are not explicitly transmitted in the slice header, but are derived based on relative distances between the current picture and the reference pictures, based on POC. Implicit mode is used only for bi-prediction coded macroblocks and macroblock partitions in B slices, including those using direct mode. The same formula for

bi-prediction as given in the preceding explicit mode section for bi-prediction is used, except that the offset values  $OO$  and  $OI$  are equal to zero, and the weighting factors  $W0$  and  $W1$  are derived using the formulas below:

$$X = (16384 + (TD_D \gg 1)) / TD_D \quad (2.10)$$

$$Z = \text{Clip3}(-1024, 1023, (TD_B \times X + 32) \gg 6) \quad (2.11)$$

$$W_1 = Z \gg 2 \quad (2.12)$$

$$W_0 = 64 - W_1 \quad (2.13)$$

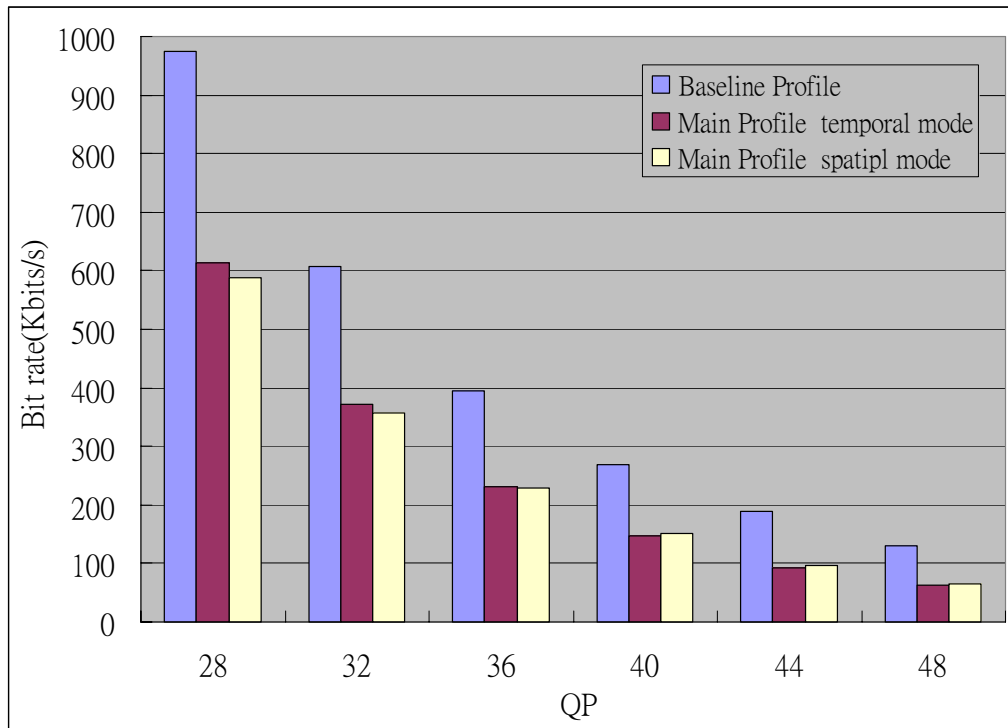
where  $TD_B$  is difference in the POC values between the list 1 reference picture and the list 0 reference picture, clipped to the range  $[-128, 127]$  and  $TD_D$  is difference in the POC values of the current picture and the list 0 reference picture, clipped to the range  $[-128, 127]$ . Macroblocks using single prediction (list 0 or list 1) do not use implicit mode WP. Implicit mode is most useful for low bit-rate applications, or for pictures that are broken into many slices for error resiliency, where the bits needed to code the WP parameters in explicit mode become significant contributors to overall bit-rate. For Bi-prediction macroblocks where the two predictors are from opposite temporal directions, as in traditional B pictures, the implicit mode WP formula becomes an interpolation formula. For example, with a traditional PBB picture pattern, weighting factors of  $(2/3, 1/3)$  are used in the first B picture and  $(1/3, 2/3)$  are used in the second B picture. For Bi-prediction macroblocks where the two predictors are both in the same temporal direction, the implicit mode WP formula becomes an extrapolation formula. For example if one predictor is from the immediately preceding picture and the other predictor is from two pictures preceding, weighting factors of  $(2, -1)$  are used. In our design, we implement division-free hardware with WP implicit mode according to equation (2.2)-(2.4). The detailed architecture is shown in the following sections.

For the PBB picture sequence, implicit and explicit mode performed similarly for the linear fade-out pattern, both with an average coding gain of 46.2%. For a non-linear S-curve

fade-out pattern with the PBB picture sequences, explicit mode outperformed implicit mode slightly, averaging a 41.3% gain vs. a 40.9% gain. The gains were lower for the fade-ins, with explicit mode outperforming implicit mode from 28.9% to 28.4% for the linear fade-in, and from 29.1% to 26.9% for S-Curve fade-in. Besides, a simple method for determining weighting factors has been described that achieves bit-rate reductions up to 67% for fade-to-black sequences.

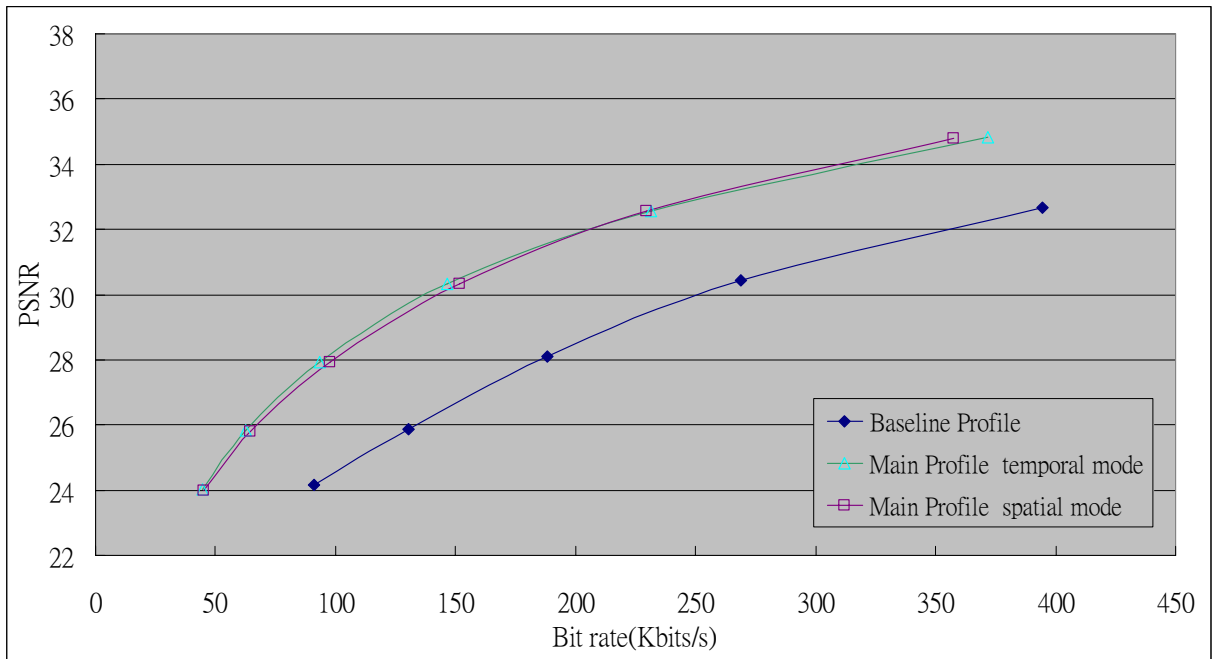
## 2.7 Analysis

H.264/AVC main profile supports new features to improve performance such as PSNR, bit rate, and quality, etc. Analysis is performed for these features to show the improvement. For analyzed environment, a test sequence with CIF format is employed at 30fps. The frame orders are I-P-P and I-B-B-P-B-B-P-B-B for baseline and main profile, respectively. Figure 2.12 shows the bit rate of baseline and main profile using different Quantization Parameter (QP). From Figure 2.12, reduction of bit rate can be observed, and bit rate of main profile may save approximately 40% compared with that of baseline profile. Thus, main profile is suitable for high bit-rate system such as HDTV, HD-DVD devices.



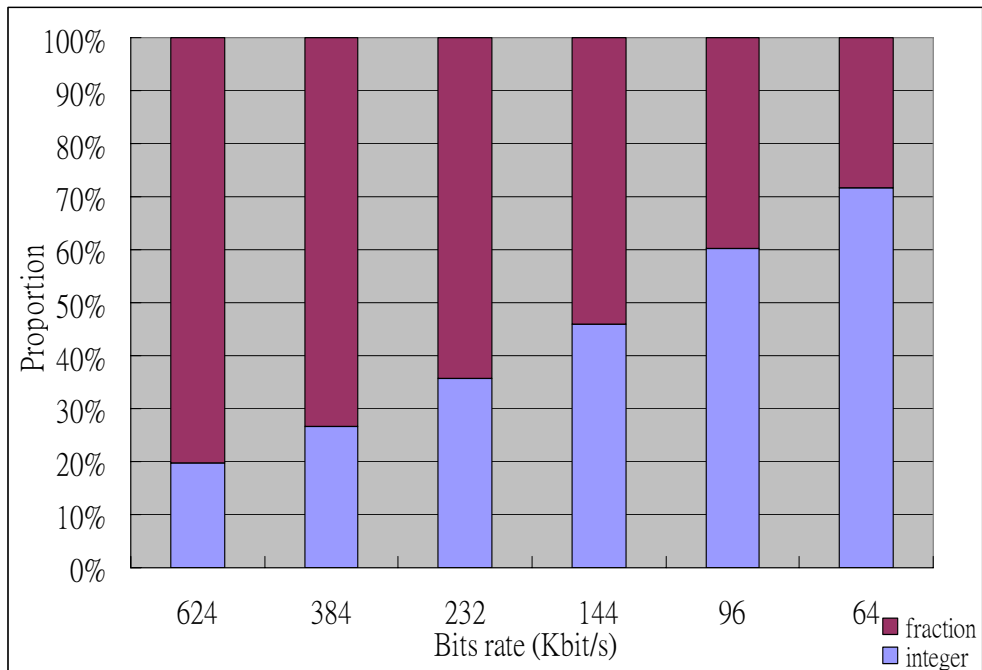
**Figure 2.12 Bit rate value between baseline and main profile**

Besides, the rate-distortion of main profile is depicted as Figure 2.13. The PSNR of main profile with WP and main profile without WP are shown in the same Figure. At 200Kbps, the PSNR of the main profile is 31.8, which is higher than PSNR of baseline is 28.4. Furthermore, performance of direct mode coding within main profile is illustrated as the same Figure. We can find that performance of spatial mode is a bit better than that of temporal mode in direct mode coding.

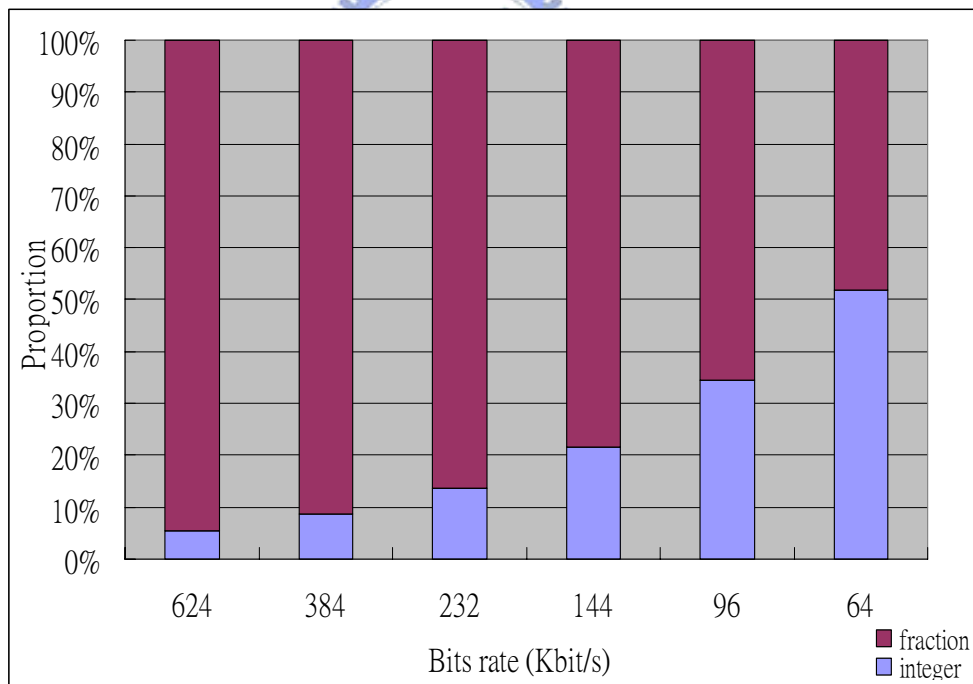


**Figure 2.13 PSNR between Baseline and Main profile**

Figure 2.14 and Figure 2.15 show proportion of integer/fraction motion vector for luma and chroma component, respectively. In order to simulate the proportion, we select a general sequence with CIF format. The picture order is I-B-B-P-B-B-P-B-B. In high bit rate applications (384 kbps), the fractional motion vector occupies about 80 % and even in low bit rate (48 kbps) fractional part has a certain proportion (30 %). The higher fractional motion vector proportion means that the longer execution time is required to read pixels from external frame memory. This gap may become more obvious especially when SDRAM is used as frame memory. To reduce requisite fetching pixels from frame memory, the efficient data-reuse technique for fractional motion compensation will be proposed in Chapter 3.



**Figure 2.14 The proportion of integer/fraction motion vector for luma component in H.264/AVC main profile**



**Figure 2.15 The proportion of integer/fraction motion vector for chroma component in H.264/AVC main profile**



## 2.8 Comparison for MC of Previous Standards

Considering the frame coding, Table 2.1 lists all fractional motion compensation features between different standards. Up to now, we can find fractional interpolation issue becomes more and more important in the state-of-the-art video coding. The interpolation window becomes larger for the same block size; namely, it requires much more cycles to interpolate each macroblock. For example, it requires  $9 \times 9$  pixels window to interpolate luma  $4 \times 4$ -block for H.264/AVC; however, the identical size of interpolation window can be used to filter  $8 \times 8$ -block for MPEG-2 video decoder. Therefore, it's requires 1,296 pixels to interpolate 16  $4 \times 4$ -blocks. Especially note that luma and chroma interpolation for H.264/AVC are different compared with previous standards. That is, no matter what on algorithm level or hardware level, the interpolated computation sources can not be shared. Hence, the combination of luma and chroma parts could be improved to reduce gate count and we will give the discussion and implementation in Chapter 3. In addition, H.264/AVC supports direct mode coding and weighted prediction which will be not adopted by previous video standard. Therefore, novel structures of direct mode coding and weighted prediction are proposed in the same chapter.

**Table 2.1 comparison with different standard**

Standard	MPEG-1/2	MPEG-4	H.264@Main
MVp prediction	Update from previous PMV value	Median prediction	Median prediction Directional prediction Direct mode prediction
Luma block unit	16 x 16	8 x 8	4 x 4
Luma motion accuracy	Half	Half, Quarter	Half, Quarter
Luma filter	Bilinear	<i><u>Half sample mode</u></i>	Half: 6-tap FIR Quarter: 6-tap FIR and bilinear
		Bilinear	
		<i><u>Quarter sample mode</u></i>	
		Half: 8-tap FIR Quarter: 8-tap FIR and bilinear	
Luma interpolation window	17 x 17	15 x 15	9 x 9
Chroma block unit	8 x 8	4 x 4	2 x 2
Chorma motion accuracy	Half	Half, Quarter	Eighth
Chorma filter	Bilinear	<i><u>Half sample mode</u></i>	Bilinear
		Bilinear	
		<i><u>Quarter sample mode</u></i>	
		Half: 8-tap FIR Quarter: 8-tap FIR and bilinear	
Chorma interpolation window	9 x 9	5 x 5	3 x 3

## 2.9 Summary

From the H.264/AVC profiling on ARM processor, we can find that an efficient hardware accelerator or ASIC design for motion compensation is crucial. For HDTV application, H.264/AVC main profile has provided several coding tools to deal with high-quality resolution. Bi-prediction and quarter-pel interpolation are proposed to improve coding efficiency. Weighted prediction is first adopted by video standard, and is a powerful tool for efficiently coding fading sequences. Bitstream size is reduced by direct mode coding which is adopted by H.264/AVC main profile for B-slices. In B-slices, inter prediction is performed by

using two frames so that motion compensation hardware are more complex. Furthermore, multiple reference frames is proposed so that memory requirement may be extremely increased. For above discussion, not only hardware accelerator but also bandwidth-efficient hardware is required to develop for high-definition system. Finally, the inter prediction for H.264/AVC and the comparison among different standards are also illustrated in this Chapter.



# **Chapter 3**

## **A Bandwidth-efficient Motion Compensation Architecture Design**

---

In video standards, such as MPEG-1/2, MPEG-4 and H.264/AVC, motion compensation is an important part of entire decoder system, and always dominates system performance due to high computing power. Furthermore, the hardware design of motion compensation is more complex than other modules such as CAVLC, DCT, intra-prediction and De-blocking filter, etc. For HDTV application, motion compensation adopts new features which are supported by H.264/AVC main profile so that procedure and hardware of motion compensation are more and more complex in ASIC designs. Besides, inter-prediction requires large pixels of previous decoded reference frame to predict current frame, and external memory is decided as frame memory in our on-chip design. Moreover, multiple reference frames are employed to lead that more memory may be used to store pixels of several previous decoded reference frames. Thus, memory bandwidth which is data traffic under external BUS will be a bottleneck of motion compensation. A bandwidth-efficient motion compensation hardware accelerator has to be designed, which can be integrated into simplex architecture.

In this chapter, we will focus on motion compensation for high throughput and low cost designs. We propose a bandwidth-efficient motion compensation architecture which is suitable for high-quality system. Firstly, we will introduce whole bandwidth-efficient motion compensation architecture for H.264/AVC main profile. The hardware of detail module such as motion vector generation, interpolator, and weighted prediction will be discussed in

sub-section 3.2-3.4, respectively. Finally, simulation and summary are given in section 3.5.

### 3.1 Motion Compensation Engine for H.264/AVC's Main Profile

The H.264/AVC main profile decoder system is illustrated as Figure 3.1. First, the frame information in bitstream is decoded by entropy coding module includes CABAD and CAVLD. According macroblock type, the frame pixels can be decoded by intra-prediction and inter-prediction. The bus traffic is treated by memory controller which can be supported for module of video decoder. Figure 3.2 illustrates the entire bandwidth-efficient motion compensation architecture for H.264/AVC main profile. In H.264/AVC, a 4x4 block is the smallest element of the prediction block types in variable block size (VBS) and each 16x16 block can be decomposed into several 4x4 blocks. We adopt a 4x4 block-based pipeline to implement this motion compensation design in this design, because the 4x4 block is the smallest processing unit of pixels that the H.264/AVC adopts and 4x4 block-based pipeline can save the cost of storage buffer and the associated power reduction.

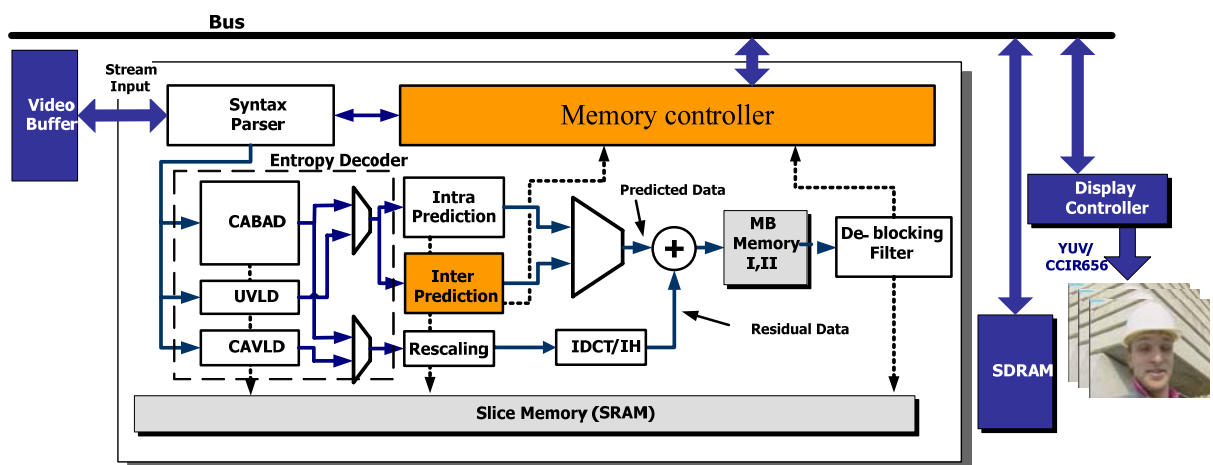
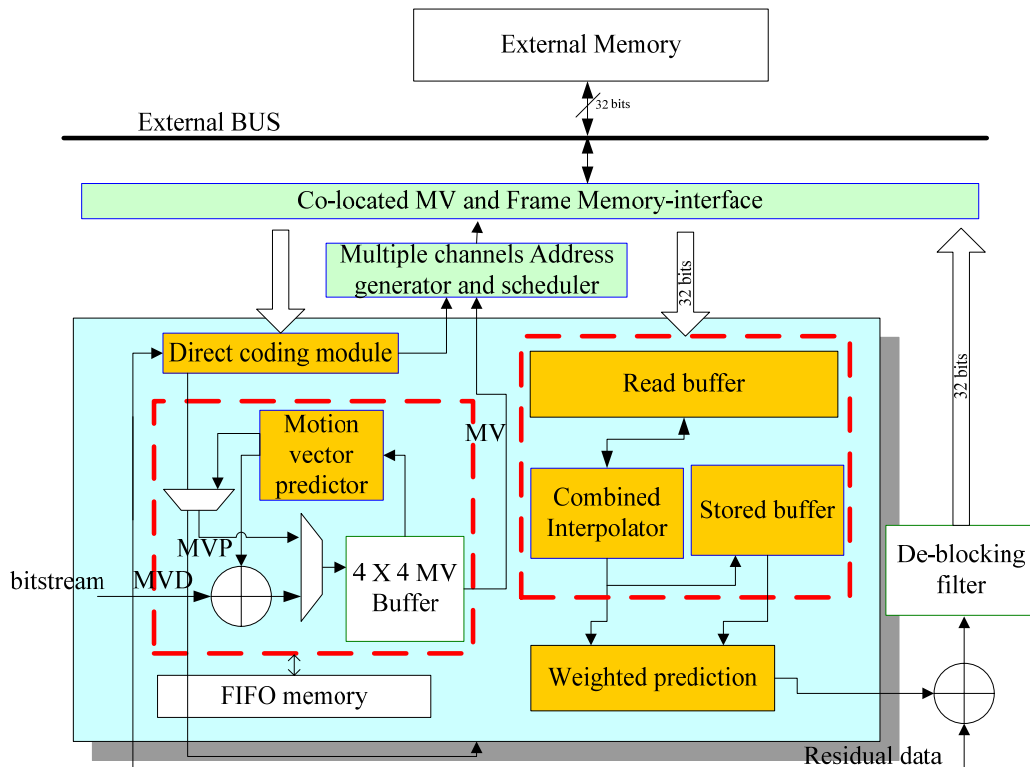


Figure 3.1 The block diagram of H.264/AVC main profile decoder system



**Figure 3.2 Motion compensation architecture for HDTV H.264/AVC main profile**



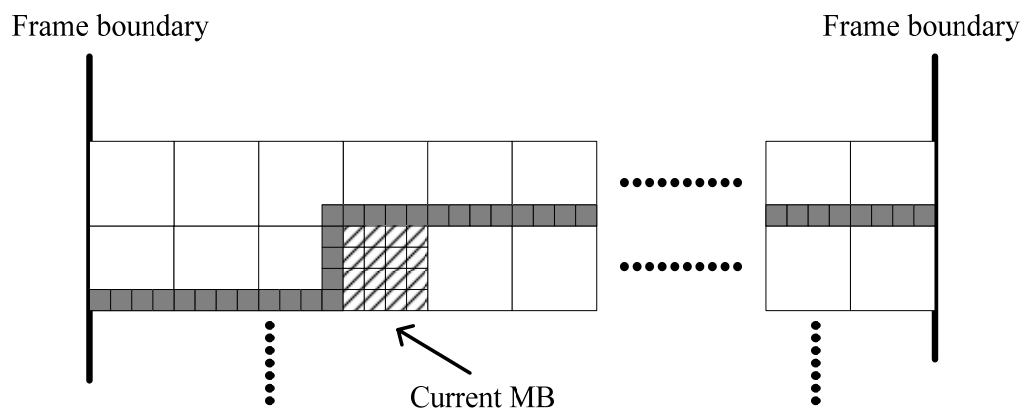
**decoder**

Excluding the memory controller, the proposed motion compensation architecture is presented in gray dotted area of Figure 3.2. The detailed discussion of frame memory access controller is shown in Chapter 4. The motion compensation architecture consists of three major parts that are motion vector generator (MVG), interpolator and weighted prediction. The decoded information is firstly loaded from bitstream into MVG. A MVG generates motion vector to predict current macroblock. In H264/AVC's main profile, motion vector is generated by two predicted methods: motion vector prediction (MVP) and direct mode coding. The details of MVG are described in the sub-section 3.2. According to motion vectors which are produced by MVG, corresponding reference pixels are loaded from external frame memory. In this chapter, we will not discuss memory such as memory controller and address generator, etc. Interpolators are invoked to produce fractional samples for both luma and chroma components. In this design, we employ two interpolators to simultaneously process

pixels of list-0 and list-1 because two motion vectors will point to two search areas in list-0 frame and list-1 frame in B-slices, respectively. When the motion vector is an integer value, corresponding reference pixels without interpolation directly feeds through weighted prediction. In the end of motion compensation processes, weighted prediction (WP) is performed by applying a multiplicative weighting factor and an additive offset in bitstream. These pixels obtained by weighted prediction add with residual data to create the unfiltered pixels. Finally, the de-blocking filter loads these pixels, and restores correct pixels into external memory after performing filter operations. Because the data bus of external frame memory is defined as 32bit, pixels which are loaded into interpolator are limited.

### 3.2 Motion Vector Predictor Design

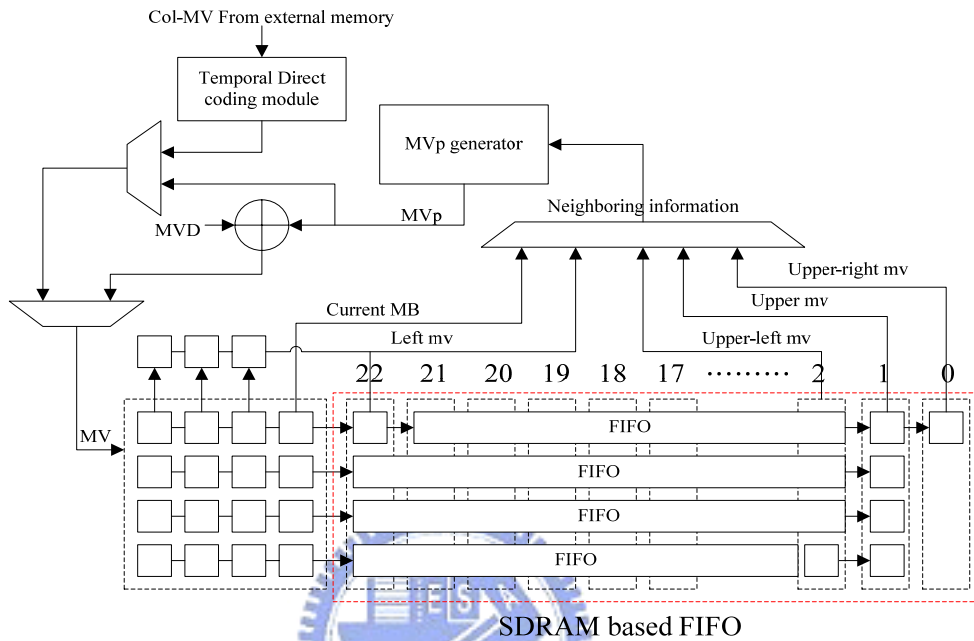
To facilitate a spatial prediction, we store motion vector for one row stripe of 4x4 blocks, four left neighboring 4x4 blocks and current 4x4 blocks into Row stored buffer. Figure 3.3 illustrates that shaded regions have to be stored for predicting oblique region.



**Figure 3.3 MV in shaded and oblique line region must be stored in row-FIFO.**

Firstly, motion vector generator is shown in Figure 3.4. Motion vector is obtained in two

predicted methods: MVP and direct mode coding. Note that direct mode coding is supported in B-slice. According to MB types, the motion vector is obtained by different predicted methods and stored into current motion vector buffer.



**Figure 3.4 Motion vector generator**

### 3.2.1 MVP Prediction Module

In the MVP generation method, the motion vector is generated by summing predicted motion vector (MVP) and MVD. For calculating MVP, we employ directional segmentation prediction in 8x16 or 16x8 block types and median prediction in other block types. These predictions are integrated into MVP generator. The MVP generator calculates MVP according to the motion vectors of the neighboring blocks in current frame. Thus the decoded motion vectors are required to be stored into FIFO buffer for the subsequent decoding. FIFO buffer stores the decoded motion vector pair (MVX, MVY). The depth and width of MV FIFO are dependent on the decoded frame width and search range respectively. For instance, for supporting 1080HD format, the total size of FIFO buffer is 968 x 10 bits (((120 blocks x 4 + 4)

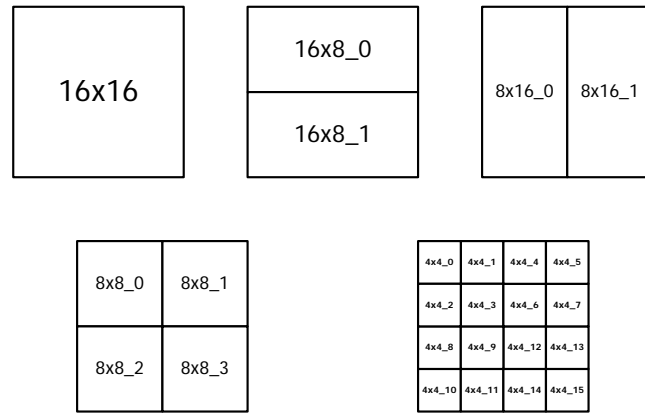


x 2) = 968 4x4-block). Therefore, SRAM is selected as a FIFO buffer to store required decoded motion vectors in our design. Once the content of FIFO buffer will not be used in the future, the restored motion vector pair in FIFO buffer can be discarded. Furthermore, the 4 x 4 size of MV buffers is required because the maximum number of motion vectors per MB is 16. The motion vectors for current MB decoding store in this 4 x 4 MV buffers. Due to a Bi-prediction, two 4 x 4 MV buffers are required to store current two motion vectors for predicting motion vectors of list-0 frame and list-1 frame.

<b>MVLU</b>	<b>MVU0</b>	<b>MVU1</b>	<b>MVU2</b>	<b>MVU3</b>	<b>MVRU</b>
<b>MVL0</b>	<b>MV0</b>	<b>MV1</b>	<b>MV4</b>	<b>MV5</b>	
<b>MVL1</b>	<b>MV2</b>	<b>MV3</b>	<b>MV6</b>	<b>MV7</b>	
<b>MVL2</b>	<b>MV8</b>	<b>MV9</b>	<b>MV12</b>	<b>MV13</b>	
<b>MVL3</b>	<b>MV10</b>	<b>MV11</b>	<b>MV14</b>	<b>MV15</b>	

**Figure 3.5 Neighboring motion vectors required for decoding all motion vectors in current macroblock**

When decoding current macroblock, the detail of required neighboring motion vectors is shown in Figure 3.5. To involve all kinds of VBS conditions, storages element is based on 4 x 4-block size that is the smallest element for H.264/AVC video decoder. Each square indicates one motion vector pair. To predict MV0-MV15 in current MB, it requires neighboring motion vectors in left-upper corner (MVLU), right-upper corner (MVRU), upper (MVU0-3) and left (MVL0-MVL3) positions., Neighboring motion vectors are shifted and stored into MV FIFO except for current MV.



(a)

size	direction	MV
16x8_0	MVB	MVU0
16x8_1	MVA	MVL2
8x16_0	MVA	MVL0
8x16_1	MVC	MVRU

(b)

size	MVA	MVB	MVC	MVD
16x16	MVL0	MVU0	MVRU	MVLU
8x8_0	MVL0	MVU0	MVU2	MVLU
8x8_1	MV1	MVU2	MVRU	MVU1
8x8_2	MVL2	MV2	MV6	MVL1
8x8_3	MV9	MV6	MV3	MV3

(c)

size	MVA	MVB	MVC	MVD
4x4_0	MVL0	MVU0	MVU1	MVLU
4x4_1	MV0	MVU1	MVU2	MVU0
4x4_2	MVL1	MV0	MV1	MVL0
4x4_3	MV2	MV1	MV0	MV0
4x4_4	MV1	MVU2	MVU3	MVU1
4x4_5	MV4	MVU3	MVRU	MVU2
4x4_6	MV3	MV4	MV5	MV1
4x4_7	MV6	MV5	MV4	MV4
4x4_8	MVL2	MV2	MV3	MVL1
4x4_9	MV8	MV3	MV6	MV2
4x4_10	MVL3	MV8	MV9	MVL2
4x4_11	MV10	MV9	MV8	MV8
4x4_12	MV9	MV6	MV7	MV3
4x4_13	MV12	MV7	MV6	MV6
4x4_14	MV11	MV12	MV13	MV9
4x4_15	MV14	MV13	MV12	MV12

(d)

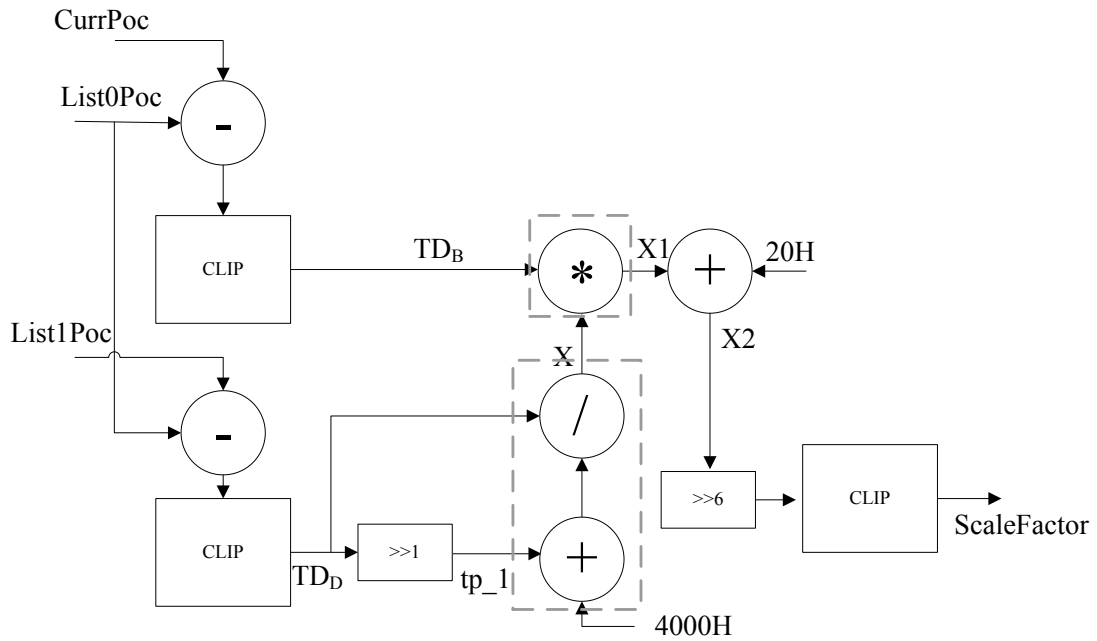
**Figure 3.6 (a) block size position index, (b) directional prediction table (16x8, 8x16), (c) median prediction table (16x16, 8x8), (d) median prediction table (4x4)**

MVp is calculated according to MVA, MVB, MVC and MVD which are obtained from neighboring motion vectors according to block size position index for different macroblock types. The block size position index in one macroblock is illustrated in Figure 3.6 (a). MVA, MVB, MVC and MVD indicate the motion vectors located at left, upper, right-upper, left-upper neighboring macroblock/partition/block respectively as shown in Figure 2.8 (c). Figure 3.6 (b)-(d) lists all MVA, MVB, MVC and MVD for different block size position index. When MB\_type of current macroblock is 16x8 or 8x16, MVp can be derived by directional

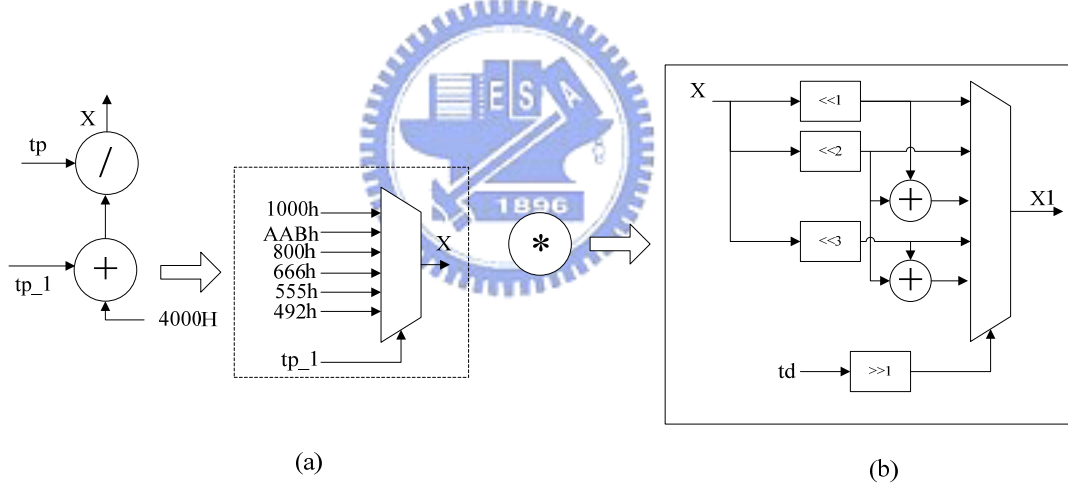
prediction, otherwise median prediction is involved. Furthermore, the above loop-up table (LUT) is required for motion vector prediction, many trivial boundary conditions and exceptions have to be handled. Here, we do not describe them for simplicity.

### 3.2.2 Direct Mode Coding Design

Except for MVp prediction, other way to predict current motion vectors is direct mode coding. In the direct mode coding, there are two types: spatial and temporal types [7] [8]. These types are user-defined in encoding processes. From above discussion, the PSNR of spatial mode is better than that of temporal mode. In our design, we implement both temporal and spatial modes and integrate it into MVG module. When a temporal mode is invoked, a temporal direct mode coding module calculates motion vector according to the picture order counts and co-located motion vectors in first list-1 frame. From above introduction of direct mode coding with temporal mode, we have to calculate the scalefactor value by equation 2.4. From Equation 2.5 and 2.6, two motion vectors from list-0 and list-1 frame are computed with scalefactor. Therefore, the scalefactor must be computed in advance. Figure 3.7 depicts hardware by which scalefactor is computed. We implement division-free and multiplication-free design to reduce hardware complexity. We employ some multiplexer and shifters to replace division and multiplication in gray dotted area and it is shown in Figure 3.8



**Figure 3.7 Pre-scalefactor generator design**



**Figure 3.8 (a) Division free replacement (b) Multiplication-free replacement**

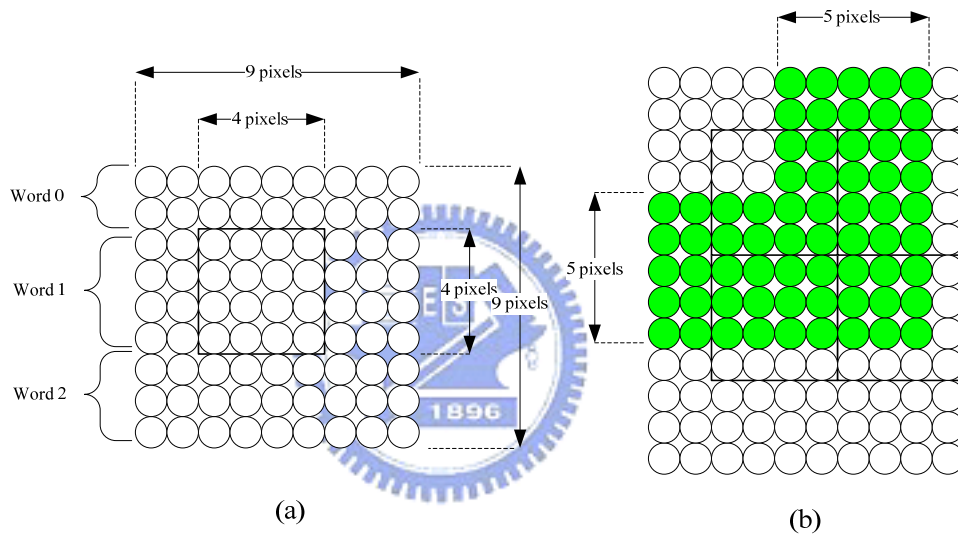
Where CLIP operation is used to restrict TDB , TDD and scalefactor within range between -128 and 127. The CLIP operation is expanded as Equation 3.1. The complexity of this module is reduced efficiently by division-free and multiplication-free.

$$CLIP = (-128, 127, input\_value) \tag{3.1}$$

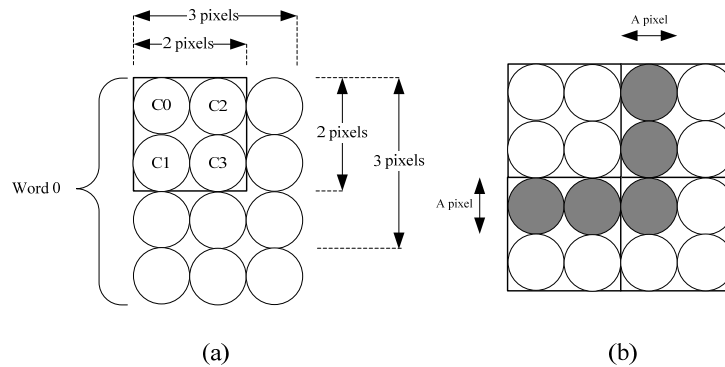
What is more, the process which produces motion vectors by spatial mode is the same as

median method for MVp prediction. Therefore, the hardware of MVp prediction module and spatial direct mode coding predictor can be shared. When the spatial mode is chosen, the predicted process acts as the MVp prediction, which needs motion vectors of the neighboring blocks to generate motion vector. Hence, we can employ MVp generator to generate motion vector without adding MVD.

### 3.3 Bandwidth-Efficient Fractional Interpolator Design



**Figure 3.9 (a) 4x4-block and 9x9 interpolation search windows for luma component interpolation (b) overlap region between neighboring blocks**



**Figure 3.10 (a) 2x2-block and 3x3 interpolation search windows for chroma component interpolation (b) overlap region between neighboring blocks**

Interpolator design always dominates the throughput of H.264/AVC decoder. To interpolate each fractional sample value for each 4x4 block of luma component, it needs 9 x 9 interpolation window illustrated in Figure 3.9 (a). If two motion vectors of neighboring 4 x 4 blocks are the same, 5 x 9 overlapped region between two interpolation windows can be data reused. The overlapped region between neighboring blocks is shown in Figure 3.9 (b). We can find that maximum overlapped region is 65 pixels for luma search windows. For each 2 x 2 block of chroma component is shown in Figure 3.10, the size of interpolating search windows is 3 x 3 and 5 pixels can be reused between neighboring blocks. For above property, when interpolating current block, overlapped region cannot be fetched again. We will introduce the proposed data-reuse approach and give some examples in sub-section 3.3.1.

### 3.3.1 Data Reuse Technique

The scanning order of residual decoding for each macroblock is row-major interpolating order as shown in Figure 3.11 (a), and column-major interpolating order illustrated in Figure 3.11 (b). A dotted line indicates transition between interpolating processes. In comparison of row-major interpolating order and column-major interpolating order, we adopt a column-major interpolating order because the transition of column-major interpolating order is 5 times less than row-major order. Each transition causes that the overlap region could not be reused. Therefore, column-major one is the better selection because of less number of transitions.

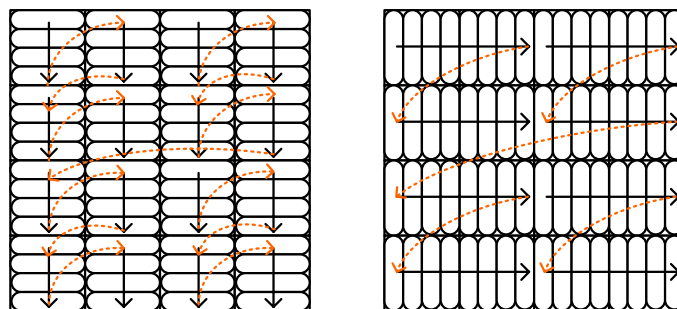
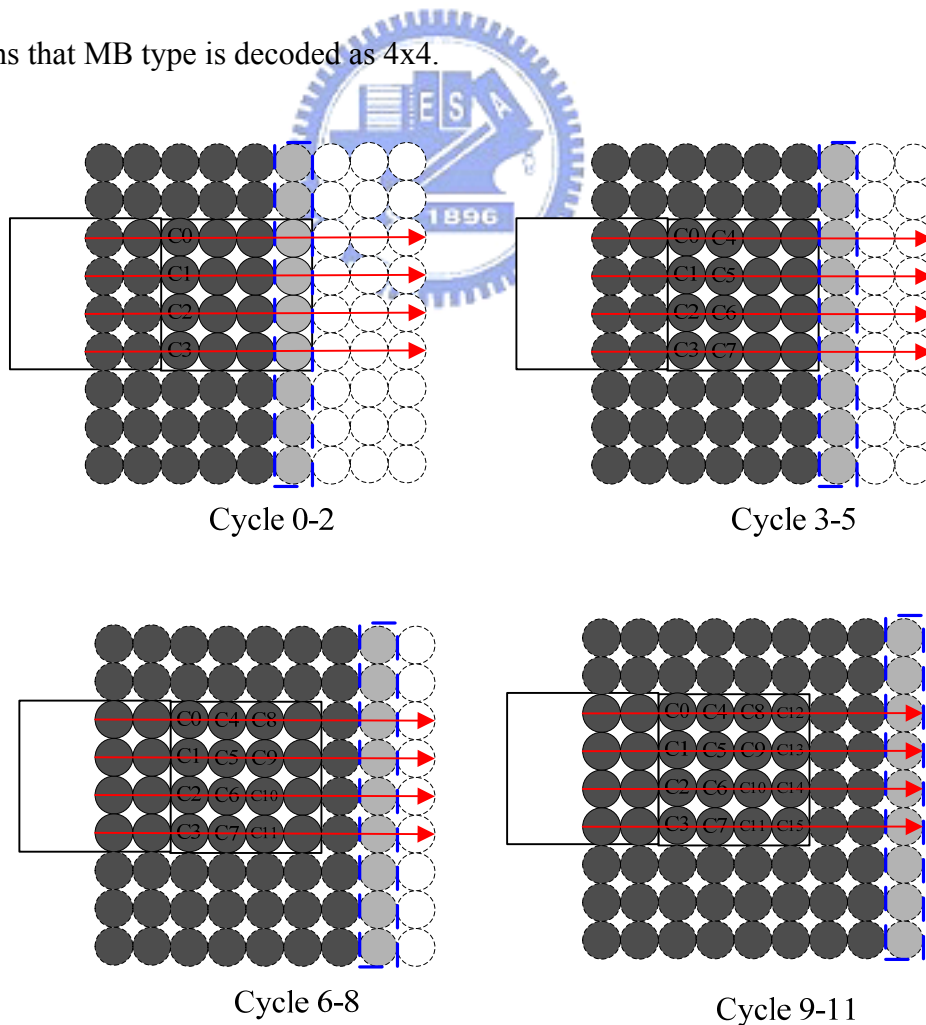


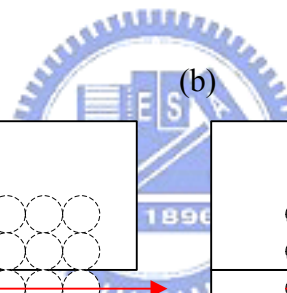
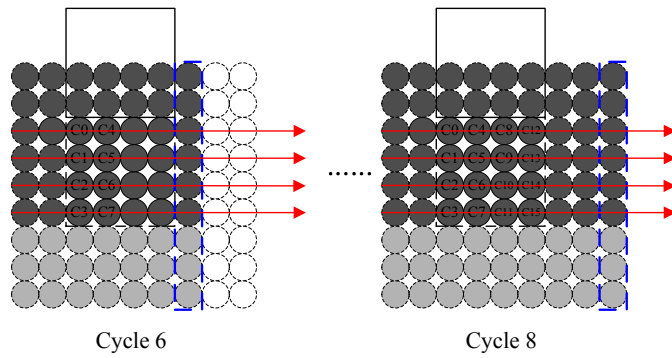
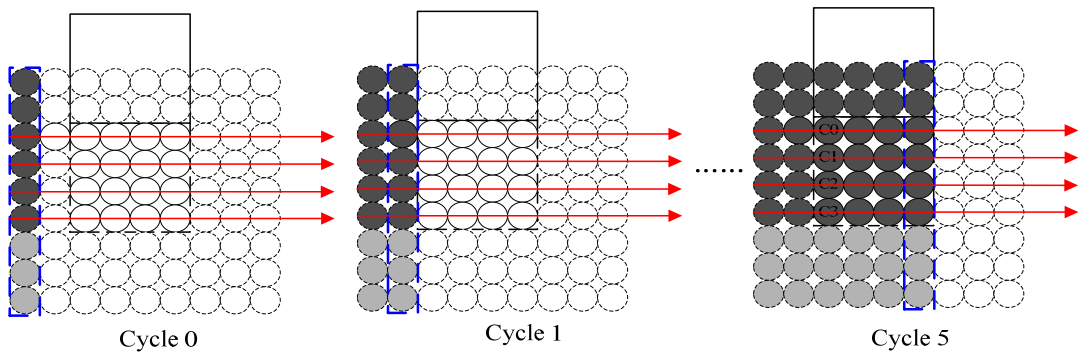
Figure 3.11 (a) row-major interpolating order (b) column-major interpolating order

For a data-reuse approach, Wang's design [10] proposed an extended 2 x 2 raster scanning order approach to increase throughput. Although 30% reduction of access cycle for motion compensation is derived by this approach, the improvement is not high enough for high-definition resolution. Therefore, based on the column-major interpolating order, we propose an *extend-2D column-major approach* (E2CMA) to reduce read access times from external memory and thereby achieve approximately 60% reduction of access cycles.

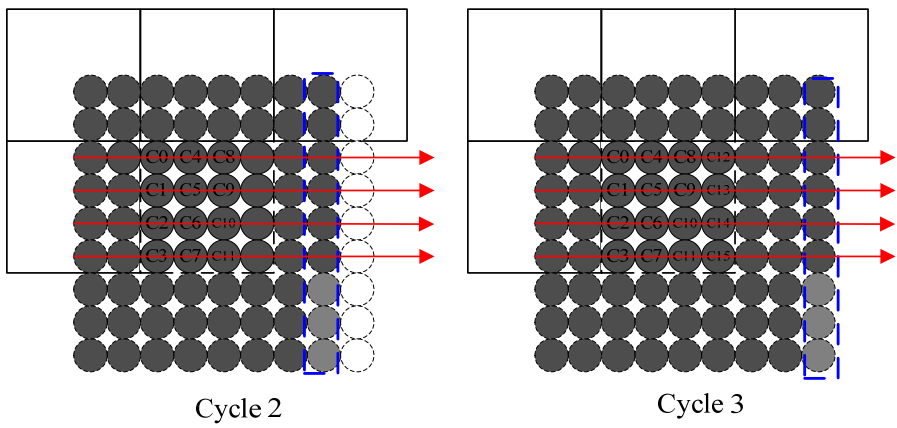
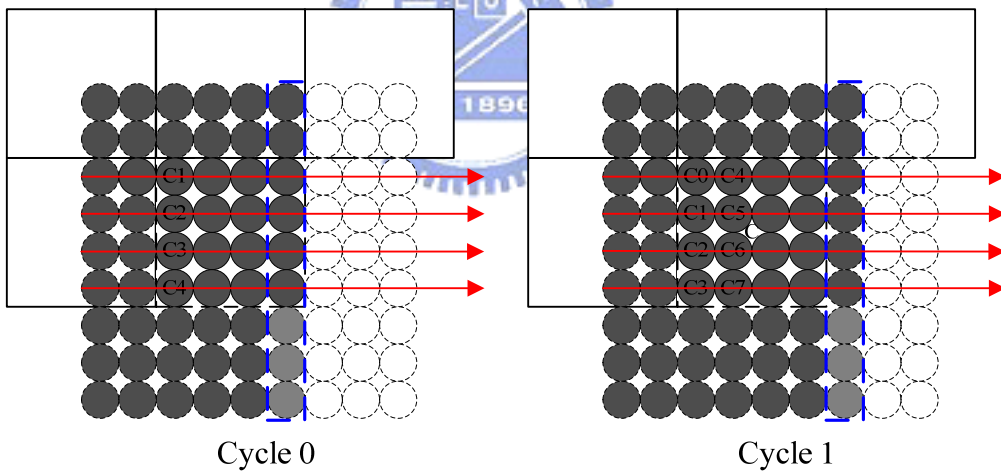
E2CMA exploits horizontal and vertical common region in interpolation search window between neighboring blocks to execute data-reuse operation. Because each 4x4 block needs 9x9 search windows to interpolate fractional pixels and word length is limited to 32 bits under data bus, it requires three cycles to load nine pixels of one column into entry. Therefore, it needs 27 cycles (3 x 9) to accomplish one 4x4 block interpolated in the worst case. The worst case means that MB type is decoded as 4x4.



(a)



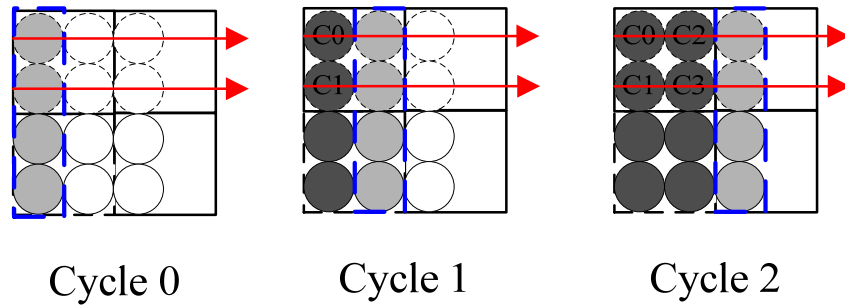
(b)



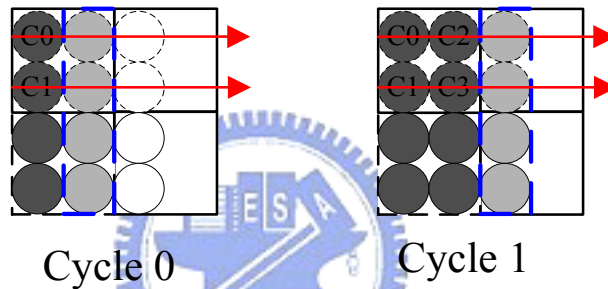
(c)



**Figure 3.12: Luma component interpolation: (a) Interpolating block 4 and (b) Interpolating block 3 (c) Interpolating block 6**



(a)



(b)

**Figure 3.13 Chroma component interpolation: (a) Interpolating 2x2-block 0 (b) Interpolation 2x2-block 1**

Some examples are given in Figure 3.12 (a)-(c) to illustrate the vertical and horizontal data-reuse approach by E2CMA. The charcoal-gray circle indicates pixels have been stored in buffer, and the light gray means pixels must be loaded from external frame memory. Figure 3.12 (a)-(c) depict three data-reuse cases: (a) horizontal data-reuse approach (b) vertical data-reuse approach (c) horizontal and vertical data-reuse approach. The MB type assumes 16x16 in these cases. Firstly, the horizontal data-reuse approach is given for interpolating block 4 in Figure 3.12 (a). The horizontal data-reuse approach is applied to content buffers for

executing a content-switch operation. Pixels in columns 0-4 have been stored in content buffers. Therefore, we only need to load pixels from external memory in column 5. After 12 cycles, 16 interpolated pixels in block 4 have been produced. The vertical data-reuse approach is illustrated in Figure 3.12(b) for interpolating block 2. In block 2, upper six pixels in each column have been shifted into Reuse-Register-File. Therefore, three lower pixels in each column must be fetched from external memory. We require one cycle to fetch three lower pixels from external memory and load upper six pixels from reused registers at the same time. Nine cycles are needed in this case. Last case is that horizontal-vertical data-reuse approach is shown in Figure 3.12(c). The least interpolating cycle for one block is 4 cycles for horizontal-vertical data-reuse approach. Because all pixels in column 0-4 and upper six pixels in column 6-9 is stored in content buffer and Reuse-Register File respectively, 4x4 block interpolated can be accomplished after four cycles. All MB types can be applied by E2CMA so that data-reuse utilization is increased excepting for MB type is 4 x 4,

Because of 4:2:0 chroma component and quarter precision of luma inter prediction, chroma inter prediction can achieve eighth motion resolution. E2CMA can be applied for chroma component interpolation as well. Similarly, for chroma component interpolation, some examples are given as Figure 3.13 (a)-(b). Chroma inter prediction must process based on 2 x 2 block and chroma interpolation search windows requires 3 x 3 pixels for each 2 x 2 block as shown in Figure 3.10 (b). For chroma component interpolation, block 0 of chroma component is interpolated is shown as Figure 3.13 (a). In this case, data-reuse approach can be not applied so that three cycles are required. Other case is shown in Figure 3.13 (b), E2CMA is used so that two interpolation cycles are needed. From Figure 2.12 (d), for chroma 2 x 2 block including A, B, C and D, the fractional sample i whose precision is eighth point. A reduction of required access cycles is 33% using E2CMA.

**Table 3.1 Analysis for different interpolating approach**

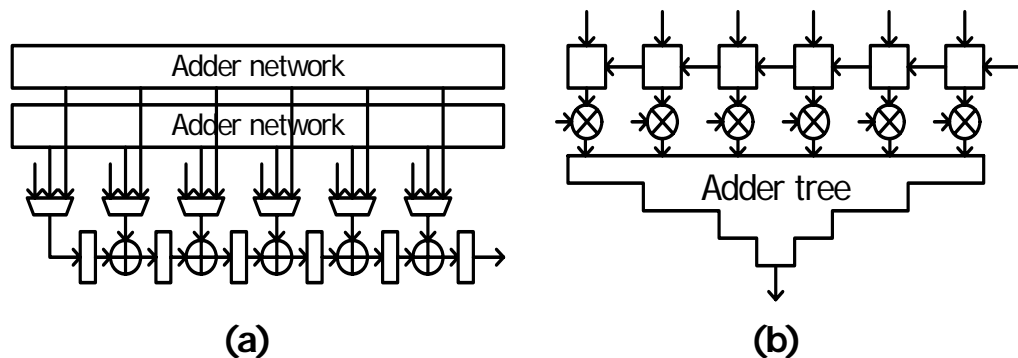
Interpolating approach	Required pixels per luma MB		Required pixels per chroma MB		MB_type for data-reuse approach
	Worse case	Best case	Worse case	Best case	
Row major approach	1296	1296	144	144	All
Column major approach	1296	936	144	144	direct, skip, 16x16, 16x8
Wang' s approach	1296	756 + 6CS	144	144	direct, skip, 16x16, 16x8
E2CMA	1296	126 + 6CS	144	108 + 6CS	All MB Except for 4x4 MB,

To give more generic and platform independent analysis, we analyze requisite pixels per MB for each interpolating approach. Table 3.1 lists required pixels per luma MB and chroma MB for different interpolating approach. Assuming that each motion vector contains fractional part, the best case has one motion vector and worst case has 16 motion vectors for one macroblock. Although requisite pixels of each approach are the same in worst case, requisite pixels of column major related approach are smaller than that of row major approach. Although column major related approach takes the best effect than row major approach, it requires additional synchronization buffer and degrades throughput due to different scan order approach with that of residual decoder. As for Wang's approach, few MB types can be data-reuse such as direct, skip, 16x16 and 16x8. Although larger block size (skip, 16x16, 16x8) occupies up to approximately 50% ~ 90% proportion depends on bit rate. For higher bit rate, improvement of Wang's method is limited. Oppositely, E2CMA can be applied all MB type except for MB type is 4x4. Therefore, the performance is better than previous approach.

### 3.3.2 Combined Luma/Chroma Interpolation Architecture

In this subsection, several different works related to interpolator designs of which have been published will be introduced. From above discussion, reviewing the fractional

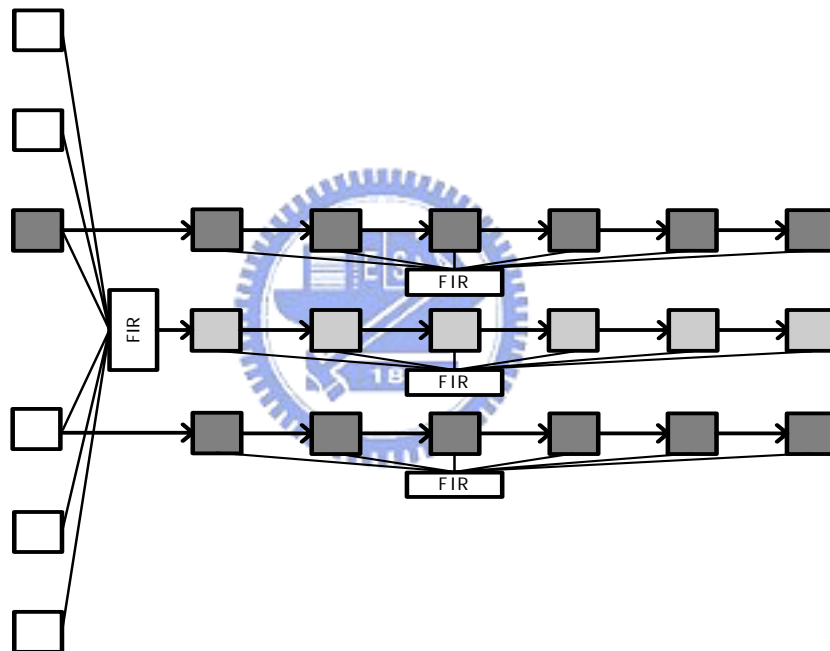
interpolation for H.264/AVC in Figure 2.11, 6-tap FIR with (1, -5, 20, 20, -5, 1) coefficient and bilinear filter are needed for half and quarter precision of luma component interpolation in H.264/AVC video decoder. For cost and PSNR loss acceptable consideration, Lie's 4-tap diagonal FIR filter and three-stage recursive algorithm is proposed in [21], and Chen's HVBi, bilinear filter in both horizontal and vertical direction, and VBi, vertical bilinear horizontal FIR, schemes are also introduced in [22]. However, when frame sequence is very long for supporting B-slices, such as I + 9 P + 4B, the propagation of PSNR loss may cause the heavy degradation of video quality, especially in high definition frame format such as 1080HD. Oppositely, considering PSNR losses and standard-compatible design, Chien's [23] and He's [24] have proposed adder-chain and adder-tree based design respectively. These two types which depicted in Figure 3.14 are categorized into 1-D linear filter design. For cost consideration, multipliers can be simplified to adders and shifters. 1-D linear interpolator is suitable for Q-CIF video sequence in mobile applications; however, as for HDTV video sequence, throughput is a very important issue and long execution cycles in 1-D linear design may lead to poor throughput. As for another choice, Chien's [23] also proposed separate 1-D design that separates horizontal and vertical interpolation and processes in parallel based on 4 x 4 block size. This design induces better throughput, although it may need more storages. Figure 3.15 shows separate 1-D interpolator design without processing in a parallel way.



**Figure 3.14 (a) Adder-chain based [23] (b) Adder-tree based [24] 1-D linear interpolator design**

**Table 3.2 Comparison of execution cycles for different architectures**

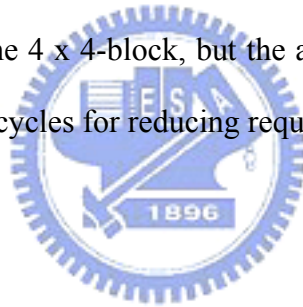
Interpolation Architecture	Interpolating cycles
Adder-chain based 1-D[10]	57
Adder-tree based 1-D[10]	52
Separate 1-D (no parallel) [10]	36
Separate 1-D (2 parallels)	18
Separate 1-D (4 parallels) [10]	9

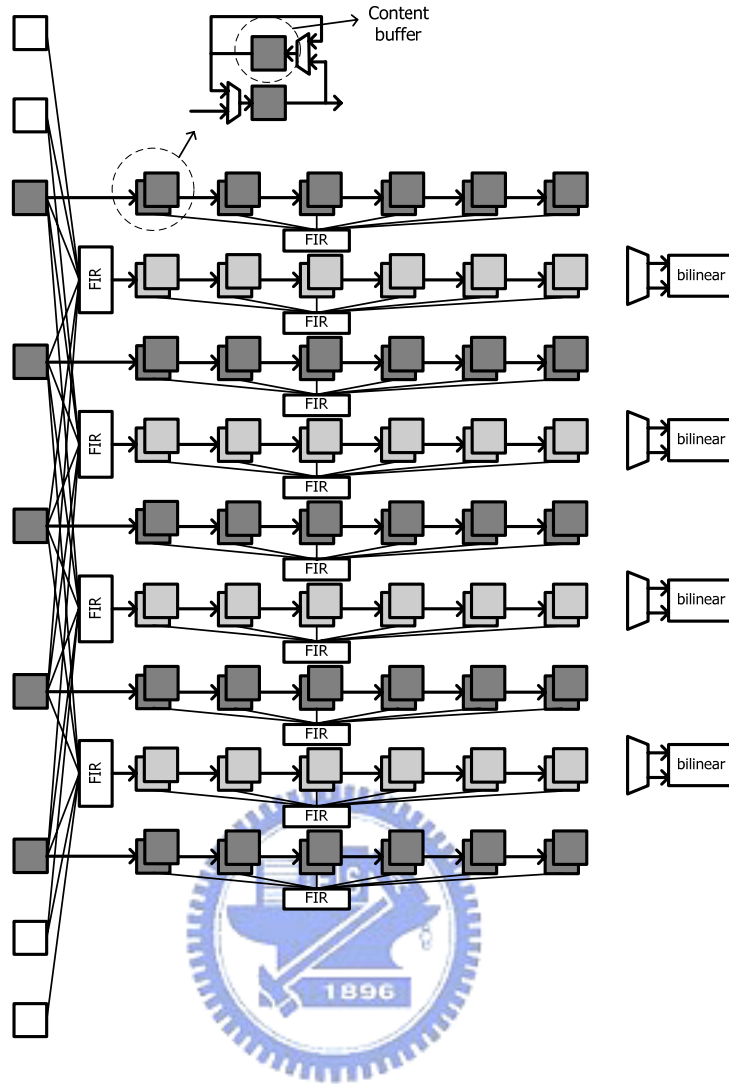


**Figure 3.15 Separate 1-D interpolator design (no parallel)**

Assuming that all  $9 \times 9$  interpolated data for each  $4 \times 4$  block are ready and they can be accessed randomly, Table 3.2 lists the execution cycles for different architectures. For adder-chain based 1-D design, the first result outputs after the 6<sup>th</sup> clock cycle. Two adder-networks are used to overlap each row inputs and eliminate the latency overhead except the first one. The total number of cycles required is 57 ( $5 + 4 \times 9 + 4 \times 4$ ) whose detailed

operation is described in Chien's. For adder-tree based 1-D design, the row data can be loaded in a parallel fashion without shift one-by-one, hence the latency overhead does not exist and total numbers of cycles are 52 ( $4 \times 9 + 4 \times 4$ ). As for separate 1-D design, the first data outputs at the 6<sup>th</sup> clock cycle and the following 3 data generates after 3 clock cycles. Therefore, the separate 1-D design without using parallel architectures needs 36 ( $(6 + 3) \times 4$ ) cycles to complete interpolation of one  $4 \times 4$  block. Wang's [10] presented a separate 1-D with 4 parallel designs and the required content buffers are  $6 \times 9$  pixels for 4-parallel design shown in Figure 3.16. Similarly, separate 1-D design with 2 and 4 parallel requires 18 ( $(6 + 3) \times 2$ ) and 9 ( $6 + 3$ ) cycles respectively. In addition, 4-parallel separate 1-D architecture is best selection due to smaller required execution cycles that can be hidden below data-read cycles from frame memory. Wang's architecture which uses separate 1-D with 4-parallel can obtain smaller execution cycles for one  $4 \times 4$ -block, but the architecture can be simplified to obtain interpolating pixels by smaller cycles for reducing required bandwidth.





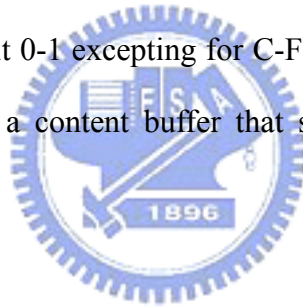
**Figure 3.16 4-parallel separate 1-D luma interpolator with content buffer**

From previous discussion, we propose a *combined luma/chroma interpolator* (CLCI) architecture for reducing cost and complexity of interpolator, which can support E2CMA. The proposed CLCI is based on separated 1-D with 4-parallel interpolator architecture. We decompose 2-D FIR interpolator into vertical and horizontal 1-D FIR interpolators. For the luma interpolation, the half samples are interpolated by performing a 6-tap filter, and quarter samples are produced by using a bilinear filter. Considering the chroma interpolation, 1/8 samples are obtained by CLCI without requiring additional interpolators. The CLCI shown in Figure 3.17 consists of four CLCI units, bilinear, Reused-Register-File and input entry units. According to the data-reuse status, input entry unit selects and packs the pixels loaded from

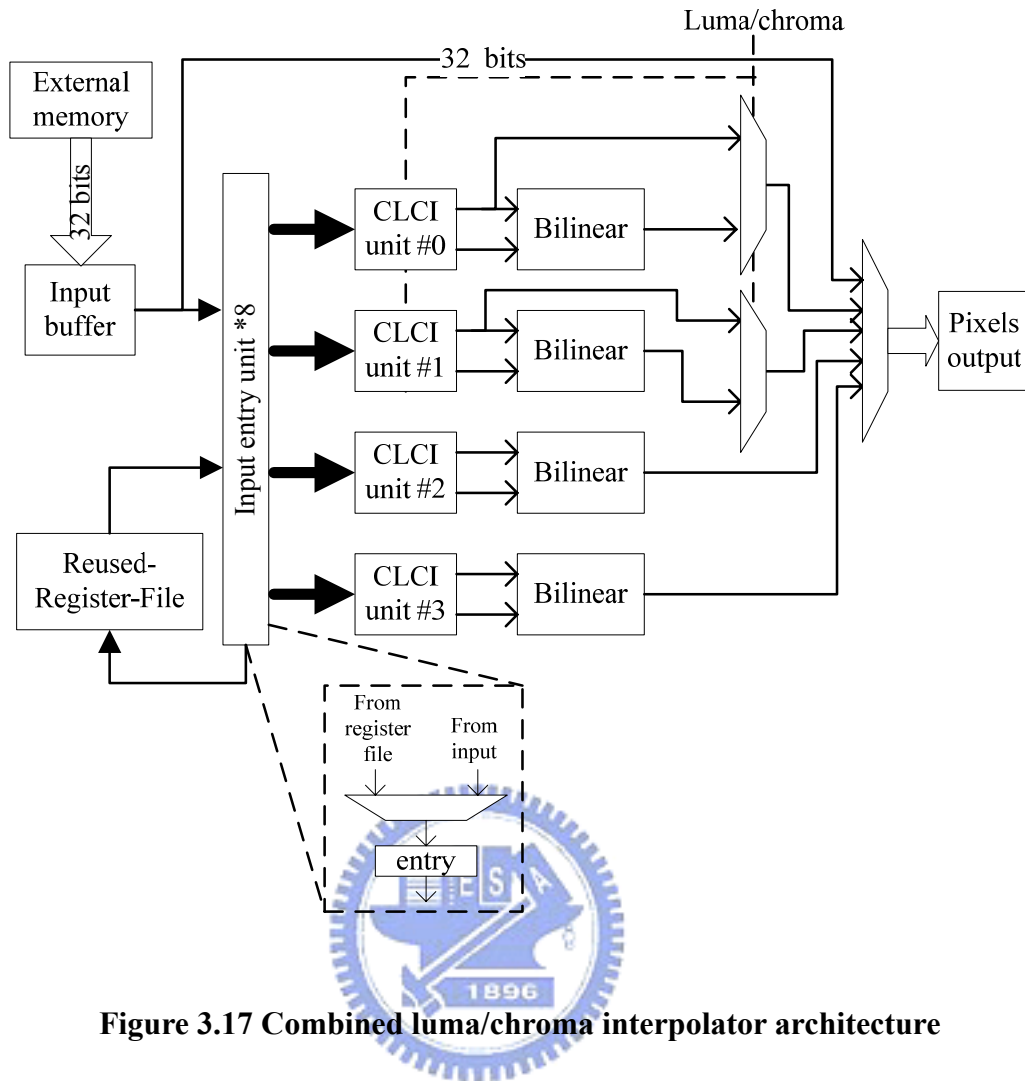
input buffers or Reused-Register-File to CLCI units.

Not only luma but also chroma interpolation can be processed by the proposed CLCI. When a luma block is interpolated, all CLCI units are invoked; otherwise, CLCI unit 0-1 is applied and transmits to pixels output without bilinear filter. Because CLCI unit 0-1 is needed to interpolate chroma component, Combined FIRs (C-FIRs) of CLCI unit 2-3 are replaced by FIR. The sizes of Reused-Register-File for vertical data-reuse are 21x48 bits. Pixels are loaded from input buffers to transmit into input entry units and are shifted into Reuse-Register-File at the same time. For horizontal data-reuse approach, content buffers of CLCI only are 48 (6 x 8) less than Wang' design so as to save gate count.

Furthermore, each CLCI unit is shown in Figure 3.18. CLCI unit 0-1 includes 6 entries, 3 Combined FIRs(C-FIR), 6 integral elements and 6 fractional elements. CLCI unit 2-3 includes the same elements as CLCI unit 0-1 excepting for C-FIRs are replaced by FIR. Each element consists of a shift buffer and a content buffer that stores pixels for horizontal data-reuse approach.

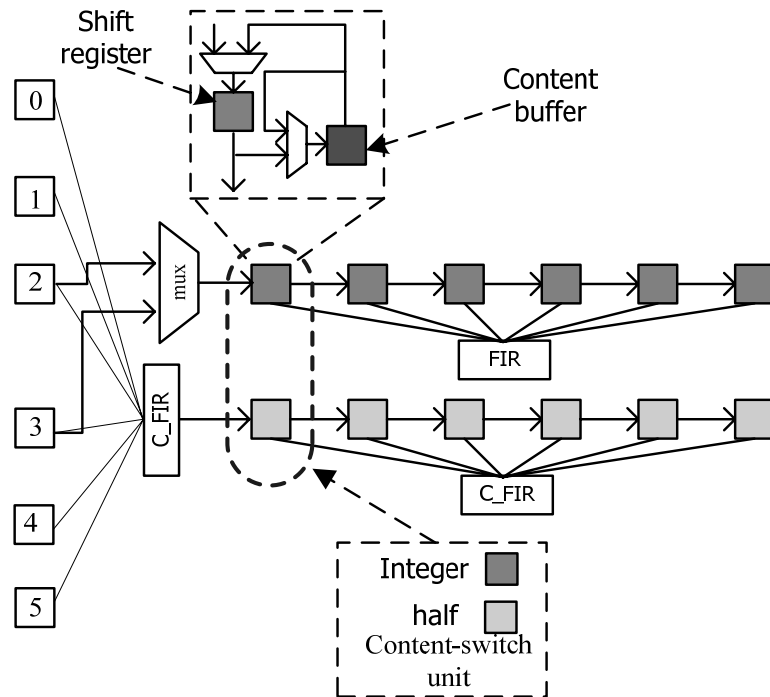






**Figure 3.17 Combined luma/chroma interpolator architecture**

Considering the horizontal data-reuse approach, shift buffers and content buffers are exchanged each other by one cycle for content switch operations. Input of integer element is selected by the multiplexer according to quarter fractions of motion vectors. For the luma interpolation, half samples are derived from C-FIR after 6 shifting cycles. For the chroma interpolation, 1/8 samples are obtained after 2 cycles due to 2x2 interpolation search regions in one chroma pixel.



**Figure 3.18 Combined luma/chroma interpolator unit**

As for luma and chroma interpolator for H.264/AVC described as above, the adder of luma and chroma component interpolation can be shared as well. Figure 3.19 shows the proposed *Combined FIR filter (C-FIR)* which can combine luma with chroma interpolating data paths. Because the chroma interpolation is based on 2x2 block processes, only two combined FIR filters are required to obtain 1/8 samples in CLCI units 0-1 respectively. By the CLCI architecture, the chroma interpolator can be combined into the luma interpolator, reducing additional hardware cost. The cost penalty of C-FIR design is MUX x 2, shifter x 3 and bitwise AND x 6 when compared with the FIR design proposed in Chen's [22]. The decoding path of luma FIR filter and chroma filter are illustrated in Figure 3.20.

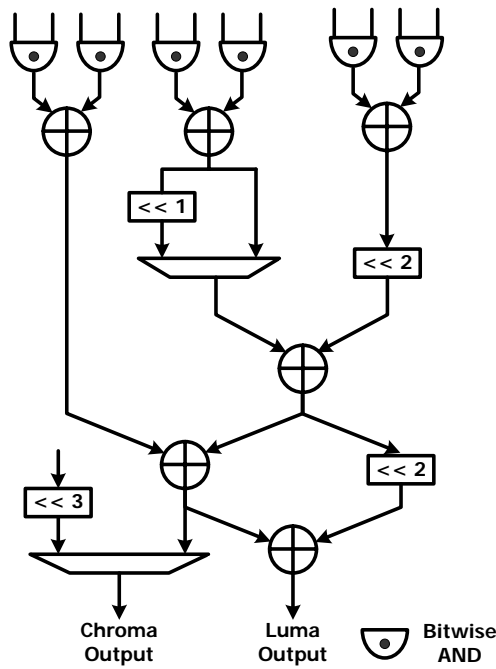


Figure 3.19 Combined luma/chroma FIR.

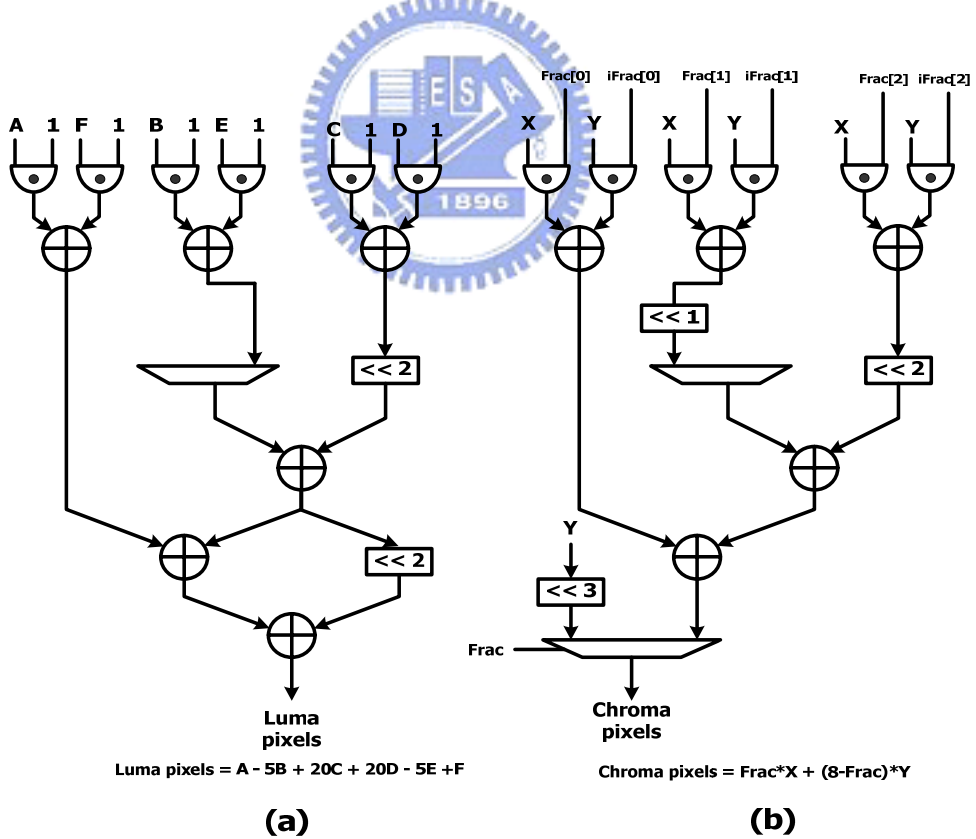
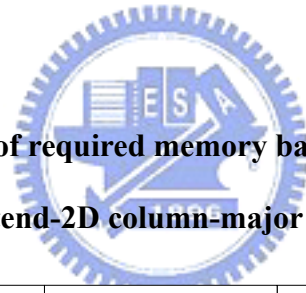


Figure 3.20 (a) Process path for luma component interpolation (b) Process path for chroma component interpolation

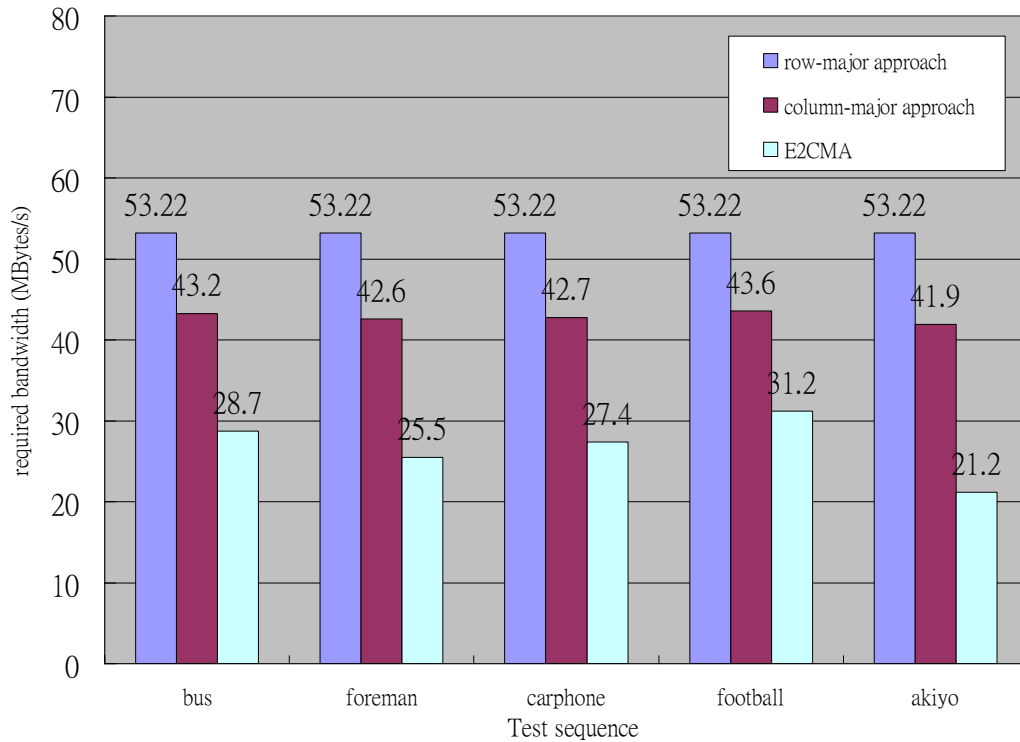
### 3.3.3 Simulation Results

We use five CIF resolution videos as our test sequence and adopt main profile as the simulated platform. The decoding order is fixed at I-B-B-P-B-B-P-B-B sequences. We assume all MB are fractional, which means that all MB need to be interpolated. The proposed bandwidth-efficient motion compensation architecture reduces the data refresh probability. In other words, data-reuse probability can be increased. Only a 4x4-block which occupies 4% can not be reused by using our proposal. Table 3.3 lists simulation results at 100MHz using E2CMA at 30fps, and the bit rate is 512Kbps. The improvement of our proposal is up to 60%. Besides, the required bandwidth is painted in Figure 3.21. From Figure 3.21, the E2CMA may obtain smaller required bandwidth so as to reduce bandwidth over external BUS.



**Table 3.3 Simulation results of required memory bandwidth (MByte/s) per MB by using extend-2D column-major approach.**

Test sequence	Row-major interpolating order	column-major interpolating order	Extend_2D column-major approach	Improvement(%)
Bus	53.22	43.2	27.7	47.95
Foreman	53.22	42.6	25.5	52.09
Carphone	53.22	42.7	27.3	48.7
Football	53.22	43.6	27.2	48.89
Akiyo	53.22	41.9	21.2	60.17



**Figure 3.21 Required bandwidth for different data-reuse approach**

Furthermore, we have implemented and synthesized the CLCI hardware using Cadence's RTL Compiler with UMC 0.18um cell library. Gate count of the proposed CLCI architecture is 44.6% less than Wang's design which includes 6-tap separate-1D luma filter and 1/8 chroma filter. We only use 4 C-FIR's, 8 FIR's and 8 input entries to perform luma and chroma interpolations.

Total gate counts of the proposed motion compensator are 83.5K. The gate count contains motion vector generator with direct mode coding, interpolator with Reuse-Register-File, and weighted prediction module. Considering an interpolator part, a comparison with other previous designs is shown in Table 3.4. The reuse-register file can be replaced by SRAM to reduce gate count. The gate count of interpolator core without reuse-register-file is 12.5k as table 3.4. The gate count of proposed combined luma/chroma interpolator can be reduced about 38% comparing with related work. In 30fps 1080HD

(1920x1088) format, proposed design can be operated at 100MHz for a real-time system.

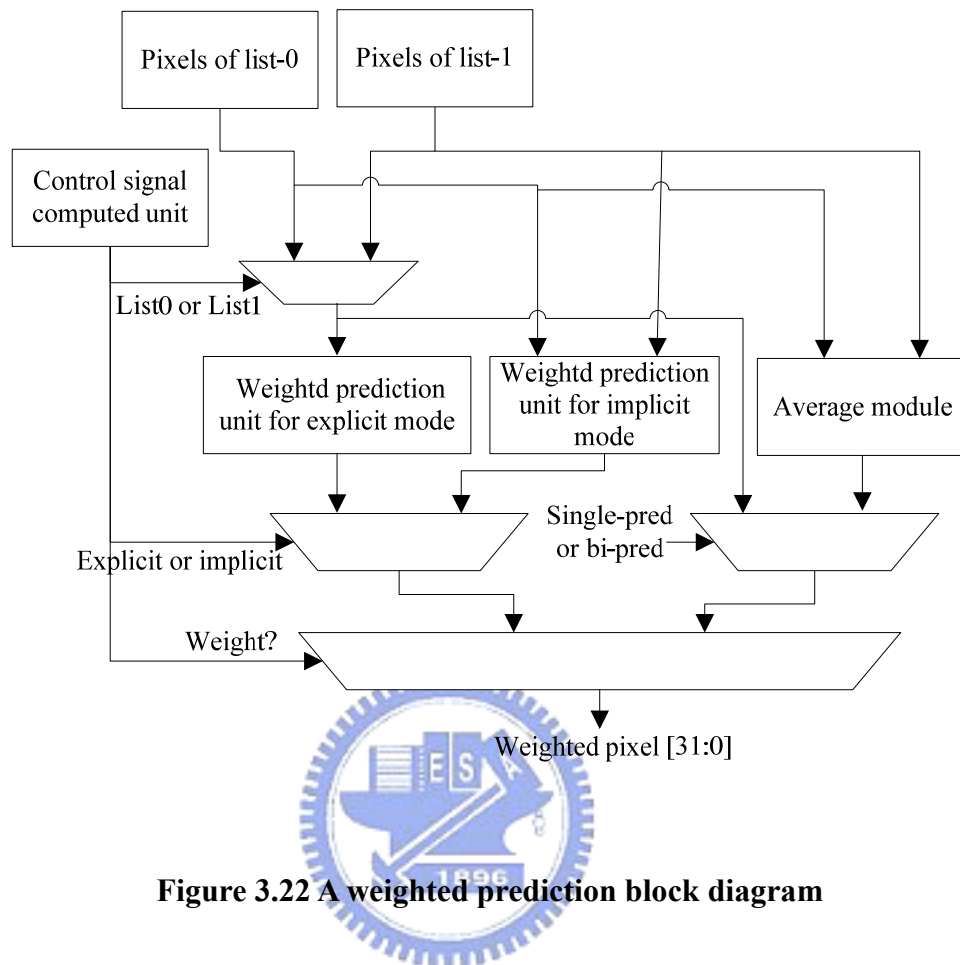
**Table 3.4 Comparison of interpolator architecture with other designs.**

	Ref. [2]	Ref. [5]	Ref. [6]	Proposed
Architecture	Separate 1-D	Separate 1-D	Separate 1-D	Combined LUMA/CHROMA Separate 1-D
Slice type	P slices	P slices	P slices	P, B slices
Component (LUMA)	Horizontal FIR x 9	Horizontal FIR x 5	Horizontal FIR x 8	Combined FIR x 4
	Vertical FIR x 4	Vertical FIR x 11	Vertical FIR x 4	Horizontal/Vertical FIR x 8
	Bilinear x 4	Bilinear	Bilinear x 4	Bilinear x 4
Component (CHROMA)	1/8 filter x 2	1/8 filter x 3	NA	
Interpolator Gate Count	20,686 (0.18 um) (Interpolator core)	23,872 (0.18 um) (Interpolator core)	21,506 (0.18 um) (Interpolator core)	12,580 (0.18 um) (Interpolator core)

### 3.4 Weighted Prediction

A weighted prediction (WP) is supported in the Main profile of the H.264 standard. The weighted prediction locates the sequence parameter set for P and B slices. There are two WP modes: an explicit mode is supported in P and B slices, and an implicit mode only supports B slices. In the explicit mode, these WP parameters may be coded in the slice header. In implicit modes, these parameters are derived based on the relative distance of the current picture and its reference pictures.

We implement a division-free WP module in our motion compensation architecture. We need one cycle to perform WP operations. The WP architecture is shown in Figure 3.22. When the WP is enabled, interpolated pixels are loaded into explicit weighted or implicit prediction. After applying the weighted prediction, pixels are derived by adding residual data.



**Figure 3.22 A weighted prediction block diagram**

### 3.5 Summary

In this chapter, we present bandwidth-efficient motion compensation architecture for HDTV H.264 decoder and support 1080HD 30fps@level4.0 format. To overcome the tremendous data access from external frame memories, the proposed data reuse technique for fractional motion compensation can efficiently reduce the requisite reference data in the high motion precision for the advanced video standard, H.264/AVC. An Extend-2D column major approach is presented, which reduces 50%-60% bandwidth with B-slices under external data BUS. The proposal implements all advanced features including MV generators with direct modes, combined luma/chroma interpolator, and weighted prediction of H.264/AVC main

profile. As for sharing design issue for luma and chroma component, proposed CLCI architecture saves 44 % gate count compared with luma and chroma separate design. Besides, the CLCI architecture is also suitable for high throughput HDTV video decoder. Altogether, memory usage and bandwidth are optimized by our proposed design.





# **Chapter 4**

## **Bandwidth-Efficient SDRAM Memory Controller**

---

In general, video decoder should deal with large amount of data due to a real-time high-quality decoding demand. Specifically, bi-prediction requires more data size than single-prediction to store several decoded reference frames. In addition, motion compensation which supports direct mode coding has to store motion vector of first list-1 frame called as co-located motion vector to predict motion vector of current macroblock. Therefore, not only reference frame pixels but also co-located motion vectors need large storage. The off-chip memory is required to store large data including reference frame pixels and co-located motion vectors, but the speed of memory is slow. Thus, the decoder system performance strongly depends on the memory bandwidth between motion compensation module and external memory. To meet this requirement, high bandwidth and large size memory are its penalties.

In order to improve memory bandwidth, DRAM families such as synchronous DRAM (SDRAM) is now widely used in high-performance video systems. For example, SDRAM has two key and special features: burst access mode and multiple bank architecture. The burst access mode makes it possible to access a number of data by changing only column addresses, and the multiple bank architecture can hide memory cycles needed for row-activations and precharge by accessing different banks alternatively. Since the number of additional cycles needed for row changes is considerable, we have to reduce the number of row changes or hide these cycles by using the high-performance features of SDRAM. This observation motivated

us to research an array address-translation technique to minimize the number of overheads cycles needed for row changes. For system requirement equip high performance, SDRAM is suitable prior to store large amount of video data in HDTV system. SDRAM has advantage that is large size, but memory overhead become high. Therefore, data would be transmitted during limited access time under external BUS. Hence, we can focus on reduction of memory access latency to obtain less execution cycles. Besides, limited access time and bandwidth is bottleneck of real-time system as HDTV system. For improving bandwidth and access time in HDTV system, we propose bandwidth-efficient memory controller in this paper.

In this chapter, we propose SDRAM memory-controller for HDTV H.264 main profile decoder to increase efficiency of data transmission. Firstly, we briefly introduce the modern SDRAM architecture and basic operations in section 4.1.

## 4.1 SDRAM Module Characteristics



In this section, we first introduce the SDRAM module characteristics and its basic concept.

### 4.1.1 Basic Concept of SDRAM

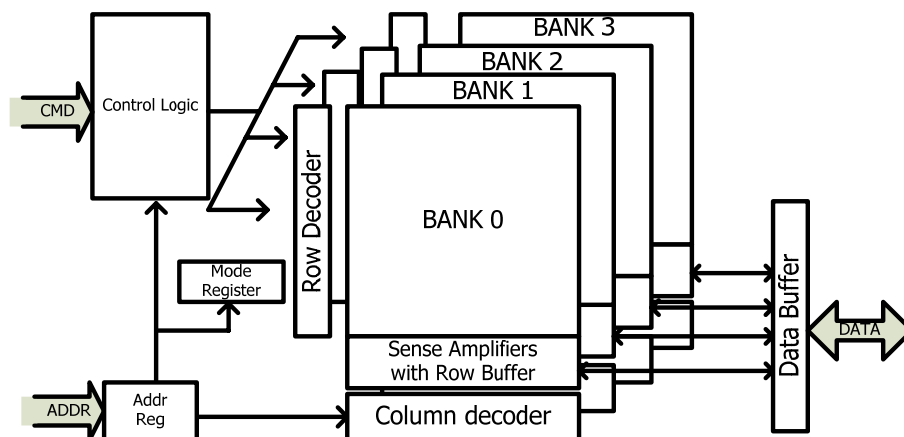


Figure 4.1 Modern SDRAM architecture

A simplified architecture of a 4-bank modern SDRAM is shown in Figure 4.1. It is basically a three-dimensional architecture with dimensions of bank, row, and column. Four banks share the address and command buses, and each bank has individual row decoder, sense amplifier, and column decoder, while data and address buses of SDRAM are shared by all banks. The mode register stores several SDRAM operation modes by user-configured process, which includes burst length (BL), Column address strobe (CAS) Latency (CL) or burst type (sequential / interleave). The content of mode register updates according to command issued from address buses. A complete memory access operation contains three steps including row activation, column access, and pre-charge. These commands are briefly introduced as follows.

➤ **PRECHARGE COMMAND**

The precharge command is used to deactivate the open row in a particular bank or the open row in all banks. The precharge command requires the use of SDRAM address bus to indicate which bank shall be precharged. In address bus, input A10 determines whether one or all banks are to be precharged, and in the case where only one bank is to be precharged, inputs BA0 and BA1 select the required bank. When all banks are to be precharged, inputs BA0 and BA1 are treated as “Don’t Care”. While processing the precharge command, the addressed bank is not allowed to accept any other commands. Bank(s) will be available after PRECHARGE command. The command latency is called tRP. Figure 4.2 (a) indicates SDRAM resource utilization while a precharge command is issued.

➤ **ACTIVE COMMAND**

The active command is used to open (or activate) a row in a particular bank for the subsequent column access. The value on the BA0 and BA1 inputs selects the bank, and the address provided on inputs A0 -A11 selects the row. This row remains active (or open) for accesses until a precharge command is issued to the indicated bank. To issue an active command, the address bus must be used to select the bank and the row which will be activated. After accepting the active command, SDRAM needs a latency called tRCD to accomplish the

command and no other banks are permissible due to the parallel processing capability of each bank. Figure 4.2(b) indicates SDRAM resource utilization while an active command is issued.

➤ **COLUMN ACCESS COMMAND (READ or WRITE)**

The column access command is used to initiate a burst read/write access from/to an active row. The value on the BA0 and BA1 inputs selects the bank, and the address provided on inputs A0-A8 selects the starting column location for read/write command. The value on inputs A10 determines whether or not auto precharge is used. If auto precharge is not selected, the row will remain open for subsequent accesses. Once a row of a particular bank has been activated, the column access command can be issued to read/write data from/to SDRAM. Figure 4.2 (c) and (d) indicate the SDRAM resource utilization while read and write column accesses are issued. To issue either a read or write column access command, SDRAM address bus is required to indicate the bank and the column of the open row in that bank. For a write column access, SDRAM data bus is needed to transfer write data at the time where the command is issued until whole burst transfer is over. On the other hand, SDRAM data bus occupied several cycles called CAS latency after the read column access is registered for the same period as the write column access.

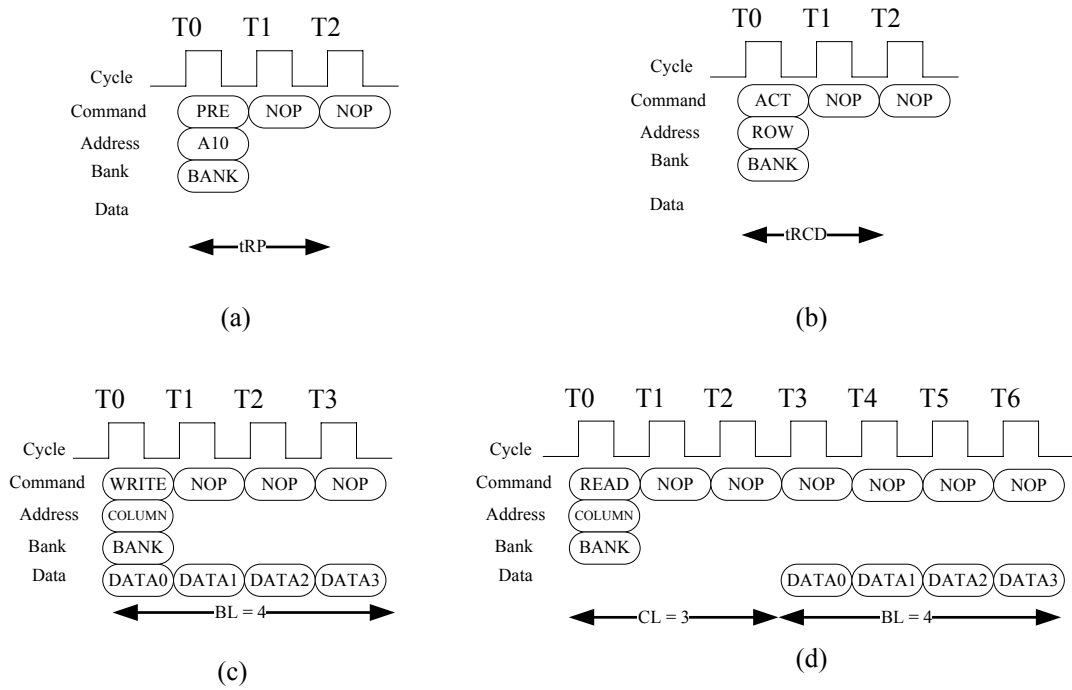


Figure 4.2 SDRAM resource utilization of several commands: (a) PRECHARGE (b)

ACTIVE (c) WRITE (d) READ

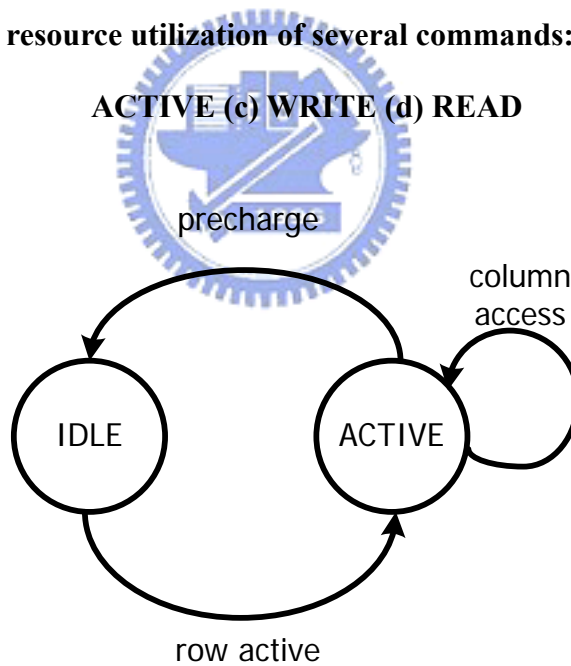
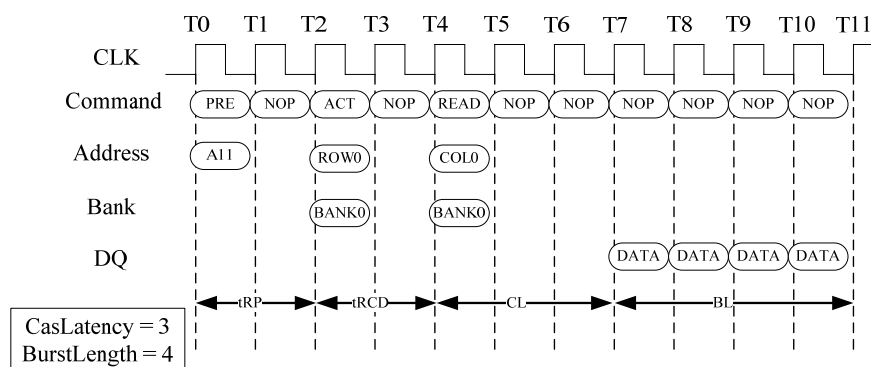


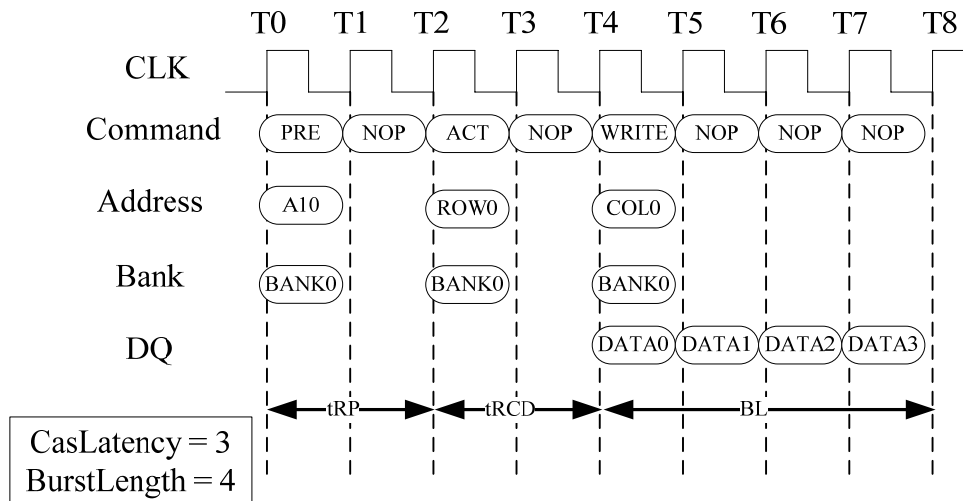
Figure 4.3 Simplified bank state diagram

These number of commands issued depend on the addressed bank states. The simplified bank state diagram is depicted in Figure 4.3. Precharge command must be issued in the initial status. If an access is addressed to a particular bank in the IDLE state, a row activation

command with the particular bank address is sent to open (or active) one row in a particular bank firstly and the designated row address is issued from the address bus. The operation of this command is transmitting the row data into the row buffer of the selected bank and row active operation needs an active latency called  $t_{\text{RCD}}$  (ACTIVE to READ or WRITE delay) to accomplish this operation. Then, column access command is employed to access sequential data or single data according to the defining burst length and burst type in the mode register. The mode register is used to define the specific mode of operation of the SDRAM. This mode includes the selection of a burst length, a burst type, CAS latency, operating mode, and a write burst mode. The mode can be programmed by LOAD MODE REGISTER command in the initial status and will retain the stored information until it is programmed again or the device loses power. The read/write data are accessed through the same data bus. For read operation, the valid data-out element from the starting column address will be available following the CAS latency after the READ command, as shown in Figure 4.4. For write operation, the first valid data-in element is coincident with the WRITE command, as shown in Figure 4.5. Finally, a precharge command must be issued before opening a different row in the same bank, whereas a precharge and active command need not to be issued if the following access still in the same row and bank. After precharge command is issued, the selected bank cannot be accessed during the precharge latency named  $t_{\text{RP}}$  (PRECHARGE command period.)



**Figure 4.4 Burst read operation with CasLatency=3 and BurstLength=4.**



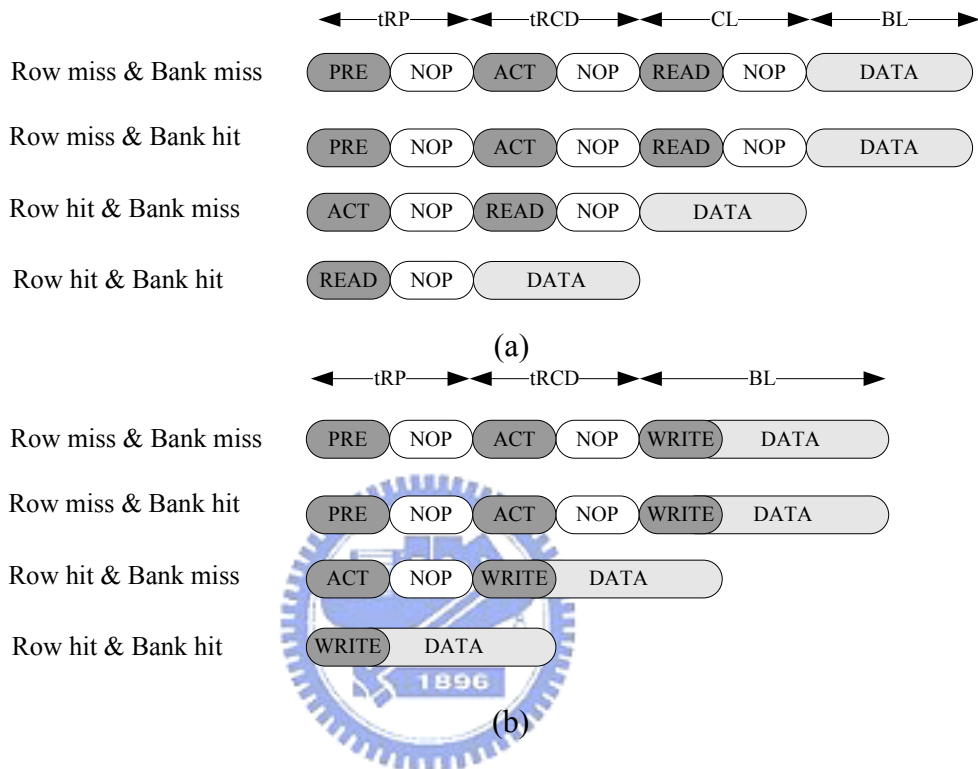
**Figure 4.5 Burst write operation with CasLatency=3 and BurstLength=4.**

#### 4.1.2 Access Latency Analysis

Lee *et al.* discussed different access latencies of different access statuses in [30]; however, detailed classification is required for exquisite access command scheduling. The memory behavior model used in our design is Micron's MT48LC2M32B2P-5 512Mb SDRAM [27]. Table 4.1 lists three different allowable maximum operating frequencies provided in this SDRAM according to the CAS latency stored in mode registers. Obviously, when setting CAS latency to 3, the SDRAM can provide higher operating frequency. However, higher operating frequency induces more stall cycles which is demanded for each read column access. Therefore, the CAS latency should be set carefully for different suitable applications. For instance, 50 Mhz with CL=1 is enough for Q-CIF format in mobile device while 166 Mhz with CL=3 is required for large frame size format such as SDTV or HDTV format.

**Table 4.1 CAS latency**

CAS Latency	1	2	3
Allowable operating frequency (MHZ)	$\leq 50$	$\leq 100$	$\leq 166$

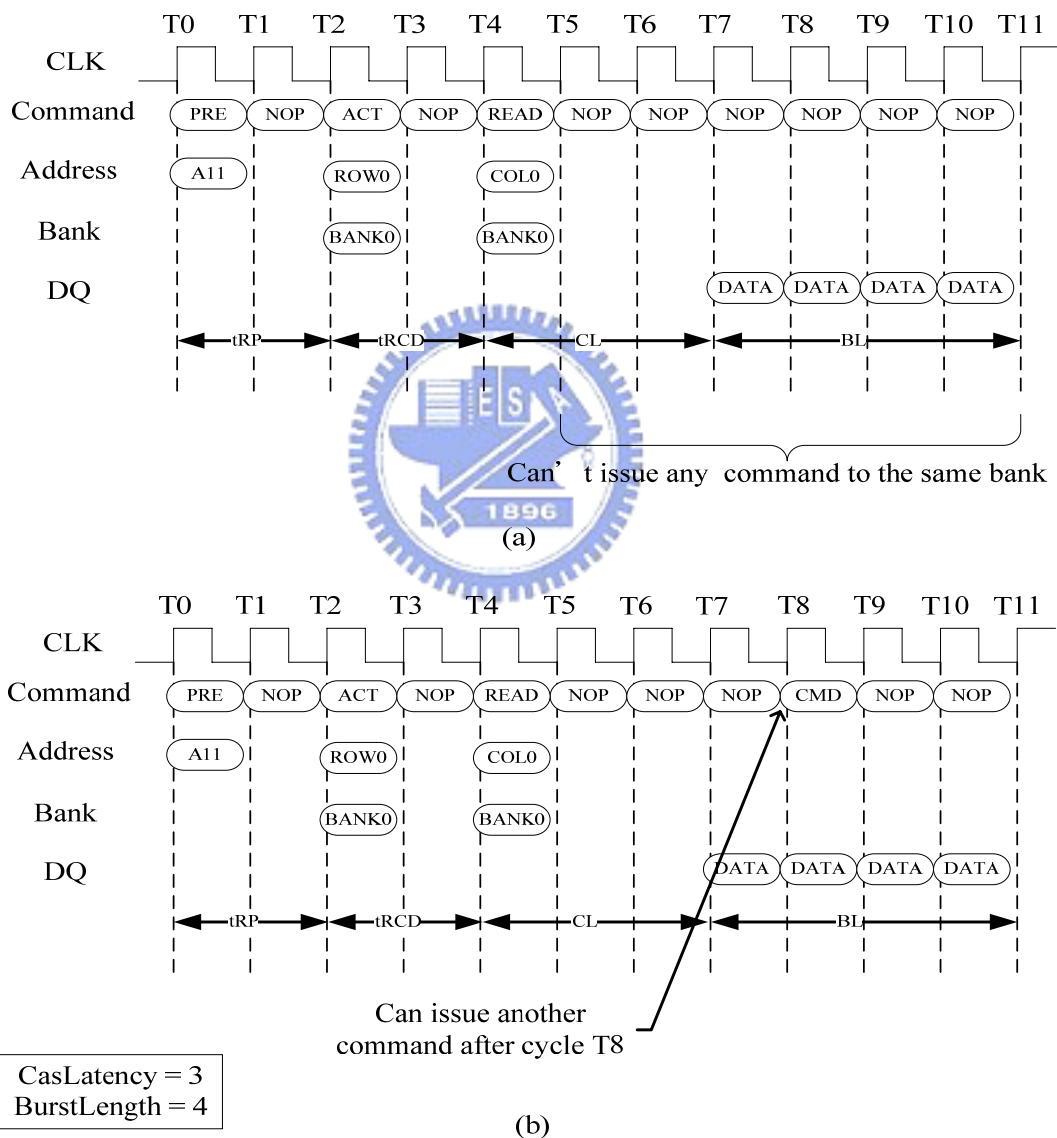


**Figure 4.6 Access latency for CL=2 (a) read access latency, (b) write access latency**

Figure 4.6 illustrates read/write access latency under different statuses when CAS latency equals 2. Bank-hit with row-miss status means that the activated row in selected bank is not identical to the incoming issued access command and it requires additional latency (PRECHARGE + ACTIVE + CAS + NOPs) for read access and (PRECHARGE + ACTIVE + NOPs) latency for write access. Bank-miss with row-miss status means that incoming bank address is different from bank address for previous command and the selected row for the incoming bank address is not activated. For this status, required latency and issued command is the same as that of row-miss status. Bank-miss with row-hit status indicates that the



incoming row has been activated in the previous command although the incoming bank is not equal to the previous one. Bank-hit with row-hit status means that incoming address is identical to the address of previous issued command. For these statuses, the column access can be directly issued for sequential access and only read access leads to CAS latency. Based on the above discussion, memory scheduling method can overlap the sequential column-access commands and hide full or partial latencies to reduce redundant cycles.



**Figure 4.7 (a) READ command with auto precharge, in the precharge period (tRP), SDRAM cannot issue another command in the same bank (ex: bank 0). (b) READ command without auto precharge, another command can be issued until the tRP is met.**

SDRAM also supports another precharge method called auto precharge without requiring an explicit precharge command. A PRECHARGE command of bank/row together with READ/WRITE command is automatically performed upon completion of READ/WRITE burst access. For the full-page burst mode, auto precharge does not apply. Auto precharge ensures that the precharge is initiated at the earliest valid stage within a burst. It is convenient for bank/row of the following data access is not the same as current data access. Precharge command may be not issued by memory controller. As shown in Figure 4.7, in the precharge period, it cannot issue another command to the same bank until the precharge time ( $t_{RP}$ ) is completed. If the following command must active to the same bank, the overlap scheduling is limited to this situation such that the following command can be issued only until the completion of  $t_{RP}$  period or reorder with the other command. For another disadvantage induced by auto precharge, READ/WRITE command with auto precharge indicates that SDRAM always de-active the selected bank at the end of a burst command implicitly. If the following data access still issues the same bank, it must waste time to re-active the same bank and lead to longer latency at the same time, and command issued is flexible. Therefore, we select manual precharge (without auto precharge) rather than auto precharge in our memory access interface design.

## 4.2 *Memory Controller Organization*

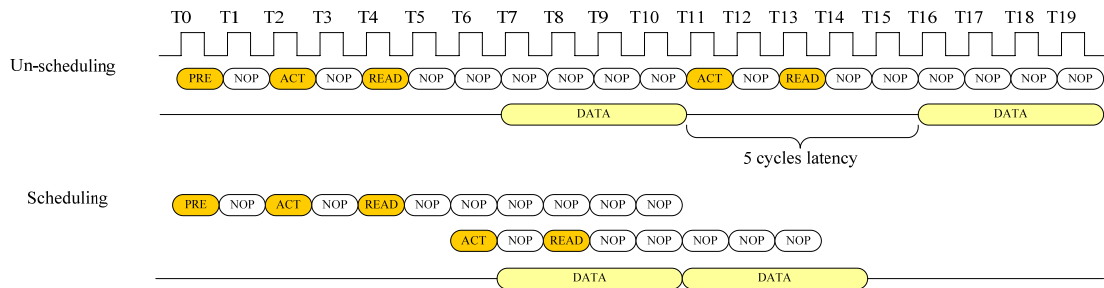
As the section 4.1, we introduced concept and limitation of SDRAM used. For most multimedia application with external memory, data access may cause bottleneck of entire video system. In H.264/AVC's main profile specification, large data are accessed between external memory and video core including pixels and motion vector. Pixels are accessed for motion compensation, and motion vectors are transmitted for direct mode prediction. For different data types, separate memory is applied by additional approach leading to lower

memory utilization. In this section, we proposed bandwidth-efficient memory controller organization to improve bandwidth under external bus efficiently. Our design can use unique SDRAM to store all required data for supporting multiple reference pictures and data of different type can be stored in same SDRAM. It is suitable to H.264/AVC main profile video decoder. A suited data mapping is applied so that data arrangements in SDRAM are tight. In section 4.2.1, we will show motion vector and frame pixels arrangement in SDRAM. For the storage of multiple reference frames, we present a novel memory mapping to increase memory utilization. Section 4.2.3 shows architecture of memory controller design, and will introduce hardware of each module within proposed memory controller. Finally, a comparison with previous designs related memory controller is made in section 4.2.4.

#### **4.2.1 Memory Access Scheduling**

The target of memory access scheduling is overlapping or reordering consecutive SDRAM commands (PRECHARGE, ACTIVE, CAS, and READ/WRITE) to improve bandwidth utilization and reduce access latency. Because the external access of video decoder is a bandwidth-sensitive channel [30], memory access scheduler must compress and even reorder SDRAM commands to achieve high bandwidth utilization. Considering the read-access and write-access respectively, the required frequency of write-access has high correlated with the ability of residual decoder in H.264 and the property of decoding bitstream, while the required frequency of read density is as tight as possible. For high bit-rate video sequence, the decoded bitstream contains more coefficients and higher precision of decoding token that may induce more requisite decoding cycles. In this situation, the write-access becomes less bandwidth-sensitive and the density of write access is not necessary very tight. The poor design of residual decoder, de-blocking filter also affects the bandwidth utilization of write access. Unlike the limitation of write access described above, read access needs high density of access scheduling because of its high bandwidth-sensitive channel. Read requests

are only sent by motion compensation, hence the bandwidth utilization of read access is influenced by the memory scheduler design, data arrangement in SDRAM and the handshake command scheme of motion compensation. The characteristics of write/read-access discussed above are summarized in Table 4.3.



**Figure 4.8 Two un-scheduling and scheduling read memory accesses for bank-miss and row-hit**

Considering read/write access from/to frame memories, the requirement of write-access is low or mediate density depend on the capability of residual decoder, whereas motion compensation requires high density of read-access. Therefore, we only concentrate on read access and design a high-density scheduler for read-access and it must be also suitable for write-access. Figure 4.9 shows an example of two unscheduled and scheduled read memory accesses when occurring row miss at different banks. For the unscheduled read, We choose (CL=2, BL=4) as an example, and then the unscheduled accesses takes 20 cycles to read eight burst data, whereas the scheduled accesses only requires 14 cycles and eight burst data can be sequential read. From the access latency discussion in section 4.1.2, the access command without auto precharge can be classified into two types, one is long command (PRE + ACT + CAS) and the other is short command (CAS), painted in Figure 4.5. Moreover, we consider the latency after access scheduling under BL=1, 2, 4 situations illustrated in Figure 4.10 - 4.12 and summaries the induced latency under each situation in Table 4.2. Obviously, we can find

that the worst latency is always located in row-miss situation. To reduce the access latency, the command request ordering and data arrangement should follow the orientation of minimizing the row-miss occurrence. The characteristics of READ and WRITE access are summarized as Table 4.2.

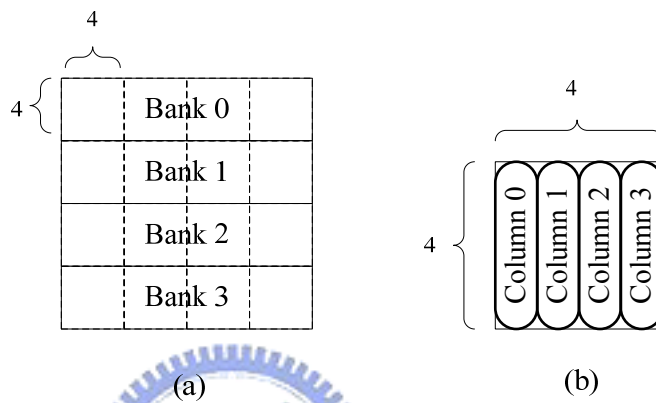
**Table 4.2 Characteristics of READ/WRITE access**

Operation	Using frequency	Module for decoder	Influence factors
READ	High	Motion compensation, Direct mode coding module	Bitstream, memory scheduling, data arrangement in memory
WRITE	Low and median (only MB level)	De-blocking filter, Direct mode coding module	Bitstream, capability of residual decoder (De-block filter only for H.264)

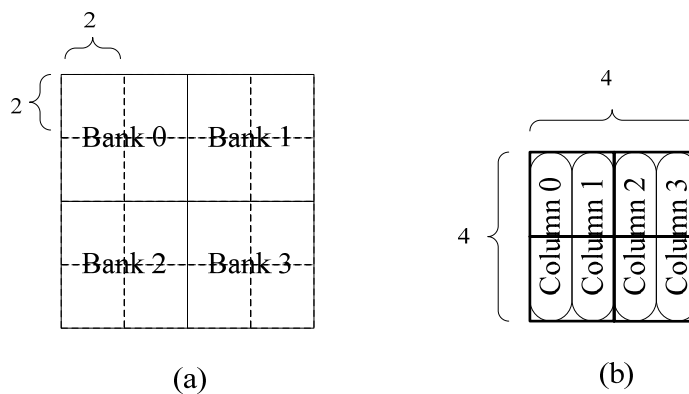
## 4.2.2 Memory Arrangement

From the above discussion, the data arrangement in SDRAM should tend to the minimization of row miss at the same bank because row miss status has to pay the longest latency. Based on this concept, row-major arrangement is adopted in our design. There are two kinds of data types, two arrangements are provided for co-located motion vector and pixels. Because YUV format is 4:2:0, luma and chroma component is 16 x 16 and 8 x 8 size, respectively. Therefore, the memory size of one 16 x 16 block require 384 bytes((16+8)\*16). Figure 4.9 (a) illustrates that the luma MB partition is dispersed to four banks, and Figure 4.9 (b) shows that memory access is a column major approach for each 4 x 4 block. The data arrangement of Chroma block is shown as Figure 4.10. Because chroma block is one quarter with luma block, memory size of one 4 x 4 luma block can contain four 2 x 2 chroma block. Similar to luma block, the chroma block is partitioned into four banks. As shown in Figure 4.10, the Cb block and Cr block are placed in four banks simultaneously. Besides, co-located motion vector where direct mode coding requires is stored into SDRAM. The motion vector arrangement is considered for reducing miss rate of access. The co-located motion vector of one MB is illustrated in Figure 4.11. One MB has to store four co-located motion vectors, that

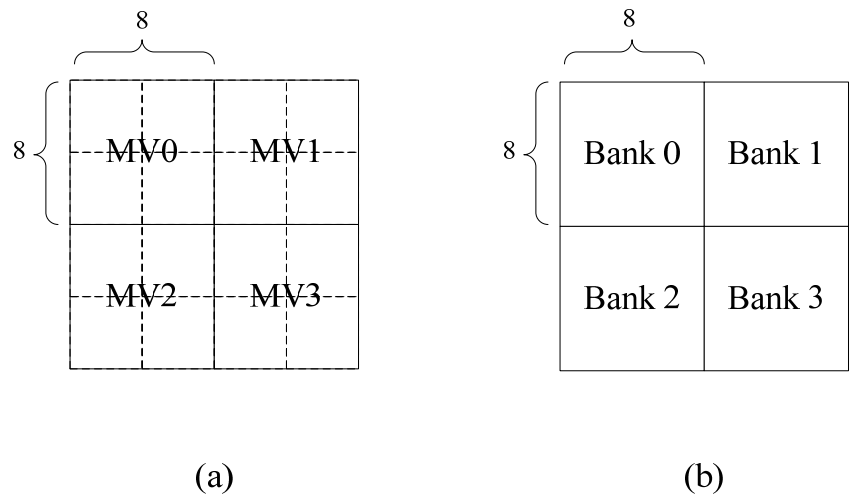
is, four motion vectors in one 8x8 MB can be predicted by one co-located motion vector. Each co-located motion vector contains 20 bits, one co-located motion vector occupies one column in SDRAM. The co-located motion vector can be stored in different banks according to corresponding positions.



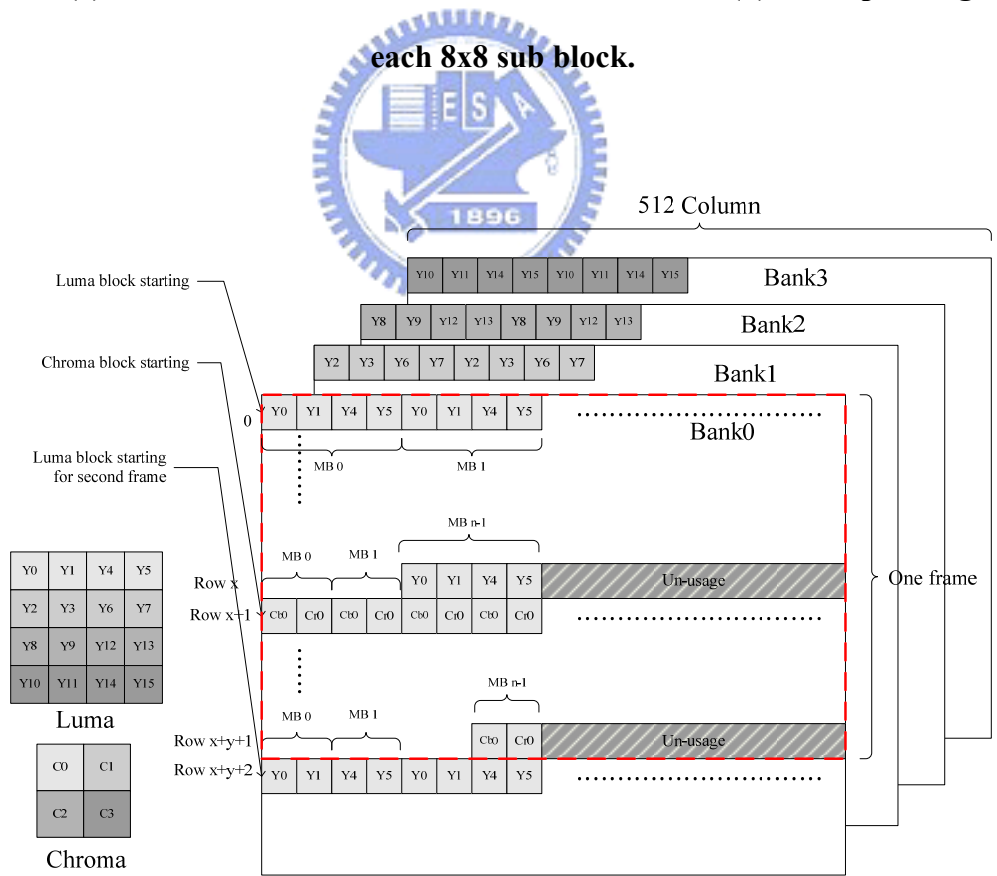
**Figure 4.9 Luma block (a) one 16 x 16 block arrangement (b) one 4 x 4 block-0 arrangement for MB-0**



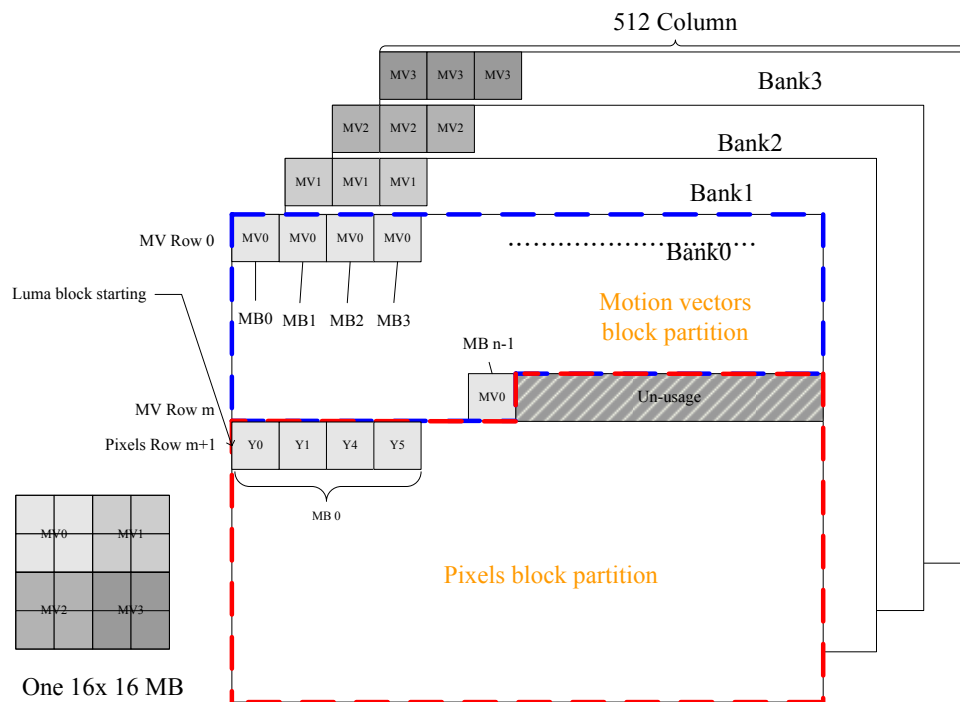
**Figure 4.10 Chroma block (a) one 8 x 8 block arrangement (b) one 4 x 4 block consist of block 0-3 arrangement**



**Figure 4.11 (a) Co-located motion vector allocation in frame (b) corresponding bank for each 8x8 sub block.**



**Figure 4.12 The pixels arrangement of one frame are stored of SDRAM. The arrangement of other banks is the same as bank0.**



**Figure 4.13 The motion vector arrangement of one frame are stored of SDRAM. The arrangement of other banks is the same as bank0.**

Figure 4.12 and Figure 4.13 show the pixels arrangement and the motion vector arrangement respectively in SDRAM for one frame, and the first MB is allocated in SDRAM. The first partition of memory is fixed to store co-located motion vector. Other partitions are used to store pixels. Each Frame can be partitioned into several MB-based row. For adopted SDRAM, each row contains 16 luma MBs or 32 chroma MBs. The Y0, Y1, Y4, Y5 and C0 of each Macroblock have to be stored in the corresponding location. Similarly, remaining pixels of macroblock are mapped in other corresponding banks. In Figure 4.12, the luma and chroma blocks are allocated sequentially into SDRAM. Due to decoding orders, luma block has to be stored in SDRAM firstly. When all luma blocks have been allocated in SDRAM, chroma block including Cb and Cr is stored in order. When frame size is small, each row (page) of SDRAM can store multiple MB-based rows of frame. Otherwise, for large frame size like SDTV or HDTV, each MB-based row may occupy several rows (pages) of SDRAM. The advantage of this arrangement is that address generator needs not be modified according to

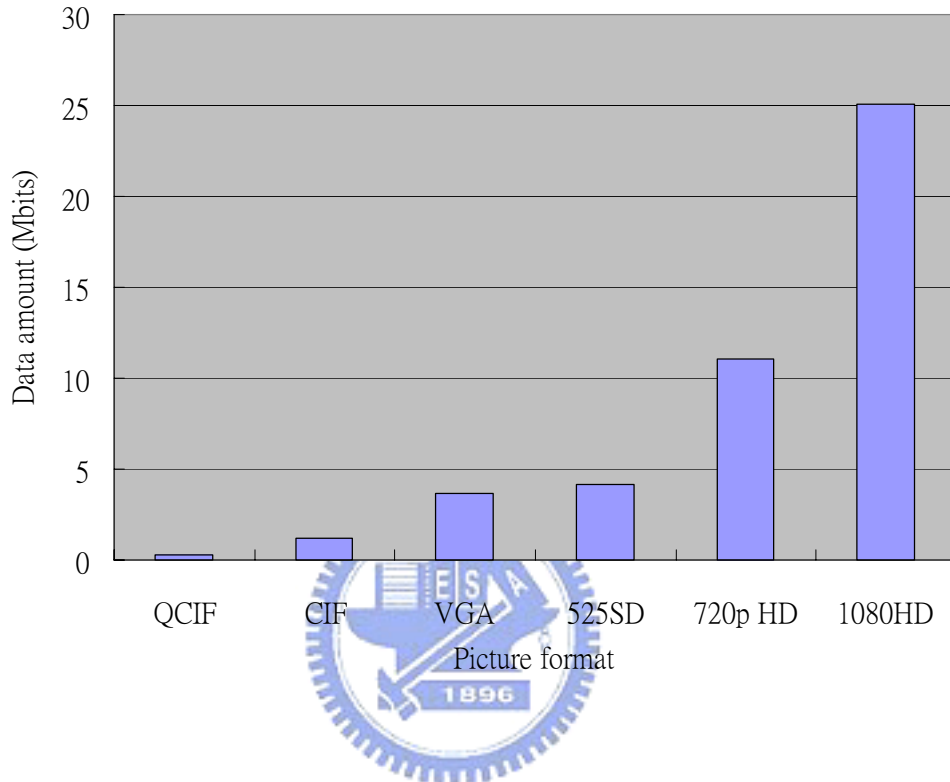


different frame sizes. For another reason, the probability of row-miss occurrence is considerably low. Obviously, it only occurs when data are located in row (page) boundary. As for physical analysis, it will be shown later. Row  $x$  indicates to last row of luma block, and chroma blocks are allocated in Row  $x+1$ . Row  $x+y+1$  means last row of one frame is stored in SDRAM. The definition of row  $x$  and  $y$  are variable depending on macroblock number of different picture formats. The memory mapping of multiple reference pictures will be introduced in the next section. The space of each frame which is stored into memory may be dynamically allocated by our memory controller. From above discussion, we can observe that some memory spaces are not used due to different macroblock numbers. The waste of space is not to avoid because memory arrangement must keep regular. Fortunately, the proportion of un-used space in SDRAM is less so that utilization of space is efficient.

### 4.2.3 Multiple Reference Prediction

H.264/AVC can support multiple reference pictures to current decoding macroblock which can be predicted from bi-directional and different frames. When multiple reference pictures are activated, decoder may store multiple frames into frame memory. In Wang's design [10], one SDRAM only stores one frame leading to poor utilization of SDRAM. For increasing the utilization of memory, we may store all frames into unique SDRAM and use shared data bus to access data. The data amount of different picture formats is shown in Figure 4.14. When decoding 1080HD picture, one frame produces 25.2 Mbits for pixels and 0.6 Mbits for motion vectors. Therefore, for 512MB SDRAM, maximum frame numbers which are stored into memory are 20. In other words, multiple reference pictures support 16 reference pictures for list0 and list1 respectively in 512MB SDRAM. In small format such as CIF format, 1.2 Mbits is produced and multiple reference pictures can support maximum 16 reference frames for bi-directional pictures. Table 4.3 lists required memory size and memory space utilization for different picture formats. We can observe that memory space utilization

can be improved in higher resolution format. In other words, limited memory space can store more pictures in SDRAM for multiple reference picture predictions. For instance, 512 MB SDRAM can contain maximum 20 1080HD pictures.



**Figure 4.14 Data amount of one decoding frame for different picture format.**

**Table 4.3 Required memory size for different picture format supporting multiple reference pictures set 16.**

Sequence format	Required memory size (Mbits)	Memory space utilization (%)
QCIF	4.87	77.34
CIF	19.46	92.84
VGA	58.98	98.68
525SD	66.35	98.98
720pHD	176.95	99.11
1080HD	401.08	99.61

## 4.2.4 Architecture of bandwidth-efficient Memory Controller

In this section, we show the block diagram of bandwidth-efficient memory controller architecture for H.264/AVC in Figure 4.15. The dotted area between video decoder and SDRAM is our proposed memory controller. The bandwidth-efficient memory controller consists of the data buffer, command queue, bank controller, command arbiter, address translator, and memory interface scheduler (MIS). Besides, a flexible address generator is considered in our design, which prior to different modules of the video decoder. In H.264/AVC, only motion compensation, de-blocking filter and direct coding units require access data from/to external memory through memory controller. The direct coding unit reads co-located motion vector to perform direct coding prediction. The motion compensation reads pixels from SDRAM to interpolating current pixels. The de-blocking filter writes complete pixels into SDRAM. Major units will be introduced sequentially as follows.

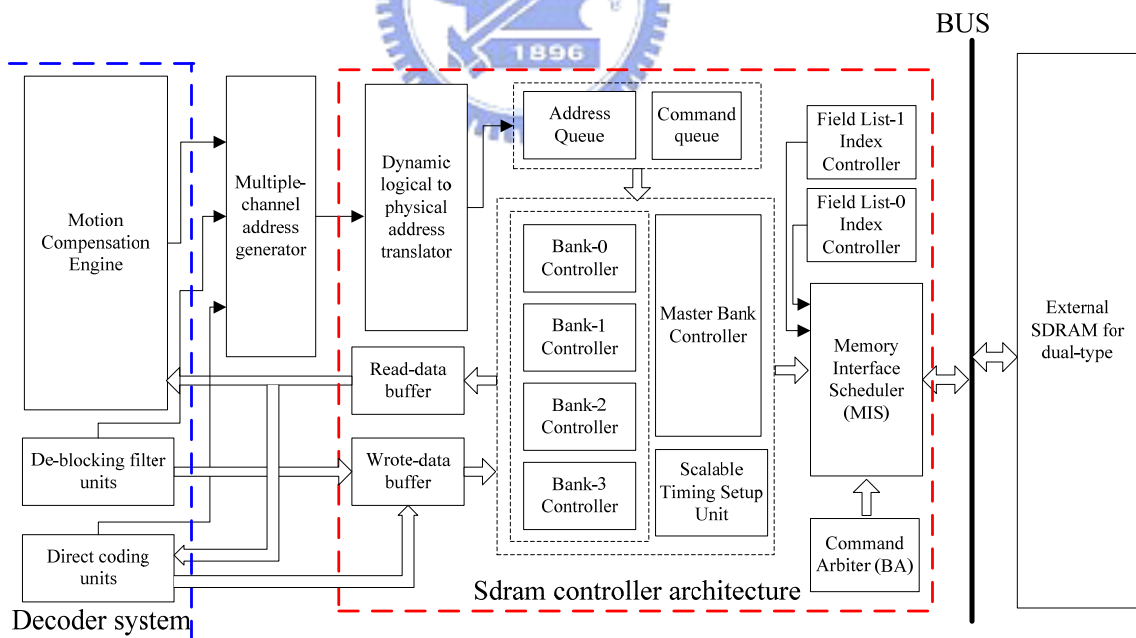
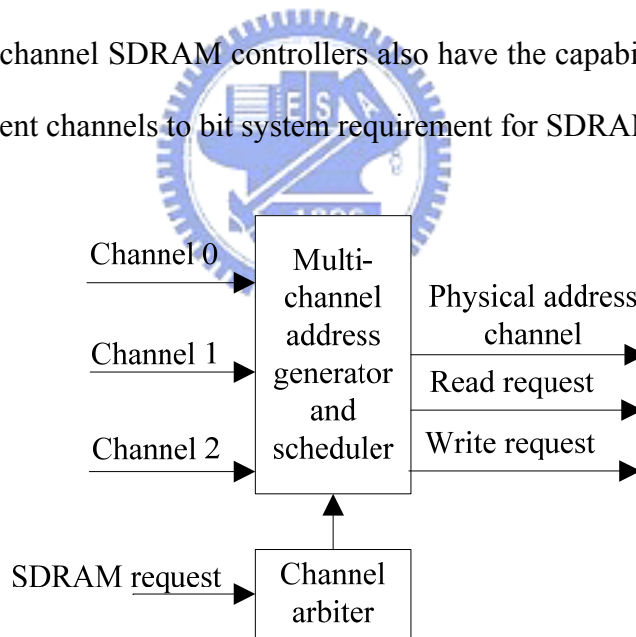


Figure 4.15 Architecture of bandwidth-efficient memory controller for H.264/AVC

➤ **Multiple-Channel Address Generator and scheduler**

For H.264/AVC decoder in which more processing units (PUs) need to access SDRAM, there are three main PUs to require memory read/write accesses. Generally, single channel memory controller design is employed in most applications, the pressure of area and cost leads to designing a single, shared off-chip SDRAM. The connection approach to sharing a SDRAM has to carefully decide because it is highly related to SDRAM efficiency. Traditional memory controllers are often connected by shared-buses. Although area and cost may be economic, the shared-bus makes the SDRAM hard to provide sufficient SDRAM performance for the increasingly complicated applications. Another issue is how to perform different SDRAM requirements for latency and bandwidth of PUs in the decoder system. In addition to offering better performance for H.264/AVC decoder compared to single-channel SDRAM controller, multiple-channel SDRAM controllers also have the capability to schedule memory accesses from different channels to bit system requirement for SDRAM performance.



Channel 0: Direct coding channel (read/write)
Channel 1: Interpolation channel (read)
Channel 2: De-blocking channel (write)

**Figure 4.16 Multi-channel address generator**

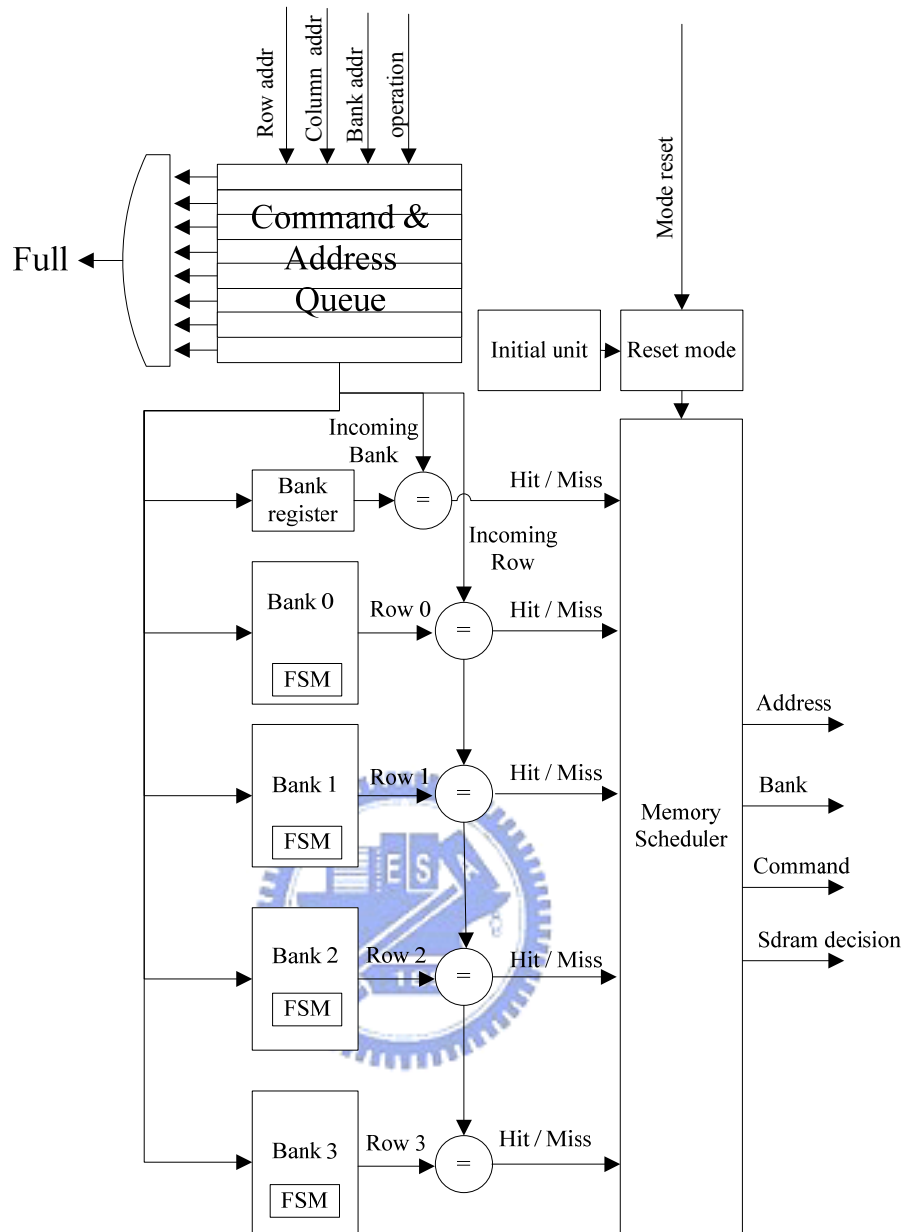
Since the memory controller can be applied to different modules in H.264/AVC decoder, we proposed multiple channels address generator and scheduler (MCAGS) to connect different PUs individually. The MCAGS can be used for several modules required. In H.264/AVC decoder, the MCAGS enables 3 channels for motion compensation, de-blocking filter, and direct mode coding module, which is shown in Figure 4.16. The MCAGS must to be provided to produce logical address prior to the memory controller. Due to different data types in motion compensation and direct mode coding, the MCAGS generates two kinds of logical addresses including pixels and motion vector addresses. The individual address is calculated according to the output ordering of the module. The output of address generator is sent into dynamic logical to physical address translator in memory controller after scheduling.

➤ **Dynamic Logical to Physical Address Translator**

For dynamic logical to physical address translator, the goal is that logical address is transferred to physical address (Row, Bank, Column) in SDRAM. The motion vector and the frame pixels are placed in different allocations, and the motion vector always allocated in the first partition. According to physical addresses, memory controller may read/write data in corresponding location.

➤ **Command and address Queue**

Due to long latency of SDRAM accesses, the module which issues a request may waste many cycles to wait data access. Therefore, a design avoids that decoder takes many cycles for waiting. Considering this reason, we design *command queue* to store incoming command including READ and WRITE commands from the decoder. The command queue can contain 7 READ or WRITE commands and sequential issue command into memory controller depending on incoming priorities. The command queue is a first-in-first-out structure according incoming priorities. The advantage of command queues is that the module needs one cycle to issue commands. Then, the module can do other processes but don't waste additional cycles to wait data.

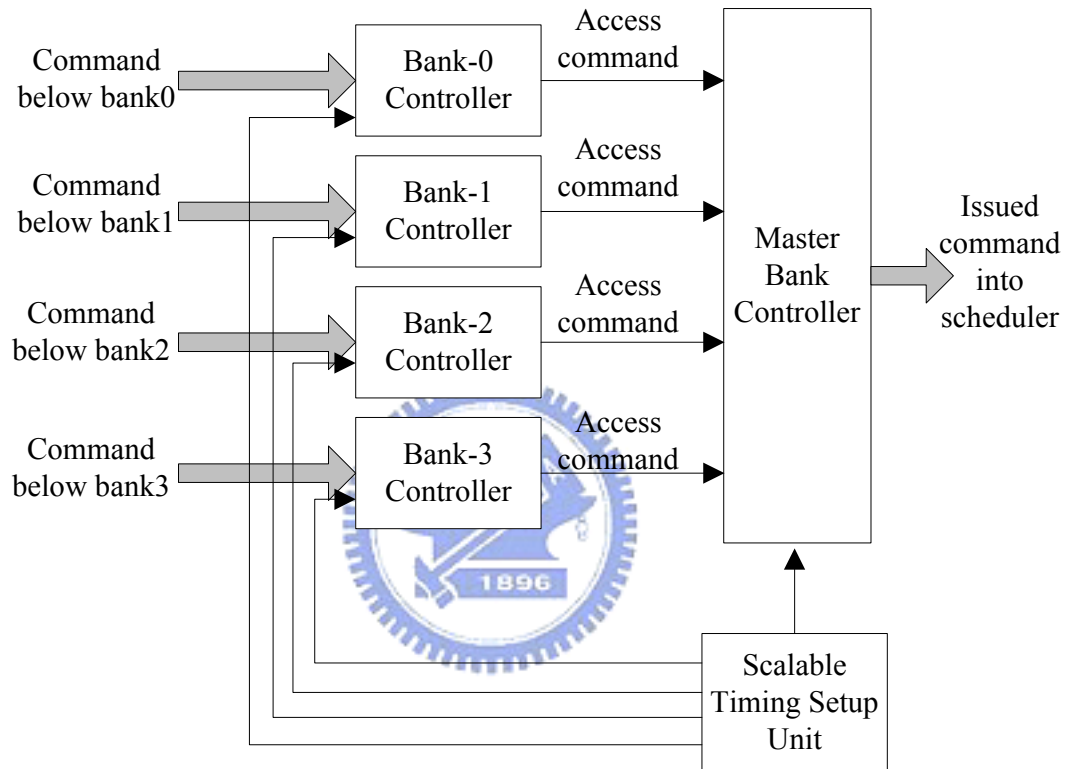


**Figure 4.17 Command and address queue and access status detection**

Besides, the address queue is used to hold incoming addresses including (Bank, Row, Column), while a command queue is used to hold incoming commands. The decoder sends request and address to memory access controller when the status of read address queue is not full. The “full” signals reflect the status of this queue. The proposed address queue must also compare the incoming and the previous address command to check row-hit and bank-hit situations.

➤ **Bank Controller**

To fully utilize the SDRAM bandwidth and apply memory scheduling, it is necessary that the memory interface can process accesses addressed to different banks in a parallel way. This work is performed by the bank controllers and the master bank controller together. The bank controller is illustrated as Figure 4.18.



**Figure 4.18** The structure of bank controllers, master bank controller and timing unit.

Each internal bank of the SDRAM is allocated an individual bank controller to process accesses that are addressed to the bank. The master bank controller assigns the incoming address commands to suitable bank controller according to the access status. Scalable timing unit records all kinds of command latency such as burst length, tRP (precharge period), tRCD (ACTIVE to READ or WRITE), and so on. The parameter of scalable timing unit is defined by user in the initial setup. After accepting an access from the input port, the bank controllers generate sequential access commands according to the burst length and latency defined in a

scalable timing unit. These access commands are collected by master bank controller, which can issue the proper command to SDRAM. Read data buffer is used to hold sequential received read data for motion compensation. Write data buffer is used to hold the length of burst data. The arbiter allocates write / read data and command flow to / from external SDRAM memories according to the access operation.

Unlike traditional SDRAM access controller design containing various “WAIT” states, Lee’s [30] proposed a configurable shared-state FSM Design. This design merges all numerous “WAIT” state into single NOP stage. After applying NOP\_count and NOP\_code status registers, the FSM becomes flexible to parameterize the command latency without redesign FSM. We design our access FSM based on this concept. The interface connection between memory scheduler and bank controller is depicted in Figure 4.17.

Each bank requires individual access FSM to control command process, and to wait until the previous access command returns to IDLE state. As for bank-miss (at the same row or not) situations, memory interface scheduler collects the access commands for the corresponding bank controllers and then sends to arbiter at the suitable time. Besides the access FSM, each bank controller needs a row address register to record the activated row. By comparing incoming commands with row address registers for each bank controller, the bank-miss with row-hit or bank-miss with row-miss status can be detected.

#### ➤ **Memory Interface Access Scheduler**

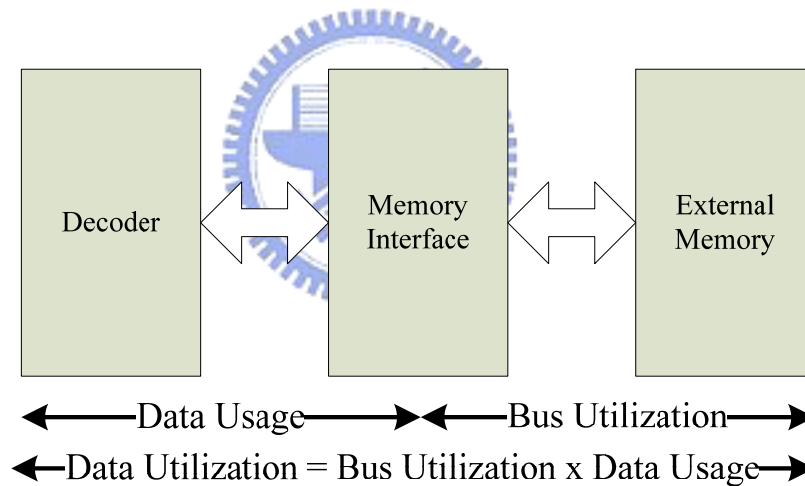
The memory interface access scheduler allocates and overlaps successive commands according to access status produced by status detection as Figure 4.17 shows. The memory interface access scheduler can perform scheduling with READ and WRITE operation. In brief, double access FSMs for individual bank controller can handle access conflict at the same bank, while master bank controller is responsible for access overlapping between different banks. After scheduling SDRAM access commands, the bus utilization can be raised efficiently; meanwhile the throughput of the entire video decoder can be improved. The



arbiter allocates write / read data and command flow to / from external SDRAM memories according to the access operation. Due to multiple reference picture supported, field list index controller is required to address frame start point.

### 4.3 Simulation Results

Considering system level analysis on decoder, memory controller, and external memory depicted in Figure 4.19, because decoder and memory controller are both in operation and data transmission only during the period of reading reference data, we only have to analyze the data transfer in this period.



**Figure 4.19 System level analysis relation**

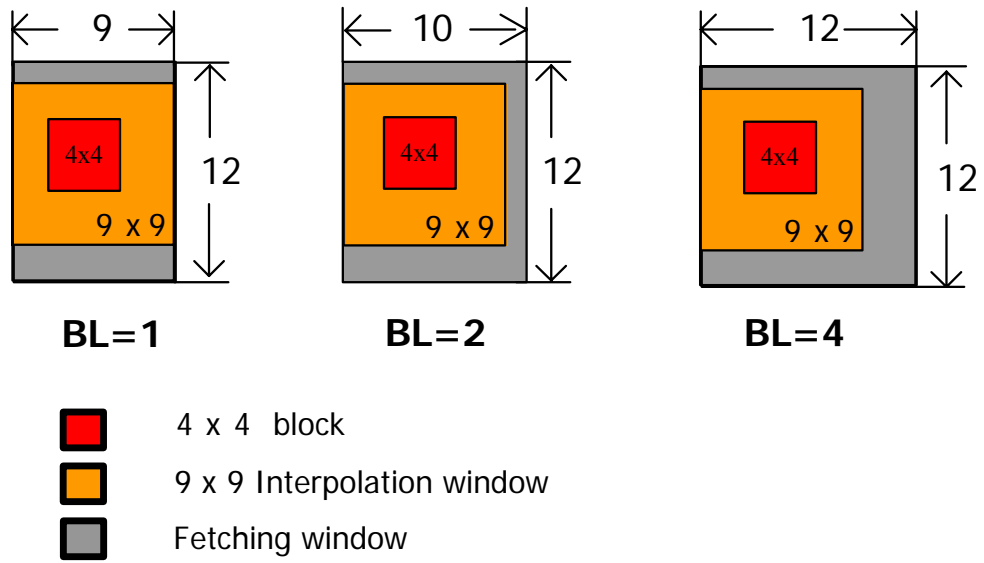
Before going into detail of the following analysis, we define the following equations to measure the performance of data transfer on the bus.

$$\text{Bus Utilization} = \frac{\frac{\# \text{ of bus cycles required by Memory interface}}{4x4 \text{ sub block}} \times \frac{\# \text{ of } 4x4 \text{ sub block}}{\text{frame}} \times \frac{\# \text{ of frame}}{\text{sec}}}{\frac{\# \text{ of bus cycles available}}{4x4 \text{ sub block}} \times \frac{\# \text{ of } 4x4 \text{ sub block}}{\text{frame}} \times \frac{\# \text{ of frame}}{\text{sec}}} \quad (4.1)$$

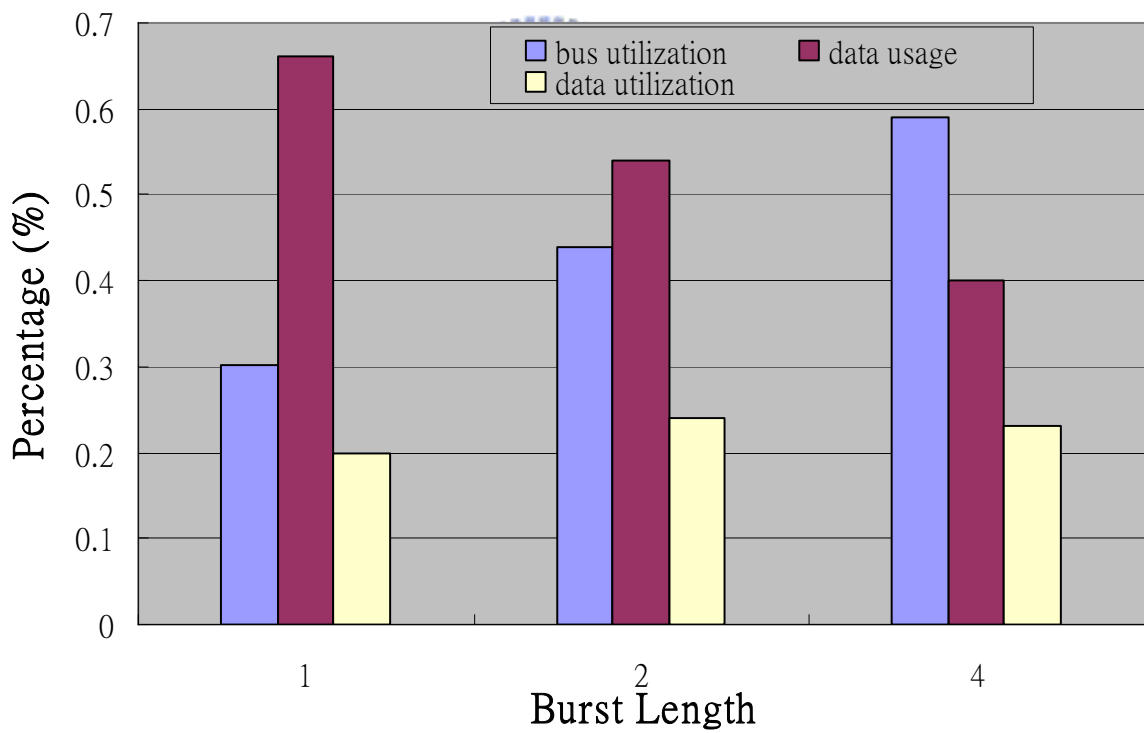
$$\text{Data Usage} = \frac{\frac{\# \text{ of data required by decoder}}{4 \times 4 \text{ sub block}} \times \frac{\# \text{ of } 4 \times 4 \text{ sub block}}{\text{frame}} \times \frac{\# \text{ of frame}}{\text{sec}}}{\frac{\# \text{ of data available from Memory interface}}{4 \times 4 \text{ sub block}} \times \frac{\# \text{ of } 4 \times 4 \text{ sub block}}{\text{frame}} \times \frac{\# \text{ of frame}}{\text{sec}}} \quad (4.2)$$

$$\text{Data Utilization} = \text{Bus Utilization} \times \text{Data Usage} \quad (4.3)$$

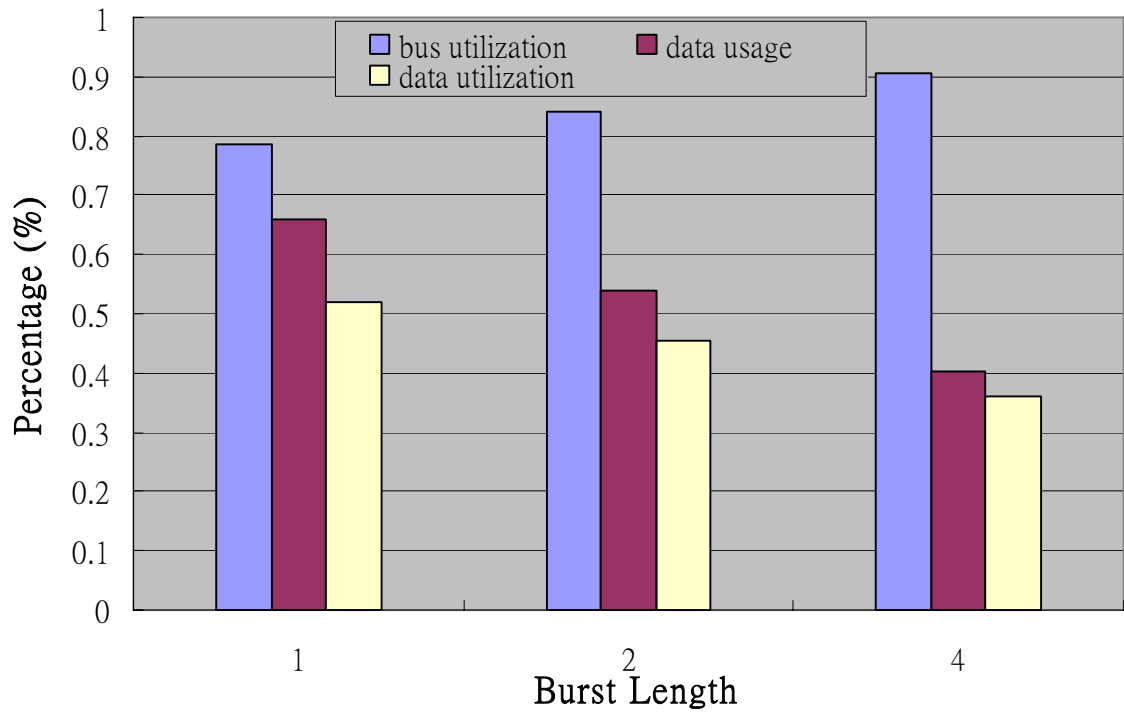
Based on the assumption of the data bus is only provided for unique frame and mv memory, higher bus utilization induces better throughput for our video decoder. The data usage is correlated to the burst length and required window size between decoder and memory controller. Hence, data usage can be treated as the proportion of required data for decoder over the available data from SDRAM controller. In other words, the data usage is related to burst length in memory setup. To explain data usage clearly, considering 9 x 9 interpolation window of a 4 x 4 block in H.264 fractional motion compensation, an example of the fetching window for four different burst lengths is illustrated as Figure 4.20. Fetching window is the total pixels that are required to be read from SDRAM controller. Since the data bus width is limited as 4-pixel (32 bits), the height of fetching window must be 12-pixel that is a multiple of 4-pixel when burst length is 4. Similarly, the width of other fetching window must be the multiple number of the burst length. Accordingly, among these burst length modes, the data usage is the poorest when the selected burst length is 4. From equation (4.3), data utilization is the multiplication of bus utilization and data usage. Therefore, the data utilization can be considered as the required data proportion in decoder over the allowable data transmission of the external bus. Higher data utilization means that we can get better throughput and less latency for the entire video decoder performance.



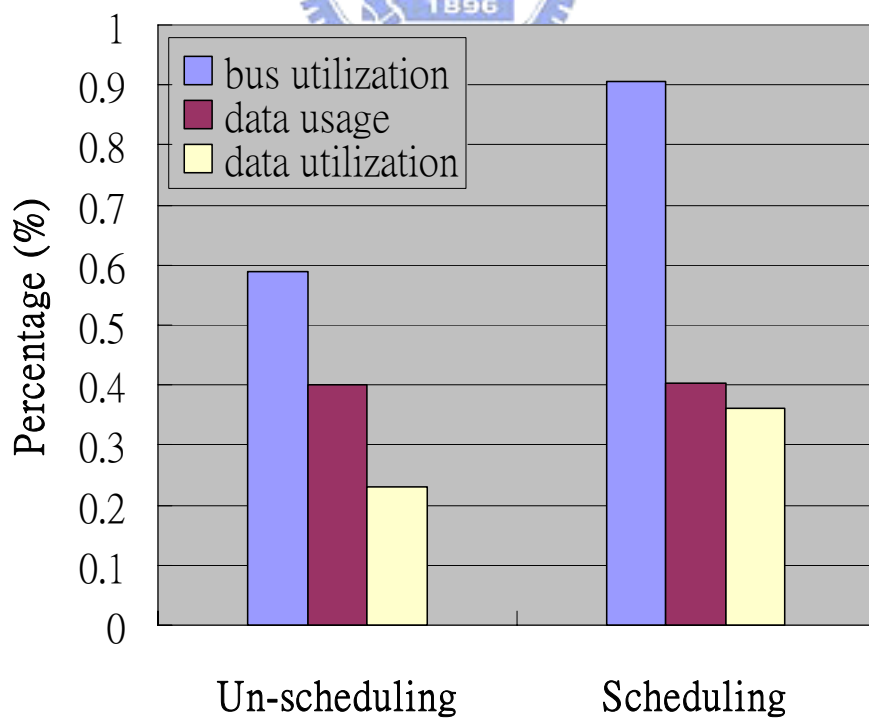
**Figure 4.20 Fetching windows of 4x4 block between different burst length**



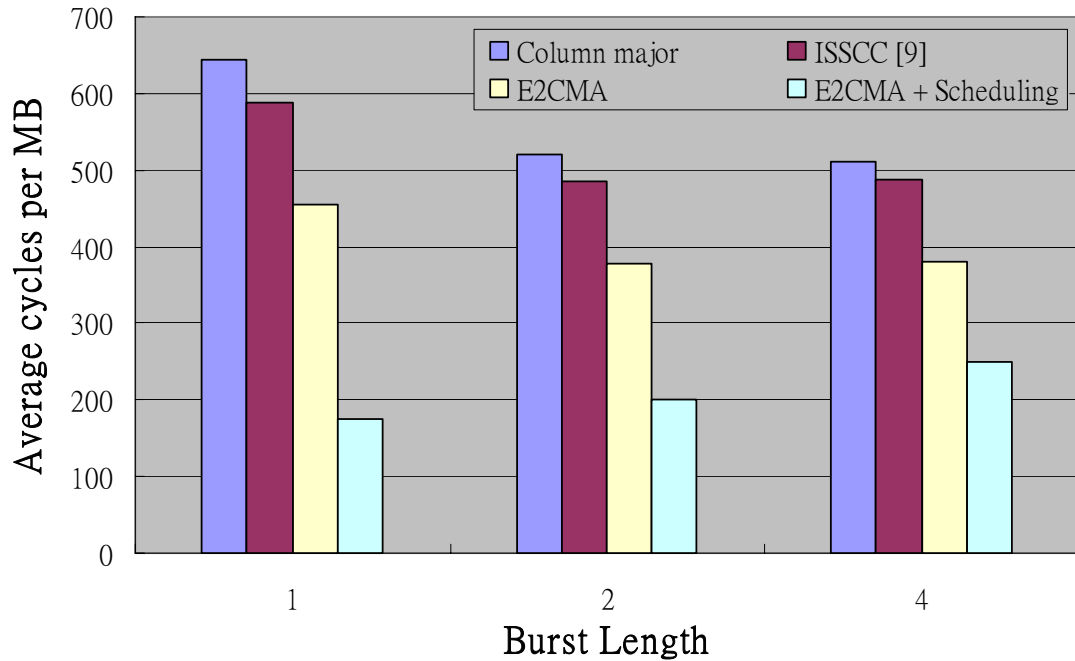
**Figure 4.21 Unscheduled Bus utilization, Data usage and Data utilization for different burst length in memory**



**Figure 4.22 Scheduled Bus utilization, Data usage and Data utilization for different burst length in memory.**



**Figure 4.23 The data utilization between un-scheduling and scheduling**



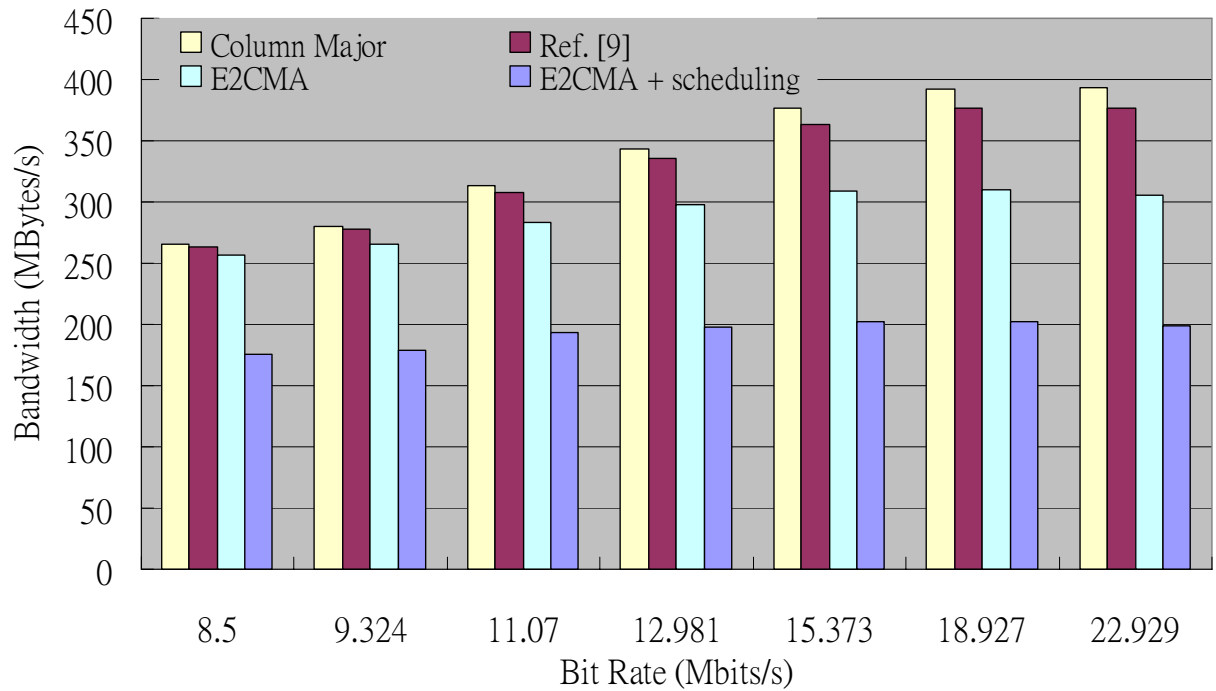
**Figure 4.24 Average access cycles per MB between different burst length for access**

**under BUS.**

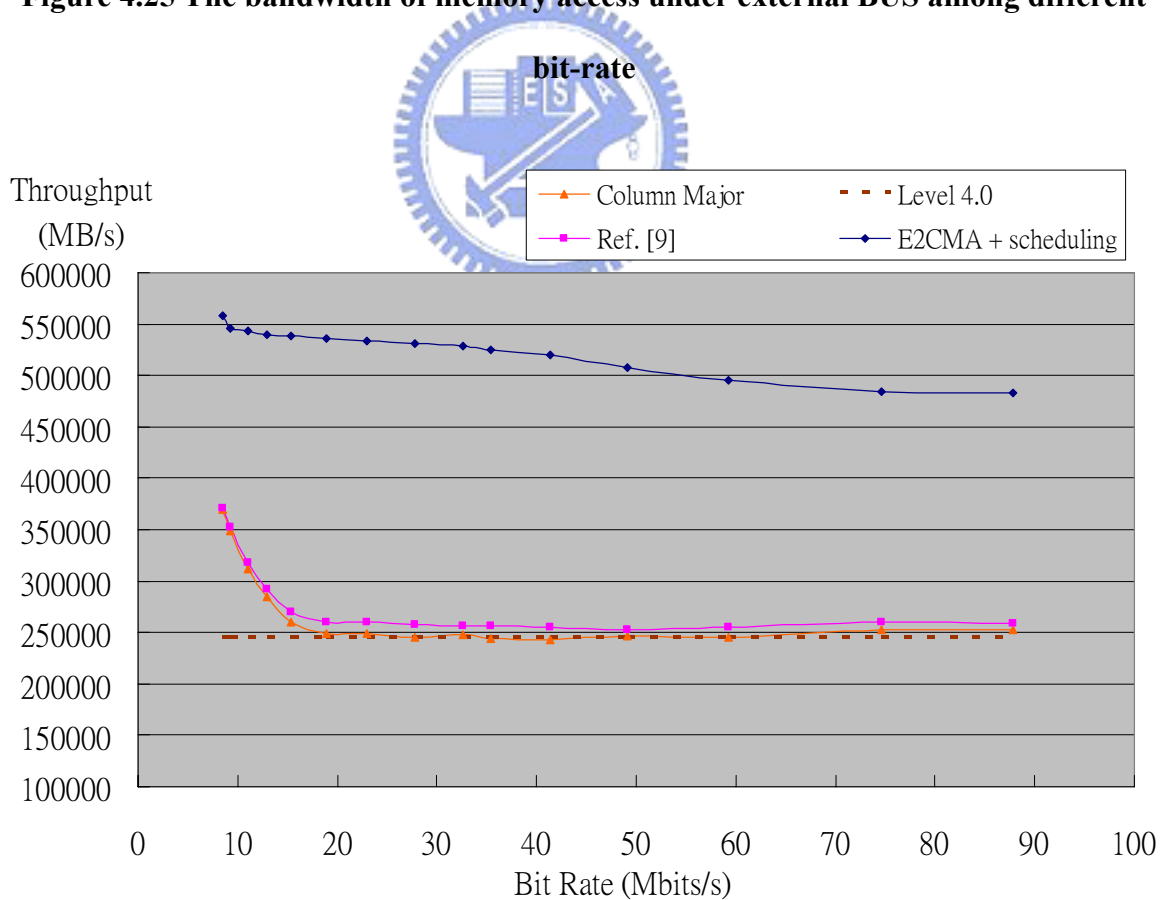
Figure 4.21 and 4.22 shows the unscheduled and scheduled system level analysis of the criteria (4.1) ~ (4.3). Obviously, the longer burst length provides higher bus utilization instinctively because the short access cycles are required for the more amount of fetching data. After scheduling, since longer read burst cycles can provides long overlapping period for the successive access commands, for instance, burst length = 4 has the highest bus utilization. Although burst length = 4 reflects the highest bus utilization, the lowest data usage leads that the data utilization become the lowest among these burst modes. The data usage is influenced extremely due to different amount of fetching windows among different burst length modes. Considering better data utilization for decoder, Burst length = 1 mode is the better choices on the high-throughput video decoding system. The Figure 4.23 shows the data utilization between un-scheduling and scheduling. Obviously, the bus utilization can be improved about 90% using memory scheduling. Therefore, the data utilization can be improved efficiently.

For H.264/AVC HDTV decoder, the average execution cycles per P\_MB and B\_MB

within 1080HD sequence at bit rate is 614Kbit/s environment for comparing different data-reuse approach is depicted in Figure 4.24. After inducing data reuse technique, E2CMA method, mentioned in Chapter 3, the execution cycles can reach 100 ~ 150 cycles approximately. After memory scheduling, the execution cycles with E2CMA approach can be reduced about 150 ~ 200 cycles again. Comparing ref [9], the execution cycles per P\_MB and B\_MB can tremendously reduce up to 55 %. Based on our decoding system, the raise of bus utilization and reduction of access latency reduce the required execution cycles per P\_MB and B\_MB. Accordingly, it can improve throughput of the entire video decoder because the computation time of motion compensation dominates the video decoder especially in H.264/AVC decoder. The bandwidth of memory access among different bit rate is depicted as Figure 4.25. The size of test sequence is 1080HD format, and the burst length within SDRAM is defined as 4. The bandwidth which is proposed by our proposed data reuse approach is better than other approach, especially at high bit-rate. Furthermore, E2CMA with memory scheduling technique is applied so that bandwidth can be further improved. Therefore, the bandwidth of memory access can be efficiently improved by our proposed data reuse approach. Besides, the throughput of entire video decoder working at 100MHz is shown as Figure 4.26. For supporting high resolution such as 1080HD, the system specification with level 4.0 has to be supported by video decoder. The throughput of decoder which applies E2CMA and memory scheduling is double than the one apply previous data-reuse approach. The decoder which applies Column major or Ref. [9] may be not arrive specification at level 4.0 in H.264/AVC standard, especially in the high bit-rate environment. That is, sequences with 1080HD format can be not decoded in a real-time system. For supporting higher resolution sequence such as 1080HD, the E2CMA and memory scheduling technique is suitable for HDTV decoder in real-time system.



**Figure 4.25 The bandwidth of memory access under external BUS among different**



**Figure 4.26 The throughput of motion compensation for different data-reuse approach when operating frequency is 100Mhz.**

## 4.4 *Summary*

In the applications requiring high performance SDRAM subsystem, any bandwidth loss may result in a system failure. For H.264/AVC decoder with main profile, the effect is the critical issue. Hence, the memory controller must be carefully designed to prevent any possible bandwidth loss. For above reason, we proposed a bandwidth-efficient memory controller that build-in device on a video decoder, and can be supported in different modules of H.264/AVC decoder. The proposed memory controller can deals with dual data type: motion vector and pixels. Allowing users to configure access mode for each SDRAM bank also gives more flexibility. We not only use the memory interface scheduler to do scheduling but also adopt the efficiently data arrangement to reduce the miss rate, and to increase utilization of memory space. From a system level analysis, we can observe that the bus utilization and access latency can be improved to 90%. The bandwidth of memory access between decoder and external memory can be improved as 50% approximately. The throughput of decoder can conform to system specification at level 4.0, especially working at high bit-rate.



# Chapter 5

## Chip Implementation

---

### 5.1 Chip Specification

**Table 5.1 H.264/AVC main profile decoder specification for motion compensation**

H.264/AVC Main Profile @ Level 4.0	
Support Slice Type	I, P, and B
Variable Block Size	16x16, 16x8, 8x16, 8x8, 8x4, 4x8, 4x4
MVp Generation	Direct mode coding, Median and Directional prediction
Direct Mode Coding	Temporal and spatial mode
Frame Prediction	Single and bi-directional reference frame
Interpolating Search Range	[-128, +127.75]
Interpolation accuracy	Quarter for luma, 1/8 for chroma
Picture AFF	Frame coding
Weighted Prediction	Explicit and implicit mode
Multiple Reference Frame	16 frames
Entropy Coding	CAVLC
Decoding Capability	1920 x 1088 HDTV, 30fps

Table 5.1 lists the specification of our bandwidth-efficient motion compensation architecture for H.264 HDTV decoder. After synthesis on Cadence RTL compiler using UMC 0.13 um COMS technology, total gate count is 557730 (including embedded SRAM) and the gate count of each component is listed for video decoder in table 5.2. The Die size of H.264

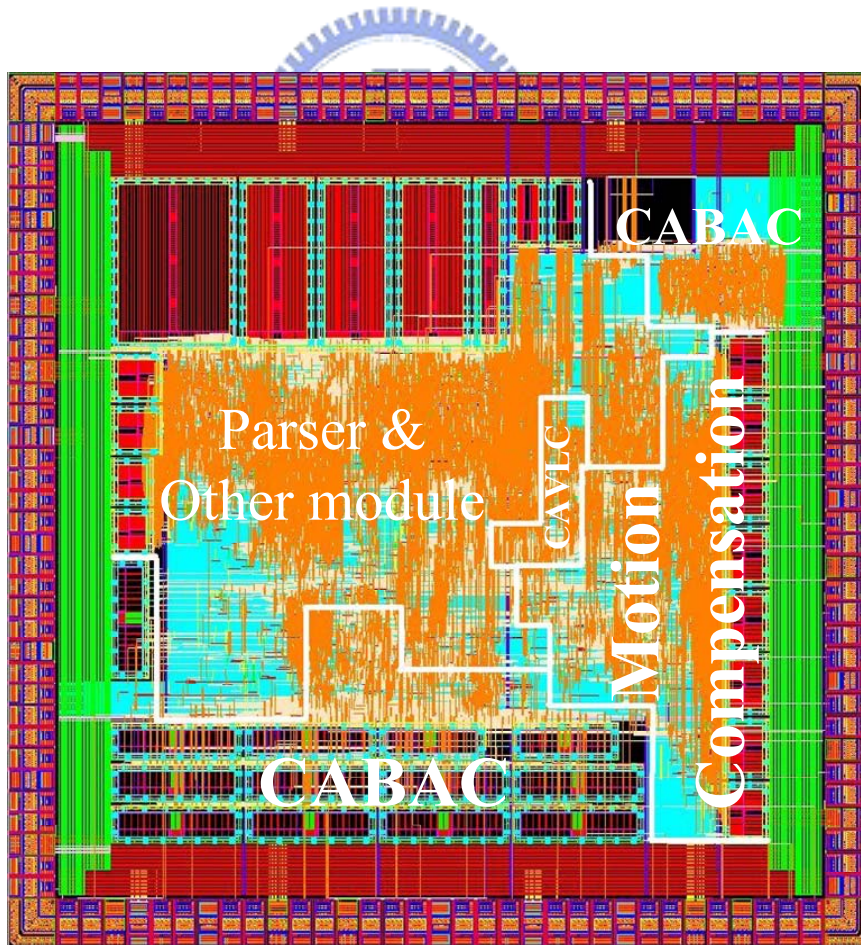
decoder is 3100 mm x 3100mm. Table 5.3 lists on/off chip memory used on each module in our design. The chip photo of H.264 decoder is illustrated as Figure 5.1. The average power consumption of system is 50mW approximately. Furthermore, about synthesis results of our proposed motion compensation and memory controller, the power consumption of motion compensation is 9.53mW and the power consumption of memory controller is 3.9mW at 100MHz, the gate count is 83515 and 8584 for motion compensation and memory controller respectively.

**Table 5.2 Synthesis results of H.264/AVC's main profile decoder including SRAM**

Component	Gate count (including SRAM)
De-blocking Filter	120957
Motion Compensation	83515
Syntax Parser & System Control	7625
Residual Adder & VL-FIFO	27562
Memory Access Controller	8584
CABAC	163573
Intra Prediction	38864
Content Memory	24263
4 x 4 Inverse Quantization	10958
Integer / Hardmard Transfrom	15346
IDCT	37483
CAVLC	7119
<b>Total</b>	<b>557730</b>

**Table 5.3 On/Off-Chip memory size for different module in H.264 main profile decoder**

	Module	Memory Size (Depth x Width)	Port	Number
On-Chip	Motion Compensation	120 x 24	Single-Port	x8
	De-blocking Filter	2048 x 32	Single-Port	x1
		1024 x 32	Single-Port	x2
		128 x 11	Single-Port	x1
	Intra Prediction	1024 x 32	Single-Port	x1
	Syntax Parser	128 x 16	Single-Port	x1
	Content Buffer	64 x 32	Single-Port	x2
		32 x 32	Single-Port	x2
	Residual unit	1024 x 5	Single-Port	x1
	CABAC	736 x 7	Dual-Port	x1
128x12		Single-Port	x2	
128x14		Single-Port	x1	
128x16		Single-Port	x10	
Total	28884 Bytes			
Off-Chip	External Memory	512K x 32 x 4 banks	x1	



**Figure 5.1 CHIP photo for H.264/AVC main profile decoder**

# **Chapter 6**

## **Conclusion**

---

In this thesis, we present a bandwidth-efficient motion compensation memory controller organization for H.264 HDTV decoder and support 1080HD 30fps@L4 high-quality format. The proposed motion compensation engine realizes all advanced features including MV generators with direct modes, combined luma/chroma interpolator, and weighted prediction of H.264/AVC main profile. Concerning the design of interpolator, 4-parallel separate 1-D architecture gives the most space on high throughput video decoder compared with other architectures proposed. An Extend-2D column major approach is presented, and the proposed data reuse technique for fraction motion compensation introduces content buffer, content-swap operation and register-file shifting attached on our interpolator design. This design improves 50%-60% bandwidth with B-slices under external data BUS. Additionally, a combined luma/chroma interpolator is proposed in order to save area, which achieves approximately 44% of cost reduction. Altogether, memory usage and bandwidth are optimized by our proposed design.

Besides, the decoder system bottleneck resulted from the performance limitation of the off-chip SDRAM subsystem leads system designers to put more efforts on SDRAM efficiency. In conventional SDRAM controller designs, though different requirements for SDRAM service of the heterogeneous system components are often considered, high bandwidth utilization can be achieved for special applications such as high definition TV. For this reason, the proposed memory controller can reduce bandwidth over external BUS using memory

scheduling and improve data access hit rate using data arrangement. For reducing bus utilization, the memory controller architecture is proposed and related approaches are employed as well. This design target of interpolator and frame memory access controller is to reduce external memory access and improve throughput of the entire video decoder. The SDRAM memory access controller appended to video decoder is presented to overcome the tremendous transfer of pixel data to/from external frame memories. To achieve efficient memory access scheduling, we discuss not only memory scheduling but also data arrangement within SDRAM. The proposed data arrangement in our scheduling scheme can minimize the miss ratio (at the same bank) that contributes the maximum latency among all scheduling cases. We create system level hardware-like C++ model and use data utilization to analyze the system performance. Compared to unscheduled situation, the experimental result shows that the access latency can be reduced by 50 % ~ 90 % and bandwidth utilization can be improved up to 90%. In the meanwhile, the throughput of the overall video decoder improves about 50 % ~ 60 % after combining extended RSO method and memory scheduling. Besides, the gate count of motion compensation and memory controller is 83515 and 8584 respectively in synthesis results. The average power consumption of motion compensation and memory controller is 9.45mW and 3.9mW approximately at 100MHz.

# *Bibliography*

- [1] “Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification,” Joint Video Team (JVT), Int. Telecommun. Union-Telecommun. (ITU-T) and Int. Standards Org./Int. Electrotech. Comm. (ISO/IEC), ITU-T Recommendation H.264 and ISO/IEC 14496-10 AVC, May 2003.
- [2] “Information technology-generic coding of moving pictures and associated audio information: Video,” ITU-T H.262, ISO/IEC 13818-2, 1994.
- [3] *Joint Video Team H.264/AVC Reference Software, Version JM 9.2.*  
<http://iphome.hhi.de/suehring/tml/download/> .
- [4] T. Wiegand, G. J. Sullivan, G. Bjntegaard, and A. Luthra, “Overview of the H.264/AVC video coding standard,” *IEEE Trans. Circuits Syst. Video Technol.*, vol 13, no 7, pp. 560-576, July, 2003.
- [5] Peng Yin; Tourapis, A.M.; Boyce, J.” Localized weighted prediction for video coding”, *IEEE Trans. ISCAS* Vol. 5, 23-26 May 2005 pp:4365 - 4368
- [6] Boyce, J.M.; “Weighted prediction in the H.264/MPEG AVC video coding standard”, *IEEE Trans. ISCAS*, Volume 3, 23-26 May 2004 pp. - 789-92
- [7] Tourapis, A.M.; Feng Wu; Shipeng Li, “ Direct mode coding for bipredictive slices in the H.264 standard”, *IEEE Trans. Circuits and Systems for Video Technology*, Volume 15, Issue 1, Jan. 2005 pp.119 - 126
- [8] Tourapis, A.M.; Feng Wu; Shipeng Li, “ Direct mode coding for bipredictive slices in the H.264 standard”, *IEEE Trans. Circuits and Systems for Video Technology*, vol 15, Jan. 2005, pp.119 - 126
- [9] Tsu-Ming Liu, Ting-An Lin, Sheng-Zen Wang, Wen-Ping Lee, Kang-Cheng Hou, Jiun-Yan Yang and Chen-Yi Lee, “A 125- $\mu$ W, Fully Scalable MPEG-2 and H.264/AVC Video Decoder for Mobile Applications”, *ISSCC Dig. of Tech. Papers*, Feb. 2006.pp. 402-403

- [10] S. Z. Wang, T. A. Lin T. M. Liu, C. Y. Lee “A New Motion Compensation Design for H.264/AVC Decoder” in *Proc. of Int. symposium on Circuits and Systems (ISCAS '05)*, 2005, pp. 4558-61
- [11] P. C. Tseng, Y. C. Chang, Y. W. Huang, H. C. Fang, C. T. Huang, and L. G. Chen, “Advances in hardware architectures for image and video coding - a survey,” in *Proc. IEEE*, vol. 93, no. 1, pp. 184-197, Jan. 2005.
- [12] T. W. Chen, Y. W. Huang, T. C. Chen, Y. H. Chen, C. Y. Tsai, and L. G. Chen, “Architecture design of H.264/AVC decoder with hybrid task pipelining for high definition videos,” in *Proc. IEEE Int.Symp. Circuits and Systems*, 2005, pp. 2931-2934.
- [13] Y. Hu, A. Simpson, K. McAdoo, and J. Cush, “A high definition H.264/AVC hardware video decoder core for multimedia SoC's,” in *Proc. IEEE Int. Symp. Consumer Electron.*, Sept., 2004, pp. 385-389.
- [14] T. A. Lin, S. Z. Wang, T. M. Liu, and C. Y. Lee, “An H.264/AVC decoder with 4x4 level pipeline,” in *Proc. IEEE Int. Symp. Circuits and Systems*, 2005, pp. 1806-1809.
- [15] T. A. Lin, T. M. Liu, and C. Y. Lee, “A low-power H.264/AVC decoder,” in *Proc. IEEE Int. Symp. VLSI-TSA*, Apr. 2005, pp. 278-281.
- [16] Azevedo, A.; Zatt, B.; Agostini, L.; Bampi, S.; ”Motion compensation sample processing for HDTV H.264/AVC decoder”, *Digital Object Identifier 10.1109/NORCHP, 2005*, Page(s):110 – 113
- [17] Haung-Chun Tseng; Cheng-Ru Chang; Youn-Long Lin; ” A hardware accelerator for H.264/AVC motion compensation”, *Digital Object Identifier 10.1109/SIPS, 2005*. Page(s):214 – 219
- [18] Chuan-Yung Tsai; Tung-Chien Chen; To-Wei Chen; Liang-Gee Chen; “Bandwidth optimized motion compensation hardware design for H.264/AVC HDTV decoder” *Digital Object Identifier 10.1109/MWSCAS, 2005*. Page(s):1199 – 1202
- [19] H. Y. Kang, K. A. Jeong, J. Y. Bae, Y. S. Lee, and S. H. Lee, “MPEG4 AVC/H.264 decoder with scalable bus architecture and dual memory controller”, in *Proc. IEEE Int.Symp. Circuits and Systems*, vol. 2, 2004, pp. II - 145-148.

- [20] V. Lappalainen, A. Hallapuro, and T. D. Hamalainen, "Complexity of optimized H.26L video decoder implementation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 717-725, July, 2003.
- [21] W. N. Lie, H. C. Yeh, Tom C. I. Lin, and C. F. Chen, "Hardware-efficient computing architecture for motion compensation interpolation in H.264 Video Coding," " in *Proc. IEEE Int. Symp. Circuits and Systems*, 2005, pp. 2136-2139.
- [22] T. C. Chen, Y. W. Huang, and L. G. Chen, "Fully utilized and reusable architecture for fractional motion estimation of H.264/AVC," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, vol. 5, 2004, pp. V - 9-12.
- [23] C. D. Chien, H. C. Chen, L. C. Huang, and J. I. Guo, "A Low-power motion compensation IP core design for MPEG-1/2/4 video decoding," in *Proc. IEEE Int. Symp. Circuits and Systems*, 2005, pp. 4542-4545.
- [24] W. F. He, Z. G. Mao, J. X. Wang, and D. F. Wang, "Design and implementation of motion compensation for MPEG-4 AS profile streaming video decoding", in *Proc. IEEE Int. Conf. ASIC*, vol. 2, 2003, pp. 942-945.
- [25] Tourapis, A.M.; Feng Wu; Shipeng Li, " Direct mode coding for bipredictive slices in the H.264 standard", *IEEE Trans. Circuits and Systems for Video Technology*, vol 15, Jan. 2005, pp.119 - 126
- [26] Micron Technology, Inc. product documents. [Online]. Available: <http://www.micron.com/products/>
- [27] Micron Technology, Inc. MT48LC2M32B2P-5 64Mb SDRAM (Jan. 2005). [Online]. Available: <http://www.micron.com/products/dram/sdram/partlist.aspx?density=64Mb>
- [28] P. R. Panda, N. Dutt, and A. Nicolau, "Memory Issues in Embedded Systems-on-Chip: Optimization and Exploration ". Boston, MA: Kluwer Academic Publishers, 1999.
- [29] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," in *Proc. IEEE Int. Symp. Computer Architecture*, Vancouver, BC, Canada, Jun. 2000, pp. 128-138.



- [30] S. Miura, and T. Watanabe, "A dynamic-SDRAM-mode-control scheme for low-power systems with a 32-bit RISC CPU," in *Proc. IEEE Int. Symp. Low Power Electron. and Design*, Aug. 2001, pp. 358-363.
- [31] K. B. Lee, T. C. Lin, and C. W. Jen, "An efficient quality-aware memory controller for multimedia platform SoC," *IEEE Trans. Circuits Syst. Video Techno.*, vol. 15, no. 5, pp. 620-633, May 2005.
- [32] H. Kim, and I. C. Park, "High-performance and low-power memory-interface architecture for video processing applications," *IEEE Trans. Circuits Syst. Video Techno.*, vol. 11, no. 11, pp. 1160-1170, Nov. 2001.
- [33] S. I. Park, Y. Yongseok, and I. C. Park, "High performance memory mode control for HDTV decoders," *IEEE Trans. Consumer Electron.*, vol. 49, no. 4, pp. 1348-1353, Nov. 2003.
- [34] J. H. Li, and N. Ling, "Architecture and bus-arbitration schemes for MPEG-2 video decoder," *IEEE Trans. Circuit Syst. Video Techno.*, vol. 9, no. 5, pp. 727-736, Aug. 1999.
- [35] J. Zhu, L. Hou, R. Wang, C. Huang, and J. Li, "High performance synchronous DRAMs controller in H.264 HDTV decoder," in *Proc. IEEE Int. Conf. Solid-State and Integrated Circuits Technol.*, vol. 3, 2004, pp.1621-1624.
- [36] J. Tajime, T. Takizawa, S. Nogaki, and H. Harasaki, "Memory compression method considering memory bandwidth for HDTV decoder LSIs," in *Proc. IEEE Int. Conf. Image Processing*, vol. 2, 1999, pp. 779-782.
- [37] T. Y. Lee, "A new frame-recompression algorithm and its hardware design for MPEG-2 video decoders" *IEEE Trans. Circuit Syst. Video Techno.*, vol. 13, no. 6, pp. 529-534, Jun. 2003.
- [38] E. De Greef, F. Catthoor, and H. De Man, "Memory organization for video algorithms on programmable signal processors," in *Proc. IEEE Computer Design: VLSI in Computers & Processors*, Oct. 1995, pp. 552-557.
- [39] L. Nachtergaele, F. Catthoor, B. Kapoor, S. Janssens, and D. Moolenaar, "Low-power data transfer and storage exploration for H.263 video decoder system," *IEEE J. Select. Areas Commun.*, vol. 16, no. 1, pp. 120-129, Jan. 1998.

- [40] E. Brockmeyer, L. Nachtergaele, F. V. M. Catthoor, J. Bormans, H. J. De Man, "Low power memory storage and transfer organization for the MPEG-4 full pel motion estimation on a multimedia processor," *IEEE Trans. Multimedia*, vol. 1, no. 2, pp. 202-216, June 1999.
- [41] K. Denolf, C. De Vleeschouwer, R. Turney, G. Lafruit, and J. Bormans, "Memory centric design of an MPEG-4 video encoder," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 5, pp. 609-619, May 2005.
- [42] Shih-Chang Hsia, "Efficient memory IP design for HDTV coding applications", *IEEE Trans. Circuits and Systems for Video Technology*, Volume 13, Issue 6, June 2003, pp. 465 – 471
- [43] Takizawa, T.; Tajime, J.; Harasaki, H. , "High performance and cost effective memory architecture for an HDTV decoder LSI", *IEEE Trans. ICASSP '99. Proceedings.*, Volume 4, 15-19 March 1999 pp.1981 – 1984
- [44] Sinnathamby, M.; Manjikian, N. "A versatile memory-interface architecture for enhancing performance of video applications", *IEEE-NEWCAS Conference*, 2005. 19-22 June 2005 pp.91 - 94
- [45] S. Wuytack, J. -P. Diguët, and F. V. M. Catthoor, "Formalized methodology for data reuse exploration for low-power hierarchical memory mappings," *IEEE Trans VLSI Syst.*, vol. 6, no. 4, pp. 529-537, Dec. 1998.

## 作者簡歷

姓名：侯康正

出生地：台灣省台北市

出生日期：1979.12.05

學歷：1985.9~1991.6 台北縣竹圍國民小學

1991.9~1994.6 台北縣立淡水國民中學

1994.9~1999.6 北台科學技術學院 電子工程科

1999.9~2001.6 國立高雄應用科技大學 電子工程系 學士

2004.9~2006.6 國立交通大學 電子研究所 系統組 碩士



## 得獎事績

1999 春 教育部單晶片微處理器設計 佳作

1999 秋 義隆盃單晶片微控制器設計組 特優

2006 春 九十四學年度大學校院積體電路(IC)設計競賽 第四名

## 發 表 論 文

Kang-Cheng Hou, Sheng-Zen Wang, Yi-Hong Huang, Tsu-Ming Liu, Chen-Yi Lee, “**A Bandwidth-Efficient Motion Compensation Architecture for H.264/AVC HDTV Decoder**”, in *Proceedings of the 17th VLSI/CAD Symposium*, August 2006.

Yi-Hong Huang, Ping-Chang Lin, Kang-Cheng Hou, Yueh-Chi Hung, Tsu-Ming Liu, Chen-Yi Lee,” **A High-Throughput SRAM-Based Context Adaptive Binary Arithmetic Decoder (CABAD) for H.264/AVC**”, in *Proceedings of the 17th VLSI/CAD Symposium*, August 2006.

Tsu-Ming Liu, Ting-An Lin, Sheng-Zen Wang, Wen-Ping Lee, Kang-Cheng Hou, Jiun-Yan Yang and Chen-Yi Lee, “**A 125- $\mu$ W, Fully Scalable MPEG-2 and H.264/AVC Video Decoder for Mobile Applications**”, is accepted by *IEEE Journal of Solid-State Circuits*.

Tsu-Ming Liu, Ting-An Lin, Sheng-Zen Wang, Wen-Ping Lee, Kang-Cheng Hou, Jiun-Yan Yang and Chen-Yi Lee, “**A 125- $\mu$ W, Fully Scalable MPEG-2 and H.264/AVC Video Decoder for Mobile Applications**”, *ISSCC Dig. of Tech. Papers*, pp. 402-403, San Francisco, USA, Feb. 2006.

Tsu-Ming Liu, Ting-An Lin, Sheng-Zen Wang, Wen-Ping Lee, Kang-Cheng Hou, Jiun-Yan Yang and Chen-Yi Lee, “**An 865- $\mu$ W H.264/AVC Video Decoder for Mobile Applications**”, *IEEE Asian Solid-State Circuit Conference*, 2005. pp. 301-304, HsinChu, Taiwan, Nov. 2005.