國立交通大學

電子工程學系　電子研究所碩士班

碩士論文

適用於雙重視訊標準的可調式動作補償記憶體架構

A Flexible Motion Compensation Memory Organization for Dual-standard Video Decoder

學生 ： 王勝仁

指導教授 ： 李鎮宜 教授

中華民國九十四年六月

E
S
A
1896

適用於雙重視訊標準的可調式動作補償記憶體架構

A Flexible Motion Compensation Memory Organization for Dual-standard Video Decoder

研究生：王勝仁　　Student：Sheng-Zen Wang

指導教授：李鎮宜　　Advisor：Chen-Yi Lee

國立交通大學
電子工程學系 電子研究所 碩士班
碩士論文

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

Electronics Engineering

June 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年六月

E
S
A
1896

適用於雙重視訊標準的可調式動作補償記憶體架構

學生：王勝仁　　　　　　　　　　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

近年來，對於已被先進的數位電視廣播系統採用的 MPEG-2 和 H.264/AVC 視訊標準，其需求是很必要的，動作補償的計算量通常占了整個視訊解碼系統的大多數，這是由於它需要對儲存畫面的記憶體有相當大量的資料傳輸。特別在目前最先進的 H.264/AVC 視訊標準支援了更高的移動解析度，因而使得所需的記憶體頻寬大量增加。我們提出的擴充性 2x2 光柵式掃描(extended 2x2 raster scanning order)除了可有效地減少所需的記憶體頻寬之外，同時維持和殘餘係數解碼器相同的解碼順序。和傳統的架構相較之下，針對 MPEG-2/ H.264 提出可重新架構的小數點內插器，可省下 20 % 的邏輯閘數量。此外，針對視訊解碼器而提出的 SDRAM 畫面記憶體存取控制器可將頻寬使用率提升至 85 ~90 % 且減少資料存取的延遲達 50 ~90%。在這同時，整個視訊解碼器的資料量處理能力也會提升。我們的視訊解碼器合併了 H.264 Baseline Profile @ 3.2 Level 和 MPEG-2 Simple Profile @ Main Level，而高畫質視訊的即時解碼能力對 H.264 而言可達到 720 HD @ 56 MHz，對 MPEG-2 而言可達到 1080 HD @ 79.4 MHz，而總邏輯閘數量為 491 K，其中包含 23.5 KB 的 on-chip SRAM。

A Flexible Motion Compensation Memory Organization for Dual-standard Video Decoder

Student : Sheng-Zen Wang Advisor : Dr. Chen-Yi Lee

Department of Electronics Engineering
Institute of Electronics
National Chiao Tung University

ABSTRACT

In recent year, MPEG-2 and H.264/AVC video decoding system, which has been adopted by the advanced digital video broadcasting terrestrial/handheld (DVB-T/H) system, is in great demand. The computation time of motion compensation always dominates the entire video decoding system due to the tremendous data transfer with frame memories. Especially in the state-of-the-art video standard, H.264/ AVC, the requisite memory bandwidth is greatly increased because the higher motion resolution requires larger interpolation. The proposed data-reuse technique, extended 2 x 2 raster scanning order, can efficiently reduce the required memory bandwidth when maintaining the same decoding order as that of residual decoding. The proposed reconfigurable interpolator providing fractional interpolation for MPEG-2/H.264 can reduce 20 % gate count compared to traditional design. In addition, the proposed SDRAM frame memory access controller for video decoder increases the bandwidth utilization up to 85~95%, and reduces access latency by 50 ~90% compared to the un-scheduling memory access. In the meanwhile, the throughput of our video decoder is also improved. Our video decoder combined H.264 Baseline Profile @ 3.2 Level and MPEG-2 Simple Profile @ Main Level and the decoding capability of high definition television can reach 720HD at 56 MHz for H.264, 1080HD at 79.4 MHz for MPEG-2 real-time video decoding with total 491K gate count included 23.5 KB on-chip SRAM.

E
S
A
1896

Acknowledgements

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

Moreover, I would like to thank NSC for their financial support, my senior **Si2 multimedia group** mates Tsu-Ming Liu for his great help during my research, and Ting-An Lin for his powerful integration on the entire video decoder platform. Especially, I want to thank sincerely VLSI SP Lab mates Yen-Chung Chang (Nelson) for his selfless suggestion and guidance about memory controller and organization. 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. I hope them happy and happy forever.

Contents

CHAPTER 1 INTRODUCTION 1

1.1 MOTIVATION 1

1.2 THESIS ORGANIZATION 3

CHAPTER 2 ALGORITHM DESCRIPTION AND ANALYSIS 4

2.1 PROFILING 6

2.2 INTER PREDICTION ALGORITHM FOR H.264/AVC STANDARD 7

2.3 COMPARISON AMONG DIFFERENT VIDEO STANDARDS 10

2.4 SUMMARY 13

CHAPTER 3 MOTION COMPENSATION DESIGN FOR MPEG-2/H.264 VIDEO DECODER 14

3.1 MOTION COMPENSATION ENGINE FOR H.264/AVC DECODER 15

3.1.1 Motion Vector Generator 16

3.1.2 Data Reuse Technique for Interpolator 20

3.1.3 Analysis for Data Reuse Technique 23

3.1.4 Luma Interpolator Design 26

3.1.5 Chroma Interpolator Design 30

3.2 COMBINED MOTION COMPENSATION ENGINE FOR MPEG-2/H.264 DUAL VIDEO DECODER 32

3.2.1 Reconfigurable dual-standard interpolator design *33*

3.2.2 Cost Analysis *39*

3.3 Summary 40

CHAPTER 4 FRAME MEMORY ORGANIZATION 41

4.1 SDRAM characteristic 46

4.1.1 Basics *46*

4.1.2 Access Latency *48*

4.2 Dual Channel Frame Memory Access Controller 51

4.2.1 Memory Access Scheduling *52*

4.2.2 Data Arrangement *59*

4.2.3 Dual-channel Frame Memory access controller Design *62*

4.2.4 Experiment Result *66*

4.3 Merging Structured Frame Memory Organization 73

4.3.1 Analysis *74*

4.4 Summary 76

CHAPTER 5 CHIP IMPLEMENTATION 77

5.1 Chip Specification 77

5.2 Comparison with Related Works 80

CHAPTER 6 CONCLUSION AND FUTURE WORK 82

6.1 Conclusion 82

6.2 Future Work 83

BIBLIOGRAPHY 85

List of Figures

Fig. 1.1 Typical communication model based on DVB system 2

Fig. 2.1 General structure of H.264 encoder 4

Fig. 2.2 General structure of H.264 decoder 4

Fig. 2.3 General structure of MPEG-2 encoder 5

Fig. 2.4 General structure of MPEG-2 decoder 5

Fig. 2.1 H.264/AVC video decoder software profile on ARM processor (JM 8.2).......... 6

Fig. 2.2 Macroblock partitions and sub-macroblock partitions 7

Fig. 2.3 (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 8

Fig. 2.4 (a) directional prediction for 8 x 16 block size, (b) directional prediction for 16 x 8 block size, (c) median prediction 9

Fig. 2.5 Luma integer/fractional motion vector proportion for H.264/AVC 12

Fig. 2.6 Chroma integer/fractional motion vector proportion for H.264/AVC 12

Fig 3.3 Neighboring motion vectors needed when decoding all motion vectors in current macroblock 17

Fig 3.4 (a) motion vector generator architecture for QCIF-format, (b) MV buffer unit 18

Fig. 3.5 Two-level memory hierarchical structure for MVP 18

Fig 3.6 (a) block size_position index, (b) directional prediction table (16x8, 8x16), (c) median prediction table (16x16, 8x8), (d) median prediction table (4x4)............ 19

Fig 3.7 (a) 4x4 block window and the corresponding 9x9 interpolation window, (b) overlapped region for neighboring interpolation window 20

Fig 3.8 (a) 2x2 raster scanning order, (b) row-major 2x2 raster scanning order, (c) column-major 2x2 raster scanning order 20

Fig 3.9 (a) 2x2 raster scanning order, (b) 4x4 raster scanning order, (c) extended 2x2 raster scanning order 21

Fig 3.10 Synchronization buffer scheme for two different scanning order in inter prediction (a) 2x2 raster scanning order, (b) 4x4 raster scanning order......... 21

Fig. 3.11 Content-swap operation (interpolator with attached content buffer)....... 22

Fig. 3.12 An example of macroblock partition (1, 3) indicates (mv_x, mv_y). 22

Fig. 3.13 Block proportion under different bit-rate environments 25

Fig. 3.14 Required memory bandwidth for different methods 25

Fig 3.15 (a) adder-chain based [10], (b) adder-tree based [11] 1-D linear interpolator design 26

Fig. 3.16 Separate 1-D interpolator design (no parallel).. 26

Fig. 3.17 4-parallel separate 1-D luma interpolator with content buffer 29

Fig 3.18 Interpolation window for each 2 x 2 chroma block....................................... 30

Fig. 3.19 (a) chroma interpolator, (b) vertical/horizontal filter 30

Fig. 3.20 2-parallel chroma interpolator 31

Fig. 3.21 Motion compensation engine for H.264/MPEG-2 decoder.............................. 32

Fig. 3.22 MPEG-2 Motion compensation engine part 33

Fig. 3.23 Shared local registers and bilinear filters for MPEG-2.............................. 34

Fig. 3.24 Data flow of (a) vertical bilinear filter, (b) horizontal bilinear filter, (c)

both vertical and horizontal bilinear filter 35
Fig. 3.25 (a) decoding scanning order based on 8 x 8 block for each macroblock, (b) 4-pixel parallel output order for each 8 x 8 block 36
Fig. 3.26 (a) luma FIR design in Chen's [9], (b) bilinear filter 36
Fig. 3.26 Combined luma/chroma interpolator design for H.264 37
Fig. 3.27 (a) path of luma FIR interpolator, (b) path of chroma 1/8 bilinear 37
Fig. 3.27 Shared local register and reconfigurable interpolator for H.264 chroma interpolation 38
Fig. 4.1 Frame recompression method 45
Fig. 4.2 Simplified architecture of a 4-bank SDRAM 46
Fig 4.3 Simplified bank state diagram 46
Fig. 4.4 Burst read operation with CL=3 and BL=4 47
Fig. 4.5 Burst write operation with CL=3 and BL=4. 48
Fig. 4.6 Access latency for CL=2 (a) read access latency, (b) write access latency 49
Fig. 4.7 READ command with auto precharge In the precharge period (tRP), SDRAM cannot issue another command in the same bank (bank 0). 50
Fig. 4.8 READ/WRITE operation in (a) I frame, (b) P frame, (c) B frame. 51
Fig. 4.9 Two row-miss unscheduled and scheduled read memory accesses (CL=2, BL=4) 53
Fig. 4.10 Scheduled consecutive read access for (BL=1, CL=2) when previous command is (a) long command (PRE+ACT+CAS), (b) short command (CAS).......54
Fig. 4.11 Scheduled consecutive read access for (BL=2, CL=2) when previous command is (a) long command (PRE+ACT+CAS), (b) short command (CAS).......55
Fig. 4.12 Scheduled consecutive read access for (BL=4, CL=2) when previous command is (a) long command (PRE+ACT+CAS), (b) short command (CAS).......56
Fig. 4.13 Scheduled consecutive read access for full-page mode 58

Fig. 4.14 (a) row-major arrangement, (b) the first luma macroblock location in SDRAM 59
Fig. 4.15 The location of SDRAM for chroma Cb and Cr 60
Fig. 4.16 QCIF video format arrangement in SDRAM 60
Fig. 4.17 Different cases of the interpolation window location when window size is 9 x 9 pixels (A, B) truncation by banks at the same row, (C) truncation by banks at different row, (D) truncation by row and banks, or window size is 4 x 4 pixels (E) best case, no truncation, (F) truncation by banks at the same row 61
Fig 4.18 Dual-channel SDRAM memory access controller design 64
Fig. 4.19 Incoming address command queue 65
Fig. 4.20 Memory bank controllers and schedulers 65
Fig. 4.21 Motion compensation processing flow for (a) H.264/AVC, (b) MPEG-2 66
Fig. 4.22 System level analysis 67
Fig. 4.23 Fetching window of H.264 4 x 4 block for different burst length (unit: pixel): (a) BL=1 or full-page, (b) BL=2, (c) BL=4 68
Fig. 4.24 Unscheduled bus utilization, data usage, and data utilization for different burst mode 69
Fig. 4.25 Scheduled bus utilization, data usage, and data utilization for different burst mode 69
Fig. 4.26 Average access latency per P MB for unscheduled/scheduled memory access 71
Fig. 4.27 Average execution cycles of P MB for foreman-QCIF @ 32 kbps 71
Fig. 4.28 Average execution cycles of P MB for foreman-QCIF @ 128kbps 72
Fig. 4.29 Merging structured frame memory organization 73
Fig. 5.1 The chip photo of MPEG-2/H.264 dual-video decoder 79

FIG. 5.1 MEMORY HIERARCHY FOR THE ENTIRE VIDEO DECODER 81

List of Tables

Table 2.1 Comparison of fractional motion compensation among different standards 11
Table 3.1 Neighboring MV checking table for content-swap operation 23
Table 3.2 Static analyses for different method in H.264/AVC. Assumption: each motion vectors contains fractional part. 24
Table 3.3 Comparison of execution cycles for different architectures 27
Table 3.4 # of adders for each filter design 39
Table 3.5 Comparison of requisite modules based on 4-parallel separate 1-D architecture Traditional design: separated H.264 luma, H.264 chroma and MPEG-2 bilinear interpolator 40
Table 4.1 Related works of SDRAM memory controller 43
Table 4.2 CAS latency 48
Table 4.3 Characteristics of read/write-access 52
Table 4.4 Latency for scheduled consecutive read access when CL=2 57
Table 4.5 The proportion of access status (foreman-QCIF @ 64kbps in H.264 decoder) 61
Table 4.6 Related works of merging structured frame memory 74
Table 4.7 Perfect match analysis JM8.2: QCIF, 30 fps, search range [-16, +15.75], QP = 30 75
Table 5.1 Video decoder specification in our design 77
Table 5.2 Synthesis result of the MPEG-2/H.264 dual-video decoder (including on-chip SRAM) 78

TABLE 5.3 ON-CHIP/OFF-CHIP MEMORY IN OUR DESIGN 78

TABLE 5.4 CHIP SPECIFICATION OF MPEG-2/H.264 DUAL VIDEO DECODER 79

TABLE 5.5 H.264 DECODER COMPARISON WITH RELATED WORK............ 80

Chapter 1
Introduction

1.1 Motivation

With the development of technology, the progress of video coding standard reflects the adaptation of video coding to different applications and networks. The early video technology, MPEG-1, mainly targets on CD-ROM based video storage. Subsequently, MPEG-2, which can be backward compatible with MPEG-1, serves a wider range of application including video-on-demand (VOD), DVD and high definition TV. Network communication includes switched networks such as PSTN (H.263, MPEG-4) or ISDN (H.261), and packet networks like ATM (MPEG-2, MPEG-4), the Internet (H.263, MPEG-4), .mobile networks (H.263, MPEG-4). Up to now, the newest video coding standard published jointly as Part 10 of MPEG-4 and ITU-T Recommendation H.264 provides dramatic video compression performance. The new H.264/AVC standard provides a technical solution for a broad range of applications, including broadcast over cable, satellite, cable modem, DSL or terrestrial, interactive or serial storage like DVD, conversational services over ISDN, Ethernet, LAN, wireless, or mobile network, multimedia messaging services over DSL, ISDN, etc. In order to provide different application and backward compatible with previous standard, video technology faces the challenge of combining different standard into the single system and providing powerful compatibility.

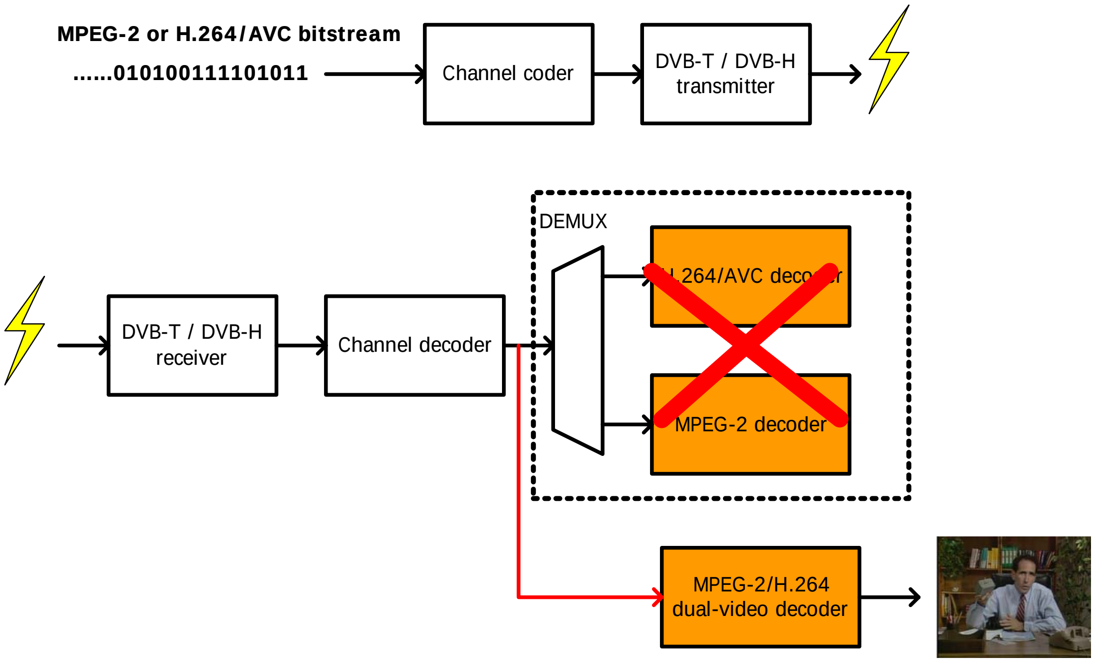


Fig. 1.1 Typical communication model based on DVB system

In recent years, Digital TV is widely adopted by the next-generation video broadcasting transmission (DVB) technology. Digital video broadcasting terrestrial, DVB-T, permits the transmission of MPEG-2 video bitstream. Moreover, in Nov. 2004, Digital video broadcasting handheld, DVB-H, has mandated support of Main Profile for H.264/AVC SDTV receivers, with an option for the use of High Profile. The support of High Profile is mandated for H.264/AVC HDTV decoder. Especially, DVB-H features backward compatibility with DVB-T but transmit different video format. However, H.264/AVC does not directly backward compatible with previous standards. Therefore, the challenge of merging H.264/AVC and MPEG-2 to single video decoding system is in great demand. Fig 1.1 shows the typical communication model based on DVB system. In addition, high definition TV requires enormous data transmission particular in frame memory, a memory access controller that efficiently communicates with frame memory is the most significant over the entire video decoding system. Within the video decoding system, motion compensation always dominates the total amount of data transmission especially when SDRAM or DDR-SDRAM is adopted as external frame memories. Video decoder should also provide efficient memory access controller to manage data transfer and access conflict.

1.2 Thesis Organization

The thesis is organized as follows. The algorithm description and analysis is discussed in Chapter 2. In Chapter 3, the motion compensation engine for H.264/AVC video decoder is described firstly. Then, the motion compensation engine for MPEG-2/H.264 dual-video decoder is illustrated. We also propose the data reuse technique to reduce the required bandwidth particularly in H.264/AVC fractional motion compensation. Chapter 4 presents frame memory organization including frame memory access controller for external SDRAM and merging structured frame organization that is one of the frame compression method. Chip implementation is given in Chapter 5. Finally, conclusion and future work is shown in Chapter 6.

Chapter 2

Algorithm Description and Analysis

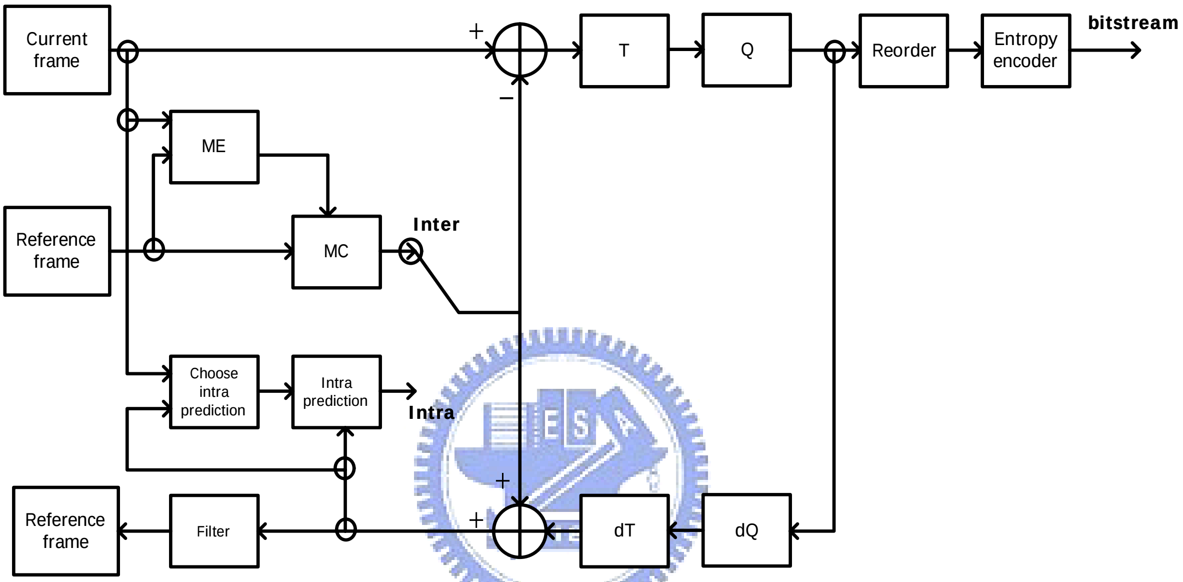


Fig. 2.1 General structure of H.264 encoder

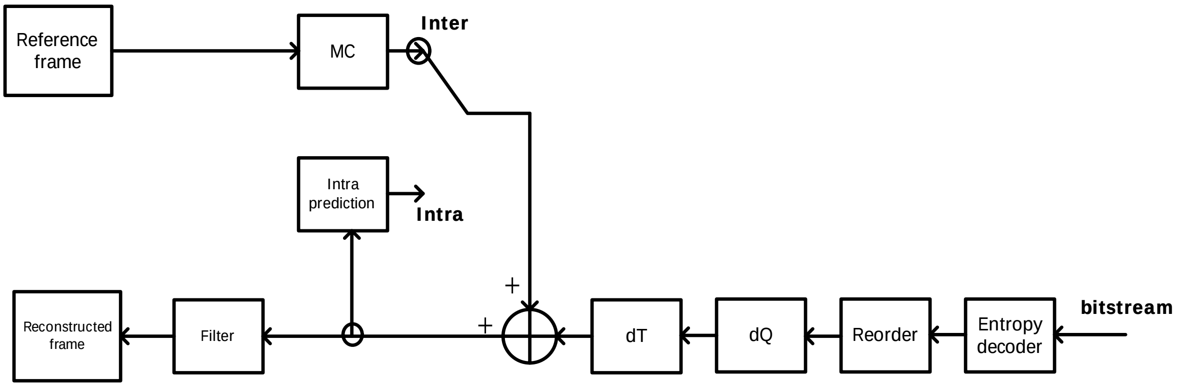


Fig. 2.2 General structure of H.264 decoder

Fig. 2.1 and Fig. 2.2 show the general structure of H.264/AVC video encoder and decoder respectively. The H.264/AVC design covers a Video Coding Layer (VCL) and

Network Abstraction Layer (NAL). We only concentrate on VCL that efficient represents the video content. The spirit of H.264/AVC follows the so-called *block-based hybrid video coding*. It consists of hybrid of temporal and spatial prediction, in conjunction with transform coding. The main additional blocks compared with prior standards are intra prediction and in-loop de-blocking filter. Fig. 2.3 and Fig. 2.4 illustrate general structure of MPEG-2 encoder and decoder respectively. Compared to H.264/AVC, the decoding flow becomes simplified without intra prediction and in-loop de-blocking filter except that only DCT/IDCT is more complicated than integer transform for H.264/AVC codec.

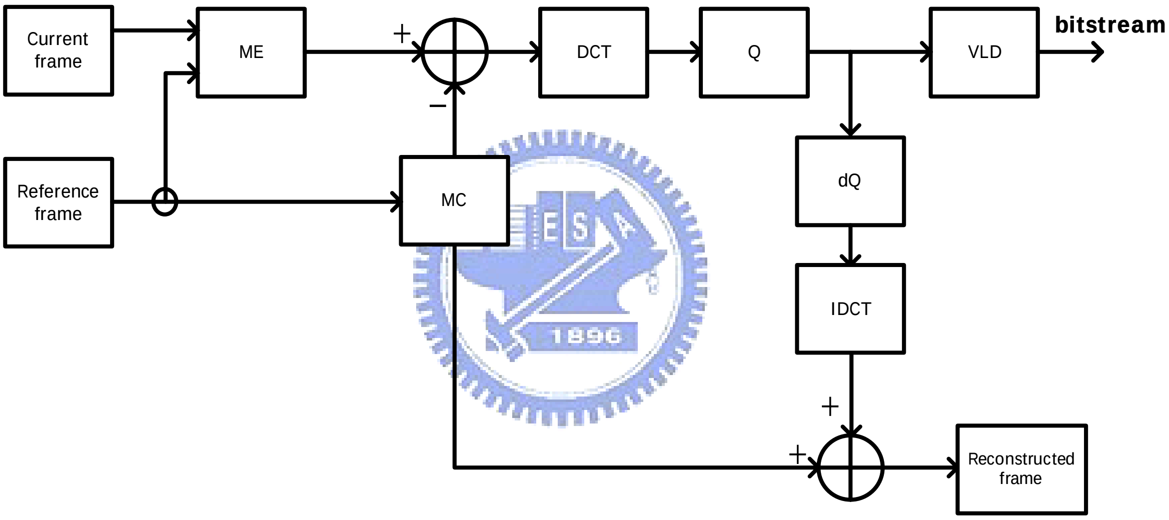


Fig. 2.3 General structure of MPEG-2 encoder

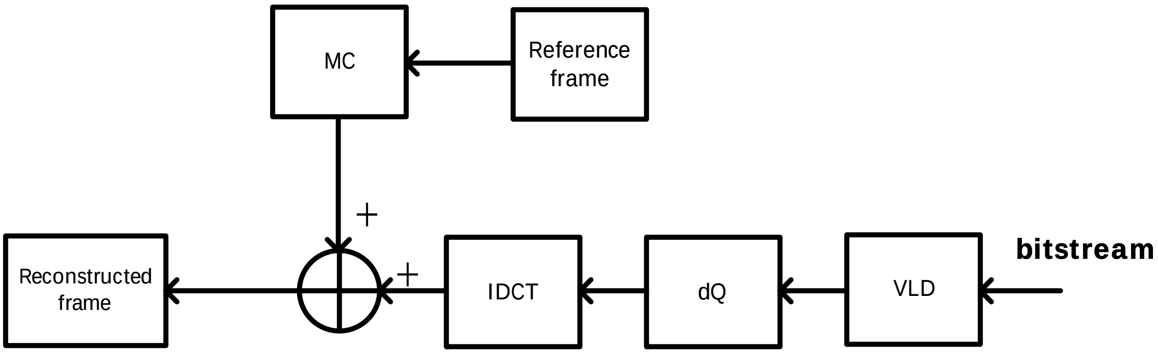


Fig. 2.4 General structure of MPEG-2 decoder

This chapter is structured as follows. The software profiling is illustrated in section 2.1. Then, the algorithm of H.264/AVC motion compensation would be described in section 2.2. Finally, the comparison with those of previous video standards would be discussed in section 2.3.

2.1 Profiling

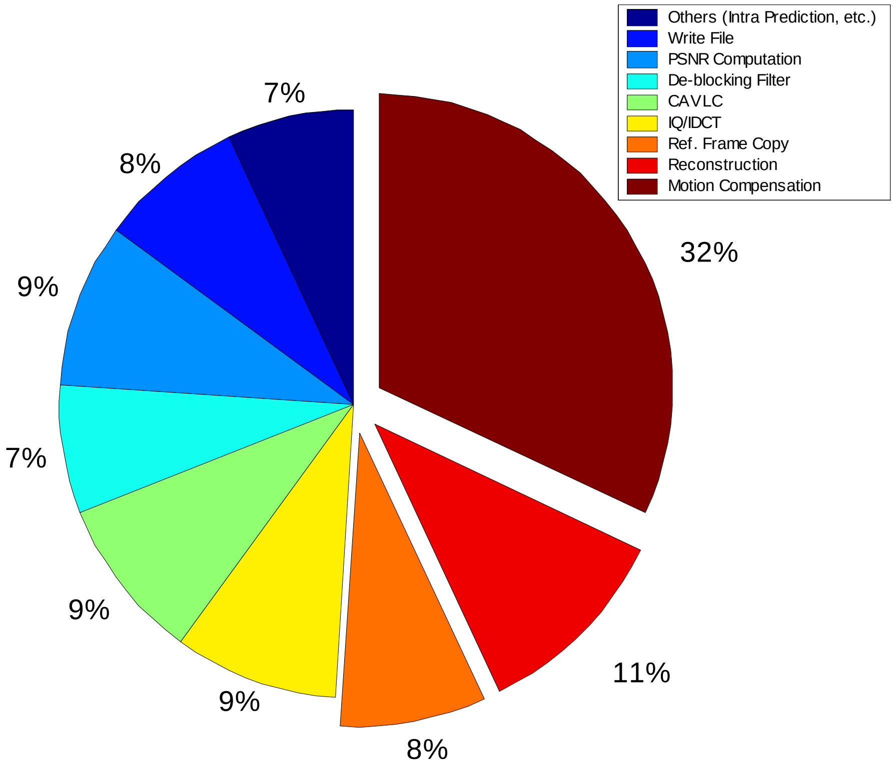


Fig. 2.1 H.264/AVC video decoder software profile on ARM processor (JM 8.2)

Fig. 2.1 shows the H.264/AVC profile on ARM processor. The reference software is JM 8.2. We can find inter prediction related modules, including motion compensation, reconstruction, and reference frame copy, occupy 50 % proportion of the entire video decoder. This dominated part can be greatly reduced by parallel processing, data reuse, or pipeline processing on ASIC design.

2.2 Inter Prediction Algorithm for H.264/AVC Standard

H.264/AVC standard supports more flexible block size selection in inter prediction compared with any previous standard [1][2]. The smallest block size selection could reach as small as 4x4 for luma and 2x2 for chroma. Fig. 2.2 illustrates all types of partitions.

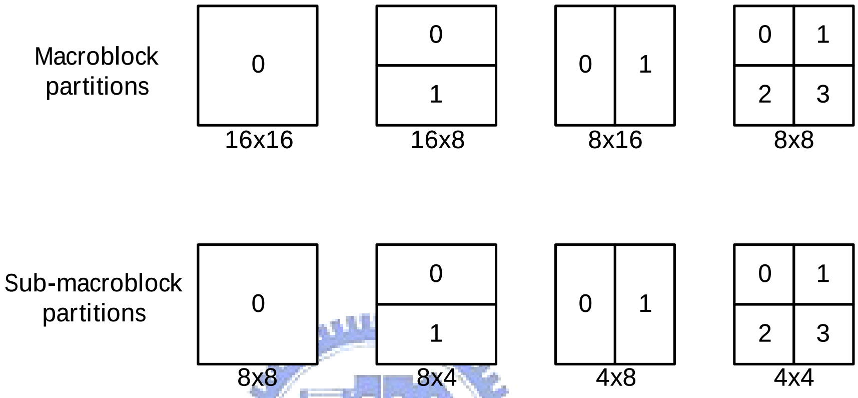


Fig. 2.2 Macroblock partitions and sub-macroblock partitions

H.264/AVC standard also supports high motion resolution that reaches quarter motion accuracy for luma sample and eighth one for chroma sample. This can be found firstly in advances profile of MPEG-4 Visual standard; however, H.264/AVC reduces the complexity of interpolation processing. Luma half sample interpolation with a 6-tap (1, -5, 20, 20, -5, 1) symmetrical FIR filter and quarter sample interpolation with bilinear filter are drawn in Fig 2.3 (a)-(c). The prediction value of chroma component is generated using bilinear interpolator illustrated in Fig. 2.3 (d), and the displacement can achieve one-eighth accuracy. From mathematical equations, they are both 2-D interpolation. However, based on hardware implementation, these equations can be separated into two 1-D to reduce hardware cost, namely, horizontal filter first and than vertical one, or vice verse.

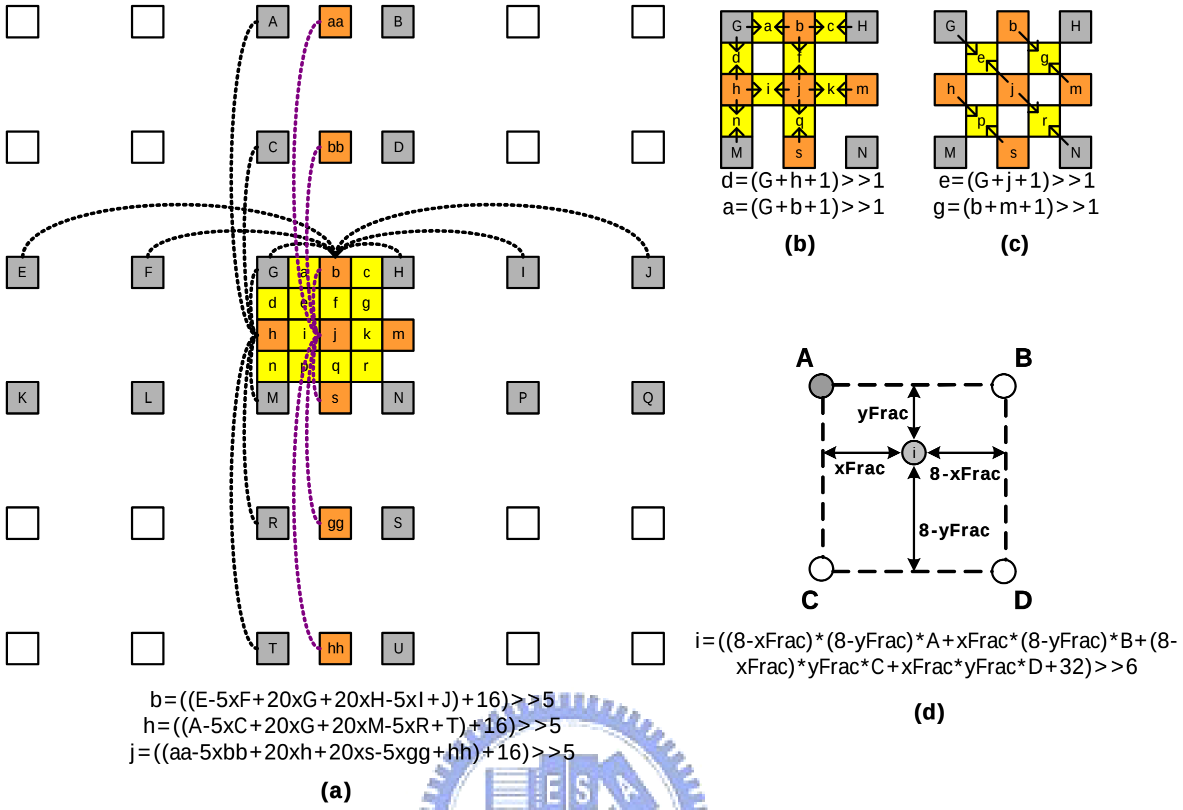

d=(G+h+1)>>1
a=(G+b+1)>>1
(b)
e=(G+j+1)>>1
g=(b+m+1)>>1
(c)
A
B
yFrac
xFrac
8-xFrac
8-yFrac
C
D
i=((8-xFrac)*(8-yFrac)*A+xFrac*(8-yFrac)*B+(8-xFrac)*yFrac*C+xFrac*yFrac*D+32)>>6
(d)
b=((E-5xF+20xG+20xH-5xI+J)+16)>>5
h=((A-5xC+20xG+20xM-5xR+T)+16)>>5
j=((aa-5xbb+20xh+20xs-5xgg+hh)+16)>>5
(a)

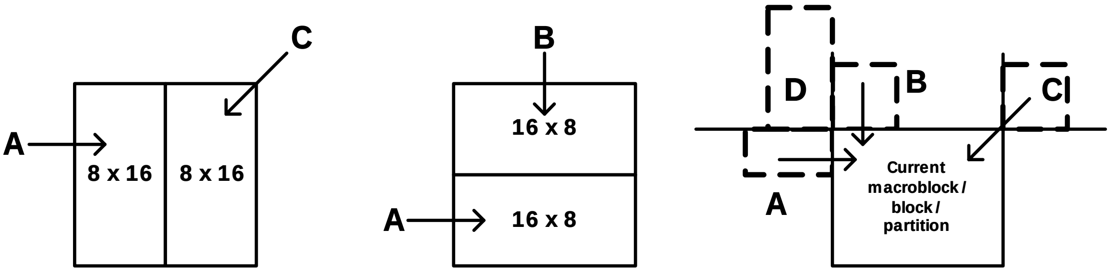
Fig. 2.3 (a) luma half sample with 6-tap FIR, (b) luma horizontal/vertical quarter sample with bilinear filter, (c) luma diagonal quarter sample with bilinear filter, (d) chroma sample with bilinear filter. Upper-case letters indicate the full samples and lower-case letter indicates the interpolated fractional samples

Motion vector is generated from motion vector difference (MVD) and motion vector prediction (MVP) which equation is expressed by (2. 1).

$$\begin{aligned} MVx &= MVDx + MVPx \\ MVy &= MVDy + MVPy \end{aligned} \qquad (2.\ 1)$$

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 Fig. 2.4. Equation of median prediction is expressed by (2. 2). In addition, some boundary conditions or exceptions have to be handled accurately. For instance, when MVC is not available, its value is replaced by MVD. We do not go into detail of those trivial boundary conditions over here.

$$MVP = median(MVA, MVB, MVC) \tag{2. 2}$$



(a) (b) (c)

Fig. 2.4 (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 Fig. 2.2, 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. Macroblock coded in P_SKIP are often located in large area with no change or low motion. Besides the above techniques, H.264/AVC also supports multiple reference frame, weighted prediction and direct mode for B slice. These tools can also improve coding efficiency efficiently. 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.

2.3 *Comparison among Different Video Standards*

Considering frame coding, Table 2.1 lists all fractional motion compensation features for different standards. Up to now, we can find fractional interpolation issue becomes more and more important in 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 x 9 pixels window to interpolate luma 4 x 4 block for H.264/AVC; however, the identical size of interpolation window can be used to filter 8 x 8 block for MPEG-2 video decoder. Fig. 2.5 and Fig. 2.6 show the luma and chroma integer/fractional motion vector proportion for H.264/AVC. 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 computation sources cannot be shared. Therefore, the combination of luma and chroma parts is the space of improvement and we will give the discussion and implantation in Chapter 3. In high bit rate application (128 kbps), fractional motion vector occupies about 80 % and even in low bit rate (32 kbps) fractional part has a certain proportion (40 %). Higher fractional MV proportion, more execution time is needed to read pixels data from frame memory. This gap may become more obvious especially when SDRAM is used as frame memory. To reduce requisite fetching pixels from frame memory, a data reuse technique for fractional motion compensation will be proposed in Chapter 3.

Table 2.1 Comparison of fractional motion compensation among different standards

Standard	MPEG-1/2	MPEG-4	H.264/AVC
MVP	Update from previous PMV value	Median prediction	Median prediction Directional prediction
Luma block unit	16 x 16	8 x 8	4 x 4
Luma motion accuracy	Half	Half, quarter	Half, quarter
Luma filter	Bilinear	**Half sample mode** Half: bilinear **Quarter sample mode** Half: 8-tap FIR Quarter: 8-tap FIR and bilinear	Half: 6-tap FIR Quarter: 6-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
Chroma motion accuracy	Half	Half, quarter	Eighth
Chroma filter	Bilinear	**Half sample mode** Half: bilinear **Quarter sample mode** Half: 8-tap FIR Quarter: 8-tap FIR and bilinear	Bilinear
Chroma interpolation window	9 x 9	5 x 5	3 x 3

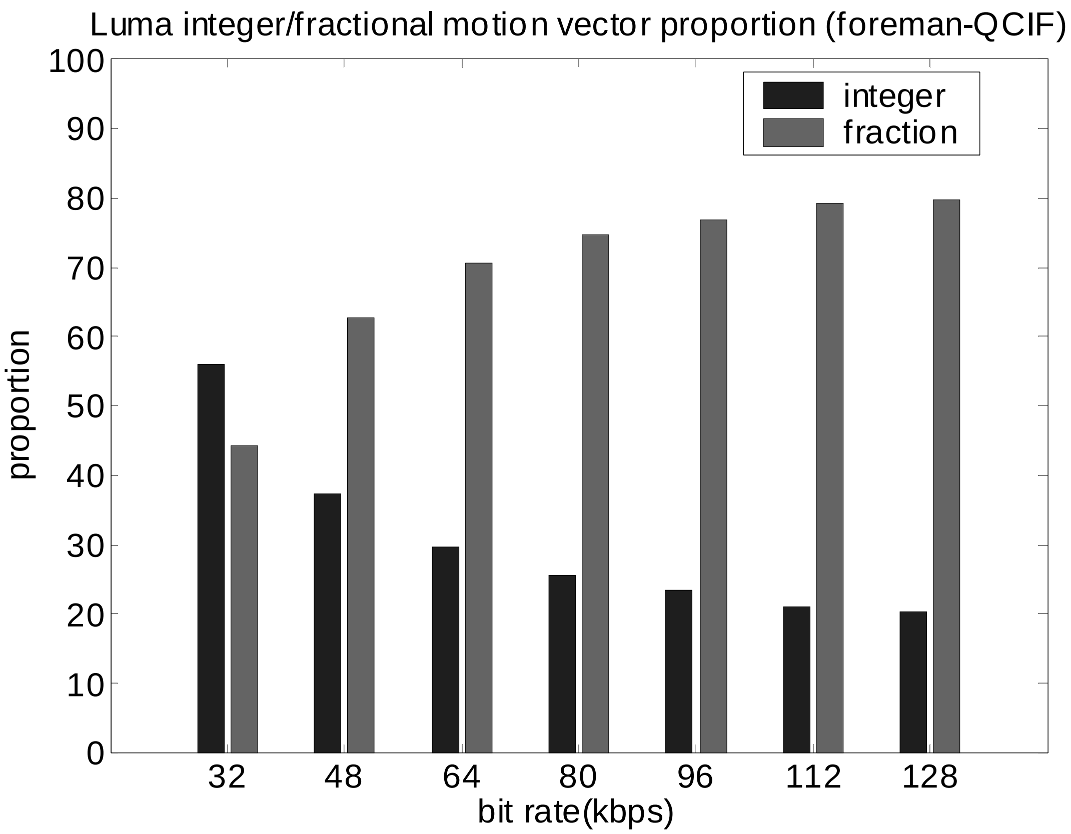


Fig. 2.5 Luma integer/fractional motion vector proportion for H.264/AVC

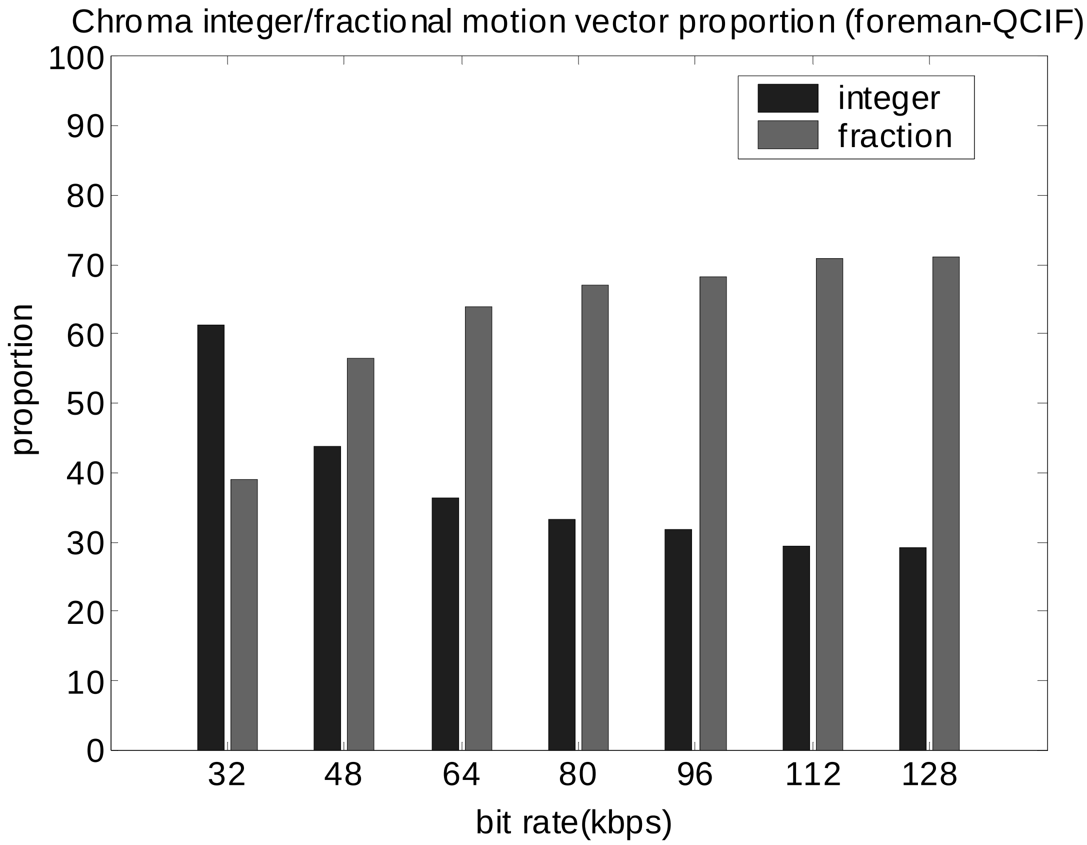


Fig. 2.6 Chroma integer/fractional motion vector proportion for H.264/AVC

2.4 *Summary*

From the H.264 profiling on ARM processor, an efficient hardware accelerator or ASIC design for motion compensation is crucial. The inter prediction for H.264/AVC and the comparison among different standards are also illustrated in this Chapter.

Chapter 3

Motion Compensation Design for MPEG-2/H.264 video decoder

The state-of-the-art video coding standard H.264/AVC provides amazing compression ratio that significantly outperforms all previous video compression standards. However, unlike traditional MPEG-x standards, H.264/AVC lacks backward compatibility to the former MPEG-x and H.26x video coding standards. Therefore, a development of combining multi-video coding standards is essential to support modern multimedia systems. For example, DVD forum adopted MPEG-2, H.264/AVC, and VC-1 (also named well-known WMV-9) as mandatory for the next generation HD-DVD and Blu-ray format. As for digital video broadcasting (DVB) application, DVB-T system, which is designed for digital terrestrial television services, is directly compatible with MPEG-2 coded TV signal. Furthermore, mobile DVB, presently called DVB-H, allows the transmission with video content of H.264/AVC due to high coding efficiency. Especially, DVB-H features backward compatibility with DVB-T but transmit different video format. Therefore, it is the demand and challenge of designing efficient video decoder for multi-standard video application.

This chapter will discuss that designing motion compensation, which dominates the amount of data transfer on the video decoder, for MPEG-2/H.264 dual video decoder. The rest part is structured as follows. Section 3.1 illustrates motion compensation engine for H.264/AVC decoder. The combined motion compensation engine for MPEG-2/H.264 and analysis is discussed in section 3.2. Finally, summary is given in section 3.3.

3.1 Motion Compensation Engine for H.264/AVC decoder

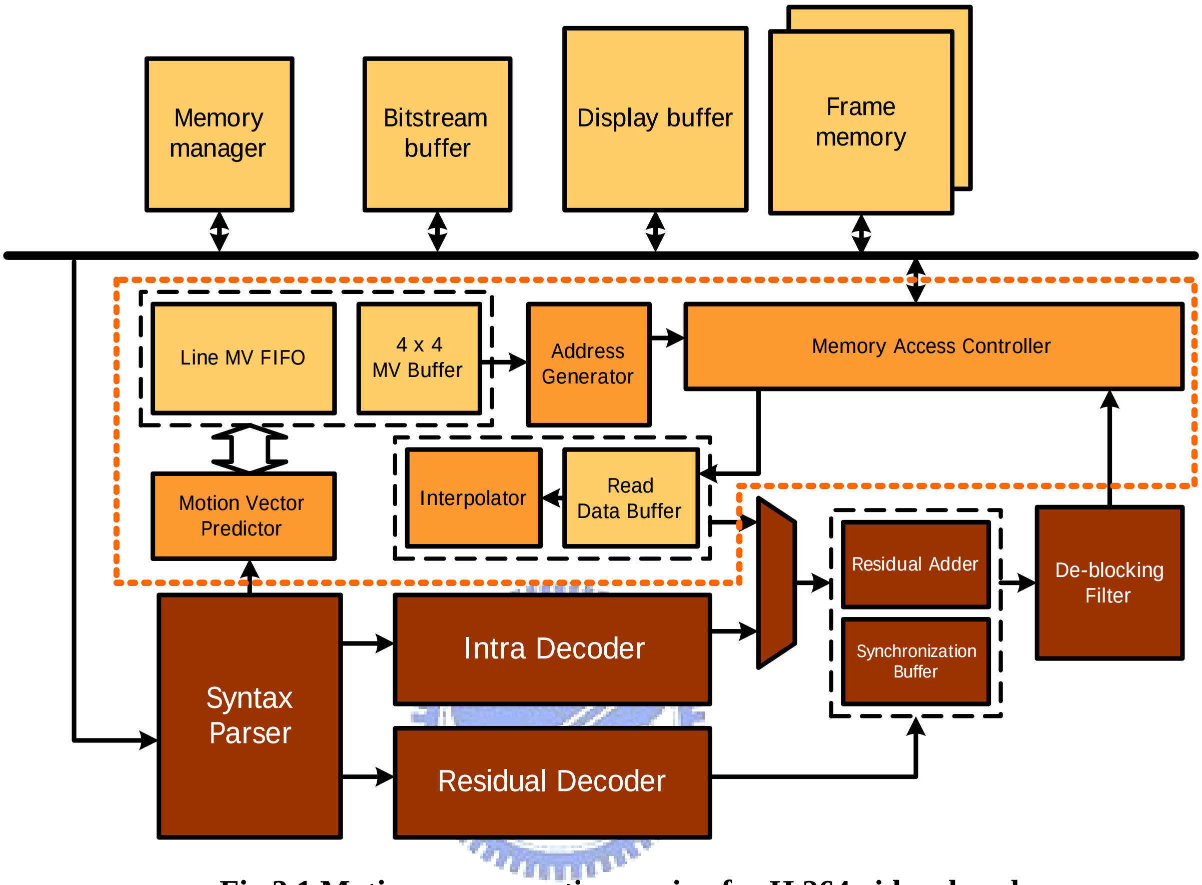


Fig 3.1 Motion compensation engine for H.264 video decoder

Fig. 3.1 illustrates the whole motion compensation engine for H.264/AVC video decoder. Firstly, line MV FIFO stores decoded motion vectors for motion vector prediction and 4 x 4 MV buffer stores the decoded motion vector for current MB decoding. Then, the address generator sends reference address to memory access controller. The tasking of motion controller is scheduling consecutive access command and sending to frame memories. The burst read data is kept in read data buffer and then filtered through interpolator. Finally, the interpolated reference data add up to the residual data and then pass through de-blocking filter. In our proposed decoder, ping-pong structured external frame memory [28], double memories stored reference and current frame reciprocally, is adopted.

The following subsection will discuss the detail of other modules except memory access controller. The detailed discussion of frame memory access controller is shown in Chapter 4. Subsection 3.1.1 illustrates motion vector generator including motion vector predictor (MVP) and the related storages. Subsection 3.1.2 gives data reuse technique for interpolator. Subsection 3.1.3 analyzes the proposed data reuse technique. Finally, luma and chroma interpolator designs are reported in subsection 3.1.4 and 3.1.5 respectively.

3.1.1 Motion Vector Generator

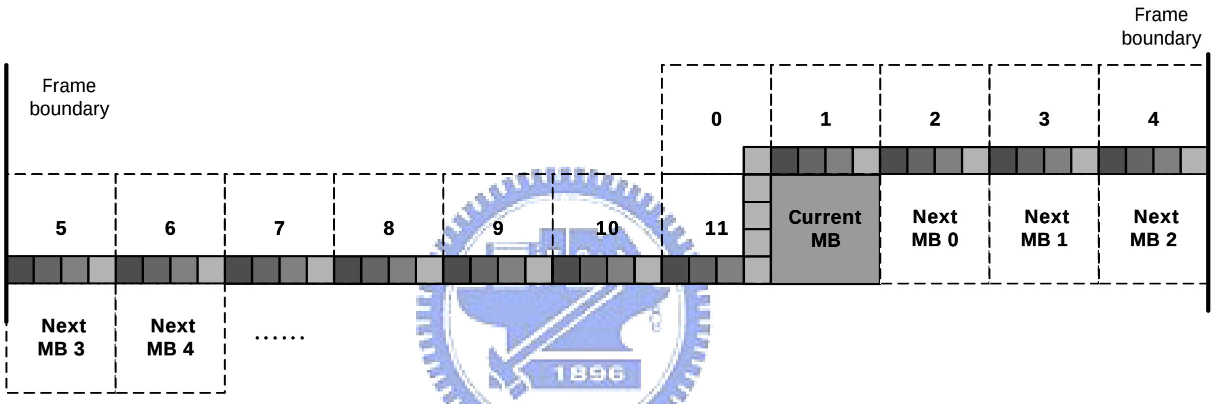


Fig 3.2 Motion vectors information storage or motion vector predictor for QCIF frame format.

Motion vector generator mainly contains motion vector predictor, line MV FIFO and 4 x 4 MV buffers. Motion vector is generated by the summation of motion vector prediction (MVP) and motion vector difference (MVD). The MVP value is calculated according to the neighboring MVs, thus the decoded motion vectors are required to be stored for the following decoding. Line MV FIFO stores the decoded motion vector pair (MVX, MVY). The depth and width of MV FIFO are dependent on the frame width and search range respectively. Once the content of MV FIFO will not be used in the future, the motion vector pair can be discarded. The 4 x 4 size of MV buffers is required since the maximum number of motion vectors per

MB is sixteen. The motion vectors for current MB decoding stores in this 4 x 4 MV buffers.

As for the requisite total storage for motion vector generator, Fig. 3.2 shows an example. Total amount of 4 x 11 motion vector pairs have to be stored for QCIF frame format. The detail of required neighboring motion vectors is shown in Fig. 3.3. To cover all kinds of 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 decode MV0-MV15 in current MB, it needs neighboring motion vectors in left-upper corner (MVLU), right-upper corner (MVRU), upper (MVU0-3) and left (MVL0-MVL3) positions.

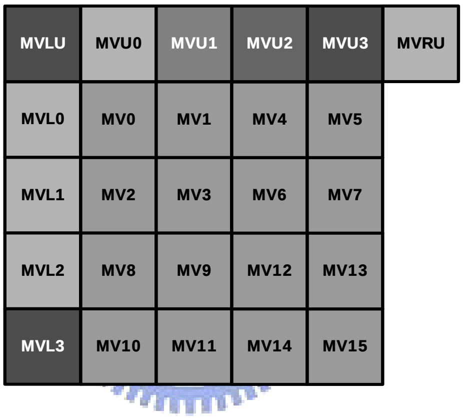
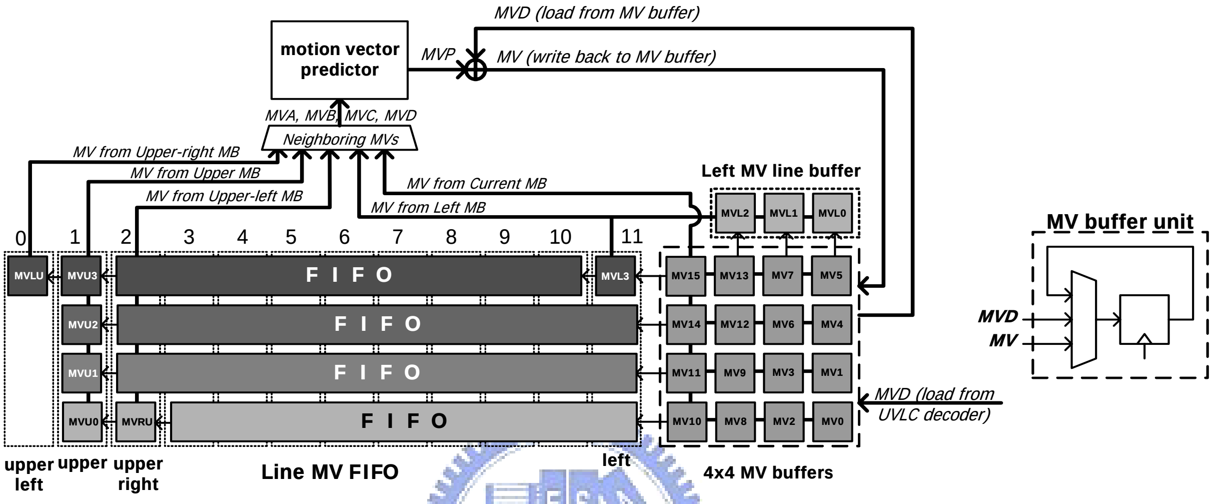


Fig 3.3 Neighboring motion vectors needed when decoding all motion vectors in current macroblock

The detailed architecture of motion vector generator is shown in Fig 3.4. Motion vector generation involves two-phase operations. The first one is loading MVD into 4 x 4 MV buffers and another is calculating MV = MVP + MVD then restoring into 4 x 4 MV buffers. The proposed memory storage can be treated as two-level memory hierarchy painted in Fig 3.5. Four line MV FIFOs are implemented using SRAM and local registers store the neighboring motion vectors for current MB. Local register that stores neighboring motion vectors includes left MV line buffer, upper-left, upper, upper-right and left MV registers. The 4 x 4 MV buffers, which contents can be accessed by other modules, store decoded motion vectors required in current MB decoding. After accomplishing current MB decoding, FIFOs need one push and one pop operation, which occupies two cycles, to update all contents of local registers for the next MB decoding.



(a) (b)

Fig 3.4 (a) motion vector generator architecture for QCIF-format, (b) mv buffer unit

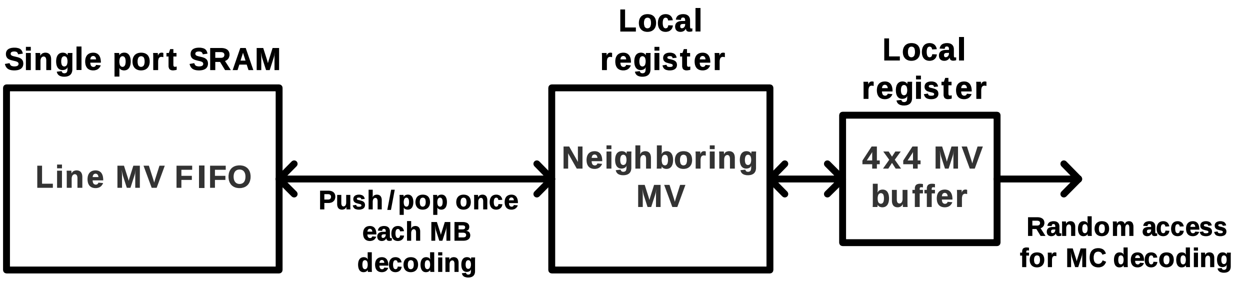
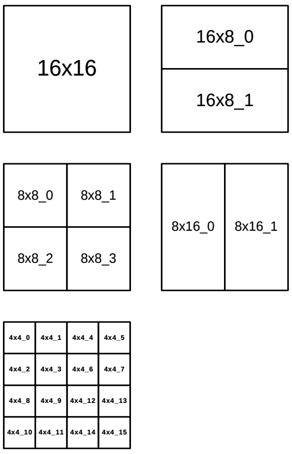


Fig. 3.5 Two-level memory hierarchical structure for MVP



(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)

Fig 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 whose values are derived from neighboring motion vectors according to block size_position index illustrated in Fig. 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 Fig. 2.3 (c). Fig. 3.6 (b)-(d) lists all MVA, MVB, MVC and MVD for different block size_position index. Besides 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 clarity.

3.1.2 Data Reuse Technique for Interpolator

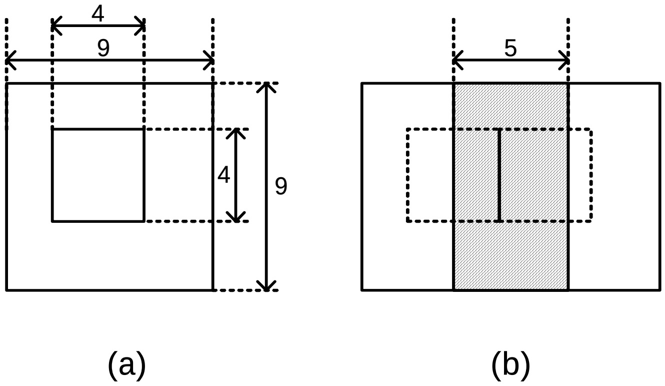


Fig 3.7 (a) 4x4 block window and the corresponding 9x9 interpolation window, (b) overlapped region for neighboring interpolation window

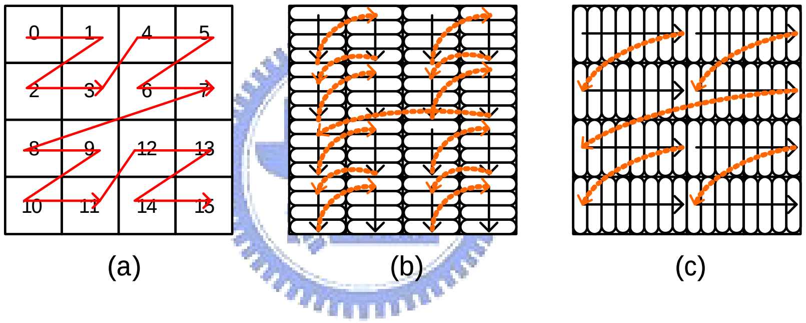


Fig 3.8 (a) 2x2 raster scanning order, (b) row-major 2x2 raster scanning order, (c) column-major 2x2 raster scanning order

From Fig 3.7 (a), to interpolate each fractional sample value for each 4x4 block, it needs 9 x 9 interpolation window. If two motion vectors of neighboring 4 x 4 blocks are the same, 5 x 9 overlap region between two interpolation windows can be data reused. The scanning order of residual decoding for each macroblock is 2x2 raster scanning order as shown in Fig 3.8 (a). Then, considering two different scanning orders illustrated in Fig 3.8 (b) and (c), row-major one needs 13 times of transitions but column-major one only needs 5 times of transitions. Each transition causes that the overlap region could not be data reused. Therefore, column-major one is the better selection because of less number of transitions.

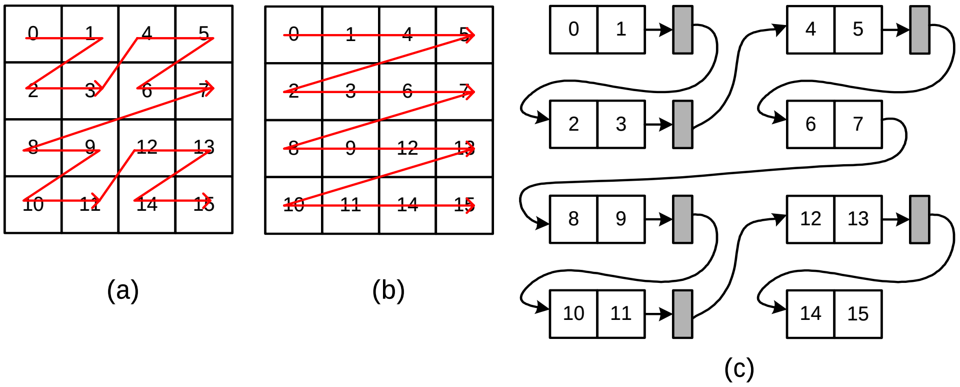


Fig 3.9 (a) 2x2 raster scanning order, (b) 4x4 raster scanning order, (c) extended 2x2 raster scanning order

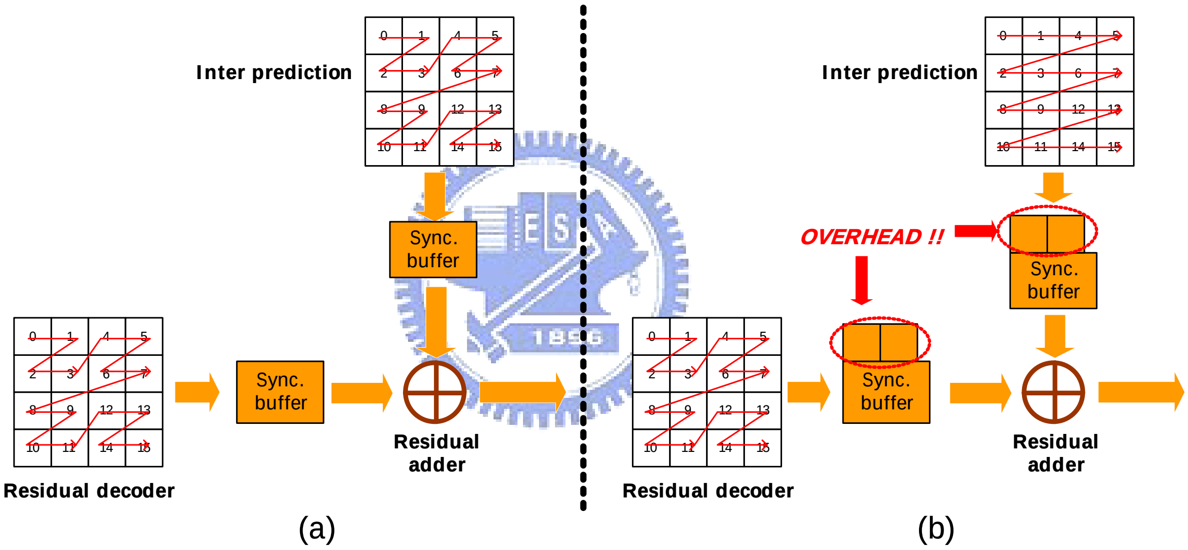


Fig 3.10 Synchronization buffer scheme for two different scanning order in inter prediction (a) 2x2 raster scanning order, (b) 4x4 raster scanning order

For video decoding system, inter prediction often processes based on macroblock level. Thus, the decoding order based on 4 x 4-block size, which is the smallest block element in H.264/AVC video decoder, is freedom for each macroblock. In view of this concept, 2 x 2 and 4 x 4 raster scanning orders are depicted in Fig 3.9 (a) and (b), and we can find column-major 4 x 4 raster scanning order only needs four transitions less than that of 2 x 2 raster scan. However, from Fig 3.10 (b), it induces extra synchronization buffers which size is (4 x 4) x 4 pixels in residual adder because of different scanning order with residual decoder which must follow 2x2 raster scanning order defined in standard [1].

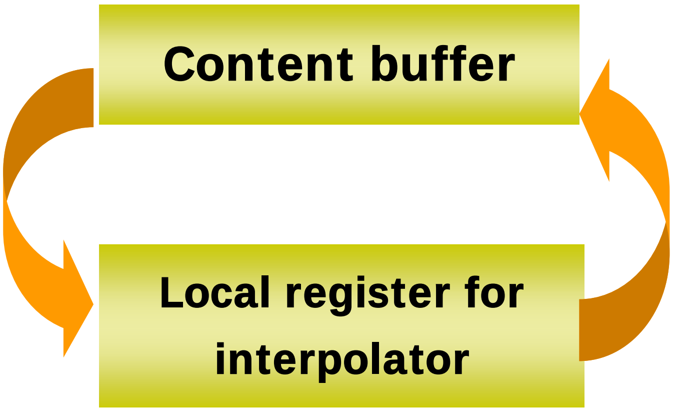


Fig. 3.11 Content-swap operation (interpolator with attached content buffer)

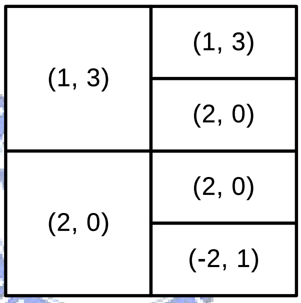


Fig. 3.12 An example of macroblock partition

(1, 3) indicates (mv_x, mv_y).

In order to resolve this problem, we can attach content register to interpolator which concept is illustrated in Fig 3.11, and adopt extended 2x2 raster scanning order as shown in Fig 3.9 (c). The size of content register depends on the local register in interpolators. Each gray block in Fig. 3.9 (c) indicates content-swap operation that swaps all content in local register in interpolation and that in content buffer. By doing that, we can find that if motion vectors of block 1 and block 4 are the same, the overlapped region in Fig. 3.7(b) need not to be re-fetched when decoding block 4. Therefore, extended 2x2 raster scanning order follows 2 x 2 raster scanning which is the same as that of residual decoder, and achieves data reuse status of 4 x 4 raster scanning order. The content-swap operation brings effect only when larger block size partition or motion vectors of the neighboring blocks are the same. The condition that executes this operation follows the expression (3. 1)

$$content\_swap\_condition = (mb\_type == 16x16) \,||\, (mb\_type == 16x8) \qquad (3.\ 1)$$

However, considering an example shown in Fig. 3.12, the condition (3.1) checking is false. Furthermore, if checking equality of neighboring motion vectors instead of block size, the example in Fig. 3.11 can be data reused. The checking table of motion vectors between neighboring blocks is listed in Table 3.1.

Table 3.1 Neighboring MV checking table for content-swap operation

Block number	Checking condition
1	MV1 = = MV4
3	(MV1 = = MV4) || (MV3 = = MV6)
5	MV3 = = MV6
9	MV9 = = MV12
11	(MV9 = = MV12) || (MV11 = = MV14)
13	MV11 = = MV14
Other	Don’t care

3.1.3 Analysis for Data Reuse Technique

To give more generic and platform independent analysis, we analyze requisite pixels per MB and cost overhead for each method. Taking account of fractional motion compensation for each macroblock, the required pixels for each MB and cost overhead for different methods are summarized in Table 3.2. Assuming that each motion vector contains fractional part, the best case has one motion vector and worst case has 16 motion vectors for one luma macroblock. Although requisite pixels per method are the same in worst case, requisite pixels of column major methods are smaller than that of row-major method. Concerning column-major methods, 4 x 4 raster scanning order (RSO) takes the best effect; however, it requires additional synchronization buffer and degrades throughput due to different RSO with that of residual decoder. As for extended methods, condition (3. 1) only takes effect in larger block partition (SKIP, 16x16, 16x8). That is, it cannot data-reuse in some case such as Fig. 3.11 even if the neighbor motion vectors are the same. To erase this disadvantage, method 5 checks the neighboring motion vectors rather than block size, and then the required bandwidth can reduce to be the same as that of 4 x 4 RSO in Fig. 3.12 case. The advantage of extended method is that it only requires content buffer which size is smaller than that of method 3 and takes a little extra cycle for content-swap operation. Although method 4 behaves better for larger block size (SKIP, 16x16, 16x8) than method 1/2/3, larger block size still occupies up to 50 ~90 % proportion from the Fig. 3.13. Furthermore, method 5 not only involves all case in method 4 but also takes effect in smaller block size such as Fig. 3.1. As shown in Fig. 3.14, after applying extended method in our design, the required memory bandwidth can be reduced about 30 % compared to column-major 2x2 RSO method.

Table 3.2 Static analyses for different method in H.264/AVC.
Assumption: each motion vectors contains fractional part.

Method			Required pixels per luma MB			Cost overhead
			Worst case	Best case	Fig 3.11	
1	**R**	**2 x 2 RSO**	1296	1296	1296	0
2	**C**	**2 x 2 RSO**	1296	936	936	0
3		**4 x 4 RSO**	1296	756	846	Checking table Sync. buffer
4		**Extended 2 x 2 RSO (condition (3.1))**	1296	756 + 6 CS	936	Extra control signal, Content buffer
5		**Extended 2 x 2 RSO (Table 3.1)**	1296	756 + 6 CS	846 + 4 CS	Checking table, Content buffer

∗ R: row-major, C: column major, RSO: raster scanning order, CS: content-swap operation (one cycle)

∗ Best case: one MB contains one motion vector

∗ Worst case: one MB contains 16 motion vectors

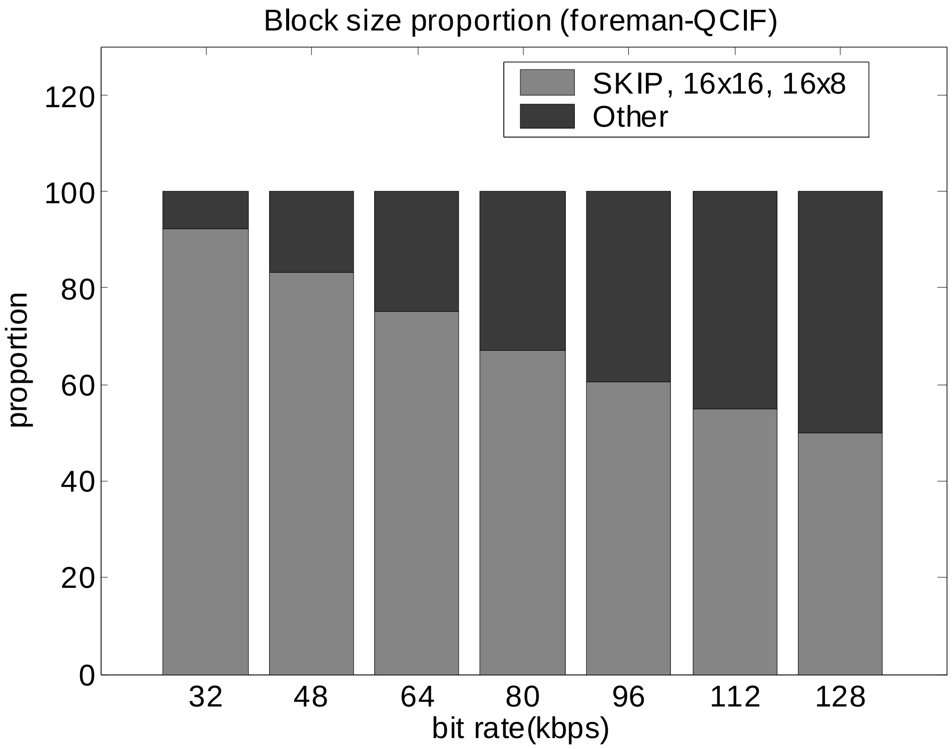


Fig. 3.13 Block proportion under different bit-rate environments

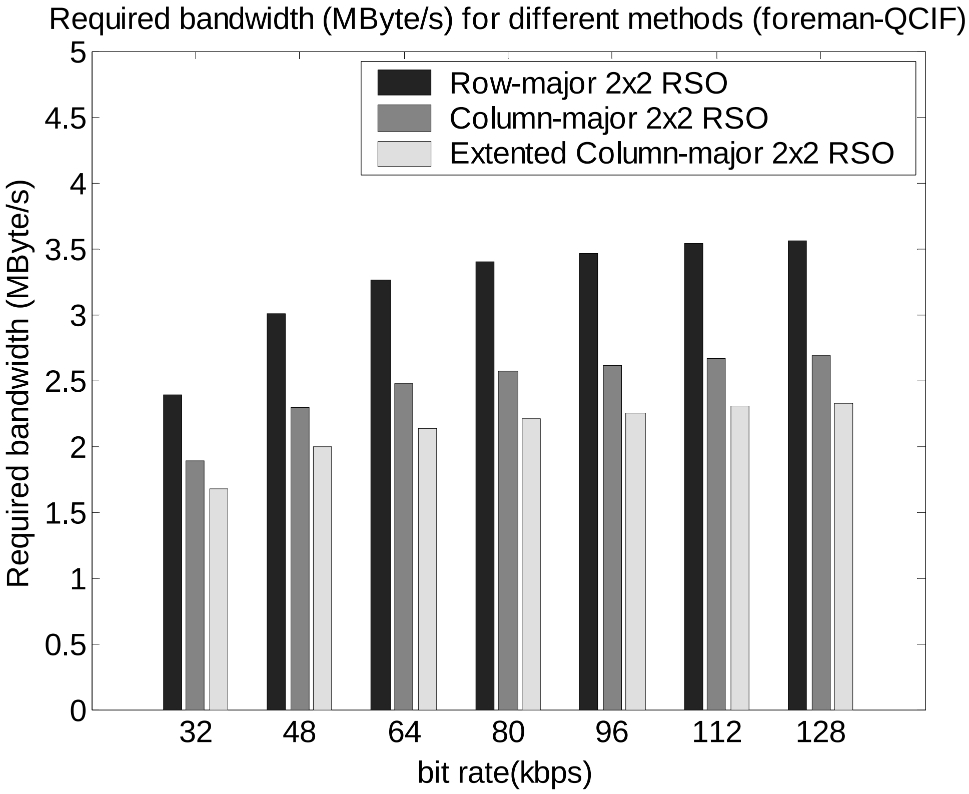


Fig. 3.14 Required memory bandwidth for different methods

3.1.4 Luma Interpolator Design

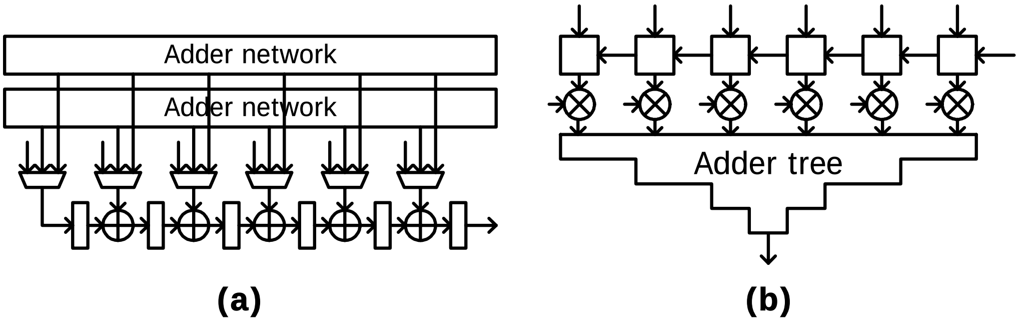


Fig 3.15 (a) adder-chain based [10], (b) adder-tree based [11]

1-D linear interpolator design

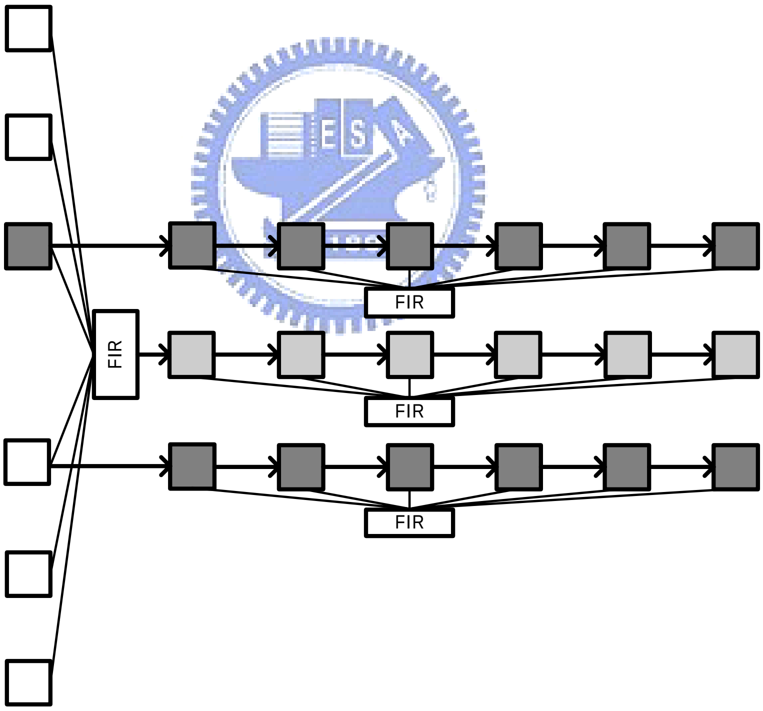


Fig. 3.16 Separate 1-D interpolator design (no parallel)

In this subsection, several different interpolator designs will be reported. Reviewing the fractional interpolation for H.264/AVC in Fig. 2.2, 6-tap FIR with (1, -5, 20, 20, -5, 1) coefficient and bilinear filter are needed for half and quarter pixel 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 [8], and Chen's HVBi, bilinear filter in both horizontal and vertical direction, and VBi, vertical bilinear horizontal FIR, schemes are also reported in [9]. However, when P frame sequence is very long, such as I + 29 P, the propagation of PSNR loss may cause the heavy degradation of video quality, especially in high definition frame format. Oppositely, considering PSNR losses and standard-compatible design, Chien's [10] and He's [11] presented adder-chain and adder-tree based design respectively. These two types depicted in Fig. 3.15 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 application; 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 [10] 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. Fig. 3.16 shows separate 1-D interpolator design without processing in parallel.

Table 3.3 Comparison of execution cycles for different architectures

Architecture	Ideal execution cycles
Adder-chain based 1-D	57
Adder-tree based 1-D	52
Separate 1-D (no parallel)	36
Separate 1-D (2 parallel)	18
Separate 1-D (4 parallel)	9

Assuming that all 9 x 9 interpolated data for each 4 x 4 block are ready and they can be accessed randomly, Table 3.3 lists the execution cycles for different architecture. For adder-chain based 1-D design, the first result outputs after the 6th 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 x 9 + 4 x 4) which detailed operation is described in Chien’s [10]. For adder-tree based 1-D design, the row data can be loaded in parallel without shift one-by-one, hence the latency overhead does not exist and total number of cycles is 52 (4 x 9 + 4 x 4). As for separate 1-D design, the first data outputs at the 6th clock cycle and the following 3 data generates after 3 clock cycles. Therefore, the separate 1-D design without parallel needs 36 ((6 + 3) x 4) cycles to complete interpolation of one 4 x 4 block. Similarly, separate 1-D design with 2 and 4 parallel requires 18 ((6 + 3) x 2) and 9 (6 + 3) cycles respectively. The required content buffers are 6 x 9 pixels for 4-parallel design shown in Fig. 3.17 and it can be implemented in local registers or SRAM. However, SRAM requires several cycles to accomplish content-swap operation, so we choose local registers in order to execute content-swap in one cycle. In addition, 4-parallel separate 1-D architecture is our selection due to smaller required execution cycles that can be hidden below data-read cycles from frame memory. For another reason, it is easier to combine with interpolation for MPEG-2 video decoding and we will show it in subsection 3.2.1.

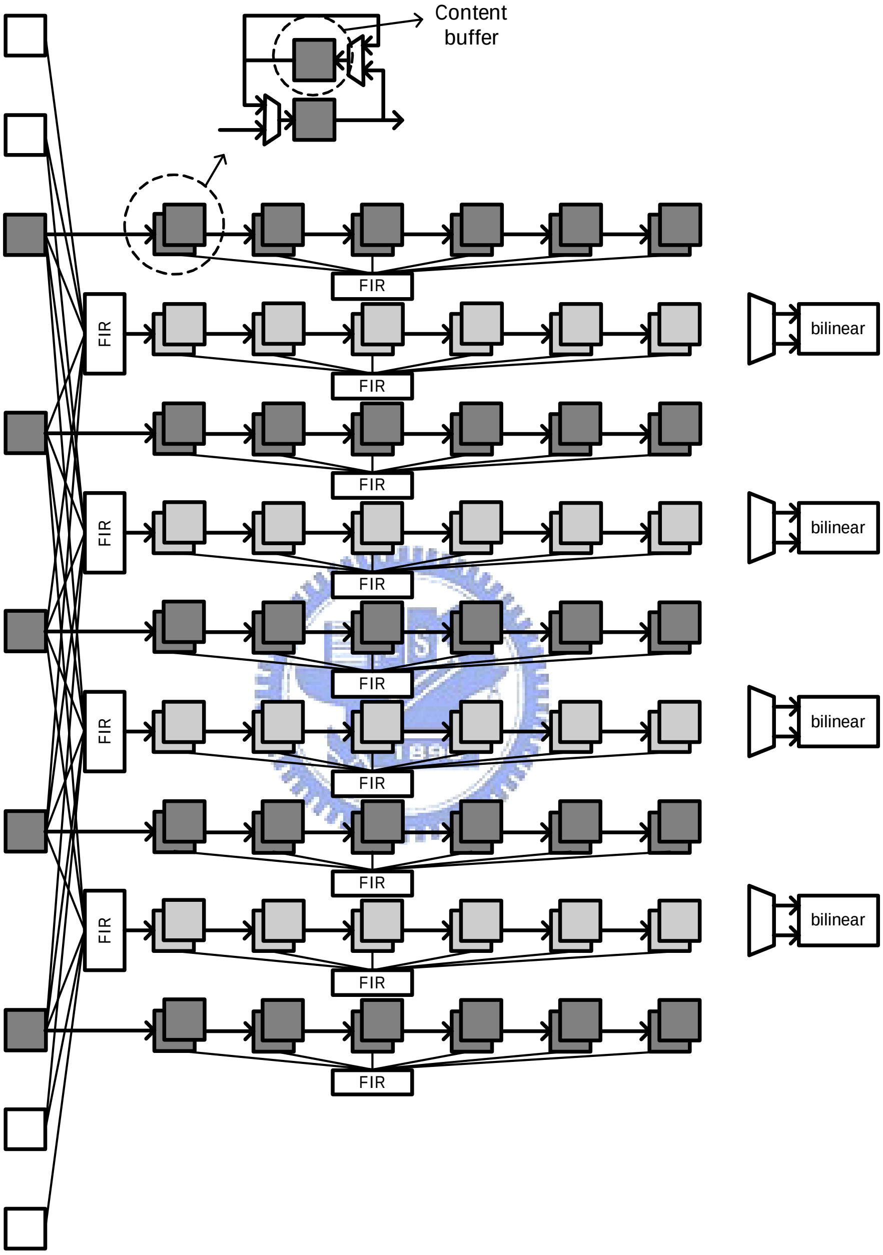


Fig. 3.17 4-parallel separate 1-D luma interpolator with content buffer

3.1.5 Chroma Interpolator Design

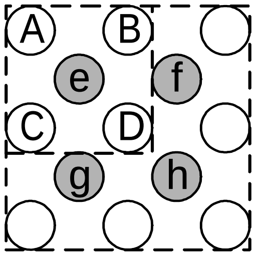


Fig 3.18 Interpolation window for each 2 x 2 chroma block

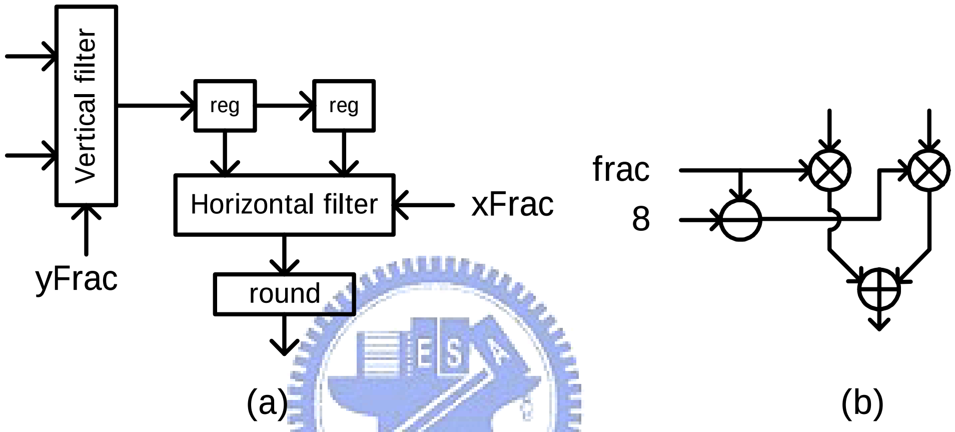


Fig. 3.19 (a) chroma interpolator, (b) vertical/horizontal filter

Because of 4:2:0 chroma format and quarter precision of luma inter prediction, chroma inter prediction can achieve eighth motion resolution. Chroma inter prediction must process based on 2 x 2 block and chroma interpolation requires 3 x 3 pixels for each 2 x 2 block as shown in Fig. 3.18. For chroma 2 x 2 block including A, B, C and D, the corresponding fractional sample is e, f, g and h whose precision is eighth point. Compared with direct mapping design with 8 multipliers which equation is listed in Fig. 2.2 (d), we rewire the equation listed in equation (3. 2) and the number of multiplier number can reduce to 4.

$$\begin{aligned} i &= (8-xFrac)*(8-yFrac)*A + xFrac*(8-yFrac)*B + (8-xFrac)*yFrac*C + xFrac* \\ &\quad yFrac*D = (8-xFrac)*[(8-yFrac)*A + yFrac*C] + xFrac*[(8-yFrac)*B + yFrac*D] \end{aligned} \tag{3. 2}$$

Similar to luma interpolator, chroma interpolator can separate into horizontal and vertical filter. The corresponding separate 1-D design is depicted in Fig. 3.19 (a) and the vertical / horizontal filter is illustrated in Fig. 3.19 (b). Double chroma interpolators are required to generate interpolated value in 2-pixel parallel, and it takes 3 cycles to filter 2 x 2 pixels if all required interpolated pixels are ready. Based on 2-parallel chroma interpolator design painted in Fig. 3.20, only one cycle latency is induced.

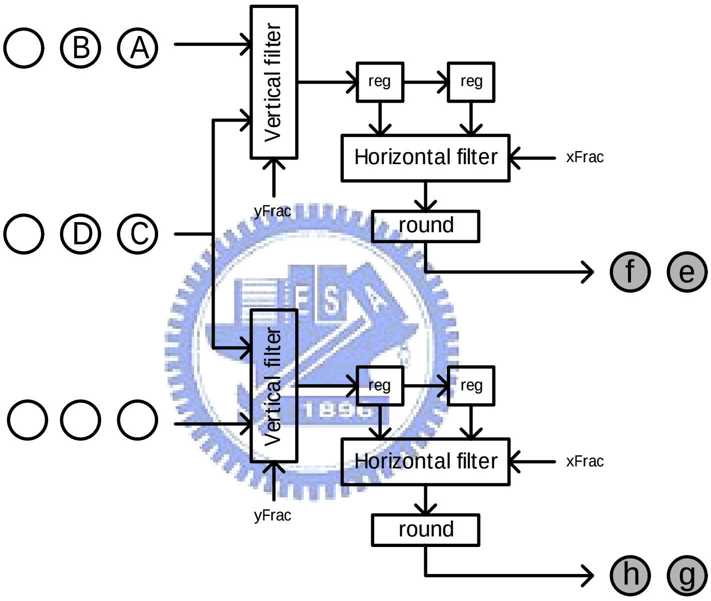


Fig. 3.20 2-parallel chroma interpolator

3.2 Combined Motion Compensation Engine for MPEG-2/H.264 Dual Video Decoder

Our H.264/MPEG-2 dual-standard video decoder is illustrated in Fig. 3.21 and the component of MPEG-2 decoder is depicted in Fig. 3.22. Compared with H.264/AVC standard, MPEG-2 does not provide intra prediction and in-loop de-blocking filter, and only supports half motion precision for both luma and chroma macroblock. Unlike median/directional prediction of MVP algorithm supported in H.264/AVC, motion vectors are only decided by updated PMV and bitstream side information like f_code, motion_residual and motion_code. The detailed algorithm of motion vector generation can refer to [2]. Besides motion vector generator, a reconfigurable interpolator design for dual-standard is proposed in section 3.2.1 and section 3.2.2 gives the cost analysis.

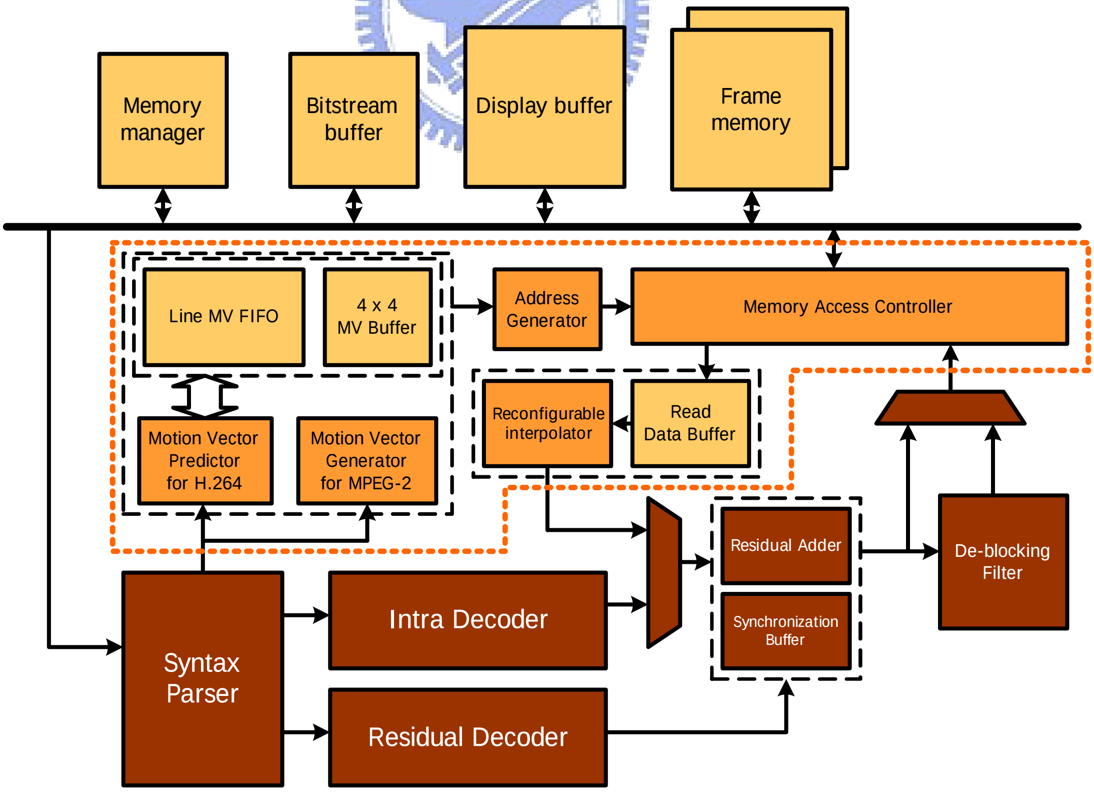


Fig. 3.21 Motion compensation engine for H.264/MPEG-2 decoder

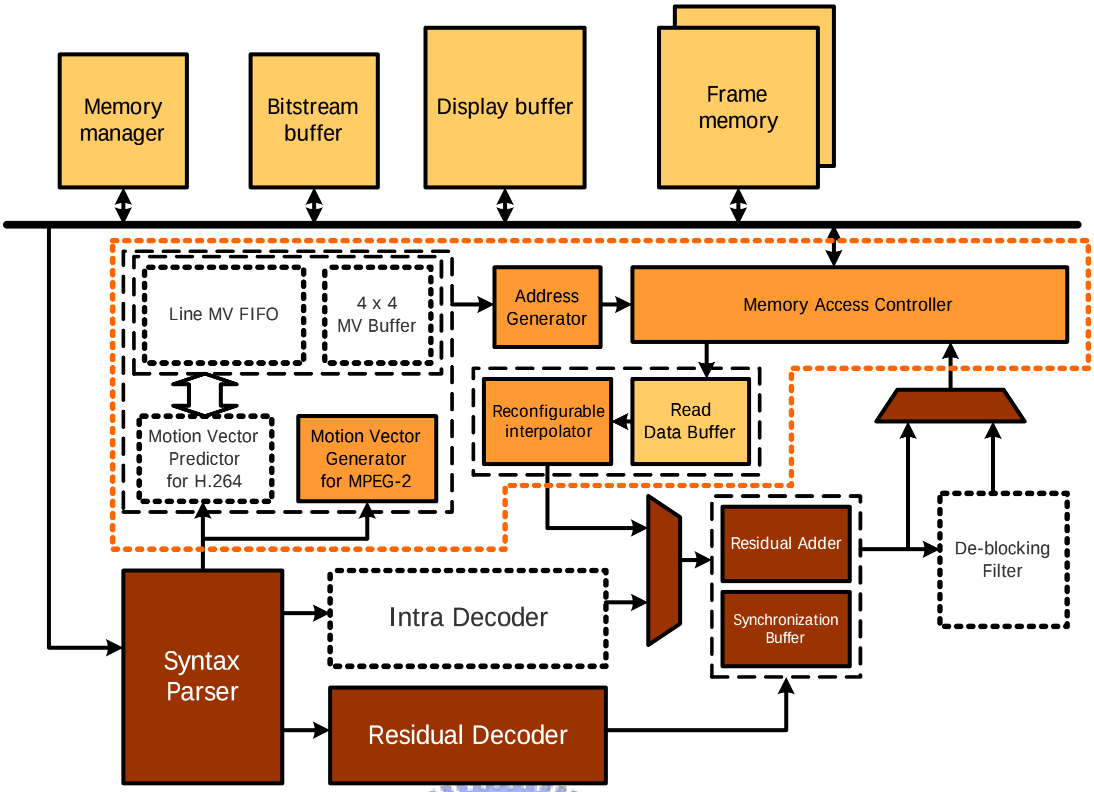


Fig. 3.22 MPEG-2 Motion compensation engine part

3.2.1 Reconfigurable dual-standard interpolator design

The main additional penalty of motion compensation engine is interpolator when combing with MPEG-2 video decoder. In this subsection, we will focus on storage and arithmetic module sharing on dual-standard to minimize area cost overhead. For macroblock based fractional motion compensation in MPEG-2, each 16 x 16 macroblock needs 17 x 17 interpolation windows to interpolate fractional samples. Each macroblock can be partitioned into four 8 x 8 blocks with 9 x 9 interpolation window of which size is identical to that of H.264/AVC luma interpolation window for each 4 x 4 block. In addition, the bilinear filter for H.264/AVC luma quarter interpolation can share with bilinear filter for MPEG-2 half interpolation. Considering 4-parallel luma interpolator as shown in Fig. 3.17, part of registers and bilinear filters, which are shaded in Fig. 3.23, can be shared with MPEG-2 interpolator.

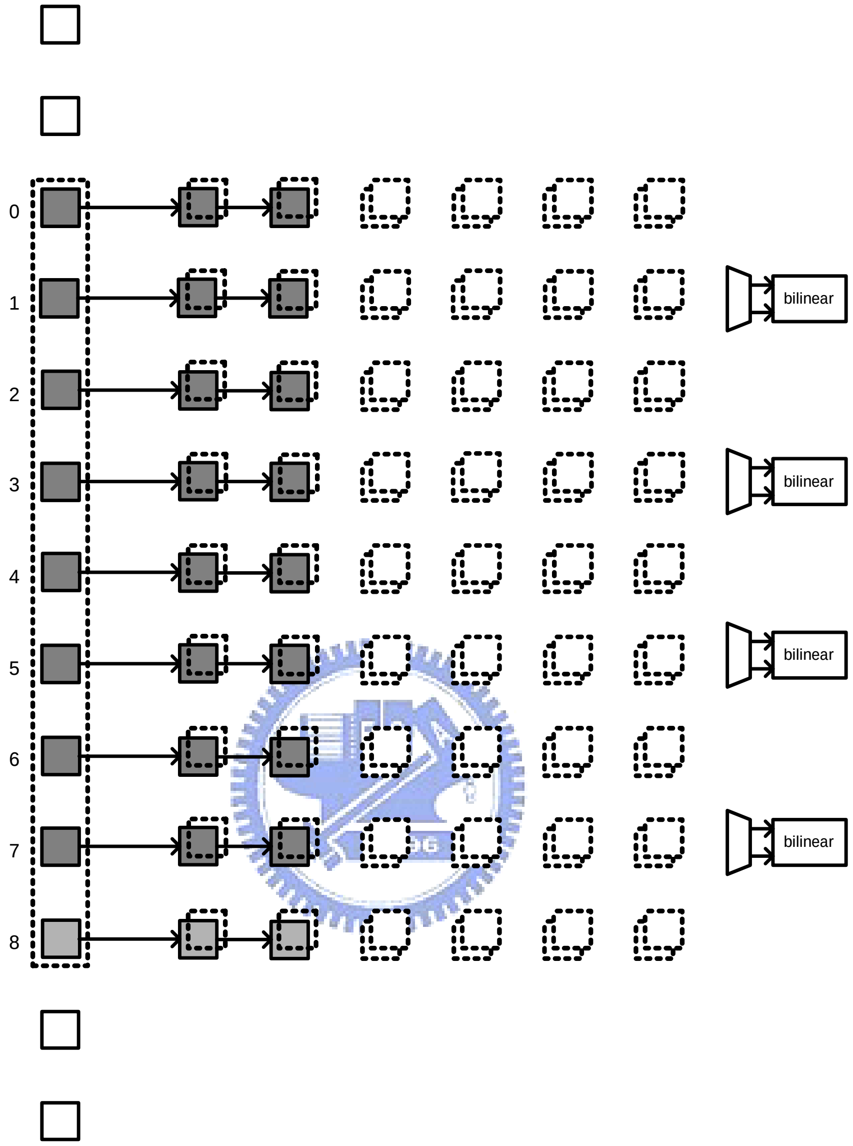


Fig. 3.23 Shared local registers and bilinear filters for MPEG-2

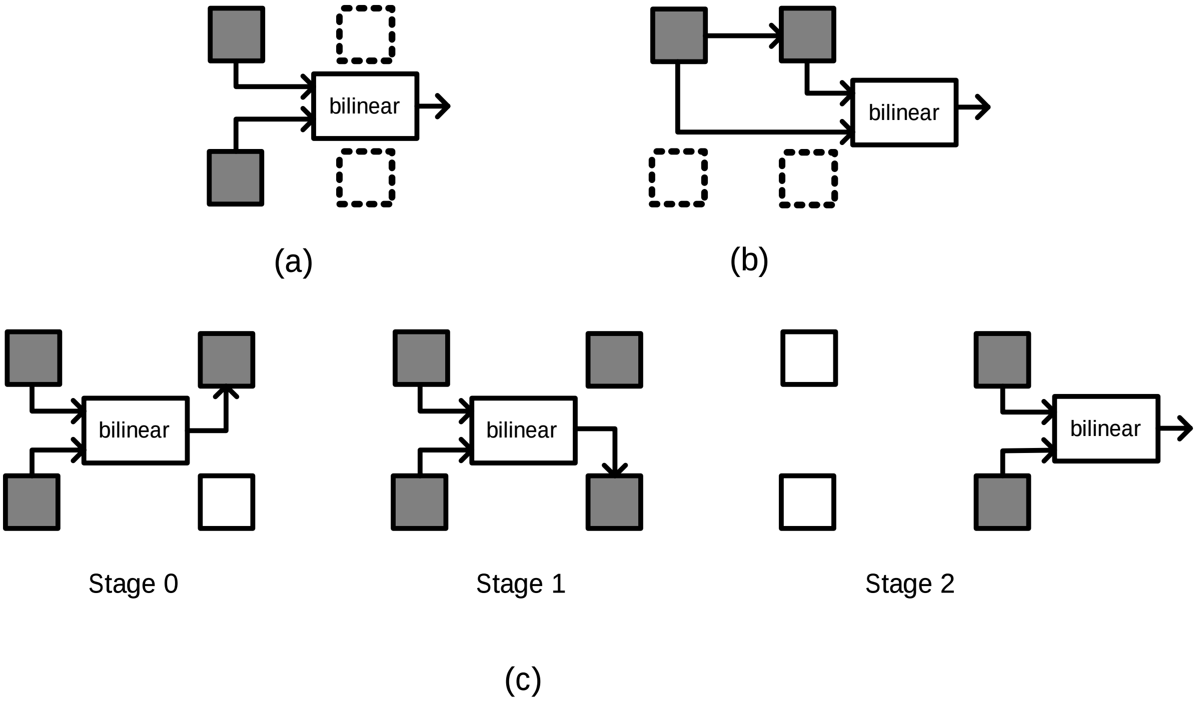
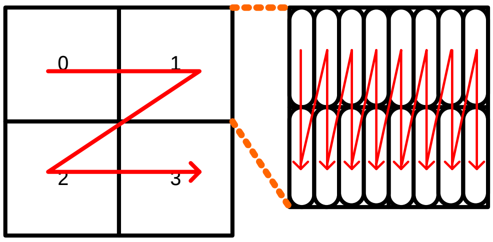


Fig. 3.24 Data flow of (a) vertical bilinear filter, (b) horizontal bilinear filter, (c) both vertical and horizontal bilinear filter

Beside the shared modules described above, only extra control circuits for data flow are required for MPEG-2 interpolation. Fig. 3.24 shows the data flow of vertical or horizontal bilinear filter and half sample flag is decided by the LSB of motion vectors. Firstly, we have to concern IDCT/IIT that is the last stage of MPEG-2/H.264 residual decoder. Inverse discrete cosine transform (IDCT) for MPEG-2 is 8 x 8-block based module, whereas inverse integer transform (IIT) for H.264/AVC is 4 x 4-block based decoding process. To achieve module combining and storage sharing, these two modules can merge to single multi-mode IDCT and the output data are 4-pixel in parallel for both standards. Besides, only four bilinear filters are available for MPEG-2/H.264, hence each column 8-pixel filtering has to separate into two stages involving upper 4-pixel filtering (0-4) and lower 4-pixel filtering (4-8). For each MB decoding, the decoding scanning order based on 8 x 8 block is shown in Fig. 3.25 (a) and the output order in 4-pixel parallel follows Fig. 3.25 (b).



(a) (b)

Fig. 3.25 (a) decoding scanning order based on 8 x 8 block for each macroblock, (b) 4-pixel parallel output order for each 8 x 8 block

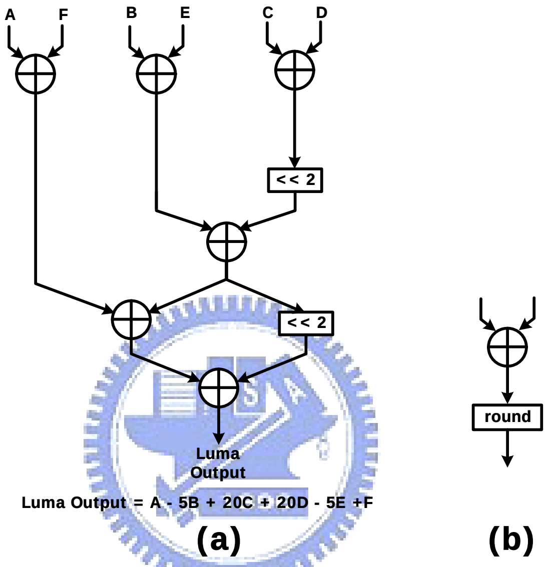


Fig. 3.26 (a) luma FIR design in Chen's [9], (b) bilinear filter

As for luma and chroma interpolator for H.264/AVC described in subsection 3.1.4 and 3.1.5, the adder also can be shared when the architecture of chroma horizontal/vertical filter in Fig. 3.19 (b) restructure to shifter and adder. The combined interpolator design is shown in Fig. 3.26 and the cost penalty is MUX x 2, shifter x 3 and bitwise AND x 6 when compared with the FIR design proposed in Chen's [9] and shown in Fig. 3.26 (a) . The decoding path of luma FIR filter and chroma horizontal/vertical filter are illustrated in Fig. 3.27. Because chroma interpolation for H.264/AVC is 2 x 2 block based process, only four luma FIR filters are required to replace with combined luma/chroma interpolators. Fig 3.27 indicates the shared storage and decoding module for chroma interpolator for H.264/AVC.

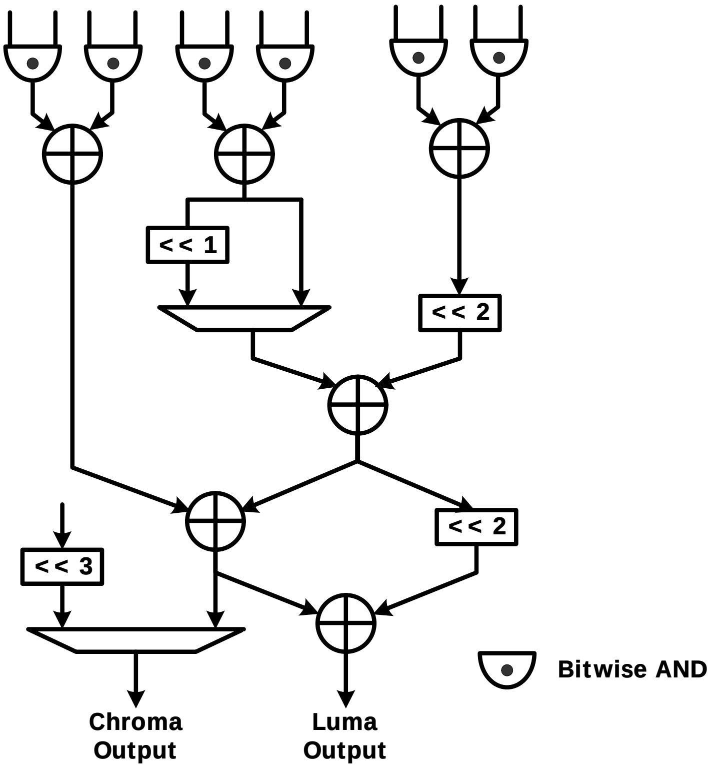
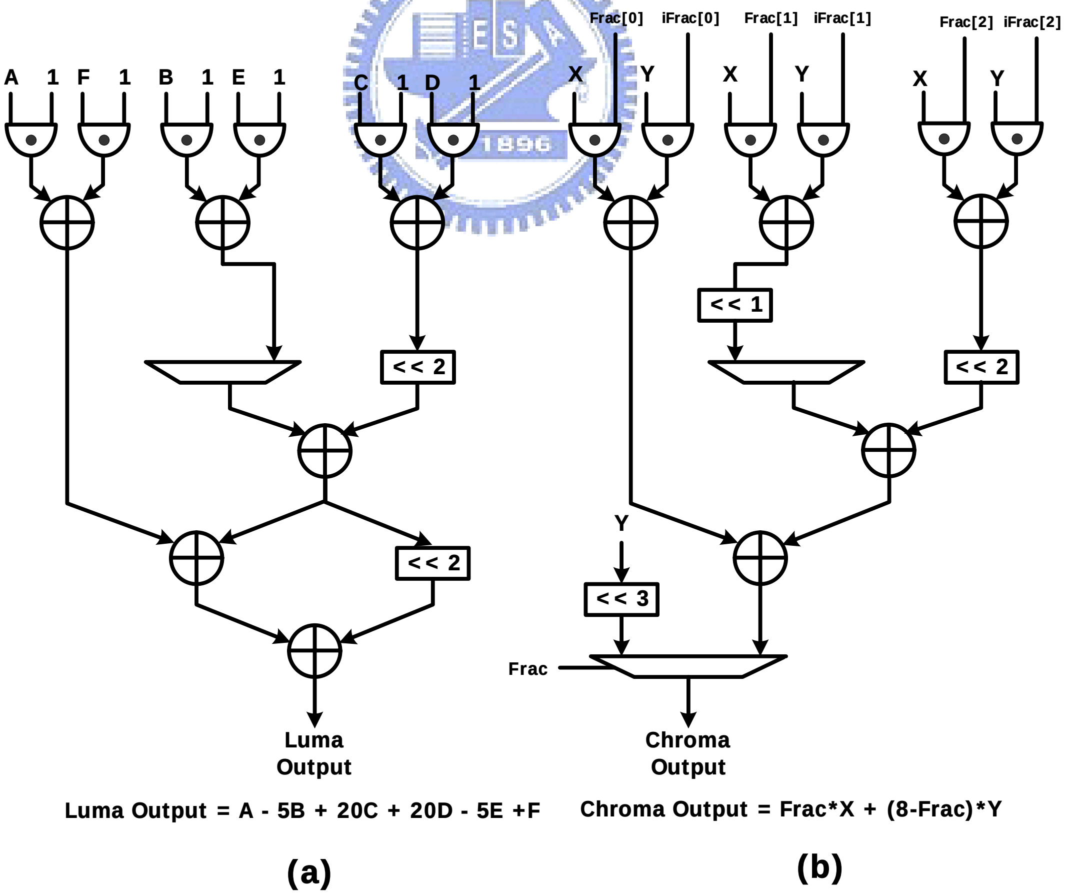


Fig. 3.26 Combined luma/chroma interpolator design for H.264



(a)

(b)

Fig. 3.27 (a) path of luma FIR interpolator, (b) path of chroma 1/8 bilinear

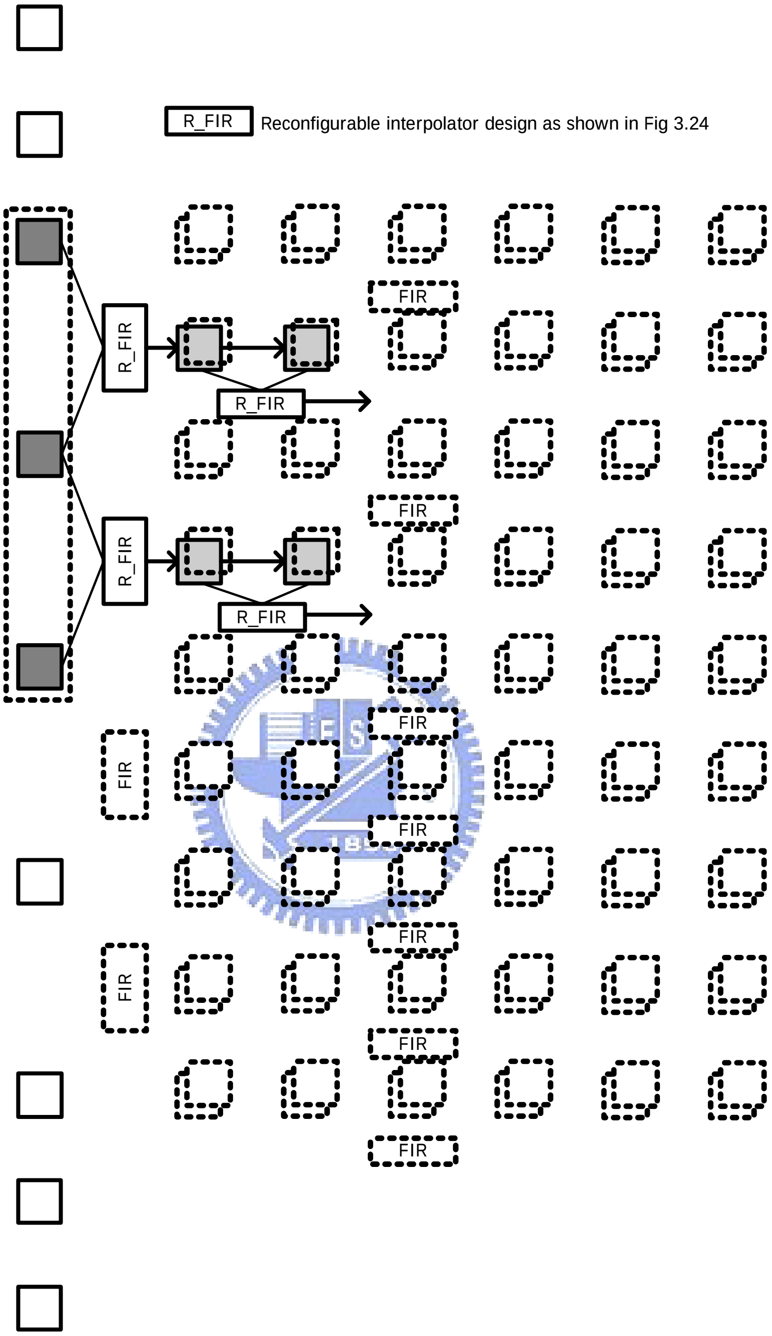


Fig. 3.27 Shared local register and reconfigurable interpolator for H.264 chroma interpolation

3.2.2 Cost Analysis

Table 3.4 # of adders for each filter design

Module	Function	# of adders
Horizontal/vertical FIR	1/2 luma for H.264	6
Bilinear (average)	1/4 luma for H.264 1/2 luma and chroma for MPEG-2	1
1/8 horizontal/vertical FIR	Chroma for H.264	5
Combined horizontal/vertical filter	1/2 luma for H.264 Chroma for H.264	6

Adder occupies the main area cost in the filter design. Firstly, Table 3.4 lists the number of adders used in each kind of filter design described in previous subsections. The horizontal/vertical FIR design presented in Chen's [9] and bilinear design are illustrated in Fig 3.25. Chroma 1/8 horizontal/vertical filter, which modifies the multiplier-based design depicted in Fig. 3.19 (b) to adder-based design painted in Fig. 3.26 (b), requires 5 adders. Table 3.5 lists the comparisons between our reconfigurable interpolator design and traditional design. It reveals that the amount of adder and register efficiently reduced although it requires paying some control circuits to support multi-mode operations. After synthesizing based on technology of UMC 0.18 um, the total area gate count can be reduced about 20 %.

Table 3.5 Comparison of requisite modules based on 4-parallel separate 1-D architecture
Traditional design: separated H.264 luma, H.264 chroma
and MPEG-2 bilinear interpolator

Module	Traditional design		Our Reconfigurable design	
Horizontal FIR	9		7	
Vertical FIR	4		2	
Bilinear (average)	8		4	
1/8 horizontal bilinear	2		0	
1/8 vertical bilinear	2		0	
Combined horizontal filter	0		2	
Combined vertical filter	0		2	
Content buffer	54 x 8 bits		54 x 8 bits	
Shift register array	(54 + 18 + 4) x 8 bits		54 x 8 bits	
Total	Adder	106	Adder	82
	Register (# of bits)	1040	Register (# of bits)	864
Gate count	16376		13013	

1896

3.3 Summary

In this chapter, a motion compensation engine for MPEG-2/H.264 dual-video decoder is presented. To overcome the tremendous data access to frame memories, especially in the high motion precision for the advanced video standard, H.264/AVC, the proposed data reuse technique for fractional motion compensation can efficiently reduce the requisite reference data. As for sharing design issue for multi-standard, our reconfigurable interpolator saves 20 % gate count compared with traditional design and it fully supports standard-compatible fractional interpolation for MPEG-2 /H.264 video decoder. Besides, the 4-parallel separate 1-D architecture is also suitable for high throughput SDTV/HDTV video decoder.

Chapter 4

Frame Memory Organization

To deal with tremendous data transfer and storage in multimedia system, software or hardware technologies must provide high data bandwidth and efficient real-time memory scheduling. As for video decoding system, irregular data access property and large storage of multidimensional organization always dominate the system performance including throughput and power consumption [12]. To flexibly support from mobile device up to high-definition TV, frame memories, which are the largest memory storage over the entire video decoder, are located on off-chip. Nevertheless, the data transfer to off-chip memory is always bound to the limited bandwidth. To improve memory bandwidth, new modern DRAM families such as synchronous DRAM (SDRAM), reduced latency DRAM (RLRAM) and double-data-rate SDRAM (DDR SDRAM) are now widely applied in video system [13]. In this chapter, we choose SDRAM as external frame memory.

Many SDRAM controllers have been proposed to improve memory bandwidth utilization and achieve efficient memory access. According to the environment, they can be categorized into two classes: single channel and multiple channel environments. For single channel environment, Rixner’s memory access scheduler [14] reorders the access addresses from each bank controller and sends command to DRAM through address arbiter. However, because the output command may be out-of-order, many command FIFOs and extra circuits are required to reorder commands and addresses. Miura‘s dynamic-SDRAM-mode-control scheme [15] eliminate the above disadvantage and it can both reduce operating current and the latency of an SDRAM. Nevertheless, it only supports scheduling of single-channel sequence. For multi-channel environment, Lee’s quality-aware memory controller [15] supports different scheduling policies according to the current channel situation. These memory controllers mainly focus on general-purposed environment. On the other way, concerning particular-purpose orientation especially in video codec application, several papers have been proposed on improvement of power consumption or memory bandwidth utilization. Kim’s memory–interface architecture [17] reorganizes data arrangement in SDRAM to reduce energy consumption and increase memory bandwidth. Park’s history-based memory mode control [18] reduces page miss to achieve 23.3 % reduced energy consumption and 18.8 % reduced memory latency. Zhu’s SDRAM controller in H.264 HDTV Decoder [20] focuses on memory mapping and data arrangement in SDRAM to reduce page active cycles; meanwhile, it also improves throughput and provides less power consumption. However, it does not provide memory scheduling and the adoption of auto precharge rather than manual precharge also leads to some loss of bus bandwidth. We will show the advantage of manual precharge in subsection 4.1.2. The above memory control techniques individually concentrate on memory scheduler or data arrangement in SDRAM. Both issues should be taken into account carefully in memory controller, especially for multi-dimensional oriented system, such as video codec and graphic processor unit (GPU). To achieve all-round integration, we consider both memory access scheduling and data arrangement to design our SDRAM controller. In addition, not only communication between SDRAM controller and data bus has to be analyzed, but also interface between motion compensation and SDRAM controller has to be taken into account carefully. The above discussion of related works is summarized in Table 4.1, and the application of our dual-channel SDRAM controller focuses on build-in video decoder. Section 4.1 will give detailed design for our dual-channel SDRAM controller.

Table 4.1 Related works of SDRAM memory controller

Related work	Application	Improvement	Techniques
Rixner's [14]	General-purpose Single-channel	Bandwidth, latency	Memory scheduling
Miura's [15]	32-bit RISC CPU Single-channel STB	Latency, Power	Memory mode control
Lee's [15] **CSVT'05**	Multimedia SoC Multiple-channel STB	Bandwidth utilization, Latency	Memory scheduling
Kim's [17] **CSVT'01**	Video Single-Channel build-in device	Memory Power Bandwidth	Data arrangement
Park's [18] **CE'03**	HDTV decoder Multi-channel STB	Memory Power, Bandwidth Latency	Memory mode control
Zhu's [20]	H.264 HDTV decoder Multi-channel	Bandwidth utilization, Decoding throughput	Data arrangement
Our work	Video decoder dual-channel build-in device	Bandwidth utilization, Latency Decoding throughput	Memory scheduling, Data arrangement

Frame memories always dominate the storage size on the video decoder. Generally speaking, at least two frame memories, which are used to store current and reference frames, are required for H.264@Baseline video decoder. Several methods have been proposed to reduce the required memory and they can be mainly classified to two solutions that one is frame recompression and another is frame memory reorganization. Concerning the first solution, which concept is depicted in Fig. 4.1, is recompressing video frame data before storing to frame memory, and equivalently decompression is required when reading reference frame data from frame memory. This recompression method must support random access capability demanded for motion compensation and low complexity property due to limited memory bandwidth. In this respect, many algorithms, such as Tajime's [22] 2-D adaptive DPCM in pixel domain, and Lee's [23] modified Hadamard transform and Golomb-Rice (GR) coding., etc have been proposed. However, frame recompression method leads to extra area cost and even requires additional execution cycles to compress data such that the throughput of video decoder degrades. As for second solution, frame memory reorganization, this idea, which combines the current frame and reference frame, can be initially found in De Greef's [24]. Besides, Interuniversity MicroElectronics Center (IMEC) widely exploited this idea to H.264 video decoder system [25], MPEG-4 motion estimation [26] and video encoder [27]. Particularly in Brockmeyer's [26] and Denolf's [27], the concept of memory hierarchy [28] combined with merging structured frame memory can achieve data reuse and reduce the redundancy of data access. However, they only focus on C level simulation and target on DSP or FPGA platform. If we want to implement on ASIC design, many issues still have to be overcame. For example, the data copy and update between background memory and intermediate memory are required being considered in ASIC design. Concerning another issue, we also need extra hardware cost to record the update status in in-place FIFO [25], the intermediate region between the new frame and old frame. For advanced development, Chang's combined frame memory architecture [29] can reduce frame memory size up to 57 %

and reduce up to 83 % average latency and 39 % average power consumption. We will discuss the methodology proposed by Chang's [29] and exploit it on H.264/AVC video decoder.

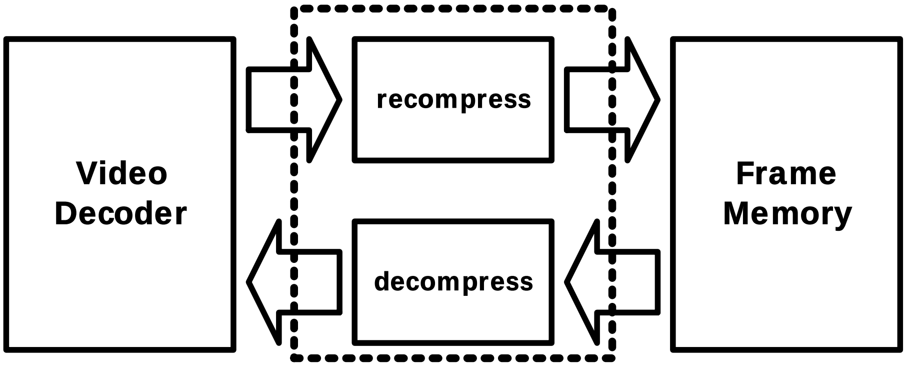


Fig. 4.1 Frame recompression method

The reset of this chapter is organized as follows. Firstly, SDRAM characteristic is described in section 4.1. Then, section 4.2 discusses our dual-channel frame memory access controller design. In addition, merging structured frame memory organization, a novel memory structure that can reduce required frame memory size, is presented in section 4.3. Finally, summary is given in section 4.4.

4.1 SDRAM characteristic

4.1.1 Basics

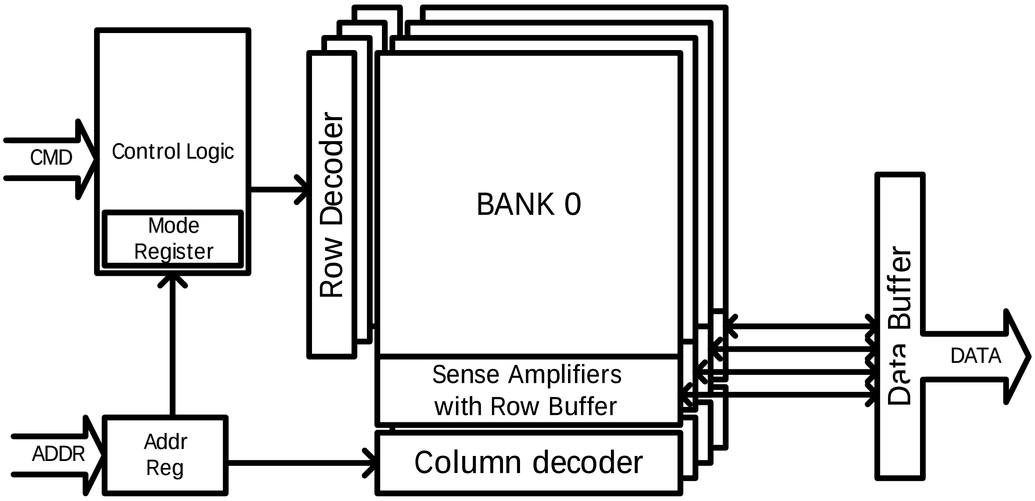


Fig. 4.2 Simplified architecture of a 4-bank SDRAM

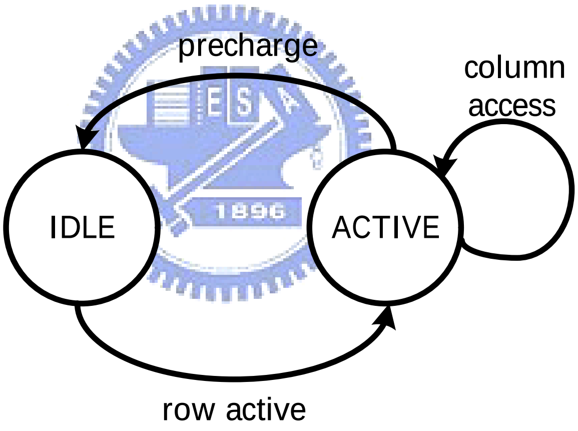


Fig 4.3 Simplified bank state diagram

A simplified architecture of a 4-bank SDRAM is shown in Fig. 4.2. Four banks share the address bus and command bus, while each bank has individual row decoder, sense amplifier, and column decoder. The mode register stores several SDRAM operation modes, including burst length (BL), column address strobe (CAS) latency (abbreviated as CL), or burst type (sequential / interleave). The content of mode register updates according to command issued from address bus. SDRAM can be treated as 3-D structure with the dimensions of bank, row, and column. A memory access operation, which simplified bank state diagram is depicted in

Fig 4.3, contains three steps including row activation, column access, and precharge. Firstly, a row activation command with bank address is sent to open (or active) one row in a particular bank and the designated row address is issued from address bus. The operation of this command is copying the row data into the row buffer of the selected bank and row activation needs a active latency called t_{RCD} (ACTIVE to READ or WRITE delay) to accomplish this operation. Then, column access command is used to sequential access data or single data according to the burst length and burst type in the mode register. The read/write data are access/send thorough 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 Fig. 4.4. For write operation, the first valid data-in element is coincident with the WRITE command, as shown in Fig. 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_{RP} (PRECHARGE command period.)

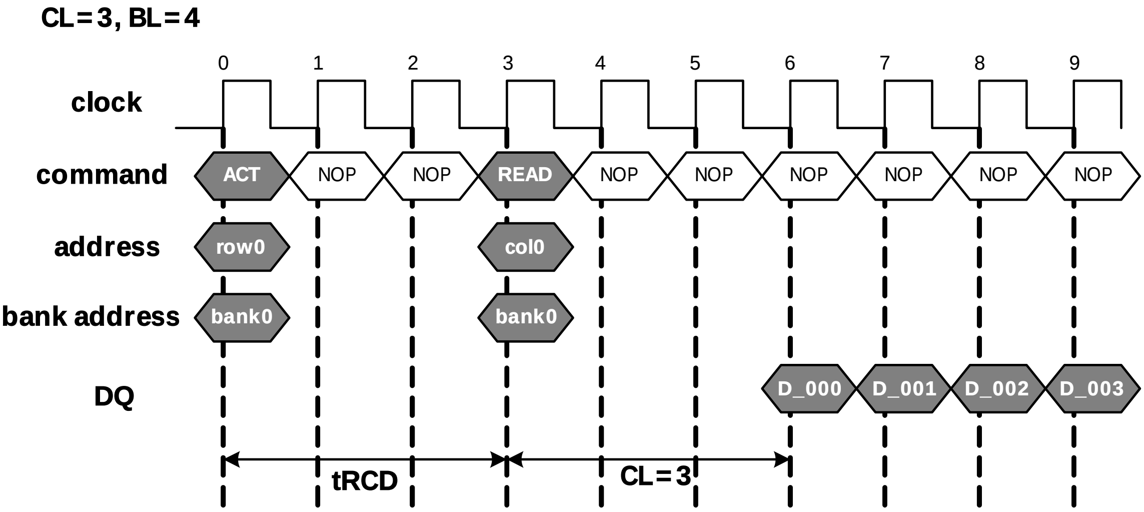


Fig. 4.4 Burst read operation with CL=3 and BL=4.

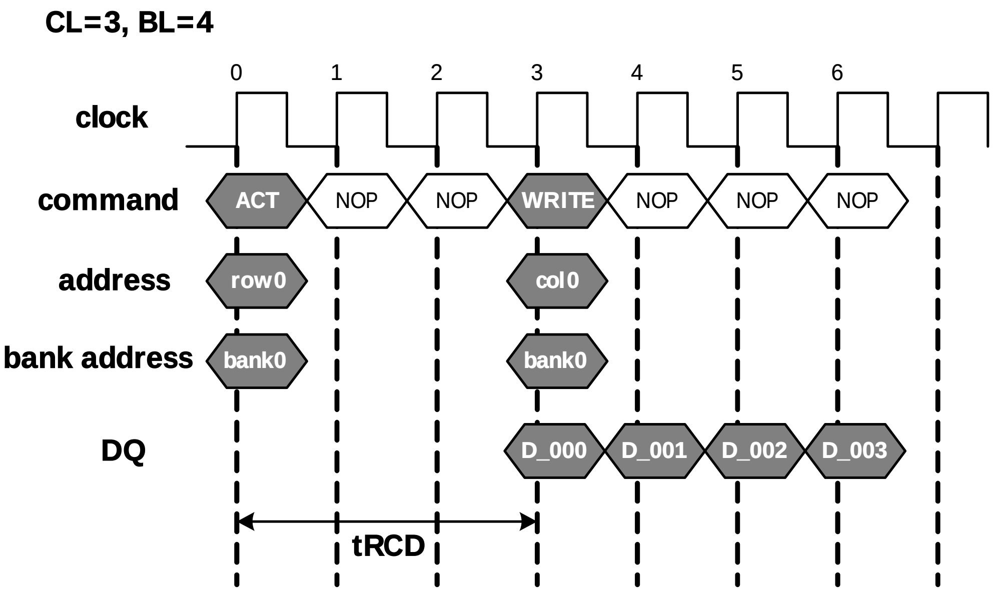


Fig. 4.5 Burst write operation with CL=3 and BL=4.

Table 4.2 CAS latency

CL	1	2	3
Allowable operating frequency (MHZ)	≦50	≦100	≦166

4.1.2 Access Latency

Lee discussed different access latencies of different access statuses in [15]; however, detailed classification is required for exquisite access command scheduling. The memory behavior model used in our design is Micron's MT48LC2M32B2P-5 64Mb SDRAM [21]. Table 4.1 lists three different allowable maximum operation frequencies provided in this SDRAM according to the CAS latency stored in mode register. Obviously, when setting CAS latency to 3, the SDRAM can provide higher operating frequency. However, higher operating frequency induces more stall cycles is demanded for each read column access. Therefore, the CAS latency should be set carefully for different applications. For example, 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.

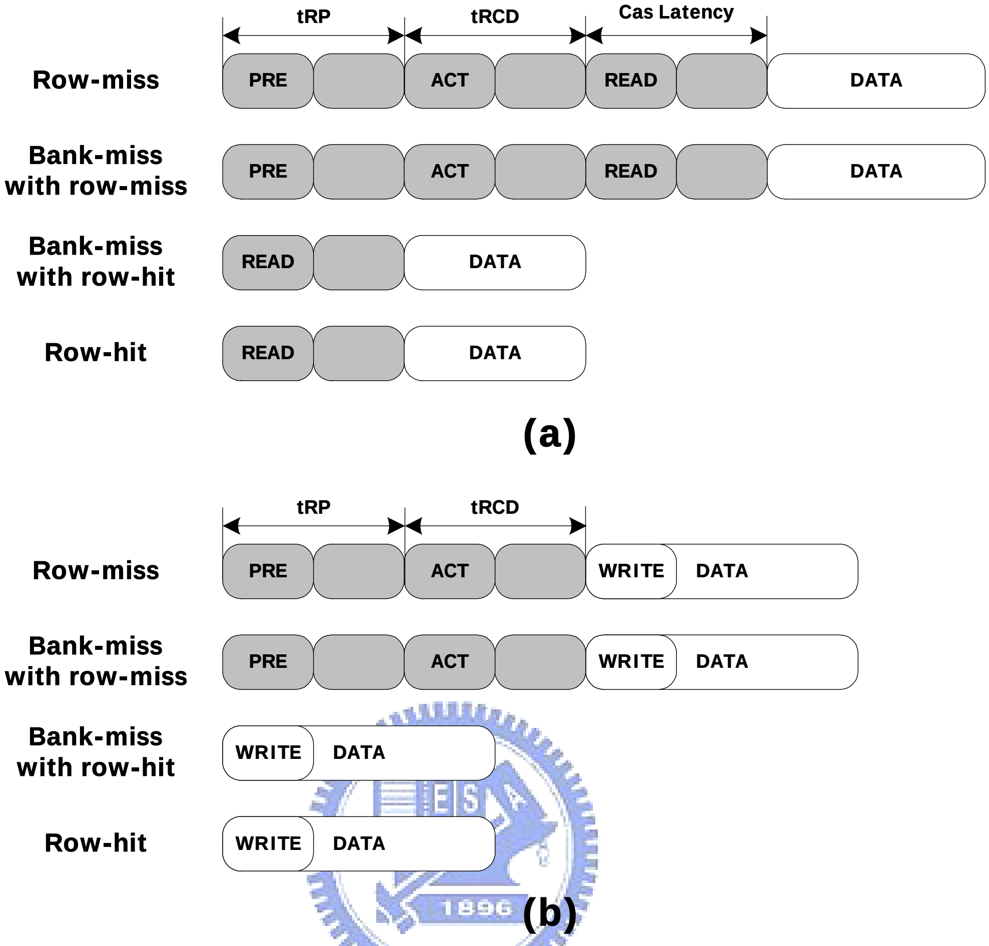
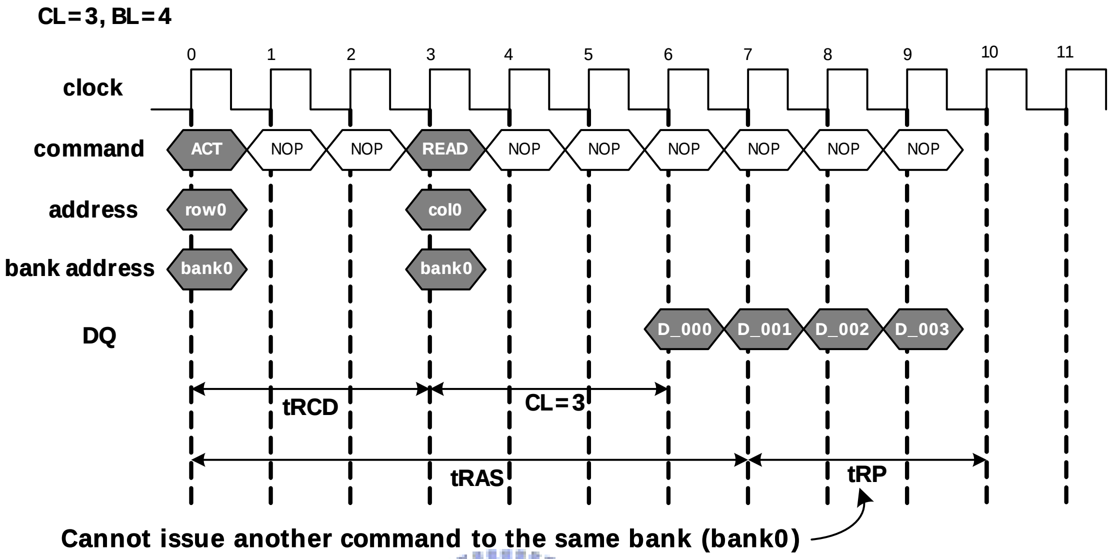


Fig. 4.6 Access latency for CL=2 (a) read access latency, (b) write access latency

Fig. 4.6 illustrates read/write access latency under different statuses when CL =2. Row miss status means that the activated row in selected bank is not identical to the incoming access command and it induces (PRECHARGE + ACTIVE + CAS) latency for read access and (PRECHARGE + ACTIVE) latency for write access. Bank-miss with row-miss status means that incoming bank address is different from previous command and the selected row for the incoming bank address is not activated. For this status, required latency 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. For this status and row-hit status, the column access can be directly issued and only read access leads to CAS latency. Based on the above discussion, memory scheduling can overlap the sequential access commands and hide full or partial latencies.



Cannot issue another command to the same bank (bank0)

Fig. 4.7 READ command with auto precharge

In the precharge period (tRP), SDRAM cannot issue

another command in the same bank (bank 0).

SDRAM also supports another precharge method called auto precharge without requiring an explicit precharge command. A PRECHARGE of bank/row together with READ/WRITE command is automatically performed upon completion of READ/WRITE burst, except in the full-page burst mode, where auto precharge does not apply. Auto precharge ensures that the precharge is initiated at the earliest valid stage within a burst. As shown in Fig. 4.7, in the precharge period, it cannot issue another command to the same bank until the precharge time (tRP) 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 tRP period or reorder with the other command. For another disadvantage induced by auto precharge, READ/WRITE with auto precharge means that

SDRAM always de-active the selected bank at the end of a burst command. If the following command still issues the same bank, it must waste time to re-active the same bank and lead to longer latency at the same time. Therefore, we select manual precharge rather than auto precharge in our memory access controller design.

4.2 Dual Channel Frame Memory Access Controller

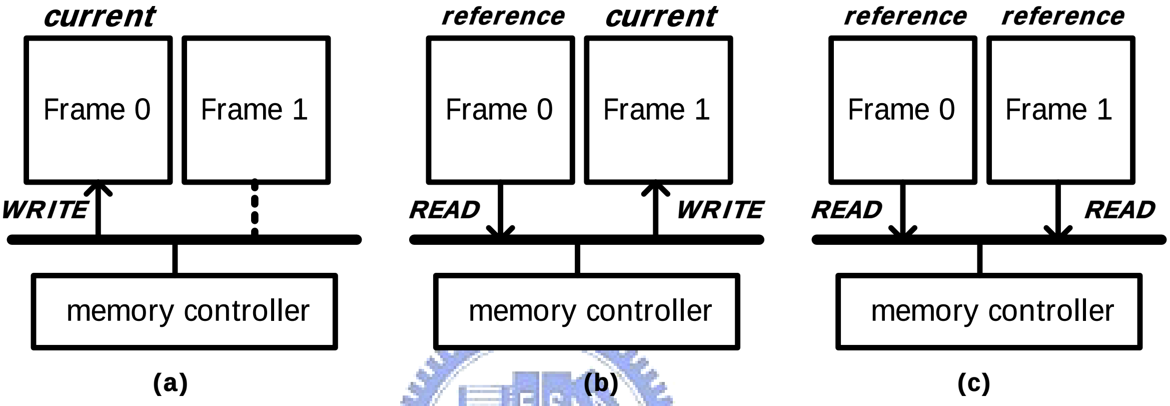


Fig. 4.8 READ/WRITE operation in (a) I frame, (b) P frame, (c) B frame.

For frame memory access in video decoding system, we only need to concentrate on consecutive read or write access instead of read-to-write or write-to-read access because read/write operation changes at frame level. Based on conventional ping-pong structured frame memories [28], which one stores reference frame and another stores current frame, Fig. 4.8 shows read/write operation in three different frame types. For I frame, memory access controller write reconstructed data to current frame. For P frame, memory access controller reads referenced data while writing reconstructed data to current frame. For B frame in MPEG-2, memory access controller reads data from previous and following reference frame because B frame is never referenced. Nevertheless, B frame has a chance that referenced by other P/B frame for H.264/AVC video decoder, so the frame memories issue becomes more complicated. We can set nal_ref_idc flag in H.264/AVC such that B frame is never used to be reference frame. In this section, we only focus on I/P frame for H.264 and I/P frame for MPEG-2 video decoder.

Table 4.3 Characteristics of read/write-access

Access	Required density	Influence factors
Read	High	Bitstream, memory scheduling, data arrangement in memory
Write	Low or medium	Bitstream, capability of residual decoder, (de-blocking filter only for H.264)

4.2.1 Memory Access Scheduling

The target of memory access scheduling is overlapping or reordering consecutive DRAM commands (PRECHARGE, ACTIVE, CAS) to improve bandwidth utilization and reduce access latency. Because the external access of video decoder is bandwidth-sensitive channel [15], memory access scheduler must compress and even reorder DRAM commands to achieve high bandwidth utilization. Furthermore, considering read-access and write-access respectively, the required density of write-access has high correlated with the ability of residual decoder and the property of decoding bitstream, while the required density 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 necessarily 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.

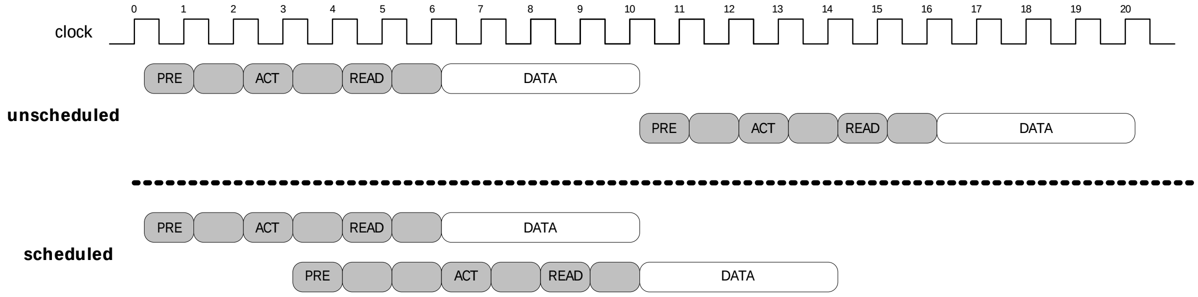


Fig. 4.9 Two row-miss unscheduled and scheduled read memory accesses (CL=2, BL=4)

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. Fig. 4.9 shows an example of two unscheduled and scheduled read memory accesses when occurring row miss at different bank. For 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 to two types, one is long command (PRE + ACT + CAS) and another is short command (CAS), painted in Fig. 4.5. Moreover, we consider the latency after access scheduling under BL=1, 2, 4 situations illustrated in Fig. 4.10-12 and summaries the induced latency under each situation in Table 4.4. 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.

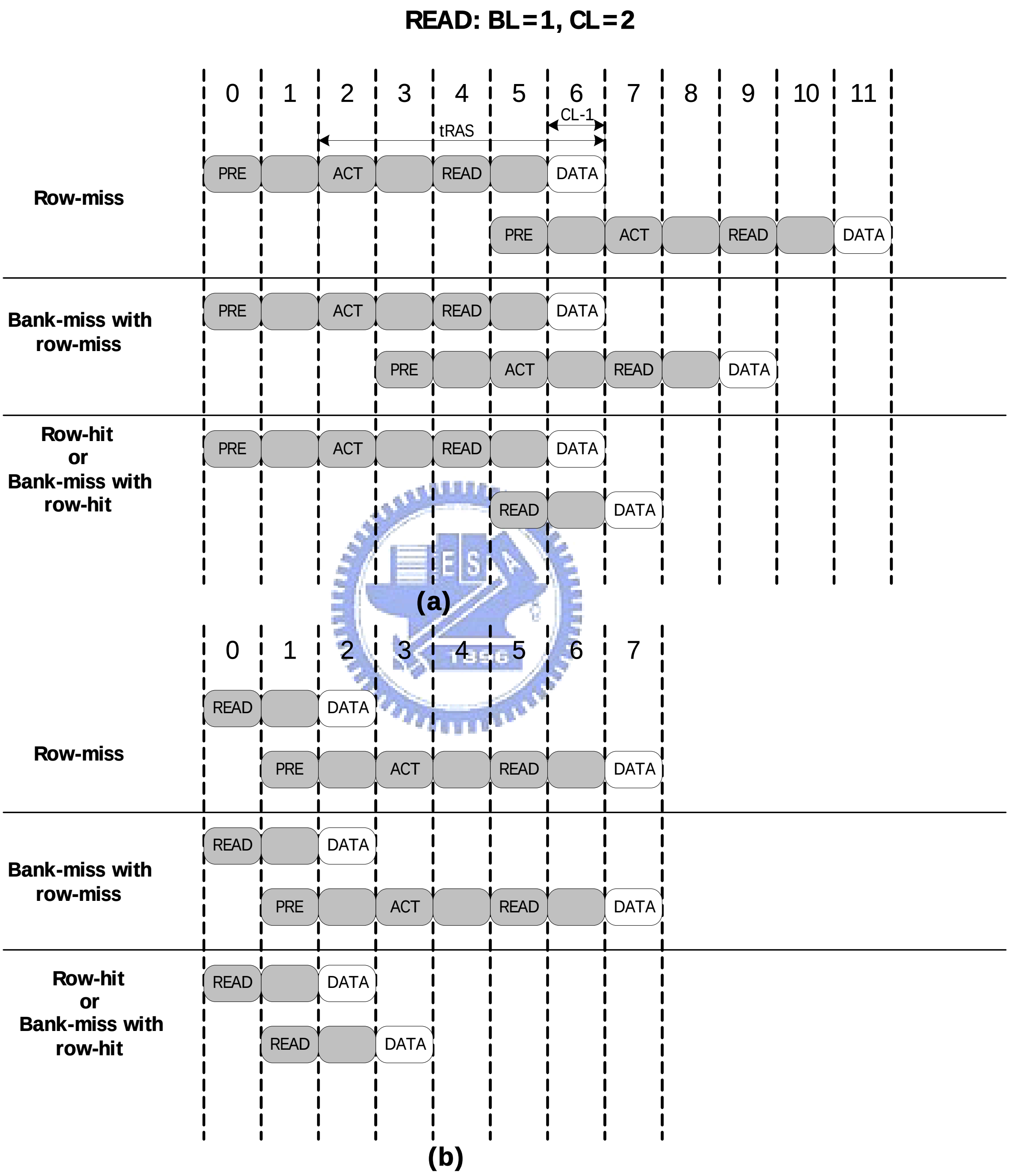


Fig. 4.10 Scheduled consecutive read access for (BL=1, CL=2) when previous command is (a) long command (PRE+ACT+CAS), (b) short command (CAS)

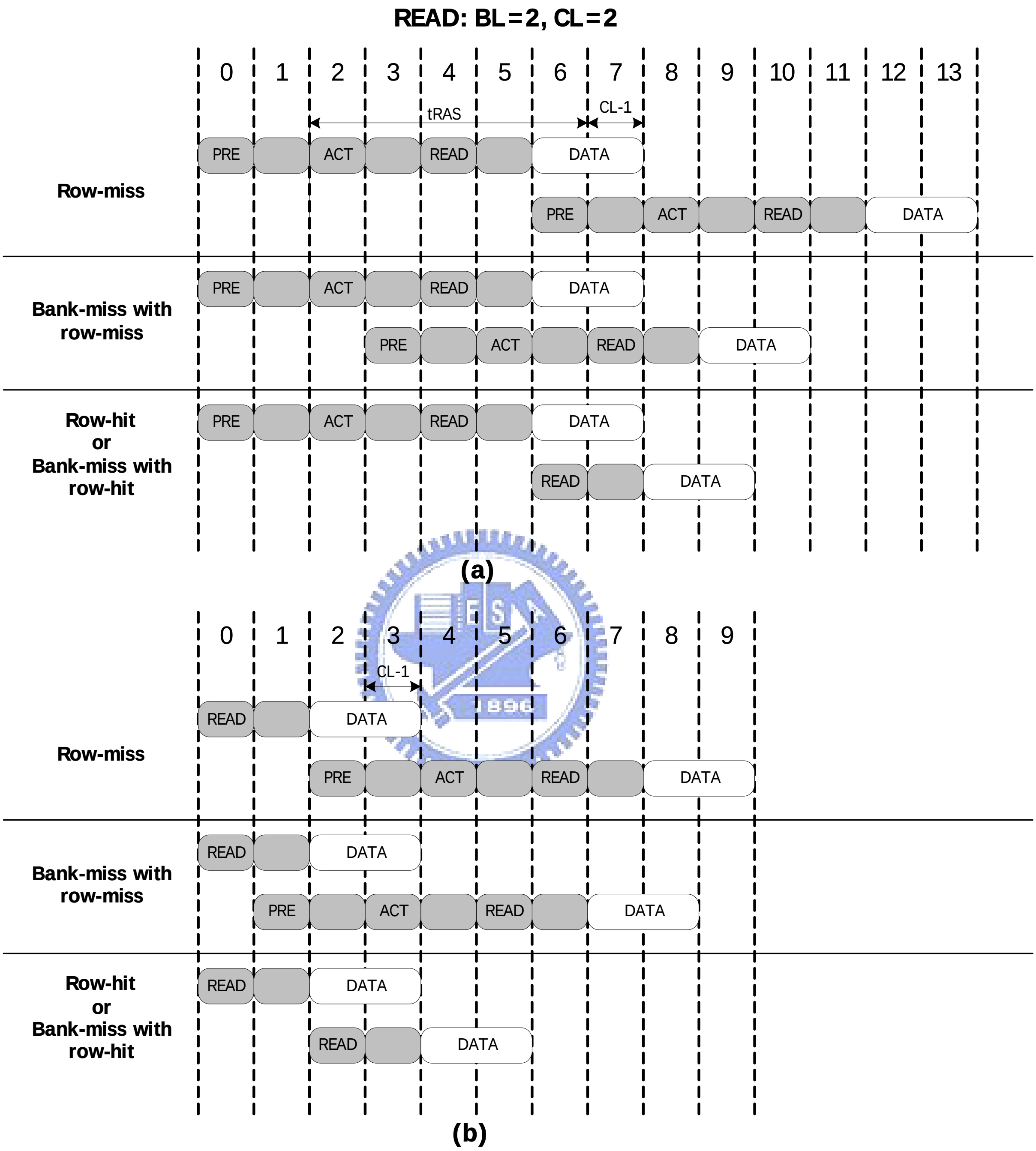


Fig. 4.11 Scheduled consecutive read access for (BL=2, CL=2) when previous command is (a) long command (PRE+ACT+CAS), (b) short command (CAS)

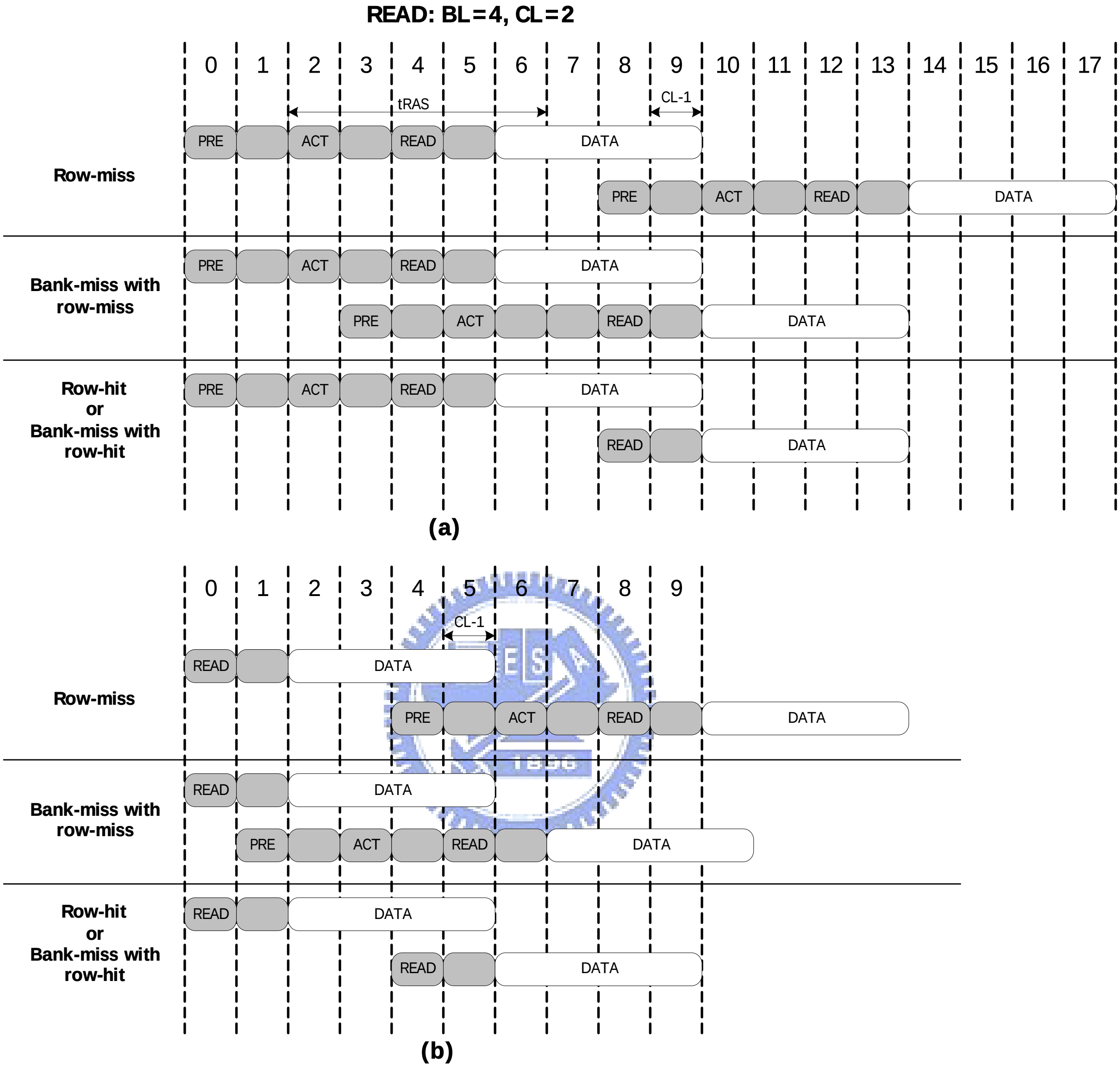


Fig. 4.12 Scheduled consecutive read access for (BL=4, CL=2) when previous command is (a) long command (PRE+ACT+CAS), (b) short command (CAS)

Table 4.4 Latency for scheduled consecutive read access when CL=2

BL	Previous command	Incoming command	Latency
1	**(PRE + ACT +CAS)**	Row-miss	4
		Bank-miss with row-miss	2
		Row-hit or Bank-miss with row-hit	0
	CAS	Row-miss	4
		Bank-miss with row-miss	4
		Row-hit or Bank-miss with row-hit	0
2	**(PRE + ACT +CAS)**	Row-miss	4
		Bank-miss with row-miss	1
		Row-hit or Bank-miss with row-hit	0
	CAS	Row-miss	4
		Bank-miss with row-miss	3
		Row-hit or Bank-miss with row-hit	0
4	**(PRE + ACT +CAS)**	Row-miss	4
		Bank-miss with row-miss	0
		Row-hit or Bank-miss with row-hit	0
	CAS	Row-miss	4
		Bank-miss with row-miss	1
		Row-hit or Bank-miss with row-hit	0

Out-of-order scheduling may cause two problems for video decoding system. The first one is that the additional FIFOs is required storing incoming access commands, and the second one is that additional read data buffer and control circuits are needed to re-order data, otherwise the disordered data leads to operation error occurred in motion compensation. To eliminate the above problems, in-order, also named first-in-first-out, policy is adopted in our priority scheduling. We just focus on command overlapping and data arrangement such that our design can meet real-time SDTV/HDTV decoding at 100 HMZ that is the allowable maximum operating frequency when CL = 2.

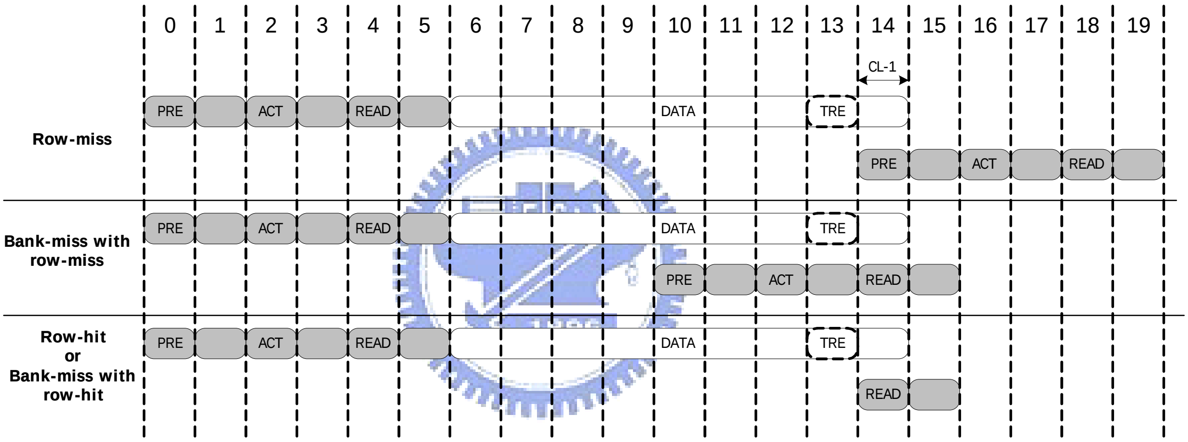


Fig. 4.13 Scheduled consecutive read access for full-page mode

Besides the fixed length burst mode, SDRAM also supports full-page mode that allows the access of arbitrary length successive data on the particular row/bank. The full-page access must be truncated by BURST TERMINATE command. Fig. 4.13 shows an example of scheduled consecutive read access for full-page mode. The BURST TERMINATE command should send before (CL-1) cycles apart from the last burst data. When a READ or WRITE command is issued, a block of columns equal to the burst length is effectively selected. All accesses for that burst take place within this block, meaning that the burst will wrap within the block if a boundary is reached.

4.2.2 Data Arrangement

From the above discussion, the data arrangement in SDRAM should tend to 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. Fig. 4.14 (a) illustrates that the luma MB partitioning is dispersed to four banks. The first MB addressed in SDRAM is painted in Fig. 4.14 (b). Similar to luma MB, the chroma block is partitioned to two banks. As shown in Fig. 4.15, the Cb block is placed in bank 0 and 1, while Cr block is located in bank 2 and 3. Fig. 4.16 illustrates the QCIF data arrangement in SDRAM. Each Frame can be partitioned into several MB based row. The length is frame width and the width is MB width. When frame size is small, each row (page) of SDRAM can stores 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 size format. For another reason, the probability of row-miss occurrence is very low. Obviously, it only occurs when data is located in row (page) boundary. As for physical analysis, we will show it later.

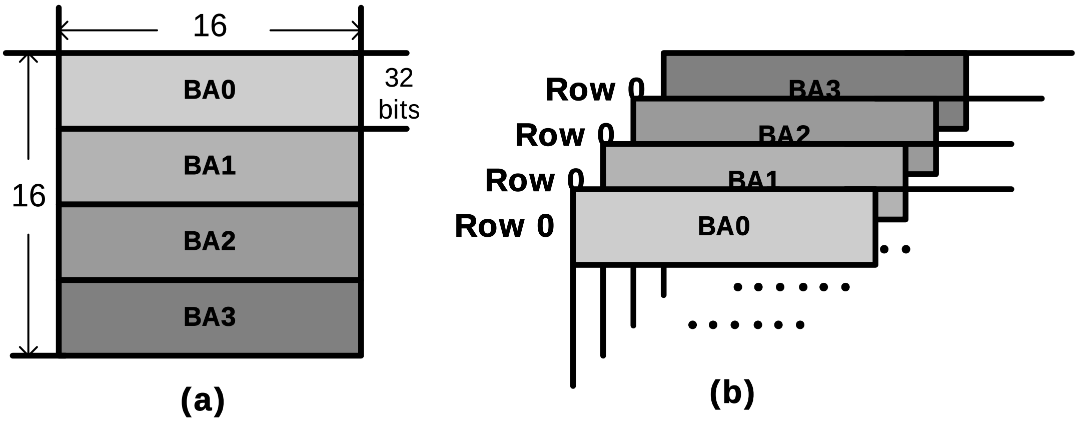


Fig. 4.14 (a) row-major arrangement,

(b) the first luma macroblock location in SDRAM

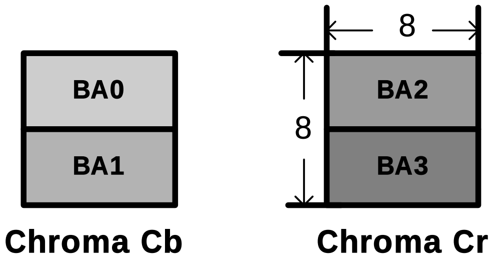
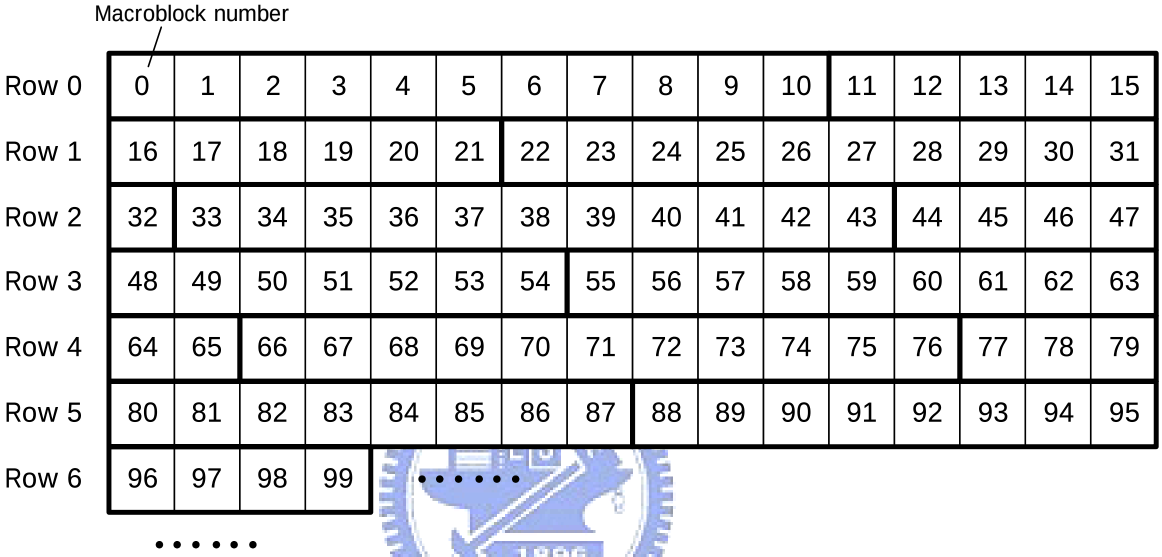


Fig. 4.15 The location of SDRAM for chroma Cb and Cr



Row 0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Row 1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Row 2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Row 3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Row 4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
Row 5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
Row 6	96	97	98	99	……											

……

Fig. 4.16 QCIF video format arrangement in SDRAM

Because the fetching window width/length is always larger than data length of SDRAM, the fetching window is truncated by row boundary or different banks. For example, considering luma MB, it required to fetch 9 x 9 interpolation window no matter what 4 x 4 block for H.264/AVC or 8 x 8 block for MPEG-2. Several cases are illustrated in Fig. 4.17. Window A and B are only truncated by different banks at the different row. That is, only bank-miss with row-hit or row-hit occurs in this window and the penalty of latency is small or zero from Table 4.4. With regard to window C, bank-miss with row-miss would take place and the required latency is greater than that of window A and B. Row-hit, which requires maximum latency, only occurs in row boundary such as window D. As for integer MV for H.264/AVC, it only need to read 4 x 4 block like window E and F. Window E is the best case since all block is addressed in the same bank/row. Window F is truncated by different bank at the same row.

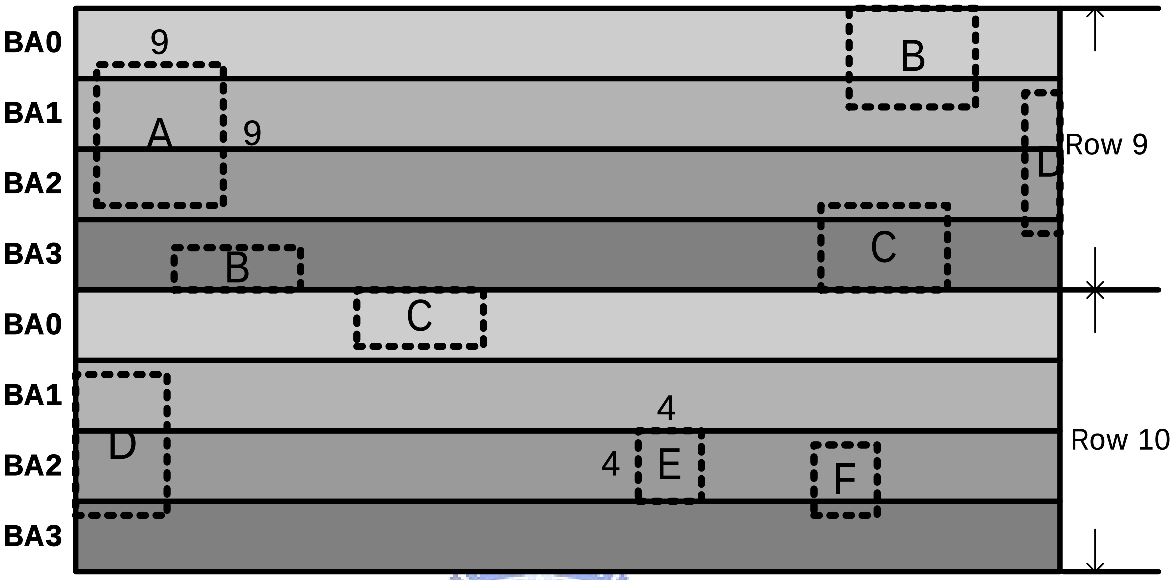


Fig. 4.17 Different cases of the interpolation window location when window size is 9 x 9 pixels (A, B) truncation by banks at the same row, (C) truncation by banks at different row, (D) truncation by row and banks, or window size is 4 x 4 pixels (E) best case, no truncation, (F) truncation by banks at the same row

Table 4.5 The proportion of access status (foreman-QCIF @ 64kbps in H.264 decoder)

BL	Row-hit (%)	Bank-miss with row-hit (%)	Bank-miss with row-miss (%)	Row-miss (%)
1	37.9152	60.2273	1.51639	0.34104
2	38.2509	58.5636	2.60708	0.578438
4	39.2019	55.9049	4.01952	0.873668
Full-page	43.3516	47.9131	7.07336	1.66188

As listed in Table 4.5, after simulation on foreman-QCIF format @ 64kbps, we can find the proportion of row-miss status only occupies less than 2 % at any burst length access. The row-major arrangement for frame memory can efficiently reduce the probability of bank-miss with row-miss or row-miss that contributes the maximum latency. More detailed experiment result will illustrate in subsection 4.2.4.

4.2.3 Dual-channel Frame Memory access controller Design

The entire dual channel SDRAM memory access controller and the connection with motion compensation are painted in Fig. 4.18. Regarding write memory access controller, write data buffer is used to hold the length of burst data, while write address queue is used to hold incoming addresses. Write address is calculated according to the output ordering of de-blocking-filter (H.264) or IDCT (MPEG-2). Master write scheduler assigns the incoming address command to suitable write controller according to the access status. Timing unit records all kinds of command latency such as burst length, tRP (precharge period), tRCD (ACTIVE to READ or WRITE), and so on. Write controllers generate sequential access commands according to the burst length and latency defined in timing unit. These access commands are collected by master write scheduler, which can issue the proper command to SDRAM. Read memory access controller has the similar operation to the write memory access controller. Read data buffer is used to hold sequential received read data for motion compensation decoding. The arbiter allocates write / read data and command flow to / from external SDRAM memories according to the frame type illustrated in Fig. 4.8.

As for motion compensation engine, besides interpolator and motion vector generator described in Chapter 3, the major FSM controller consists of three parts: request FSM, receive FSM and output stage FSM. Request FSM send request and address to memory access controller when the status of read addr queue is not full. The detailed design of wirte/read addr queue is painted in Fig. 4.19. 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. Receive FSM controls the mask to filter the read data and then spits to interpolator. Output stage FSM handles the interpolator and output data flow. When the fractional part of motion vector is zero, read reference data directly output; otherwise, the reference data pass through interpolator and generates interpolated sample data.

Unlike traditional SDRAM access controller design containing various “WAIT” states, Lee’s [15] 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 Fig. 4.20. Each bank requires two access FSMs to overlap two successive row-miss (at the same bank) accesses; otherwise the incoming row-miss command has to wait until the previous access command returns to IDLE state. As for bank-miss (at the same row or not) situations, access 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 (RA) register to record the activated row for each bank. Compared incoming commands with RA registers for each bank controller, the bank-miss with row-hit or bank-miss with row-miss status can be detected. The access scheduler allocates and overlaps successive commands according to these status flags. In brief, double access FSMs for individual bank controller can handle access conflict at the same bank, while master access scheduler is responsible for access overlapping between different banks. After scheduling SDRAM access commands, the bus utilization can raise efficiently; meanwhile the throughput of the entire video decoder can be improved.

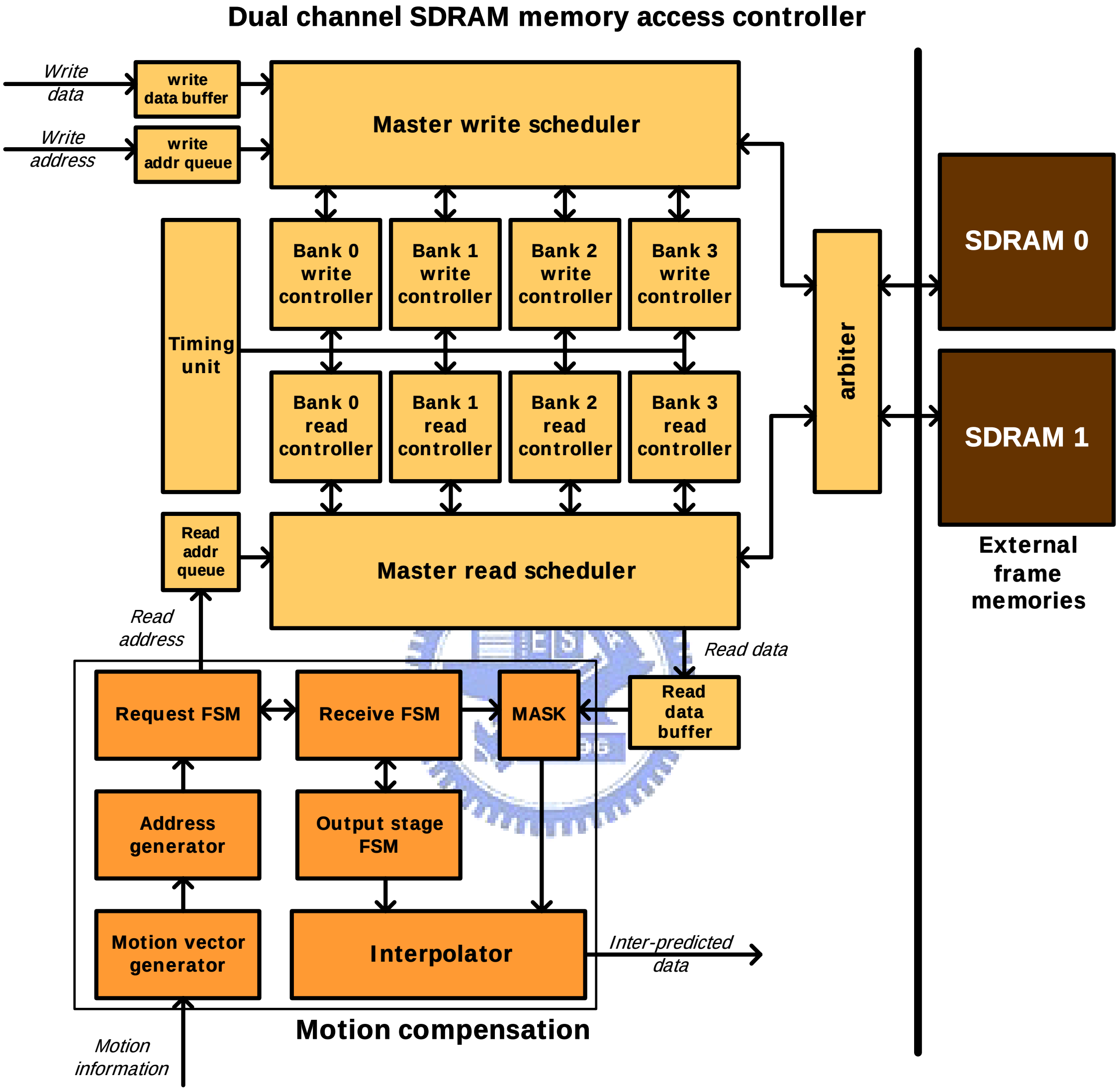


Fig 4.18 Dual-channel SDRAM memory access controller design

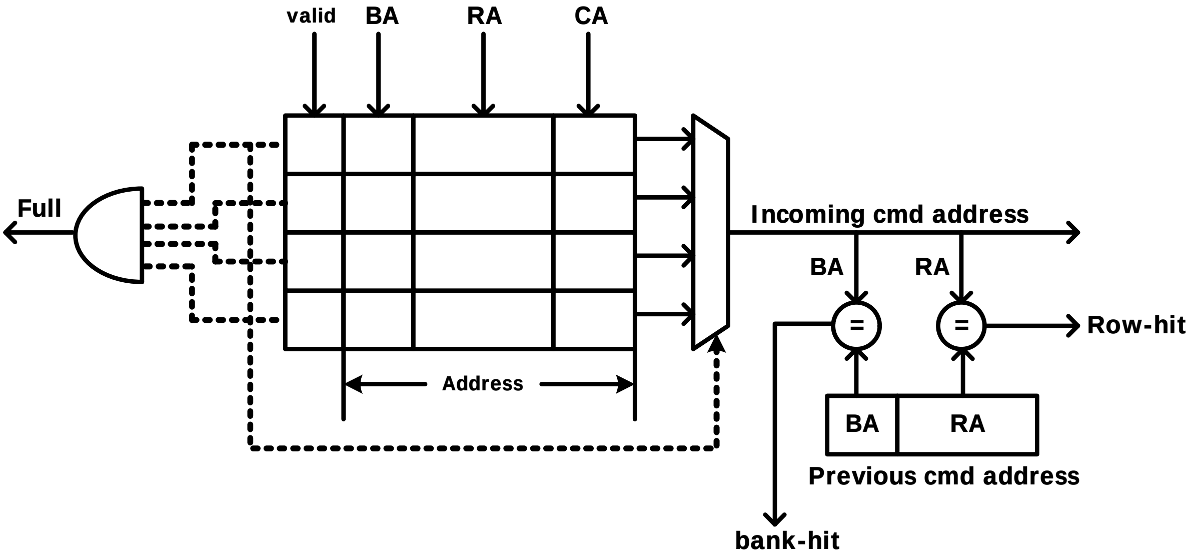


Fig. 4.19 Incoming address command queue

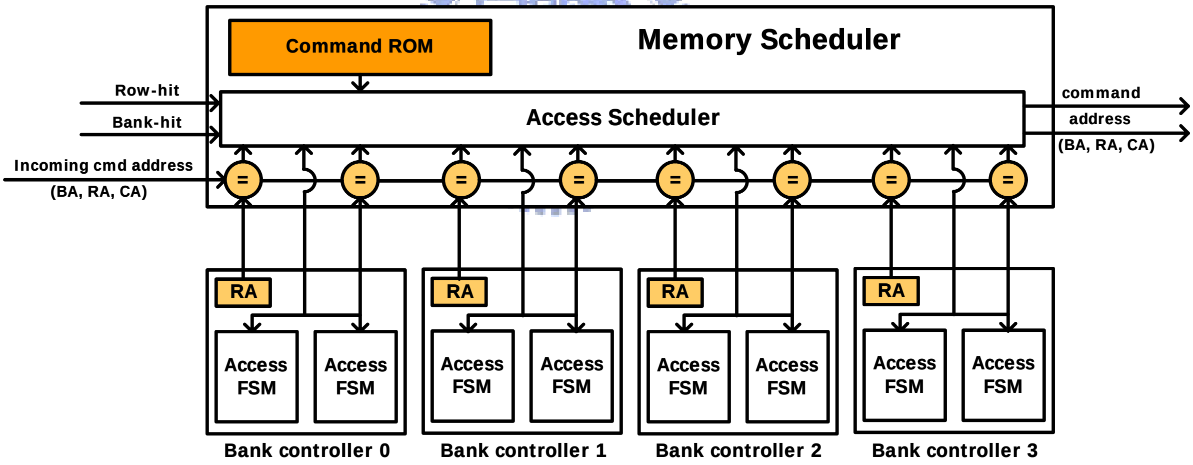


Fig. 4.20 Memory bank controllers and schedulers

4.2.4 Experiment Result

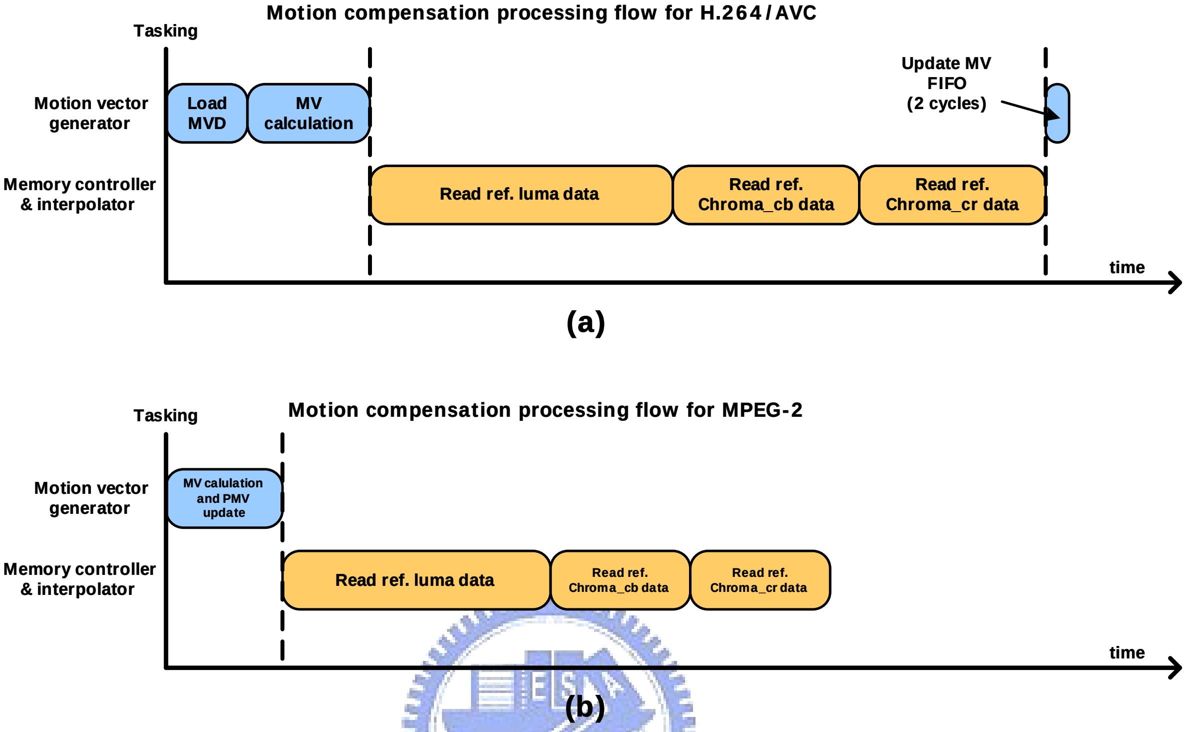


Fig. 4.21 Motion compensation processing flow for (a) H.264/AVC, (b) MPEG-2

Firstly, review the motion compensation processing flow depicted Fig. 4.21 for MPEG-2 and H.264/AVC video decoder. For H.264/AVC video decoding, motion vector generator needs two-phase operations including loading MVD and calculation of MV. After reading reference luma/chroma data and interpolation processing, motion vector generator takes two cycles to update MV FIFO for the next MB decoding. As for MPEG-2 tasking, motion vector generator can immediately calculate MV when receiving required information like motion_code, f_code, and motion_residual and then read reference data to reconstruct current frame. Then, considering system level analysis including motion compensation, SDRAM controller, depicted in Fig. 4.22, because motion compensation and SDRAM controller are both in operation and data transmission only during the period of reading reference data, hence we only have to analysis the data transfer in this period.

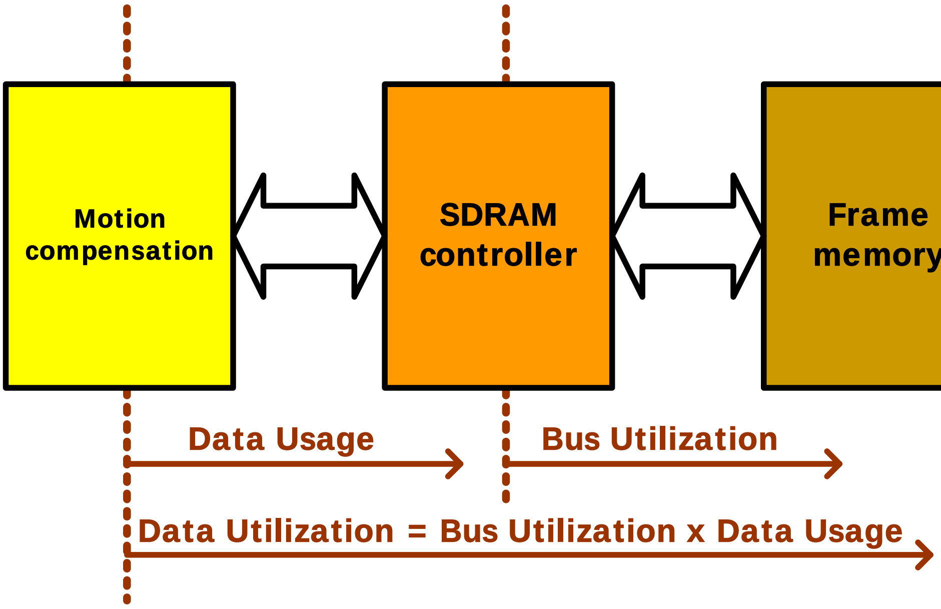


Fig. 4.22 System level analysis

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

$$\text{Bus Utilization} = \frac{\frac{\text{\# of bus cycles required by SDRAM controller}}{\text{4x4 sub block}} \times \frac{\text{\# of 4x4 sub block}}{\text{frame}} \times \frac{\text{\# of frame}}{\text{sec}}}{\frac{\text{\# of bus cycles available}}{\text{4x4 sub block}} \times \frac{\text{\# of 4x4 sub block}}{\text{frame}} \times \frac{\text{\# of frame}}{\text{sec}}} \tag{4. 1}$$

$$\text{Data Usage} = \frac{\frac{\text{\# of data required by MC}}{\text{4x4 sub block}} \times \frac{\text{\# of 4x4 sub block}}{\text{frame}} \times \frac{\text{\# of frame}}{\text{sec}}}{\frac{\text{\# of data available from SDRAM controller}}{\text{4x4 sub block}} \times \frac{\text{\# of 4x4 sub block}}{\text{frame}} \times \frac{\text{\# of frame}}{\text{sec}}} \tag{4. 2}$$

$$\text{Data Utilization} = \text{Bus Utilization} \times \text{Data Usage} \tag{4. 3}$$

Based on the assumption of that the data bus is only provided for double frame memories, generally speaking, higher bus utilization induces better throughput for our video decoder. The data usage is correlated to the burst length and required window size. Thus, data usage can be treated as the proportion of required data over the available data from SDRAM controller. To explain data usage clearly, considering 9 x 9 interpolation window of 4 x 4 block in H.264 fractional motion compensation, Fig. 4.23 illustrates an example of the fetching window for four different burst lengths. Fetching window is the total pixels that needed to be read for SDRAM controller. Since the data bus width is 4-pixel (32 bits), the height of fetching window must be 12-pixel that is a multiple of 4-pixel. Similarly, the width of fetching window must be the multiple number of the burst length except that the length can be arbitrary length for full-page mode. Accordingly, among these burst mode, the data usage is poorest when the selected burst length is 4. Data utilization is the multiplication of bus utilization and data usage. Therefore, the data utilization can be seemed as the required data proportion over the allowable data transmission on the bus. Higher data utilization means that we have chance to get better throughput and less latency on the entire video decoder.

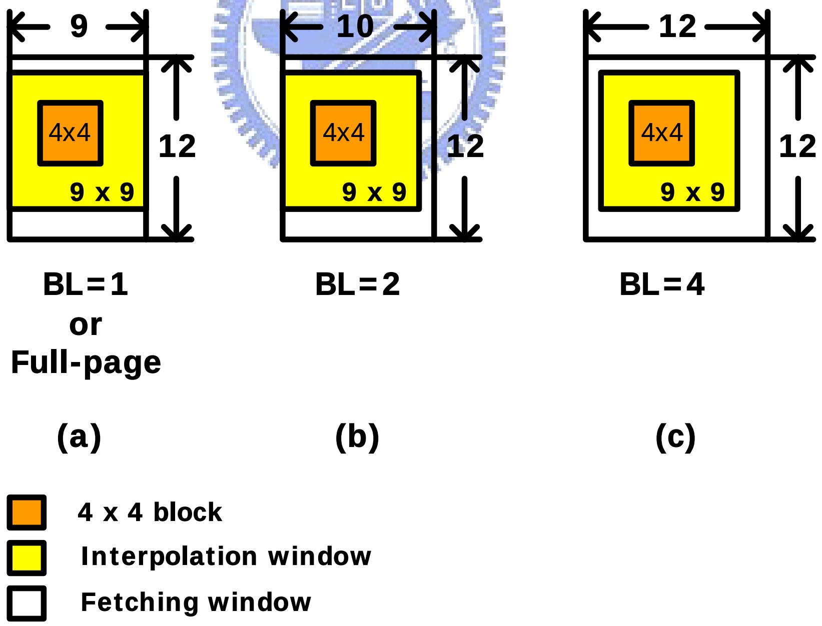


Fig. 4.23 Fetching window of H.264 4 x 4 block for different burst length (unit: pixel): (a) BL=1 or full-page, (b) BL=2, (c) BL=4

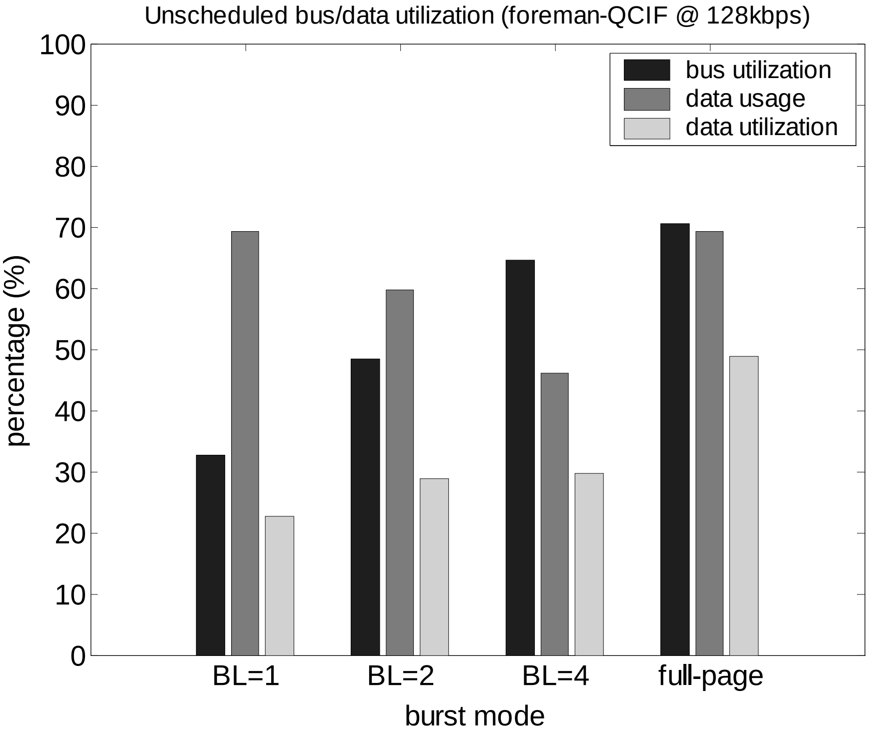


Fig. 4.24 Unscheduled bus utilization, data usage, and data utilization for different burst mode

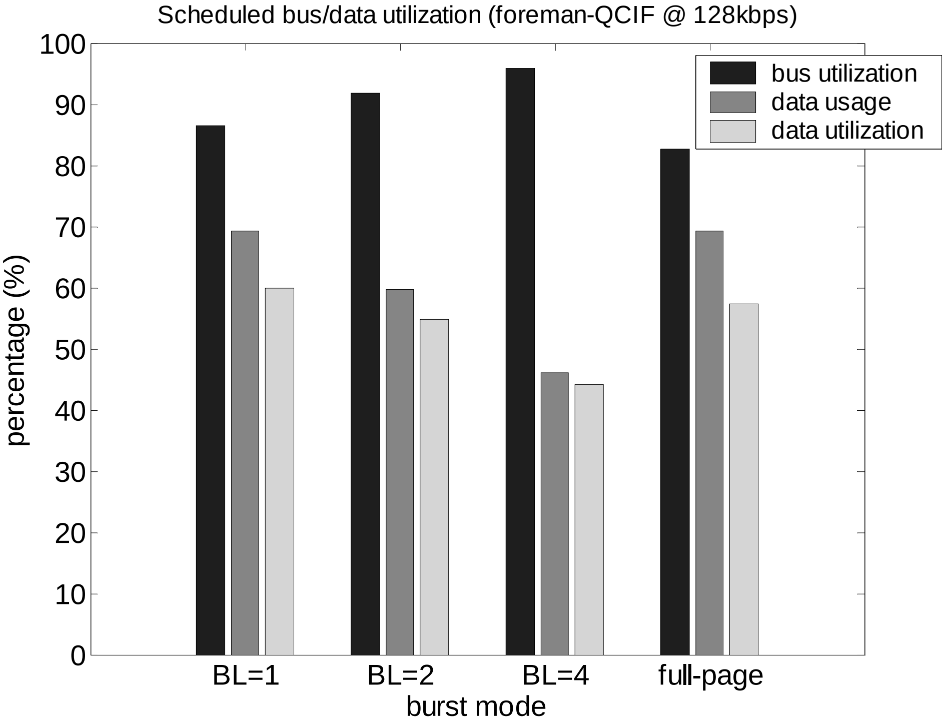


Fig. 4.25 Scheduled bus utilization, data usage, and data utilization for different burst mode

Fig. 4.24 and 4.25 shows the unscheduled and scheduled system level analysis of the above criteria. Before scheduling, the longer burst length provides higher bus utilization instinctively because the short access latency is required for the same amount of fetching data. After scheduling, since longer read burst cycles can provides long overlapping period for the successive commands, burst length = 4 has the highest bus utilization. Nevertheless, the improvement of full-page is smaller than the other case because the BURST TERMINATE command at the end of burst read impedes the possibility of command overlapping. Although bust length = 4 reflects the highest bus utilization, the lowest data usage makes the data utilization become lowest among these burst modes. Therefore, BL = 1 or full-page mode is the better choices on the high-throughput video decoding system. The disadvantage of full-page mode is that it needs longest read data buffers to hold the arbitrary length read data and BURST TERMINATE command also makes the access FSM design more complicated. The average access latency per P MB for unscheduled and scheduled memory accesses is depicted in Fig. 4.24. For the same reason, the reduction of access latency for full-page mode is also bounded by BURST TERMINATE command. For H.264/AVC decoder, Fig. 4.27 and Fig. 4.28 give the requisite execution cycles of P MB at low-bit-rate (32 kbps) and high-bit-rate (128kbps) environments. After inducing data reuse technique, extended RSO method, mentioned in Chapter 3, the execution cycles can reduce about 50 ~100 cycles. Because the amount of fractional motion vectors at high-bit-rate environment is greater than that at low-bit-rate environment, the improvement is more obvious in Fig. 4.28. Furthermore, after applying memory scheduler, the execution cycles per P MB can tremendously reduce up to 15 ~55 % especially when burst length is equal to one. Based on our decoding system, the raise of bus utilization and reduction of access latency reduce the required execution cycles per P 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.

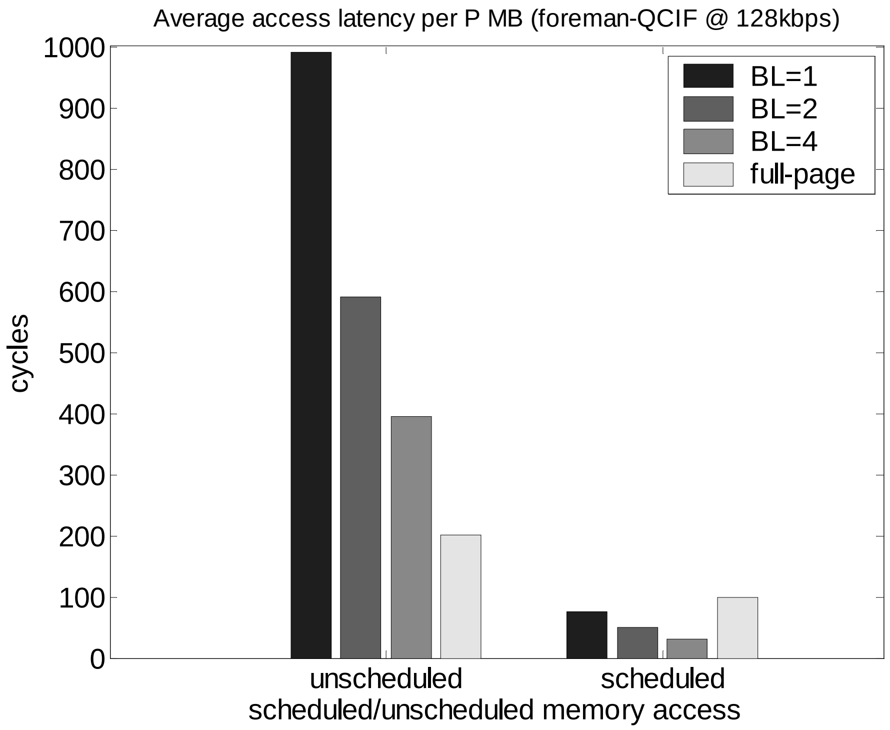


Fig. 4.26 Average access latency per P MB for unscheduled/scheduled memory access

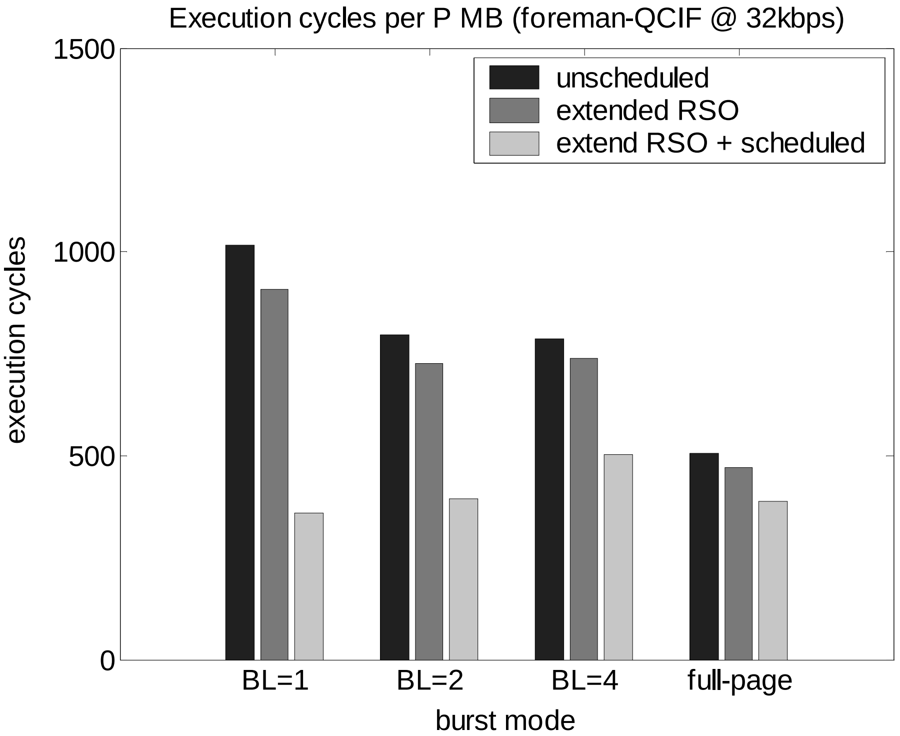


Fig. 4.27 Average execution cycles of P MB for foreman-QCIF @ 32 kbps

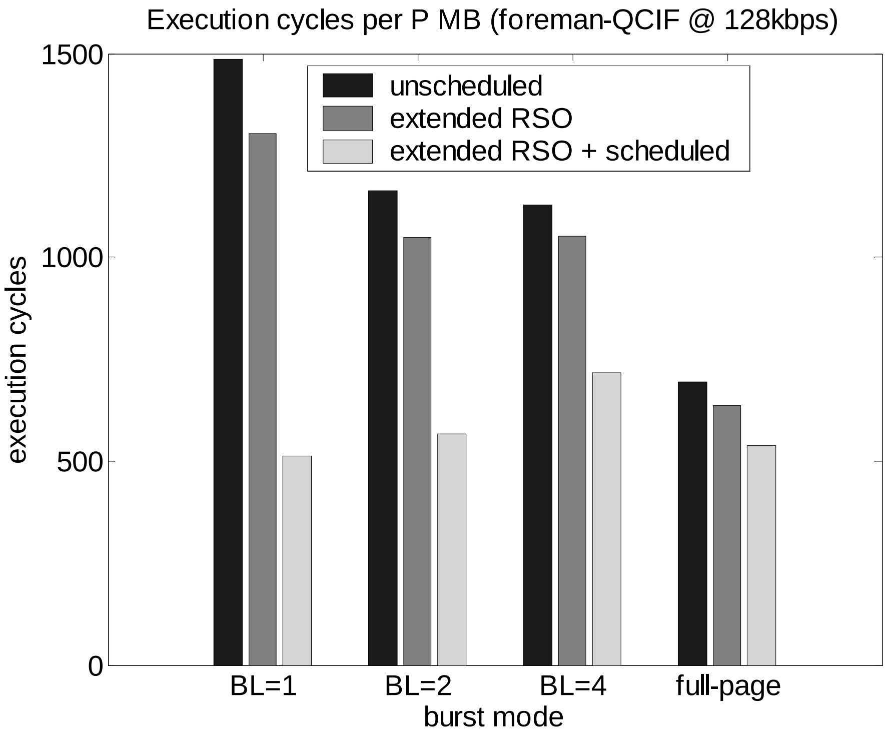


Fig. 4.28 Average execution cycles of P MB for foreman-QCIF @ 128kbps

4.3 Merging Structured Frame Memory Organization

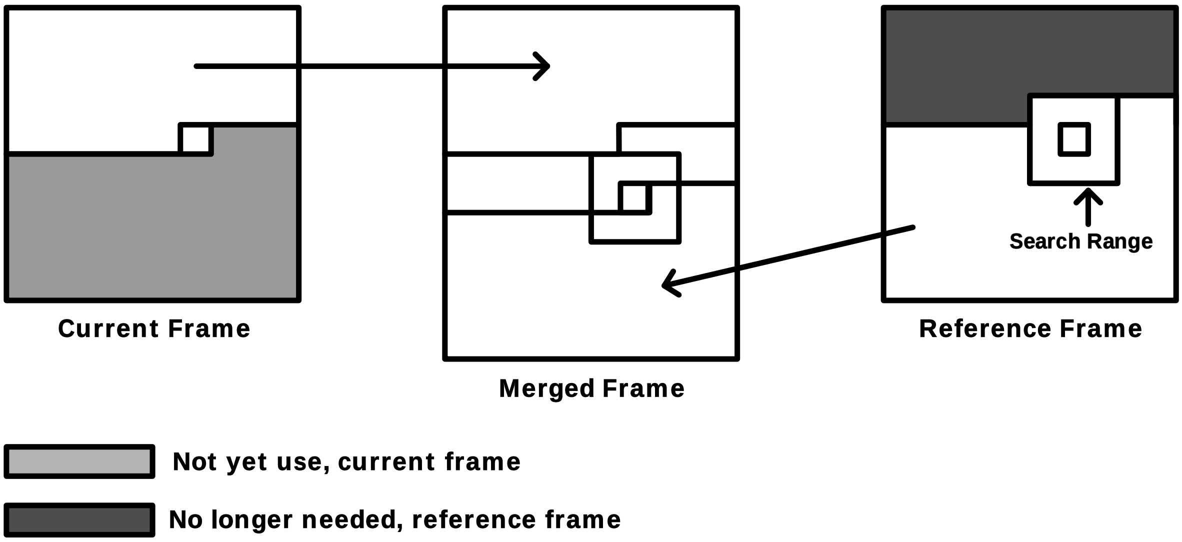


Fig. 4.29 Merging structured frame memory organization

The related works of merging structure discussed, of which concept is shown in Fig. 4.29, in the beginning of this chapter is summarized in Table 4.6. We only focus on Chang's [29] because it presents the possibility of implementation on ASIC design rather than other works just target on DSP or FPGA platform. Chang's combined frame memory architecture [29] can reduce frame memory size up to 57 % and reduce up to 83 % average latency and 39 % average power consumption. To reducing the memory access of MBs with zero-valued MVs and no residual (perfect match), this architecture introduces search range strip buffer (SRSB) and dirty table (DT) to record the status of each MB in in-place region. After simulation in MPEG-4 video decoder system, this architecture can achieve certain effect especially in slow motion or large background video sequence. Now, considering applying this scheme in H.264/AVC system, Chen's [29] only concentrates on MB level. We will expand this idea on 4 x 4-block level for H.264/AVC video decoder in 4.3.1 and illustrates the simulation result and performance.

Table 4.6 Related works of merging structured frame memory

Related work	Platform	Discussion
De Greef's [24]	Video codec on DSP	Idea, initial concept: reduce frame size
Nachtergaele's [25]	H.263 video decoder on DSP	combined frame memory with in-place FIFO: reduce frame size
Brockmeyer's [26]	MPEG-4 ME on DSP	Hierarchy combined frame memory with in-place FIFO: reduce frame size
Denolf 's [27]	MPEG-4 video encoder on DSP or FPGA	3-level hierarchy combined frame memory: reduce frame size
Chang's [29]	MPEG-4 video decoder on Software or ASIC	Perfect match MB skip method: reduce latency, power, and frame size

4.3.1 Analysis

A MB with perfect match is one that has zero-valued MV and no residual. The definitions of prefect match follow different conditions for different video standards. For example, *not-coded* in MB in MPEG-4 is a MB with zero-valued MV and no residual; thus *not-coded* MB is MB with perfect match. However, there is no particular mode that defines zero-valued MV and no residual for H.264/AVC standard. In H.264/AVC, P_SKIP mb_type means that one MB without motion information and residual data in bitstream. That is, motion vector is derived only according to MVP and there has no any guarantee that MVP is zero. If based on MB level statistic foe H.264/AVC@Baseline, the zero valued MB without residual only occurs in P_SKIP mb_type. Hence, we only have to check P_SKIP MB with zero MV. Furthermore, based on 4 x 4 block level analysis, we not only have to check zero MV, but also have to check coded_block_pattern for residual information. The coded_block_pattern specifies which of the six 8 x 8 blocks – luma and chroma – contain non-zero transform coefficient levels. Accordingly, the definition of luma 4 x 4 with perfect match for H.264/ AVC is that motion vector is zero and CBP_luma , LSB 4 bits of coded_block_pattern, is zero.

Similarly, chroma 2 x 2 with perfect match contains zero-valued MV and CBP_chroma, MSB 2 bits of coded_block_pattern.

Table 4.7 Perfect match analysis
JM8.2: QCIF, 30 fps, search range [-16, +15.75], QP = 30

Level	MB based	4 x 4 block based	
Sequence	**MB (%)**	**4 x 4 luma block (%)**	**2 x 2 chroma block (%)**
Salesman	80.4118	85.6572	86.3453
News	75.6982	80.3539	80.6434
Paris	66.969	76.9667	78.177
Table	55.1456	58.9625	59.7103
Suzie	37.1582	39.7623	39.8924
Carphone	21.3745	25.1419	25.5195
Foreman	10.6055	13.5096	13.746
Stefan	7.26982	9.33182	9.56115
Coastguard	2.3439	5.53886	7.13318
Mobile	0.834331	3.42174	4.29115

From the analysis of perfect match in Table 4.7, we can find the ratio of 4 x 4 block based perfect match is greater than that of MB based perfect match. That is to say, 4 x 4 block based perfect match has much chance to save more power and reduce more latency. However, if merging structured frame memory is adopted in H.264/AVC decoder, we have to modify some block. Unlike MPEG-4 decoder, H.264/AVC provides in-loop de-blocking filter that filters the data from adding of residual data and inter-predicted data. In the methodology of skip block with perfect match, the output data of residual adder must directly store to reference frame. Accordingly, we have doing the following assumption: after turning off de-blocking filter, the PSNR degradation is very small for small frame size format such as QCIF or CIF. If we want to exploit methodology mentioned in Chang's [29] on H.264/AVC decoder, the reduction of power consumption is emergent such that users can ignore the degradation of video quality. For that reason, the application must target on small frame size for mobile device based on the above assumption. Here, we only give a little analysis for H.264/AVC decoder. Many problems still have to be resolved such as large search range and we will leave it in future work.

4.4 Summary

In this chapter, we proposed a dual-channel SDRAM frame memory access controller that is build-in device on video decoder. We not only focus on the memory scheduler design, but also adopt the row-major data arrangement to reduce the row-miss ratio dramatically. After system level analysis, the proposed memory access controller design can increase bus utilization and reduce access latency efficiently; in the meanwhile, it also improves the throughput on the entire video decoder. In addition, we also analyze the skip ratio of 4x4 perfect match block based on the merging structured frame memory in H.264/AVC video decoder. The analysis result shows that it has a chance to save more power and latency compared with MB level skip proposed in [29].

Chapter 5
Chip Implementation

5.1 Chip Specification

Table 5.1 Video decoder specification in our design

H.264/AVC Baseline Profile @ 3.2 Level
I, P slice
Variable block size: 16x16, 16x8, 8x16, 8x8, 8x4, 4x8, 4x4
Single reference frame
Search range: [-128, +127.75]
Fractional motion resolution: quarter for luma, 1/8 for chroma
Frame coding
All intra prediction mode
Context Adaptive Variable Length Coding (CAVLC)
Decoding capability: 1280 x 720 HDTV, 30fps at 56 MHz
MPEG-2 Simple Profile @ Main Level
I, P picture
Search range: [-256, +255.5]
Fractional motion resolution: half luma and chroma
Frame coding
Decoding capability: 1920 x 1080 HDTV, 30fps at 80.92 MHz

Table 5.1 lists the specification of our dual-standard video decoder. After synthesis on Cadence RTL compiler using UMC 0.18 um COMS technology, total gate count is 491260 (including SRAM) and the gate count of each component is listed in Table 5.2. Table 5.3 lists on/off chip SRAM used on each module. The total size of on-chip SRAM is 23.5 KBytes. The chip specification and chip layout are shown in Table 5.4 and Fig. 5.1 respectively.

Table 5.2 Synthesis result of the MPEG-2/H.264 dual-video decoder (including on-chip SRAM)

Component	Standard	Gate count
De-blocking Filter	MPEG-2/H.264	187825
Motion Compensation	MPEG-2/H.264	84266
Syntax Parser & System control	MPEG-2/H.264	38715
Residual Adder & VL-FIFO	MPEG-2/H.264	13214
SDRAM Memory access controller	MPEG-2/H.264	7933
Combined VLC/CAVLC	MPEG-2/H.264	6688
Intra Prediction	H.264	56936
Content Memory	H.264	24263
4 x 4 IQ	H.264	14184
Integer / Hardmard Transform	H.264	5170
IDCT	MPEG-2	39727
8 x 8 IQ	MPEG-2	12339
Total	491260	

Table 5.3 On-chip/off-chip memory in our design

	Module	Depth x Width	Port	Number
On-chip	Motion compensation	120 x 10	Single-port	x 8
	De-blocking filter	2048 x 32	Single-port	x 1
		1024 x 32	Single-port	x 2
		128 x 11	Single-port	x 1
	Intra prediction	1024 x 32	Single-port	x 1
	Combined CAVLC/VLC	1024 x 5	Single-port	x 1
	Syntax parser	128 x 16	Single-port	x 1
	Content buffer	64 x 32	Single-port	x 2
		32 x 32	Single-port	x 2
Total	**23520 Bytes**			
Off-chip	Frame memory	512K x 32 x 4 banks		x 2

Table 5.4 Chip specification of MPEG-2/H.264 dual video decoder

Technology	**UMC 0.18um CMOS 1P6M**
Package	208 CQFP
Die Size	3900 um × 3900 um
Core Size	3500 um × 3500 um
Max Clock Rate	100 MHz
Power Consumption (Core Power)	44.35 mW @ 720 HD for H.264 19.96 mW @ 625 SD for H.264 68.34 mW @ 1080 HD for MPEG-2 13.57 mW @ 625 SD for MPEG-2
Total Gate Count	**491260**
On-chip Memory	23.5 KB SRAM
Off-chip Memory	3.2 MB SDRAM x 2
Pad number	**115**
Input Pad	12
Output Pad	41
IO Pad	62

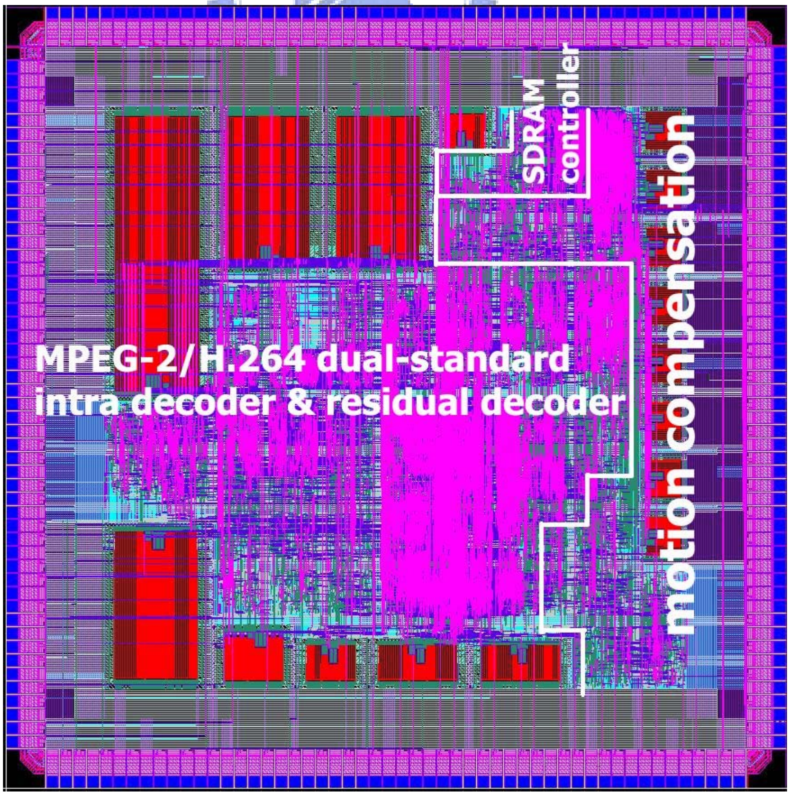


Fig. 5.1 The chip photo of MPEG-2/H.264 dual-video decoder

5.2 *Comparison with Related Works*

Table 5.5 H.264 decoder comparison with related work

	C & S [6] ISCAS'04	Conexant [30] ISCE'04	Chen's [5] ISCAS'05	Our work[31][32] ISCAS'05 VLSI-DAT'05
Process	0.13 um	0.13 um	0.18 um	0.18 um
Decoder Spec.	**H.264 @ Baseline** (multi-standard)	**H.264 @ Main**	**H.264 @ Baseline**	**H.264 @ Baseline** (MPEG-2 @ SP)
Pipeline	4-stage	N/A	Hybrid	Hybrid
Design	ARM based	ARM based	ASIC	ASIC
Frame memory	External SDRAM	External DDR RAM	External SRAM	External SDRAM
Memory hierarchy	External memory	Local RSB + External frame memory	External memory	Local RSB + External frame memory
Gate count	910K (Logic),	300 K (Logic)	217 K (Logic) 10 KB on-chip SRAM	491K (with 23.5 KB on-chip SRAM)
Frame resolution	1920 x 1080 HD, 30fps	2048 x 1024, 30fps	2048 x 1080, 30fps	1280 x 720 HD, 30fps
Operating frequency	CPU, Local Bus: 170 MHz System Bus: 130 MHz	200 MHz	120 MHz	56 MHz
Power	554 mW for 1080HD 159mW for CIF	160 mW for 2048 x 1024 HD	N/A	44.35 mW for 720HD 4.51 mW for CIF

Table 5.5 lists the comparison with related works about H.264 video decoder. We only focus on memory related comparison. The proposed ASIC decoder design is hybrid pipeline processing. The frame memories adopt external SDRAM. The memory hierarchy of the entire video decoder can be treated as two-level memory hierarchy including internal local row store buffer (RSB) and external frame memory buffer. The concept of memory hierarchy is

illustrated in Fig. 5.1. Row store buffer is the particular feature that is different from the previous video standards for H.264 video decoder. Many modules such as inter prediction, intra prediction, in-loop de-blocking filter and CAVLC requires RSB, of which size is equal to the frame width, to store the decoding information like pixels, motion vector and coefficient token. The benefit of partitioning these row data from external memory is reduction of required bandwidth to external memory and global bus power. It also eliminates some data access conflict on global bus. The overhead is requiring more internal memory to store the row information.

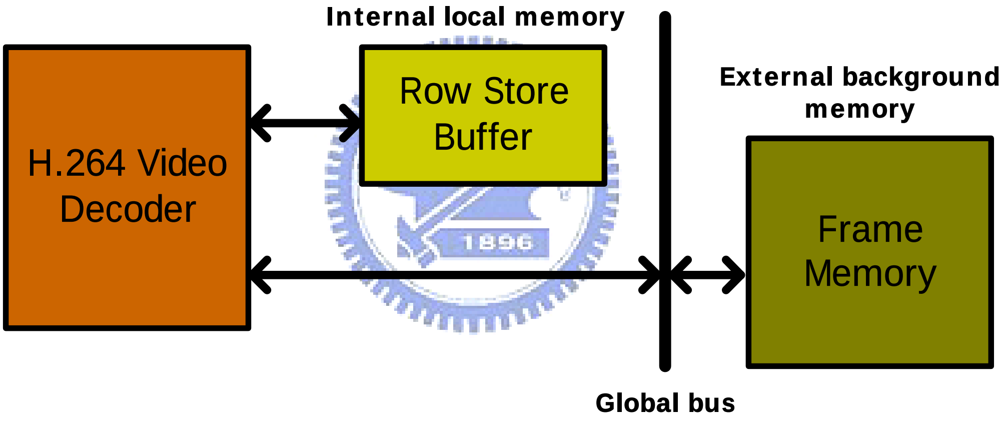


Fig. 5.1 Memory hierarchy for the entire video decoder

Chapter 6
Conclusion and Future Work

6.1 Conclusion

Based on the prevalent application of Digital TV adopted in digital video broadcasting (DVB) transmission system, combined motion compensation engine for MPEG-2/H.264 dual-standard video decoder is proposed in this dissertation. Motion compensation engine consists of three parts: motion vector generator, interpolator, and external frame memory access controller. After mapping MVP algorithm to ASIC design on the concept of two-level memory hierarchy. The design target of interpolator and frame memory access controller is to reduce external memory access and improve throughput of the entire video decoder.

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. The proposed data reuse technique for fraction motion compensation introduces content buffer and content-swap operation attached on our interpolators design. After applying this concept, the extended 2x2 RSO for H.264/AVC can reduce the required bandwidth about 30 % compared with 2x2 column-major one.

Beside, the dual-channel 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. Row-major arrangement in our scheduling scheme can minimize the row-miss (at the same bank) ratio that contributes the maximum latency among all scheduling cases. We build system level hardware-like C++ model and analyze the system performance. Compared to unscheduled situation, the experiment result shows that the access latency can be reduced by 50 % ~ 90 % and bandwidth utilization can be increased by 20 % ~ 2.8 times. In the meanwhile, the throughput of the overall video decoder can improve about 10 % ~ 60 % after combining extended RSO method and memory scheduling.

6.2 *Future Work*

The proposed motion compensation for dual-video standard only supports P frame decoding. If we want to H.264@Main Profile, the subjects such as B-frame, weighted prediction, and direct mode should be taken into account. Because B-frame may contains double motion vectors, it requires read reference data from forward and backward frames and then write reconstructed data to display buffer. These operations are more complicated than P frame; thus, the scheduling of the read/write accesses efficiently to meet our decoder specification is a challenge.

Considering memory organization topic on this dissertation, a dual-channel SDRAM controller for ping-pong structured frame memories has been presented in this thesis. However, there are still some important issues should be considered in order to provide more complete system integration. For example, if display buffer shares the data bus with frame memories, a smarter bus arbitration and memory controller should be designed. As for merging structured frame memory organization based on H.264/AVC decoder, we give the analysis for 4 x 4 block based perfect match. However, there are still many issues have to be analyzed if we want to exploit this idea on ASIC design. The related work only targets on search range [-16, +15] since this size is identical to one MB size. Based on this search range, the design of in-place FIFO and dirty table becomes easier. If we want to target on large search range, the size of in-place FIFO becomes very large and design of the address generator becomes more complicated. Another problem is that the proportion of perfect match block must be larger than the backup of the content of in-place FIFO; otherwise, the update and backup between background memory and in-place FIFO will degrades the throughput of the entire video decoder. In brief, the feasibility of the merging structured memory organization on ASIC design still has to be evaluated.

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] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] P. R. Panda, N. Dutt, and A. Nicolau, *Memory Issues in Embedded Systems-on-Chip: Optimization and Exploration.* Boston, MA: Kluwer Academic Publishers, 1999.

[13] Micron Technology, Inc. product documents. [Online]. Available: http://www.micron.com/products/

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] Micron Technology, Inc. MT48LC2M32B2P-5 64Mb SDRAM (Jan. 2005). [Online]. Available: http://www.micron.com/products/dram/sdram/partlist.aspx?density=64Mb

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] S. Wuytack, J. –P. Diguet, 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.

[29] Y. C. Chang Nelson, and T. S. Chang, “Combined frame memory architecture for motion compensation in video decoding,” *in Proc. IEEE Int. Symp.Circuits and Systems*, 2005, pp. 1806-1809.

[30] 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.

[31] 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.

[32] 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.

作者簡歷

姓名　　：王勝仁

出生地　：台灣省台南市

出生日期：1980. 12. 11

學歷：	1987. 9 ~ 1993. 6	台南市立大橋國民小學
	1993. 9 ~ 1996. 6	台南私立瀛海中學
	1996. 9 ~ 1999. 6	台南市立第一高級中學
	1999. 9 ~ 2003. 6	國立交通大學 電子工程系 學士
	2003. 9 ~ 2005. 6	國立交通大學 電子研究所 系統組 碩士

得獎事績

2002 春 第二學期電子研究所書卷獎

2002 秋 般之同專題計畫獎學金

2003 春 般之同專題成果獎學金

發表論文

- S. Z. Wang, T. A. Lin, T. M. Liu, and C. Y. Lee, “A new motion compensation design for H.264/AVC decoder,” *in Proc. IEEE Int. Symp.Circuits and Systems*, 2005, pp. 4558-4561.
- 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*, vol. ?,2005 , pp. 1806-1809.
- T. M. Liu, S. Z. Wang, W. H. Peng, and C. Y. Lee, “Memory efficient and low complexity scalable soft VLC decoding for the video transmission,” *in Proc. IEEE Asia-Pacific Conf. Circuits and Systems*, 2004. Proceedings, vol. 2, 2004, pp. 673-676.