國立交通大學

電機學院　電子與光電學程

碩 士 論 文

適用於 H.264/AVC 之降低記憶體頻寬的動作補償

A Memory Bandwidth-Reduction Motion Compensator for H.264/AVC Application

研 究 生：陳浩民

指導教授：李鎮宜　教授

中 華 民 國 九 十 九 年 十二 月

適用於 H.264/AVC 之降低記憶體頻寬的動作補償

A Memory Bandwidth-Reduction Motion Compensator for H.264/AVC Application

研 究 生：陳浩民　　Student：Hao-Min Chen

指導教授：李鎮宜　　Advisor：Chen-Yi Lee

國 立 交 通 大 學
電機學院　電子與光電學程
碩 士 論 文

A Thesis
Submitted to College of Electrical and Computer Engineering
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master of Science
in
Electronics and Electro-Optical Engineering
December 2010

Hsinchu, Taiwan, Republic of China

中華民國九十九年十二月

適用於 H.264/AVC 之降低記憶體頻寬的動作補償

學生：陳浩民　　指導教授：李鎮宜 教授

國 立 交 通 大 學　電 機 學 院　電 子 與 光 電 學 程 碩 士 班

摘　　要

近年來，對於已被數位視訊廣播的高傳真電視廣播服務和藍光光碟所採用的 H.264/AVC High Profile 視訊標準，其需求是很必要的。而動作補償單元的計算量通常占了整個視訊解碼系統的大多數，這是由於它需要對參考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 Main/High Profile 視訊標準支援了雙向參考畫面，因而使得所需的記憶體頻寬大量增加。我們提出的記憶體頻寬縮減策略除了可有效地減少所需的記憶體頻寬高達 80 %之外，同時維持和整個視訊解碼系統相同的解碼順序。和傳統的架構相較之下，針對 H.264 提出的可重新架構的內插器，可省下 20 % 的邏輯閘數量。我們的動作補償單元同時支援了 H.264 Baseline Profile @ 4.0 Level 和 H.264 Main/High Profile @ 4.0 Level，對即時解碼能力而言可達到 1080 HD @ 100.0 MHz，而總邏輯閘數量為 68 K。

A Memory Bandwidth-Reduction Motion Compensator for H.264/AVC Application

Student : Hao-Min Chen Advisor : Dr. Chen-Yi Lee

Degree Program of Electrical and Computer Engineering

National Chiao Tung University

ABSTRACT

In recent years, H.264/AVC High Profile video standard, which has been adopted by the Digital Video Broadcasting (DVB) HDTV broadcast service and the Blu-ray Disc storage format, is necessary in demand. The computation time of motion compensation unit is usually accounted for most of the video decoding system because of the enormous data transfer with reference frame memories. Particularly in the most advanced H.264/AVC Main/High Profile video standard supports bi-prediction reference frame, which makes the memory bandwidth required for a significant increase. Our proposed reduction strategies of memory bandwidth cannot only effectively reduce the required memory bandwidth up to 80% but also maintaining the same decoding order as that of entire video decoding system. The proposed restructured interpolator can save 20% of the number of logic gates compared to traditional design. Our motion compensator also support H.264 Baseline Profile @ 4.0 Level and Main/High Profile @ 4.0 Level, in terms of real-time decoding up to 1080 HD @ 100 MHz, while the total number of 68k NAND2 CMOS logic gate count.

Acknowledgements

首先要感謝的是我的指導教授李鎮宜研發長在我的碩士生涯中給我的指導與鞭策，在他熱心指導過程中，雖然常常我有很爆笑的回應，但總是能很有耐心的指導我。

接下來要感謝，帶領我的博士班學長，也是我們 Si2 多媒體組 leader 李曜，在他有效的領導與熱心的幫助，讓我的研究持續有進展；另外要感謝王勝仁學長，他的論文非常優秀，很值得我借鏡，對我的研究幫助很大；接下來要感我們 Si2 實驗室的成員們，耀琳，謝謝你爲了教我 ICLAB 常常陪我熬夜；還有見縫就插針的的明瑜；要把好的 idea 分給我的建辰；分享酒店文化的義澤，分享把妹經驗的人偉；常常幫我介紹的勝舜；下課常一起聊天的元雍與盈鋒；老是說真的的長宏；常回來的子明學長；老是叫我捐錢的欣儒；大家一起做研究，一起敖夜，一起唱歌，一起聊天，一起聚餐。在苦悶的研究生涯帶來了豐富的歡樂色彩。

最後要感謝的是我的家人，強列的要求我再進修，也由於有他們的支持，讓我可以在沒有後顧之憂的，全心完成我的研究。

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Motivation

In recent years, the newest video coding standard published jointly as Part 10 of MPEG-4 and ITU-T Recommendation H.264 [1] provides fine video compression performance. The new H.264/AVC standard provides a technical solution for a wider range of applications, including video-on-demand (VOD), mobile networks, high definition TV, broadcast over cable, satellite, cable modem, DSL or terrestrial, interactive or serial storage like BD, conversational services over ISDN, Ethernet, LAN, wireless, or mobile network, multimedia messaging services over DSL, ISDN, etc.

Besides, in Nov. 2004, Digital video broadcasting handheld, DVB-H [5], 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. Moreover, high definition TV requires huge data transmission particular in frame memory, a memory 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. Motion compensation should also provide efficient memory bandwidth reduction to reduce memory bandwidth.

1.2 Thesis Organization

This 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 presented firstly. Then, the motion compensation engine for H.264 high profile is illustrated. In Chapter 4, we propose the bandwidth reduction strategies to reduce the required bandwidth particularly in H.264/AVC integral and fractional motion compensation. We also presents frame memory organization, and memory bandwidth analysis. Implementation result is given in Chapter 5. Finally, conclusion and future work is shown in Chapter 6.

Chapter 2
Algorithm Description and Analysis

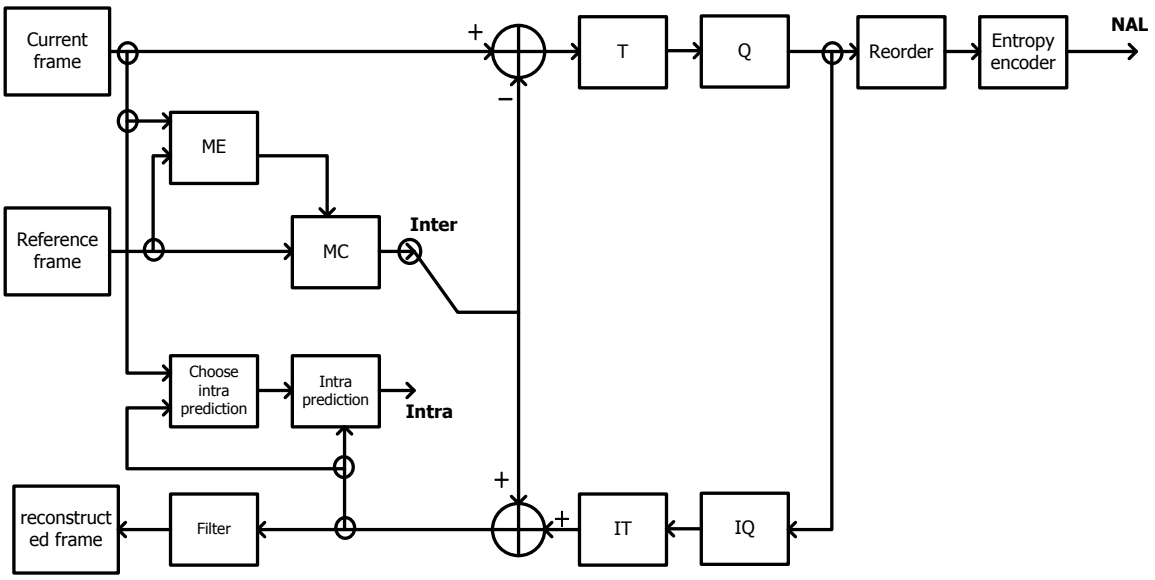

Fig 2.1 General structure of H.264 encoder

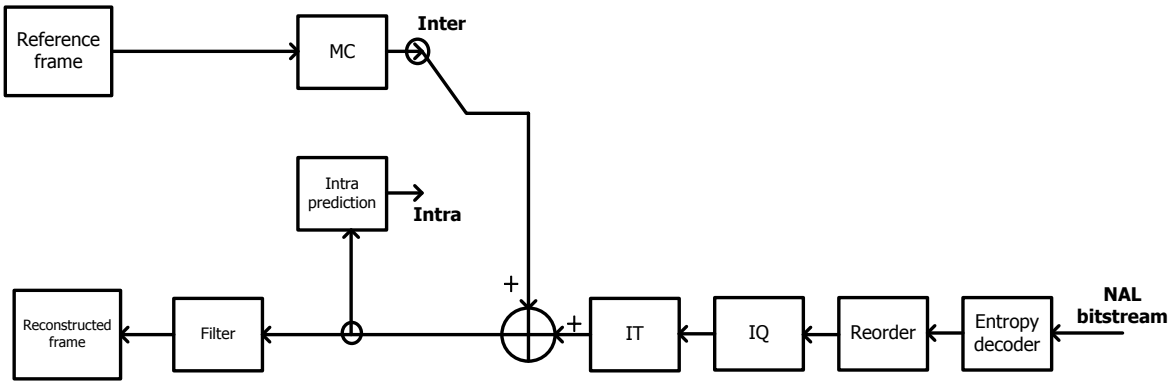

Fig 2.2 General structure of H.264 decoder

Fig 2.1 and Fig 2.2 shows the general structure of H.264/AVC video encoder and decoder respectively [6]. The H.264/AVC design covers a Video Coding Layer (VCL) and

Network Abstraction Layer (NAL). We only discuss on VCL that efficient represents the video content. The concept of H.264/AVC submits the so-called *block-based hybrid video coding*. It consists of hybrid of temporal and spatial prediction and is simultaneous with transform coding.

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 H.264/AVC high profile is presented in section 2.3

2.1 Profiling

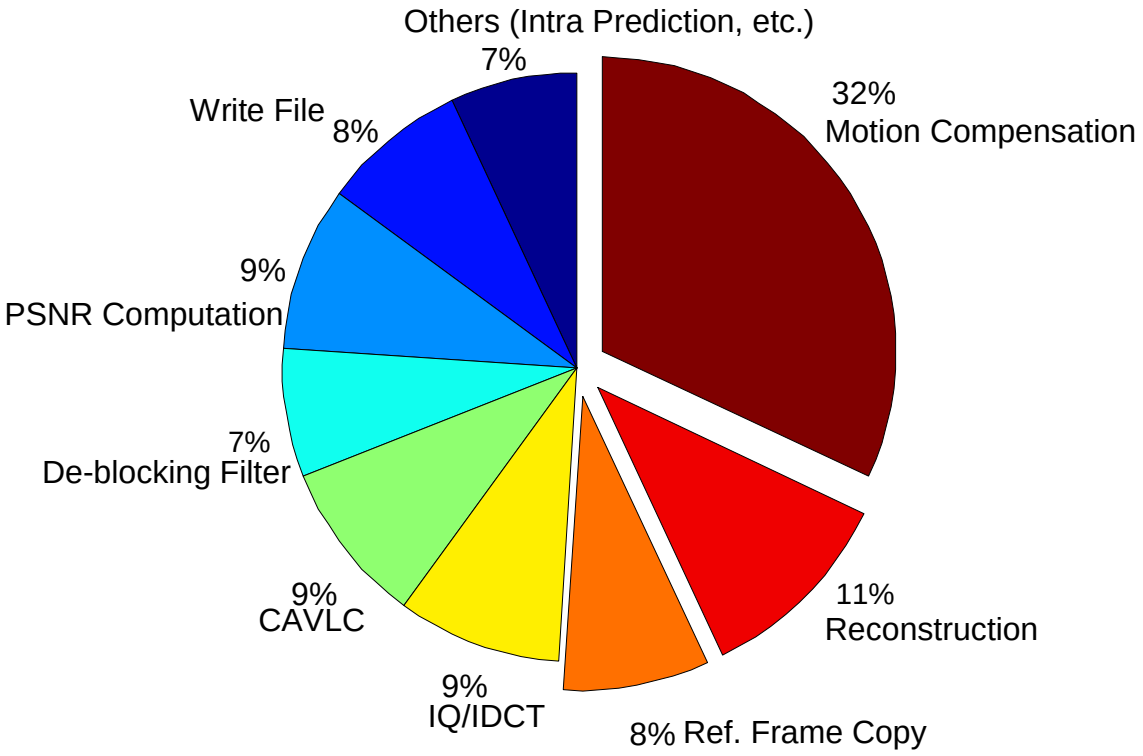

Fig 2.3 H.264/AVC video decoder software profile on ARM processor (JM 8.2)

Fig 2.3[8] shows the H.264/AVC profile on ARM processor. The reference software is JM 8.2 [7]. We can find motion compensation related modules, including motion compensation, reconstruction, and reference frame copy, occupy 51 % proportion of the entire video decoder. Parallel processing, bandwidth reduction, or pipeline processing on ASIC design can significantly reduce this dominated part.

2.2 Inter Prediction Algorithm for H.264/AVC Standard

H.264/AVC standard supports variable block size (VBS) in inter prediction [1] [2]. The smallest block size could reach least 4x4 for luma and 2x2 for chroma. Fig 2.4 [1] illustrates all types of partitions.

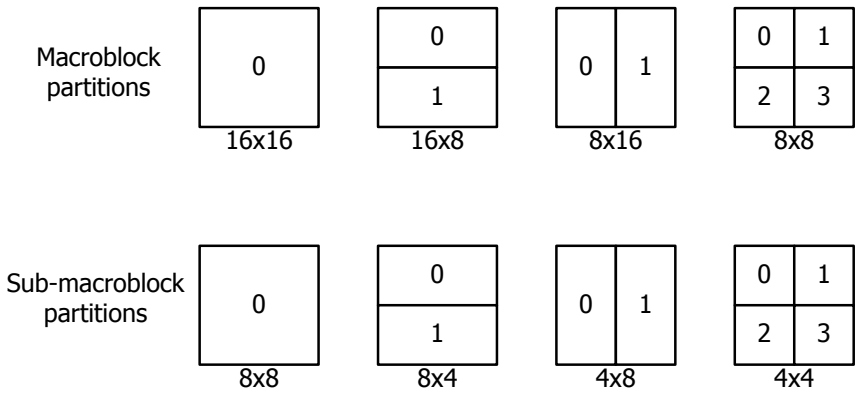

Fig 2.4 Macroblock partitions and sub-macroblock partitions

H.264/AVC standard also supports high motion resolution that reaches quarter motion accuracy for luma sample and one-eighth for chroma sample. 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 illustrated in Fig 2.5 (a)-(c). The prediction value of chroma component is generated using bilinear interpolator illustrated in Fig 2.5(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 divided into two 1-D to reduce hardware cost, in other words, horizontal filter first and then vertical one, or vice versa.

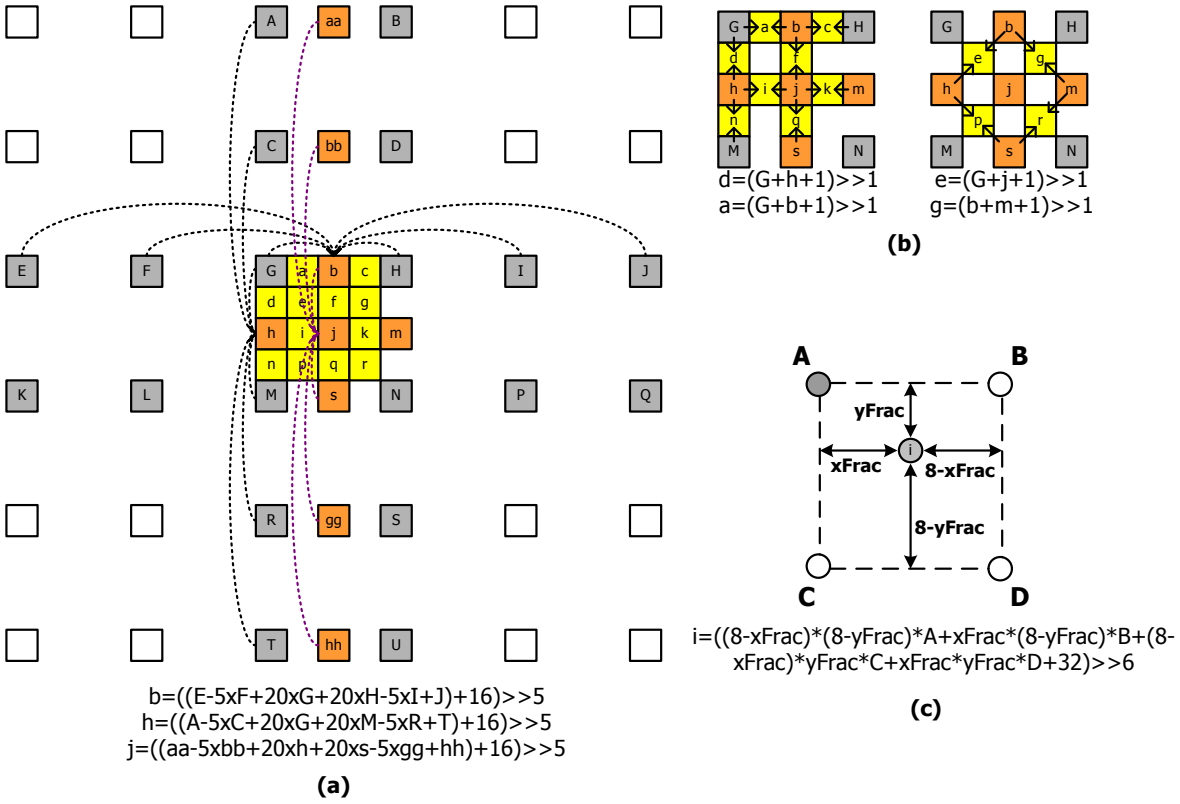

Fig 2.5 (a) Luma half sample with 6-tap FIR, (b) luma quarter sample with bilinear filter, (c) chroma sample with bilinear filter. Upper-case letters indicate the full samples and lower-case letters indicates the interpolated fractional samples

Motion vector difference (MVD) and motion vector prediction (MVP) generate the motion vector which Eq. 2.1 express the equation.

$$MVx = MVDx + MVPx$$
$$MVy = MVDy + MVPy \qquad \text{Eq. 2.1}$$

MVD is decoded from bit-stream and MVP is predicted according to neighboring motion vectors. MVP algorithm, 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 2.6 [8]. Eq. 2.2 expresses the equation of median prediction. Besides, some boundary conditions or exceptions have to be handled carefully. For instance, when MVC is not available, its value is replaced by MVD. We do not go into detail of those trivial boundary conditions in here.

$$MVP = median(MVA, MVB, MVC) \quad \text{Eq. 2.2}$$

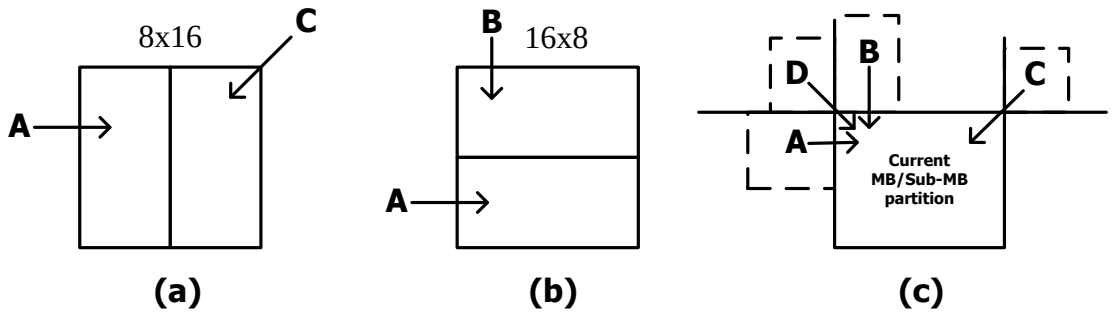

Fig 2.6 (a) Directional prediction for 8 x 16 block size, (b) directional prediction for 16 x 8 block size, (c) median prediction

In addition to the motion-compensated block size described in Fig 2.4, a P macroblock can also be coded to P_SKIP mode. For this coding mode, neither residual signal nor motion information is transmitted. In other words, motion vectors are only decided according to MVP. The reconstructed reference pixels are obtained similar to macroblock type P_16x16. Macroblock coded in P_SKIP are often located in large area with no change or slow motion. In addition to the above techniques, H.264/AVC also supports multiple reference frame, weighted prediction and direct mode for B slice, which we will present in section 2.3. These tools can also improve coding efficiency efficiently.

2.3 Inter Prediction for H.264/AVC High profile Standards

Considering motion compensation, the tools supported by H.264/AVC Main/High Profile are B slices, Weighted Prediction and Interlace video.

In an inter-coded macroblock of B slice, each macroblock partition may be predicted from one or two reference pictures, forward and backward the current picture in temporal order. This tool provides better coding efficiency with more possibilities to select the best-match prediction references for the macroblock partitions in B slice. Fig 2.7 shows the 3 reference directions: (a) Forward and backward reference pictures, the so-called bi-directional reference, (b) backward reference, and (c) forward references [6]. B slices use two lists of coded reference pictures, LIST_0 and LIST_1. These two lists can include backward and/or forward coded pictures respectively.

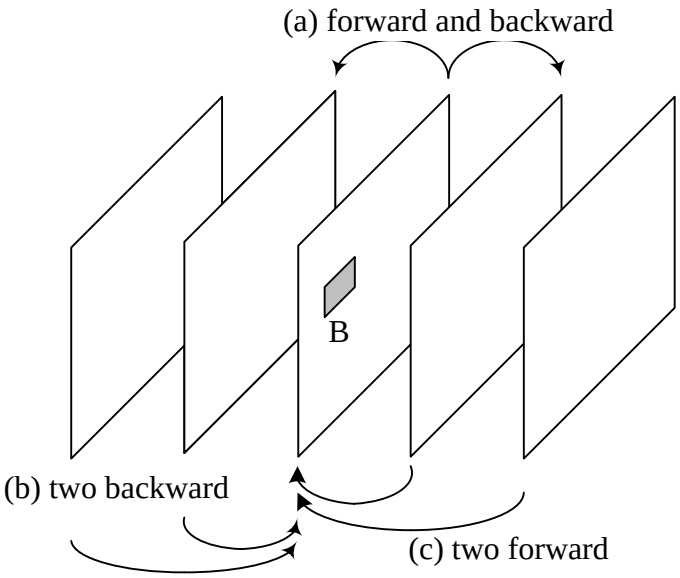

Fig 2.7 Bi-prediction examples

In B slice, there are four prediction modes: (a) direct mode, (b) LIST_0 mode, (c) LIST_1 mode, and (d) bi-predictive mode. For a macroblock, each partition can choose different prediction modes. When the 8 x 8 partition size is used, the chosen mode for each 8x8 partition is applied to all sub-partition within that partition. Fig 2.8 shows two examples of prediction mode combinations. In Bi-predictive mode, two motion-compensated reference regions are obtained from LIST_0 and LIST_1 picture respectively. The motion vectors from LIST_0 and/or LIST_1 in a bi-predictive macroblock or block are predicted form neighboring motion vectors with the same temporal direction. For instance, a motion vector from the current macroblock pointing to a forward picture is predicted from other neighboring vectors that also point to forward pictures.

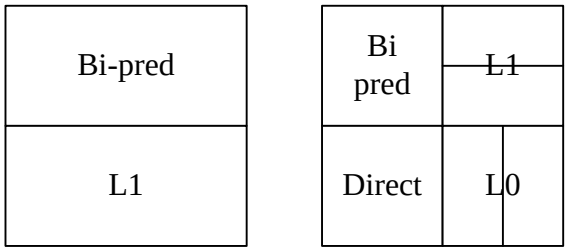

Fig 2.8 Examples of prediction modes in B slice macroblocks

Similar to the skipped P macroblock coded in P_SKIP mode, a B macroblock can also be coded in direct mode. In direct mode, no motion vector is transferred for a B slice macroblock or macroblock partition encoded. Instead, the decoder predicts the motion vectors of LSIT_0 and LIST_1 with neighboring vectors and carries out bi-predictive motion compensation block. There are spatial and temporal mode can be used to calculate the LIST_0 and LIST_1 motion vectors for direct mode macroblocks or partitions.

Spatial direct mode is similar to P_SKIP mode. Furthermore, it supports bi-prediction and 4x4 block size accuracy. The double motion vectors are decided according to MVP.

However, some conditions or exceptions have to be handled carefully. For example, in case of the co-located MB or the partition in the picture that contains the co-located macroblock has a motion vector that is less than +/- 1/2 luma samples in magnitude (and in some other conditions), one or both of the predicted vectors are set to zero. We do not go into detail of those trivial conditions here.

Temporal direct mode differs from P_SKIP mode. The same with the spatial direct mode, the block size is also 4 x 4 block size accuracy, the motion vectors mvL0, mvL1 are derived as scaled versions of the motion vector mvCol of the co-locate sub-macroblock partition. The scaled method is based on the picture-order-count (POC) distance between the current and LIST_1/LIST_0 picture. Fig 2.9 shows the illustration of temporal direct-mode motion vector inference. When the object is constant velocity motion, it is suitable-coded in temporal direct mode. When the object is the average form backward and forward, it is suitable-coded in spatial direct mode. When the object is still, it is suitable-coded in skip mode. Encoder can use skip/direct mode to save one/two motion vector differences (mvd) in every skip/direct mode partition for further enhance compression efficiency.

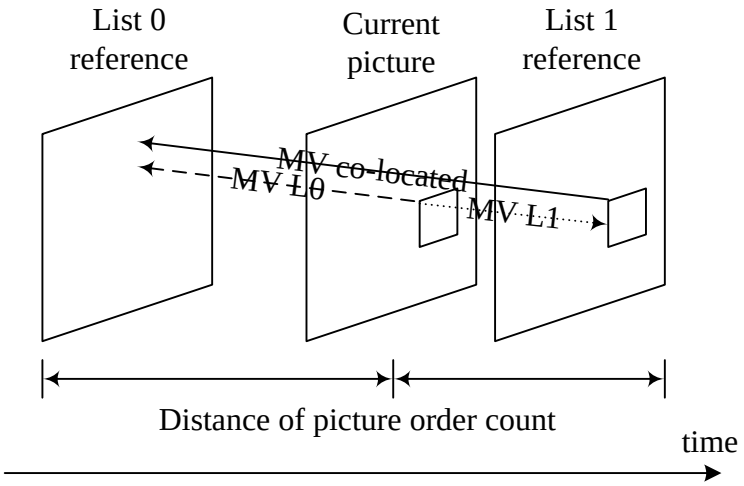

Fig 2.9 Example for temporal direct-mode motion vector

Another tool supported in Main/High Profile is Weighted Prediction (WP), which is a method of scaling the samples to increase the video quality in H.264/AVC video decoding. An application of weighted prediction is to control the relative weighted of interpolated regions to the motion compensated prediction process. For example, WP may be effective in coding of 'fade' transitions (where one scene fades into another). There are three modes in Weighted Prediction. When Default mode is in use, two motion compensated reference regions are obtained from LIST_0 and LIST_1 picture respectively and each sample of the prediction block is calculated as an average of the LIST_0 and LIST_1 prediction samples. Eq. 2.3 expresses the equation

$$predPart = (predPartL0 + predPartgL1 + 1) >> 1 \quad \text{Eq. 2.3}$$

When explicit or implicit mode is in use, Eq. 2.4 is used to calculate the sample of the prediction block. The difference between explicit and implicit mode is the weighting factors are calculated based on the picture-order-count distance between LIST_0 and LIST_1 reference pictures in implicit mode. It is similar to temporal direct mode in motion vector prediction. When explicit mode is in use, the encoder determines weighting factors. In other words, implicit mode objection is to save weighted prediction parameter in bit-stream for further enhance compression efficiency.

$$predPart = ((predPartL0 * w_0 + predPartL1 * w_1 + 2^{\log WD}) >> (\log WD + 1)) + ((o_0 + o_1 + 1) >> 1)) \quad \text{Eq. 2.4}$$

As for interlace video tool, video signal may be sampled as a sequence of complete frames or interlaced fields. An interlaced video sequence contains a series of fields. A field consists of either the odd-numbered or the even-numbered lines within a complete video

frame. Fig 2.10 illustrates the fields in video sequence. Half of the data in a complete video frame is represented as a field and is sampled at each temporal interval. The advantage of interlaced video coding is that it is possible to send twice as many fields per second as the number of frames in an equal progressive sequence with the same data rate, giving the appearance of smoother motion. For instance, a NTSC video sequence consists of 60 fields per second and, when played back, motion can appears smoother than in an equivalent progressive video sequence containing 30 frames per second.

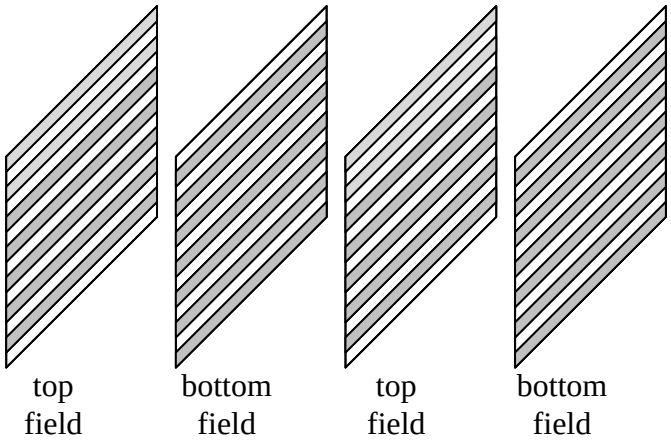

Fig 2.10 Interlaced video sequence

Frame coding is more efficient than field coding for progressive video and static pictures in interlaced video. Oppositely, field coding is more efficient for moving pictures in interlaced video. However, sometimes not complete frames are fast moving. Hence, H.264/AVC Main/High profile provides another tool in interlaced video, macroblock-adaptive frame/field (MBAFF), to provide macroblock level interlacing. Similar to MBAFF, the picture level interlacing sometimes is called PicAFF. As an extension of PicAFF, MBAFF is used to improve coding efficiency of picture with both static and moving regions [21]. In MBAFF mode, the current slice is processed in units of 16 luma samples wide and 32 luma samples high, each of which is coded as a “macroblock pair” as shown in Fig 2.11. The encoder can

choose to encode each MB pair as (a) frame macroblock pair (b) field macroblock pair and may select the optimum coding mode for each region of the picture.

(a)frame MB mode **(b)field MB mode**

Fig 2.11Macroblock-Adaptive Frame-Field Coding

2.4 Bandwidth Requirement for Inter Prediction

Up to now, we can find interpolation issue becomes more and more important in state-of-the-art video coding. The interpolation window becomes double for the same block; In other words, it requires double cycles to interpolate each macroblock. For instance, it requires two 9 x 9 interpolation windows to interpolate a luma 4 x 4 block and four 3 x 3 interpolation windows to interpolate two chroma 2 x 2 blocks in B macroblock.

In worst case, interpolator needs 398MB/s in P frame, 796MB/s in B frame when supporting 1920 x 1088 30fps. In other words, motion compensation needs huge memory bandwidth requirement. Huge data also means large power consumption for bus activity and data operation.

To reduce bandwidth requirement from frame memory, strategies of memory bandwidth reduction for motion compensation will be proposed in Chapter 4.

2.5 Summary

From the H.264/AVC profiling on ARM processor, an efficient hardware accelerator or ASIC design for motion compensation is important. The inter prediction for H.264/AVC Baseline, Main/High profiles, and the bandwidth requirement are also illustrated in this Chapter.

Chapter 3
Motion Compensation Design for H.264/AVC Main/High Profile video decoder

The state-of-the-art video coding standard H.264/AVC provides better compression ratio that significantly outperforms all previous video compression standards. However, H.264/AVC supports Main/High profile and provides many tools compare with Baseline Profile for further enhance compression ratio. Therefore, a development of combining multi-video coding profiles is essential to support modern multimedia systems. Therefore, it is the challenge of designing efficient video decoder for multi-profile video application without significantly increase complexity.

This chapter will discuss that designing of motion compensation, which dominates the amount of data transfer on the H.264/AVC 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 H.264/AVC Baseline/Main/High profile and the 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

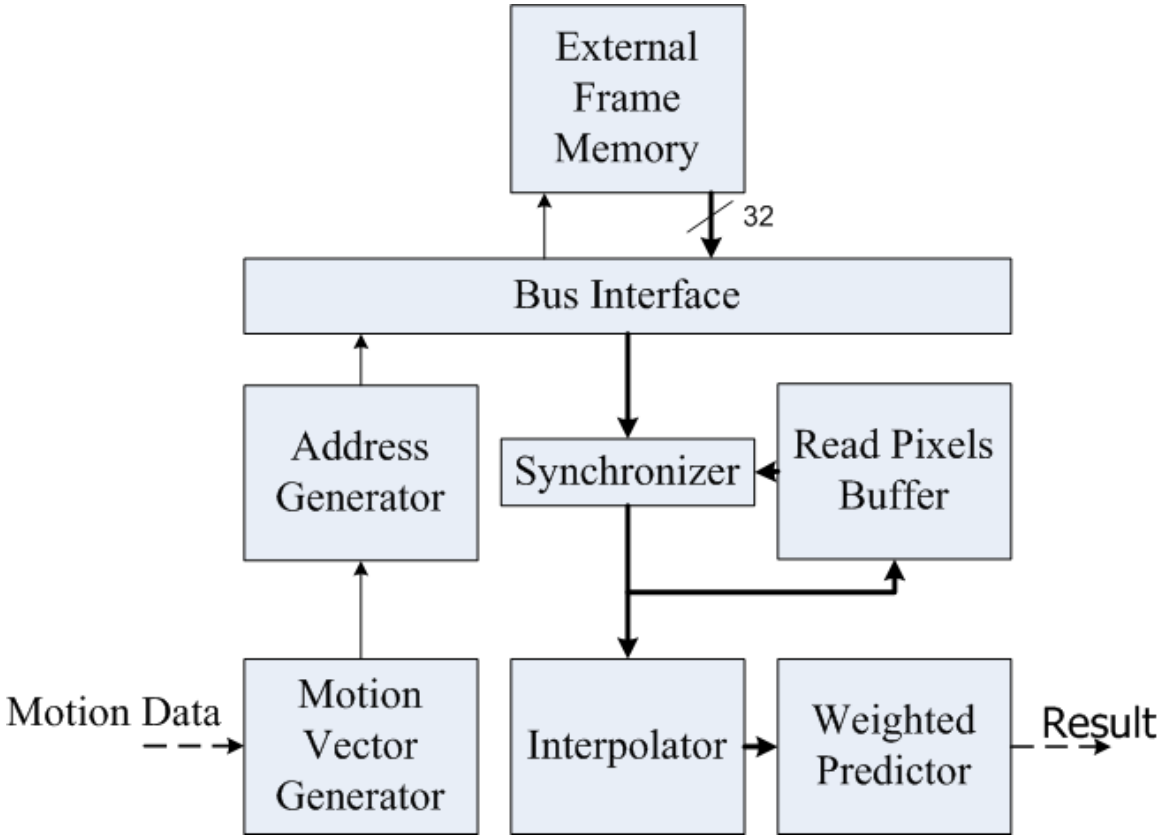

Fig 3.1 Motion compensation engine for H.264 video decoder

Fig 3.1 illustrates the whole motion compensation engine for H.264/AVC video decoder. Firstly, Motion vector generator generates motion vector according to motion data. Then, the address generator uses motion vector with reduction strategies of memory bandwidth to generate address of reference region. Moreover, transfer reference address to system memory controller (also named well-known arbiter). The tasking of memory access 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 pass through Weighted Predictor to produce motion compensation result. The

result will be added to the residual data and then pass through de-blocking filter. In our proposed decoder, ping-pong structured external frame memory [9], double memories stored reference and current frame reciprocally, is adopted.

The following subsection will discuss the detail of other modules except reduction strategies of memory bandwidth. The detailed discussion of reduction strategies of memory bandwidth are shown in Chapter 4. Subsection 3.2 illustrates motion vector generator (MVG) Supports Main/High Profile including motion vector predictor and the related storages. Subsection 3.3 combines luma and chroma interpolator design. Subsection .3.4 shows Weighted Predictor design. Finally, summary is presented in section 3.5

3.2 MVG support Main/High profile

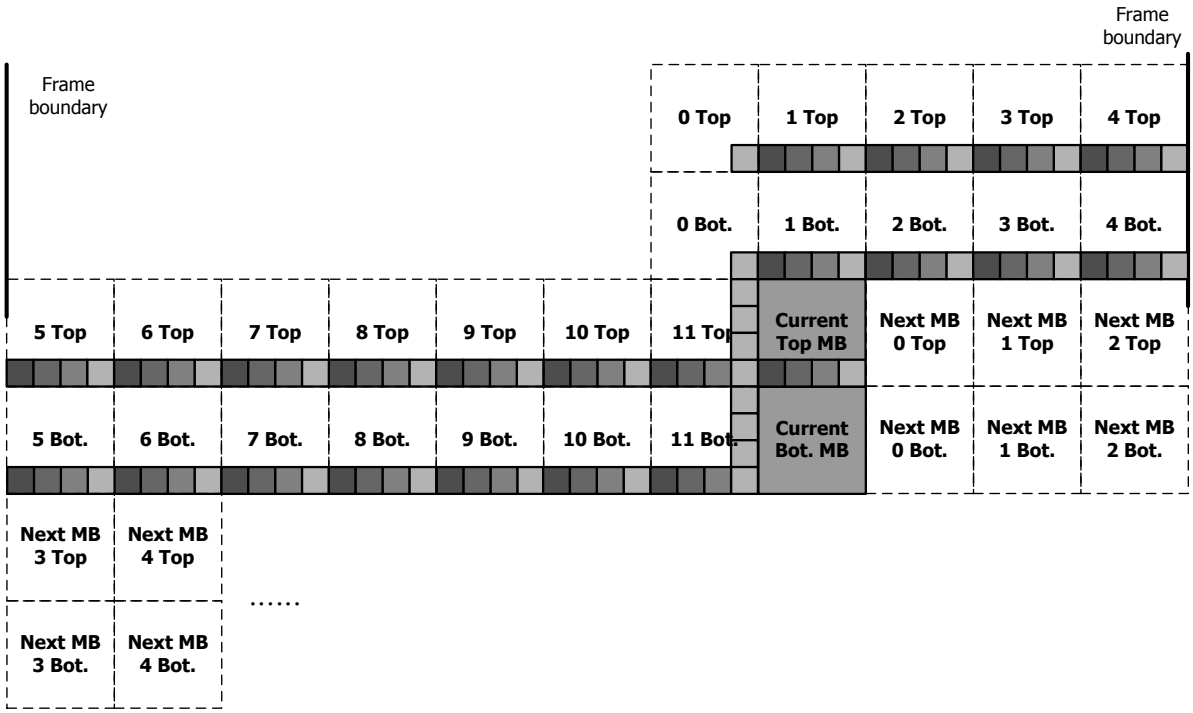

Fig 3.2 Motion vectors information storage for motion vector predictor for QCIF frame format.

There are two tools in MVG for supported Main/High profile. The first one is B slice type, which has double motion vectors. The second one is MBAFF mode. In MBAFF mode, the handle of macroblock is Macroblock pair. The same with P slice, the required total storage for motion vector generator, Fig 3.2 shows an example. Total amount of 4 x 11 x 2 both components of the motion vector have to be stored for QCIF frame format. Fig 3.3 (a) shows the detail of required neighboring motion vectors. To decode T0-T15 in current top MB, it needs neighboring motion vectors in left (TL0-TL3, MVL0-MVL3), above (TU0-TU3, MVU0-MVU3), above-right (TRU, MVRU), and above-left (TLU-MVLU) position. The 4 x 8 size of MV buffers is required because the maximum number of motion vector per MB pair is thirty-two. If we reuse the same 4 x 4 size of MV buffers and add a number of buffers (T10, T11, T14, and T15), the MV buffers can be further reduced. Fig 3.3 (b) shows the reduced version.

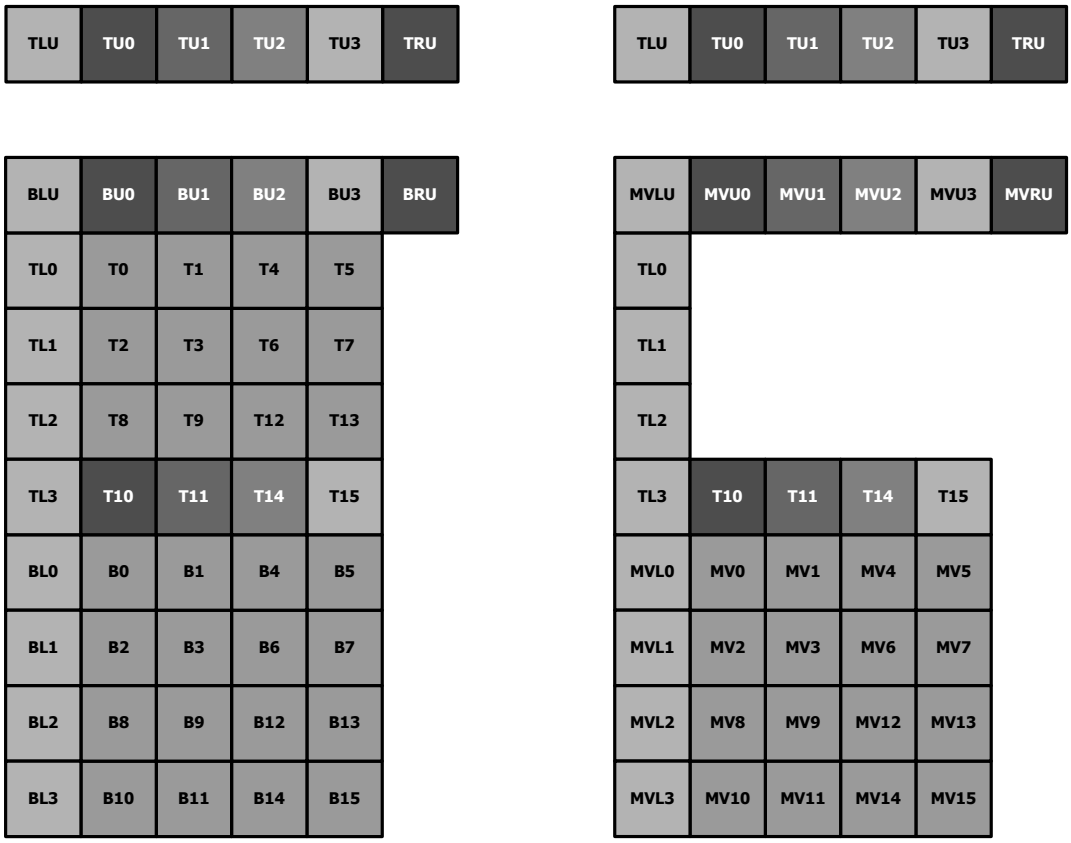

Fig 3.3 (a) Neighboring motion vectors needed when decoding all motion vectors in current MBAFF macroblock, (b) reduced and combined with non-MBAFF version

Fig 3.4 shows the detailed architecture of motion vector generator. This architecture combine non-MBAFF and MBAFF mode. When operation in non-MBAFF TX (with X being 5, 7, 13, 15, and so on) storages can be closed for saving power. The same with P slice, Table 3.1 lists all MVA, MVB, MVC, and MVD for different block size_position index. The difference is MBAFF mode not only size_position index but also current MB pair is Frame/Field coding, current MB is Top/Bottom MB, and relative MB pair is Frame/Filed coding. Therefore, LUT in MBAFF mode is eight times complexity than non-MBAFF mode. For cost and area efficiency consideration, we combine MBAFF and non-MBAFF LUT. Fortunately, we can find the condition of MVA, MVB, MVC, and MVD is the same with non-MBAFF mode when condition of MBAFF mode is fixed in current MB pair is Field, current MB is Bottom MB, and relative MB pair is Field. As mentioned above, we can use the

same LUT to deal with non-MBAFF and MBAFF mode.

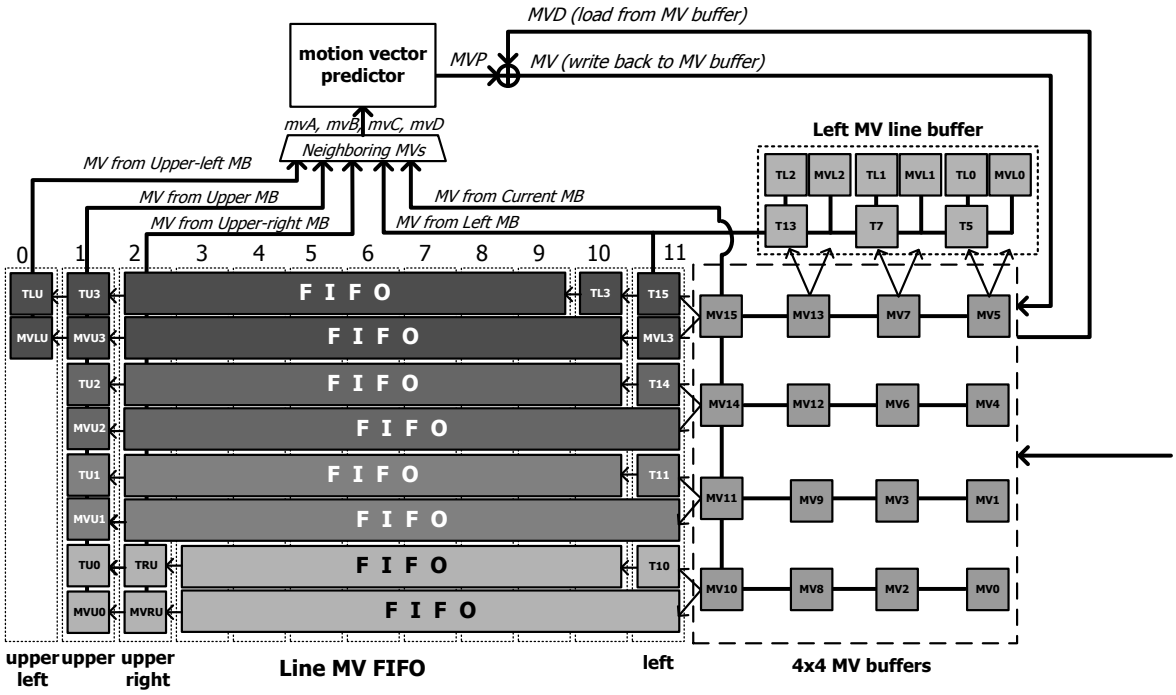

Fig 3.4 Motion vector generator architecture for QCIF-format support MBAFF

Table 3.1 Median prediction table in MBAFF frames

	current MB	T/B	relative MB	mvA	mvB	mvC	mvD
16x16	Frame	Top	Frame	TL0	BU0	BRU	BLU
			Field				
		Bot.	Frame	BL0	T10	X	TL3
			Field	TL2			BL1
	Field	Top	Frame	TL0	BU0	BRU	BLU
			Field		TU0	TRU	TLU
		Bot.	Fram	TL0	BU0	BRU	BLU
			Field	BL0			
16x8_0	Frame	Top	Frame	TL0	BU0	BRU	BLU
			Field				
		Bot.	Frame	BL0	T10	X	TL3
			Field	TL2			BL1
	Field	Top	Frame	TL0	BU0	BRU	BLU
			Field		TU0	TRU	TLU
		Bot.	Fram	TL0	BU0	BRU	BLU
			Field	BL0			
16x8_1	Frame	Top	Frame	TL2	MV2	X	TL1
			Field	TL1			BL0
		Bot.	Frame	BL2	MV2	X	BL1
			Field	TL3			BL2
	Field	Top	Frame	BL0	MV2	X	TL3
			Field	TL2			TL1
		Bot.	Fram	BL0	MV2	X	TL3
			Field	BL2			BL1
8x16_0	Frame	Top	Frame	TL0	BU0	BU2	BLU
			Field				
		Bot.	Frame	BL0	T10	T14	TL3
			Field	TL2			BL1
	Field	Top	Frame	TL0	BU0	BU2	BLU
			Field		TU0	TU2	TLU
		Bot.	Fram	TL0	BU0	BU2	BLU
			Field	BL0			
8x16_1	Frame	Top	Frame	MV1	BU2	BRU	BU1
			Field				
		Bot.	Frame	MV1	T14	X	T11
			Field				
	Field	Top	Frame	MV1	BU2	BRU	BU1
			Field		TU2	TRU	TU1
		Bot.	Fram	MV1	BU2	BRU	BU1
			Field				
8x8_0	Frame	Top	Frame	TL0	BU0	BU2	BLU
			Field				
		Bot.	Frame	BL0	T10	T14	TL3
			Field	TL2			BL1
	Field	Top	Frame	TL0	BU0	BU2	BLU
			Field		TU0	TU2	TLU
		Bot.	Fram	TL0	BU0	BU2	BLU
			Field	BL0			
8x8_1	Frame	Top	Frame	MV1	BU2	BRU	BU1
			Field				
		Bot.	Frame	MV1	T14	X	T11
			Field				
	Field	Top	Frame	MV1	BU2	BRU	BU1
			Field		TU2	TRU	TU1
		Bot.	Fram	MV1	BU2	BRU	BU1
			Field				
8x8_2	Frame	Top	Frame	TL2	MV2	MV6	TL1
			Field	TL1			BL0
		Bot.	Frame	BL2	MV2	MV6	BL1
			Field	TL3			BL2
	Field	Top	Frame	BL0	MV2	MV6	TL3
			Field	TL2			TL1
		Bot.	Fram	BL0	MV2	MV6	TL3
			Field	BL2			BL1
8x8_3	Frame	Top	Frame	MV9	MV6	X	MV3
			Field				
		Bot.	Frame	MV9	MV6	X	MV3
			Field				
	Field	Top	Frame	MV9	MV6	X	MV3
			Field				
		Bot.	Fram	MV9	MV6	X	MV3
			Field				

	current MB	T/B	relative MB	mvA	mvB	mvC	mvD
8x4_0	Frame	Top	Frame	TL0	BU0	BU2	BLU
			Field				
		Bot.	Frame	BL0	T10	T14	TL3
			Field	TL2			BL1
	Field	Top	Frame	TL0	BU0	BU2	BLU
			Field		TU0	TU2	TLU
		Bot.	Fram	TL0	BU0	BU2	BLU
			Field	BL0			
8x4_1	Frame	Top	Frame	TL1	MV0	X	TL0
			Field	TL0			BL0
		Bot.	Frame	BL1	MV0	X	BL0
			Field	TL2			BL2
	Field	Top	Frame	TL2	T0	X	TL1
			Field	TL1			TL0
		Bot.	Fram	TL2	B0	X	TL1
			Field	BL1			BL0
8x4_2	Frame	Top	Frame	MV1	BU2	BRU	BU1
			Field				
		Bot.	Frame	MV1	T14	X	T11
			Field				
	Field	Top	Frame	T1	BU2	BRU	BU1
			Field		TU2	TRU	TU1
		Bot.	Fram	MV1	BU2	BRU	BU1
			Field				
8x4_3	Frame	Top	Frame	MV3	MV4	X	MV1
			Field				
		Bot.	Frame	MV3	MV4	X	MV1
			Field				
	Field	Top	Frame	T3	T4	X	T1
			Field				
		Bot.	Fram	MV3	MV4	X	MV1
			Field				
8x4_4	Frame	Top	Frame	TL2	MV2	MV6	TL1
			Field	TL1			BL0
		Bot.	Frame	BL2	MV2	MV6	BL1
			Field	TL3			BL2
	Field	Top	Frame	BL0	T2	T6	TL3
			Field	TL2			TL1
		Bot.	Fram	BL0	MV2	MV6	TL3
			Field	BL2			BL1
8x4_5	Frame	Top	Frame	TL3	MV8	X	TL2
			Field	TL1			BL1
		Bot.	Frame	BL3	MV8	X	BL2
			Field	TL3			BL3
	Field	Top	Frame	BL2	T8	X	BL1
			Field	TL3			TL2
		Bot.	Fram	BL2	MV8	X	BL1
			Field	BL3			BL2
8x4_6	Frame	Top	Frame	MV9	MV6	X	MV3
			Field				
		Bot.	Frame	MV9	MV6	X	MV3
			Field				
	Field	Top	Frame	MV9	MV6	X	MV3
			Field				
		Bot.	Fram	MV9	MV6	X	MV3
			Field				
8x4_7	Frame	Top	Frame	MV11	MV12	X	MV9
			Field				
		Bot.	Frame	MV11	MV12	X	MV9
			Field				
	Field	Top	Frame	MV11	MV12	X	MV9
			Field				
		Bot.	Fram	MV11	MV12	X	MV9
			Field				
4x8_0	Frame	Top	Frame	TL0	BU0	BU1	BLU
			Field				
		Bot.	Frame	BL0	T10	T11	TL3
			Field	TL2			BL1
	Field	Top	Frame	TL0	BU0	BU1	BLU
			Field		TU0	TU1	TLU
		Bot.	Fram	TL0	BU0	BU1	BLU
			Field	BL0			

	current MB	T/B	relative MB	mvA	mvB	mvC	mvD
4x8_1	Frame	Top	Frame	MV0	BU1	BU2	BU0
			Field				
		Bot.	Frame	MV0	T11	T14	T10
			Field				
	Field	Top	Frame	MV0	BU1	BU2	BU0
			Field		TU1	TU2	TU0
		Bot.	Fram	MV0	BU1	BU2	BU0
			Field				
4x8_2	Frame	Top	Frame	MV1	BU2	BU3	BU1
			Field				
		Bot.	Frame	MV1	T14	T15	T11
			Field				
	Field	Top	Frame	MV1	BU2	BU3	BU1
			Field		TU2	TU3	TU1
		Bot.	Fram	MV1	BU2	BU3	BU1
			Field				
4x8_3	Frame	Top	Frame	MV4	BU3	BRU	BU2
			Field				
		Bot.	Frame	MV4	T15	X	T14
			Field				
	Field	Top	Frame	MV4	BU3	BRU	BU2
			Field		TU3	TRU	TU2
		Bot.	Fram	MV4	BU3	BRU	BU2
			Field				
4x8_4	Frame	Top	Frame	TL2	MV2	MV3	TL1
			Field	TL1			BL0
		Bot.	Frame	BL2	MV2	MV3	BL1
			Field	TL3			BL2
	Field	Top	Frame	BL0	MV2	MV3	TL3
			Field	TL2			TL1
		Bot.	Fram	BL0	MV2	MV3	TL3
			Field	BL2			BL1
4x8_5	Frame	Top	Frame	MV8	MV3	MV6	MV2
			Field				
		Bot.	Frame	MV8	MV3	MV6	MV2
			Field				
	Field	Top	Frame	MV8	MV3	MV6	MV2
			Field				
		Bot.	Fram	MV8	MV3	MV6	MV2
			Field				
4x8_6	Frame	Top	Frame	MV9	MV6	MV7	MV3
			Field				
		Bot.	Frame	MV9	MV6	MV7	MV3
			Field				
	Field	Top	Frame	MV9	MV6	MV7	MV3
			Field				
		Bot.	Fram	MV9	MV6	MV7	MV3
			Field				
4x8_7	Frame	Top	Frame	MV12	MV7	X	MV6
			Field				
		Bot.	Frame	MV12	MV7	X	MV6
			Field				
	Field	Top	Frame	MV12	MV7	X	MV6
			Field				
		Bot.	Fram	MV12	MV7	X	MV6
			Field				
4x4_0	Frame	Top	Frame	TL0	BU0	BU1	BLU
			Field				
		Bot.	Frame	BL0	T10	T11	TL3
			Field	TL2			BL1
	Field	Top	Frame	TL0	BU0	BU1	BLU
			Field		TU0	TU1	TLU
		Bot.	Fram	TL0	BU0	BU1	BLU
			Field	BL0			
4x4_1	Frame	Top	Frame	MV0	BU1	BU2	BU0
			Field				
		Bot.	Frame	MV0	T11	T14	T10
			Field				
	Field	Top	Frame	MV0	BU1	BU2	BU0
			Field		TU1	TU2	TU0
		Bot.	Fram	MV0	BU1	BU2	BU0
			Field				

	current MB	T/B	relative MB	mvA	mvB	mvC	mvD
4x4_2	Frame	Top	Frame	TL1	MV0	MV1	TL0
			Field	TL0			BL0
		Bot.	Frame	BL1	MV0	MV1	BL0
			Field	TL2			BL2
	Field	Top	Frame	TL2	MV0	MV1	TL1
			Field	TL1			TL0
		Bot.	Fram	TL2	MV0	MV1	TL1
			Field	BL1			BL0
4x4_3	Frame	Top	Frame	MV2	MV1	X	MV0
			Field				
		Bot.	Frame	MV2	MV1	X	MV0
			Field				
	Field	Top	Frame	MV2	MV1	X	MV0
			Field				
		Bot.	Fram	MV2	MV1	X	MV0
			Field				
4x4_4	Frame	Top	Frame	MV1	BU2	BU3	BU1
			Field				
		Bot.	Frame	MV1	T14	T15	T11
			Field				
	Field	Top	Frame	MV1	BU2	BU3	BU1
			Field		TU2	TU3	TU1
		Bot.	Fram	MV1	BU2	BU3	BU1
			Field				
4x4_5	Frame	Top	Frame	MV4	BU3	BRU	BU2
			Field				
		Bot.	Frame	MV4	T15	X	T14
			Field				
	Field	Top	Frame	MV4	BU3	BRU	BU2
			Field		TU3	TRU	TU2
		Bot.	Fram	MV4	BU3	BRU	BU2
			Field				
4x4_6	Frame	Top	Frame	MV3	MV4	MV5	MV1
			Field				
		Bot.	Frame	MV3	MV4	MV5	MV1
			Field				
	Field	Top	Frame	MV3	MV4	MV5	MV1
			Field				
		Bot.	Fram	MV3	MV4	MV5	MV1
			Field				
4x4_7	Frame	Top	Frame	MV6	MV5	X	MV4
			Field				
		Bot.	Frame	MV6	MV5	X	MV4
			Field				
	Field	Top	Frame	MV6	MV5	X	MV4
			Field				
		Bot.	Fram	MV6	MV5	X	MV4
			Field				
4x4_8	Frame	Top	Frame	TL2	MV2	MV3	TL1
			Field	TL1			BL0
		Bot.	Frame	BL2	MV2	MV3	BL1
			Field	TL3			BL2
	Field	Top	Frame	BL0	MV2	MV3	TL3
			Field	TL2			TL1
		Bot.	Fram	BL0	MV2	MV3	TL3
			Field	BL2			BL1
4x4_9	Frame	Top	Frame	MV8	MV3	MV6	MV2
			Field				
		Bot.	Frame	MV8	MV3	MV6	MV2
			Field				
	Field	Top	Frame	MV8	MV3	MV6	MV2
			Field				
		Bot.	Fram	MV8	MV3	MV6	MV2
			Field				
4x4_10	Frame	Top	Frame	TL3	MV8	MV9	TL2
			Field	TL1			BL1
		Bot.	Frame	BL3	MV8	MV9	BL2
			Field	TL3			BL3
	Field	Top	Frame	BL2	MV8	MV9	BL1
			Field	TL3			TL2
		Bot.	Fram	BL2	MV8	MV9	BL1
			Field	BL3			BL2

	current MB	T/B	relative MB	mvA	mvB	mvC	mvD
4x4_11	Frame	Top	Frame	MV10	MV9	X	MV8
			Field				
		Bot.	Frame	MV10	MV9	X	MV8
			Field				
	Field	Top	Frame	MV10	MV9	X	MV8
			Field				
		Bot.	Fram	MV10	MV9	X	MV8
			Field				
4x4_12	Frame	Top	Frame	MV9	MV6	MV7	MV3
			Field				
		Bot.	Frame	MV9	MV6	MV7	MV3
			Field				
	Field	Top	Frame	MV9	MV6	MV7	MV3
			Field				
		Bot.	Fram	MV9	MV6	MV7	MV3
			Field				
4x4_13	Frame	Top	Frame	MV12	MV7	X	MV6
			Field				
		Bot.	Frame	MV12	MV7	X	MV6
			Field				
	Field	Top	Frame	MV12	MV7	X	MV6
			Field				
		Bot.	Fram	MV12	MV7	X	MV6
			Field				
4x4_14	Frame	Top	Frame	MV11	MV12	MV13	MV9
			Field				
		Bot.	Frame	MV11	MV12	MV13	MV9
			Field				
	Field	Top	Frame	MV11	MV12	MV13	MV9
			Field				
		Bot.	Fram	MV11	MV12	MV13	MV9
			Field				
4x4_15	Frame	Top	Frame	MV14	MV13	X	MV12
			Field				
		Bot.	Frame	MV14	MV13	X	MV12
			Field				
	Field	Top	Frame	MV14	MV13	X	MV12
			Field				
		Bot.	Fram	MV14	MV13	X	MV12
			Field				

As for B slice, we can use hardware sharing to process twice mvp for B slice type because motion vector prediction of LIST_1 can be hidden below data-read cycles of LIST_0 from frame memory. However, it is not only process twice but also need consider many extra conditions. For example, Fig 2.8 shows one partition predicted by L0 direction, and neighboring partition predicted by L1 direction. When predicting direction is different, the neighboring MV cannot be used to predict current MV. Here, we do not discuss them for clarity.

In addition to considered predicting direction, B slice has new direct mode. There are two direct modes in B slice, one is spatial direct mode (SDM) and the other one is temporal direct mode (TDM). The challenge of direct mode in B slice is to find where the co-located

macroblock is and where the co-located partition is. Because current picture and co-located picture can be field, frame, and MBAFF coding types. Therefore, both of the co-located macroblock and co-located partition determine formula is about eight kinds and these determine formula will involve multiplier, divider, and remainder, which are high complexity component. However, if we use macroblock coordinate (x and y) which originally transferred from system to motion compensation unit to find co-located macroblock/partition. We can significantly reduce complexity. Table 3.2 shows the mapping table of co-located macroblock after coordinated method reduction. The Y means y-axis. Table 3.3 shows the mapping table of co-located partition. Fig 3.5 shows the entire motion compensation architecture.

Table 3.2 Co-located macroblock table

Curr	Col	Original equation	New equation
FLD	FRM	2 * PicWidthInMbs * (CurrMbAddr / PicWidthInMbs) + (CurrMbAddr % PicWidthInMbs) + PicWidthInMbs * (yCol / 8)	Y<<1 +blk_Num[3]
FLD	AFRM -FRM	2 * CurrMbAddr + (yCol / 8)	Y<<1 +blk_Num[3]
	AFRM -FLD	2 * CurrMbAddr + bottom_field_flag	Y<<1 +bottom_field_flag
FRM	FLD	PicWidthInMbs * (CurrMbAddr / (2 * PicWidthInMbs)) + (CurrMbAddr % PicWidthInMbs)	Y>>1
AFRM	FLD	CurrMbAddr / 2	Y>>1
AFRM -FRM	AFRM -FLD	2 * (CurrMbAddr / 2) + ((topAbsDiffPOC < bottomAbsDiffPOC) ? 0 : 1)	Y[0]=0
AFRM -FLD	AFRM -FRM	2 * (CurrMbAddr / 2) + (yCol / 8)	Y[0]=0 +blk_num[3]
Other		CurrMbAddr	Don’t change

Table 3.3 Co-located partition table

Current	Co-located	Original equation	New equation
FLD	FRM	(2 * yCol) % 16	**y=0/8 =>0** **y=4/12 =>8**
FLD	AFRM-FRM	(2 * yCol) % 16	
FRM	FLD	8 * ((CurrMbAddr / PicWidthInMbs) % 2) + 4 * (yCol / 8)	**Y[0]=0** **y=0/4 =>0** **y=8 12 =>4** **Y[0]!=0** **y=0 /4 =>8** **y=8/12 =>12**
AFRM -FRM	FLD	8 * (CurrMbAddr % 2) +4 * (yCol / 8)	**X[0]=0** **y=0/4 =>0** **y=8/12 =>4** **X[0]!=0** **y=0/4 =>8** **y=8/12 =>12**
AFRM -FRM	AFRM -FLD	8 * (CurrMbAddr % 2) + 4 * (yCol / 8)	
AFRM -FLD	AFRM -FRM	(2 * yCol) % 16	**y=0/8 =>0** **y=4/12 =>8**
Other		yCol	**Don't change**

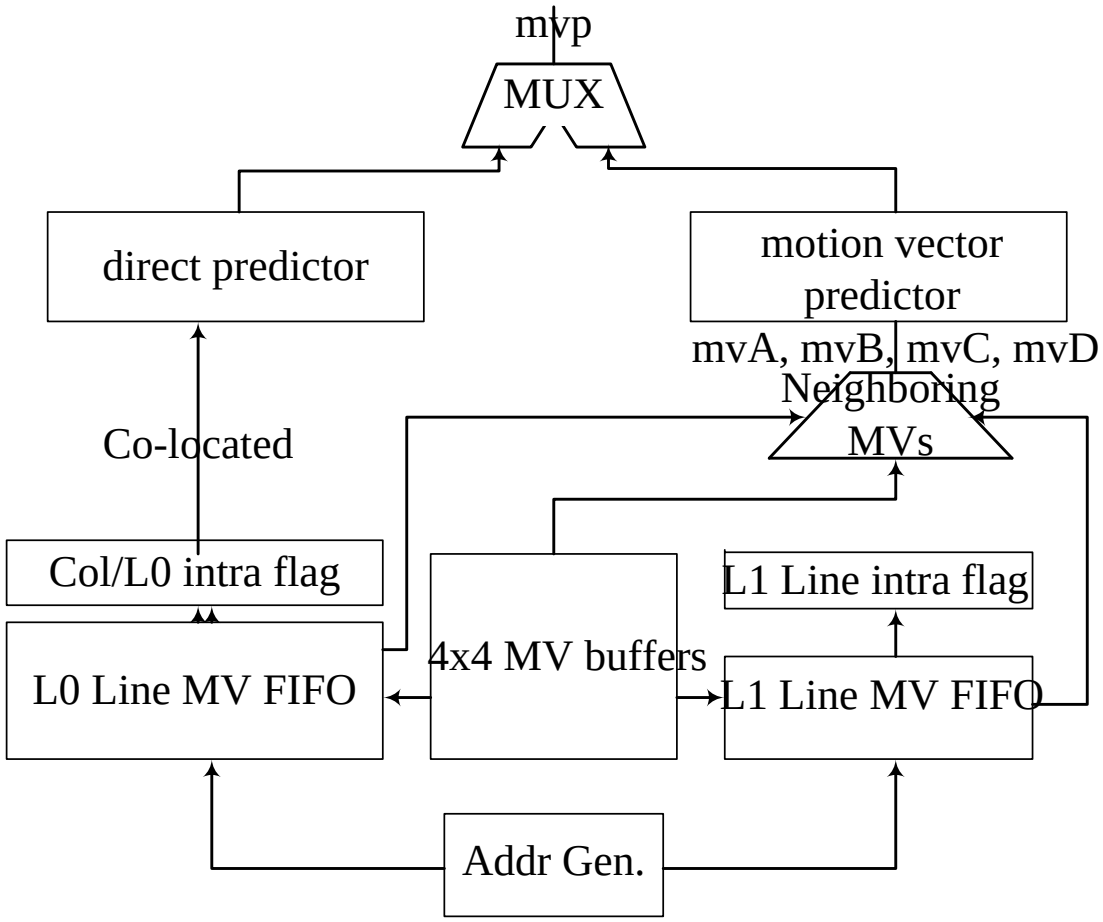

Fig 3.5 Motion vector generator architecture

3.3 Interpolator Design

3.3.1 Luma Interpolator Design

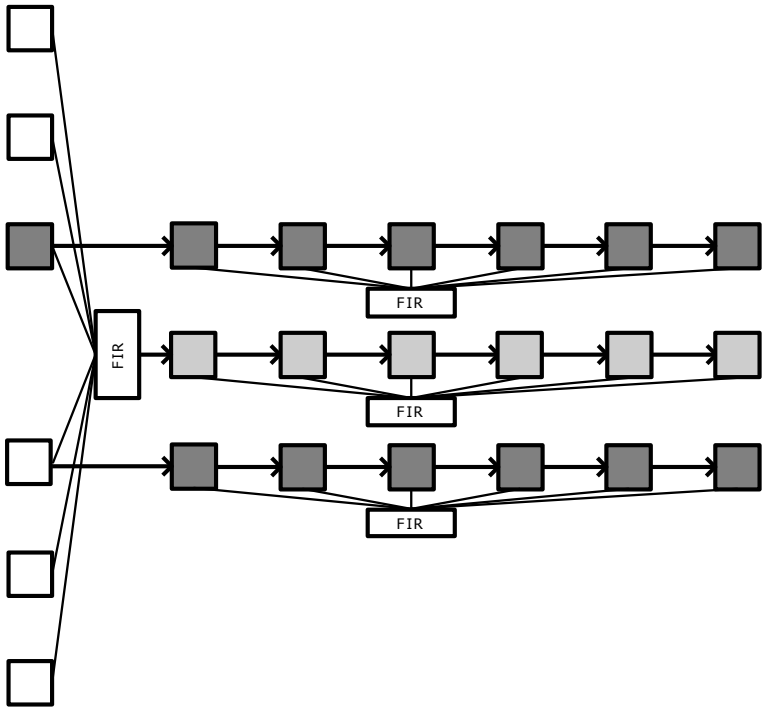

Fig 3.6 Separate 1-D interpolator design (no parallel)

In this subsection, several different interpolator designs will be presented. Reviewing the fractional pixel interpolation for H.264/AVC in Fig 2.5, 6-tap FIR with (1, -5, 20, 20, -5, 1) coefficient and bilinear filter are needed for half and quarter pixel interpolation. For cost and area efficiency consideration, Li's and Shen's interpolator filter unit and two-stage recursive algorithm is proposed in [10] and [11]. These designs are area efficiency and suitable for P slice. However, as for B slice, throughput is a very important issue and long execution cycles

in these designs cause the real-time of video decoding cannot be meted.

Oppositely, consider throughput and standard-compatible design, Chien's [4] proposed separate 1-D design that separates horizontal and vertical interpolation and processes in parallel based on 4 x 4 block size. This design owns better throughput, although it may need more storages. Fig 3.6 shows separate 1-D interpolator design without processing in parallel.

Table 3.3 Comparison of execution cycles for different architectures

Architecture	Ideal execution cycles
Shen's and Li's desing	13
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 Shen's and Li's design, the result outputs depend on fractional pixel positions. For a, b, c, d, h, and n position 4 clock cycles are needed to finish one 4x4 block. For e, g, p, and r, it takes 8 cycles to finish one 4 x 4 block interpolator. For f, j, q, i, and k, the cycles to finish one 4x4 block are 13 cycles which detailed operation is described in Li's [10] and Shen's [4]. As for separate 1-D design, the first data outputs at the 6^{th} 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. Finally, 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.

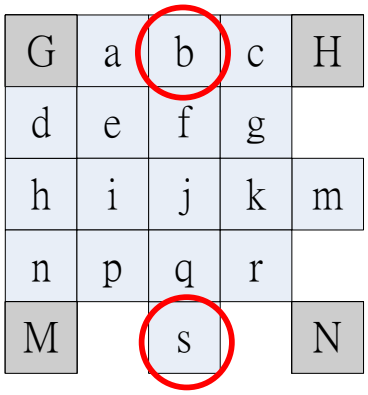

Fig 3.7 Only one half pixel is needed

Fig 3.8 shows original 4-parallel separate 1-D luma interpolator. For cost consideration, multiplier in FIR can be simplified to adders and shifters. We will discuss FIR design later. Because the original 4-parallel separate 1-D interpolator produces b and s half pixels at the same time for produce any position fractional pixel. However, either b or s half pixels is needed when produce interpolated pixel. If we check MV, we can know which half is needed after all. Therefore, we can modify 4-parallel separate 1-D interpolator to reduce the path storages and one FIR. The similar design can be seen in [12][13] and [14], but these designs require four multiplexers and we require only one multiplexer. Fig 3.9 shows the enhance 4-parallel separate 1-D interpolator.

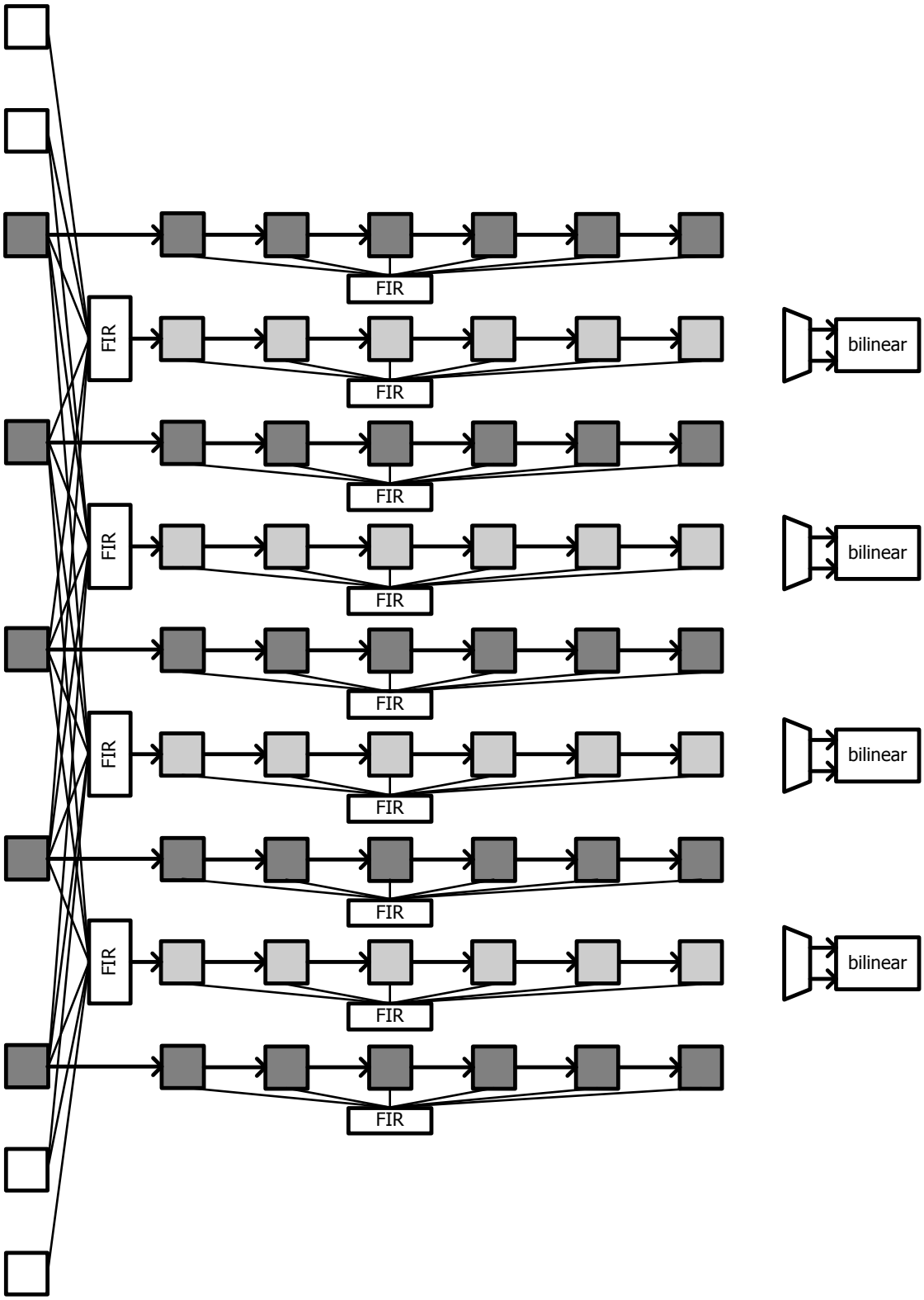

Fig 3.8 Original 4-parallel separate 1-D luma interpolator

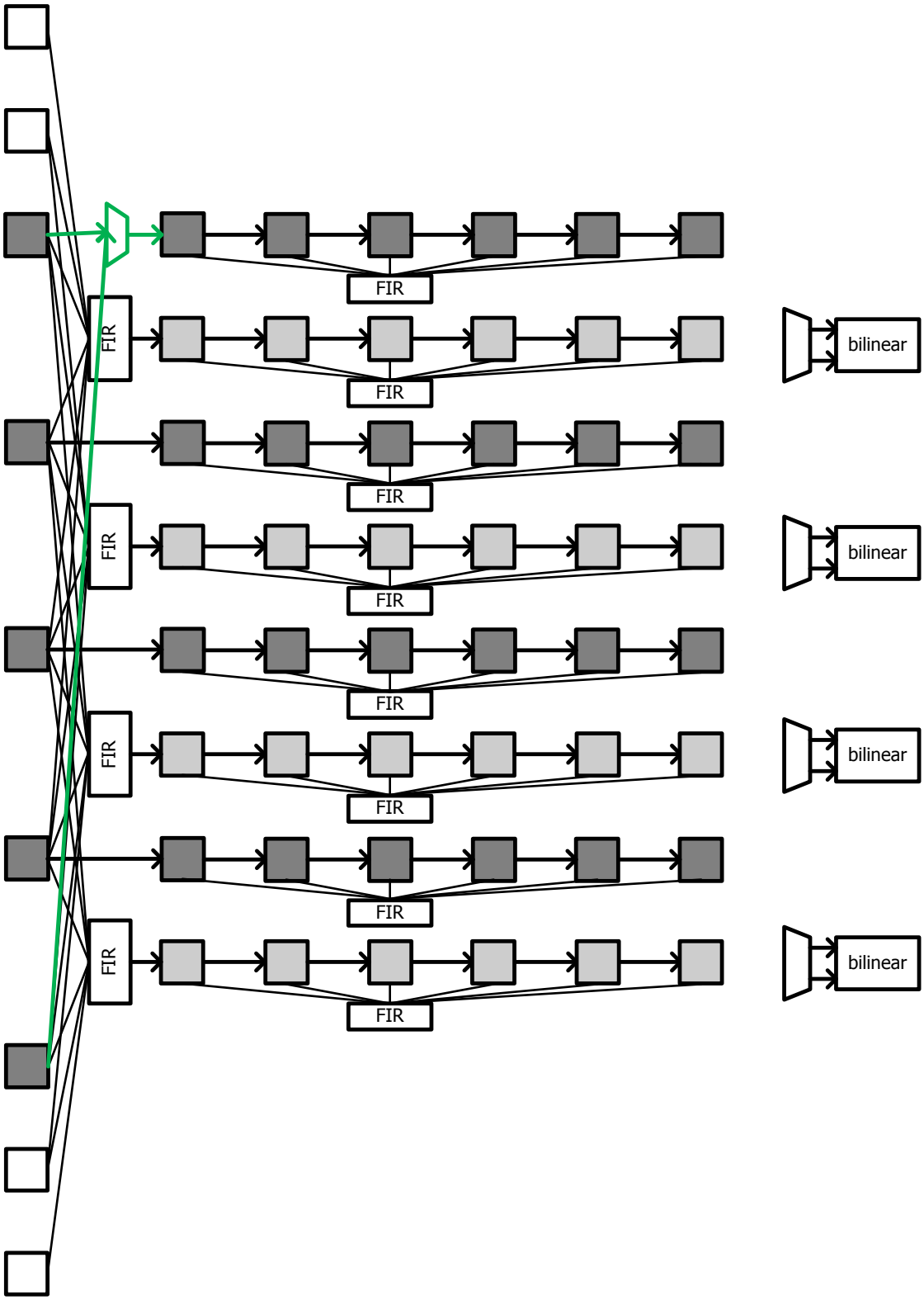

Fig 3.9 Enhance 4-parallel separate 1-D luma interpolator

3.3.2 Chroma Interpolator Design

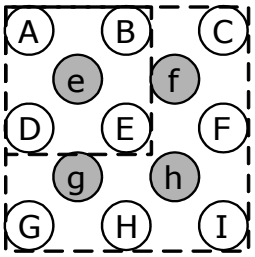

Fig 3.10 Interpolation window for each 2 x 2 chroma block

$$i = (8 - xFrac\ ) * (8 - yFrac\ ) * A + xFrac\ * (8 - yFrac\ ) * B + (8 - xFrac\ ) * yFrac\ * C + xFrac\ * yFrac\ * D = (8 - xFrac\ ) * [(8 - yFrac\ ) * A + yFrac\ * C] + xFrac\ * [(8 - yFrac\ ) * B + yFrac\ * D] \quad \text{Eq. 3.1}$$

Because of 4:2:0 chroma format and quarter precision of luma inter prediction, chroma inter prediction displacement can achieve one-eighth motion accuracy. Chroma inter prediction must process based on 2 x 2 block size when luma inter prediction process based on 4 x 4 block size. Chroma interpolation requires 3 x 3 interpolated data for each 2 x 2 block as shown in Fig 3.10. 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 one-eighth. Compared with direct mapping design with 8 multipliers which equation is listed in Fig 2.5 (c), we rewrite the equation listed in Eq. 3.1 and the number of multiplier number can be reduced to 4.

$$i = (8 - xFrac) * [(8 - yFrac) * A + yFrac * C] + xFrac * [(8 - yFrac) * B + yFrac * D] \quad \text{Eq. 3.2}$$
$$= (8 - Frac) * M + Frac * N$$
$$M = N = (8 - Frac) * O + Frac * P$$

We can also rewrite the equation listed in Eq. 3.2. The Frac, O, and P are any corresponding value in Eq. 3.2. We can find as luma interpolator, chroma interpolator can

separate into horizontal and vertical filter. The corresponding separate 1-D design is illustrated in Fig 3.11 (a) and the vertical / horizontal filter is illustrated in Fig 3.11 (b). 2-parallel separate 1-D chroma interpolator 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 data are ready and they can be accessed randomly. Based on 2-parallel separate 1-D chroma interpolator design illustrated in Fig 3.12, only one cycle latency is required.

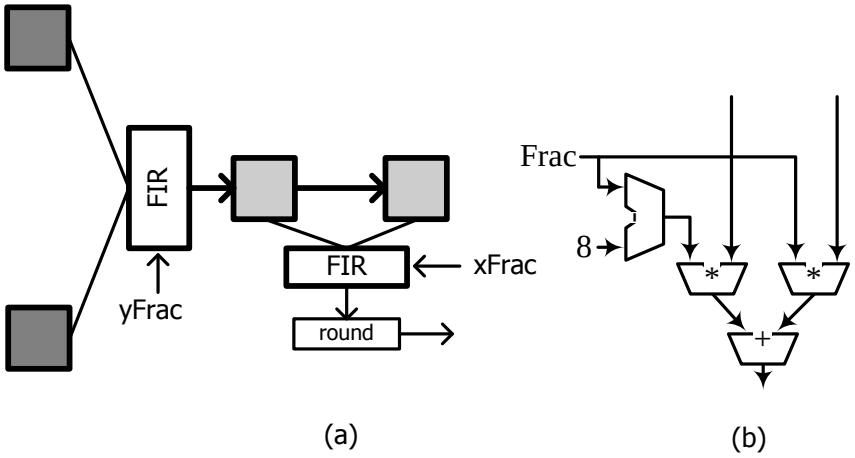

Fig 3.11 (a) Chroma interpolator, (b) vertical/horizontal filter

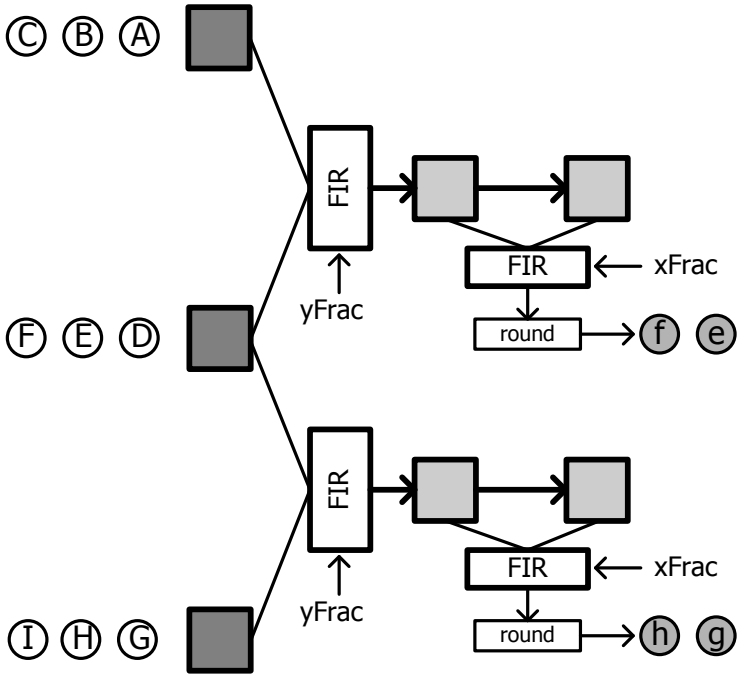

Fig 3.12 2-parallel chroma interpolator

3.3.3 Combine Luma and Chroma FIR Design

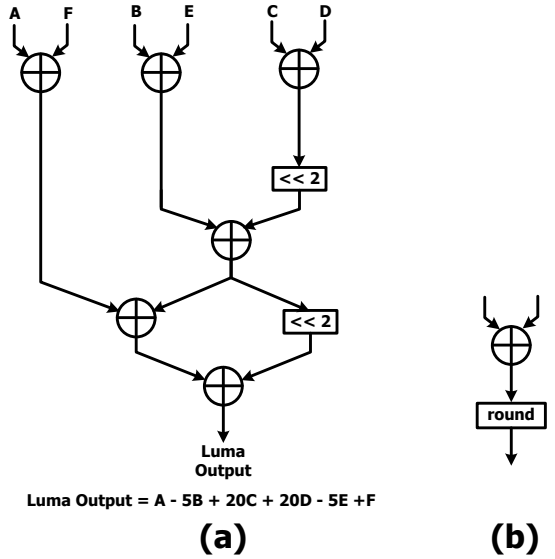

Fig 3.13 (a) Luma FIR design in Chen's [3], (b) bilinear filter

Especially note that luma and chroma interpolation for H.264/AVC are different. 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. As luma and chroma interpolator filter described in above, the adder and shifter can be shared when the architecture of chroma horizontal/vertical filter in Fig 3.11 (b) restructure to adder and shifter. Besides, we can further reduce critical path by merge rounding stage. The combined interpolator design is shown in Fig 3.14 and the cost penalty is MUX x 2 and bitwise AND x 6 when compared with the FIR design proposed in Chen's [3] and shown in Fig 3.13. Fig 3.15 illustrates the decoding path of luma FIR filter and chroma

horizontal/vertical filter. Because chroma interpolation for H.264/AVC is 2 x 2 block size basis, only eight luma FIR filters are required to replace with combined luma/chroma interpolators. Fig 3.16 indicates the entire interpolator architecture for H.264/AVC.

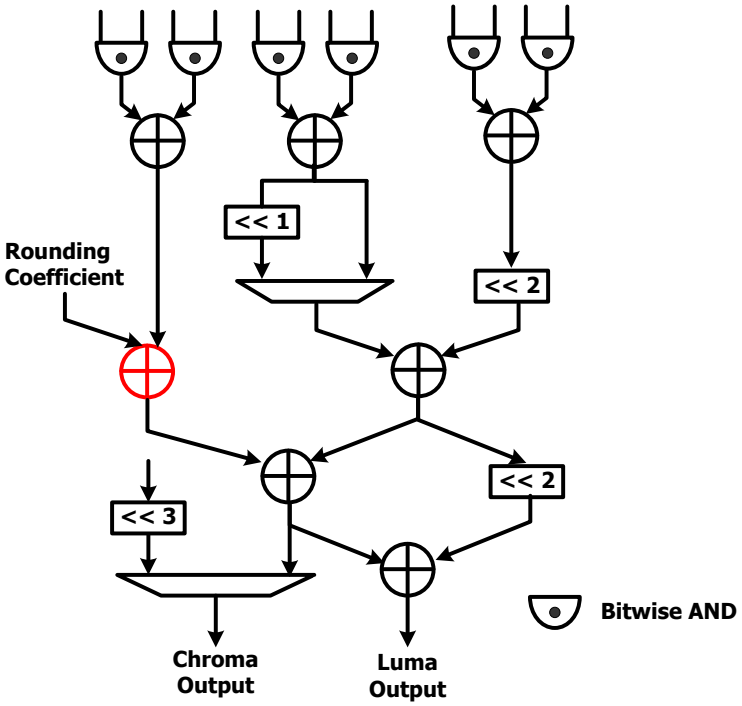

Fig 3.14 Combined luma/chroma interpolator design for H.264

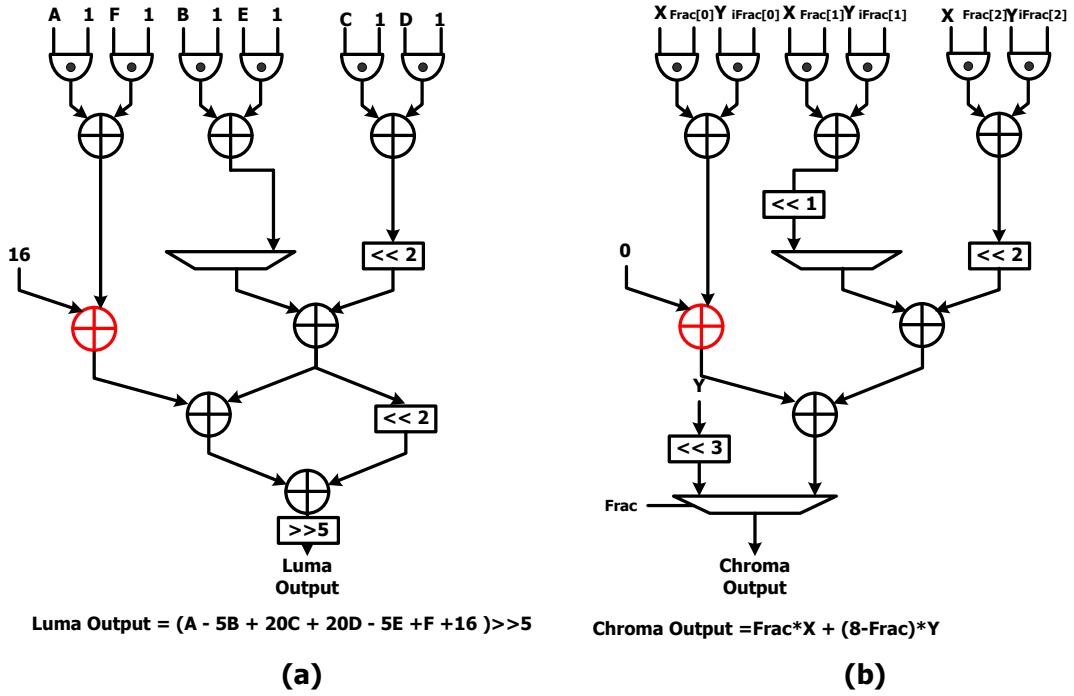

Fig 3.15 (a) Path of luma FIR interpolator, (b) path of chroma 1/8 bilinear

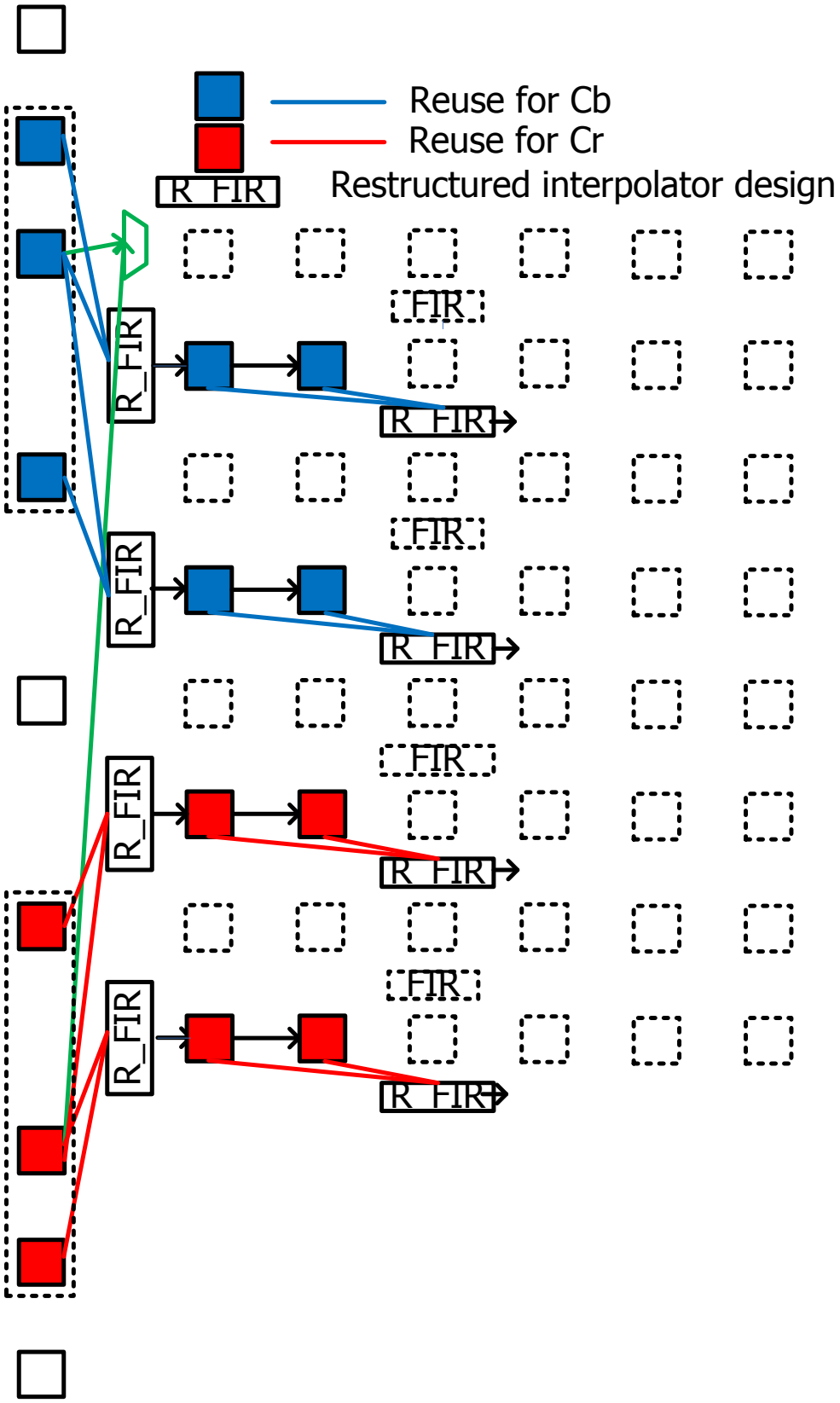

Fig 3.16 Entire interpolator architecture

3.3.4 Cost Analysis

Table 3.2 Comparison of requisite modules

	Wang's [15] ISCAS'05	Chen's [16] ICASSP'06	Li's [10] ISCAS'07	Tsai's[14] MWSCAS'05	Shen's [11] ICME'09	Proposed
FIR	13	12	4	12	4	12
Bilinear	2	12	4	4	4	0
Technology (um)	0.18	0.18	0.18	0.18	0.18	0.09
Gate count	20,686	15,000	13,027	21,506	11,823	13,201
Working Frequency (MHz)	100	150	100	125	100	100
Latency (Cycles/MB) luma+chroma	560	320	304	144+NA	288+NA	144+48

Because of multipliers of 6-tap filter are simplified to adders and shifters in all references. Therefore, in literature [10] and [11] use hardware sharing 6-tap FIRs to compute twice to reduce area cost in interpolator design. However, throughput is a very important issue and long execution cycles in interpolator design lead to not enough throughput in B slice. Our restructured interpolator combines luma and chroma filter and through determine MV to reduce a filter and one-path storages in traditional design. Table 3.2 lists the comparisons between our restructured interpolator design and other design. It shows our interpolator can almost achieve as gate count of [10] and [11] and owns enough throughputs although it requires paying some control overhead to support multi-mode operations.

3.4 Weighted Prediction

$$p = ((pL0 * W_0 + pL1 * W_1 + 2^{\log WD}) >> (\log WD + 1)) + ((o_0 + o_1 + 1) >> 1) \quad \text{Eq. 3.3}$$

$$p = \{[((pL0 * W_0 + 2^{\log WD - 1}) >> \log WD) + o_0] + [((pL1 * W_1 + 2^{\log WD - 1}) >> \log WD) + o_1] + 1\} >> 1) \quad \text{Eq. 3.4}$$

$$p = (pL0 * W_0 + 2^{\log WD - 1}) >> \log WD) + o_0$$
$$p = \{pL0 * W_0 + [2^{\log WD - 1} + (o_0 << \log WD)]\} >> \log WD \quad \text{Eq. 3.5}$$

Weighted prediction is the final stage of motion compensation behind the interpolator. Weighted prediction is a tool of scaling motion compensated samples to increase the video quality in H.264/AVC video decoding. In this subsection, weighted predictor architecture is proposed to collocate with interpolator and eliminate the latency overhead. Chen's [16] proposed weighted prediction architecture has low complexity. However, it has long critical path and large memory requirement (1.5kb). The design of Azevedo's [12] weighted predictor is simply implemented by direct mapping design and require an embedded memory to store rounding coefficient. Compared with direct mapping design which equation is listed in Eq. 3.3, we can use the same predictor twice to generate predicted value, first is LIST_0 prediction and second is LIST_1 prediction as shown in Eq. 3.4. The component of rounding and offset can be advanced and combined in the same stage. Therefore, the predictor can be further modified to reduce the critical path as shown in Eq. 3.5. Moreover from Eq. 3.5, the W_0 means weight factor and the value depend on weight flag from bit-stream. When weight flag is equal to 1, the value of weight factor shall be in the range of -128 to 127, inclusive.

When weight flag is equal to 0, weight factor shall be in the range of 2^0 to 2^7, inclusive [1]. From the above discussion, if we determine the highest weight factor two bit we can use an eight bits multiplier and shifter instead of a nine bits multiplier. The predictor is shown in Fig 3.17.

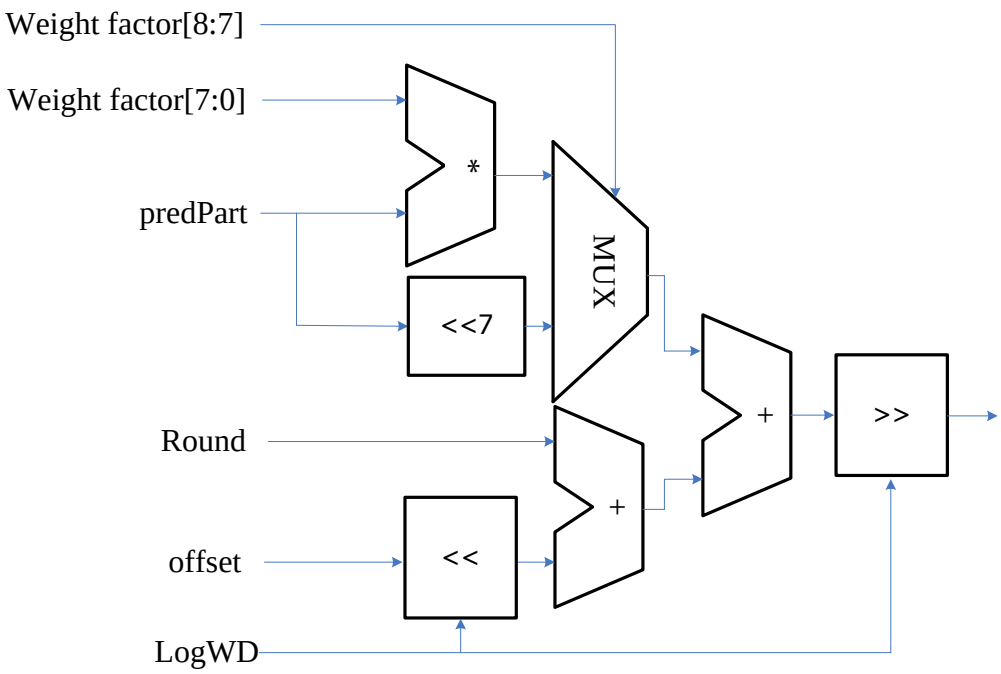

Fig 3.17 Weighted predictor design

Moreover, when B slice is involved, we use hardware sharing to operate twice. In addition, a 4 x 4 storages array is required to store intermediate results. Fig 3.18 illustrates the complete weighted predictor design. The same as temporal direct mode in motion vector generator, weighted predictor has implicit mode which weighting factors are calculated based on the relative temporal positions of LIST_0 and LIST_1 reference picture. Weighting factor in the implicit mode is derived from temporal direct mode data-path in order to reduce hardware cost. Furthermore, divider occupies the main area cost and computation time in the temporal direct mode design. We can use loop-up table (LUT) to replace divider because the

dividend is a constant value. Table 3.2 lists the comparison for implementation results. For [12], it was not presented in comparison because lack of related detail information.

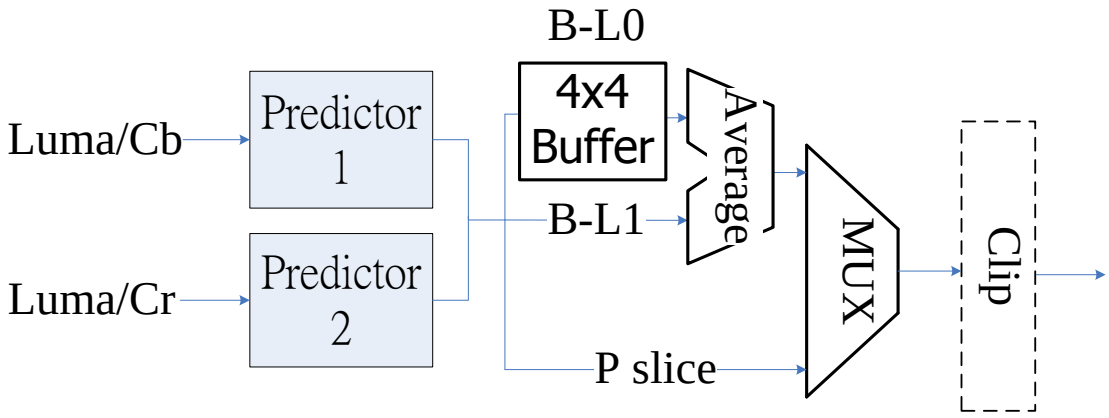

Fig 3.18 Entire weight predictor architecture

	ICASSP'06[16]	proposed
Multiplier (bits)	9	8
Technology	.18um	.90um
Gate count	12,960	6,412
Working frequency	87MHz	100MHz

Table 3.2 Comparison of execution cycles for different architectures

3.5 Summary

In this chapter, a motion compensation engine for H.264/AVC Main/High Profile decoder is presented. As for sharing design issue for multi-profile, our MVG use the same module and storages to deal with P slice and B slice which include MBAFF and non MBAFF. Our restructured interpolator presents the area efficiently compared with traditional design and it is suitable for high throughput application such as coded in B slice video decoder. Besides, the weighted predictor through hardware sharing with temporal direct mode and critical path shorten to achieve area efficiency. When weighted predictor collocates with interpolator, it only requires one cycle latency overhead.

Chapter 4
Memory Bandwidth Reduction

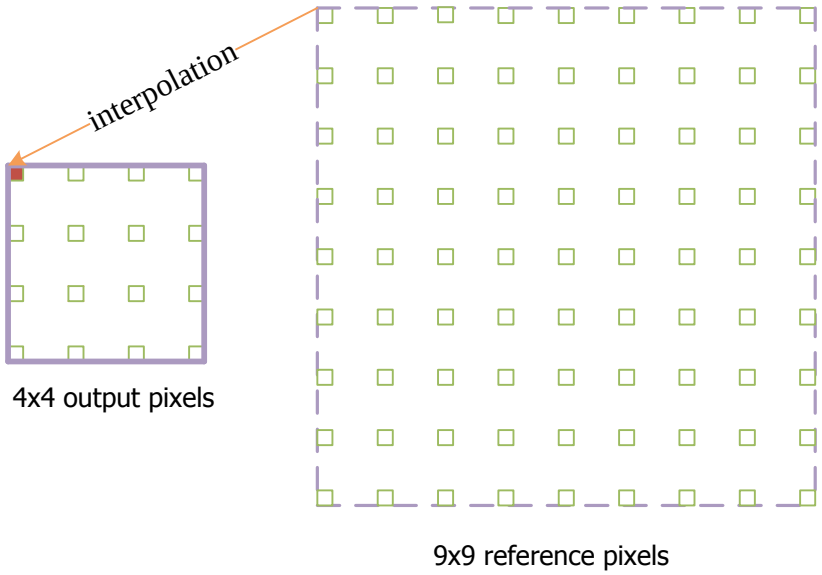

Fig 4.1 4 x 4 block window and the corresponding 9 x 9 interpolation window

Considering luma interpolation, the half position samples interpolated by applying 6-tap FIR filter and quarter position samples performed by applying using bilinear filter. It means interpolator needs six reference pixels to produce one interpolated pixel. Fig 4.1 shows to interpolate each fractional sample value for each 4 x 4 block size; it needs 9 x 9 interpolation window. Chroma interpolation, of which concept is similar to luma, interpolates each fractional sample value for each 2 x 2 block size, it needs 3 x 3 interpolation windows. When frame size is large and frame rate is high, interpolation causes heavy loading of memory bandwidth. Moreover, motion compensation involves Main/High Profile; it supports B slices in which reference frame from one direction increase to two directions. From the above discussion, Main/High Profile doubles the memory bandwidth requirement. In worst case,

interpolator needs memory bandwidth requirement, 398MB/s in P slices and 796MB/s in B slices, when support 1080 HD @ 30 fps. The heavy loading of memory bandwidth also means huge power consumption for bus activity and data operation.

The rest of this chapter is organized as follows. Firstly, section 4.1 discusses our reduction strategies of memory bandwidth. In addition, an analysis of bandwidth reduction limit is presented in section 4.2. Finally, summary is given in section 4.3.

4.1 Reduction strategies of memory bandwidth

Memory bandwidth always dominates the performance of entire video decoder. Several methods have been proposed to reduce the required memory bandwidth and they can be mainly classified to two directions, first one is frame recompression and another one is redundancy reduction of pixels transmission. With regard to the frame recompression, Fig 4.2 illustrates the concept. Frame data will be compressed before writing to frame memory, and reference frame data will be decompressed before reading into video decoder. However, frame recompression method must consider many issues which like necessary random access capability demanded from motion compensation, low complexity property due to area cost and power saving, and minimize required additional execution cycles to compress/decompress data such that meet the real time throughput requirement of video decoder. Here we do not go into detail because our system have two dedicated modules, embedded compressor, between motion compensation and frame memory and embedded decompressor between frame memory and de-blocking module respectively.

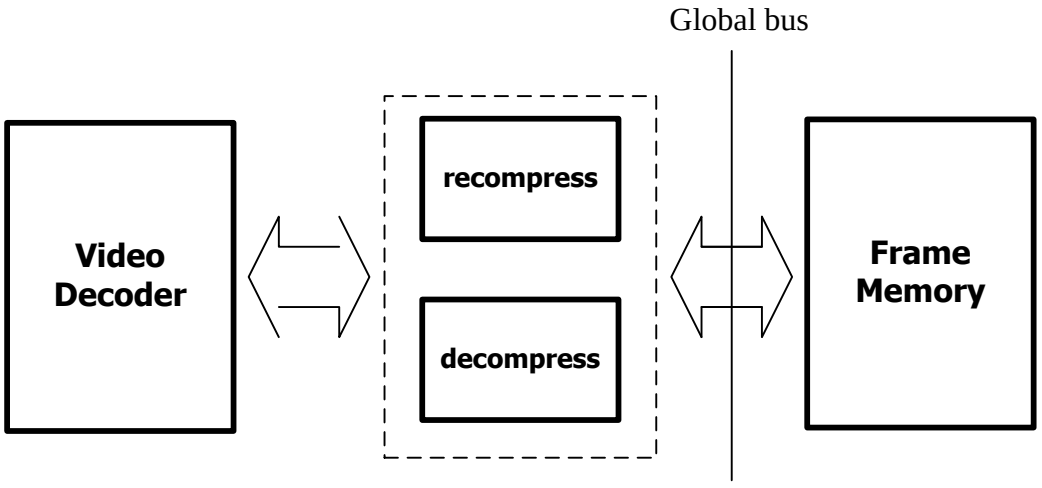

Fig 4.2 Embedded compress/decompress method

As for second solution, transmission reduction of redundant pixels, which can be classified into two solutions that first one is data fetch time reducing and the other one is data (pixel) reusing. The following subsection will discuss the detail of reduction strategies of memory bandwidth. Subsection 4.2.1 illustrates first strategy of data fetch times reducing. Subsection 4.2.2 gives second strategy of data fetch times reducing. Subsection 4.2.3 illustrates first strategy of data reusing. Finally, subsection 4.2.4 presents second strategy of data reusing.

4.1.1 Exact Fetch Necessary Pixels

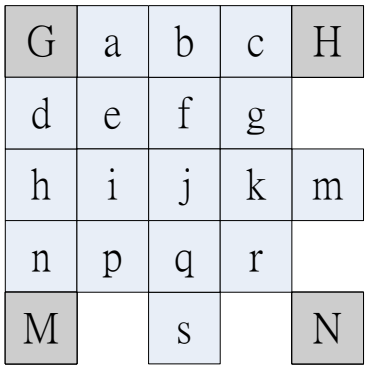

Fig 4.3 Fractional sample positions for quarter sample luma interpolation

Fig 4.3 illustrates the luma samples ‘a’ to ‘s’ at fractional sample positions. In traditional method, when interpolate fractional pixel, it always fetch 9x9 interpolation windows. However, there are not all pixels required in all fractional sample position. For example, the sample at half sample position labeled b is derived by the nearest integer position samples in the horizontal direction. Similarly, the sample at half sample position labeled h is derived by the nearest integer position samples in the vertical direction. Fig 4.4 illustrates interpolation of the samples at a, b, and c positions only need 9 x 4 interpolation windows. Fig 4.5 illustrates

interpolation of the samples at d, h, and n positions only need 4 x 9 interpolation windows. We can depend on motion vector value to exact fetch necessary pixels instead of fetch 9 x 9 interpolation window. Similar to luma interpolation, chroma interpolation can determine motion vector to decide interpolation window as well. Table 4.1 shows the summary of luma interpolation windows. Table 4.2 shows the summary of chroma interpolation windows. The strategy is also used in other design [14], [10], and [11]. As for bandwidth reduction result, we will show it later.

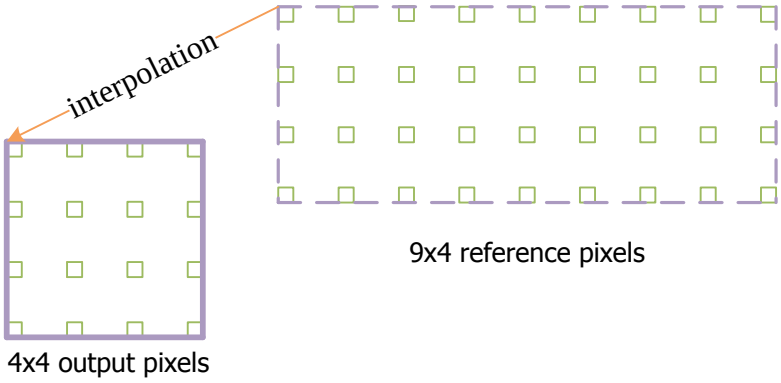

Fig 4.4 Fractional sample only need horizontal samples

.

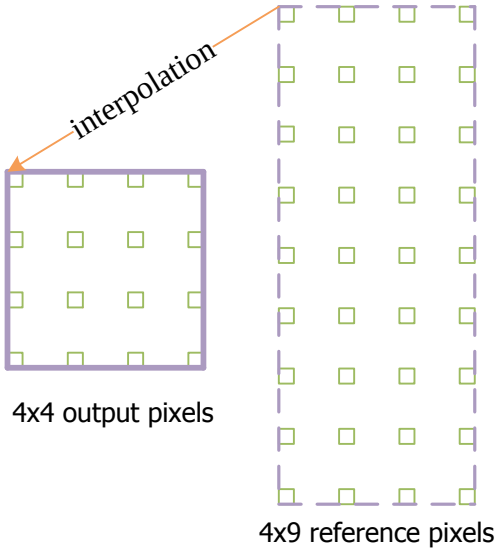

Fig 4.5 Fractional sample only need vertical samples

Table 4.1 Summary of luma interpolation windows

Pixel position	Interpolation Window Size
G (Integer)	4x4
a, b, c (Horizontal)	4x9
d, h, n (Vertical)	9x4
e, g, p, r	9x4+4x5
others	9x9

Table 4.2 Summary of chroma interpolation windows

Pixel position	Interpolation Window Size
Integer	2x2
Horizontal	3x2
Vertical	2x3
Others	3x3

4.1.2 Pre-fetch Mechanism

The second strategy of reduced fetching times is Pre-fetch Mechanism. Frame memories are such the largest memory storage over the entire video decoder that it are located on off-chip. Because bus interface has fixed width, every fetching may fetch unneeded pixels when fetch interpolation windows. If we save these unneeded pixels, it may be used in the future. Hence, we can further reduce fetching times. Fig 4.6 illustrates the interpolation window mismatch with bus interface and pre-fetch mechanism. The strategy is also used in other design [11]

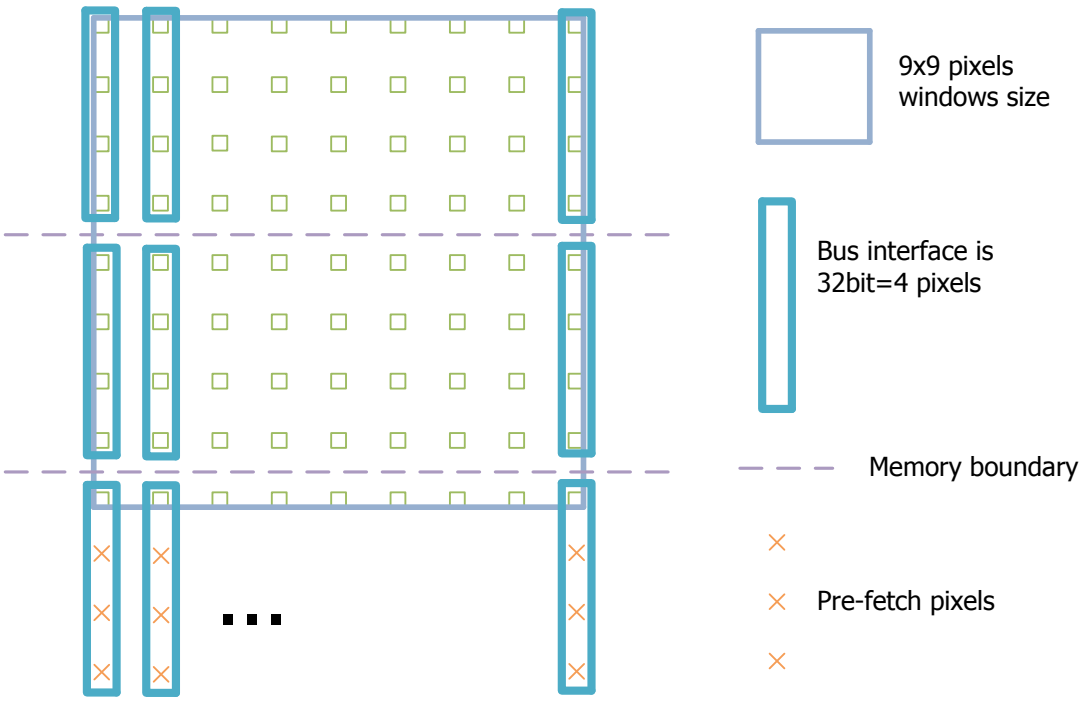

Fig 4.6 Pre-fetch mechanism

4.1.3 Intra MB Pixel Reusing

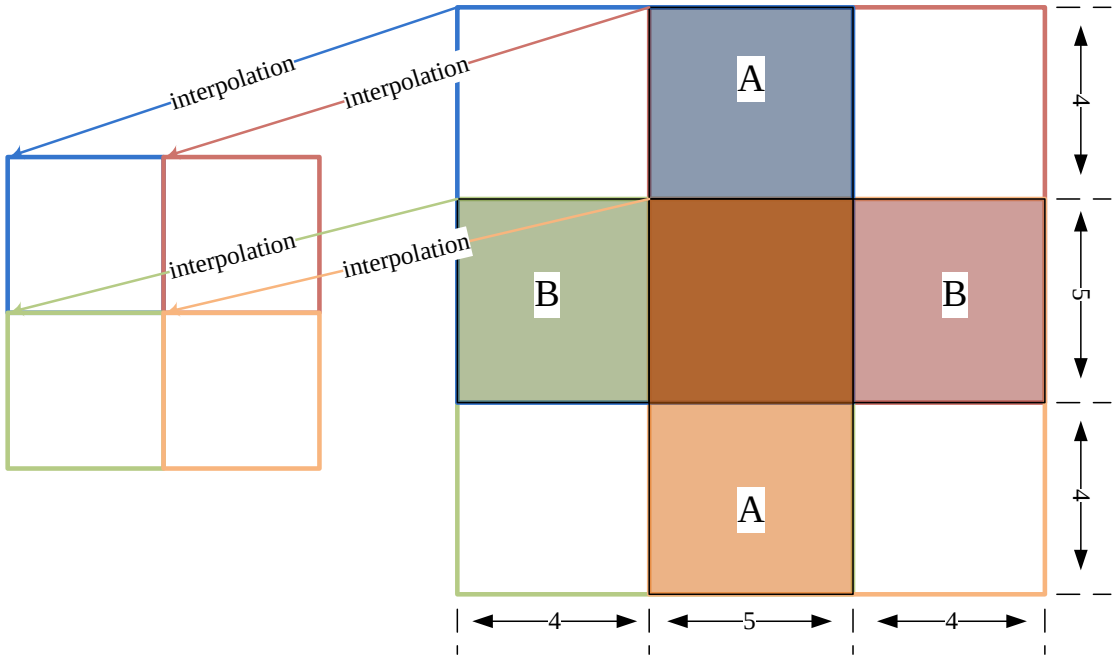

Fig 4.7 4x4 block window and the corresponding 9x9 interpolation window and overlapped region for neighboring interpolation window

Similar to reduced fetching times, pixel reusing can separate into intra MB overlap pixels reusing and inter MB overlap pixels reusing. The concept of overlap pixels reusing is if two motion vectors of horizontal neighboring 4 x 4 blocks are the same, 5 x 9 overlap region between two interpolation windows can be reused. Similarly, if two motion vectors of vertical neighboring 4 x 4 blocks are the same, 9 x 5 overlap region between two interpolation windows can be reused. Fig 4.7 illustrates four motion vectors of neighboring 4 x 4 blocks are the same and the corresponding 9 x 9 interpolation windows. We can see there are two vertical 5 x 9 overlap region indicated by “A” and two horizontal 9 x 5 overlap region indicated by “B” can be reused.

The first strategy of overlap pixels reusing is Intra MB Overlap Pixels reusing. Fig 4.8

illustrates the Intra MB overlap pixels reusing. There are some methods have been proposed in [14-16]. In Tsai's [14]，Tsai proposed VIDZ to achieve horizontal and vertical data reusing. Besides, based on the VIDZ flow, all vertically overlapped interpolation windows can be reused without additional storages. However, the violation of the inherent double-z-scan order, VIDZ cannot fit into a 4 x 4-block level pipeline. Moreover, in system view, VIDZ induces extra synchronization buffers because of different scanning order with other modules (for example, residual decoder) which must follow scanning order in standard [1].

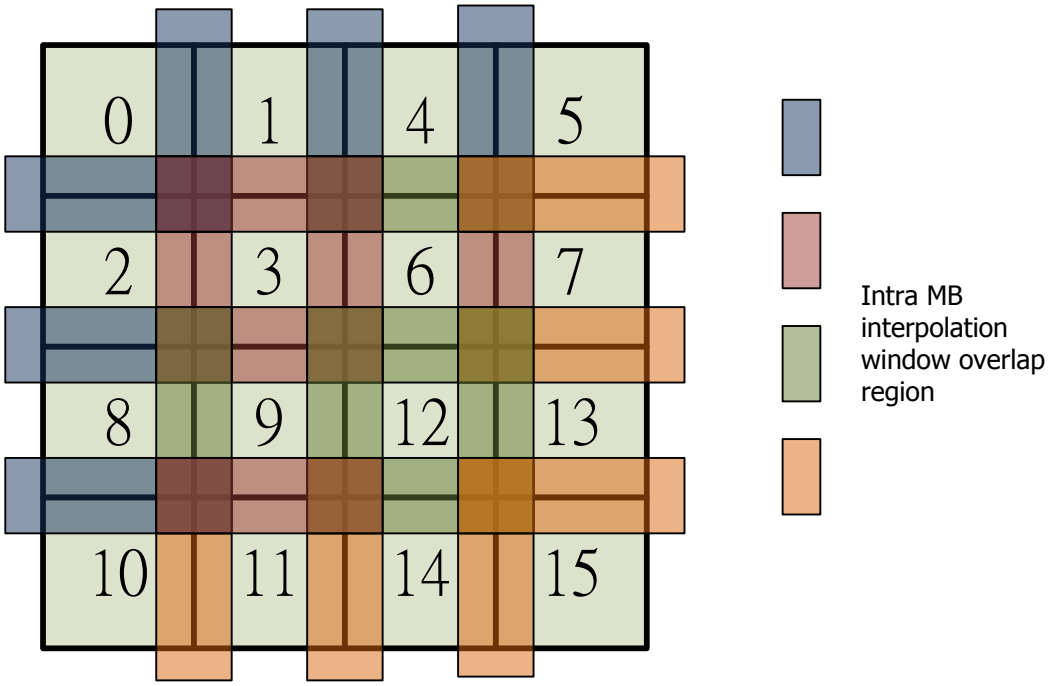

Fig 4.8 Intra MB overlap pixels reusing

4.1.4 Inter MB Pixel Reusing

The second strategy of overlap pixel reusing is Inter MB Overlap Pixels reusing. Up to now, literatures of neighboring-based pixels reusing almost focus on reusing pixels which inside the same MB. However, there are overlap region between interpolated windows which located on neighboring MB can be reused. Fig 4.9 illustrates overlapped region for

neighboring interpolation windows on horizontal neighboring MB. Only stores horizontal MB overlap regions is our selection. This is because if we want to reuse vertical MB overlap regions, there are MB regions of entire frame width needed to be store and only provide limited space of improve efficiency. Subsection 4.3 will show the analysis.

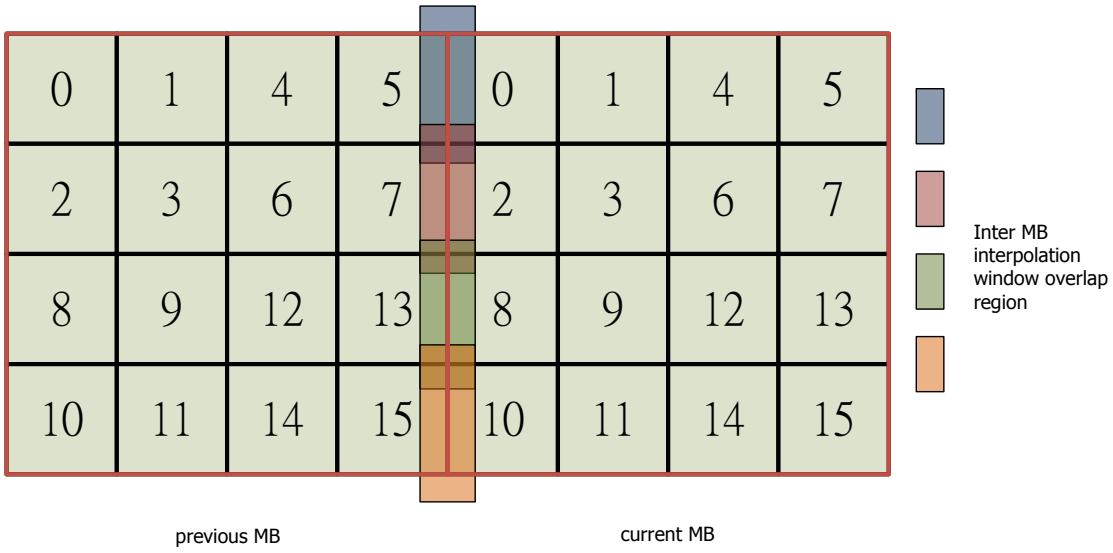

Fig 4.9 Inter MB overlap pixels reusing

The required content buffers are 5 x 9 pixels and 9 x 5 pixels for horizontal and vertical overlapped region for neighboring interpolation windows respectively. In order to minimize the content buffer size, the lifetime analysis of reference data shows that only three horizontal and three vertical blocks is required to be saved in the worst case.Table **4.3** shows the lifetime analysis. Horizontal axis shows 4 x 4 partition ordering, vertical axis shows the used storages, and filed is which partition horizontal or vertical overlap region of partition is stored. For example, in partition 1, horizontal overlap region of partition 1 will be stored in H0 and vertical overlap region of partition 1 will be stored in V0. Content buffers can be implemented in local registers or SRAM. However, SRAM needs several cycles to finish content-swap operation, so we use local registers in order to minimize latency on carrying out content-swap.

Table 4.3 Storage requirement and lifetime analysis

4 x 4 Storage	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	…
H0		1	1	3	3	5										15					…
H1								7										1	1	3	…
H2										9	9	11	11	13							…
V0	0	1	2					7					12	13			0	1	2		…
V1				3					8	9										3	…
V2					4	5	6					11									…
Time→																					

4.2 Limit of Reduced Memory Bandwidth

Our memory bandwidth reduction can be classified into data fetch time reducing and pixel reusing. In ideal condition, there exists reduction limit of memory bandwidth in terms of reduced fetch times. In ideal condition, all pixels locate on integer position. Table 4.4 shows all pixel position and their reduction percent. The reduction percent is original 9 x 9 interpolation window compare with exact interpolation window that only fetch required pixel. In Table 4.4, even though all pixels locate on integer position, G, we can see the limit of memory bandwidth reduction is 80%. However, all pixels located in integer position is impossible in real sequence

Table 4.4 Summary of luma interpolation windows and reduction percent

Pixel position	Interpolation Window Size	Reduction percent
G	4x4	80.25%
a, b, c	4x9	55.56%
d, h, n	9x4	55.56%
e, g, p, r	9x4+4x5	30.86%
f, i, j, k, q	9x9	0%

In another aspect, in terms of pixel reusing, Fig 4.10 illustrates all partitions can use all overlap region include previous upper MB overlap region and previous left MB overlap

region. Table 4.5 shows summary of reduction percent in different overlap region. We can see even though all partition have horizontal and vertical overlap regions can reuse in ideal condition, the limit of bandwidth reduction is 80%. However, if we want to reuse previous upper MB overlap region, each MB needs to be saved and only after process all following MB of frame width then can be reused and discarded because of characteristic of raster scanning. In other words, if we want to achieve upper MB overlap pixels reusing which is required to store MB overlap region of the entire row of frame. The storage depend on frame width and often very large. For example, it needs 21.6KB in 1080 HD. The storage is too large and only enhances 6% of memory bandwidth reduction. Hence, our selection is Intra MB with left MB. In ideal condition, we can achieve up to 74% bandwidth saving which is close to idea limit without huge overhead.

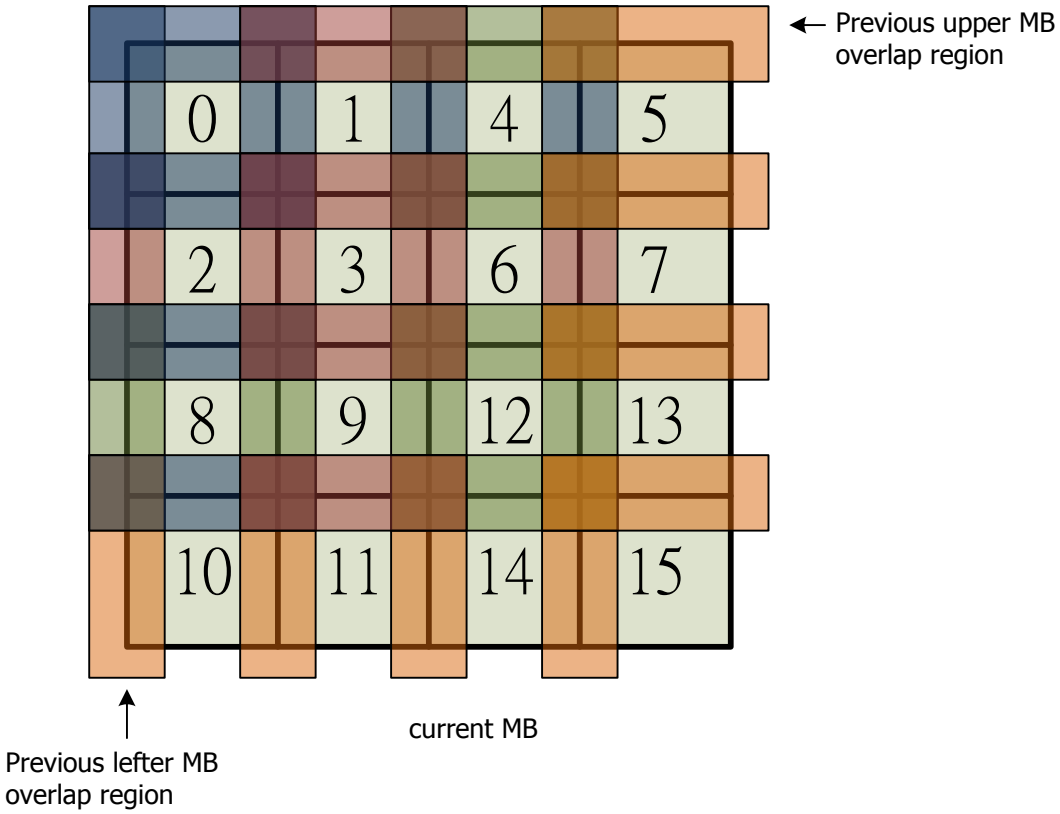

Fig 4.10 All overlap region include between previous upper MB and left MB

Table 4.5 Summary of reduction percent in different overlap region

Overlap region	Reduction percent
(all) Intra MB	65.97%
Intra MB + left MB	74.07%
Intra MB + left MB + upper MB	80.25%

In terms of data fetch times reducing and data reusing, it will not both happen all in ideal condition at the same time. This is because of integer pixel need not other pixels to interpolate result. In other words, it only bypasses reference pixels, so there are no overlap pixels to be reused. Fig 4.11 illustrates two motion vectors of neighboring 4 x 4 blocks are the same, there is no overlap region between two interpolating windows for data reusing.

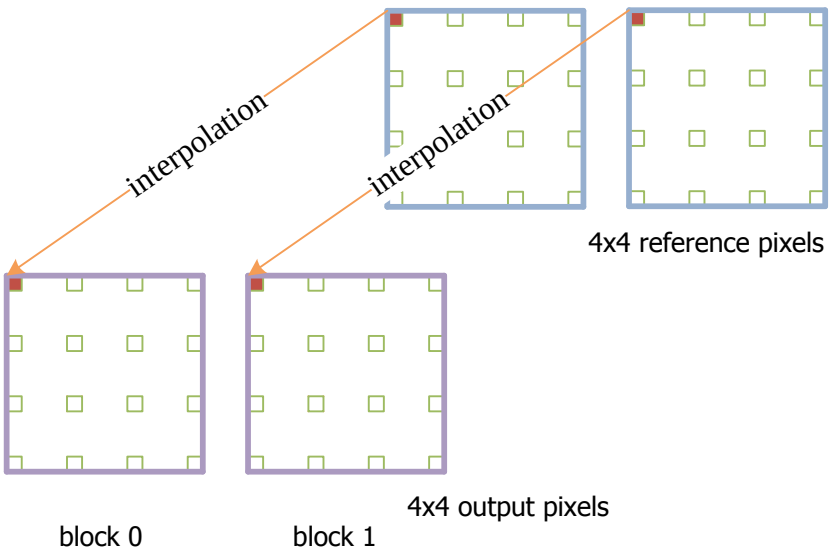

Fig 4.11 No overlap region can be reused

4.3 Summary

In this chapter, memory bandwidth, there are two directions adopted to reduce requirement of memory bandwidth. In these two directions, there are four strategies to achieve efficiently reducing memory bandwidth. Finally, the analysis of .reduced memory limit is discussed. The simulation result will show in chapter 5 and present our strategies is effective because of the close to limit of reduced memory bandwidth.

Chapter 5
Experiment Result

5.1 System Specification

Table 5.1 Video decoder specification in our design

H.264/AVC decoder
I, P, B slice
Variable block size: 16x16, 16x8, 8x16, 8x8, 8x4, 4x8, 4x4
Single reference frame (each direction)
Search range: [-128, +127.75]
Fractional motion resolution: quarter for luma, 1/8 for chroma
Frame/Field coding
Scalable High Profile (future)
Decoding capability: H.264/AVC: 1080 HD, 30fps
SVC: 720 HD – 1080 HD, 30fps (future)
Working Frequency:
H.264/AVC: 100 MHz
SVC: 150 MHz (future)
External Memory and Bus

Table 5.1 lists the specification of our H.264 video decoder. Fig 5.1 shows the whole H.264/AVC video decoder. We can see there exists embedded compressor and embedded decompressor to further reduce memory bandwidth requirement. Fig 5.2 shows the simulation result that applies our reduction strategies of memory bandwidth [17]. Memory bandwidth can be saved 71~80% and is very close to the limit of our analysis result shown in subsection

4.3. Fig 5.3 shows the comparison with related work [14]. If we pay attention to the extreme conditions, we can see that the reduction of memory bandwidth is very close to the limit in Akiyo sequence. In addition, the difference of memory bandwidth reduction in Stefan sequence is the largest. After we further analyze Akiyo and Stefan sequence, Fig 5.4 shows the ratio of pixels position in Akiyo and Stefan sequence. The reviewing of fractional sample position for luma interpolation is showed in Fig 4.3. In Akiyo sequence, the ideal condition (integral pixels) occupy up to 90%, so the memory bandwidth reduction is very close to the limit of memory bandwidth. In Stefan sequence, the pixels position is closely uniform distribution. In other words, ideal condition is less. That is, when the ratio of fractional position increases, comparing with other works will shows we can significantly enhance bandwidth reduction (Up to 11%).

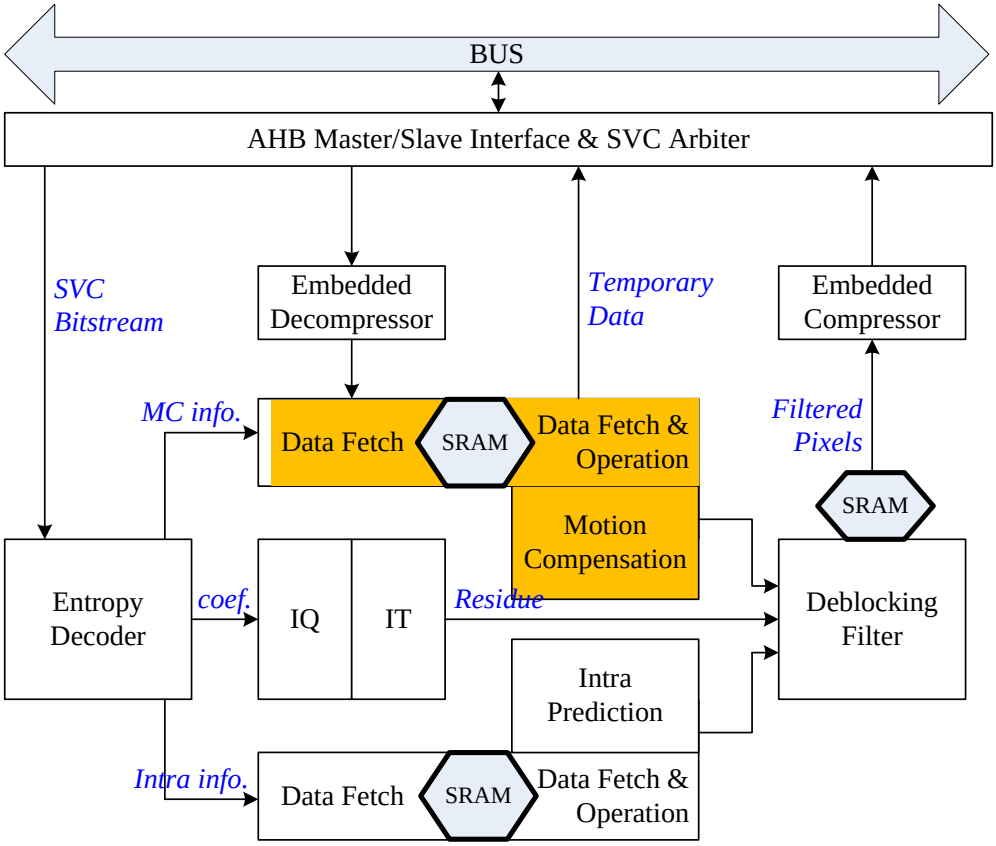

Fig 5.1 Motion compensation engine for H.264 video decoder

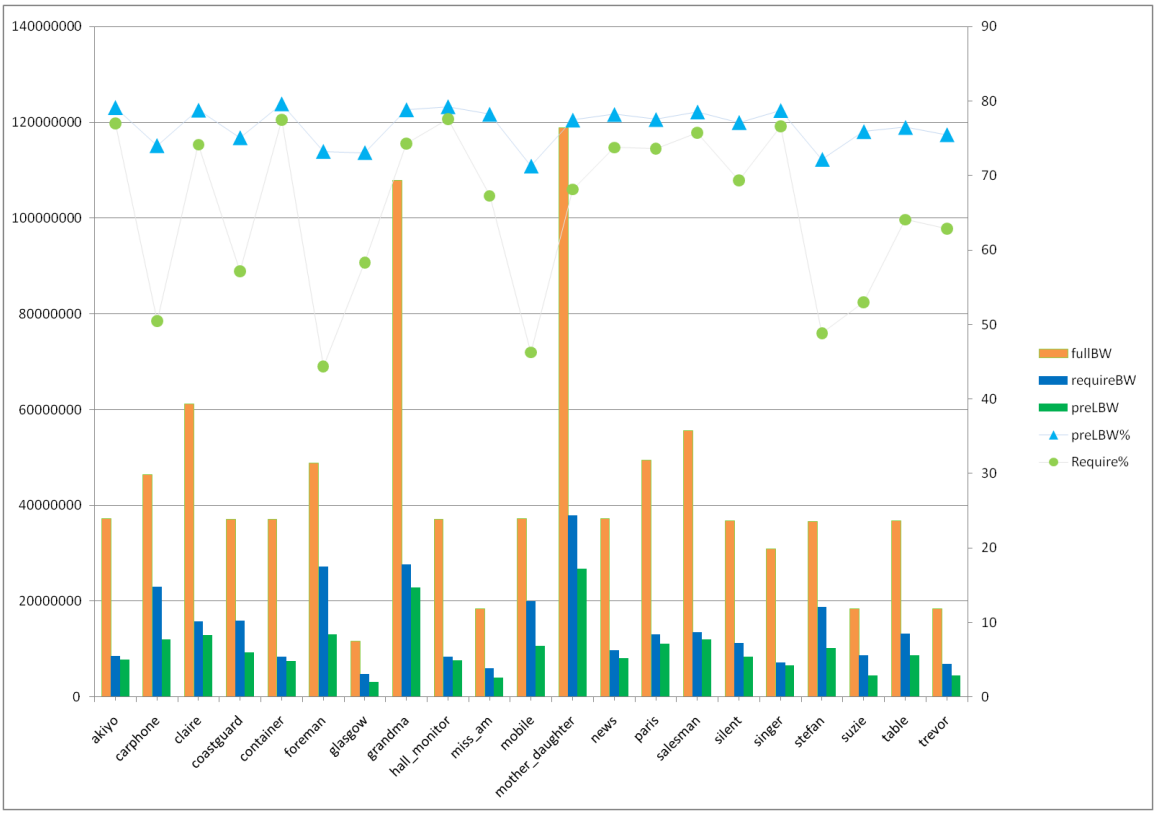

Fig 5.2 Simulation results of bandwidth reduction strategies

Fig 5.3 Compare related works

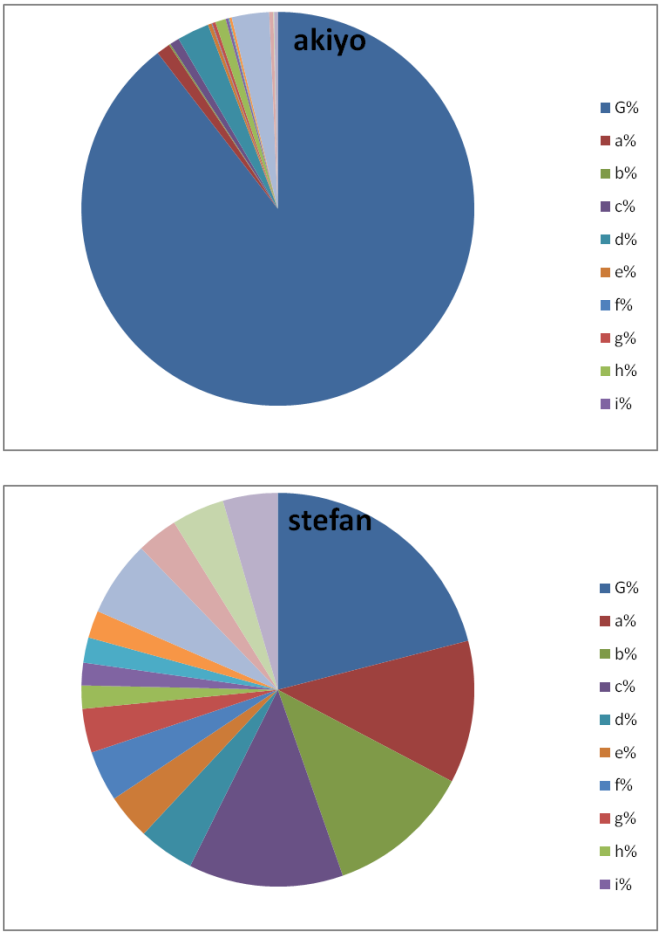

Fig 5.4 Ratio of pixels position in Akiyo and Stefan sequence

Even though the above discussion depend on different sequence characteristic, however, Fig 5.5 and Fig 5.6 [8] show the luma and chroma integer/fractional motion vector proportion for different foreman-QCIF bit-rate. In high bit rate coding (128 kbps), fractional motion vector occupies about 80 %. However, in low bit rate (32 kbps), fractional part only occupies 40 %. Higher bit-rate, higher fractional MV proportion, has better quality with more execution time to read pixels data from frame memory than integer motion vector. This gap may become more obvious especially when SDRAM is used as frame memory. In other words, our proposal is more suitable in high bit-rate than previous works for higher reduction of memory bandwidth.

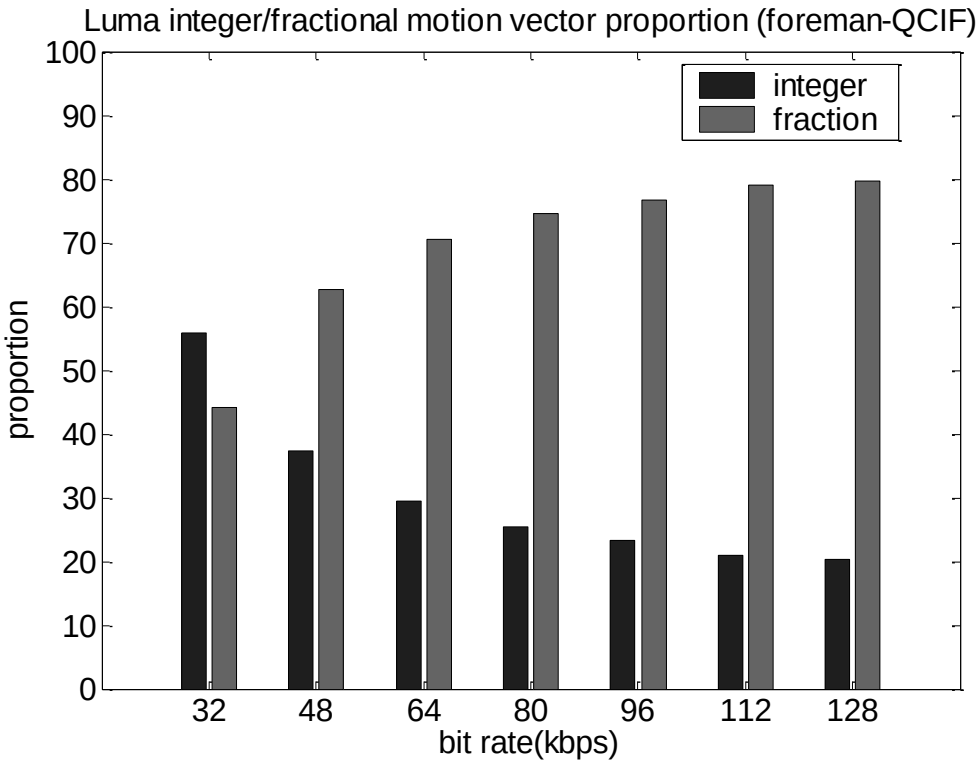

Fig 5.5 Luma integer/fractional motion vector proportion for H.264/AVC

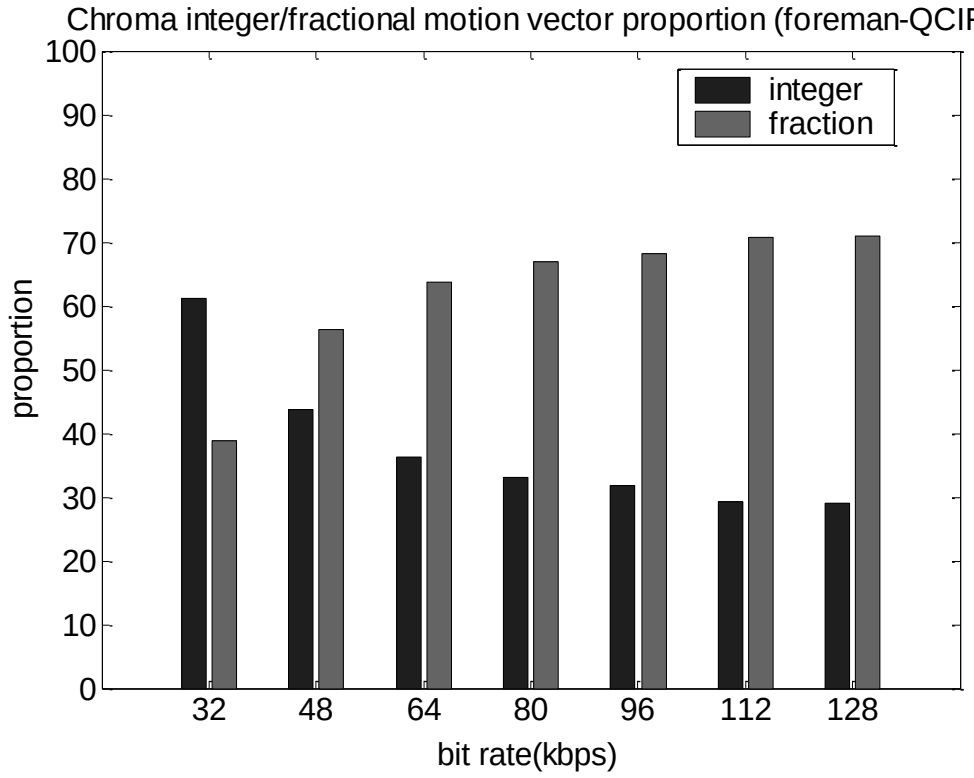

Fig 5.6 Chroma integer/fractional motion vector proportion for H.264/AVC

5.2 Comparison with Related Works

Table 5.2 lists the comparison with related works about motion compensation. We only focus on memory bandwidth reduction and interpolator design comparison. This is because memory bandwidth always is bottleneck of motion compensation and interpolator is key module in motion compensation. For another reason, each related works support different specification. We can see our memory bandwidth optimization is better than previous works although our storage is not least. However, our storage size is after trade-off and can get better performance. In terms of interpolator, [10] and [11] use hardware sharing to operate twice to achieve area efficiency. Even though these hardware sharing is suitable for Baseline Profile, but the poor throughput is not meet real-time decode in Main/High Profile. Moreover, our interpolator gate count is very close to these previous work [10] [11] and provide enough throughput performance in Main/High Profile.

Table 5.2 H.264decoder comparison with related work

		ISCAS '05[15]	ICASSP '06[16]	ISCAS '07[10]	MWSCAS '05[14]	ICME '09[11]	Proposed
Data paths	6-tap	13	12	4	12	4	**12**
	Bilinear	2	12	4	4	4	**0**
Technology (um)		0.18	0.18	0.18	0.18	0 .18	**UMC .09**
Gate count	Interpolator	20,686	15,000	13,027	21,506	11,823	**13,201**
	total	43k	61k	32k	47k	N/A	**68k**
Storage(Bytes)		54	81	2000	228	432	**396**
Bandwidth Optimization		30%	48%	60~80%	60~79%	70%	**71~80%**
Working Frequency (MHz)		100	150	100	125	100	**100 (max:176)**
Profile		BL	BL	BL	BL	BL	**HP**
Latency (Cycles/MB) luma+chroma		560	320	304	144+NA	288+NA	**144+48**

Chapter 6
Conclusion and Future Work

6.1 Conclusion

Motion compensation engine consists of three parts: motion vector generator, interpolator, and weighted predictor. Firstly, motion vector generator needs to support many tools in Main/High Profile. The challenge of motion vector generator is high complexity. We use hardware sharing to deal with double motion vectors, use coordinate mapping method to process direct modes, and merge MBAFF mode LUT and non-MBAFF mode LUT effectively to reduce the complexity. The design of interpolator, 4-parallel separate 1-D architecture gives the most space on high throughput compared with other proposed architectures. Hence, our interpolator is suitable for B slice and our restructured design can significantly reduce area cost. Lastly, weighted predictor located on last stage of motion compensation engine, we use LUT to deal with complicated implicit mode and collocate with interpolator in order to execute operation only occupies one cycle.

The design target of memory bandwidth reduction is to reduce external memory access and improve throughput of motion compensation engine. The proposed reduction strategies of memory bandwidth for motion compensation need 319 pixel storages is after trade-off and own better performance than other works. After applying these strategies, the memory bandwidth requirement can save the required bandwidth about 71~80 %. Moreover, achieve efficient memory access scheduling.

6.2 Future Work

The proposed motion compensator for H.264/AVC standard only supports up to Main/High Profile. If we want to support H.264/SVC/MVC, there are many issues should be taken into account. For example, hierarchical B pictures [18] [19]. In addition, a successor to H.264/AVC, High Efficiency Video Coding (HEVC) [20], is a proposed video compression standard, currently under development. If we want to support HEVC, the subjects such as extended macroblock size (EMS), decoder-side motion vector derivation (DMVD), 2-D non-separable adaptive interpolation filter (AIF), separable AIF, Direction AIF, Competition-based scheme for motion vector selection and coding, and so on tools should be taken into account for a next generation motion compensator.

In terms of memory bandwidth, our proposed mechanism can effectively reduce bandwidth requirement. However, there only focus on one single module in system view. Hence, there are still many important issues should be considered in order to provide bandwidth reduction in the viewpoint of overall system. For example, when embedded compressor/decompressor is disabled, a smarter SDRAM controller should be designed include scheduled memory accesses.

Bibliography

[1] Joint Draft ITU-T Rec. H.264 | ISO/IEC 14496-10 / Amd.3 Scalable video coding

[2] 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.

[3] T. C. Chen, Y. W. Huang, and L. G. Chen, "Fully utilized and reusable architecture for fractional motion estimation of H.264/AVC," *IEEE International Conference of Acoustics, Speech, and Signal Processing 2004*, vol. 5, pp.V-9-12, May 2004

[4] 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," *IEEE International Symposium of Circuits and Systems*, 2005, Vol. 5, pp.4542-4545, May 2005

[5] Digital Video Broadcasting - Wikipedia, the free encyclopedia Available from: <http://en.wikipedia.org/wiki/Digital_Video_Broadcasting>

[6] Iain E. G. Richardson, "H.264 and MPEG-4 VIDEO COMPRESSION", WILEY, 2003.

[7] Joint Video Team (JVT) reference software JM 8.2

[8] S. Z. Wang, "A Flexible Motion Compensation Memory Organization for Dual-standard Video Decoder", National Chiao-Tung University Taiwan, Master Thesis, June 2004.

[9] 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.

[10] Y. Li, Y. Qu, and Y. He, "Memory Cache Based Motion Compensation Architecture for HDTV H.264/AVC Decoder ", *IEEE International Symposium on Circuits and Systems*, pp. 2906-2909, May 2007

[11] D. Y. Shen, T. H. Tsai,"A 4X4-block level pipeline and bandwidth optimized motion compensation hardware design for H.264/AVC decoder", *IEEE International Conference on Multimedia and Expo 2009*, pp.1106-1109, Jul. 2009.

[12] A. Azevedo, B. Zatt, L. Agostini, S. Bampi, "Motion Compensation Decoder Architecture for H.264/AVC Main Profile Targeting HDTV", *International Conference on Very Large Scale Integration 2006,* pp.52-57, Oct. 2006.

[13] B. Zatt, A. Susin, S. Bampi, L. Agostini, "HP422-MoCHA: A H.264/AVC High Profile Motion Compensation Architecture for HDTV", *IEEE International Symposium on Circuits and Systems 2008,* pp.25-28, May 2008.

[14] C. Y. Tsai, T. C. Chen; T. W. Chen; L. G. Chen, "Bandwidth optimized motion compensation hardware design for H.264/AVC HDTV decoder", *Circuits and Systems, 2005. 48th Midwest Symposium*, Vol. 2, pp.1199-1202, Aug. 2005.

[15] S. Z. Wang, T. A. Lin, T. M. Liu, and C. Y. Lee, "A new motion compensation design for H.264/AVC decoder," *IEEE International Symposium on Circuits and Systems* 2005, Vol. 5, pp.4558–4561, May 2005

[16] J. W. Chen, C. C. Lin, J. I. Guo, J. S. Wang, "Low Complexity Architecture Design of H.264 Predictive Pixel Compensator for HDTV Application", *IEEE International Conference Acoustics, Speech and Signal Processing* 2006. (ICASSP), Vol. 3, pp. III - III, May 2006.

[17] Joint Video Team (JVT) reference software JM 17.0

[18] H. Schwarz, D. Marpe, and T. Wiegand, "ANALYSIS OF HIERARCHICAL B PICTURES AND MCTF", *IEEE International Conference on Multimedia and Expo 2006*, pp.1929-1932, Jul. 2006

[19] M. Winken, H. Schwarz, D. Marpe, and T. Wiegand, "JOINT OPTIMIZATION OF TRANSFORM COEFFICIENTS FOR HIERARCHICAL B PICTURE CODING IN H.264/AVC", *IEEE International Conference on Image Processing* 2007 (ICIP), Vol. 4, pp.IV-89-IV-92, Sep. 2007

[20] .High Efficiency Video Coding - Wikipedia, the free encyclopedia Available from: <http://en.wikipedia.org/wiki/HEVC>

[21] L. Yu, J. Li, Y. Zhang, “Fast Picture and Macroblock Level Adaptive Frame/Field Coding for H.264”, *IEEE International Conference of Acoustics, Speech, and Signal Processing 2006*, pp. 768-771, Dec. 2006

Vita

姓名：陳浩民

出生地：台灣省彰化市

出生日期：1977.11.18

學歷：　彰化縣立南興國民小學

彰化縣立彰安國民中學

國立虎尾科技大學　電機工程科

明新科技大學　電機工程系

國立交通大學　電機學院 (電子與光電學程) 碩士班

工作經歷：　太和科技服份有限公司　研發一處　工程師

研發處專案一部　高級工程師

研發處硬碟陣列部　科長

研發處硬碟陣列部　經理