國 立 交 通 大 學

電子工程學系　電子研究所碩士班

碩 士 論 文

H.264/AVC 可調式視訊解碼器之分析與動作補償設計

Analysis of H.264/AVC Scalable Extension Decoder and Its Motion Compensation Design

研究生：許博淵

指導教授：張添烜 博士

中華民國　九十八 年　十一 月


E S A
1896

H.264/AVC 可調式視訊解碼器之分析與動作補償設計

Analysis of H.264/AVC Scalable Extension Decoder and Its Motion Compensation Design

研 究 生：許博淵　　Student: Po-Yuan Hsu

指導教授：張添烜 博士　　Advisor: Tian-Sheuan Chang

A Thesis
Submitted to Department of Electronics Engineering & Institute of Electronics
College of Electrical and Computer Engineering
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of Master
in
Electronics Engineering
November 2009
Hsinchu, Taiwan

中華民國 九十八 年 十一 月


E S A
1896

H.264/AVC 可調式視訊解碼器之分析與動作補償設計

研究生: 許博淵　　指導教授: 張添烜 博士

國立交通大學

電子工程學系電子研究所

摘　要

可調式視訊編碼，一個新生代的視訊標準，除了繼承 H.264 的高壓縮率外，還提供了空間、時間、品質三種可調性，使得可調式視訊編碼較原本的 H.264 複雜許多。本篇論文的主要目標在於針對 H.264/AVC 可調式視訊解碼器做分析以及實現一個適用於可調式視訊解碼器的動作補償硬體設計。

首先，我們針對可調式視訊解碼器整個系統的記憶體使用作分析，藉由挑選以畫面幀為基礎的解碼流程，系統可以得到最低的內部記憶體使用量和頻寬需求。接著是我們針對可調式視訊解碼器提出的四級管線架構設計，並且提出單次品質層解碼的方法，在同個空間層平行處理基底品質層和品質增強層，使得巨圖塊的處理週期在三個品質層的位元流之中可以有 66%的縮減。最後則是提出一個高性能的動作補償設計，藉由同個分割區塊資料共用和縮減參考資料等方法，資料頻寬有 62-74%的縮減；更進一步，藉由使用兩套內插單元硬體，處理雙向預測區塊的週期數將得以減半。

根據實驗結果，我們提出的動作補償硬體設計的巨圖塊平均處理週期低於 160 個週期，也低於我們系統限制的 227 個週期。換句話說，我們所提出的硬體設計可以在 135MHz 的時脈下，達到每秒處理超過 59 萬 4 千個巨圖塊，也就是每秒 60 張 CIF、SD 480p 以及 HD 1080p 的影像。


E S A
1896

Analysis of H.264/AVC Scalable Extension Decoder and Its Motion Compensation Design

Student: Po-Yuan Hsu Advisor: Tian-Sheuan Chang

Department of Electronics Engineering & Institute of Electronics

National Chiao Tung University

Abstract

Scalable Video Coding (SVC), a new generation video codec, not only inherits the high coding efficiency of H.264/AVC standard, but also supports scalabilities of spatial, temporal, and quality domain, inducing SVC much more complicated than H.264/AVC standard. The main goals of this thesis are to perform an analysis of SVC decoder and implement a motion compensation hardware design for it.

At first, a memory analysis of SVC decoder system is proposed. By choosing frame-based decoding flow, the system would enjoy the minimum internal memory usage and bandwidth requirement. Then, we propose a four-stage MB-pipeline architecture for our SVC decoder, and there is also a one-pass quality layer decoding method proposed, which parallel processes base quality layer and quality enhancement layers in same spatial domain for 66% reduction of MB processing cycles in bitstream of three quality layers. Finally, a high performance motion compensation design is presented. By *Block Size Based Data Request* and *Precision Based Data Request*, the data bandwidth is reduced by 62-74%. Moreover, by *Doubled Hardware of Interpolation Unit* scheme, the processing cycles for bi-pred

block will be halved.

According to experiment results, the average processing cycles in our proposed motion compensation design are below 160 cycles/MB which is under our system constraint 227 cycles/MB. On the other hand, the proposed hardware design can process more than 594k macroblocks per second operating at 135MHz clock rate, which is equivalent to 60 frames of CIF, SD 480p, and HD 1080p resolutions.

誌　謝

首先，要感謝我的指導教授—張添烜博士，在研究所的生涯給我的支持和鼓勵，引領我以正確的態度賴面對與解決問題。在研究上讓我自由的發揮，並在我遇到瓶頸時給予建議與協助。感謝老師提供豐富的實驗室資源，讓我有個舒適的環境來充分利用各種軟硬體設備進行研究工作。老師不僅是研究上的良師，也是生活上的益友。

感謝我的口試委員們，清華大學電機工程系陳永昌教授與交通大學電子工程系李鎮宜教授。感謝你們百忙之中抽空前來指導，因為教授們的寶貴意見，讓我的論文更加的充實而完備。

接著我要感謝實驗室的好夥伴們，謝謝引我入門的李國龍學長，從我剛加入實驗室就帶領我進入視訊壓縮的這門學問中，您的指導是我研究所生涯最大的收穫。感謝曾宇晟學長，不論是在研究的過程或是工作站軟體的使用中，總是能幫助我解決問題。謝謝林佑昆學長和張彥中學長，你們對於經驗與知識的分享，都讓我受用無窮。感謝蔡宗憲學長、詹景竹學長、戴瑋呈學長與張瑋城學長，你們認真負責的態度是我學習的榜樣。再來要感謝我的戰友們，陳之悠(大哥)、黃筱珊(一姊)、沈孟維(雄哥)、蔡政君(A 君)，不論是出遊或是打球，都讓我的研究所生涯充滿新奇和有趣。此外還有王國振學長，以及廖元歆、陳宥辰、許博雄、陳奕均、洪瑩蓉等學弟妹，有你們的陪伴，讓我的碩士生涯充滿歡笑。謝謝實驗室的所有成員們，和你們一起努力的日子，是我在交大的研究所生涯最寶貴的回憶。

最後要感謝永遠支持我的家人們，我的爸爸、媽媽、哥哥，還有女友，你們的支持與鼓勵，是我順利完成學業的最大動力。

在此，謹將本論文獻給所有愛我以及我愛的人。


E S A
1896

Contents

List of Figures

List of Tables


E S A
1896

Chapter 1. Introduction

Recently, the advances of network bandwidth and wireless access techniques boost the development of multimedia services. The state-of-the-art video codec H.264/AVC promises the dominant status over multimedia content service. It provides high compression and high quality video but with only fixed resolution. Due to the heterogeneities on user devices and network environments, multimedia stream with scalable features is demanded. A single bitstream to satisfy various clients becomes more and more desired. Therefore, the new Scalable Video Coding (SVC) standard was developed based on H.264/AVC [1] by the Joint Video Team (JVT) to provide this service. The high coding performance makes H.264/AVC suitable for high resolution video compression. However, the huge computation remains a problem for hardware implementation.

1.1. Motivation

In today's technology, the High-Definition Television (HDTV) seems to be a basic equipment in multimedia entertainments. Besides HDTV, there are also cell-phone, PDA, and notebook that may use multimedia applications. With one single SVC bitstream, these receiving devices can get their own video quality by extracting the bitstream according to their requirements.

SVC has been standardized in 2007. However, there is no hardware implementation of SVC decoder which has been published yet. In this work, we want to development a motion compensation design of an SVC decoder that support

today's full-HD specification which is HD 1080p with 60Hz frame rate.

1.2. Thesis Organization

The organization of this thesis is described as follows. Chapter 1 makes a brief introduction of SVC and motivation of this work. Chapter 2 gives an overview on SVC standard and introduces the components of SVC decoder. In Chapter 3, an analysis on SVC decoder is presented, with the decoding flow selection and pipeline stage design. Chapter 4 shows the hardware design and experiment results of proposed motion compensation design. Finally, the conclusion and future work will be given in Chapter 5.

Chapter 2. Overview of SVC Standard

Digital video compression techniques have played an important role in the world of multimedia systems. Hence, video coding techniques are for reducing the amount of information needed for a video sequence without losing much of its quality. Fig. 2.1 shows the history of video coding standards. With the improvements of network techniques and multimedia applications, a variety of devices with different capabilities has become very popular. Traditional standards provide fix video content which cannot deal with the heterogeneous of receiving devices. Therefore, a new video coding standard, SVC, was standardized as H.264 Annex G in 2007 [2]. SVC not only inherits the high coding efficiency of H.264/AVC standard but also supports scalabilities of spatial, temporal, and quality domain [3]. The term "scalability" means that certain parts of the bitstream can be removed in order to adapt to the requirements of receivers. The video bitstream of SVC is structured in layers, consisting of a base layer and one or more enhancement layers. Each enhancement layer improves the resolution or the quality of the video sequence. Fig. 2.2 shows an example of SVC streaming with heterogeneous receiving devices. The SVC encoder presents a fully content of video bitstream that provides the highest resolution, highest frame-rate, and highest quality video. On decoder side, the receiving devices get their own video quality by extracting the fixed bitstream according to the network bandwidth and their requirements.

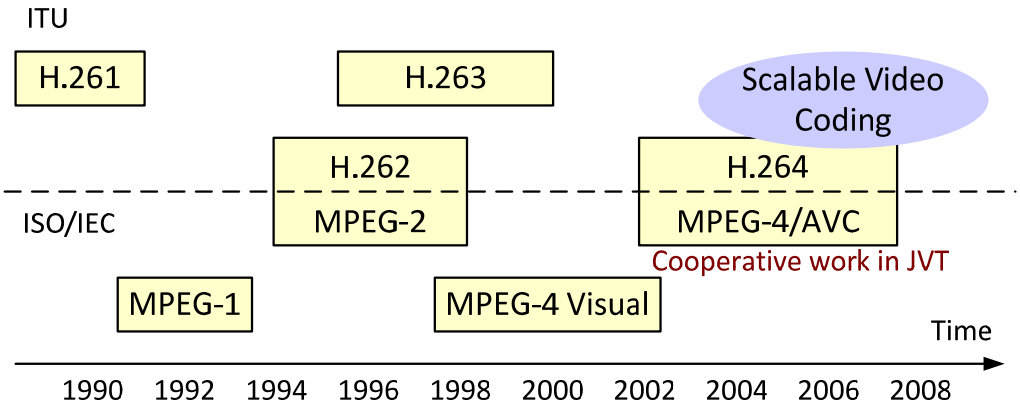


Fig. 2.1. History of video coding standards

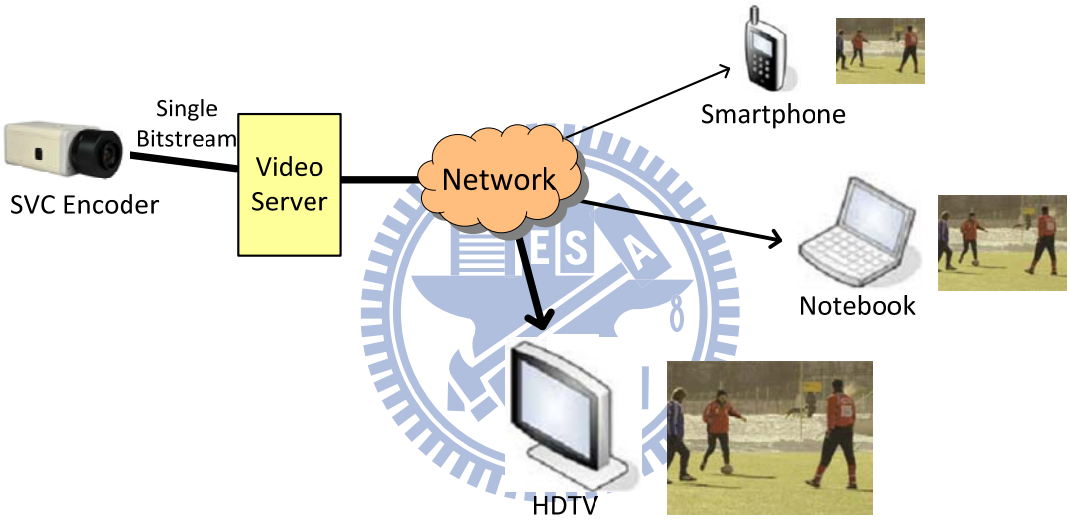


Fig. 2.2. Example of video streaming with heterogeneous receiving devices

2.1. Fundamentals of SVC

SVC is based on H.264/AVC and adds new features to achieve the scalabilities. Fig. 2.3 shows an architecture of SVC encoder with two spatial layers. The dotted line block in this figure is an H.264/AVC compatible encoder. The input video first down-sampled to lower resolution, and then goes through the base layer coding and produces base layer bitstream just like H.264/AVC. The block named “Progressive

SNR refinement texture coding" enables the quality scalability by generating more transform coefficients to improve the video quality. The base layer information, such as intra texture, motion vectors, and residuals, are upsampled to spatial enhancement layer for inter-layer prediction. Spatial enhancement layer reuses these base layer information as much as possible to improve coding efficiency as spatial scalability. The temporal scalability comes from the hierarchical-B picture structure [4], [5] as shown in Fig. 2.4. By receiving more temporal levels, the output video sequence will be more fluent.

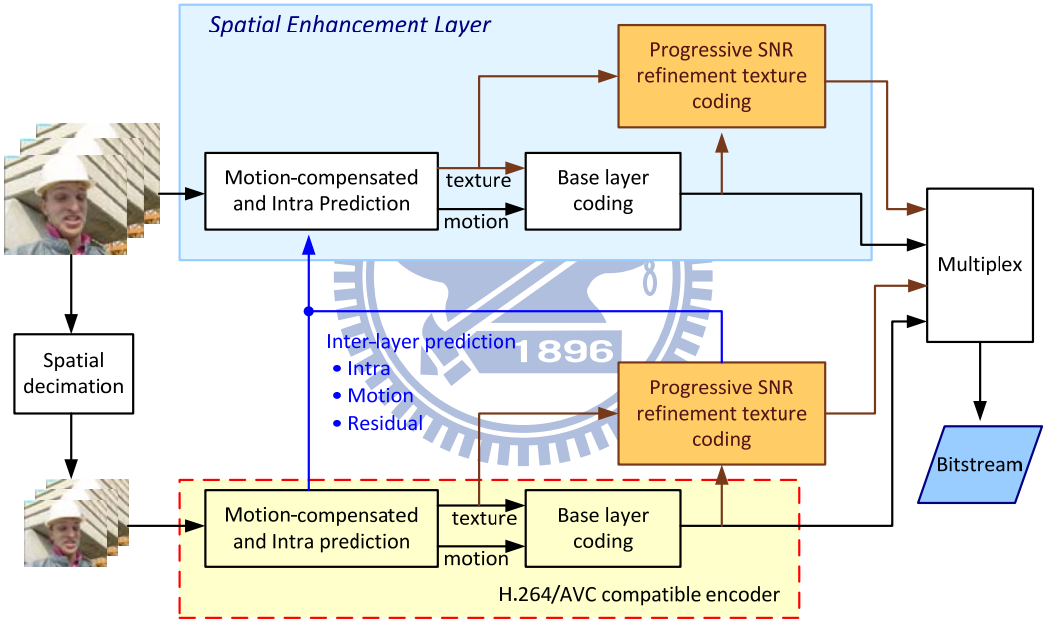


Fig. 2.3. Block diagram of an SVC encoder with two spatial layers

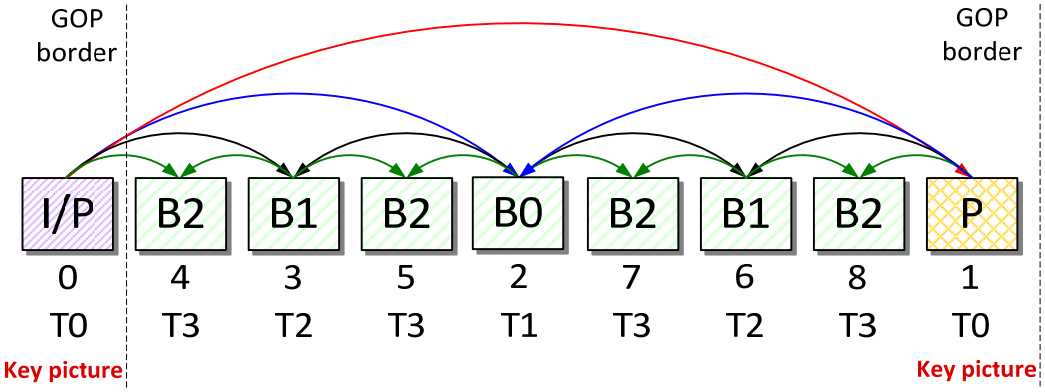


Fig. 2.4. Reference scheme of hierarchical-B picture structure

2.2. Components of SVC Decoder

SVC inherits most of the components from H.264/AVC and adds three upsampling functions for inter-layer prediction as shown in Fig. 2.5. First, the SVC decoder parses and decodes the bitstream by Entropy Decoding module, then the prediction type and transform coefficients will be known. Transform coefficients need inverse quantization and inverse transform in Residual Decoding module to generate residual data, and intra or inter prediction will be enables according to the prediction type. Intra Prediction module produces prediction samples based on the reconstructed pixels from neighboring pixels in the same frame, and contains three different prediction types for luminance part as illustrated in Fig. 2.6. For inter prediction, Motion Vector Reconstruction module first generates MV information, then Motion Compensation module produces prediction samples based on previous decoded frames. Each inter-coded macroblock can be divided into smaller partitions with variable block size from 4x4 to 16x16 as shown in Fig. 2.7. After prediction and residuals are ready, they are summed together in Sample Reconstruction module.

Finally, the Deblocking Filter module is applied to improve visual quality by smoothing the vertical or horizontal block edges between transform blocks as shown in Fig. 2.8.

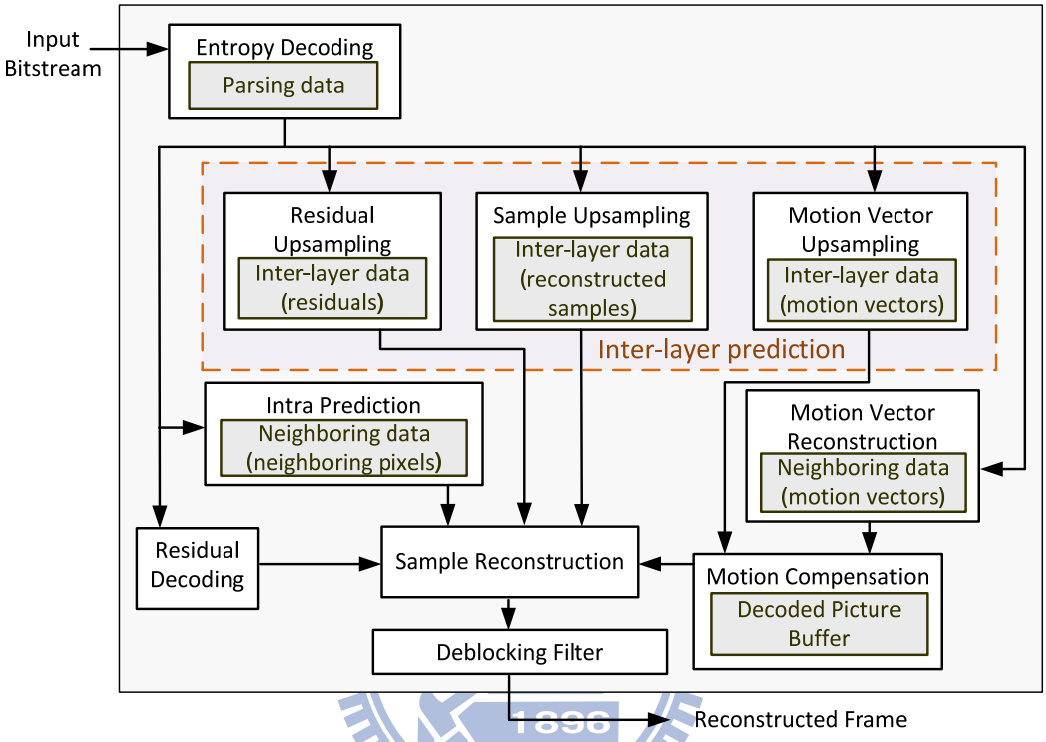
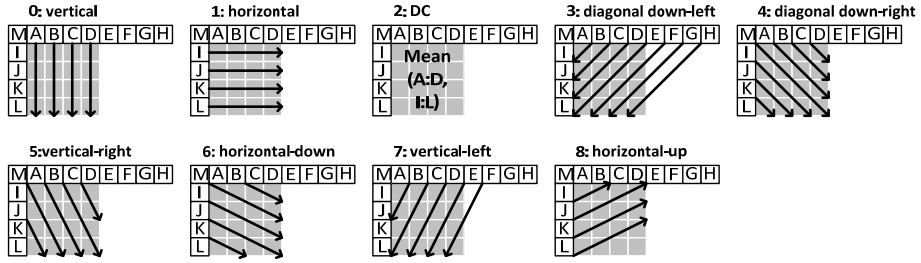
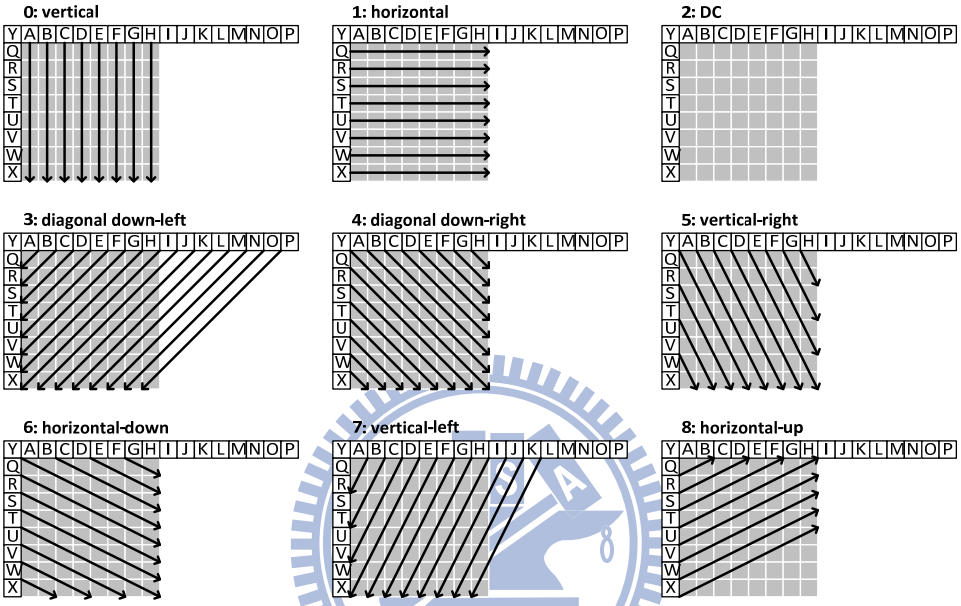


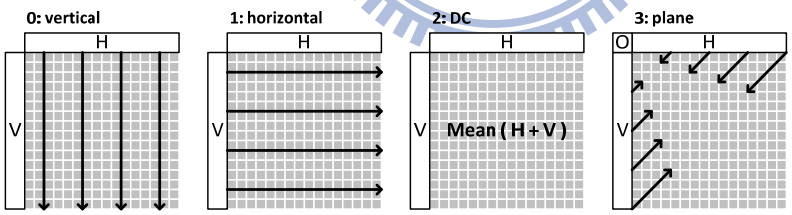
Fig. 2.5. Block diagram of SVC decoder



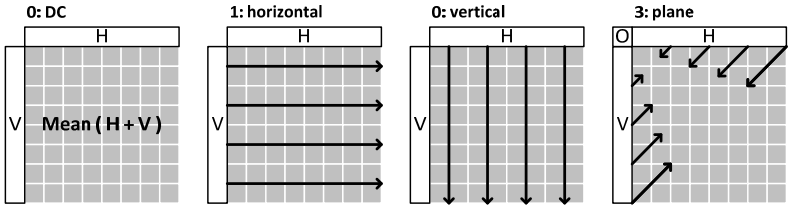
(a)



(b)



(c)



(d)

Fig. 2.6. Intra prediction: (a) 4x4 block; (b) 8x8 block; (c) 16x16 block; (d) chroma block

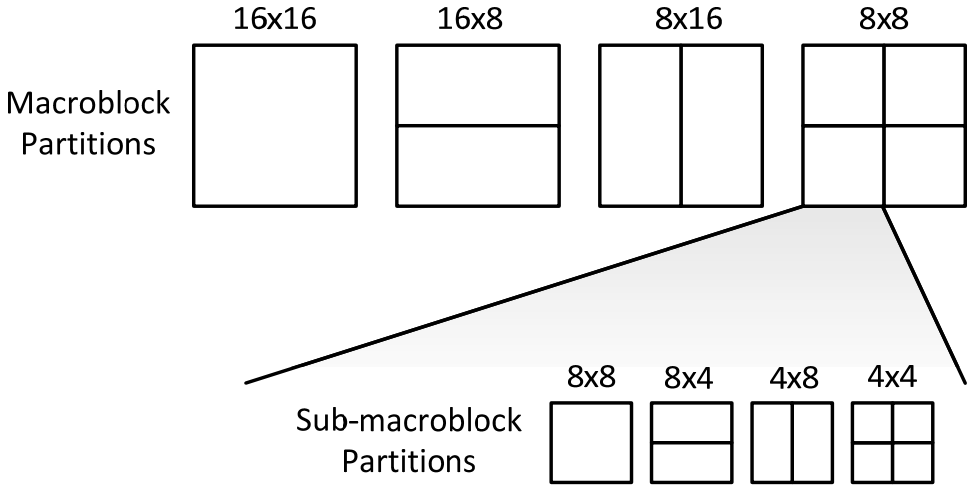


Fig. 2.7. Block partition modes of inter prediction

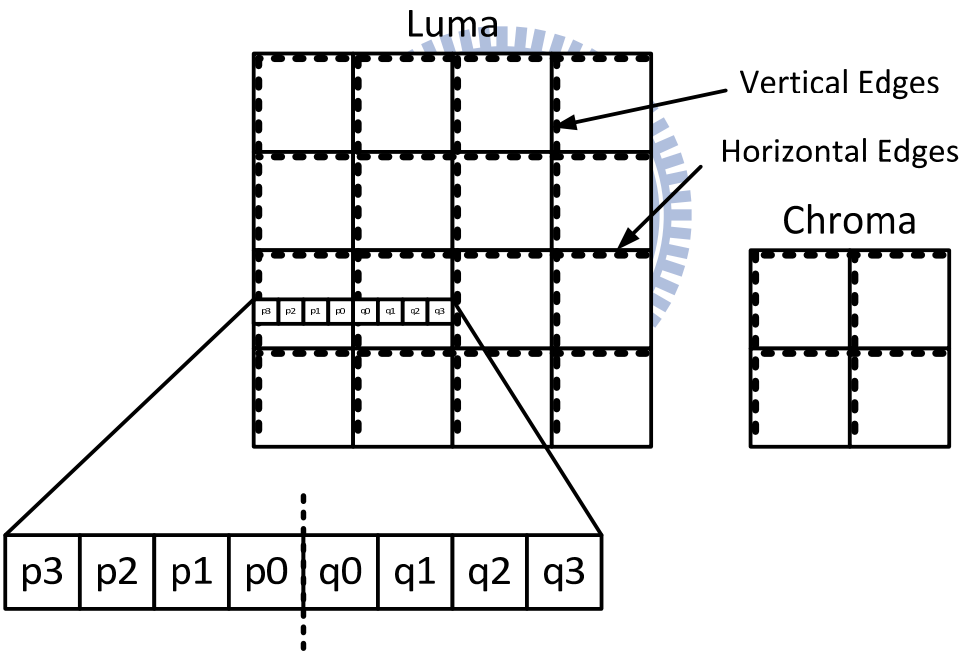


Fig. 2.8. Horizontal edges and vertical edges in one macroblock

The inter-layer prediction contains Sample Upsampling module (inter-layer intra texture prediction), Motion Vector Upsampling module (inter-layer motion prediction), and Residual Upsampling module (inter-layer residual prediction) for spatial scalability as shown in Fig. 2.9. By restricting the inter-layer intra texture prediction

to intra-coded block only [9], a single-loop decoder for spatial scalability can be realized requiring only the motion data of inter-coded block and reconstructed intra pixels in base layer. Quality scalability can be considered as a special case of spatial scalability with identical resolution for base quality layer and quality enhancement layer. In quality enhancement layer, it contains progressive refinement coefficients to be added to base quality layer.

For more details, please refer to [2], [3], and [6].

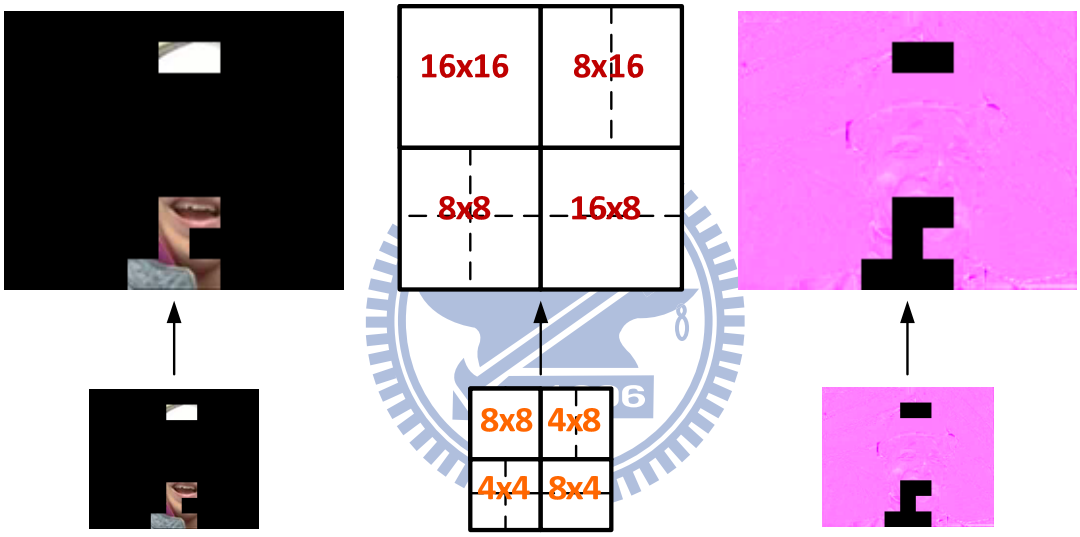


Fig. 2.9. Illustration of inter-layer prediction features in dyadic spatial scalability: (a) inter-layer intra texture prediction, (b) inter-layer motion prediction, (c) inter-layer residual prediction

Chapter 3. Analysis of SVC Decoder

Since SVC supports several new coding features, the complexity of SVC decoder is much higher than H.264/AVC. As a result, a completely analysis of SVC decoder before hardware implementation is required. In this chapter, we first analyze the memory requirements of different decoding flows, trying to find out a better way for SVC decoder, then present the pipeline architecture for our hardware design, and propose a one-pass quality layer decoding method to parallel process quality layers in a spatial domain at last.

3.1. Decoding Flow

For an H.264/AVC decoder, a straightforward decoding process is macroblock by macroblock. The decoding flow first parses the input bitstream by entropy decoding and recovers the residual and related prediction mode data needed for the decoding process. Then the residual is added with the prediction samples from inter or intra prediction to recover the pixel values.

In addition to the inherent decoding operations of H.264/AVC, the SVC decoder supports three scalabilities, spatial, temporal, and quality. For the purpose of memory analysis [8], we only consider spatial scalability in this paper since temporal scalability should be fully supported. For the spatial scalability, base layer data needs to be decoded before its corresponding macroblock in enhancement layer for the purpose of inter-layer data reuse. Enhancement layer reuses base layer information such as motion vectors, residuals, or reconstructed samples for the reconstruction

process. The block diagram of SVC decoder has shown in Fig. 2.5. There are three new blocks in the figure within dashed line compared to H.264. In which Sample Upsampling process is used in inter-layer intra texture prediction that upsamples the corresponding reconstructed samples, Motion Vector Upsampling upsamples macroblock partition and motion vectors from reference layer, and Residual Upsampling is corresponding to inter-layer residual prediction and it block-wised upsamples residuals. With these three inter-layer predictions, SVC decoder can decode its data to recover the pixel values.

3.1.1. Memory Analysis for H.264/AVC Decoder

The memory requirements of H.264/AVC decoder are mainly dominated by four parts: parsing data, macroblock processing data, neighboring data, and decoded picture buffer. In the following, a macroblock based decoding flow is assume. The parsing data stores the information parsed from the bitstream, such as the SPS, PPS, and slice header data. The transform coefficients parsed from bitstream are also included in it. The parsing data consumes about 4.4 KB memory derived statistical results. The macroblock processing data stores the information that may be used to reconstruct a macroblock such as prediction mode, residuals, and reconstructed samples. This part requires 4.8 KB memory spaces. Since the basic processing unit is macroblock, above memory usage is irrelevant to the frame size actually. The neighboring data stores previous decoded neighboring macroblock information, i.e. left, up, upper-right, or upper-left macroblock, which contains motion vectors, reference picture, prediction mode, and neighboring pixels. Pre-deblocking coefficients are also included. The neighboring data is stored in a row of macroblocks in frame width, for example, a row consists of 22 macroblocks in CIF size. For neighboring pixels, the size is one line of

samples of the frame width plus one column height of a macroblock. The decoded picture buffer (DPB) stores previous decoded frames as reference frame for inter prediction. The DPB is refreshed after decoding one GOP. However, the data of DPB are stored in external memory since such significant memory requirement is unreasonable to be stored in internal memory. From the analysis described above, the internal memory usage and external memory access of H.264 decoder are summarized in Table 3.1 and Table 3.2, respectively.

Table 3.1. Internal memory usage of H.264/AVC decoder

Name	Size (KB)
*Parsing data	~4.4
*Macroblock processing data	~4.8
Neighboring data	0.2 * PicWidthInMbs + 1.06

***PicWidthInMbs**: number of macroblocks in a row of a frame*

**: fixed data: will not change with the variation of frame resolution*

Table 3.2. External memory access of H.264/AVC decoder

Name	Size (KB)
Input bitstream	*0.0375*PicSizeInMbs
Reference samples	**1.35 *PicSizeInMbs
Reconstructed samples	0.375*PicSizeInMbs

***PicSizeInMbs**: number of macroblocks in a frame*

**: assume the compression rate is 10% of the original data*

***: assume every macroblock has 16 4x4 subblocks with one direction prediction*

3.1.2. Memory Analysis for SVC Decoder

Memory requirement of SVC inherits the entire requirement from H.264 with additional memory from inter-layer prediction as shown in Fig. 3.1. For the inter-layer prediction, it reuses the reference layer data such as motion vectors, residuals, and reconstructed samples. With this inter-layer dependency, different decoding flows will cause different memory requirements. In the following, we specify three decoding flows, macroblock-based, row-based, and frame-based, that can be applied to SVC decoding process. Furthermore, their corresponding internal memory usage and external memory access will be analyzed in the following article.

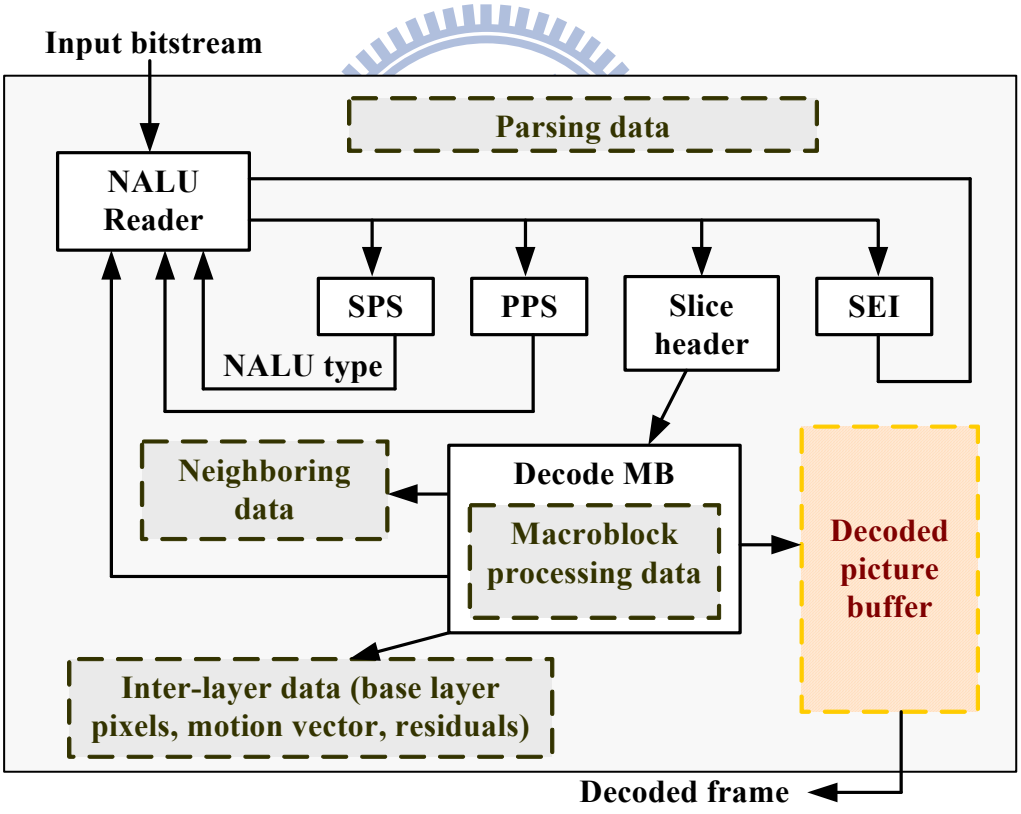


Fig. 3.1. SVC decoder memory map

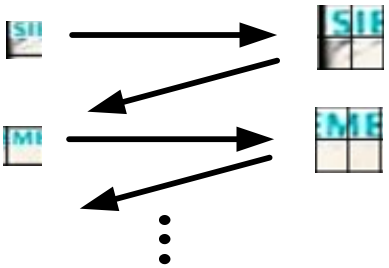
a) Macroblock (MB)-based

The first decoding flow is macroblock-based, which decodes one macroblock in base layer and then the corresponding four macroblocks in the enhancement layer as illustrated in Fig. 3.2. In this flow, inter-layer data can be reused immediately for enhancement layer. In this method, we have to store the neighboring data corresponding to its current decoding macroblock of each spatial layer separately since multiple layers are decoded at the same time. Therefore, the internal memory usage, especially the neighboring data, is increased because of more than one layer as shown in Table 3.3. Besides, we also have to store the base layer decoded information as inter-layer data for inter-layer prediction reference. Therefore, Table 3.4 shows the composed elements of inter-layer data and its corresponding additional memory requirement. The NumMB_{ref} is set as follows:

$$\mathrm{NumMB}_{ref} = \sum_{n=0}^{d-2} 2^{2n} \tag{3.1}$$

***d**: number of spatial layers*

In this method, these data can be immediately reused in the following enhancement layer decoding. Table 3.5 shows the corresponding external memory access.



(a)

```
while( spatial_layer_id < d ){
  while( CurrMbAddr < 4^spatial_layer_id ){
    decode_one_macroblock();
    CurrMbAddr ++;
  }
  spatial_layer_id ++;
}
```

(b)

***spatial_layer_id**: the index of current decoding spatial layer*

***d**: number of spatial layers*

***CurrMbAddr**: current decoding macroblock address*

Fig. 3.2. Macroblock-based decoding flow: (a) graphical illustration; (b) pseudo code

Table 3.3. Internal memory usage of multi-layer decoding

Name	Size (KB)
Parsing data	~4.5
Macroblock processing data	~5
Neighboring data	$\sum_{n=0}^{d-1} (0.2 * \text{PicWidthInMbs}_n + 1.06)$

***d**: the number of spatial layers*

***PicWidthInMbs$_n$**: number of macroblocks in a row of spatial layer n*

Table 3.4. Inter-layer data requirement of SVC decoder

Name	Size (KB)
Reference layer motion vectors	$0.125 * \text{NumMB}_{ref}$
Reference layer residuals	$0.75 * \text{NumMB}_{ref}$
Reference layer samples	$0.375 * \text{NumMB}_{ref}$
Reference layer others (mb_type, sub_mb_type, ref_idx, …)	$0.013 * \text{NumMB}_{ref}$

***NumMB$_{ref}$**: number of macroblocks of reference layer*

Table 3.5. External memory access of SVC decoder

Name	Size (KB)
Input bitstream	$0.0375 * \sum_{n=0}^{d-1} \text{PicSizeInMbs}_n$
Reference samples	$1.35 * \sum_{n=0}^{d-1} \text{PicSizeInMbs}_n$
Reconstructed samples	$0.375 * \sum_{n=0}^{d-1} \text{PicSizeInMbs}_n$
Quality coefficients	$0.75 * q * \sum_{n=0}^{d-1} \text{PicSizeInMbs}_n$

***d**: number of spatial layers*

***PicSizeInMbs$_n$**: number of macroblocks in a frame*

***q**: number of FGS layers*

b) Row-based

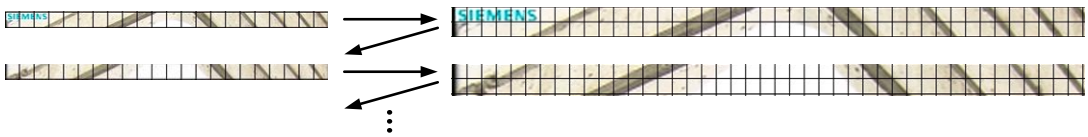
Row based decoding expands the decoding flow to row by rows. After decoding one row of macroblocks in base layer, the corresponding two rows in higher enhancement layer will be decoded as shown in Fig. 3.3. In this flow, inter-layer data is reused for enhancement layer after one row processing. Row-based decoding flow is similar to macroblock-based decoding flow since both methods decoding multiple layers at the same time. However, the control of row-based decoding method is much easier than macroblock-based method since the decoding process change between each layer is more regular, i.e. the decoding macroblocks are continuous in each layer. The main difference of memory requirement between row-based and macroblock-based is that the size of inter-layer data. The row-based manner should buffer whole row(s) of inter-layer data. The $NumMB_{ref}$ in Table 3.4 in this method is set as follows:

$$NumMB_{ref} = \sum_{n=0}^{d-2} PicWidthInMbs_n \tag{3.2}$$

***d**: number of spatial layers*

***PicWidthInMbs$_n$**: number of macroblocks in a row in spatial layer n*

The external memory access is almost the same as macroblock-based method.



(a)

```
while( spatial_layer_id < d ){
  while( CurrMbAddr < PicWidthInMbs * 2^spatial_layer_id ){
    decode_one_macroblock();
    CurrMbAddr ++;
  }
  spatial_layer_id ++;
}
```

(b)

***spatial_layer_id**: the index of current decoding spatial layer*

***d**: number of spatial layers*

***CurrMbAddr**: current decoding macroblock address*

***PicWidthInMbs**: picture width in the unit of MBs*

Fig. 3.3. Row-based decoding flow: (a) graphical illustration; (b) pseudo code

c) Frame-based

Frame-based decoding means that the decoder processes each frame layer by layer. In this flow, inter-layer data is reused for enhancement layer after one frame processing as shown in Fig. 3.4. The frames of enhancement layer are decoded after the base layer decoding. This method has to store a whole frame of inter-layer data. The amount of inter-layer data is to substitute (3.3) into Table 3.4.

$$\mathrm{NumMB}_{ref} = \sum_{n=0}^{d-2} \mathrm{PicSizeInMbs}_n \qquad (3.3)$$

***d**: number of spatial layers*

***PicSizeInMbs**$_n$: number of macroblocks in spatial layer n*

However, these huge inter-layer data are unreasonable to be stored in internal memory. Furthermore, due to different layers are decoded at different time, neighboring data can be stored in only one set of highest spatial layer without overlap. Therefore, the internal memory size is reduced to by setting $d = 1$ in Table 3.3 compared to macroblock-based and row-based decoding manners. The total external memory access is same as Table 3.5 plus inter-layer data mentioned above.

(a)

```
while( spatial_layer_id < d ) {
  while( CurrMbAddr < PicSizeInMbs ) {
    decode_one_macroblock();
    CurrMbAddr ++;
  }
  spatial_layer_id ++;
}
```

(b)

***patial_layer_id**: the index of current decoding spatial layer*

***d**: number of spatial layers*

***CurrMbAddr**: current decoding macroblock address*

***PicSizeInMbs**: picture size in the unit of MB*

Fig. 3.4. Frame-based decoding flow: (a) graphical illustration; (b) pseudo code

3.1.3. Analysis

For a clearer picture of the memory usage, we show some quantitative results in this Section. To calculate the memory usage, we make several assumptions in our analysis: 95% of macroblocks are coded in inter prediction, 90% of macroblocks in enhancement layers are coded in Intra_BL mode if the corresponding block in base layer is encoded as Intra mode, and 10% macroblocks in enhancement layers use

residual prediction in average. This assumption is a general statistic according to our experiment. The encoding settings are listed in Table 3.6. Table 3.7 shows the analysis results, in which Type I stores all inter-layer prediction data in the internal storage while Type II stores all inter-layer prediction data in the external memory.

Table 3.6. Simulation settings

GOP	8
QP	32, 26, 20
Intra period	-1
Frame resolution	CIF, 4CIF, 16CIF

Table 3.7. Comparison of memory requirements

Decoding Flow		**Internal Memory (KB)**		**External Memory Access (MB)**	
		size	**ratio**	**size**	**ratio**
Original H.264		27.9	100%	11.5	100%
Type I	MB	50.6	181%	21.7	189%
	Row	126.6	454%	21.5	187%
	Frame	2529	9065%	21.4	186%
Type II	MB	44.3	159%	24.2	210%
	Row	43.5	156%	24	209%
	Frame	28.2	101%	23.9	208%

The result shows that inter-layer prediction data has great impact to both internal memory storage as well as the external memory access. For type I, internal memory size will be increased by 81% to 8965% when compared to single layer H.264 decoding, especially for frame-based decoding that needs to store 1980 MBs of

residuals, prediction modes and motion vectors for inter-layer prediction. For row-based decoding method, 110 MBs of inter-layer prediction data need to be stored in internal memory. For MB-based decoding, only 5 MBs data are needed. This is the reason why the macroblock-based decoding method results in lower internal memory usage. For external memory access, all these flows are the same due to the same reference data. Thus, the MB-based decoding is the best choice due to its smallest internal memory usage and the same external memory access when compared to frame-based and row-based decoding methods, if an efficient internal memory design can be supported by the technology provider.

Beyond type I, another design possibility is to store the inter-layer prediction data into external memory to reduce the chip cost, just as type II. From the table, it is interesting to find that the extra external memory bandwidth due to inter-layer prediction data is insignificant compared to the large reference data. Thus, the bandwidth increasing in type II is just 12% more when compared to that in type I. However, the internal memory usage varies a lot for different coding flow. MB-based decoding has the least reduction due to each layer has its own neighboring data to be stored in internal memory for each layer decoding. Same situation also occurs in row-based decoding method as well. For frame-based decoding, the internal memory storage is just 1% more than the single layer H.264 decoding since all the extra storage is within the external memory now. Therefore, the frame-based decoding is the best choice for smallest internal memory size with the acceptable memory bandwidth.

3.2. Pipeline Architecture

In the hardware design of video coding, pipeline architecture is widely used in previous works [10][11][12][13][14][15]. The components in H.264/AVC decoder are entropy decoding (CAVLD/CABAD), residual decoding (IQ/IT), intra prediction (INTRA), inter prediction (INTER), and deblocking filter (DEBLOCK).

SVC decoder is more complicated than H.264/AVC since it supports spatial scalability and quality scalability. We will discuss the quality scalability in later Section. The three new features in spatial scalability, inter-layer intra texture prediction (IL-INTRA), inter-layer motion prediction (IL-MOTION), and inter-layer residual prediction (IL-RESIDUAL), should be also put into consideration. In terms of functionality, IL-INTRA is a new prediction type in SVC, IL-MOTION can be seen as a special case in INTER, and IL-RESIDUAL is a part in residual decoding parallel to IQ/IT.

In Inter Prediction, there are three processes to go: “Motion Vector Generation”, “Reference Data Accessing”, and “Interpolation”. Our proposed INTER stage is decomposed into two stages. The first stage, named MVG, has “Motion Vector Generation” and “Reference Pixel Accessing” processes, and the second stage, named INTERP, contains “Interpolation” and pixel reconstruction. The pipeline architecture in our design is illustrated in Fig. 3.5.

Our proposed design has four MB-pipeline stages. In the 1st MB Stage, CAVLD/CABAD outputs coefficient data and other information. Then, the 2nd MB Stage generates residuals by IQ/IT. Other prediction information such as motion

vectors for motion compensation is generated by MVG. Moreover, if current macroblock contains inter-layer prediction mode, IL-INTRA, IL-MOTION, or IL-RESIDUAL will upsample the corresponding data of co-located block in base layer as the inter-layer prediction data. In the 3rd MB Stage, either INTERP or INTRA will be activated dependents on MB prediction type. RECONST is used to reconstruct inter-layer intra texture prediction and intra-inter prediction mode in SVC enhancement layers. These reconstructed samples and MB information are sent to the 4th MB Stage, DEBLOCK, to generate the final filtered pixels.

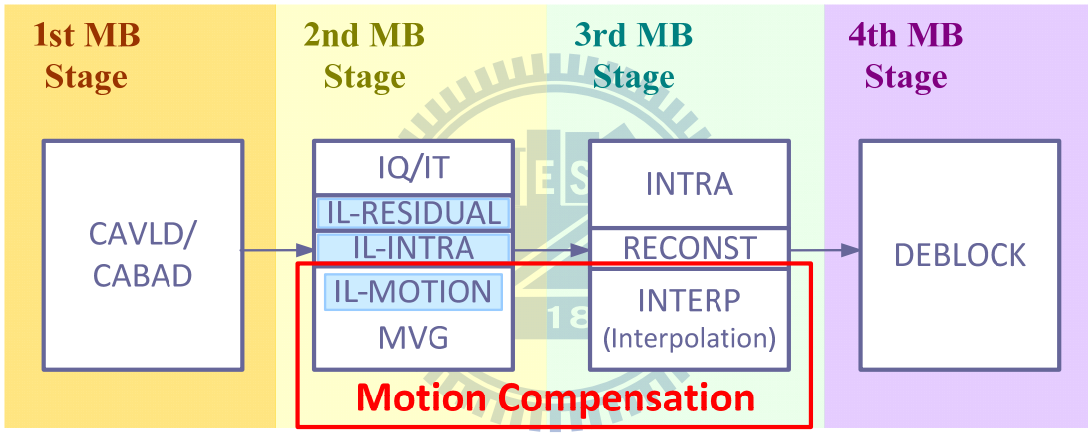


Fig. 3.5. Proposed pipeline stage in our SVC decoder

3.3. Proposed One-Pass Quality Layer Decoding

The target specification of our design is: three spatial layers (CIF, SD 480p, and HD 1080p), three quality layers, and three temporal layers, with YUV420 format in 60 Hz frame rate. Fig. 3.6 shows the structure of SVC bitstream for our target design specification. The SVC contains spatial, temporal, and quality scalabilities. In one temporal domain, there are 3x3 layers of different spatial and quality combinations as

shown in Fig. 3.7. In the pipeline architecture described in Section 3.2, the decoding flow is macroblock by macroblock in one layer, and layer by layer in one temporal domain as illustrated in Fig. 3.8. On the other hand, there are three quality layers in one spatial domain, so the total macroblocks needed to be decoded in SVC may be three times larger than former H.264/AVC standard.

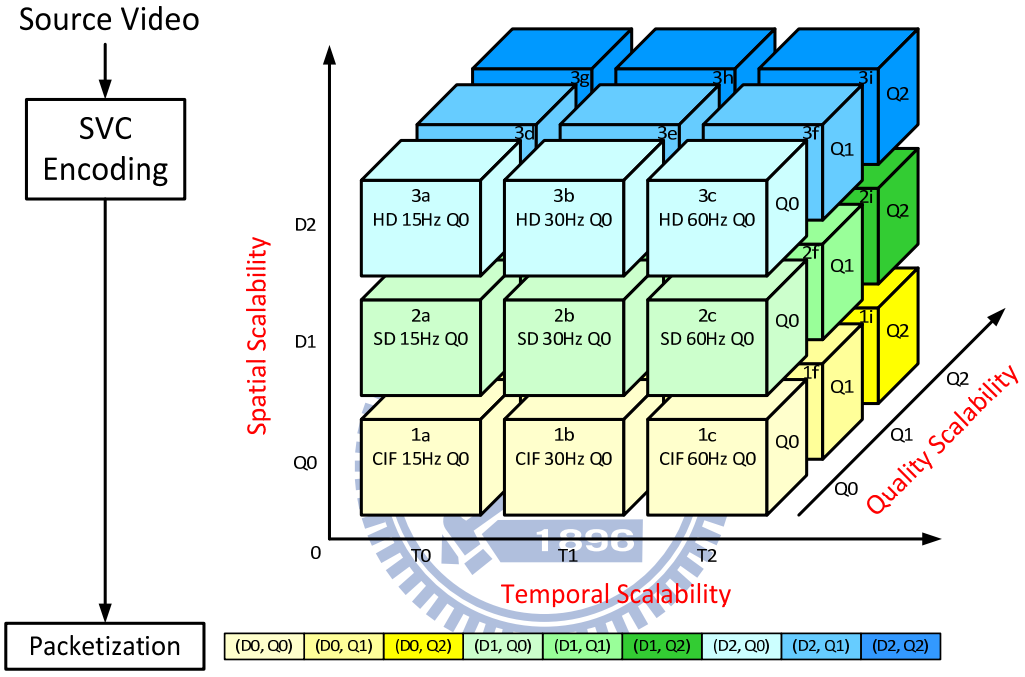


Fig. 3.6. Structure of the SVC bitstream

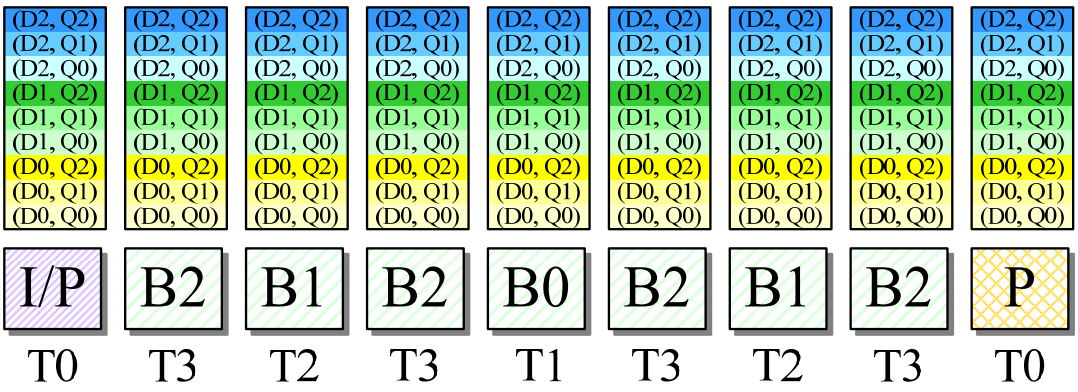


Fig. 3.7. Packets in different temporal domain

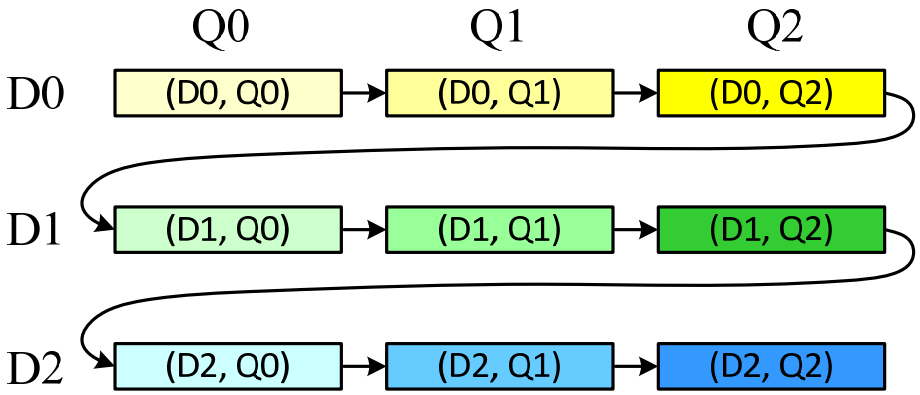


Fig. 3.8. The order of slice packet in SVC bitstream

For example, the number of macroblocks in one frame in a 1080p video sequence with three spatial layers (CIF, 480p, and 1080p) in SVC is

$$(352 / 16) \times (288 / 16) = 396 \tag{3.4}$$

$$(720 / 16) \times (480 / 16) = 1{,}350 \tag{3.5}$$

$$(1{,}920 / 16) \times (1{,}088 / 16) = 8{,}160 \tag{3.6}$$

$$396 + 1{,}350 + 8{,}160 = 9{,}906 \tag{3.7}$$

If every MB needs *d* cycles for processing, the total MB processing cycles will be 9,906*d*.

The number of macroblocks in one frame in a 1080p video sequence with three spatial layers (CIF, 480p, and 1080p) and three quality layers in SVC will be

$$3 \times (396 + 1{,}350 + 8{,}160) = 29{,}718 \tag{3.8}$$

If the decoding flow in SVC follows the pipeline architecture macroblock by macroblock and layer by layer, the total MB processing cycles will be 29,718*d*, which is three times larger than 9,906*d* with only one quality layer. The situation is: First, the

macroblocks of base quality layer go through these pipeline stages just like H.264 bitstream, then the quality enhancement layer. After the refinement coefficients of current enhancement layer are decoded, these coefficients are added to coefficients of previous layer, and the transformed residuals are added to the prediction samples of previous layer, too. It can be seen that the data in previous decoded layer will be used in quality enhancement layers. That is, these data are stored to external memory after current quality layer, and then loaded to internal memory in next quality enhancement layer.

In SVC, quality scalability is achieved using coarse-grain scalability (CGS) or medium-grain scalability (MGS) where the quality enhancement layers contain refinement coefficients. These reconstructed residuals in enhancement layers are added to the prediction or reconstructed samples in base quality layer for a better quality video. If we can deal with the coefficients in all quality layers at the same time, after transforming them to residuals, the residuals can be direct added to prediction samples of current macroblock without external memory access. Consequently, we proposed a One-Pass Quality Layer Decoding method in SVC decoder. The main idea of the one-pass decoding is that we parallel processing all quality layers in a spatial domain. Fig. 3.9 illustrates the pipeline stages in an example of three quality layers, Q0, Q1, and Q2, in D0 spatial domain. In the 1st MB stage, the coefficients in three different quality layers are decoded. The 2nd MB stage accumulates these coefficients and forms the highest refinement coefficients. Then, the IQ/IT module transforms these coefficients to base quality residuals and highest quality residuals. The residuals will be added to prediction samples in the 3rd MB stage. As a result, when the parallel decoding method is adopted, the number of total MB processing cycles in pipeline

architecture will be reduced from 29,718*d* to 9,906*d*, which is 66% reduction.

However, in order to parallel decoding different quality layers, the hardware of entropy decoding (CAVLD/CABAD) is doubled, one for base quality layer, and one for quality enhancement layer, and the hardware of residual decoding is increased due to additional buffers and coefficient accumulators for different quality layers as shown in Fig. 3.10. Moreover, a fast scan is needed in parser to identify the head of each quality layers in the bitstream as illustrated in Fig. 3.11. After the position of each quality layer is located, the parser can parallel parse the MB data in different quality layers in the bitstream to entropy decoding.. With the help of One-Pass Quality Layer Decoding method, the MB processing cycles with three quality layers is 66% reduced and is just the same with only one quality layer. In other words, the additional MB processing cycles in MB-pipeline architecture and external memory accessing due to quality scalabilities are eliminated.

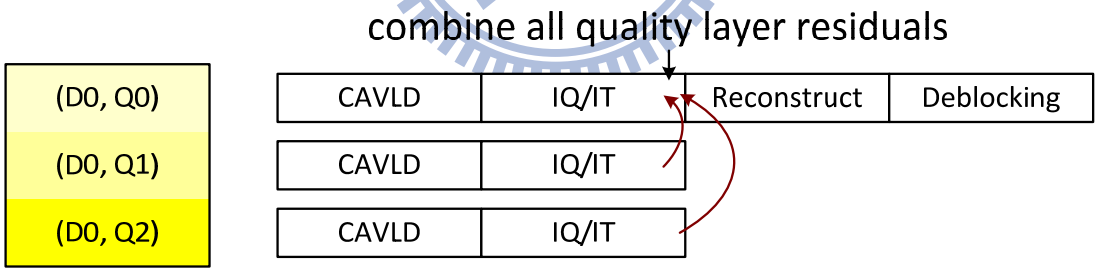


Fig. 3.9. One-pass quality layer decoding concept with three quality layers

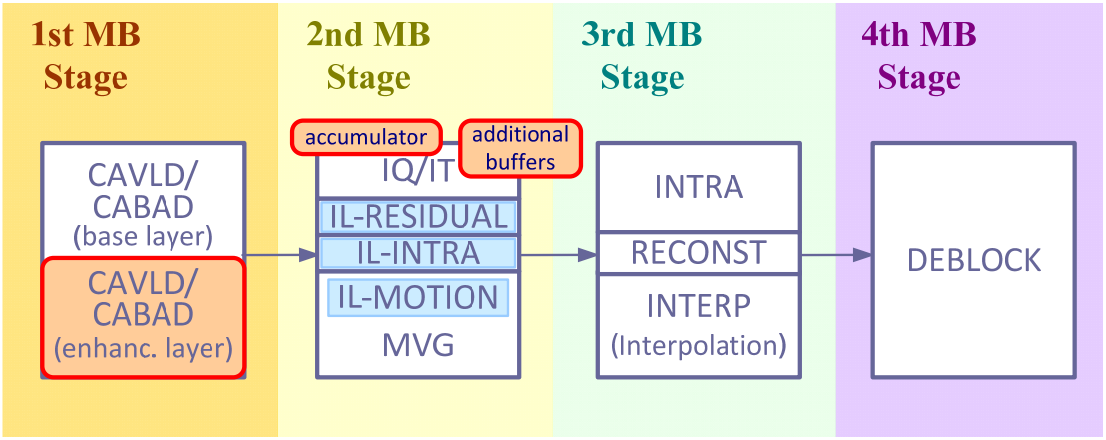


Fig. 3.10. Hardware increased with one-pass quality decoding method

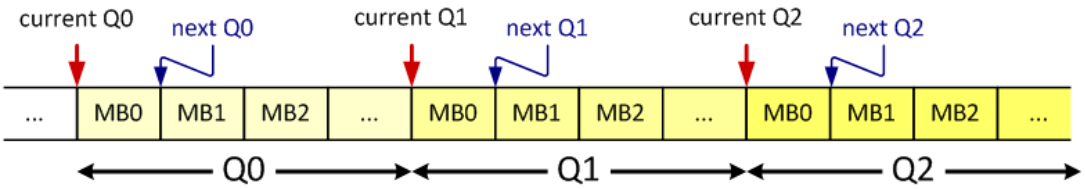


Fig. 3.11. The identifiers of different quality layers

3.4. Summary

In this chapter, the analysis of SVC decoder is discussed. First, a frame-based decoding flow is adopted in our hardware design for smallest internal memory size with acceptable memory bandwidth requirement. Second, the pipeline stage in our proposed hardware architecture is presented. We use four-stage MB-pipeline architecture for our proposed design. Finally, One-Pass Quality Layer Decoding method is introduced in SVC decoder to eliminate the additional MB processing cycles in MB-pipeline architecture and external memory accessing due to quality scalabilities.


E S A
1896

Chapter 4. Motion Compensation Design

The implementation of motion compensation design is introduced in this chapter. First, an introduction of the algorithms to motion compensation will be presented. Then, the bandwidth optimization methods used in this work will be introduced. Finally, the hardware implementation, experiment results and the conclusion will be discussed.

4.1. Introduction

With several advanced coding features, such as variable block size and complex spatial motion vector prediction, H.264/AVC stands for the state-of-the-art of current video codec. However, this high irregularity of block size and motion vector prediction algorithm also makes it the main challenges of motion compensation hardware implementation. The main design issues are to lower the computation complexity and decrease memory bandwidth requirement.

Fig. 4.1 shows the variable block size in H.264/AVC standard. The minimum block size in the standard is 4x4 block. Thus, it is an intuition to decompose a macroblock to 4x4 blocks with double-z scan order as shown in Fig. 4.2, which is 4x4-block pipeline. The motion vector of current block is added motion vector difference (mvd), which is decoded from bitsream to motion vector predictor (MVp), which is generated by neighboring blocks. The accuracy of motion vectors in H.264/AVC standard is a quarter of a pixel. In the case of fractional MVs, 6-tap FIR filter is used to interpolate fractional pixels in luminance component. For a 4x4 block,

it needs 9x9 reference pixels to do interpolation. In this case, large amount of memory bandwidth is required in fractional MV blocks. To reduce the bandwidth requirement of external memory, some bandwidth optimization methods are used.

The algorithms of motion compensation will be introduced in the rest of this Section.

Fig. 4.1. Variable block size in inter-coded block

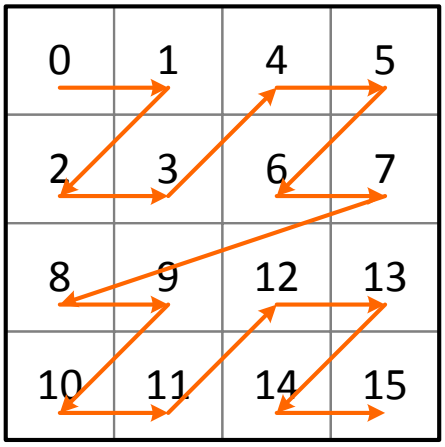


Fig. 4.2. Double-z scan order

4.1.1. Motion Vector Predictor

Since the motion vectors of neighborings are often highly correlated, the motion vector of each block is predicted from previously coded partitions, and only the prediction error is transmitted in H.264/AVC standard to reduce the bit rate. In motion vector prediction process, the first thing is to generate motion vector predictor (MVp), then add it together with the decoded motion vector difference (mvd). The derivation process for MVp is described as below and shown in Fig. 4.3:

- For macroblock partitions excluding 16x8 and 8x16 partition sizes: MVp is the median of the motion vectors for partitions A, B and C.
- For 16x8 partitions: MVp for the upper 16x8 block is predicted from B, MVp for the lower is predicted from A.
- For 8x16 partitions: MVp for the left 8x16 block is predicted from A, MVp for the right is predicted from C.

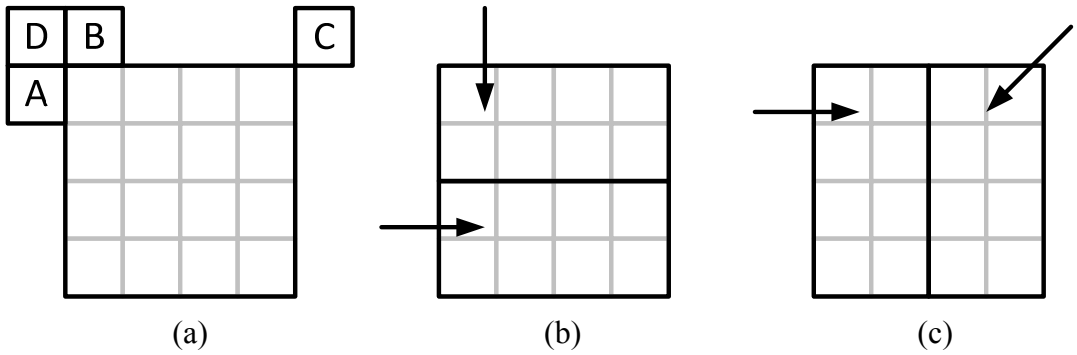


Fig. 4.3. Motion vector predictor scheme (a) macroblock partitions excluding 16x8 and 8x16 partition sizes, (b) 16x8 partitions, (c) 8x16 partitions

4.1.2. Fractional Pixel Interpolation

The accuracy of motion vectors in H.264/AVC standard is a quarter of a pixel. In the case of fractional MVs, the fractional pixels of luminance component are generated by interpolation of the integer-pixels. The interpolation method is based on 6-tap FIR filter with tap values (1, -5, 20, 20, -5, 1). Fig. 4.4 shows the interpolation scheme of luminance component. The half-pixels, b, h, m, and s, are derived by applying 6-tap FIR filter using integer-pixels as inputs.

$$b_1 = (E - 5 * F + 20 * G + 20 * H - 5 * I + J) \quad (4.1)$$

$$b = \mathrm{Clip1}_Y((b_1 + 16) >> 5) \quad (4.2)$$

The half-pixel j is obtained by first calculating the intermediate values of the six half-pixel locations in the horizontal or vertical direction then applying 6-tap FIR filter with these intermediates as shown in equation (4.3) to (4.5).

$$j_1 = cc - 5 * dd + 20 * h_1 + 20 * m_1 - 5 * ee + ff, \text{ or} \quad (4.3)$$

$$j_1 = aa - 5 * bb + 20 * b_1 + 20 * s_1 - 5 * gg + hh \quad (4.4)$$

$$j = \mathrm{Clip1}_Y((j_1 + 512) >> 10) \quad (4.5)$$

Table 4.1 shows the bit-width of data during luma interpolation. Notice that the input bit-width of the interpolation process in half-pixel j is 15-bit. Fortunately, a simplification to equation (4.3) to (4.5) can make the implementation much easier with negligible quality degradation at about 0.01 dB [16] in which the intermediates are truncated to 8-bit.

$$j_1 = cc' - 5 * dd' + 20 * h + 20 * m - 5 * ee' + ff', \text{ or} \quad (4.6)$$

$$j_1 = aa' - 5 * bb' + 20 * b + 20 * s - 5 * gg' + hh' \quad (4.7)$$

$$j = \mathrm{Clip1}_Y((j_1 + 16) >> 5) \quad (4.8)$$

Fig. 4.5 shows the interpolation scheme of quarter-pixels. The quarter-pixels labeled as a, c, d, n, f, i, j k, and q are derived by averaging two nearest integer-pixel and half-pixel. The quarter-pixels e, g, p, and are derived by averaging two nearest half-pixels in diagonal direction.

Table 4.1. Bit-width of data during first luma interpolation

Interpolation	**Min**	**Max**	**Bit-width**
x	0	255	8
-5x	-1275	0	12
20x	0	5100	14
Σ	**-2550**	**10710**	**15**
(Σ+ 16)>>5	-80	335	10
Clip((Σ+ 16)>>5)	0	255	8

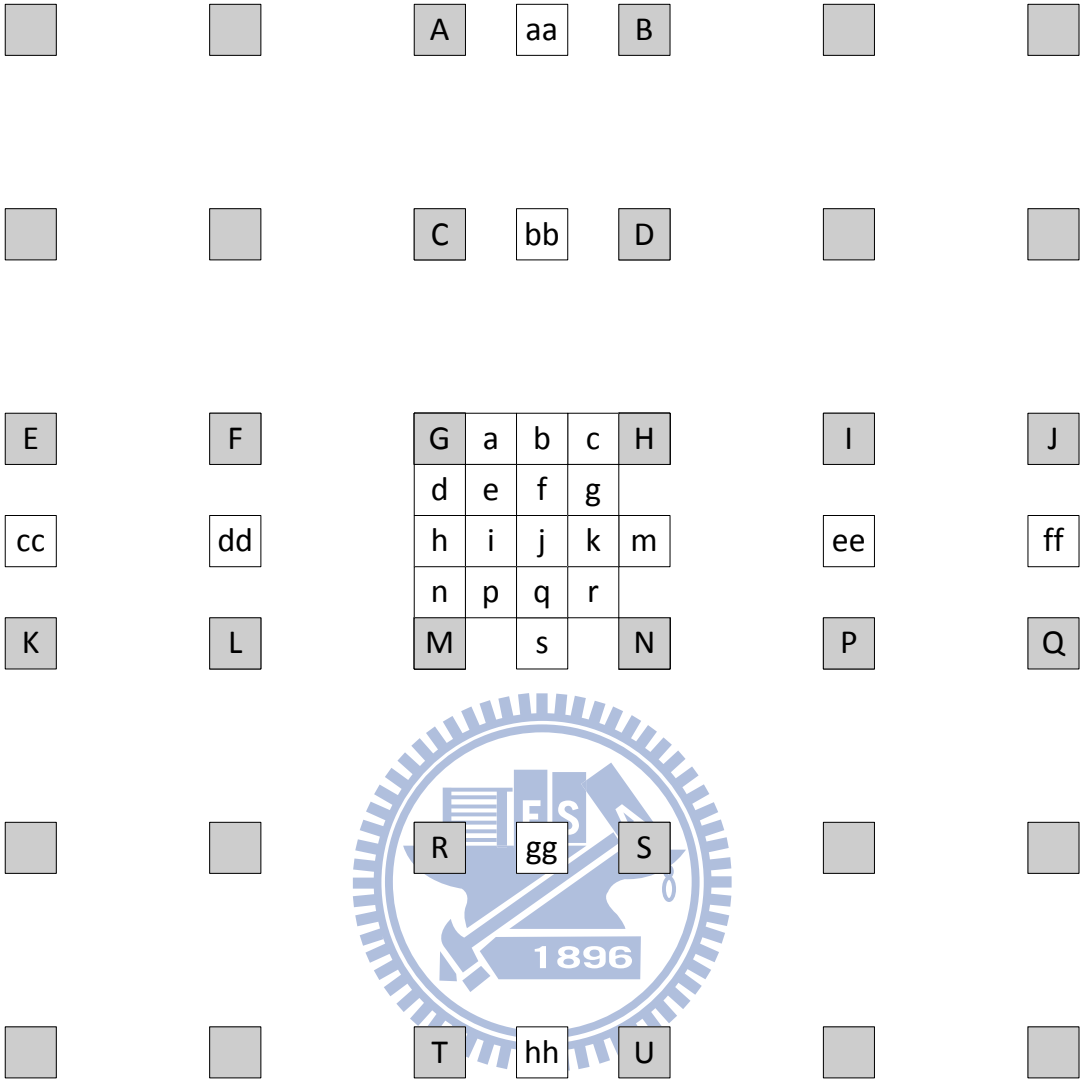


Fig. 4.4. Interpolation scheme for luminance component (grey blocks represent integer pixels, which are denoted by upper-case letter)

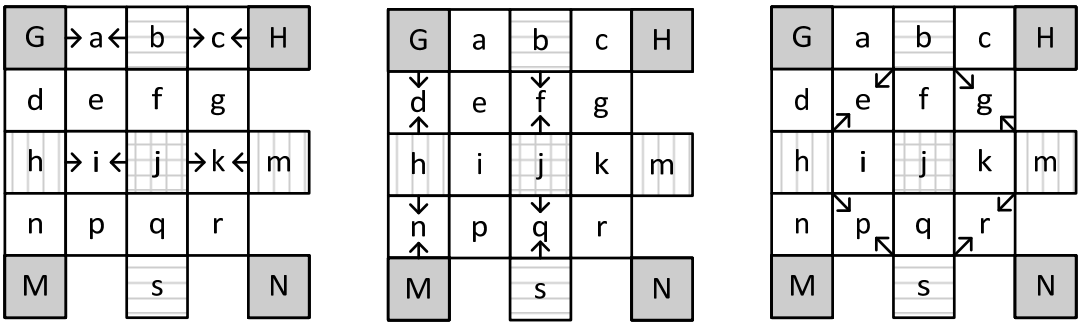


Fig. 4.5. Interpolation scheme of quarter-pel positions for luminance component

Fractional pixels of chrominance component are derived by averaging weighted samples of nearest four integer pixels. The interpolation scheme of chrominance component is shown in Fig. 4.6 and the equation is:

$$a = ((8 - x) * (8 - y) * A + x * (8 - y) * B + (8 - x) * y * C + x * y * D + 32) >> 6 \quad (4.9)$$

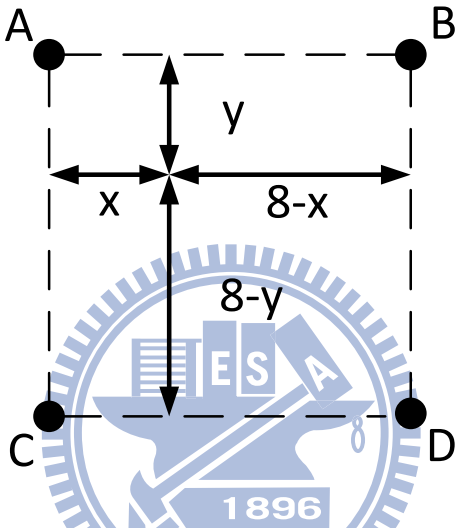


Fig. 4.6. Interpolation scheme for chrominance component

4.2. Bandwidth Optimization

The high memory bandwidth requirement in motion compensation is the bottleneck in video decoder design. To alleviate this situation, we first reuse the overlapped data inside a partitioned block, and then we reduce the required reference data according to fractional-pel position. These two bandwidth optimization methods are discussed in this Section.

4.2.1. Block Size Based Data Request

The partition size in a macroblock is often larger than 4x4, and the reference data of 4x4 blocks in a same block partition is highly overlapped as shown in Fig. 4.7. The methods of reusing this overlapped data have been realized in several ways [11] [12]. In [11], a Vertical Integrated Double Z (VIDZ) flow adding a 21x64-bit on-chip memory to reuse the vertical and horizontal overlapped regions between two 4x4 decomposed blocks. In [12], an exploiting data reuse in hybrid block size memory access from 4x4 to 8x8 is presented, which reuse the overlapped data inside a 4x8, 8x4, or 8x8 block. However, the external data requests of these methods are based on small block size such as 4x4 to 8x8, and the numerous external data access may influence the latency of MC hardware.

In 4x4-block pipeline design, the general case of memory access scheme is loading 9x9 reference pixels as the interpolation window for a decomposed 4x4 block; we call it *Conventional 4x4 Based Data Request*. The data reuse only existed between two neighboring blocks with additional buffers. To increase the reusing rate of overlapped data and reduce the frequency of external data access, the processing element in our proposed design is scaled up to the block partition size, and we call it *Block Size Based Data Request*. For example, a 16x8 block consisting of eight 4x4 blocks, instead of requesting 9x9 block eight times, the reference data request would be only one 21x13 block. The reference data in both requests are the same, but the request is reduced from eight times to only one and the accessing pixels of reference data request is down from 648 pixels to 273, which is 58% reduction. The ideal reduction rates corresponding to different partition size are shown in Table 4.2.

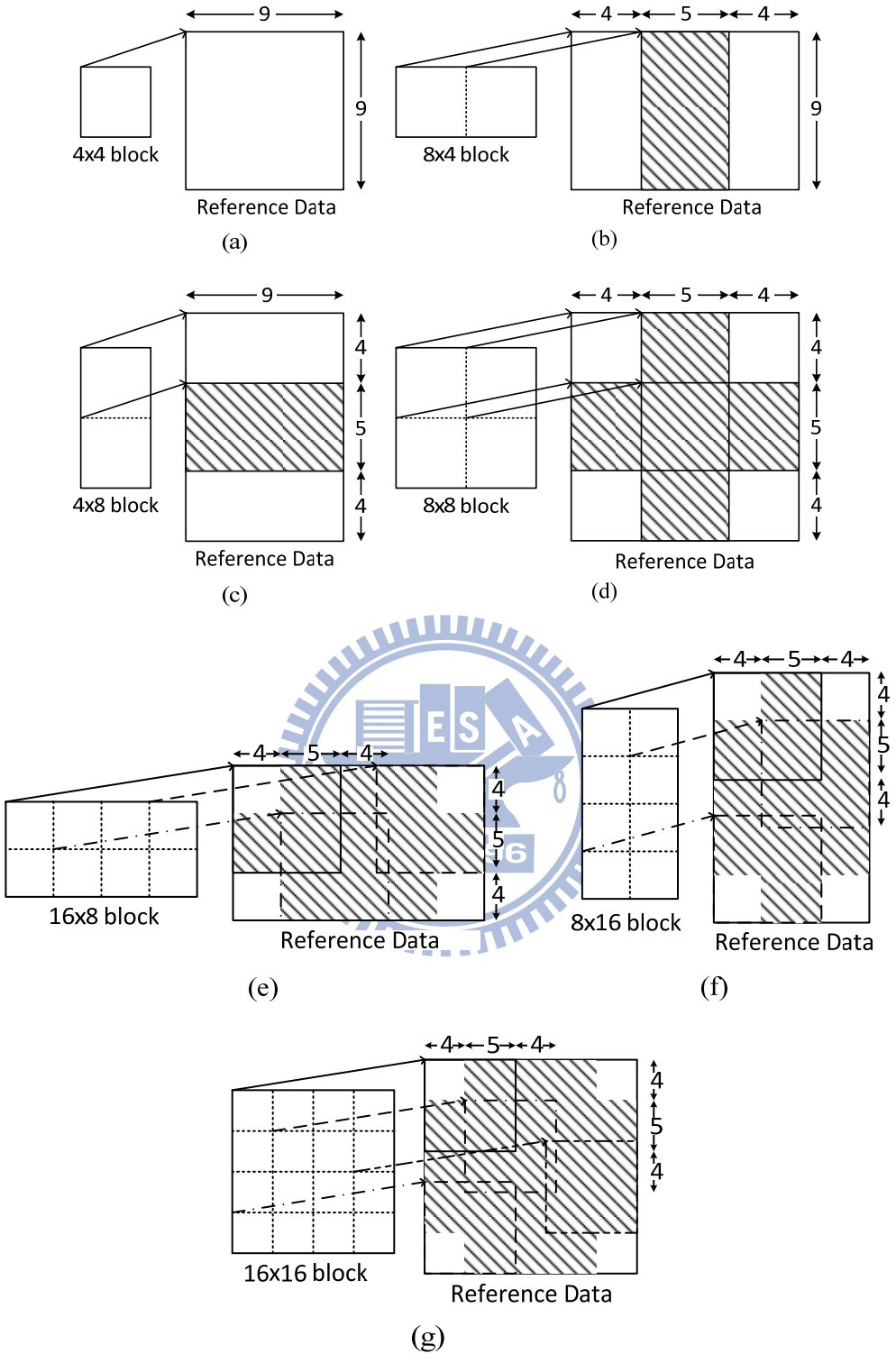


Fig. 4.7. Data reuse of case: (a) 4x4 block, (b) 8x4 block, (c) 4x8 block, and (d) 8x8 block, (e) 16x8 block, (f) 8x16 block, (g) 16x16 block (shaded region means reusable)

Table 4.2. Ideal reduction of reference data accessing between C*onventional 4x4 Based Data Request* and *Block Size Based Data Request*

Partition Size	*4x4 Based*	*Block Size Based*	Reduction
4x4	9 x 9 = 81	9 x 9 = 81	0 %
8x4	2 x 9 x 9 = 162	13 x 9 = 117	28 %
4x8	2 x 9 x 9 = 162	9 x 13 = 117	28 %
8x8	4 x 9 x 9 = 324	13 x 13 = 169	48 %
16x8	8 x 9 x 9 = 648	21 x 13 = 273	58 %
8x16	8 x 9 x 9 = 648	13 x 21 = 273	58 %
16x16	16 x 9 x 9 = 1296	21 x 21 = 441	66 %

4.2.2. Precision Based Data Request

A strategy for the interpolation filters and the valid reference data size of different positions has been proposed [18]. In 4x4-block pipeline design, it is inefficient to load 9x9 reference pixels for a decomposed 4x4 block to interpolator since the required reference data is not always need as large as 9x9. Fig. 4.8 shows integer-pixel and fractional-pixel positions in H.264/AVC standard which has the accuracy of a quarter of a pixel. To minimize the memory bandwidth access of motion compensation, a classification of different pixel positions should be discussed. In Fig. 4.8, the sub-pixel a, b and c are located at vertical integer positions and being interpolated by horizontal interpolation only (vertical interpolation is not used), which means the required reference data can be reduced to 9x4 as shown in Fig. 4.9 (b). Same situation comes in sub-pixel d, h and n, the horizontal interpolation is not used and the required reference data is reduced to 4x9 (Fig. 4.9 (c)). In the case of vertical or horizontal

integer positions, the required reference data can be is reduced from 9x9 to 9x4, 4x9, or even 4x4. This strategy can be combined with Block Size Based Data Request in Section 4.2.1. Table 4.3 shows a classification of required reference data size for different positions with block partition size *M* x *N* (*M* for width and *N* for height of current partition).

Table 4.3. Reducing required reference data according to different pixel positions with block partition size *M* x *N*

Pixel Position	**Interpolation Filters**	**Required Reference Data**
G	None	*M* x *N*
a, b, c	Horizontal	(*M* + 5) x *N*
d, h, n	Vertical	*M* x (*N* + 5)
e, f, g, i, j, k, p, q, r	Horizontal and Vertical	(*M* + 5) x (*N* + 5)

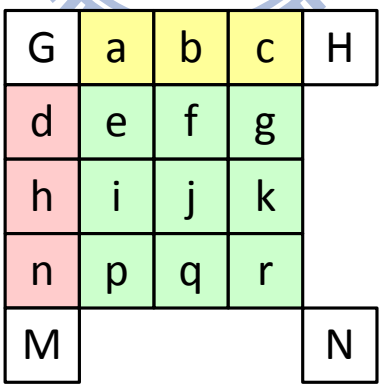


Fig. 4.8. Integer pixels (blocks with upper-case letter) and fractional pixels (blocks with lower-case letter) of luminance

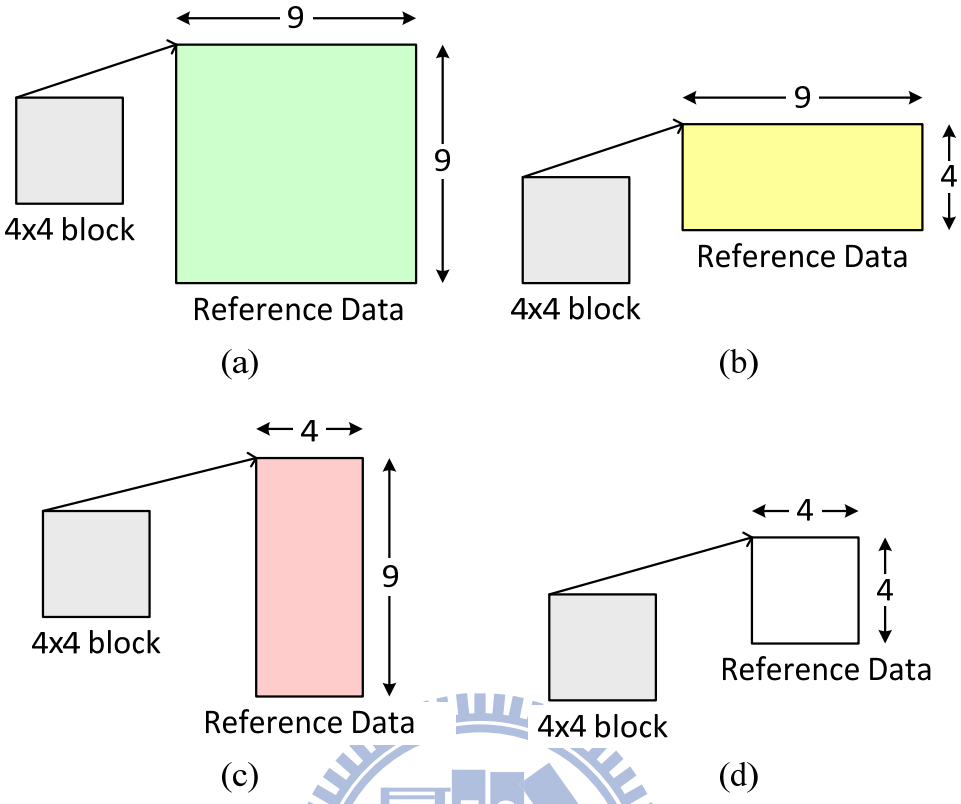


Fig. 4.9. (a) Horizontal and vertical interpolation (x and y components of MV point to fractional positions). (b) Horizontal interpolation only (y component points to integer position). (c) Vertical interpolation only (x component points to integer position). (d) No interpolation (both x and y point to integer positions)

4.2.3. Simulation Results

In order to verify the effect of the bandwidth optimization methods, a simulation is performed based on our C model with a DDR400 SDRAM model. We compare the memory bandwidth requirements with (1) *Block Size Based Data Request,* (2) *Block Size Based Data Request* and *Precision Based Data Request,* and without these bandwidth optimization methods. Six HD 1080p video sequences, *IBBBBBBBP* (GOP=8) hierarchical-B prediction structure, and four QP values are used for the simulation. As illustrated in Table 4.4, the results show that about 62-74% of the data

bandwidth is reduced with these bandwidth optimization methods. With higher QP values, the block partition size trends to large block size, and the bandwidth reduction rate will be higher.

Table 4.4. Reduction of data bandwidth in different sequences and QPs

	QP=16		**QP=24**		**QP=32**		**QP=40**	
	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)
tractor	59.57	63.5	60.72	64.7	62.05	65.86	62.52	66.83
sunflower	60.07	64.8	62.4	66.2	62.99	66.45	63.13	68.39
rush_hour	58.36	62.8	61.17	68.7	62.58	71.6	63.09	74.53
station	60.34	63.2	62.23	67.9	62.89	67.24	63.15	69.58
blue_sky	60.12	66.2	62.26	70.9	62.83	70.53	63.13	70.97
pedestrian_area	59.91	66.7	61.9	68.8	62.64	69.77	63.01	72.65
Average	**59.73**	**64.5**	**61.78**	**67.2**	**62.66**	**68.8**	**63**	**70.5**

4.3. Hardware Design

The system specification of our hardware design is an SVC decoder operating at 135MHz clock rates with 3 spatial layers (CIF, 480p, and 1080p), three quality layers, and 60 frames per second. According to the analysis in Chapter 3, a frame-based decoding flow with one-pass quality layer decoding MB-pipeline architecture, the total number of processing stages per set of frame is

$$(396 + 1{,}350 + 8{,}160) \times 60 = 594{,}360 \qquad (4.10)$$

For 135MHz clock rate, the available cycles per stage would be

$$135{,}000{,}000 / 594{,}360 = 227 \text{ cycles} \tag{4.11}$$

As a result, the constraint of processing cycles per MB-pipeline stage is 227 cycles.

There are three processes in our motion compensation design which are “Motion Vector Generation”, “Reference Pixel Accessing”, and “Interpolation”. It’s hard to finish all these works within 227 cycles. As a result, we propose a two-stage motion compensation design with separated data access and interpolation. Our proposed INTER stage is decomposed into two MB-pipeline stages, trying to reduce the processing cycles. The block diagram of proposed motion compensation architecture is shown in Fig. 4.10. The first stage, named MVG, has “Motion Vector Generation” and “Reference Pixel Accessing” processes. The main functions of the first stage are to generate MVs and collect all reference data of current MB. First, the Motion Vector Generation reconstructs MVs of current MB and then Data Request Generator in Reference Pixel Accessing generates data request to Memory Controller of external memory for accessing reference pixels. The returned reference pixels are collected in a 21x21-pixel Register Array, and then written to Reference Data Buffer for next MB-pipeline stage. The second stage, named INTERP, contains “Interpolation” and pixel reconstruction. The main function of this stage is to interpolate fractional pixels. The Interpolator produces fractional pixels from reference data, and then Pixel Reconstruction collects these interpolated pixels adding to residuals as the output reconstructed pixels.

The details of these three processes are described in follows.

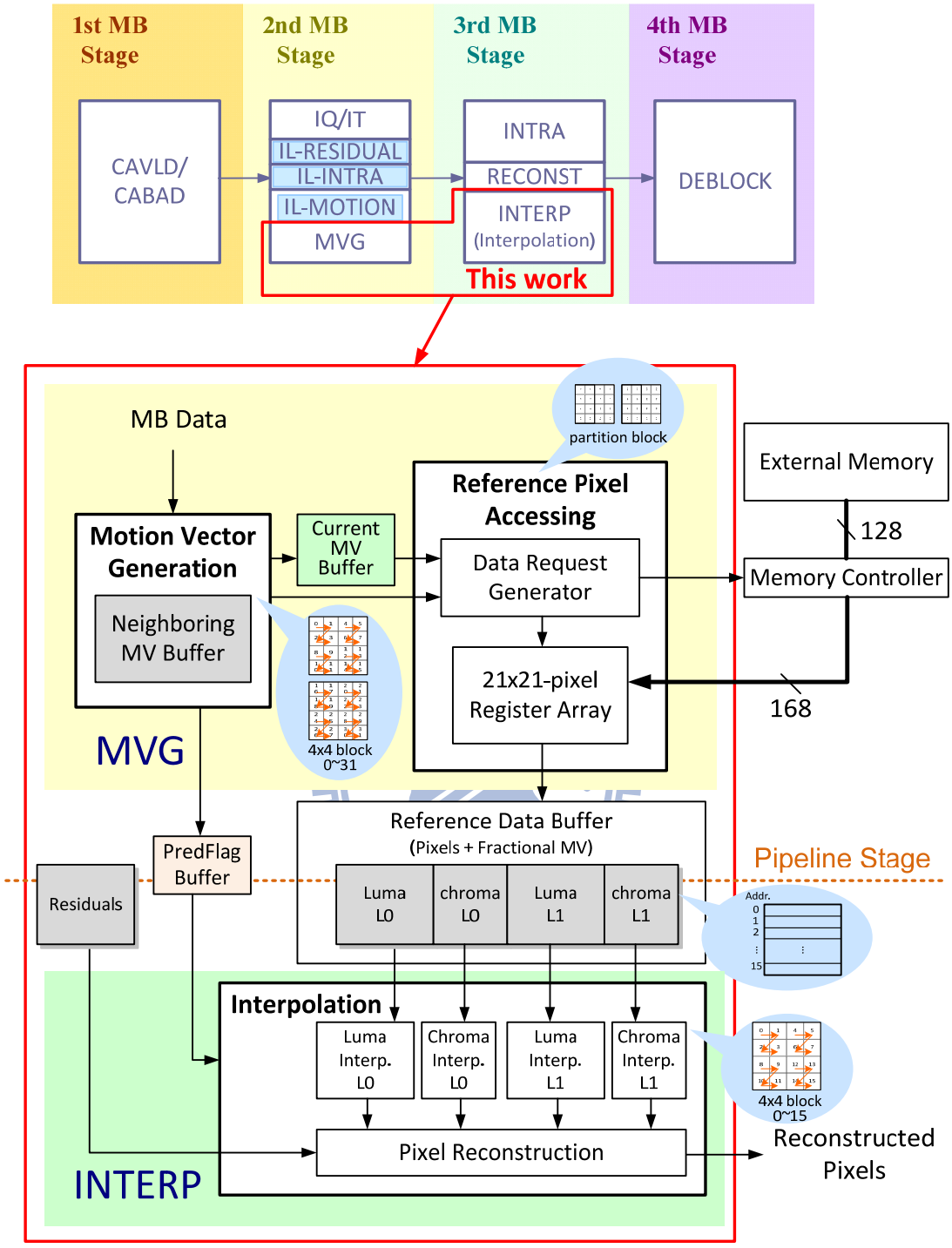


Fig. 4.10. Proposed pipeline architecture of motion compensation design

4.3.1. Motion Vector Generation

In this section, a highly regular architecture of Motion Vector Generation is presented and the reference is [20]. This process is to generate MVs of current macroblock which is first generating motion vector predictor (MVp) and then adding to the motion vector difference (mvd). The proposed motion vector prediction design takes three cycles to derive the motion data for a 4x4 block, getting neighboring MVs in cycle 1, finding out MVp of current block in cycle 2, and adding mvd to MVp in cycle 3 as illustrated in Fig. 4.11.

The prediction of inter-coded block in H.264/AVC comes from two directions, one is forward reference frame, called List0 (L0), and the other is backward reference frame, called List1 (L1). The prediction types in inter-coded block are combined with L0 and L1, which could be L0, L1, or bi-direction (both L0 and L1) prediction. The unit of the direction of prediction is based on 8x8 block, so a 2-bit predFlag, one bit for L0, and one bit for L1, is used to identify the prediction types for an 8x8 block as illustrated in Fig. 4.12. To deal with the variable block size, a 16x16 macroblock is decomposed into sixteen 4x4 forward blocks and sixteen backward blocks and the block index is shown in Fig. 4.13. The decoding flow is in the order of block index from 0 to 31. In Section 4.1.1, the algorithm of motion vector predictor is introduced. Before the process of current MB, all neighboring MVs are loaded to internal buffer and take *preload* cycles. The neighboring blocks come from upper MB, left MB or current MB, and the MVp is predicted from these neighboring MVs. The neighboring blocks of upper MB come from a Neighboring MV Buffer which stores MV information of previous decoded macroblocks as shown in Fig. 4.14. We use an

internal SRAM as the neighboring MV buffer and this SRAM stores the bottom four 4x4 blocks in a row of macroblocks, i.e., a 1080p video sequence with 120 macroblocks per row of frame. Both L0 and L1 need a SRAM and the total size is 2.46 KBytes. While an MV of a 4x4 block is reconstructed, the MV of current 4x4 block is sent to Current MV Buffer. In the case of partition size bigger than 4x4, only the first 4x4 block of this partitioned block needs to enter the process of motion vector prediction, and the rest blocks just copy the MV result of the first block. On the other hand, those rest blocks of the corresponding partitioned block can skip the motion vector prediction process. After the motion vector generation process of current MB is finished, the MVs of bottom four 4x4 blocks will be stored in the neighboring MV buffer for upper neighborings of next MB-row. The worst case occurs when the MB is consisted of 16 4x4 blocks with bi-direction prediction, and the motion vector prediction process will take *preload* + 96 cycles.

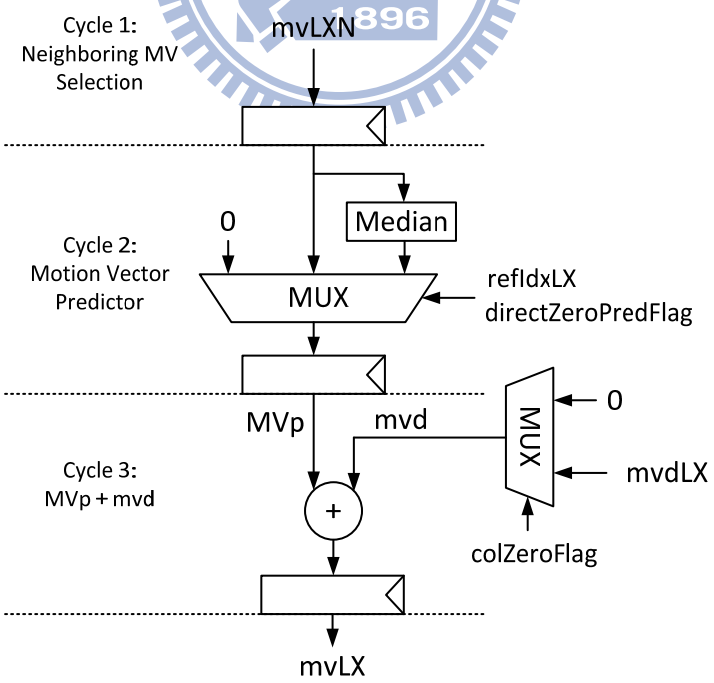


Fig. 4.11. Proposed processing element for motion vector prediction

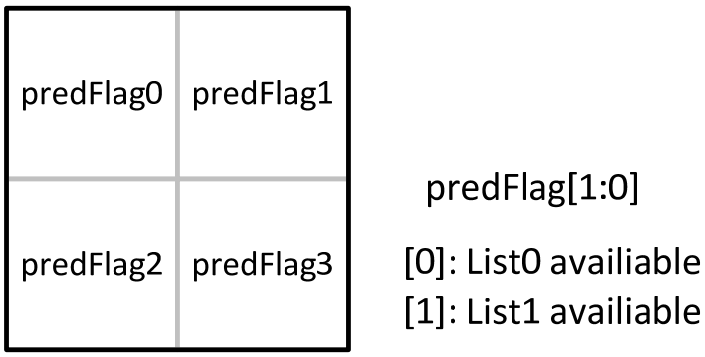


Fig. 4.12. The four prediction flags in a MB

Fig. 4.13. Block index of all 4x4 subblocks

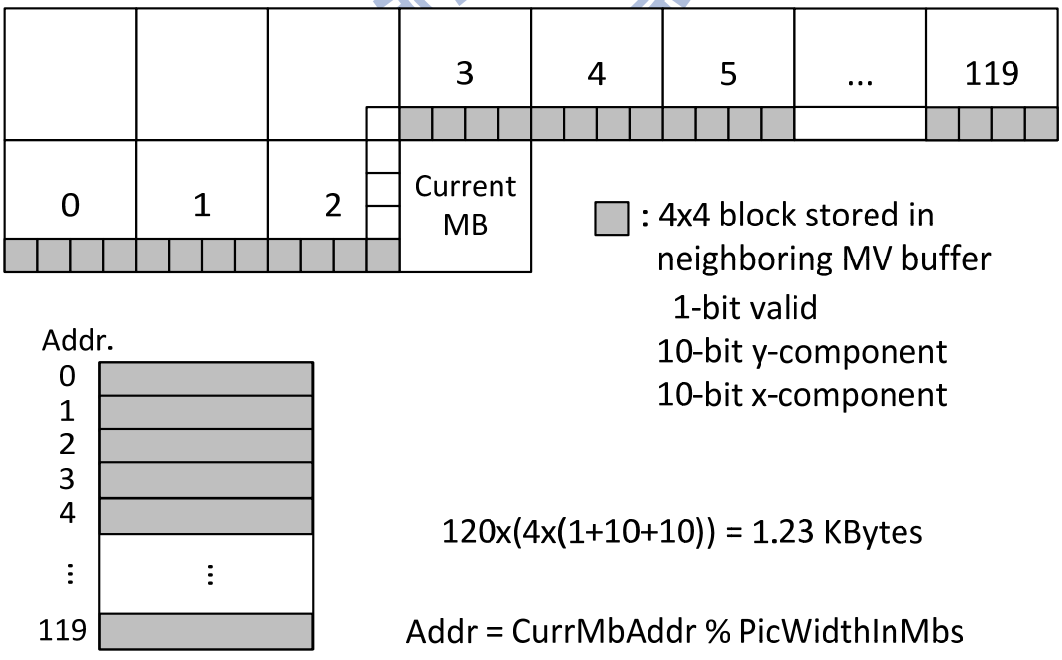


Fig. 4.14. SRAM buffer for neighboring MVs

4.3.2. Reference Pixel Accessing

After the MV of a 4x4 block is reconstructed, the current block information will be sent to Data Request Generator for accessing reference pixels in external memory. Refer to Section 4.2.2, the size of requested reference data is based on the partition size of current block. That is, a maximum 21x21-pixel register array is used in Reference Pixel Accessing for request data collection. The returned reference data from memory controller is row by row, which has the maximum bitwidth of 21-pixel (168-bit), and these data are collected by 21x21-pixel Register Array. The requested data size is not always 21x21. However, for every kinds of block size, the requested data can be fitted in this 21x21 register array from upper-left corner with a fixed position mapping to every 4x4 blocks inside current partitioned block. For example, if current requested data size is 21x21 for a 16x16 partition, these data fit into the 21x21-pixel register array. If the current partition size is 16x8, then the requested data will be 21x13. After one partition is finished, these 21x13 data in 21x21-pixel register array can be refreshed by the requested data of other partition. The 21x21 Register Array first modifies the returned reference data to recovery boundary extension, and then collects these modified data into register array. When there are enough reference data for a 4x4 block, i.e., 9 rows of luma pixels or 3 rows of chroma pixels, the corresponding reference data, which will be 9x9 luma pixels or fractional MVs, 3x3 Cb, and 3x3 Cr samples, are written to Reference Data Buffer for next MB-pipeline stage, INTERP. Fig. 4.15 shows the data mapping of the Reference Data Buffer. These buffers are consisted of 16 rows and each row corresponding to the 4x4 block index of a macroblock. The row of luma buffer contains 9x9 reference pixels of luminance components, and the row of chroma buffer includes fraction MV of x-direction,

y-direction, 3x3 Cb, and 3x3 Cr pixels of chrominance components. This Reference Data Buffer is consisted of four on-chip SRAMs which are luma and chroma buffer in both L0 and L1 direction. For regularly, we use ping-pong buffer to utilize this Reference Data Buffer, and the buffer size is 6.25 Kbytes.

In the external memory access, the returned data is row by row. That is, the reference data of current partition block will be ready at the order from left-to-right and top-to-bottom. As a result, writing reference data to Reference Data Buffer is in raster-scan order within current partitioned block as shown in Fig. 4.16.

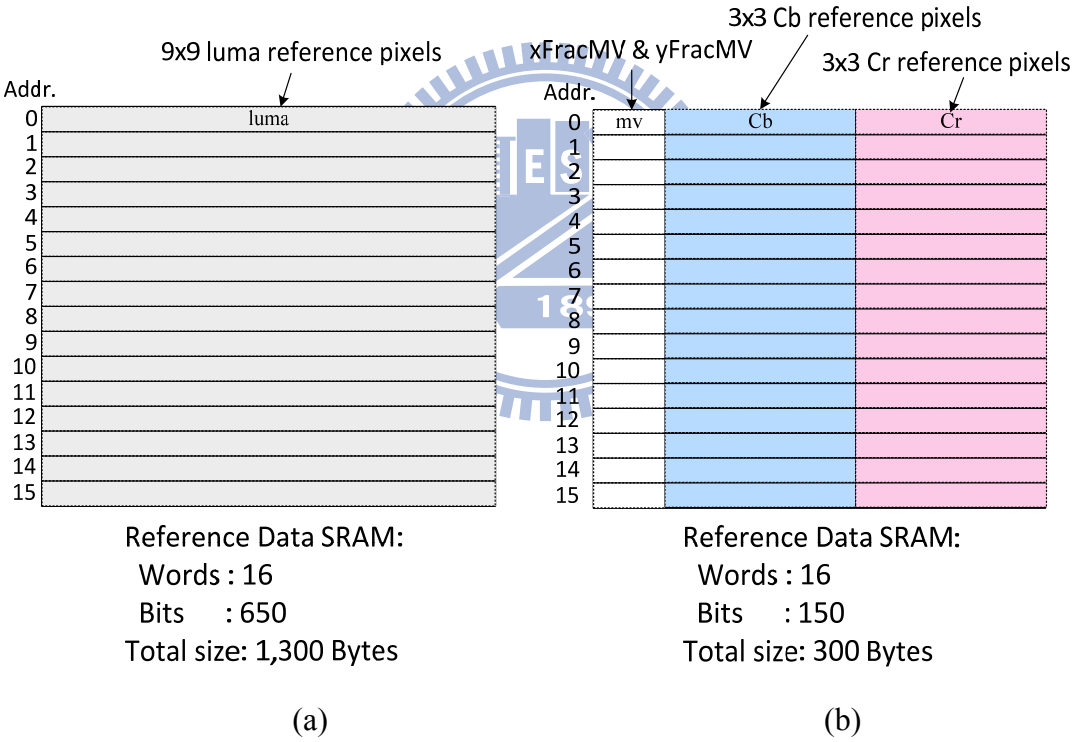


Fig. 4.15. (a) Data mapping in SRAM of luma reference data buffer, (b) data mapping in SRAM of chroma reference data buffer

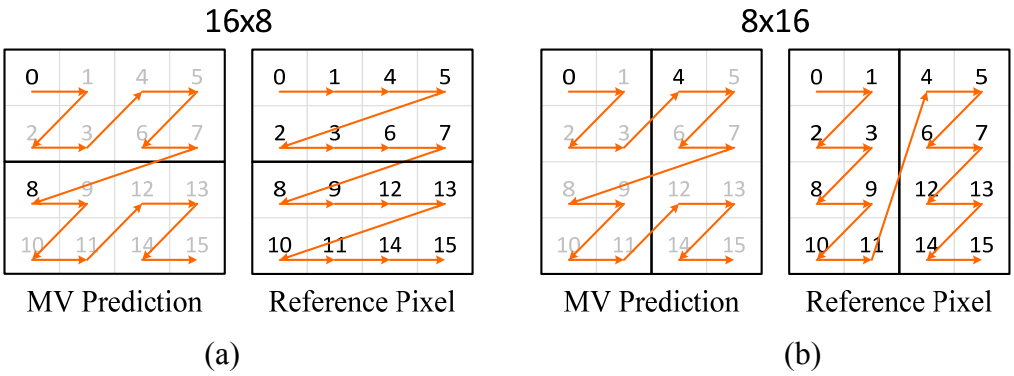


Fig. 4.16. Raster-scan order of writing reference data in (a) 16x8 and (b) 8x16 partition size

4.3.3. Interpolation

Our proposed interpolation design adopts a Doubled Hardware of Interpolation Unit (DHIU) scheme that two separate interpolation units handle L0 and L1 directions in a 4x4 block simultaneously. The interpolation unit is consisted of a luma interpolator and a chroma interpolator. The elements in a 4x4 block in YUV420 format are consisted of 4x4 luma, 2x2 Cb, and 2x2 Cr samples. In our proposed architecture design, all of the reference data for a MB is ready before current Interpolation MB-pipeline stage, so we parallel processes luminance and chrominance components. The 6-tap FIR filter design for luma is shown in Fig. 4.17, which translates multiplier-adder to shift operation and addition. In our proposed luma interpolator, separate 1-D structure is used. The components of proposed luma interpolator are thirteen 6-tap FIR filters and four bilinear filters and the throughput is 4 pixels/cycle as shown in Fig. 4.18. According to different fractional MVs of x-direction and y-direction, the processing cycles for one 4x4 block would be 5-6 cycles. For the chroma interpolator, a low-cost two-stage chroma filter design was proposed [15], and the throughput was 1 pixel/cycle. We extend the parallelism to

handle Cb and Cr components simultaneously, and the throughput of our chroma interpolator is (1 (Cb) + 1 (Cr)) pixels/cycle. The processing cycles of our proposed chroma interpolator are fixed on 5 cycles for a 4x4 block. It can be seen that the overall latency of interpolation unit for one 4x4 block is 5-6 cycles. The Pixel Reconstruction takes another 2-3 cycles and 1 cycle for changing 4x4 block index. As a result, the latency of Interpolation takes 128-160 cycles for an MB.

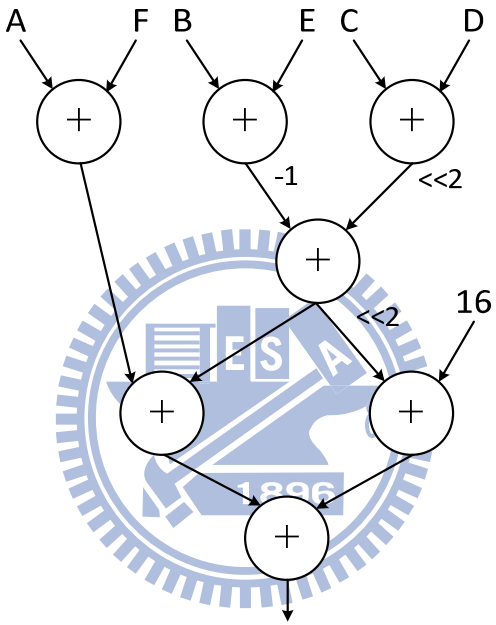


Fig. 4.17. 6-tap FIR filter design

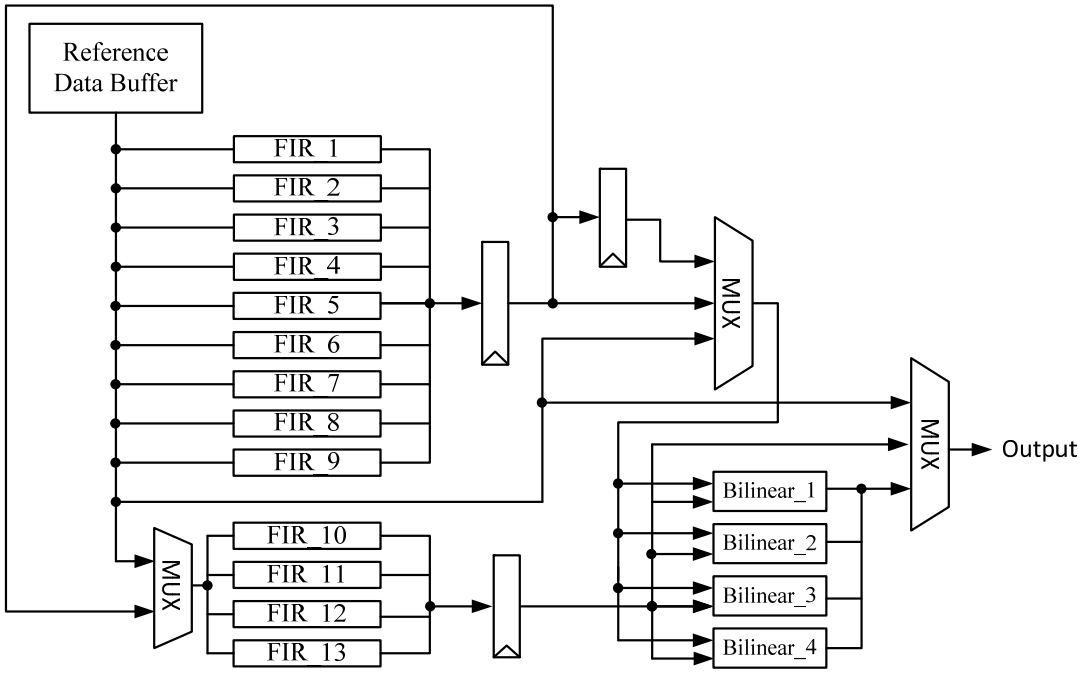


Fig. 4.18. Interpolator of luminance component

4.4. Implementation Results

This chapter summarizes the implementation results of this work. The proposed motion compensation hardware architecture is implemented in Verilog HDL with UMC 90nm 1P9M CMOS technology. The critical path of this design is in 6-tap FIR filter due to the tree adder architecture.

4.4.1. Design Flow

The design flow in our proposed design is shown in Fig. 4.19. First, the system specification is made, and we develop a behavior model in C language. Then, we formulate and analyze the design problem from algorithmic and architectural levels. After the system architecture is confirmed, the hardware design is implemented with

RTL coding. After verifying the functionality of HDL, logic synthesis and gate-level simulation are performed.

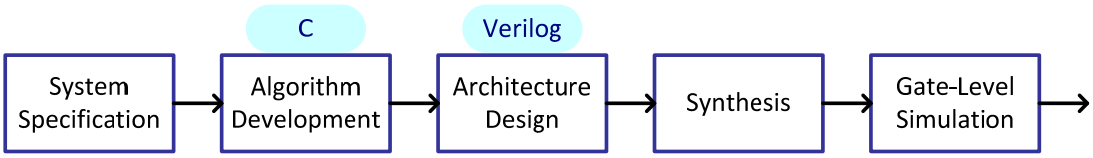


Fig. 4.19. Design flow in this work

4.4.2. Experiment Results

In order to observe the behavior of proposed design, we integrate all components with a top module. The input pattern such as *slice_type, mb_type, CurrMbAddr* and other MB information are generated from our C model, and the output results are compared with the golden data generated from C.

The experiment results in this Section are simulated in C with a DDR400 SDRAM model. We first calculate the MB processing cycles in the two stages of our MC architecture, and the overall MC processing cycle is the maximum of these two values. Table 4.5 and Table 4.6 show the results in our simulation.

Table 4.5. Average processing cycles per MB in MVG stage (motion vector generation and reference pixel accessing) and INTERP stage (interpolation)

	QP=16		QP=24		QP=32		QP=40	
	MVG	INTERP	MVG	INTERP	MVG	INTERP	MVG	INTERP
tractor	166	137	148	136	141	136	139	135
sunflower	137	136	132	136	142	138	144	134
rush_hour	170	137	137	135	135	134	136	131
station2	163	138	129	136	136	137	143	135
blue_sky	142	136	141	134	142	135	139	133
pedestrian_area	157	135	134	135	132	135	130	132
Average	**156**	**137**	**137**	**135**	**138**	**136**	**139**	**133**

Table 4.6. Experiment results of average processing cycles per MB in our proposed motion compensation design

	QP=16	QP=24	QP=32	QP=40
tractor	172	159	156	154
sunflower	153	151	157	156
rush_hour	176	152	150	148
station2	169	149	153	156
blue_sky	155	152	153	152
pedestrian_area	166	150	149	146
Average	**165**	**152**	**153**	**152**

For small QP values, the block partition size trends to be small, causing the increment of processing cycles in MVG stage. The INTERP stage is independent with QP since it is only affected by the x- and y-components of fractional MV. A small QP also results in more intra-coded MBs in the bitstream. For example, there are 26%,

39% and 44% intra-coded MBs in *tractor*, *rush_hour* and *pedestrian_area* bitstream (excluding I_Slice) with QP setting to 16 while there are only 8%, 4% and 13% with QP setting to 32. For QP value higher than 20, the average processing cycles are all below 160 cycles/MB. With repetitive background sequences such as *sunflower* with numerous stamens and petals, *station2* with numbers of rails, or *blue_sky* with textureless blue sky and shadow leaves, different QPs may affect the position pointed by MV, causing average processing cycles vibrate in a range. However, for the specific object or clearly demarcated sequences such as *tractor* or *pedestrian_area*, the average processing cycle would be lower in high QP values.

The results here are only considered about inter-coded MBs. If the intra-coded MBs are also taken into consideration, the average processing cycles would be lower.

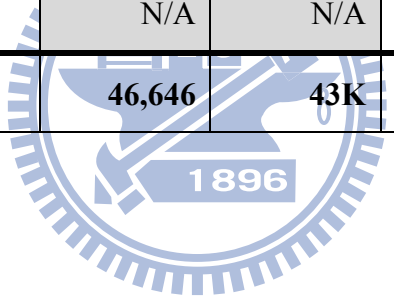
4.4.3. Gate Count

For 135MHz synthesis frequency (clock period is set to 7.4 ns), the total gate count of this motion compensation design excluded internal memory is about 107k. The gate count of each component is listed in Table 4.7.

Compared to previous works [11][14][15], the gate count increment of this work is mainly caused by 21x21-pixel register array in Reference Pixel Accessing part for Partitioned Block Data Reuse (PBDR). The Doubled Hardware of Interpolation Unit in Interpolation (DHIU) scheme also takes much gate space.

Table 4.7. List of gate count for previous works and proposed design

Module	**[11]**	**[14]**	**[15]**	**Proposed**
Motion Vector Generation	N/A	N/A	16,907	14,780
Reference Pixel Accessing	N/A	N/A	44,542	53,754
Interpolation	N/A	N/A	55,728	38,546
- interpolation unit	N/A	20,686	2 * 14,960	2 * 15,571
- luma interpolator (components)	21,506 (FIR*12 Bilinear*4)	N/A (FIR*12 Bilinear*4)	N/A (FIR*12 Bilinear*4)	13,235 (FIR*13 Bilinear*4)
- chroma interpolator	N/A	N/A	N/A	2,336
- pixel reconstruction	N/A	N/A	7,133	7,404
Total	**46,646**	**43K**	**117,177**	**107,080**



4.4.4. Comparison

Since there is no SVC decoder has been published yet, this work is compared with other H.264/AVC designs. Table 4.8 gives the comparison of motion compensation. Compared to [15], the gate count of our design is smaller and the on-chip SRAM is a little larger while the average processing cycle is almost the same. Moreover, the worst cast of interpolation cycles is one-fourth to two-thirds compared to previous works. Generally speaking, the hardware design of this work is competitive with other works.

Table 4.8. Comparison with other motion compensation designs

	[11]	**[14]**	**[15]**	**Proposed**
Technology	0.18 um	0.18 um	0.13 um	0.09 um
Gate Count	46.6 k	43 k	117.2 k	107 k
SRAM (Bytes)	168	-	8 k	8.71 k
# of Interpolation Unit	1	1	2	2
Clock Rate	125 MHz	100 MHz	200 MHz	135 MHz
Standard	H.264/AVC	H.264/AVC	H.264/AVC	SVC
Frame Rate	30 fps	30 fps	30 fps	60 fps
Specification	2048x1024 (P Slice)	1920x1088 (P Slice)	3840x2160 (P, B Slice)	1920x1088 + 720x480 + 352x288 (P, B Slice)
Average Processing Cycles / QP value	-	-	155 / unknown	152-165 / 16-40
Worst Case of Interpolation Cycles	-	560	224	160

Chapter 5. Conclusion and Future Work

5.1. Conclusion

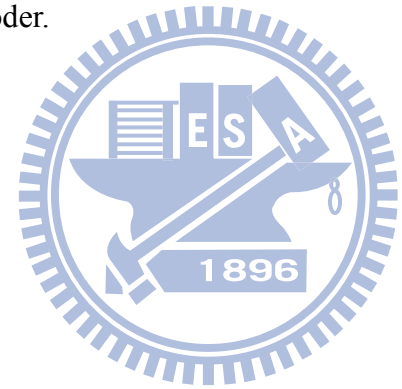
In this thesis, an analysis of SVC decoder and a high performance motion compensation hardware design is proposed. First, by choosing the frame-based decoding flow, the system would enjoy the minimum internal memory usage and bandwidth requirement. Second, our pipeline architecture adopts One-Pass Quality Layer Decoding method, in which the MB processing cycles for in three quality layers bitstream are 66% reduction. Finally, a high performance motion compensation architecture is presented. We decompose the motion compensation into two MB-pipeline stages. The first stage is for motion vector generation and reference pixel accessing, and the second one is for fractional pixel interpolation. The bandwidth requirement is 62-74% reduced by two bandwidth optimization methods. In our MC architecture, a high performance interpolation unit is proposed with only 6 cycles latency to complete a 4x4 block. We further double the hardware of interpolation unit to process L0 and L1 prediction simultaneously, and the latency of bi-pred blocks is halved. In worst case, the latency of our proposed interpolation design will be 160 cycles/MB.

According to the experiment results, the average processing cycles in our proposed motion compensation design are below 160 cycles/MB which is under our system constraints 227 cycles/MB. On the other hand, the proposed hardware is capable for decoding more than 594k macroblocks per second operating at 135MHz

clock rate, which is equivalent to 60 frames of CIF, SD 480p, and HD 1080p resolutions.

5.2. Future Work

The proposed motion compensation hardware design supports Inter Prediction at present. In the future, it should be further integrated with Inter-Layer Motion Prediction as the completely Motion Compensation module for SVC decoder. Moreover, entropy decoding, inverse quantization, inverse transform, intra prediction, deblocking filter, and other SVC hardware components should be merged together to form a complete SVC decoder.



Reference

[1] Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264/ISO/ IEC 14 496-10 AVC," JVT-G050, 2003.

[2] Joint Draft 11 of SVC Amendment, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, Oct. 2007.

[3] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the Scalable Video Coding Extension of the H.264/AVC Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1103-1120, Sep. 2007.

[4] H. Schwarz, D. Marpe, and T. Wiegand, "Hierarchical B Pictures," Joint Video Team, Doc. JVT-P014, Jul. 2005.

[5] H. Schwarz, D. Marpe, and T. Wiegand, "Analysis of Hierarchical B Pictures and MCTF," in *Proceedings of IEEE International Conference on Multimedia & Expo*, pp. 1929-1932, Jul. 2006.

[6] C. A. Segall and G. J. Sullivan, "Spatial Scalability Within the H.264/AVC Scalable Video Coding Extension," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1121-1135, Sep. 2007.

[7] H. Schwarz, D. Marpe, T. Schierl, and T. Wiegand, "Combined Scalability Support for the Scalable Extension of H.264/AVC," in *Proceedings of IEEE International Conference on Multimedia & Expo*, pp. 446-449, Jul. 2005.

[8] P.-Y. Hsu, G.-L. Li and T.-S. Chang, "Memory Analysis for H.264/AVC Scalable Extension Decoder," in *Proceeding of APSIPA Annual Summit and Conference*,

Oct. 2009.

[9] H. Schwarz, T. Hinz, D. Marpe, and T. Wiegand, “Constrained Inter-Layer Prediction for Single-Loop Decoding in Spatial Scalability,” in *Proceedings of IEEE International Conference on Image Processing*, vol. 2, pp. 870-873, Sep. 2005.

[10] T.-W. Chen, Y. Huang, T.-C. Chen and Y. Chen, “Architecture Design of H.264/AVC Decoder with Hybrid Task Pipelining for High Definition Videos,” in *Proceeding of IEEE International Symposium on Circuits and Systems*, pp.2931–2934, May 2005.

[11] C.-Y. Tsai, T.-C. Chen, T.-W. Chen and L.-G. Chen, "Bandwidth Optimized Motion Compensation Hardware Design for H.264/AVC HDTV Decoder," in *Proceedings of IEEE International Midwest Symposium on Circuit and Systems*, vol. 2, pp. 1199–1202, Aug. 2005.

[12] C.-C. Lin, J.-I. Guo, H.-C. Chang, Y.-C. Yang, J.-W. Chen, M.-C. Tsai and J.-S. Wang, “A 160kGate 4.5kB SRAM H.264 Video Decoder for HDTV Applications,” *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pp. 406-407, Feb. 2006.

[13] T.-A. Lin, S.-Z. Wang, T.-M. Liu and C.-Y. Lee, “An H.264/AVC Decoder with 4x4-block Level Pipeline,” in *Proceeding of IEEE International Symposium on Circuits and Systems*, pp.1810-1813, May 2005.

[14] S.-Z. Wang, T.-A. Lin, T.-M. Liu and C.-Y. Lee, "A New Motion Compensation Design for H.264/AVC Decoder," in *Proceeding of IEEE International Symposium on Circuits and Systems*, pp. 4558-4561, May 2005.

[15] P. Chao and Y.-L. Lin, “A Motion Compensation System with a High Efficiency Reference Frame Pre-Fetch Scheme for QFHD H.264/AVC Decoding,” in *Proceedings of IEEE International Conference on Circuits and Systems*, pp. 256-259, May 2008.

[16] H.-C. Tseng, C.-R. Chang and Y.-L. Lin, “A Motion Compensator with Parallel Memory for H.264 Advance Video Coding,” in *Proceedings of the 16th VLSI Design/CAD Symposium*, Aug. 2005.

[17] M. Alle, J. Biswas, and S. K. Nandy, “High Performance VLSI Implementation for H.264 Inter/Intra Prediction,” in *Proceedings of IEEE International Conference on Consumer Electronics*, pp. 1-2, Jan. 2007.

[18] R.-G. Wang, J.-T. Li and C. Huang, “Motion Compensation Memory Access Optimization Strategies for H.264/AVC Decoder,” in *Proceeding of IEEE International Conference Acoustics, Speech, and Signal Processing*, vol. 5, pp. 97-100, Mar. 2005.

[19] Y. Li and Y. He, "Bandwidth Optimized and High Performance Interpolation Architecture in Motion Compensation for H.264/AVC HDTV Decoder," *Journal of Signal Processing Systems*, vol. 52(2), pp. 111-126, Aug. 2008.

[20] H.-B. Yin, D.-P. Zhang, X.-M. Wang, and Z.-L. Xia, "An Efficient MV Prediction VLSI Architecture for H.264 Video Decoder," in *Proceeding of International Conference on Audio, Language and Image Processing*, pp. 423-428, Jul. 2008.

[21] K. Yoo, J.-H. Lee and K. Sohn, “VLSI Architecture Design of Motion Vector Processor for H.264/AVC,” in *Proceeding of International Conference on Image Processing*, pp. 1412-1415, Oct. 2008.

[22] *H.264/AVC Reference Software*, JM 12.4,

http://iphome.hhi.de/suehring/tml/download/old_jm/

[23] *SVC Reference Software,* JSVM 9.14.

Biographical Notes

姓名：許博淵

學歷：

國立交通大學電子所系統組	碩士	(民國 96 年 09 月 ~ 民國 98 年 11 月)
國立交通大學電子工程學系	學士	(民國 92 年 09 月 ~ 民國 96 年 06 月)
臺北市立成功高級中學		(民國 89 年 09 月 ~ 民國 92 年 06 月)

著作：

[1] **Po-Yuan Hsu**, Gwo-Long Li and Tian-Sheuan Chang, "Memory Analysis for H.264/AVC Scalable Extension Decoder," in *Proceeding of APSIPA Annual Summit and Conference*, pp. 299-302, Oct. 2009.

得獎事蹟：

✓ 96 學年度大專院校積體電路設計競賽標準元件數位電路設計組「佳作」