國立臺灣大學電機資訊學院資訊網路與多媒體研究所

碩士論文

Graduate Institute of Networking and Multimedia

College of Electrical Engineering & Computer Science

National Taiwan University

Master Thesis

應用於超高畫質液晶顯示系統之
多畫面插補演算法與硬體架構設計
Algorithm and Architecture Design of Multi-rate Frame Rate Up-conversion in Quad High Definition Liquid Crystal Display System

陳復禎

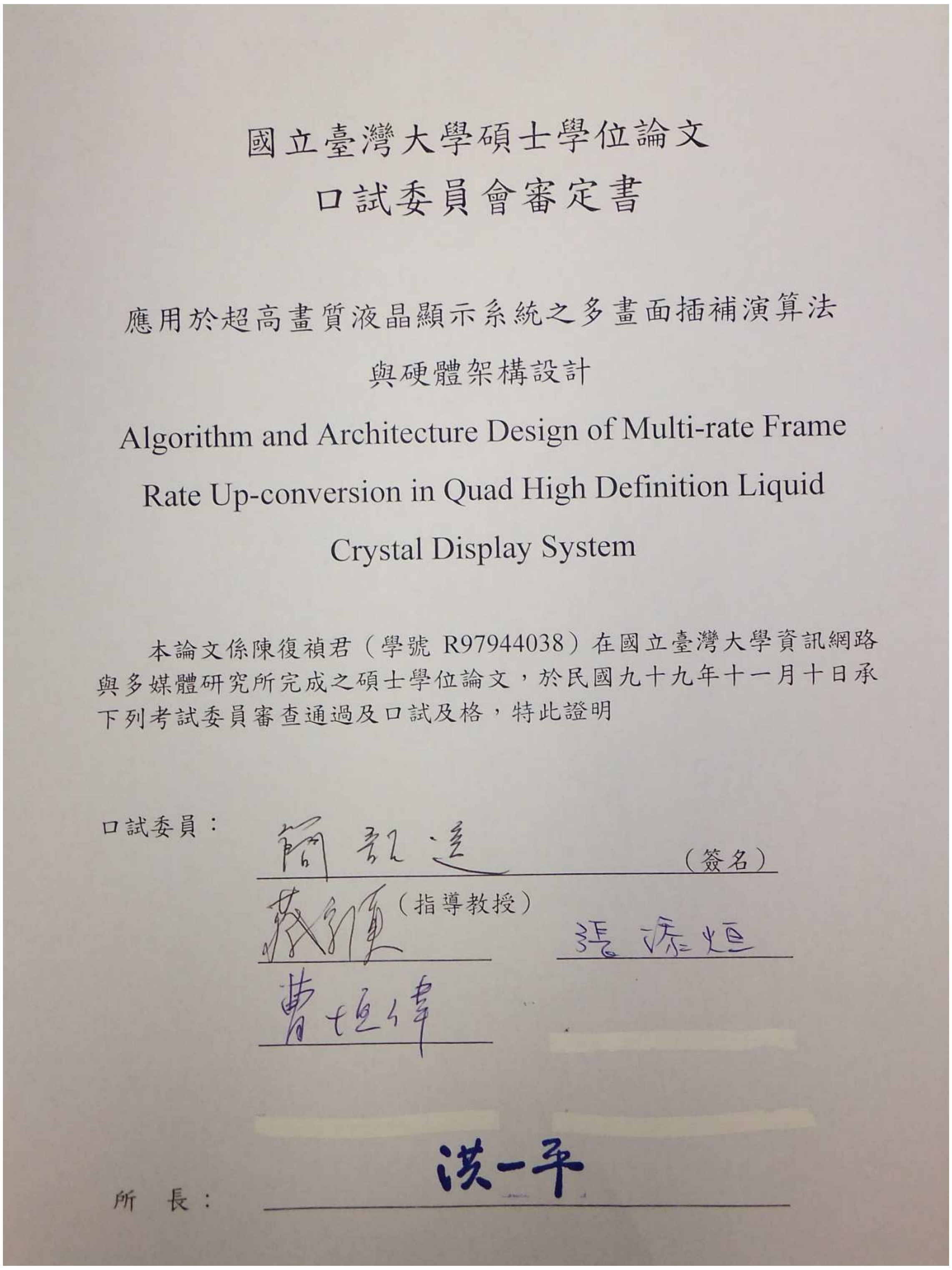
Chen Fu-Chen

指導教授：簡韶逸 博士

Advisor: Chien Shao-Yi, Ph.D.

中華民國 99 年 11 月

November, 2010



國立臺灣大學碩士學位論文
口試委員會審定書

應用於超高畫質液晶顯示系統之多畫面插補演算法
與硬體架構設計

Algorithm and Architecture Design of Multi-rate Frame Rate Up-conversion in Quad High Definition Liquid Crystal Display System

本論文係陳復禎君（學號 R97944038）在國立臺灣大學資訊網路與多媒體研究所完成之碩士學位論文，於民國九十九年十一月十日承下列考試委員審查通過及口試及格，特此證明

口試委員： 簡韶逸 （簽名）

[illegible]（指導教授） 張添烜

曹恆偉

所 長： 洪一平

致謝

這篇論文得以產生，最重要的要感謝指導教授：簡韶逸老師。老師一直以來都很關心我們進度，包括每兩周個人 meeting，每四周投影片報告，還有碩二後的每周進度報告。研究上有問題時，常常老師指點一兩句，回去想一下就通了。記得老師說過一段話：一個真正厲害的人，可以把很難的東西解釋得很簡單，讓台下的人都聽得懂。雖然我不是什麼厲害的人，但我會一直努力實踐這個目標。

再來感謝我的女友，這段日子以來幾乎都是她在照顧我生活起居，平常會煮飯給我吃，以免吃外面太油又太貴，身體不好過敏的時候也帶藥給我吃，甚至還會幫我打掃環境，不管是精神上或實質上都給我很大幫助。口試的時候要是沒她幫忙我甚至連電腦都沒辦法搬過去，更何況是友達的那塊大螢幕，還幫我買便當、拿水果甜點、幫口委們泡茶等等。我答應論文完成後會好好感謝她，但其實我無以回報。

再來感謝我的父母，從小他們對於我念書一直都沒太大干預，念碩班常常會有奇怪的現象比如整個禮拜都沒去學校，但他們也沒說什麼，生活上有問題也義不容辭幫我的忙。我知道他們是信任我，所以我也盡自己的力量完成學業。

接著要感謝學位考試的口委們：曹恆偉教授、蔡宗漢教授以及張添烜教授，謝謝你們對於論文初稿沒有太多苛責，對於論文成果的建議也非常實際與中肯，讓我受益良多，希望我分享的研究內容對你們也有幫助。

實驗室的同學們對我幫助也很大，尤其是博班的學長們。Cary 提供很多演算法上的幫助，小儂對於系統的規格給了很多有用的情報，Larry 則免費提供我很多 Full HD 的影片，讓我實驗和 demo 可以更加順利。冠林對於 encoder 和 ME 的細節幾乎是有問必答，則瑋給了我 SRAM 使用上的觀念，偉凱讓我對 MRF 有更進一步認識，peacer 時常提醒我一個嚴謹的研究該如何作，宥融彥瑜在 meeting 時對我的發問也讓我受益良多。畢業的碩班學長中，功炎的論文讓我對 FRUC 的硬體有很大的認識，去友達報告了一年大概就知道硬體實作方式，Inverse-MC 的想法也是源自他的 GME 硬體，nana 陪我搞那個友達螢幕讓我們有了革命情感，阿戎給

了我很多研替的資訊和情報，家恆豪弟常常和我一起玩樂，shimii 提供了我很多 perceptual 的意見，paulman 解決了我某些 layout 上的疑惑。同居的夥伴們更不用說，大家一起同甘共苦了兩年，知道了什麼祕訣一定會馬上和其他人講，完全都不會藏私，坐彥彰春益隔壁的好處是可以聽他們鬥嘴還有罵 encounter，和潔立怡婆一起修課的好處就是她們一定會好好上課，有問題可以直接問她們，省時又省力哈哈，登元常常都約大家吃飯出遊，而且看他研究作的量可以拿來督促自己要多加把勁，小暴坐在隔壁除了髒了點還有每次都要幫他買飯之外，可以看他打電動輸了然後笑他作為消遣，忠爸則是提供了一個成家男子該有的形象。學弟妹們雖然主要的功用是幫我們買烏鐵，但你們也為 421 帶來不少的活力。嗆翔皴是我的樂趣之一，看昱絜罵遊戲裡的隊友也非常好笑，熊嗆博班學長的方式也是十分經典，一圈和孟璘則扮演著冷眼旁觀者看他們搞笑。口試的時候除了可惡的熊之外你們也全來幫我助陣，真是太感謝了。新進來的學弟妹們，雖然和你們還不太熟，可是和你們在明達館和澎湖的相處覺得你們都很不錯，搞笑功力也是一流（尤其是吳柏辰！），恭喜你們進到很好的實驗室，動作要快一點不要像我這樣。

和我同居的同學朋友，我也很感謝你們。每次聚會打嘴泡都是我最快樂的時候，讓人有好像還是大學生的錯覺，我相信不管過了多久，每次聚在一起時講的話都還是會一樣的好笑。特別感謝 LPG 和左哥修 CVSD 時給了我許多想法，和 Larry 文甫潔立合作的 VFX 期末廣受大家好評，阿泡和張廚餘修課常一起討論，Lily 則是不辭辛勞幫我處理 umc 製程的問題。PG 大大常在危急存亡之時從中路幫我助拳，碰狗在國家最需要他的時候挺身而出。

網媒所的系辦陳雅琳小姐算是我見過最負責的行政人員，碩班大大小小的事務她都處理得無微不至，很多問題找她都一下就解決，真的很感謝她。

還要感謝研替錄取我的公司，因為他們在碩二下就提供我每個月的獎學金，讓我可以過著衣食無缺甚至到奢華的碩士生活，不必擔心錢的問題而可以專心的作研究。

最後感謝我自己這段日子可以撐下去，希望未來不管是工作或甚至回到學校作研究，我還是能維持同等的熱情度，辛苦了，謝謝你。

要謝的人太多了，真的就，謝天吧！

中文摘要

畫面插補演算法，是經由分析輸入的視訊，插補出額外的畫面，藉以提升視訊的顯示頻率。而多畫面插補演算法是在兩張已有的畫面之間，插補出兩張以上的新畫面。早期這項技術是在視訊壓縮中被討論，而近幾年來則是被應用在液晶螢幕顯示器，將畫面顯示頻率提升到 120 赫茲或甚至更高，以解決液晶螢幕的動態模糊問題。

此項技術大致可分四個階段，第一階段先找出相鄰兩張畫面的移動向量，第二階段將移動向量作分析與優化,第三階段根據新的移動向量插補出額外的畫面，第四階段則根據已補好的畫面修正補不好的區域。由於液晶螢幕畫面的解析度越來越高，此項技術最主要的挑戰是運算量、頻寬以及記憶體的大量需求，因此成本相較於其他視訊處理的晶片是更為昂貴的。

我們發展出一套多畫面插補演算法與硬體架構，能夠符合現行液晶螢幕顯示系統的基準。演算法先以預測方形搜尋法，利用移動向量空間相依的特性，快速的找出畫面大致的移動向量。接著我們以馬可夫隨機場域為基礎，對已有的移動向量作修正，以極低的運算量找出畫面中真正的移動向量。將原本的移動向量轉移到中間的畫面主要有三種方法，但每種都不是完美的。我們綜合各種方法的優點，以方格穿越式移動補償來出補出中間的畫面。接著我們提出一個簡單有效的方法，保證找出補不好的區域。對於這些區域，我們將他的方格切小，用雙向的方式幫他們找到缺失最少的移動向量，再以疊加式方格移動補償將此區域畫面給補得更好。

硬體架構上，由於支援的向量範圍為負一百二十八到正一百二十八，因此我們以特殊的記憶體存取安排，使記憶體使用量小非常多。為了去除各個步驟相依性造成的排程空缺，我們提出乒乓雙向的特殊排程將空缺給填滿。對於差異計算的部分，我們提出以可變性加法器構成的加法樹架構，僅僅只需要八十五個加法器。整個系統的記憶體和和加法樹是被所有的分部給共用的。對於馬可夫隨機場

域向量修正，我們發展出一套向量群聚的演算法與硬體架構，使相似向量的所需的資料可以共用，並且利用已計算過的差異值讓運算量可以更進一步減少，省下大量的頻寬和運算時間。多畫面插補的部分，我們提出反向插補排程技巧，使得這部分的頻寬和運算能達到最小值，而硬體架構也可以和方格穿越式相量轉移來共用。最後對於小區域的修正，我們以模擬進行平行度分析，使運算速度達到要求且不需要額外的硬體。為了要充分利用頻寬，我們也用了特殊的記憶體排列，使得資料可以任意的被存取。

實驗分析上，我們挑選三篇文獻作演算法比較，主觀比較是由受測者從不同演算法插補的畫面中，挑選他認為最好的；客觀比較則是將原始視訊的顯示頻率減半，用不同演算法插補出新的視訊，再和原始視訊比較信噪比。結果顯示我們的演算法無論在主觀比較和客觀比較上，都優於其他三種論文的演算法。硬體實作以 Verilog 硬體描述語言實現，用 UMC 90nm 製程元件庫以及 SYNOPSIS Design Compiler 來合成。所得之全部閘數為 274K，記憶體使用量為單端口 9984 bytes。其運算頻率為 300MHz，能提供 24 赫茲轉 120 赫茲及 60 赫茲轉 120 赫茲的多畫面插補，並支援到下一代液晶螢幕的 3840x2160 解析度。硬體使用效率部分，比較其它文獻的硬體實作，我們的使用效率也是最佳的。

Abstract

Frame rate up-conversion is a technique that up-converts the frame rate of video sequence by analyzing it and interpolating additional frames. Multi-rate frame rate up-conversion interpolates two or more frames between two existing frames. This technique is previously discussed for video compression and applied on LCD to convert frame rate up to 120Hz or even higher for eliminating the LCD motion blur problem in recent years

This technique roughly consist of four steps, the first step finds the motion vectors between two successive frames, the second step then analyzes and optimizes the motion vectors, the third step interpolates additional frames according to new motion vectors, and the forth step corrects the region with artifact on interpolated frames. As the resolution of LCD getting higher and higher, the main challenges of this technique are the huge demands of computation, bandwidth and on-chip SRAM. Therefore the cost of it is more expensive than other video processing DSPs.

We develop a multi-rate frame rate up-conversion algorithm and architecture that is compatible with current LCD system's standard. The algorithm first performs predictive square search motion estimation which utilizes the spatial coherence of motion vector field. This motion estimation algorithm can roughly find the true motion of existing frames quickly. Then we apply motion vector processing based on Marcov random field with a very low-cost minimization method to find the true motion of existing frames. There are three general methods for mapping motion vector to inter-frames but none of them is perfect. We employ the advantages of each method, proposed a block-based through motion compensation for interpolating inter-frames. And then we bring up a simple and precise technique that guarantees to detect the region with artifact. For the region, we perform sub-block division, find new motion vector with the least artifact in bilateral directions and interpolate it by overlapped block motion compensation for better visual quality.

As regards to hardware architecture, since the supporting search range of motion vector is ±128x±128, we provide a special SRAM arrangement that reduces a huge amount of SRAM size. For eliminating the dependencies of each step causing pipeline bubbles, we propose ping-pong two-way scheduling to fill-up the bubbles. For distortion computation, we devise sub-trees composed of 85 flexible adders. The sum-trees and SRAM are shared by all modules of architecture. For Marcov random field motion vector correction, we develop a motion vector grouping algorithm and architecture for data reuse of similar motion vectors. We also employ the computed distortion result to further reduce the computation, saving lots of cycles and bandwidth. For multi-frame interpolation, an inverse motion compensation scheduling is proposed that reaches the minimum requirement of computation and bandwidth. The architecture here is also shared by block-based through motion vector mapping. For post-processing on the sub-blocks, we analyze the parallelism by simulations and reach the demand of speed without addition area overhead. For exhaustively utilizing the bandwidth, we bring up special SRAM interleaves such that the data can be read or written in an easy manner.

As regards the experiments, we select three literatures for algorithm comparison. The subjective evaluation is that the subjects choose the best one from all frames interpolated by different algorithms. The objective evaluation is that halves the frame rate of original video sequence, interpolates new sequences by different algorithms and generates PSNR to the original video sequence. The results indicate that our algorithm is better than other three algorithms on both subjective and objective evaluation. We use Verilog-HDL for hardware implementation and synthesize it by SYNOPSIS Design Compiler with UMC 90nm cell library. The implementation shows that the total number of gate count is 274K and on-chip SRAM is single-port 9984 byte. It works at 300MHz frequency, providing 24Hz to 120Hz and 60Hz to 120Hz multi-rate up-conversion and supporting 3840x2160 resolution for next LCD generation. For the hardware efficiency, our architecture is also the best comparing to other previous implementations.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Introduction to Frame Rate Up-conversion

Frame rate up-conversion (FRUC) is a technique that up-converts the frame rate of input video sequence. Like Figure 1.1, it is mainly composed of two parts: finding the motion vectors presenting objects' movement and interpolating frames according to motion vectors between existing frames of input video sequence. The first part is called motion estimation (ME), and the second part is called motion compensation (MC).

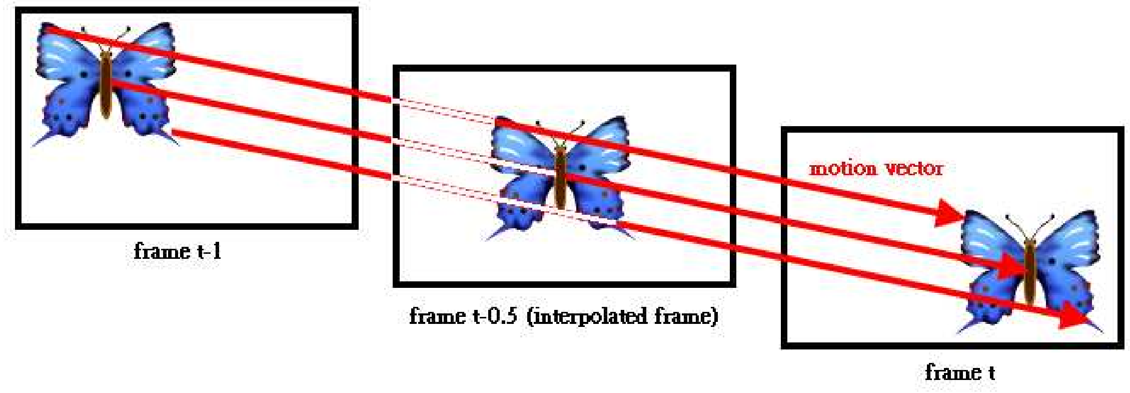


Figure 1.1 The concept of FRUC.

FRUC is discussed previously for video compression. Imaging if we can drop a half of frames at the encoder side and reconstruct them at the decoder side by FRUC, then the size of compressed video can also decrease almost by a half [1]. In this case, the interpolated frames originally exist, so it is easy for us to know how do them being interpolated.

For recent years, it is applied on liquid crystal display (LCD) for converting frame rate of input video stream to 120Hz or even higher. The purpose is to reduce the hold-type motion blur on LCD which will be introduced in next section [2]. In this case the converted frame rate is higher than the original and the interpolated frames do not exist before, so it is hard to know whether the interpolated frames looks good or bad.

1.2 Motion Blur on LCD

Figure 1.2 shows the general structure of LCD. The backlight walks through the first polarizer, causing the orientation of light to become uniform. Then it passes the liquid crystal layer controlled by the voltage between two slices of thin film transistor (TFT). The voltage value influences the rotating angle of light passing the liquid crystal layer. In the end the light walks through the color filter (red, green and blue) and second polarizer with different angles. Since the second polarizer will filter the light with different orientation to it, controlling the voltage means adjusting the intensity of each color displayed.

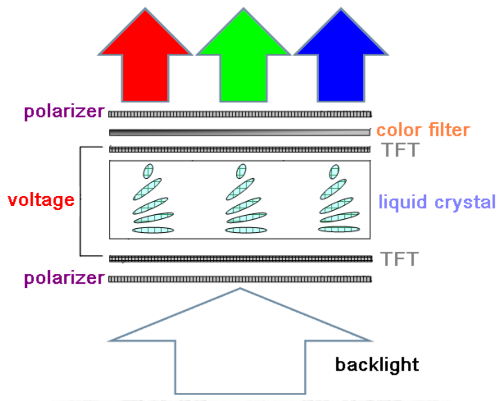


Figure 1.2 General LCD structure.

There are two types of motion blur occur on LCD with different solutions [3]. The first type of motion blur is caused by the slow response of liquid crystal. As in Figure 1.3, the black solid line is the targeting brightness and the dotted line is the actually displayed brightness. The smooth variation of brightness looks blurred by human eyes. To overcome this problem, first set the voltage higher (or lower) than the targeting brightness, and after the brightness becomes close to the target, set the voltage to the ordinary value. By doing this, the slope of brightness will be sharper like red solid light in Figure 1.3, which reduce the smooth variation of brightness.

The second type is called hold-type motion blur. As shown in Figure 1.3, the maintenance of brightness is called the period of hold which is equal to the inverse of frame rate. In Figure 1.4, when human eyes track objects along their movement with velocity ***v***, they integrate the intensity continuously, but the real intensity changes discretely. This divergence makes the integrated signal of the object's boundary on retina smoothly decrease (increase). The range of decreasing (increasing) is called blur width and can be directly expressed as

$$Blur\ width_{direct} = \boldsymbol{v}/frame\ rate \tag{1.1}$$

Another way for evaluating hold-type motion blur is based on the sampling and reconstruction theory of integrated signal on human's retina [4]. In the case of idle display (without slow response), the blur width is equal to

$$Blur\ width_{theory} = 0.8 \times \boldsymbol{v}/frame\ rate \tag{1.2}$$

Therefore, the blur width is inverse- proportional to the frame rate. Among the solutions of hold-type motion blur, FRUC is regarded as the best method since it can directly reduce the effect of motion blur without visual quality drop [3].

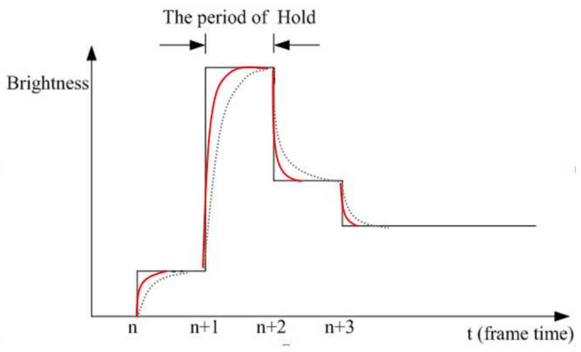


Figure 1.3 Hold-type display with slow response.

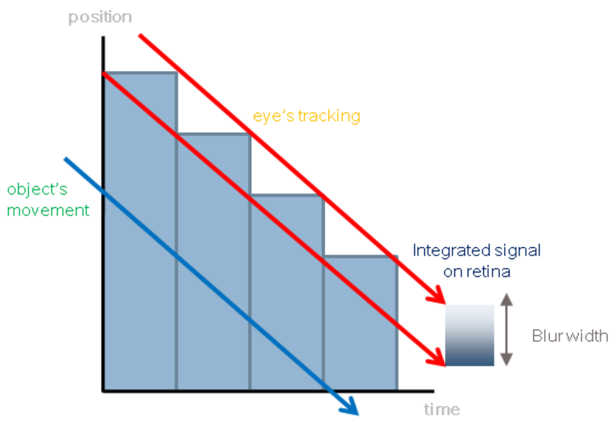


Figure 1.4 Direct evaluation of blur width.

The first real-time DSP appeared in ICCE1995 announced by Philips as a commercial product for motion blur reduction on LCD [5]. Figure 1.5 is its chip photo. In recent years, the FRUC topic is frequently discussed, but don't have a generally accepted conclusion. There is also a lack of academic literature or announcement of FRUC DSP since the cost is so high that academic institution cannot burden.

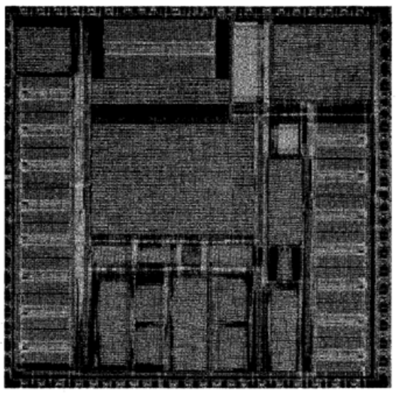


Figure 1.5 Chip photo of the first FRUC DSP.

1.3 Design Motivation and Target

As for hold-type motion blur reduction on LCD, we want to design a FRUC algorithm and architecture which fits with the current LCD system (Figure 1.6). The capability of the design has 24Hz to 120Hz and 60Hz to 120Hz multi-rate up-conversion, and supports 3840x2160 (Quad HD, 4Kx2K) resolution for next LCD generation. The computational cost and area cost must be low with the reasonable bandwidth consumption.

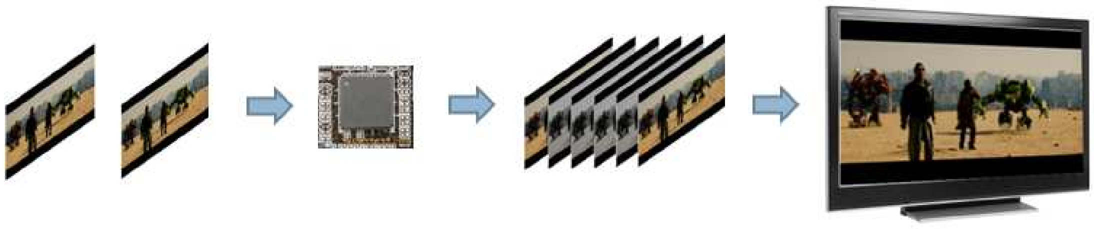


Figure 1.6 Our design motivation and target.

For the various types of videos, we choose many sequences for experiment including sport game lives and movies. Due to the deficiency of 4Kx2K sequences, we use 1920x1080 (Full HD, 1080p) sequences instead. Figure 1.7 shows some of the 1080p sequences.

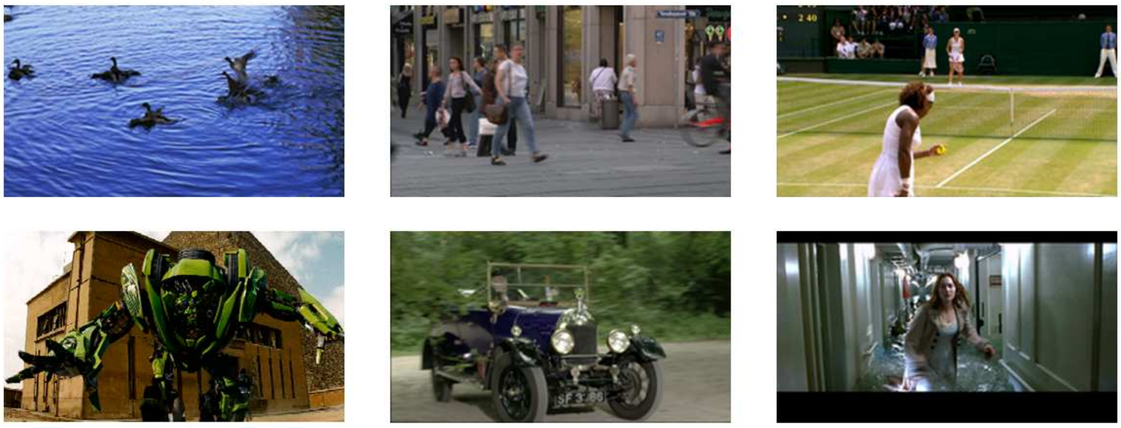


Figure 1.7 1080p sequences for experiment, from top-left with raster order: ducks_take_off, pedestrian_area, Wimbledon open 2010, transformer2, vintagecar, titanic.

1.4 Thesis Organization

In this thesis, the detailed steps of up-conversion algorithm is introduced and analyzed in the first section of next chapter. In next section we introduce proposed FRUC algorithm based on Marcov random field with precise artifact detection. In each step, we will tell how to operate and the reasons of proposed algorithm. The third section we compare the proposed algorithm with three different FRUC methods and show the results of subjective and objective evaluation. The final section we give a short conclusion of the proposed algorithm design. Chapter 3 is about architecture of FRUC on LCD. The first section analyzes the specification of our target, and the second section shows the overview of our architecture. Section 3 to section 6 describes the problems encountered of each part and explains how we solve these problems by proposed architecture. Section 7 shows the implementation results of resource consumption and hardware specification. In the last chapter we give an overall conclusion of proposed design, and the possible future works.

Chapter 2 Marcov Random Field Based Algorithm with Precise Artifact Detection

2.1 Introduction to Up-conversion Algorithm

Figure 2.1 shows a general FRUC flow. First it gets the motion vector field (MVF) of existing frames from motion estimation or decoder. Then it operate on the MVF to let it be more reliable, called MV processing. The blocks may be divided into sub-blocks for presenting more detail motion and reducing block artifact (sub-block division). After getting new MVF, MV mapping procedure maps MVF of existing frames to inter-frames. Motion compensation interpolates inter-frames according mapped MVF, and may apply bilinear, filtering or overlap block motion compensation (OBMC) [6] for interpolation. After that, post-processing procedure employs initially interpolated inter-frames' information to refine the artifact and make the inter-frames have better visual quality.

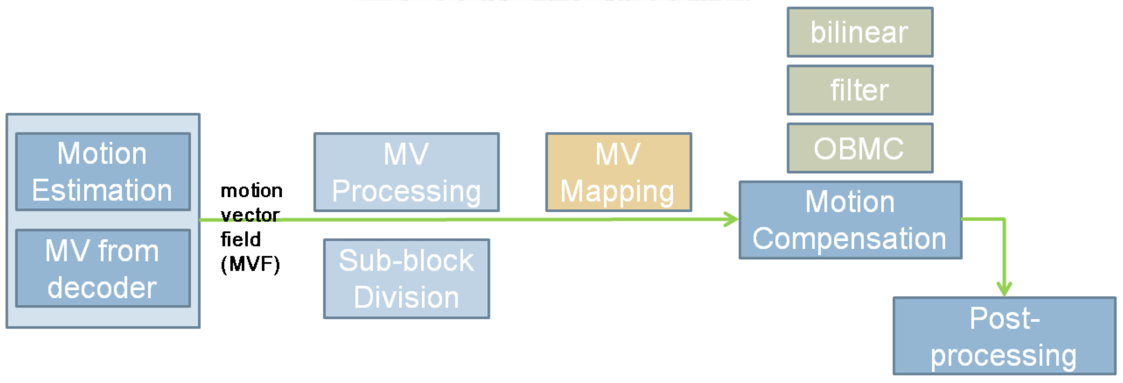


Figure 2.1 General FRUC flow.

2.1.1 Motion Estimation

Unlike conventional motion estimation for encoder, the purpose of it in FRUC is finding true motion presenting objects' movement [7], not just to reduce the residual energy of each block comparison. It is often a time and area consuming part of the design since the number of required candidates of block matching is usually big. Recently researches suggest that decreasing the number of candidates is helpful to find true motion [7], [8], [9]. These algorithms often utilize the spatial and temporal coherence of MVF. At first it is hard to estimation the performance of each algorithm since no ground truth for comparison. By 2007, Microsoft research team provides a ground truth database generated by synthesis or capturing motion with photosensitive pigment [10].

It is possible getting MVF from decoder without motion estimation process. Some of FRUC algorithms use this trick and focus on following operations [11], [12], [13]. We think there are two problems of this trick. The first is that we can't not sure which ME algorithm is performed by encoder, and even the condition of intra-block. If the received MVF is untidy or the number of intra-blocks is too much, the true motion is hard to find by the following operations. The second is that in current LCD system, it is unable for DSP to get MVF information from decoder. By the reasons, we determine to perform ME, not getting MVF from decoder.

2.1.2 MV Processing

The purpose of this operation is to make MVF more reliable for presenting objects' movement. Since the true motion has spatial and temporal coherence, some simple operations such as median filter [14] or weighted averaging have just acceptable effect. Dane, G. and Nguyen, T.Q. [15] provide a motion smooth method by global energy minimization with a matrix closed form solution. Demin Wang et al. [16] model the MVF as a 3D Marcov random field and minimize the energy by iterated condition mode. Many of the algorithms here are heuristic, without a theoretical principle. Some of algorithms are too complex for hardware implementation.

2.1.3 Sub-block Division

Many FRUC algorithms divide block into sub-blocks to reduce block artifact and get more detailed MVF after motion estimation [13], [17]. The problem is whether divide all blocks or only at sub-region. Divide all blocks will make sure all motion vectors are finer, but with the highest complexity. If only divide sub-region, the efficiency is an important concern since only the region with artifact have to be divided. After division, there should be another process that determines the new motion vector of sub-blocks. The way of determination directly influences the motion compensation results.

2.1.4 MV Mapping

In general, there are three mapping methods and none is perfect. The first one is called tradition mapping, which performs ME on existing frame and copy the motion vectors to the corresponding blocks of inter-frames. The second one is called through mapping, performing ME on existing frames too but MC through exist MV's direction. The third is called bilateral mapping, which performs ME on inter-frame with two reversely motion vectors [18], [19]. Figure 2.2 is the graphic illustration of the three methods.

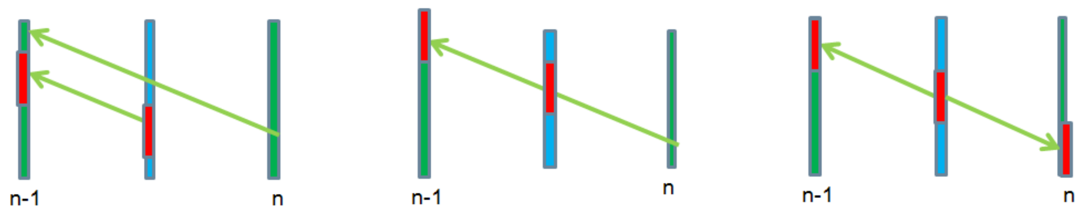


Figure 2.2 Three general mapping methods: tradition, through and bilateral.

The problem of tradition mapping is time domain mismatch. The corresponding blocks of inter-frames and existing frames may belong to different object since their timing is not the same. The motion vectors of these blocks are not totally equal so we can not directly copy. The through mapping does not have this problem, but the interpolated frames usually have hold and overlap because the motion vectors passing through them may not be aligned as shown in Figure 2.3. The hole and overlap require other technique to handle like in-painting [20], which is often so complex. Non-block interpolation is also a nightmare for hardware implementation. The bilateral mapping seems to be the best one without time domain, hold and overlap problem, but by many researches, it normally fails at flat region and need more special concerns [21]. For bilateral mapping, the number of ME times is also equal to the number of inter-frames, which is too costly for multi-rate up-conversion.

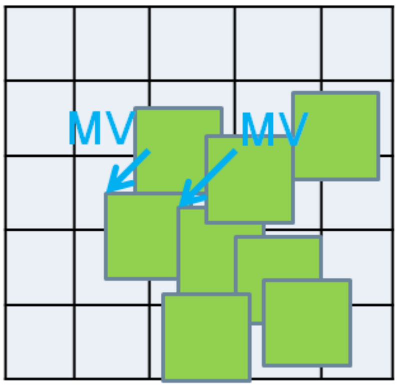


Figure 2.3 Hole and overlap of through mapping.

2.1.5 Motion Compensation

After getting the MVF of inter-frames, motion compensation interpolates the inter-frames according to MVF. Since the motion compensation is block-based, there are many ways for block artifact reduction like bilinear, filtering or OBMC. A zero motion preserving technique called graceful degradation [22] is devised for text protection in videos. Occlusion handling is proposed in [23] by applying adaptive weighted-interpolation for pixels only appear in one of the existing frames. These techniques often applied to whole frame with computation overhead, and may make inter-frames more blurred than directly interpolation.

The hardware consideration here is seldom discussed since for a regular decoder, it only compensates one frame each time. For multi-rate up-conversion, up to four or more frames have to be generated so the bandwidth consumption is very huge (Ex. 120Hz Quad HD = 1.44GByte / sec). A well-designed architecture is very crucial for completely utilizing the bandwidth available.

2.1.6 Post-processing

After compensating all inter-frames, this step employs the interpolated frames' information for finding new motion vectors and re-interpolating on particular region [17], [23]. There are various ways of this step and we conclude them into three parts: artifact region detection, motion vector refinement and artifact-reducing interpolation.

The first part is tough because there are no references of inter-frames for comparative artifact detection. The second and third part must be based on artifact reduction for better visual quality of re-interpolated frames.

2.1.7 Summary

In this section, the steps of FRUC are introduced. Motion estimation is a computation and area costly part, and its target is finding true motion of objects' movement. Motion vector from decoder is not reliable and not the current LCD's FRUC standard. There are many methods for MV processing, from simple to complex, either heuristic or theoretical. Dividing all block requires more computation overhead, and dividing sub-region must be careful of the efficiency. MV mapping has three general types, but none of them is perfect. Motion compensation is a bandwidth-consuming part for multi-rate up-conversion seldom discussed in hardware. Some technique may applied here for block artifact reduction or occlusion handling. Post-processing utilize the interpolated frames, and take care of region with artifact for better visual quality.

2.2 Proposed Algorithm Based on Marcov Random Field with Precise Artifact Detection

We describe the proposed algorithm in detail with reasons. The video size in this section is regarded as 1080p, not 4Kx2K. For motion estimation, we provide a true motion based search algorithm whose computational complexity is very low. The block size starts from 32x32, and the matching criterion is 8x8 MSEA for hardware consideration. We perform MV processing based on Markov random field modeling with a low cost but robust version of iterated conditional mode (ICM) minimization. We propose a MV mapping technique that determines inter-frames' MVF by block-based through mapping. After motion compensation, we divide sub-blocks only on necessary region by precise artifact detection. In the end for those sub-blocks with artifact, we search new motion vectors for them and re-interpolate with occlusion consideration. The experiments show that the proposed algorithm is better than the others by both subjective and objective evaluation.

2.2.1 Predictive Square Search Motion Estimation

This motion estimation algorithm is very similar to predictive diamond search, except we use square pattern (Figure 2.4) instead of diamond pattern. Figure 2.5 shows the graphic illustration of predictive square search algorithm. First we perform median filtering on three neighboring motion vectors and getting a predictor. Then we apply 4-step square pattern on the predictor. If the minimum distortion appears at center or its value is smaller than the threshold, we think the predictor is good and proceed to apply 2-step and 1-step square pattern for converge. Else we back to the origin and search motion vector like normal diamond with 8-step square pattern. If the minimum distortion is at the center of 8-step square pattern, then we apply 4-step, 2-step and 1-step for converge.

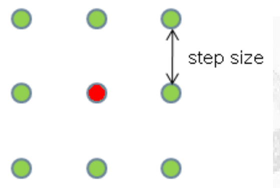


Figure 2.4 Square pattern with 9 candidates.

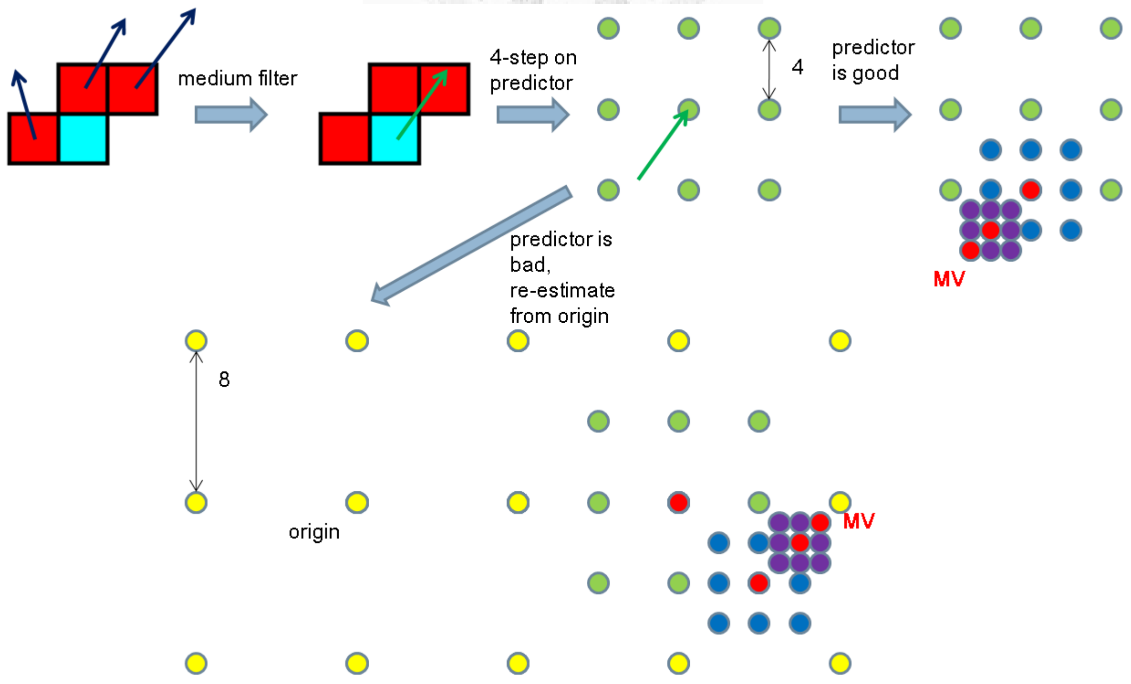


Figure 2.5 Graphic illustration of predictive square search algorithm.

In the following we show the pseudo code of proposed search algorithm. Here ***SP*** means square pattern and $\boldsymbol{\varepsilon}$ means minimum distortion of applied ***SP***. ***MV*** means motion vector value as the center of ***SP*** and ***threshold*** here is equal to 1024 in our implementation.

```
1.  Set MV = median of three neighboring blocks' motion vectors
2.  Apply 4-step SP on MV
    If ε is at center or ε < threshold
        Apply 2-step & 1-step SP for converge
    Else
        Set MV = origin, go to step 3
3.  Apply 8-step SP on MV
    If ε is not at center
        Set MV = ε's position, repeat step 3
    Else
        Apply 4-step, 2-step & 1-step SP for converge
```

The proposed algorithm is very similar to PMVFAST [24] which is frequently used for true motion estimation [25] exploiting the spatial coherence of MVF. We abandon temporal coherence since blocks of the same coordinate at different timing may not belong to the same object, so the temporal prediction is not certainly accurate. Another reason for proposing this algorithm is the ability of rejecting predictor and re-estimating from origin. The algorithm is also very cost efficient. Figure 2.6 shows the percentage of blocks' converging type in the worst cases of each sequence. For the most complex sequence (vintagecar), there are near 60% blocks converge around predictor.

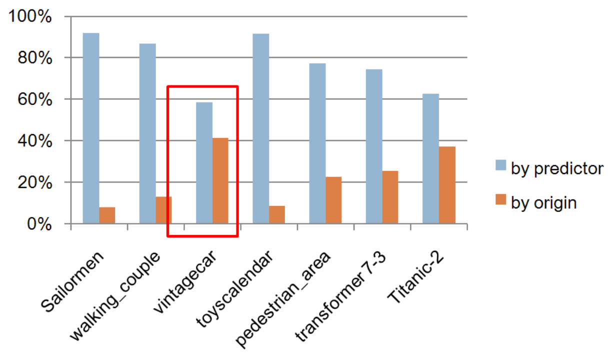


Figure 2.6 Percentage of blocks' converge type (the worst cases of each sequence).

We set block size to be 32x32 in our design (for 1080p) for two reasons. Like overlapped block motion estimation (OBME [11]), the first reason is aperture problem since the bigger size for block-matching, the more reliable motion vector generated. The second is instead of merging smaller block for motion vector unity, dividing the blocks on necessary region spends less computation overhead [1].

For matching criterion, we choose 8x8 MSEA [26] for block-matching. As shown in Figure 2.7, it divides 32x32 blocks into 16 8x8 sub-blocks, sums up each sub-block, then compute 16 absolute differences (Abs.) of each summed up sub-block pair. The 8x8 MSEA is equal to sum of 16 Abs. values as

$$8x8\ MSEA = \sum_{\forall pair} Abs.\ of\ sub-block\ sum\ pair \tag{2.1}$$

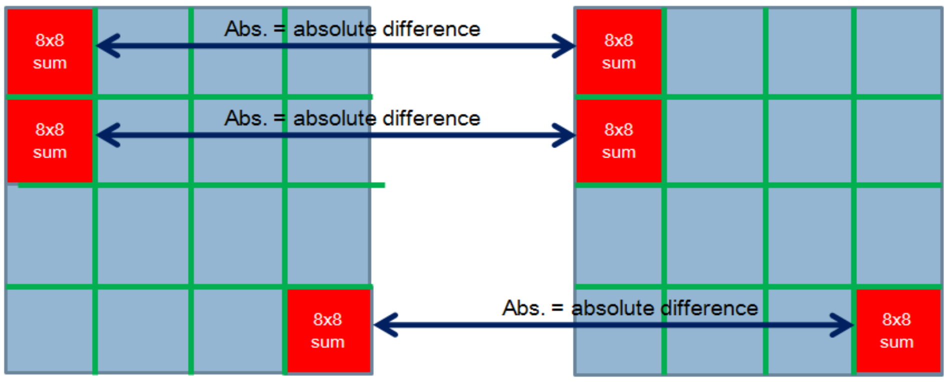

8x8
sum
Abs. = absolute difference
8x8
sum
8x8
sum
Abs. = absolute difference
8x8
sum
8x8
sum
Abs. = absolute difference
8x8
sum

Figure 2.7 Illustration of 8x8 MSEA.

8x8 MSEA can be regarded as the down-sample version of sum of absolute difference (SAD), without down-sampled motion vector value. The original purpose of this criterion is fast full search, and we found that to co-operate with square pattern it reduces lots of computation and bandwidth cost in hardware design. The generated MVF is almost the same if we use SAD as the matching criterion.

By experiments, we notice that there are 24Hz sequences whose maximum motion vector value reaches 128, so we determine the search range of motion estimation to be ±128x±128 for hardware design.

2.2.2 Markov Random Field Motion Vector Correction

In this step, we process in raster order to refine the new motion vectors for each block for three iterations. For a block, we select eight neighboring motion vectors and the motion vector of itself as nine new motion vector candidates (Figure 2.8). For these nine candidates, we compute the corresponding Marcov random field energy (MRF energy)

$$MRF\ energy_{candi.} = 8x8\ MSEA + weight \times \sum_{\forall neigh.} \left| MV_{candi.} - MV_{neigh.} \right| \qquad (2.2)$$

From nine these candidates, we select the smallest one as the new motion vector of this block.

$$new\ MV = \underset{MV_{candi.}}{\boldsymbol{arg\,min.}}\ MRF\ energy_{candi.} \qquad (2.3)$$

This process is called iterated conditional mode (ICM) minimization and the weight is equal to 48 in our design. Unlike the general ICM that selects all candidates in search range, we only choose nine candidates adjacent to the block. Figure 2.9 shows the visualization of MVF after ME and MV processing. The color of visualization presents the direction of motion vectors, and the intensity presents the magnitude of motion vectors. After ME, the MVF roughly forms the shape of objects in the frame such as two walking people and the trunk of trees but with some motion vector outliers. After iteration 1, those outliers are corrected and the MVF looks closer to objects' movement. The MVF of iteration 2 and iteration 3 looks almost the same, but they do remove more outliers than previous iterations.

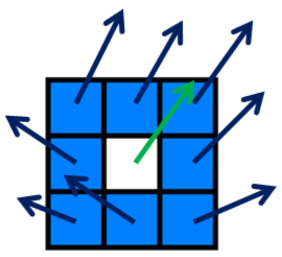


Figure 2.8 Nine candidates of new motion vector.

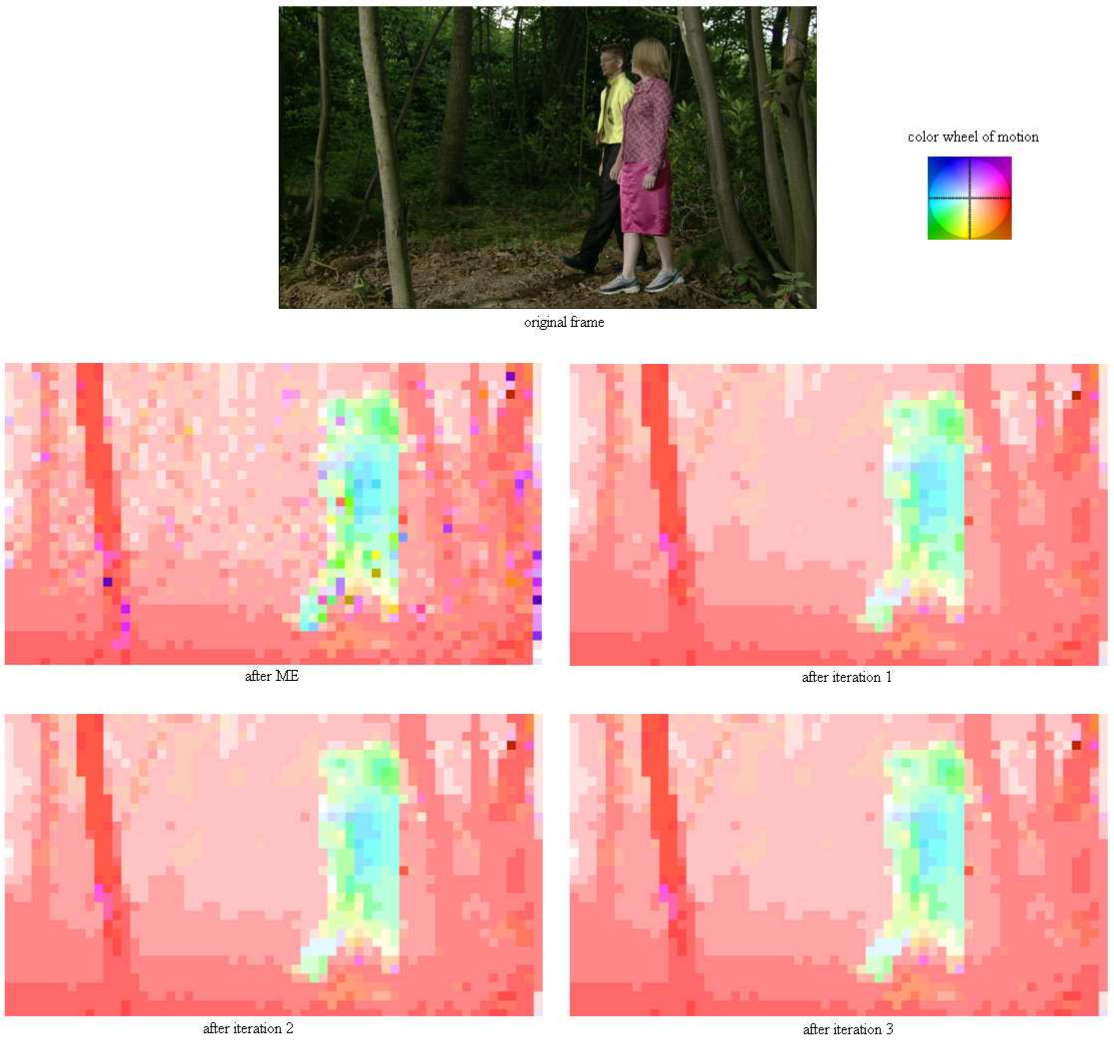


Figure 2.9 Visualization of MVF after ME and MV processing.

Marcov random field (MRF) is a very theoretical modeling method based on Bayesian's framework, applied to computer vision for many years [27] such as optical flow or true motion estimation [16], [28]. The global minimization is a NP-complete problem [29] so many fast algorithms for finding local minimum are proposed [30]. For a well-known method called belief propagation, the related hardware design requires 633K gate count and 1.88M byte ob-chip SRAM, which is too expensive for us. Thus we choose the ICM for minimization. By researches, there is a very high probability for a block to find its true motion from nearby block's motion vectors [31], thus choosing neighboring nine candidates is already enough. Another benefit of choosing neighboring nine candidates is preventing over-smoothing, and the complexity is also lower than selecting all candidates in search range.

2.2.3 Block-based Through Motion Compensation

Though mapping has no timing problem, but with hole and overlap. Here we divide inter-frames into inter blocks (size is also equal to 32x32), and project the motion vectors to these inter blocks. Each projection of motion vectors may has overlap area with inter blocks as illustrated in Figure 2.10. In [23], it assigns inter block's motion vector to be the weighted sum of motion vectors projected to it, and the weight is equal to overlap area. In this way, the MFV of inter-frames may be over-smoothing by weighted sum operation. For preventing over-smoothing, we accumulate the total overlap area of each motion vector projected to it and find the motion vector with the maximum overlap area. If the maximum overlap area is bigger than a half of block size (512 pixels), we set the motion vector of this inter block to be the motion vector with the maximum overlap area. If not, we set the motion vector of inter block to be the motion vector of co-located block on existing frames. After determining all inter-blocks' motion vectors, we perform block-based motion compensation to interpolate inter-frames. By doing so, we prevent the timing mismatch of tradition mapping and still perform block-based motion compensation.

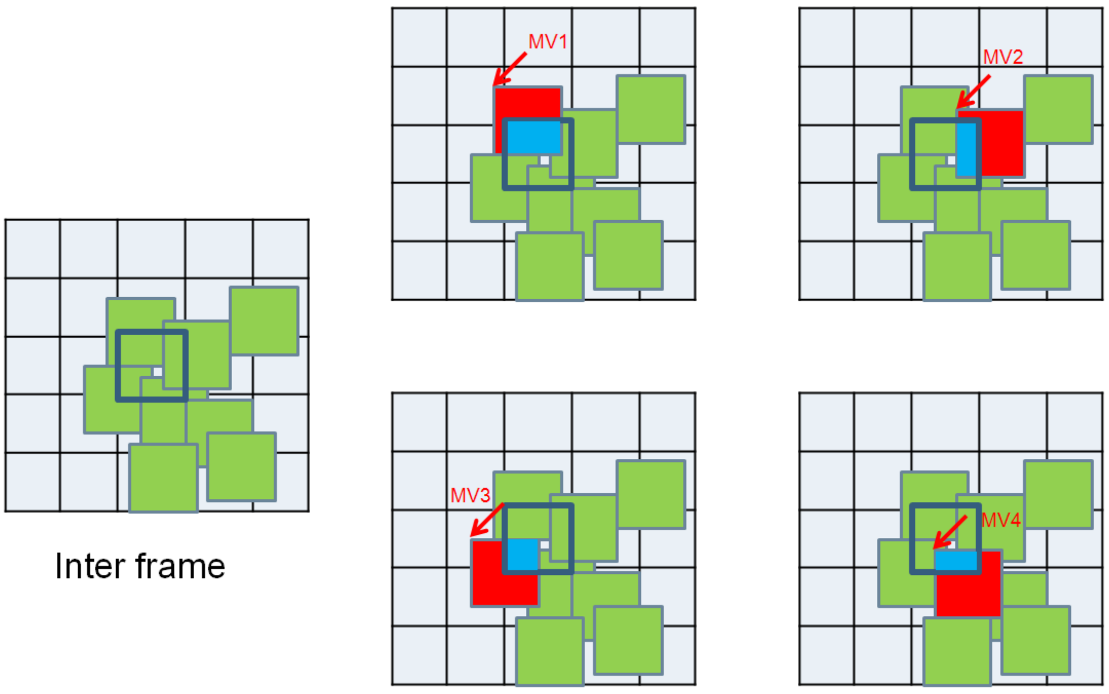


Figure 2.10 Inter block's overlap area of motion vectors' projection.

For multi-rate up-conversion of our target, we perform motion estimation twice to get forward and backward MVFs (green dotted arrows in). The first and the second inter-blocks' motion vectors are mapped by backward MVF, the third and the forth inter-blocks' motion vectors are mapped by forward MVF. For preventing blur whole inter-frame and lower the complexity, we perform uni-directional interpolation that only gets pixels from one of existing frames. Similarly, the first and the second inter-frames are interpolated by the pixels in frame n-1, the third and the forth inter-frames are interpolated by the pixels in frame n.

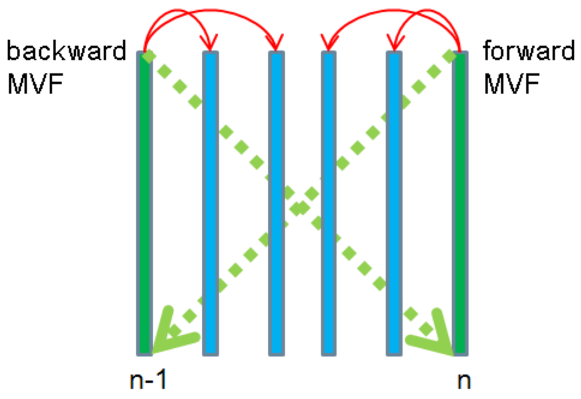


Figure 2.11 ME and MC of multi-rate up-conversion.

2.2.4 Sub-block Division with Precise Artifact Detection

Typical, the sub-block division performs on existing frames [13], [17]. But since the artifact does not appears on existing frames, dividing blocks on existing frames may not directly reduce the block artifact. For this reason, we perform sub-block division on inter-frames where artifact really appears. With the analysis of the appearance of block artifact, we found that it always appears when neighboring blocks' motion vectors are not continuous. So we simply pay attention to the blocks' whose motion vector is not continuous with the others. In this way, it is guaranteed to locate the block artifact on inter-frames, so the detection method is simple and precise.

If only consider motion vector's value, although it is precise, two neighboring blocks will be detect as they share the same motion vector discontinuity boundary. As shown in Figure 2.12, these blocks' motion vectors are discontinuous to the other, but only the yellow one should be divided into sub-blocks. For preventing this situation, we must determine which one of these two blocks should be detected. Here we apply a block matching criterion called bilateral MSEA (bi-MSEA) for the determination. In Figure 2.13, the two gray solid arrows present the inter-block's motion vector and its opposite direction. Bilateral MSEA is 8x8 MSEA of the two blocks (red solid rectangles in Figure 2.13) pointed by these two motion vectors on existing frame. By and large, the bi-MSEA value indicates the reliability of inter-block's motion vector, as well as how this inter-block being interpolated. Thus this criterion can help us to determine which one of two blocks should be detected.

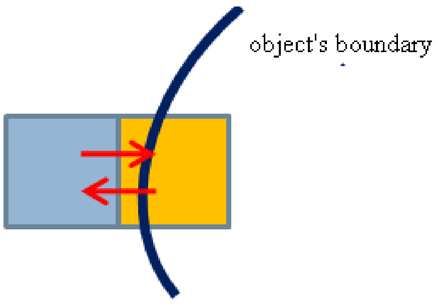


Figure 2.12 Two blocks sharing he same motion vector discontinuity boundary.

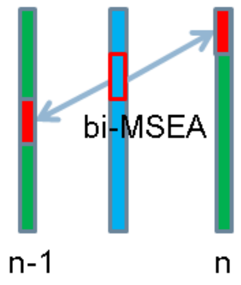


Figure 2.13 Graphic illustration of bilateral MSEA.

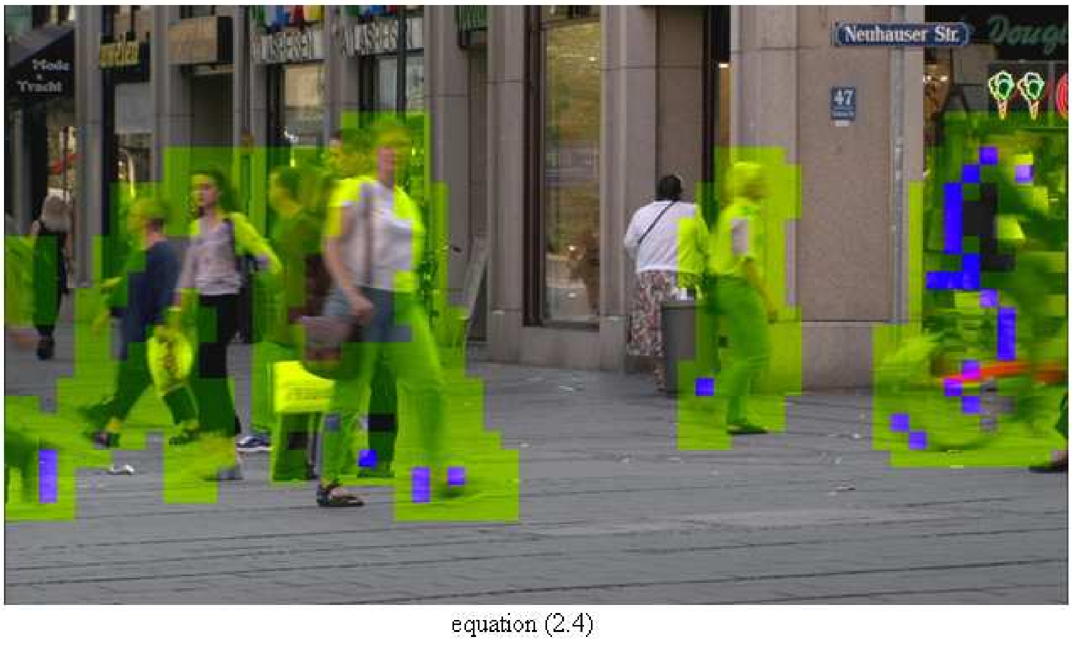
We devise an artifact detection condition that indicates whether this block has artifact. The condition is satisfied if

$$\left|MV_{x_{inter.}} - MV_{x_{4-neigh.}}\right| > 2 \ \ or \ \ \left|MV_{y_{inter.}} - MV_{y_{4-neigh.}}\right| > 2 \quad (2.4)$$

and

$$bi - MSEA_{inter.} > bi - MSEA_{4-neigh.} \quad (2.5)$$

where equation(2.4) stands for block artifact detection (4-neighbor) and equation(2.5) stands for determining one of the two blocks. Figure 2.14 show different detecting results (yellow and blue region). If only consider equation(2.4), many non-essential blocks are detected. With the help of equation(2.5), those non-essential blocks are removed from detection.



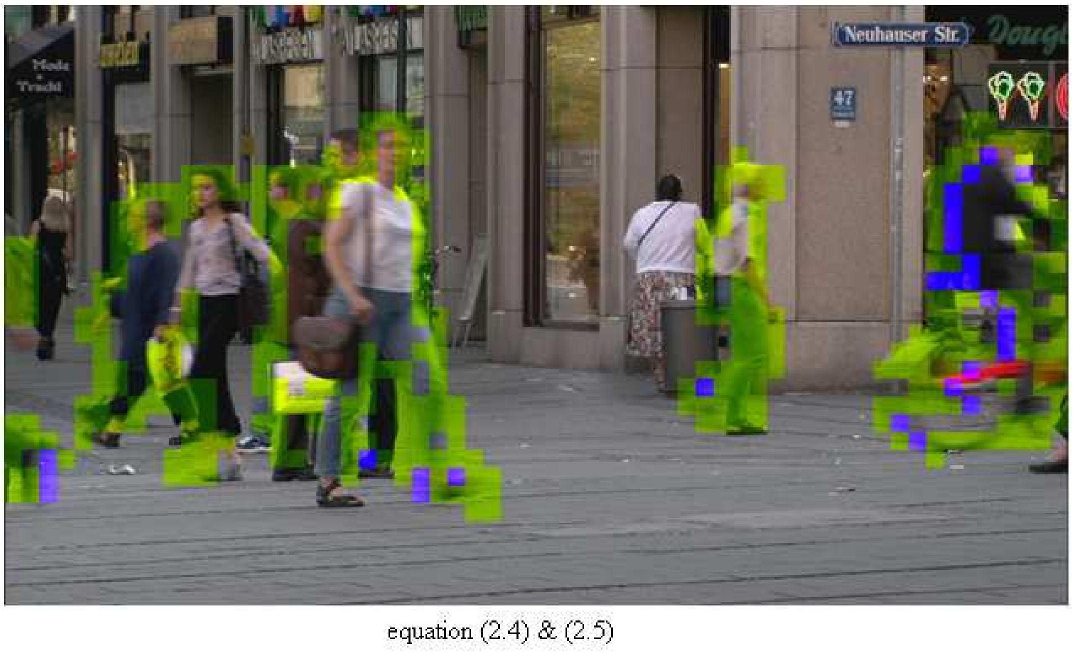


Figure 2.14 Detected 32x32 blocks by different conditions.

Combining artifact detection and sub-block division, we regard all blocks of inter-frames as 16x16 sub-blocks and check all the artifact detection conditions of its four neighbors. Among these four neighbors, only two of them are able to be satisfied since the other two (gray rectangles in Figure 2.15) lie in the same 32x32 block of the sub-block with the same motion vector value. If at least one of the conditions is satisfied, we label this sub-block (dark blue rectangle in Figure 2.15) for post-processing and assign an initial motion vector (dark blue arrow in Figure 2.15) to it. If the bi-MSEA value of this labeled sub-block is smaller than 512, we set the initial motion vector equal to the original motion vector of it, else the initial motion vector equal to the weighted sum of neighboring motion vectors whose condition is satisfied (pink arrow in Figure 2.15). Since the post-processing only operate on labeled sub-blocks, we do not actually divide all blocks into sub-blocks.

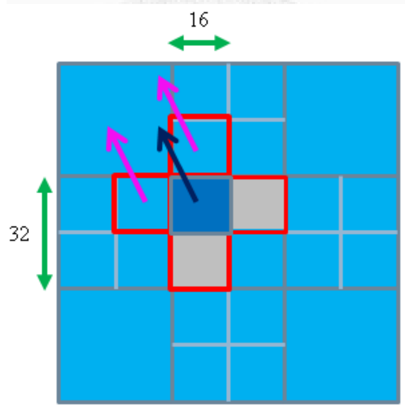


Figure 2.15 Sub-block division with precise artifact detection.

The purpose of initial motion vector assignment is to give labeled sub-block a predictor for searching new motion vector. With the predictor, we don't have to re-estimate the new motion vector in large search window. Since true motion often comes from nearby block, the predictor is equal to the weighted sum of neighboring motion vectors if the bi-MSEA value of labeled sub-block is bigger than them. Figure 2.16 shows the effect of initial motion vector assignment. The result of sub-block division with artifact detection is shown in Figure 2.17. It is apparent that the most of labeled sub-blocks lie on moving objects' boundary since there is motion vector discontinuity. The non-labeled region is guaranteed to have no artifact as in the figure.

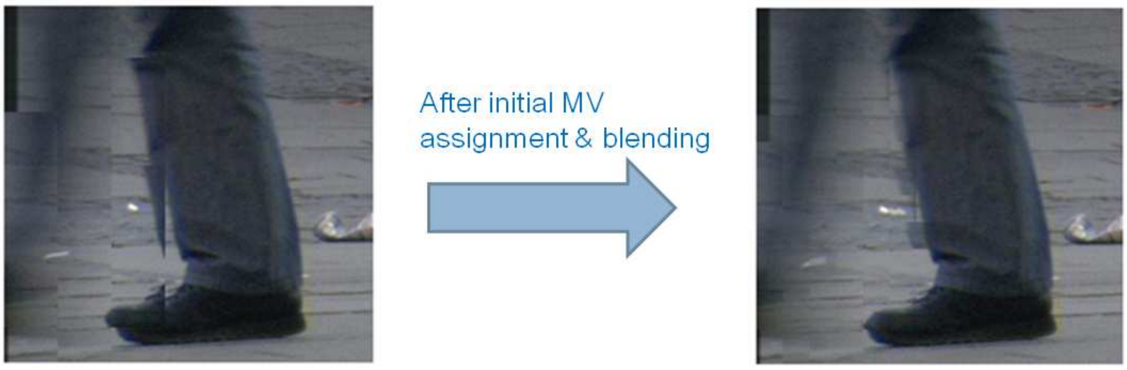


Figure 2.16 The effect of initial motion vector assignment.

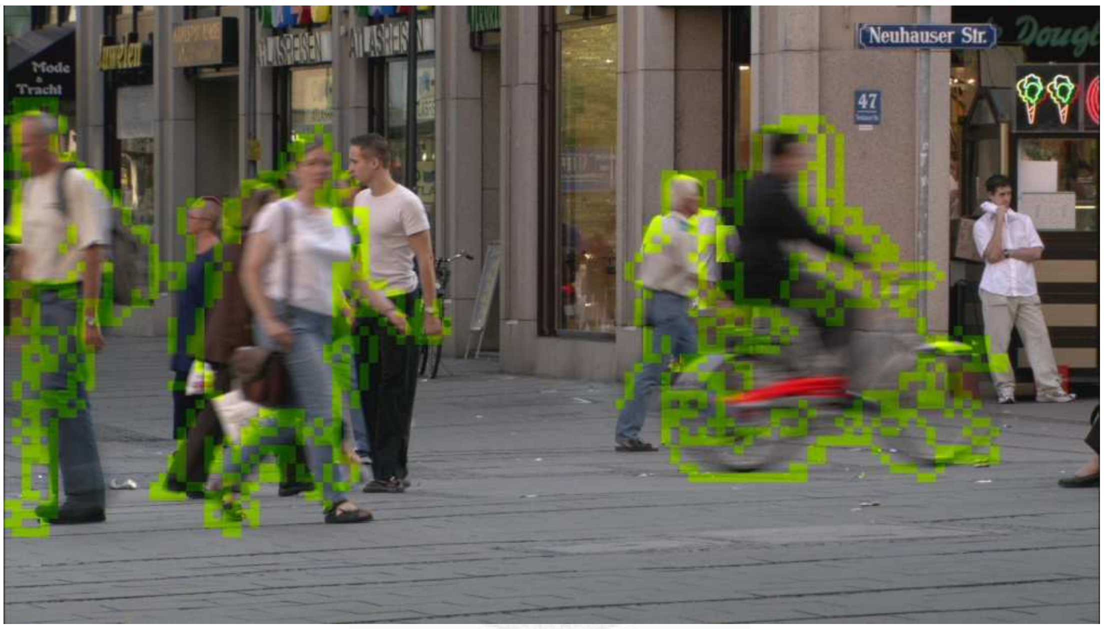


Figure 2.17 Labeled 16x16 sub-blocks on inter-frame.

Table 2.1 Total # of labeled sub-blocks in the worst cases of each sequence.

sequences	**total # of sub-blocks**	**percentage**
pedestrian_area	2787	8.54%
Titanic-2	1820	5.58%
Vintagecar	3074	9.42%
ducks_take_off	1340	4.11%
park_joy	2474	7.58%
Tractor	1969	6.03%
transformer 7-3	3947	12.09%

We count the number of labeled sub-blocks and list the results of the worst cases in Table 2.1. For the most complex sequence (transformer 7-3), there are only about 12% sub-blocks with block artifact labeled for the following post-processing.

2.2.5 Sub-block Refinement with Bilateral Motion Vector Search and Overlapped Block Motion Compensation

The last step is post-processing which consist of three parts: artifact region detection, motion vector refinement and artifact-reducing interpolation. Since we divide the sub-blocks only on artifact region, we don't have to perform artifact region detection in this step.

For motion vector refinement, we want to find the motion vector with the least block artifact. Similar to [23], we employ side match technique [32] as our matching criterion for motion vector refinement. For a labeled sub-block, around its initial motion vector and the opposite direction we open two search windows with range ±8x±8 as in Figure 2.18. The search window of opposite direction stands for occlusion handling. There are outside pixels around labeled sub-block's boundary (dark blue hollow rectangles in Figure 2.19) on inter-frames. Go along with the motion vector in search windows, there are inside pixels around corresponding sub-block's boundary (dark green hollow rectangles in Figure 2.19) on existing frames. The boundary error is defined as the sum of absolute different of each outside and inside pixel pairs as shown in right graph of Figure 2.19. Among all motion vectors in search windows, we choose the one with the least BE. as the new motion vector of labeled sub-block. After finding new motion vector of labeled sub-block, we temporarily interpolate the inside boundary pixels of this sub-block for next labeled sub-block's boundary error computing.

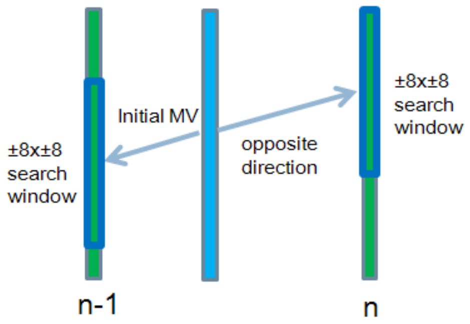


Figure 2.18 Search window for motion vector refinement.

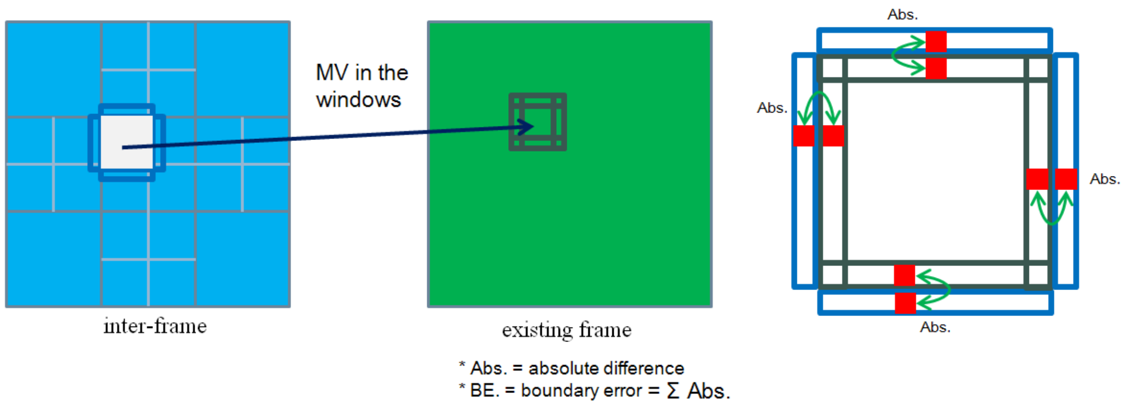


Figure 2.19 Boundary error computation.

After finding new motion vectors of all the labeled sub-blocks, we perform OBMC on these sub-blocks like [11], [19]. First we get five motion vectors from the original, up, right, down, left neighboring sub-blocks (Figure 2.20). Then we apply each motion vector to this sub-block to get pixels from existing frames, multiply them by the weighting map show in Figure 2.21. The original motion vector is applied to whole sub-block with diamond-shape weighting map. The up neighboring motion vector is applied to top-half of the sub-block with descending weighting map and other neighboring motion vectors work as well. After multiplying all weighting map, we sum up those weighted pixels and divide them by 16 to generate normalized weighted sum pixels of interpolated sub-block.

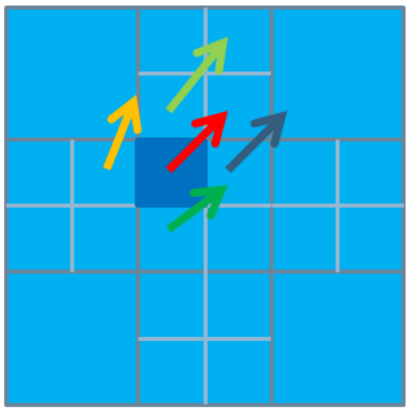


Figure 2.20 Five different motion vectors for OBMC.

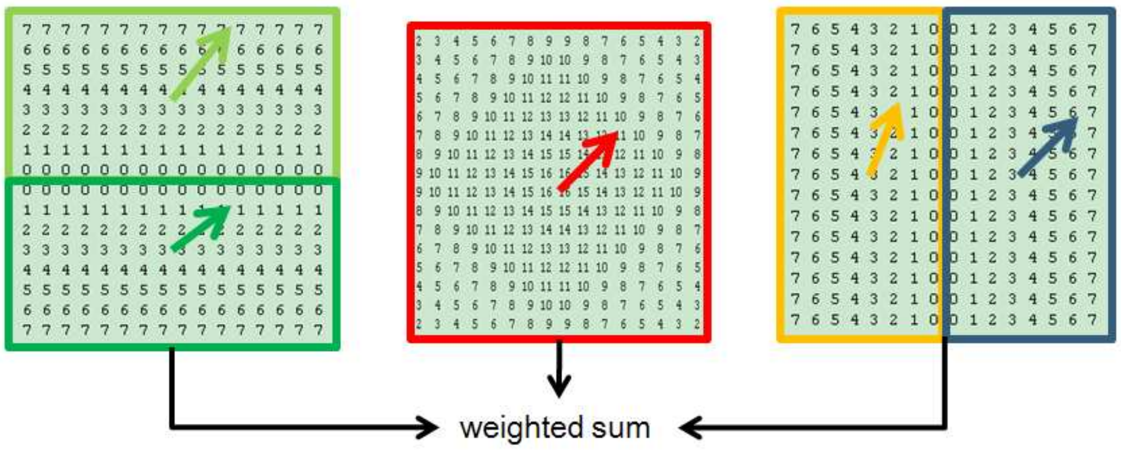


Figure 2.21 Weighting map of each motion vector for OBMC.

Since the weighting map of neighboring motion vectors are descending from outside and the origin weighting map is descending from inside, the pixel values are partially continuous to each side. This technique only blurs the boundary with motion vector discontinuity, thus further reduces the block artifact only on necessary region. Figure 2.22 shows the sub-region of inter-frames before and after post-processing. Many labeled sub-blocks are corrected with less block artifact by proposed motion vector refinement and OBMC technique.

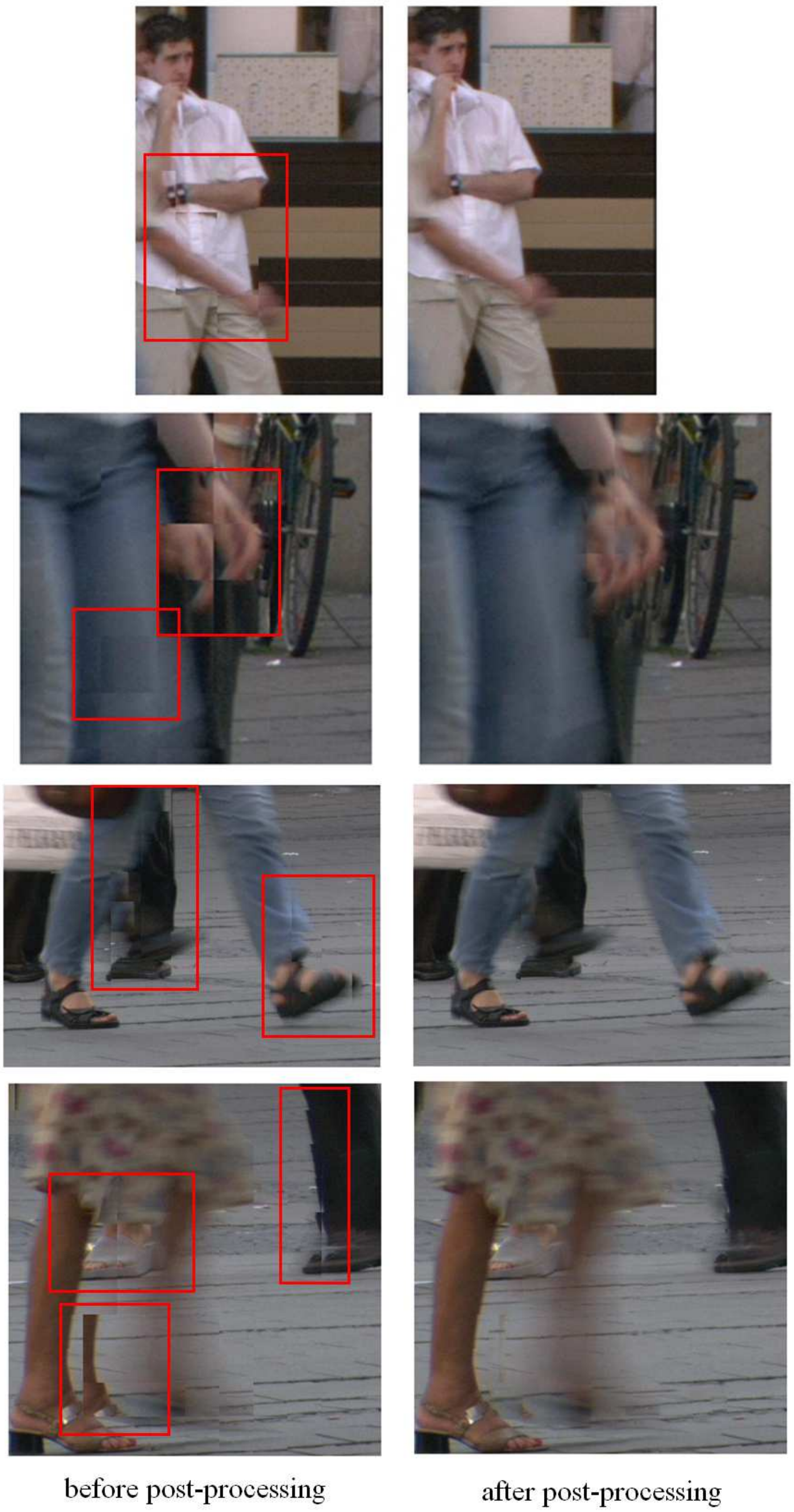


Figure 2.22 Artifact reduction after post-processing.

2.3 Performance Evaluation

We select three papers for performance evaluation. Yang’s algorithm [11] extracts MVF from JM reference software 15.1, performs OBME and OBMC; Percept’s algorithm [12] also extracts MVF from JM, and ignores motion vectors that are perceptually unapparent; GME’s algorithm [17] performs global motion estimation and sub-block division.

2.3.1 Subjective Evaluation

There are thirty eight subjects for subjective evaluation. Twenty two of them come from National Taiwan University electronic engineering students. Other sixteen come from internet. The experimental method is letting them watch twice up-converted 1080p sequences (Figure 2.23) of 4 algorithms frame by frame at the same time. In the end each subject votes the best one among four algorithms of all the sequences. Figure 2.24 shows the final experiment results. At least 79% of subjects choose our algorithm as the best one among all sequences.

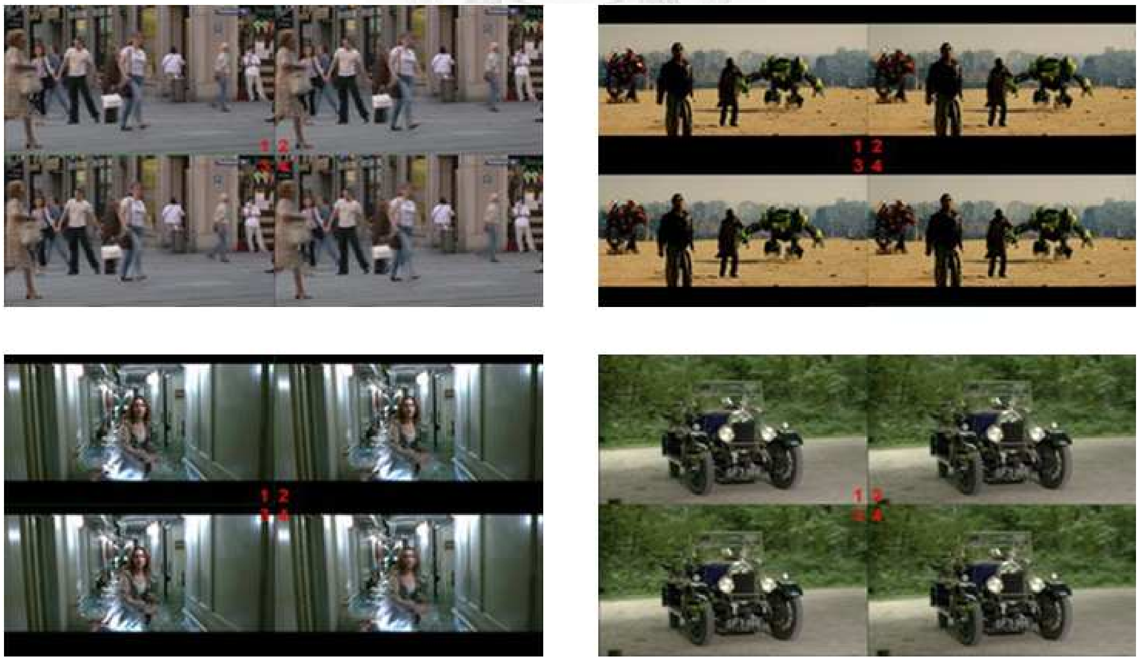


Figure 2.23 Four 1080p 24Hz sequences for evaluation: pedestrian_area, transformer 7-4, Titanic-2, vintagecar.

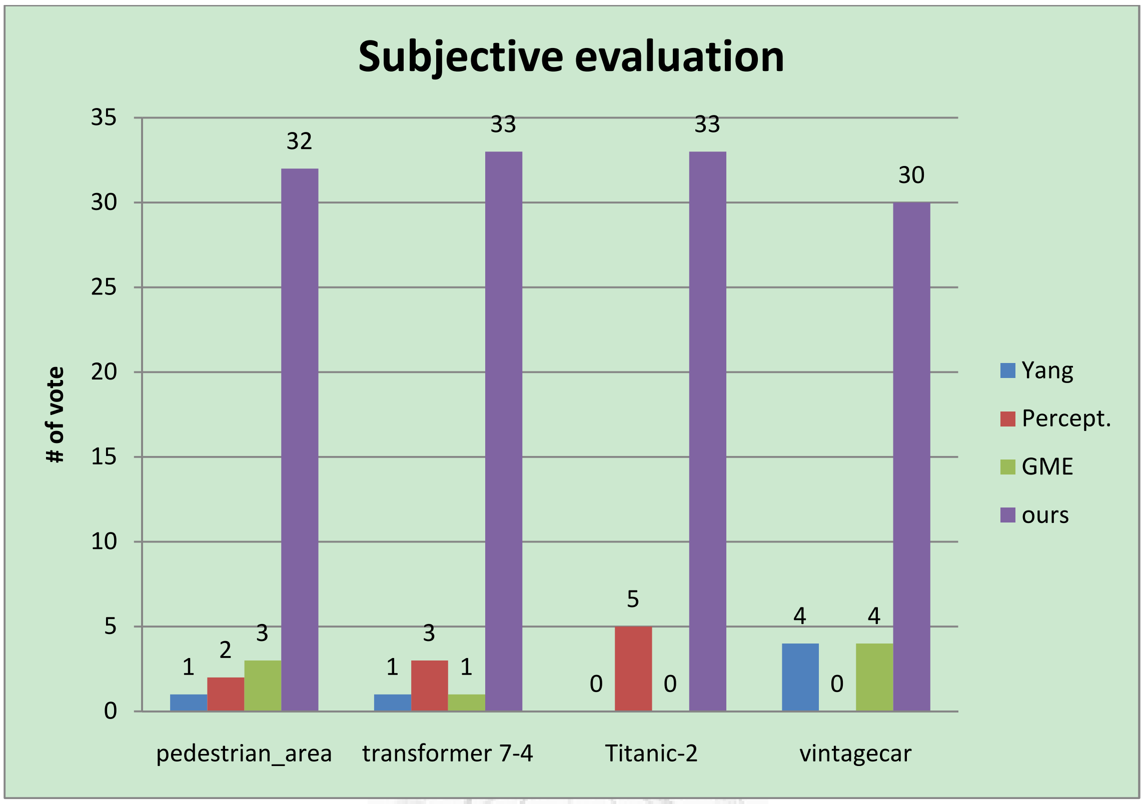


Figure 2.24 Experiment results of subjective evaluation.

2.3.2 Objective Evaluation

By twice up-converting the odd frames of original sequences, we compute the PSNR values of interpolated frames to even frames of the original as ground truth (Figure 2.25). The PSNR values frame by frame are showed in also with average and PSNR gain. Figure 2.26 shows the results of each sequence. It indicates that the PSNR values of our algorithm are the best in the most of frames. The averaging value is also the best with 0.63 to 5.47 PSNR gain.

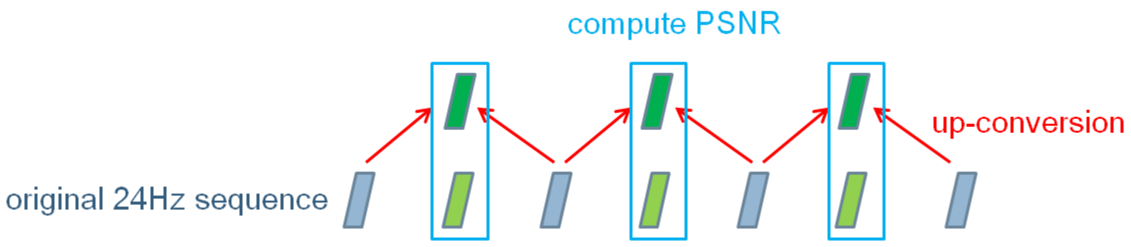
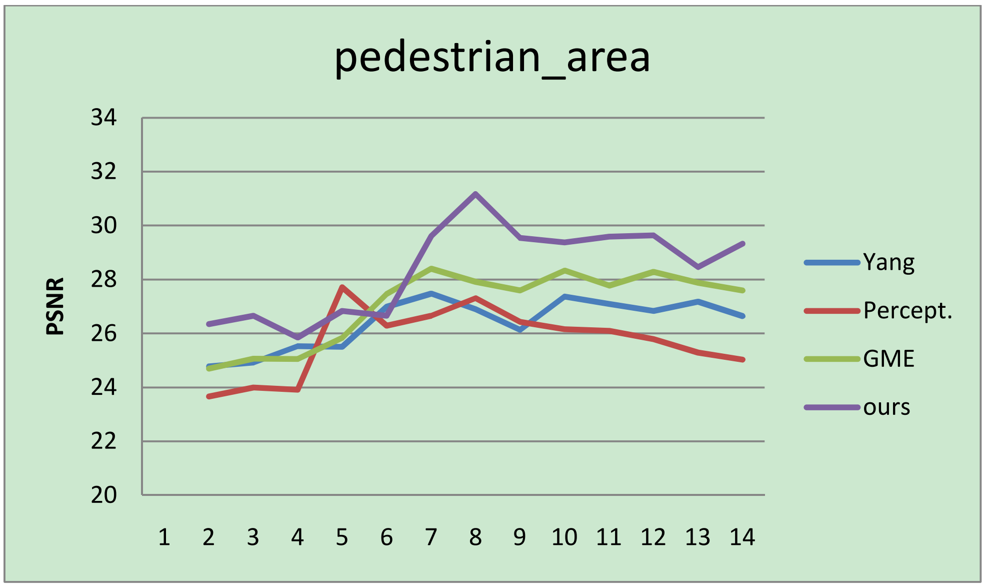
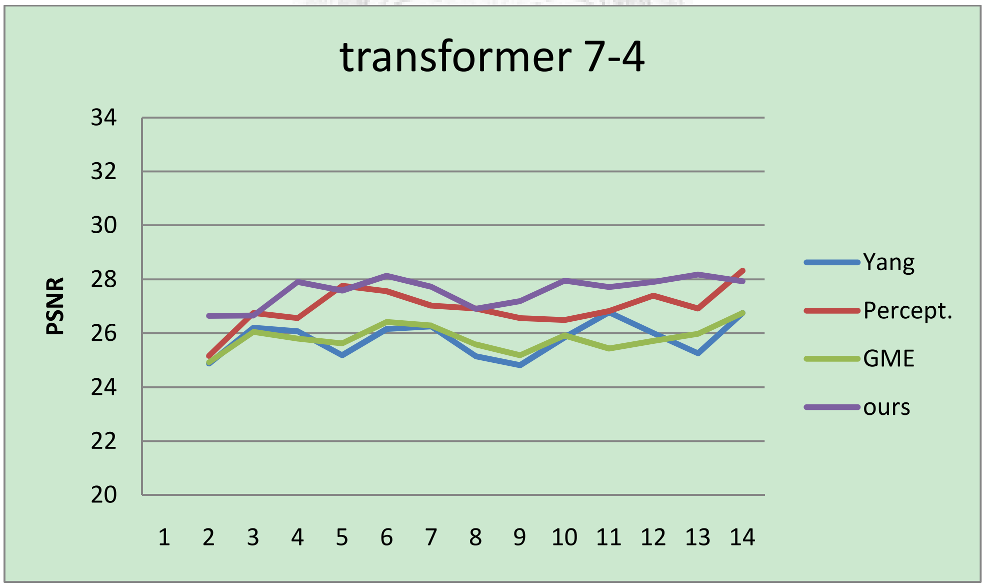


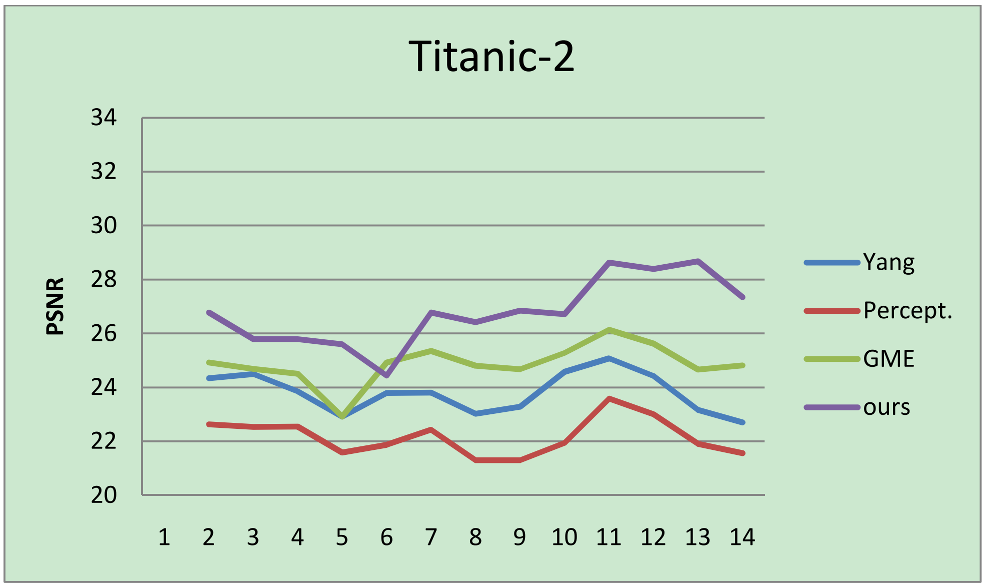
Figure 2.25 PSNR comparison for objective evaluation.



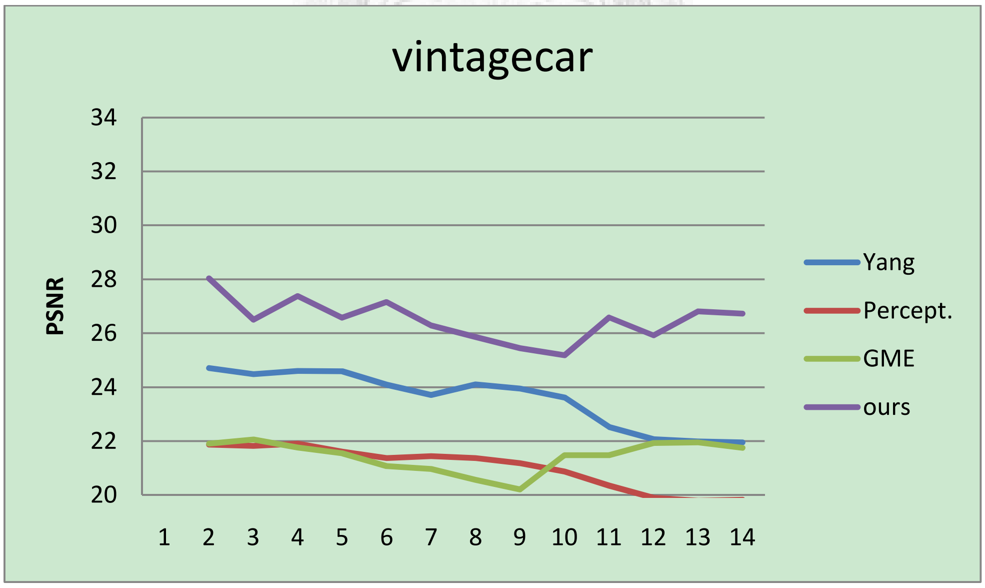
	Yang	Percept.	GME	ours
avg.	26.41	25.72	27.07	28.39
PSNR gain	1.97	2.67	1.32	0.00



	Yang	Percept.	GME	ours
avg.	25.79	26.94	25.82	27.57
PSNR gain	1.78	0.63	1.75	0.00



	Yang	Percept.	GME	ours
avg.	23.80	22.17	24.87	26.78
PSNR gain	2.98	4.62	1.91	0.00



	Yang	Percept.	GME	ours
avg.	23.57	21.02	21.44	26.50
PSNR gain	2.93	5.47	5.06	0.00

Figure 2.26 PSNR values of each sequence for objective evaluation.

2.4 Summary of Algorithm Design

In this chapter, we introduce the steps of FRUC algorithm and describe the proposed algorithm in detail with reasons.

For motion estimation, we propose a predictive square search based on true motion estimation. Experiment shows that at least 60% of blocks converge at predictor with only 25 distortion computation. The block size is equal to 32x32 and the matching criterion is 8x8 MSEA for complexity concern, and the search range is determined to be ±128x±128.

We use Marcov random field modeling for motion vector processing, which is also based on true motion estimation. Among the minimization method, we choose the simplest ICM with selected candidates. Research shows that using nine candidates is already enough with the benefit of preventing over smoothing and low computation cost.

Although none of three general motion vector mapping methods is perfect, we perform a block-based through motion compensation which has no timing mismatch and interpolates inter-frames block by block.

We introduce a precise detecting technique for locating the block artifact on inter-frames. With the help of bi-MSEA, we only label necessary sub-blocks for post-processing and assign them an initial motion vector. Experiment shows that there are at most 12% of sub-blocks have to be labeled.

In post-processing, we adopt the boundary error criterion to find the new motion vector of labeled sub-block with the least block artifact. We open a search window in opposite direction for occlusion consideration. In the end the OBMC technique is applied to further reduce the block artifact.

By the results of performance evaluation, the proposed algorithm is better than the other algorithms both in subjective and objective evaluation.

Chapter 3 Architecture Design of Multi-rate Frame Rate Up-conversion on LCD

In this chapter, we first analyze the specification of our design target, including DRAM selection and compute the resource available. Since the proposed algorithm consists of variety of steps, the hardware and on-chip SRAM reusing becomes an important topic. In the algorithm, there are many dependencies between blocks causing data pre-fetching and pipeline bubble reduction to be tough. And since we interpolate multi-frames, how to efficiently utilize the available bandwidth is also a significant matter. After specification analysis, we give an overview of the architecture. The procedures of proposed FRUC algorithm are performed one by one, frame by frame. And then we explain proposed hardware architecture of each part for resolving the architecture design problems in detail. Later we show the cycles and bandwidth consumption by simulations, and the final hardware specification with efficiency comparison. In the end we give a summary of proposed architecture design.

3.1 Specification Analysis

3.1.1 Introduction

Figure 3.1 is the general LCD system. Beside the modules for I/O and display, there are many DSPs for video processing connected by a system bus. The ARM CPU handles the AMBA protocol to let them share the DDR SRAM (DRAM). Since FRUC DSP consuming larger bandwidth than the others, it often has its own DRAM without sharing by the others for up-conversion. The path between FRUC and DRAM is showed in Figure 3.2 and we assume there is fifty cycle latency with uncertainty. The clock frequency is assumed 300M Hz and the I/O width of system bus is 16 bytes per cycle. Pixels in DRAM are arranged as four successive pixels per address in two banks, raster scan order (Figure 3.3), thus it is possible to get 16x1 or 8x2 pixels at a time.

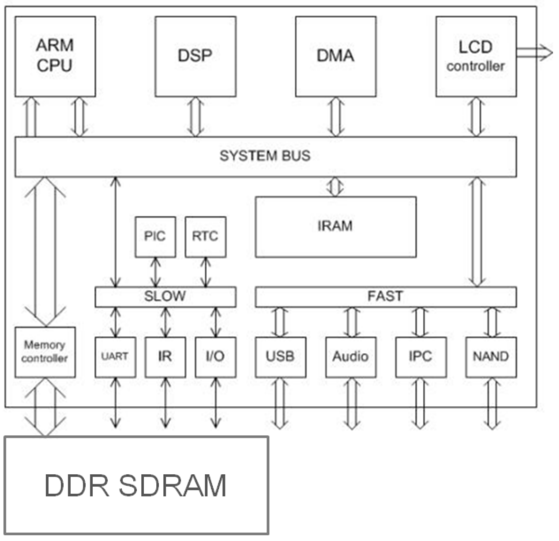


Figure 3.1 General LCD system [33].

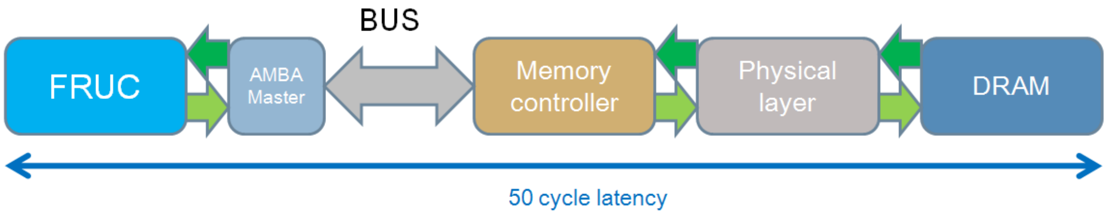
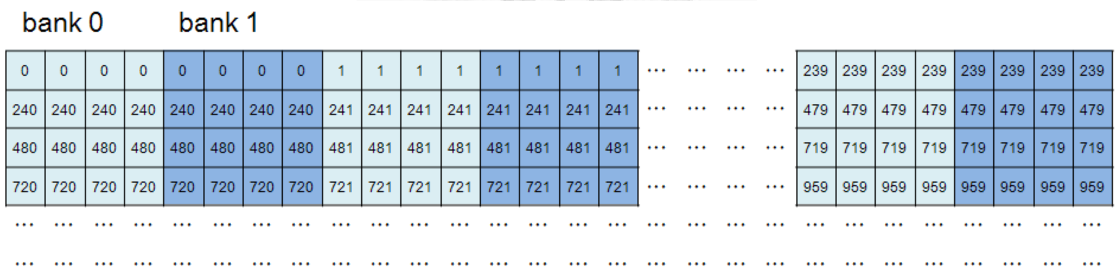


Figure 3.2 The path between FRUC and DRAM.



bank 0				bank 1																							
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	…	…	…	…	239	239	239	239	239	239	239	239
240	240	240	240	240	240	240	240	241	241	241	241	241	241	241	241	…	…	…	…	479	479	479	479	479	479	479	479
480	480	480	480	480	480	480	480	481	481	481	481	481	481	481	481	…	…	…	…	719	719	719	719	719	719	719	719
720	720	720	720	720	720	720	720	721	721	721	721	721	721	721	721	…	…	…	…	959	959	959	959	959	959	959	959
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…

Figure 3.3 Pixel arrangement in DRAM.

3.1.2 DRAM Selection

With the reference to NXP 5100 FRUC solution [34], it generates 1080p 120Hz video sequence with 2pcs 16bit x 512Mb DDR2-667 DRAM. Since our target is 4Kx2K 120Hz video sequence which is four times of NXP 5100, we choose 4pcs 16bit x 1Gb

DDR3- 1333 (MT41J64M16JT-15E in [35]), each bank with 2pcs as our DRAM for FRUC whose data rate and storage are four times than NXP 5100. By the spot price of [36] we found that the price gap between DDR2 and DDR3 becomes smaller, so we think for next generation of LCD DDR3 will replace DDR2 as the DRAM on the system.

3.1.3 Resource Available

For the selected DDR3 DRAM, the declared maximum is equal to 1333M Hz x 4pcs x 16bits / 8bits = 10666MB per second. But since there is a huge DRAM random access penalty, we assume 65% probability of request failure. The real bandwidth available is 10666MB x (100% - 65%) = 3733.3 MB per second. For 24Hz to 120Hz up-conversion, there is 3733.3MB / 24fps = 155.6MB for all the FRUC operations, and for 60Hz to 120Hz up-conversion, there is 3733.3Mb / 60fps = 62.2MB. The maximum cycles available for 24Hz to 120Hz is 300M Hz / 24 fps = 12.5M cycles, and for 60Hz to 120Hz is 5.0M cycles.

3.1.4 Summary

By the above analysis, we found that the resource is critical. Bandwidth is the most important issue since FRUC performs motion estimation like encoder and motion compensation like decoder, which are the most consuming parts of them, not to mention multi-frame up-conversion. The available bandwidth only support reading or writing up to 13.0 frames for 24Hz and 5.2 frames for 60Hz. The cycles available are also tight if we do not handle data pre-fetching and pipeline bubble reduction problems well. The dependencies between blocks of each operation make this task to be harder. For the variety of each step in the algorithm, how to employ hardware re-use is important for decreasing area overhead. The on-chip SRAM arrangement is also significant for supporting ±128x±128 search range and shared by all of the modules. As for limited resource, all pixel comparisons operate at 1080p’s scale, such as 8x8 MSEA, MRF energy, bi-MSEA and boundary error.

3.2 Architecture Overview

Figure 3.4 shows the overview of proposed architecture. The FRUC control is composed of controls of each procedure. For post-processing we fetch job queue beforehand to tighten the schedule, so its control consists of two parts for data requesting control and data computation control. The Write request module handles the task of writing pixels or motion vector information to DRAM. Read request is in charge of requesting data from DRAM, and Read receive delivers the received data to each module. For the uncertainty of bus latency, there is a Job queue for request pushing and popping. SRAM of out design is shared by all of the modules and Write SRAM unit provide address generators for writing pixels into SRAM with interleaves of different procedures. The pixels read from SRAM are mostly sent to Sum-trees and Accumulators for distortion computation. These two modules are the main computation unit with small area cost. Predictor control and Origin control stand for generating address to SRAM for ME control and MRF control. MV grouping unit computes the motion vector discontinuity for motion estimation and motion vector processing and IMC unit manages the block-based through motion compensation.

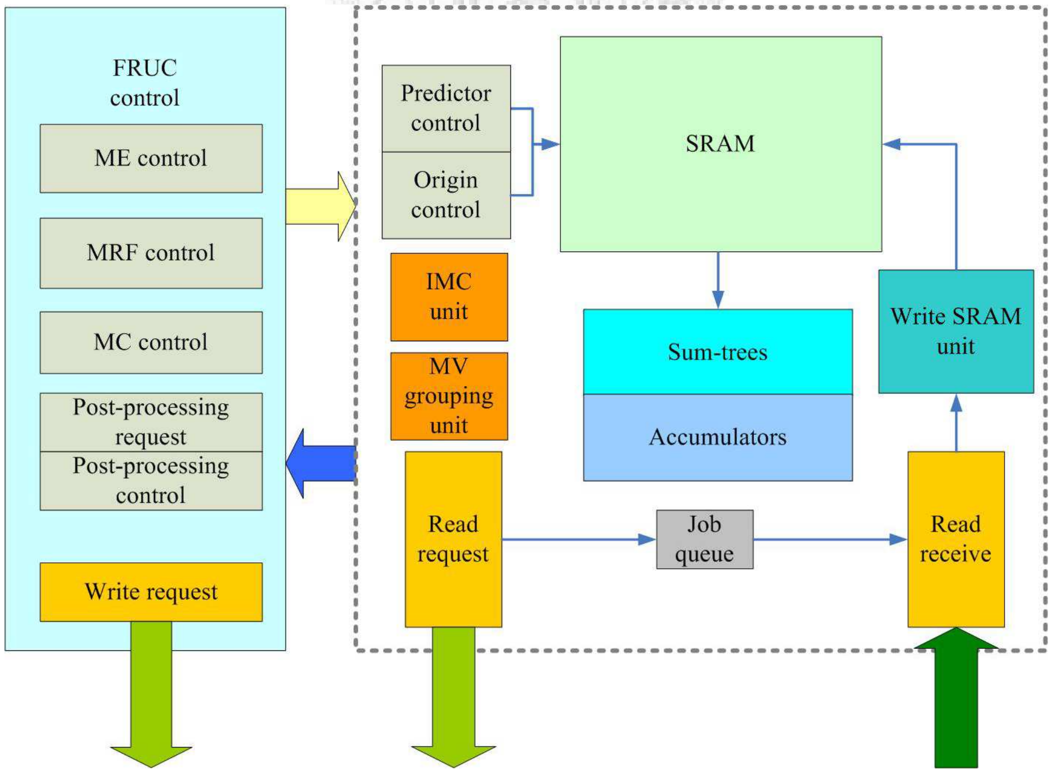


Figure 3.4 The overview of proposed architecture.

The flowchart is showed in Figure 3.5 and each procedure is done frame by frame. Down-sample cuts down the frame scale from 4Kx2K to 1080p for pixel comparison. 8x8 sum computes the sum-up of 8x8 sub-blocks of 1080p frames every eight pixels for 8x8 MSEA computing of re-estimation step in motion estimation. ME performs predictive square search motion estimation and MRF refines MVF for three iterations. MV mapping maps motion vectors to inter-blocks, and MC performs uni-directional motion compensation according to mapped MVF. Bi-MSEA computing computes the bi-MSEA value of all the sub-blocks that are possible to be labeled. MV search finds new motion vectors of labeled sub-blocks in two search windows then OBMC performs overlap block motion compensation of these sub-blocks.

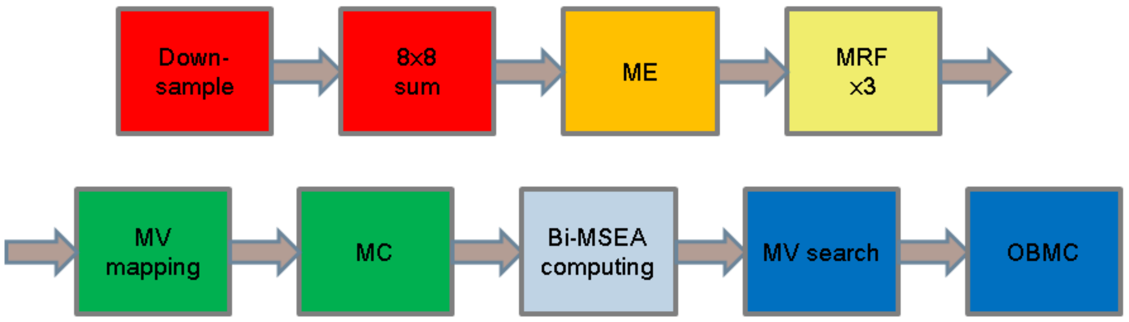


Figure 3.5 Flowchart of each procedure.

3.3 Motion Estimation Architecture

3.3.1 On-chip SRAM Issue

The first challenge encountered is on-chip SRAM issue. For supporting ±128x±128 search range, if scheme C data re-use is adopted, the SRAM size is (128 + 128 + 32)^2 = 82944 byte. For fetching data to next block, the bandwidth consumption is 2MB x (128 + 128 + 32) / 32 = 18MB per motion estimation. The cycles for fetching are (128 + 128 + 32) x 32 / 16 = 576 cycles per block. All of the resource depleted above is too much for the limited resource available.

By employing the characteristics of proposed motion estimation algorithm, we can greatly reduce the depleted resource mentioned above. For 4, 2 and 1-step convergence in the algorithm, we prepare a set of SRAM call M1 with range ±8x±8 for saving all the possible required pixels (M1 in Figure 3.6). Although setting the range of M1 to be ±7x ±7 is already enough for motion estimation, we set the size to be ±8x±8 for sharing by other modules. For 8-step re-estimation from origin with 8x8 MSEA criterion, we can directly apply scheme C on 8x8 sum generated before with much less resource consumption (O1 in Figure 3.6). The bandwidth depleted here is reduced to (2304 x 60% + 2304 x 2 x 40% + 36 x 4 x 2) x 2040(total # of blocks) = 7.2MB per motion estimation.

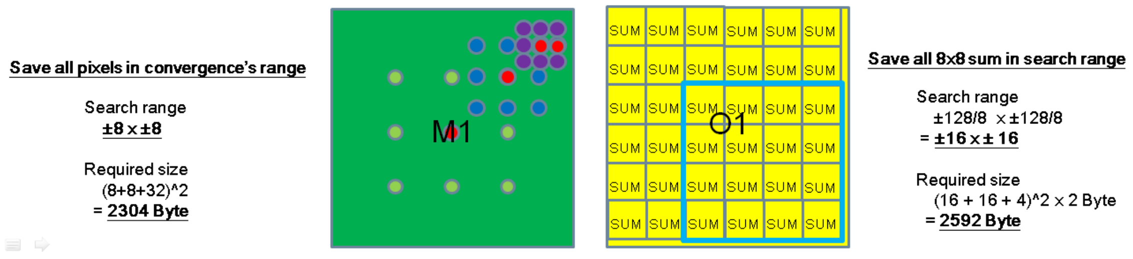


Figure 3.6 SRAM usage for motion estimation.

3.3.2 Ping-pong Two-way Scheduling

There are dependencies between blocks for motion estimation. One block must waits until the motion vector of previous block is determined for applying median filter. If we directly implement the scheduling, there will be lots of pipeline bubbles as shown in Figure 3.7. For eliminating the dependencies, we append an additional SRAM pair of M2 and O2 like original M1 and O1. Two pairs operate according to the proposed ping-pong two-way scheduling. The ping-pong means one of the pair is computing while the other is pre-fetching data, as two players hit ping-pong ball in turns. The two-way means one of the pair is raster scan and the other is inverse raster scan. By doing so, the pipeline bubbles are eliminated with ping-pong data pre-fetching (Figure 3.8). For the best balance between data fetching and MSEA computing, the cycles consumed for 4, 2 and 1-step convergence should be near 18 + 144 = 162 cycles which is explained in next sub-section.

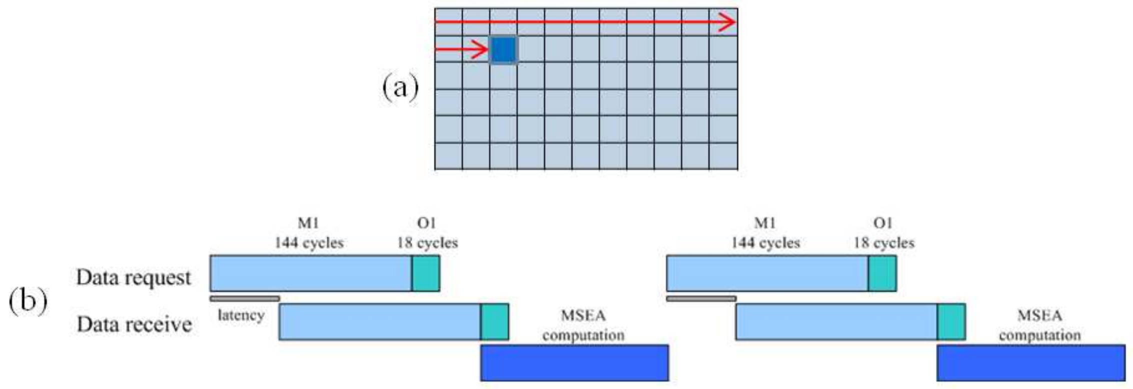


Figure 3.7 Directly scheduling. (a) raster scan order. (b) pipeline bubbles.

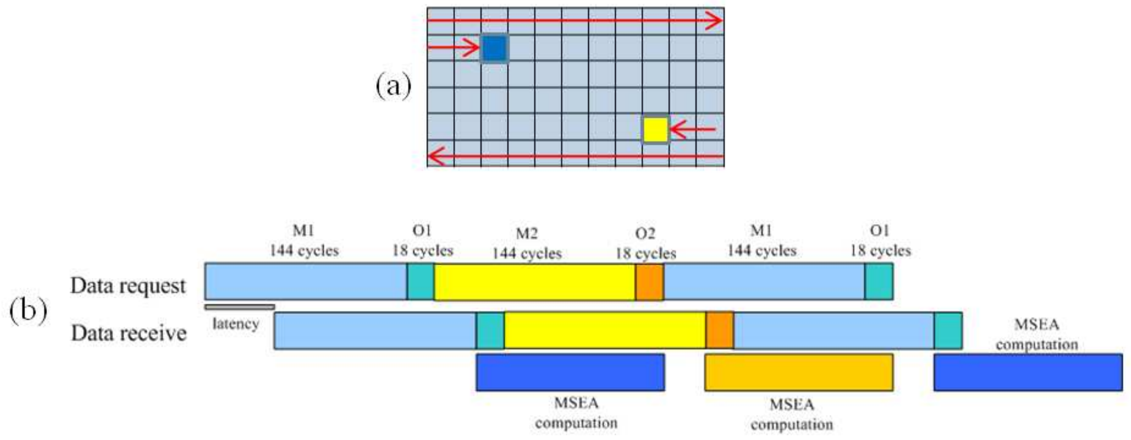


Figure 3.8 Ping-pong two-way scheduling.

(a) two-way scan order. (b) ping-pong usage without pipeline bubbles.

3.3.3 Flexible Adders

The 8x8 sums of blocks on current frame are already generated after down-sampling step, so we have to generate 8x8 sums of nine candidates of square pattern for 8x8 MSEA computing on reference frame. First we read one line of pixels in M1 (M2). As shown in Figure 3.9, this pixel line is composed of four 8x1 lines of 8x8 sub-blocks of 9 candidates for 8x8 MSEA computing. As the result, we send this pixel line into sum-tress with flexible adders to generate 8x1 sums of 8x8 sub-blocks simultaneously. Figure 3.10 showed the different arrangements of flexible adders for 4, 2 and 1-step. Black adders (with four hollow black line at the top) generate four 8x1 sums for 3 candidates at left side. Green adders are for 3 candidates in the middle and blur adders are for 3 candidates at the right side. Figure 3.11 shows that different 8x1 sums belong to different 8x8 sub-blocks of 9 candidates. The number of adders required is 49 for sum-trees.

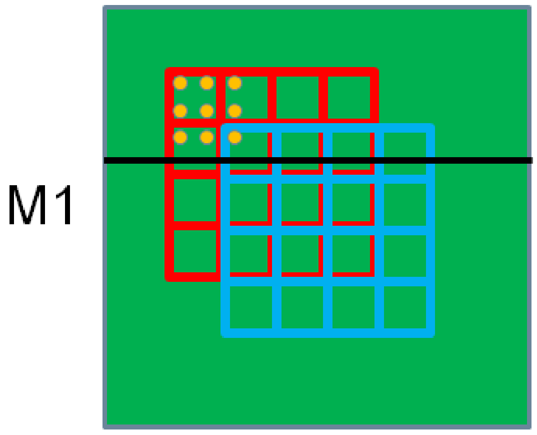


Figure 3.9 One line of pixels in M1 consist of four 8x1 lines of 9 candidates.

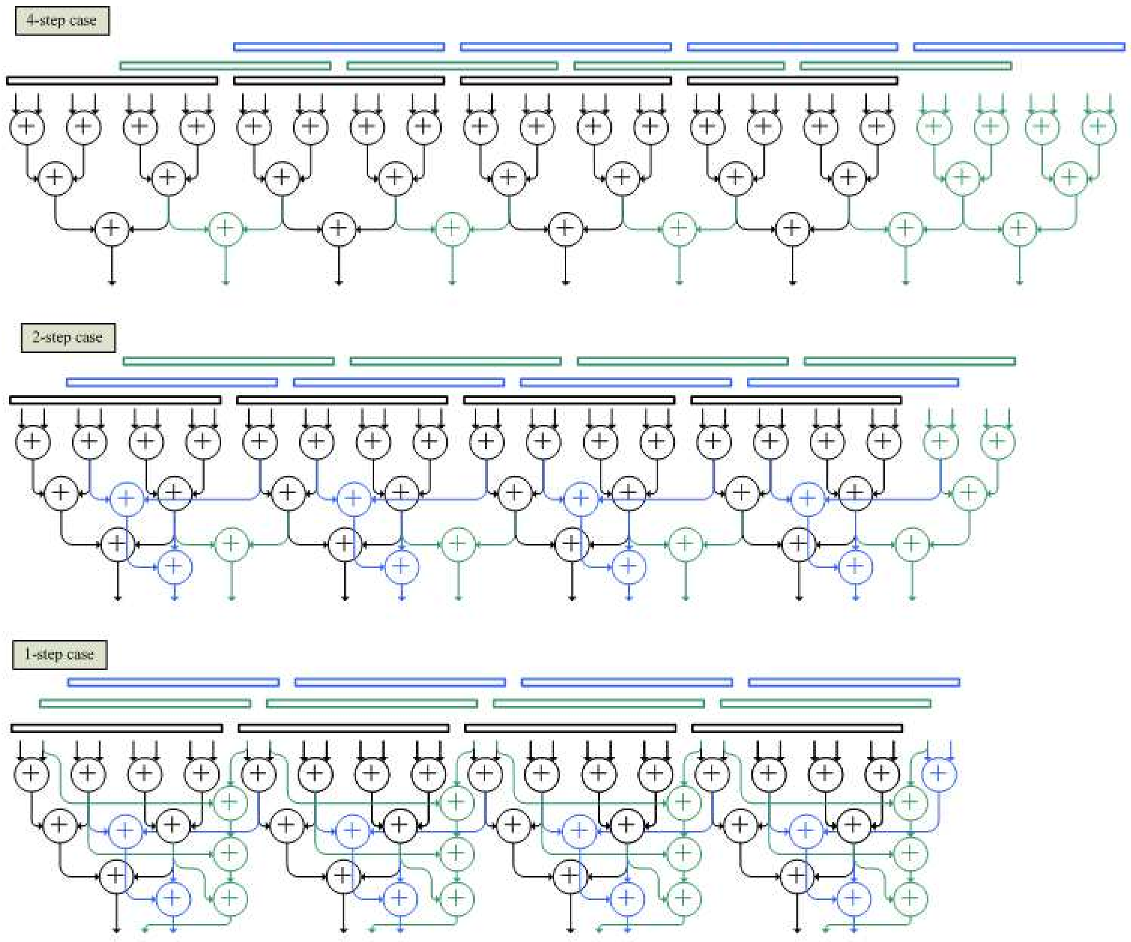


Figure 3.10 Sum-trees with flexible adders for generating 8x1 sum of 9 candidates.

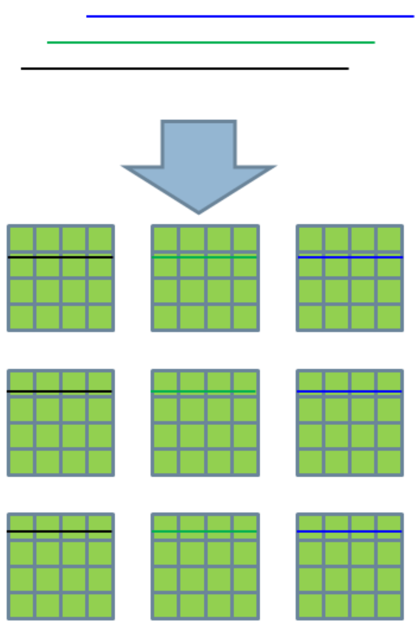


Figure 3.11 8x1sums generated by sum-trees for 9 candidates.

After generating all the 8x1 sums of all the candidates, we use accumulators to sum-up 8x1 sums for 16 8x8 sums of each candidate in parallel. Since one candidate receive four 8x1 sums at a time, we needs 9x4 = 36 adders for accumulation (Figure 3.12). The total bits of shift registers are 14bit x 16 x 9 = 2016 bits. After all of the 8x8 sums are accumulated, we send them to a 16-to-1 SAD tree candidate by candidate for 8x8 MSEA computing. The SAD tree is a part of sum-trees with 16 ABS units for outputting sum or absolute difference (Figure 3.13). After 9 cycles for sending all the 8x8 sums and 4 cycle latency, all the 8x8 MSEA of 9 candidates are generated. For 4-step square pattern, it needs 40 cycles to read all lines in M1 (M2). For 2 and 1-step it needs 36 and 34 cycles. Thus for 4, 2 and 1-step convergence, it takes (40 + 9 + 4) + (36 + 9 + 4) + (34 + 9 + 4) = 149 cycles, which are near the targeting cycles for data fetching balance. For re-estimation from origin, we directly send the existing 8x8 sums from O1 (O2) of 16 banks to SAD tree. The cycles required for 8-step is (5 + 4) or (3 + 4) = 9 or 7 cycles since each 8-step needs to compute additional 5 or 3 candidates. So for the worst of sequences, it needs 149 x 60% + (149 + 128 / 8 x 9 + 149) x 40% = 266 cycles per block. The sum-trees proposed here are also used for down-sample, 8x8 sum on whole frame, MRF energy, bi-MSEA and boundary error computing.

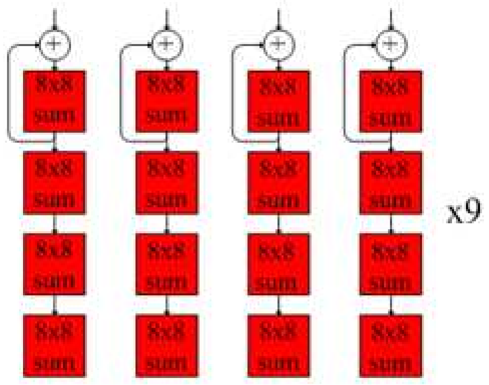


Figure 3.12 Accumulators for 8x8 sums generation.

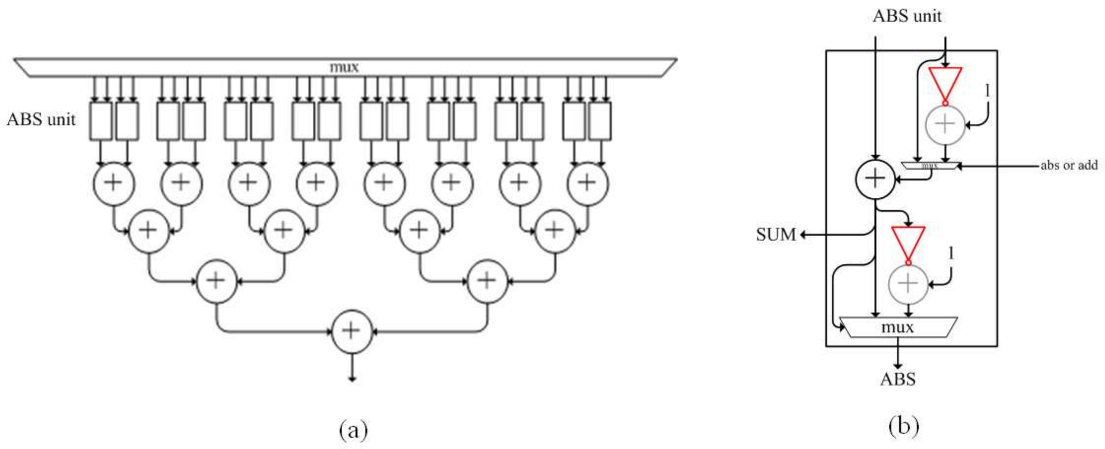


Figure 3.13 SAD tree with ABS unit. (a) SAD tree. (b) ABS unit.

3.3.4 Summary

In this section we propose a ping-pong two-way scheduling for eliminating the dependencies of blocks and pipeline bubbles. We only read the pixels needed for 4, 2, 1-step convergence and 8x8 sum for 8-step, with a huge amount of bandwidth and SRAM size reduction. The sum-trees of flexible adders and accumulators only consume 49 + 36 = 85 adders and balanced with data fetching. The number of cycles for computing a block is reduced from 576 to 266 cycles. The throughput of 4, 2, 1-step convergence is 149 cycles / 25 candidates ~= 6 cycles per candidate. With the same throughput by SAD criterion and 2D SAD trees, it needs (32x32x2) / 6 ~= 341 adders and more SRAM banks. The final synthesized SRAM size is 48 address x 128 bits x 3 banks x 2 = 4608 bytes for M1 and M2, and 84 address x 16 bits x 16 banks x 2 = 5376 bytes for O1 and O2. The SRAM size reduction is from 82944 bytes to 4608 + 5376 =

9984 bytes, and the bandwidth reduction is from 18MB to 7.2MB per motion estimation. By the way, sum-trees, accumulators and SRAM proposed in the section are all shared by other modules.

3.4 Markov Random Field Correction Architecture

3.4.1 Motion Vector Grouping Algorithm

After motion estimation, the 8x8 MSEA value of block itself will be written out to DRAM for MRF energy computing. But there are still neighboring 8 candidates with unknown 8x8 MSEA value. The motion vectors of these 8 candidates may not be the same. If we fetch pixels candidates by candidates for 8x8 MSEA computing, it will consume 2MB x 8 = 16MB per iteration and 64 x 8 = 512 cycles per block. The resource depleted is also too much by this directly implementation.

By employing the characteristics of MVF generated by our motion estimation algorithm, neighboring motion vectors are similar with many pixels required are overlapped. As shown in Figure 3.14, if we can determine the center motion vector of a group, by fetching all pixels around this center motion vector into M1 (M2), all the motion vectors whose discontinuity with center motion vector is smaller or equal to 8 (search range of M1 and M2) are able to get the pixels in M1 (M2). Thus we have to perform grouping algorithm to 8 candidates for bandwidth reduction. The group size must be bigger than 2 for bandwidth gain, so there are at most 2 groups.

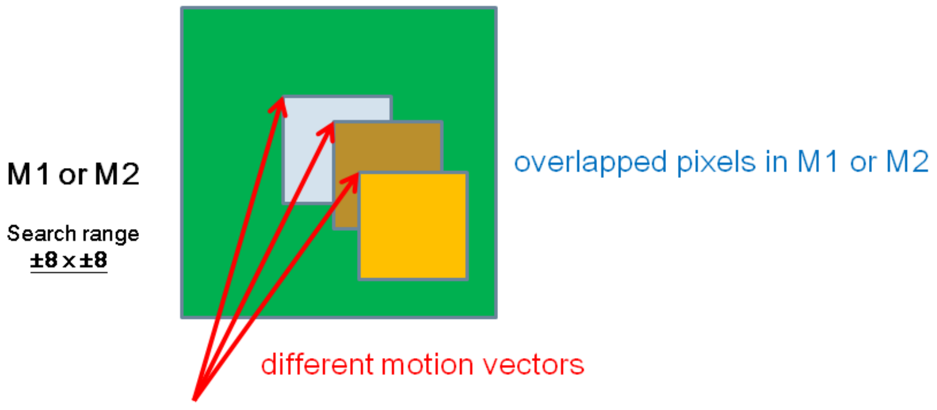


Figure 3.14 Overlapped pixels of different motion vectors in M1 or M2.

As shows in Figure 3.15, we regard 8 candidates as 8 nodes with 28 edges presenting motion vector discontinuity of nodes it connect. The discontinuities generation consumes no computation overhead since they are a part of MRF energy. The edges with discontinuity smaller or equal to 8 are labeled as dark blue is the figure, and we count the total number of labeled edges connecting to the nodes. The node with maximum number is the center motion vector of group, and the nodes with labeled edges connecting to center motion vector are the members of this group (blue nodes in the figure). After generating group 1, we must remove the nodes and edge of this group and generate group 2. The nodes that are not grouped called non-group.

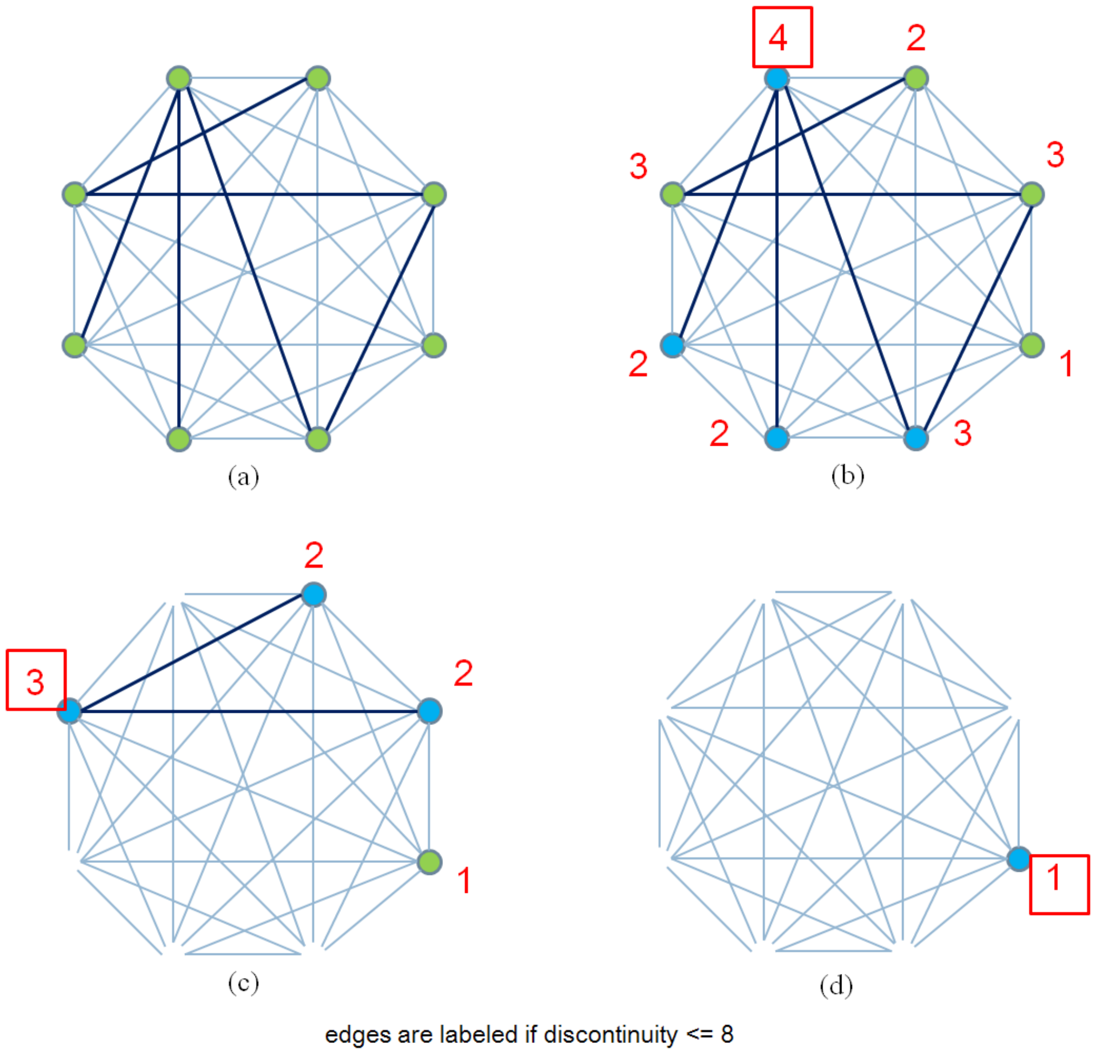


Figure 3.15 Motion vector grouping algorithm. (a) 8 nodes with 28 edges. (b) result of group 1. (c) result of group 2. (d) non-group node.

3.4.2 Grouping Architecture

Figure 3.16 show the proposed grouping architecture. We generate motion vector discontinuities one by one, and accumulate them into Total MV dis. registers for MRF energy computing. In the meanwhile, we judge whether the discontinuities are smaller or equal to 8, and label the corresponding edge registers. After the generation of all the discontinuities, we perform proposed grouping algorithm by Logic control, push the center and member motion vectors into three types of queue. The architecture is also used for median filter computing for motion estimation

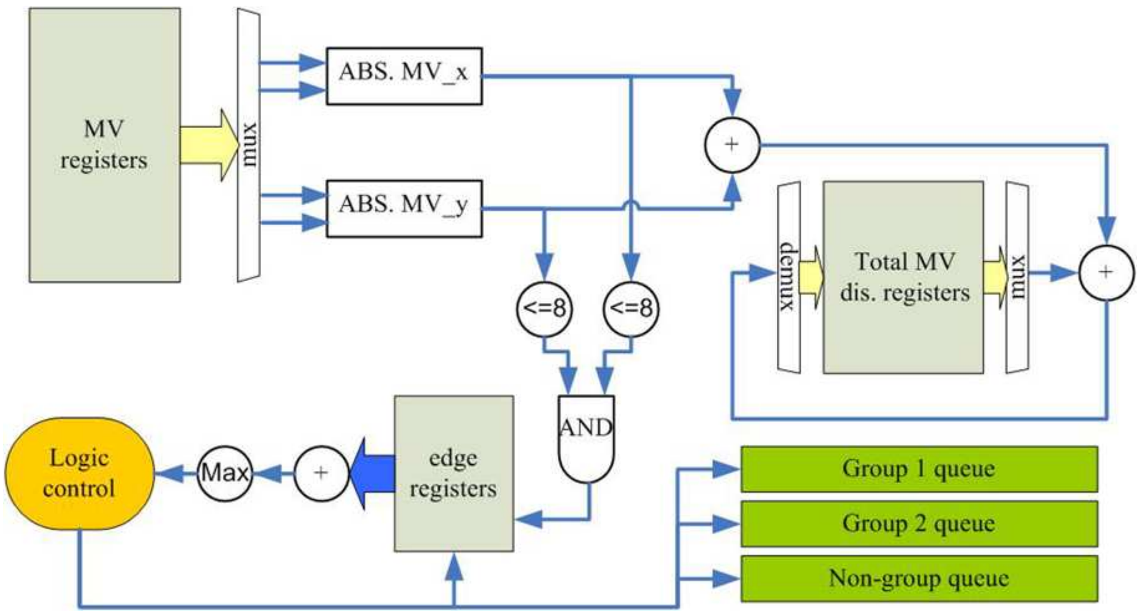
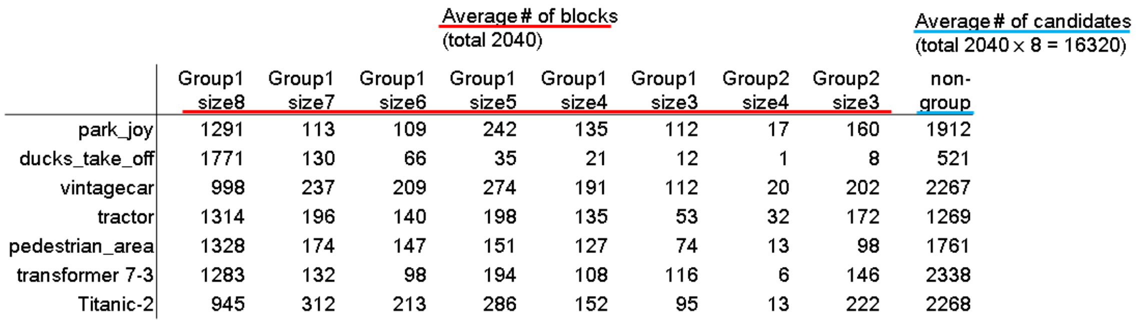


Figure 3.16 Proposed grouping architecture.

3.4.3 MRF energy computing

The discontinuity energy is already computed during grouping, so we only have to compute 8x8 MSEA value of each candidate. Table 3.1 shows the grouping results of MRF iteration 1. As we can see, there are many blocks with group 1 size 8 or size 7, and the total number of non-group candidates is at most 2338, 14% of total candidates. For MRF iteration 1, the 8x8 MSEA is computed by sum-trees and accumulators according to the three types of queue one by one. The scheduling is also ping-pong two-way like motion estimation, since there are dependencies between blocks. After getting 9 candidates' 8x8 MSEA values, we write them out to DRAM for further re-use. Take transformer 7-3 in Table 3.1 as the worst case, the cycles for a block = (32 + 4) x (16320 - 2338) + 64 x (2338) / 2040 = 320 cycles per block. And the bandwidth = 48 x 48 x (# of group 1 + # of group 2) + 32 x 32 x (# of non-group) = 4.8 MB + 2.4 MB = 7.2 MB for MRF iteration 1.

Table 3.1 Grouping results of MRF iteration 1.



	Average # of blocks (total 2040)								Average # of candidates (total 2040 × 8 = 16320)
	Group1 size8	Group1 size7	Group1 size6	Group1 size5	Group1 size4	Group1 size3	Group2 size4	Group2 size3	non-group
park_joy	1291	113	109	242	135	112	17	160	1912
ducks_take_off	1771	130	66	35	21	12	1	8	521
vintagecar	998	237	209	274	191	112	20	202	2267
tractor	1314	196	140	198	135	53	32	172	1269
pedestrian_area	1328	174	147	151	127	74	13	98	1761
transformer 7-3	1283	132	98	194	108	116	6	146	2338
Titanic-2	945	312	213	286	152	95	13	222	2268

As in Figure 2.9, the MVF changes a little after MRF iteration 2 and iteration 3. Thus for these two iteration, we load previous iteration's 9 computed 8x8 MSEA results. If the candidates whose motion vector is equal to one of the 9 computed results, we can directly use this result without 8x8 MSEA computing. For the candidates whose motion vector is not equal to all the 9 computed results, we fetch the pixels needed for 8x8 MSEA computing. By simulations, in these two iterations the worst case is Titanic-2. The cycles per block and bandwidth are 82 cycles and 1.1MB for iteration 2, 57 cycles and 0.3MB for iteration 3.

3.4.4 Summary

In this section, we propose a motion vector grouping algorithm and corresponding architecture also used for discontinuity energy generation and median filter. For 8x8 MSEA computing, we reuse sum-trees and accumulators previously proposed for motion estimation. We also employ the computed results to further reduce the resource consumption. The cycle reduction is from 512 x 3 = 1536 to 320 + 82 + 57 = 450 cycles per block for 3 MRF iterations. The bandwidth reduction is from 16MB x 3 = 48MB to 7.2 + 1.1 + 0.3 = 8.6MB for 3 MRF iterations. The SRAM and address generators for 8x8 MSEA computing are also shared with motion estimation.

3.5 Motion Compensation Architecture

3.5.1 Inverse Motion Compensation Scheduling

For tradition motion compensation, it processes block by block on inter-frames, read the required pixels of the block and write out to DRAM with proper address. For 60Hz to 120Hz up-conversion there is no problem in this way since it consumes the minimum requirement for motion compensation. But for 24Hz to 120Hz up-conversion, it needs to read pixels of four frames and write out fours inter-frames. The bandwidth = (3840 x 2160 x 1.5) x (4 + 4) = 99.5MB, and the cycles = 99.5M / 16 = 6.5M cycles. The consumption is too large since there should be a way that exhaustively utilizes the pixels read and consumes the minimum requirement for multi-frame up-conversion.

For reaching the minimum requirement, we process block by block on middle existing frame, interpolate the inter-frames near to it (blue frames in Figure 3.17). First we read on block's pixels of existing frame (red rectangle in Figure 3.17). Then for all the possible inter-blocks that may use these pixels, we derive the overlapped region along their motion vectors, and write out pixels in the overlapped region (Figure 3.18). By doing this, the resource consumed is the minimum requirement since we read only one frame and write out four frames. It is called inverse motion compensation since it operates from the view of existing frame, not from the view of inter-frames.

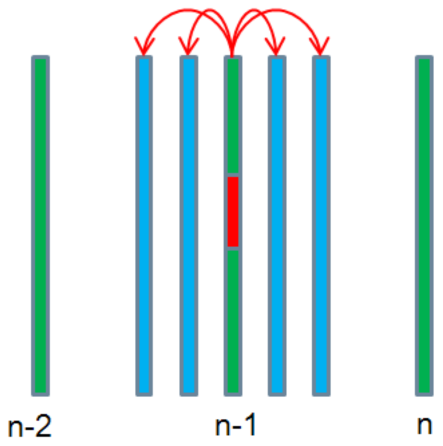


Figure 3.17 Interpolated frames of inverse MC scheduling.

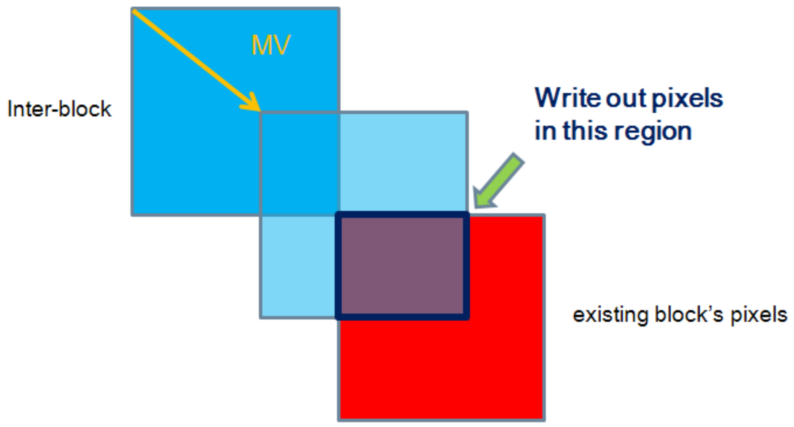


Figure 3.18 Illustration of overlapped region.

The procedure of deriving the overlapped region is similar to block-based through motion vector mapping. The only difference is that for inverse motion compensation it derive overlapped region from inter-blocks to existing block, while block-based through motion vector mapping derive overlapped region from existing blocks to inter-block (Figure 3.19). Thus we propose inverse motion compensation architecture which is responsible for both motion vector mapping and inverse motion compensation.

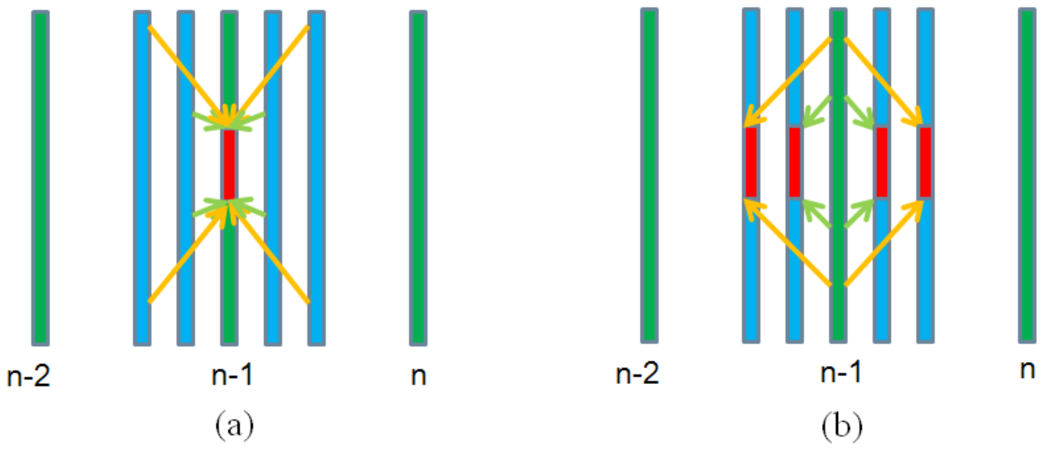


Figure 3.19 Overlapped region derivation. (a) for inverse MC. (b) for MV mapping.

3.5.2 Inverse Motion Compensation unit

Figure 3.20 shows the proposed architecture for block-based through motion vector mapping and inverse motion compensation. For motion vector mapping, the first part is deriving overlapped region's corner coordinates. Then we compute the width and length of overlapped region, multiply them to generate area of the region. At last we accumulate the area of each motion vector, select one with the biggest area as the inter-block's motion vector. For inverse motion compensation, the corner coordinates are sent to SRAM address generator and SRAM rotate unit, generating 8x2 pixels per cycles in the overlapped region for motion compensation. The SRAM here are also used in ping-pong manner, and we process Y value first then process U and V value for interpolation.

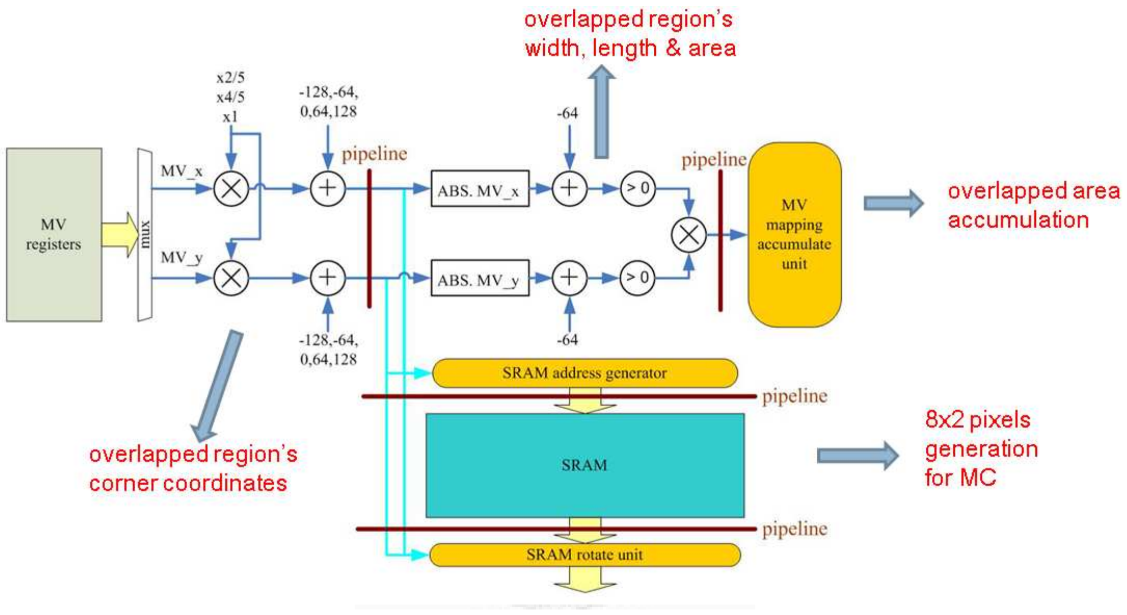


Figure 3.20 Proposed inverse MC unit.

3.5.3 On-chip SRAM interleave for Motion Compensation

For writing out 8x2 pixels per cycle, we need to random access 8x2 pixels arranged in SRAM of existing block at a time. We save the Y value of existing 64x64 blocks with additional 8 pixel columns at the right side of the block for overlapped region striding across the right boundary of the existing block (thus we save 72x64 pixels for Y value and 40x32 for U and V value). The odd lines of pixels are saved in M1 (M2) and the even lines of pixels are saved in O1 (O2). So we have to random access 8x1 pixels in M1 and O1 (M2 and O2) at a time. Figure 3.21 shows the SRAM interleaves. Different colors mean different banks, and the numbers present the corresponding addresses.

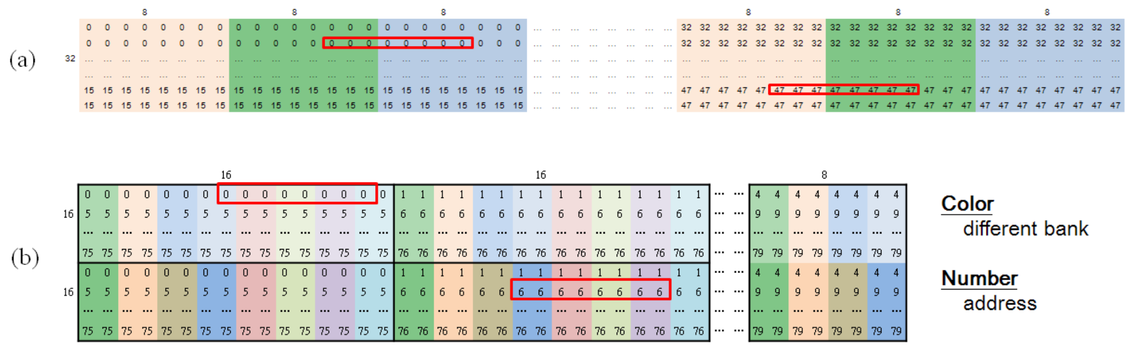


Figure 3.21 Pixel interleaves for random access. (a) interleave of M1 and M2. (b) interleave of O1 and O2.

3.5.4 Summary

For reaching the minimum requirement of multi-frame up-conversion, we propose an inverse motion compensation technique. The derivation of overlapped region is similar to block-based through motion vector mapping, thus we propose an architecture that is responsible for these two procedures. The bandwidth reduction is from 99.5MB to (72 x 64 + 40 x 32 x 2 + 64 x 64 x 1.5 x 4) x 2040 = 64.8MB. The cycle reduction is from 6.2M cycles to 64.8M/16 = 4.0M cycles. Both of them are very close to the minimum requirement. The SRAM is also shared, operating in ping-pong usage with pixel interleaves.

3.6 Post-processing architecture

Since bi-MSEA computing operates only on partial sub-blocks with motion vector discontinuity, and bilateral motion vector search and overlapped block motion compensation operates only on labeled sub-blocks, we create queues in DRAM for pushing and popping sub-blocks' information for operating. The queue of bi-MSEA computing is pushed during inverse motion compensation since there are inter-blocks' motion vectors at the moment. The queue of labeled sub-blocks is pushed right after the bi-MSEA computing.

3.6.1 Bi-MSEA computing

The computation of bi-MSEA also shares the sum-trees and accumulators proposed for motion estimation. At the beginning we pop sub-block's information from queue. And then we fetch pixels of two directions into M1 and M2 (Figure 3.22 (a)) in ping-pong manner. We generate 8x8 sums of M1 first and save them into block registers of current frame used for motion estimation. After that we generate 8x8 sums of M2 then the 8x8 sums and block registers' value (M1's 8x8 sums) to SAD tree for bi-MSEA computing. For pipeline bubble reduction, we pre-pop next queue's information for data pre-fetching of next sub-block (Figure 3.22 (b)).

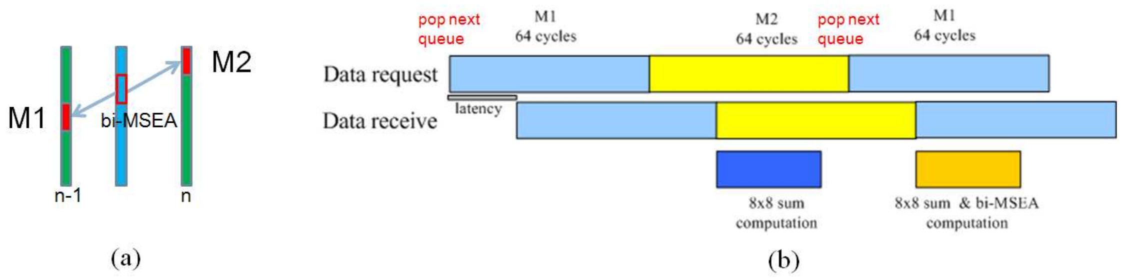


Figure 3.22 Bi-MSEA computing. (a) SRAM usage. (b) scheduling.

3.6.2 Parallelism determination

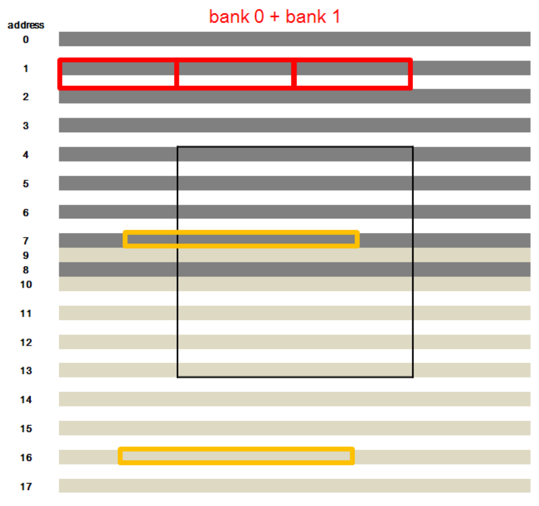
Since bilateral motion vector search performs like full search with lot of computation, we have to determine the parallelism of this procedure. After all of the above operations, the cycles left for labeled sub-blocks are show in Table 3.2 by simulations. In the worst case, there are 1064 cycles left for bilateral motion vector search and overlapped block motion compensation on one labeled sub-block. In the end we decide the throughput of bilateral search to be 16 absolute differences per cycle. And the search range changes from ±8x±8 to even point in ±8x±8. By this parallelism, the number of cycle consumed for bilateral search = ((8 + 8) / 2 + 1)^2 x (64 / 16) x 2 = 648 cycles per sub-block. Since the least cycles for overlapped block motion compensation are 32 x 32 x 1.5 x (3 + 1) / 16 = 384 cycles, 384 + 648 = 1032 cycles which is close to our target. By the way, we can reach the throughput by the existing SAD tree, without additional hardware cost.

Table 3.2 Cycles left for labeled sub-blocks.

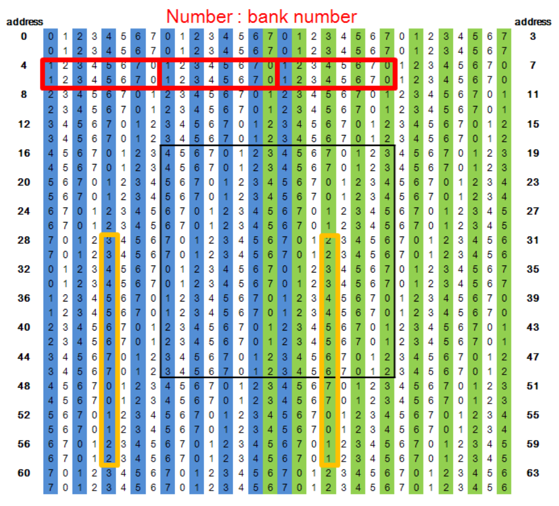
(all of the worst cases)	cycle left (60Hz)	cycle left (24Hz)	# of sub-block (24 Hz)	cycle left for one sub-block (24 Hz)
park_joy	1126796	4462482	2474	1804
ducks_take_off	1219792	4718998	1340	3522
vintagecar	1049260	4258462	3074	1385
tractor	1191664	4641346	1969	2357
pedestrian_area	1115964	4410934	2787	1583
transformer 7-3	1048000	4199558	3947	1064
Titanic-2	1039672	4246090	1820	2333

3.6.3 On-chip SRAM interleave for Bilateral Motion Vector Search

We save boundary pixels in SRAM for boundary error computing. The horizontal pixel lines are saved in M1 (M2) and vertical pixel lines are saved in O1 (O2). Figure 3.23 illustrate the specified pixel interleaves for boundary pixel windows of original motion vector direction (the opposite direction is in similar manner). By the interleaves, we can save 8x2 pixels into SRAM of different banks (red hollow rectangles in Figure 3.23) and access any line of boundary (yellow hollow rectangles in Figure 3.23) in a cycle. The data pre-fetching of SRAM is also in ping-pong manner.



(a)



(b)

Figure 3.23 The pixel interleaves for boundary error computing. (a) horizontal lines in M1 and M2. (b) vertical lines in O1 and O2.

3.6.4 On-chip SRAM interleave for Overlapped Block Motion Compensation

For this operation, we save pixels of center weighting map in M1 (M2) and of other four weighting map in O1 (O2). When performing compensation, we read 16x1 pixel lines (blue hollow rectangles in Figure 3.24) of weighting maps at a time; multiply them by corresponding weight, and fuse multiplied 16x1 pixels lines then write out. The 16x1 weights of center, up and down map change by time, so we need multipliers for them. The 16x1 weights of left and right map are constant, so we just use constant multipliers for them. The data pre-fetching of SRAM is also in ping-pong manner.

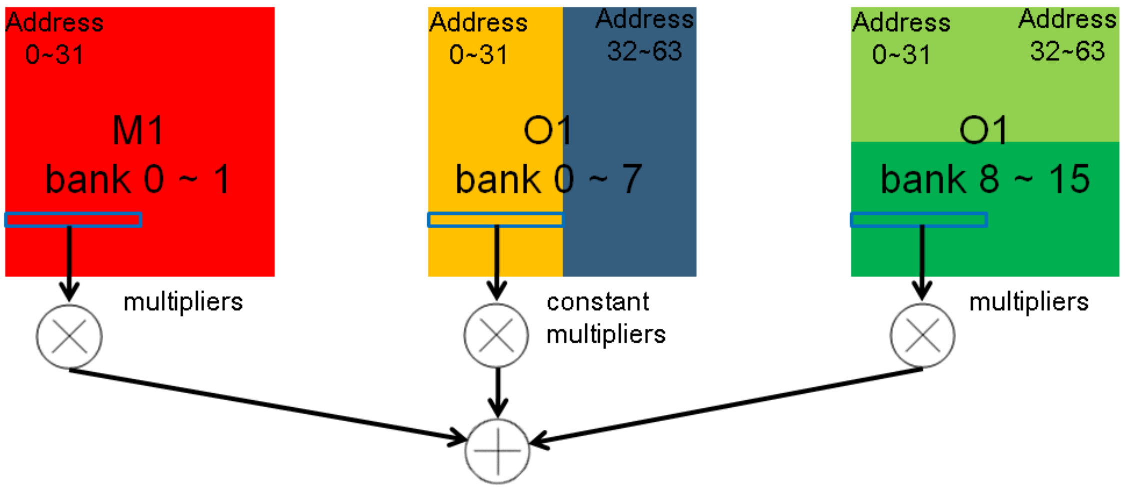


Figure 3.24 Illustration of OBMC.

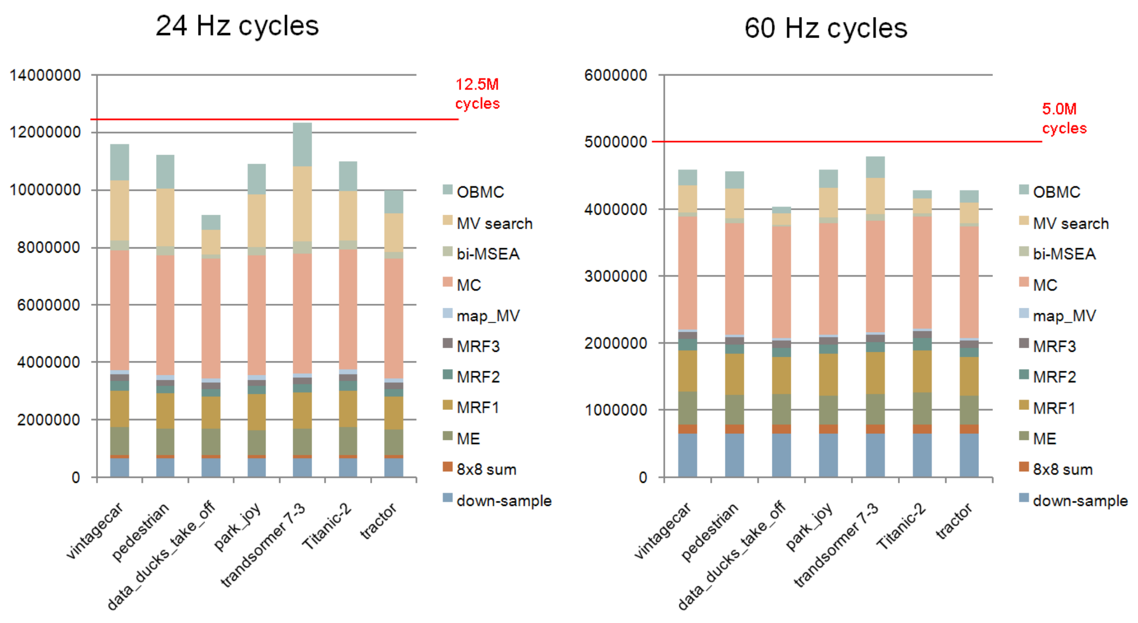
3.6.5 Summary

For post-processing, we create queues in DRAM for pushing and popping sub-blocks' information. By parallelism analysis, we can cope with at least 4069 sub-blocks and 1015 sub-blocks for 24Hz to 120Hz and 60Hz to 120Hz up-conversion without additional hardware cost for computation. With the specified pixel interleaves in SRAM, we only read and write the required pixels for minimum amount of bandwidth consumption. The schedules are tight by SRAM ping-pong usage and next queue pre-popping.

3.7 Implementation Results

3.7.1 Cycles and Bandwidth Consumption

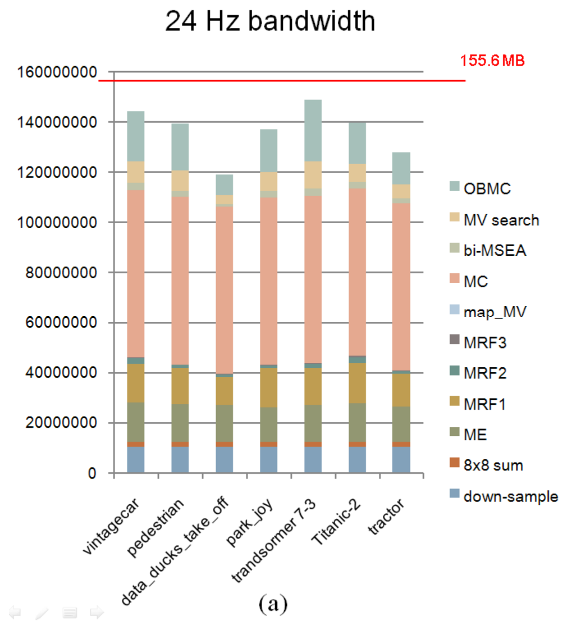
Figure 3.25 shows the simulation results of cycle consumption. All of the sequences' consumptions are lower the resource available (since we apply parallelism analysis). For 24Hz to 120Hz up-conversion, the cycles are depleted mainly by MC and MV search. For 60Hz to 120Hz up-conversion, the cycles are depleted mainly by MC and MRFx3.



(a) (b)

Figure 3.25 Total cycles. (a) 24Hz to 120Hz. (b) 60Hz to 120 Hz.

Figure 3.26 shows the simulation results of bandwidth consumption. All of the sequences' consumptions are lower the resource available except 60Hz to 120Hz up-conversion of transformer 7-3. It is acceptable since all the sequences for simulation are 24Hz with stronger motions than 60Hz, which take more resource for many operations. If the sequences are really 60Hz, it will take less bandwidth (and cycles) for 60Hz to 120Hz up-conversion than our simulation results. Near a half of bandwidth is depleted by MC where OBMC consumes the second amount of bandwidth for 24Hz to 120Hz and down-sample consumes the second amount for 60Hz to 120Hz



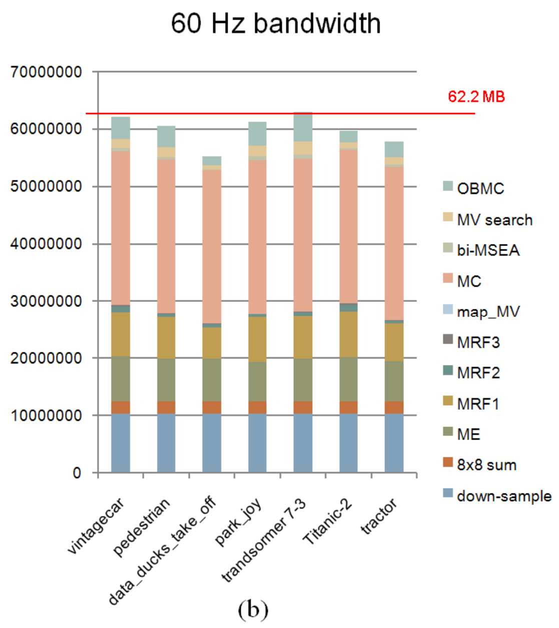


Figure 3.26 Total bandwidth. (a) 24Hz to 120Hz. (b) 60Hz to 120 Hz.

3.7.2 Hardware Specification and Efficiency

Table 3.3 shows the final implementation results. We use Verilog-HDL for hardware implementation, synthesize it by SYNOPSIS Design Compiler with UMC 90nm cell library. The total number of gate count is 274K, on-chip SRAM is single-port 9984 byte, working at 300MHz frequency on 128 bits bus. It provides 24Hz to 120Hz and 60Hz to 120Hz multi-rate up-conversion with ±128x±128 search range and supports 3840x2160 resolution for next LCD generation. The differences between proposed algorithm and architecture are two-way scan order and ±8x±8 on even points for motion vector refinement

Table 3.3 Specification of implementation.

Design Specification	
Technology	UMC 90nm
Clock rate	300MHz
Bus width	128 bits/cycle
DRAM	DDR3-1333
Gate count with SRAM	537652
Gate count without SRAM	273845
SRAM size	9984 Bytes single port

Capability Specification	
FRUC mode	24 Hz -> 120 Hz 60 Hz -> 120 Hz
Frame size	3840x2160
Search range	±128 x ±128

For hardware efficiency evaluation, we also compare to Percept. [12] and GME [17]. The specifications are listed in Table 3.4 and we normalize the gate count by regard all SRAM as single port with 3.3 gate count per bit. We provide four times of resolution and additional 24Hz to 120Hz up-conversion mode to other two implementations. The hardware efficiency comparison is shown in Figure 3.27. As the slope of our design is the smallest, the proposed architecture has the best hardware efficiency of all.

Table 3.4 Normalized specifications of the references.

*Assume single port SRAM with 3.3 gate count / bit

	Percept.	GME	Ours
Technology	UMC 90nm	UMC 90nm	UMC 90nm
Clock rate	200MHz	133MHz	300MHz
Gate count with SRAM	292732*	1627900*	537652
Gate count without SRAM	212582	1301464	273845
SRAM size (Byte)	3036*	12365*	9984
FRUC mode	60Hz -> 120Hz	60Hz -> 120Hz	24Hz -> 120Hz 60Hz -> 120Hz
Frame size	1920x1080	1920x1080	3840x2160

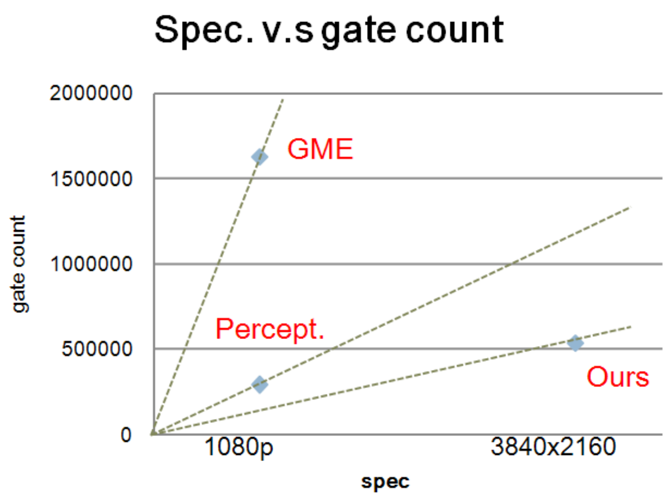


Figure 3.27 Hardware efficiency comparison.

3.8 Summary of Architecture Design

In this chapter, the system of LCD, specification of our design target, DRAM selection and resource available are introduced at first. By analysis, there are many issues for architecture design to be concerned.

We propose a ping-pong two-way scheduling for motion estimation to eliminate the dependencies of blocks and pipeline bubbles. By only reading the pixels needed for 4, 2, 1-step convergence and 8x8 sum for 8-step, we reduce a huge amount of bandwidth and SRAM size. The sum-trees of flexible adders and accumulators consume small amount of adders and shared by other modules.

For Marcov random field correction, a motion vector grouping algorithm is proposed for bandwidth and cycles reduction. The corresponding architecture is also used for discontinuity energy generation and median filter. For MRF iteration 1 and iteration 2, we employ the computed results to further reduce the resource consumption. The SRAM and computation unit are shared here with motion estimation.

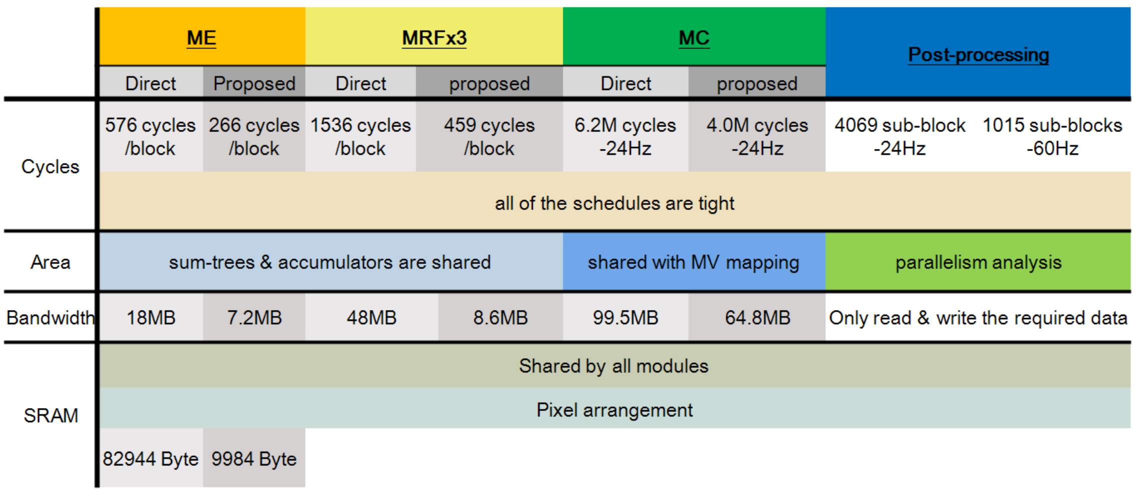
Multi-frame motion compensation is seldom discussed in hardware. For reaching the minimum resource requirement, we propose an inverse motion compensation scheduling. The proposed architecture is also responsible for block-based through motion vector mapping. For writing 8x2 pixels at a time, we provide a pixel interleaves on SRAM.

We create queues in DRAM saving sub-blocks' information for post-processing. By parallelism analysis, we can deal with the worst case of sub-blocks without additional hardware cost for computation. There are specified pixel interleaves of bilateral motion vector search and overlapped block motion compensation for minimum amount of bandwidth consumption.

The simulation results show that the cycles and bandwidth consumed are under the upper bound of resource available. The final implementation results indicate that our design only consumes 274K gate count and 10K byte single port SRAM and supports 24Hz to 120Hz and 60Hz to 120Hz up-conversion for 4Kx2K resolution. Comparison to other design, the proposed architecture has the best hardware efficiency.

Table 3.5 shows the overall resource saving and characteristics of proposed architecture. Lots of cycles and bandwidth are saved by our hardware design of each part. With the help of ping-pong two-way scheduling, all of the schedules are tight. Sum-trees and accumulators are shared by ME and MRF (also shared by down-sample, 8x8 sum, bi-MSEA and boundary error). MC shares the architecture with MV mapping, and Post-processing consumes no additional area overhead by parallelism analysis. The SRAM is also shared by all of the modules, with different pixel arrangements for applications.

Table 3.5 Overall resource saving and characteristics of the architecture.



	ME		MRFx3		MC		Post-processing	
	Direct	Proposed	Direct	proposed	Direct	proposed		
Cycles	576 cycles /block	266 cycles /block	1536 cycles /block	459 cycles /block	6.2M cycles -24Hz	4.0M cycles -24Hz	4069 sub-block -24Hz	1015 sub-blocks -60Hz
	all of the schedules are tight							
Area	sum-trees & accumulators are shared				shared with MV mapping		parallelism analysis	
Bandwidth	18MB	7.2MB	48MB	8.6MB	99.5MB	64.8MB	Only read & write the required data	
SRAM	Shared by all modules							
	Pixel arrangement							
	82944 Byte	9984 Byte						

Chapter 4 Conclusion and Future Work

In this thesis, we introduce the motion blur problem on LCD, and the general steps of frame rate up-conversion technique. For our design motivation and target, we develop an algorithm and architecture implementation for 24Hz to 120Hz and 60Hz to 120Hz up-conversion with resolution 3840x2160.

For the FRUC algorithm design, we propose a true motion based predictive square search algorithm for motion estimation with 32x32 block size, 8x8 MSEA criterion and ±128x±128 search range. The experiments show that there are at least 60% blocks converge at predictor, thus the algorithm is very low-cost. For motion vector processing, we apply Marcov random field modeling and minimize the energy by low-cost ICM. The low-cost ICM is true motion based, reducing energy computation from 65536 to 9 and preventing over-smoothing. The general types of MV mapping are not perfect, so we use block-based through motion compensation better than the three general types. A precise artifact detection technique is provided with only 12% of sub-blocks labeled. In post-processing, bilateral search considers occlusion, boundary error criterion finds motion vectors with the least block artifact, and OBMC blur the necessary region with block artifact.

For the proposed architecture, ping-pong two-way scheduling eliminates the dependencies of blocks for data pre-fetching. The pipeline bubbles are therefore dissolved. The careful arrangement of SRAM reduces the size from 90K byte to 10K byte. The sum-trees with flexible adder and accumulators consume only 85 adders and shared by lot of modules. MV grouping for MRF correction reduces cycles from 1536 to 459 cycles per block and bandwidth from 48MB to 8.6MB for three iterations. Its architecture is shared for MRF energy computing and median filter. The inverse-MC scheduling requires near minimum amount of resource, reduces cycles from 6.2M to 4.0M cycles and bandwidth from 99.5MB to 64.8MB. The architecture is also shared by MV mapping. In post-processing, parallelism analysis is performed for minimum cost of bilateral search. There are specified pixel interleave for boundary error computing and OBMC.

By subjective evaluation, above 79% subjects vote proposed algorithm as the best choice. By objective evaluation, there is 0.63 to 5.47 PSNR gain to other algorithms.

Simulation results indicate that the proposed architecture consumes reasonable amount of cycles and bandwidth. The final implementation results show that our design only consumes 274K gate count and 10K byte single port SRAM. By supporting 24Hz to 120Hz and 60Hz to 120Hz up-conversion for 4Kx2K resolution, our implementation has the best hardware efficiency comparing to previous works.

For the possible future works, we list in the following:

- There may be a criterion for scene change detection, for example: threshold of MRF energy. The scene change detection is important since we don’t want to interpolate frames between two non- successive frames.
- The block size can be smaller by repeating similar operation of proposed sub-block division on labeled sub-blocks.
- Taking the perceptual criterion into account for motion vector refinement in post-processing may be a good way for enhancing the visual quality.
- Among all the FRUC discussions, the FRUC algorithms are operating step by step. There is no modeling-based FRUC framework. If we can model the inter-frames well, then finding the motions, occlusion labels and pixels value of inter-frames such that the modeling energy is minimized is equal to interpolate the inter-frames. The modeling-based FRUC is more theoretical and can be pixel-based, and surely have more complexity.

References

[1] A. -M. Hung and T. Nguyen, “A Novel Multi-Stage Motion Vector Processing Method For Motion Compensated Frame Interpolation,” in Proc. of IEEE Int. Conf. on Image Processing, vol. 5, pp. V389-V392, Sept. 2007.

[2] A. K. Michiel, “Temporal Impulse Response And Bandwidth of Display in Relation to Motion Blur,” in SID Symp. Digest, vol. 36, pp. 1578-1581, May 2005.

[3] H. Pan, X. Feng, S. Daly, “Quantitative analysis of LCD motion blur,” and performance of existing approaches, in SID Symp. Digest, 2005, pp. 1590–1593.

[4] H. Pan, X. Feng, S. Daly, “LCD Motion Blur Modeling and Analysis,” ICIP 2005

[5] G. de Haan, J. Kettenis, B. de Loore, “IC for Motion-Compensated 100 Hz TV with Smooth-Motion Movie-mode,” ICCE 1995

[6] Orchard, M.T., Sullivan, G.J., "Overlapped block motion compensation: an estimation-theoretic approach", IEEE Transactions on Image Processing, page 693 - 699, Sep 1994

[7] de Haan, G.; Biezen, P.W.A.C.; Huijgen, H.; Ojo, O.A.; "True-motion estimation with 3-D recursive search block matching", IEEE Transactions on Circuits and Systems for Video Technology, page 368 - 379, 388, Oct 1993

[8] Jisheng Wang, Dong Wang, Wenjun Zhang, "Temporal compensated motion estimation with simple block-based prediction," IEEE Transactions on Broadcasting, page 241 - 248, Sept. 2003

[9] Tourapis, Alexis M., "Enhanced predictive zonal search for single and multiple frame motion estimation", Visual Communications and Image Processing , page 1069-1079, Feb. 2002

[10] Simon Baker, Daniel Scharstein, J.P. Lewis, "A Database and Evaluation Methodology for Optical Flow," ICCV 2007

[11] Ya-Ting Yang, Yi-Shin Tung, Ja-Ling Wu, “Quality Enhancement of Frame Rate Up-Converted Video by Adaptive Frame Skip and Reliable Motion Extraction,” IEEE Transactions on Circuits and Systems for Video Technology, Dec. 2007, pp.1700 – 1713

[12] Ya-Ting Wang, Shao-Yi Chien, “Algorithm and Hardware Architecture Design of Perception-Aware Motion Compensated Frame Rate Up-Conversion,” Master thesis of GIEE, NTU, Feb. 2010

[13] Ai-Mei Huang, Truong Q. Nguyen, "A Multistage Motion Vector Processing Method for Motion-Compensated Frame Interpolation," IEEE Transactions on Image Processing, VOL. 17, NO. 5, MAY 2008

[14] Y. Neuvo J. Astola, P. Haavisto, “Vector median filters,” in Proc. IEEE, pp. 678–689., Apr.1990

[15] Dane, G.; Nguyen, T.Q.; "Smooth motion vector resampling for standard compatible video post-processing,", ACSSC 2004

[16] Demin Wang; Liang Zhang; Vincent, A.; "Motion-Compensated Frame Rate Up-Conversion—Part I: Fast Multi-Frame Motion Estimation," IEEE Transactions on Broadcasting, page 133 - 141, June 2010

[17] Hsu Kung-Yen, Shao-Yi Chien, “Frame Rate Up-Conversion Algorithm and Hardware Architecture Design for High Definition Liquid Crystal Display,” Master thesis of GIEE, NTU, Nov. 2008

[18] Byung-Tae Choi; Sung-Hee Lee; Sung-Jea Ko; "New frame rate up-conversion using bi-directional motion estimation," IEEE Transactions on Consumer Electronics, page 603 - 609, Aug. 2000

[19] Byeong-Doo Choi; Jong-Woo Han; Chang-Su Kim; Sung-Jea Ko; "Motion-Compensated Frame Interpolation Using Bilateral Motion Estimation and Adaptive Overlapped Block Motion Compensation," IEEE Transactions on Circuits and Systems for Video Technology, page 407 - 416, April 2007

[20] Demin Wang; Vincent, A.; Blanchfield, P.; Klepko, R.; "Motion-Compensated Frame Rate Up-Conversion—Part II: New Algorithms for Frame Interpolation," IEEE Transactions on Broadcasting, page 142 - 149, June 2010

[21] Sugiyama, K.; Aoki, T.; Hangai, S.; "Motion compensated frame rate conversion using normalized motion estimation," Signal Processing Systems Design and Implementation, IEEE Workshop, page 663 - 668, Nov. 2005

[22] Erwin B. Bellers and Johan G.W.M. Janssen, "An Evolutionary Architecture for Motion-Compensated 100 Hz Television," IEEE Transactions on Circuits and Systems for Video Technology, Vol. 5, No. 3., June 1995

[23] Yang Ling; Jin Wang; Yunqiang Liu; Wenjun Zhang; "A novel spatial and temporal correlation integrated based motion-compensated interpolation for frame rate up-conversion," IEEE Transactions on Consumer Electronics, page 863-869, May 2008

[24] Alexis M. Tourapis, Oscar C. Au, Ming L. Liou, “Predictive motion vector field adaptive search technique (PMVFAST): enhancing block-based motion estimation,” Visual Communications and Image Processing 2001, pp.883-892

[25] Huska, J.; Kulla, P.; “A New Recursive Search with Multi Stage Approach for Fast Block Based True Motion Estimation,” International Conference Radioelektronika 2007

[26] Gao, X.Q.; Duanmu, C.J.; Zou, C.R.; "A multilevel successive elimination algorithm for block matching motion estimation," IEEE Transactions on Image Processing, page 501 - 504, Mar. 2000

[27] S. Z. Li, "Markov random field models in computer vision," ECCV 1994

[28] Keng Pang Lim, Das, A., Man Nang Chong, “Estimation of occlusion and dense motion fields in a bidirectional Bayesian framework,” IEEE Transactions on Pattern Analysis and Machine Intelligence, May 2002, pp. 712-718

[29] Michael R. Garey, David S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," Series of Books in the Mathematical Sciences, 15 January 1979

[30] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother "A Comparative Study of Energy Minimization Methods for Markov Random Fields," ECCV 2006

[31] Yung-Lin Huang, Yi-Nung Liu, and Shao-Yi Chien, "MRF-based True Motion Estimation Using H.264 Decoding Information," SiPS 2010

[32] Zhang, J.; Arnold, J.F.; Frater, M.R.; "A cell-loss concealment technique for MPEG-2 coded video," IEEE Transactions on Circuits and Systems for Video Technology, page 659 - 665, June 2000

[33] SYMBIAN developer, ”Symbian OS Internals /2. Hardware for Symbian OS,” http://developer.symbian.org/wiki/Symbian_OS_Internals/2._Hardware_for_Symbian_OS

[34] NXP Semiconductors , “NXP 5100 FRUC solution,” http://www.nxp.com/

[35] Micron Technology, Inc., “DDR3 SDRAM Part Catalog,” http://www.micron.com/products/dram/ddr3_sdram.html

[36] DRAMeXchange, “DRAM Spot Price,” http://www.dramexchange.com/