國 立 交 通 大 學

電子工程學系　電子研究所碩士班

碩 士 論 文

適用於高畫質視訊之移動估測設計

Hardware Efficient Motion Estimation Designs for High Definition Video Compression

研究生：林嘉俊

指導教授：張添烜

中華民國　九十六年　七月


E
S
A
1896

適用於高畫質視訊之移動估測設計

Hardware Efficient Motion Estimation Designs for High Definition Video Compression

研 究 生：林嘉俊　　Student: Chia-Chun Lin
指導教授：張添烜　博士　　Advisor: Tian-Sheuan Chang

國　立　交　通　大　學
電子工程學系　電子研究所碩士班
碩　士　論　文

A Thesis
Submitted to Department of Electronics Engineering & Institute of Electronics
College of Electrical Engineering and Computer Science
National Chiao Tung University
in Partial Fulfillment of Requirements
for the Degree of
Master of Science
In
Electrical Engineering
July 2007
Hsinchu, Taiwan, Republic of China

中華民國　九十六年　七月


E
S
A
1896

適用於高畫質視訊之移動估測設計

研究生：林嘉俊　　指導教授：張添烜 博士

國立交通大學電子研究所碩士班

摘　要

移動估測在視訊編碼的過程中，具有非常高的複雜度，因此而成為即時影像編碼的瓶頸。為了克服移動估測的諸多困難性，包含：龐大的運算量，高面積成本以及大量的記憶體頻寬，我們將提出許多演算法以及其相對應的硬體架構來解決以上種種問題。首先，我們提出一個高效能低成本適應性的跳躍方塊預測演算法及架構，藉由預測可跳躍的靜態影像而降低運算量。其次，我們提出一個快速的模式決定演算法來切割以及平衡整數點移動估測以及非整數點移動估測之間所需花費之時間，藉此來增進整體硬體架構的平行度以及效能。此外，我們針對不同的應用層面來設計不同的演算法及相對應架構來適應其使用環境。對於小畫面的可攜性元件，我們提出一個有效率低成本低耗能的調適性四分之一像素快速移動估測設計。另一方面，對於大畫面高解析之視訊應用，我們提供另一個高效率平行化之多解析度移動估測設計來支援大搜尋範圍，而提供高品質低位元率之效能。最後，我們將上述方法整合到一顆支援每秒 30 張之 1080p 畫面的高規範視訊晶片，製作一個完整的視訊壓縮晶片。

Hardware Efficient Motion Estimation Designs for High Definition Video Compression

Student: Chia-Chun Lin Advisor: Tian-Sheuan Chang

Institute of Electronics

National Chiao Tung University

Abstract

Motion estimation (ME) processing is the most complex part and the bottle neck of a real time video encoder due to its heavy complexity, high area cost, and large memory bandwidth. In this thesis, we propose fast algorithms and architectures to solve these issues. For the fast algorithms, first, we introduce a low cost adaptive skip mode detection algorithm and its architecture to encode the static portion of video in an efficient way. Second, a fast mode decision algorithm is presented to save hardware computing cycles by separating the integer-pixel ME and fractional-pixel ME phase. In the architecture designs, we propose two different ME designs for portable and high definition applications. For portable small size video gadgets, we propose low cost and low power refined quarter motion estimation hardware to solve the cost problem. For large frame size high definition video, we use parallel multi-resolution motion estimation to offer large search region. Finally, we integrate these methods into a high profile encoder chip which supports 1080p video under 145MHz.

誌　謝

首先，要感謝我的指導教授—張添烜博士，這兩年來給我的支持和鼓勵，帶領我由淺入深一步一腳印的學習與研究，也讓我在想法上能自由發揮，發揮我最大的潛能與創意，而每當遇到問題和疑問時能夠給予我建議與協助。因此，我對與張教授的感激之情溢於言表。謝謝我的口試委員們，交大電子李鎮宜系主任和清華大學陳永昌教授，感謝你們百忙中抽空來指導我，因為你們寶貴的意見讓我的論文更加完備。

感謝 VSP 實驗室的好伙伴們，特別要謝謝引我入門的林佑昆學長，帶領我從零開始，一點一滴紮實的研究與實作，也給予我不少中肯有用的建議。感謝張彥中學長、李國龍學長，你們傳給我的經驗與知識，讓我受用不盡。謝謝古君偉和王裕仁學長教導我許多 IC 設計的觀念與技巧，也感謝廖英澤同學，陪我連續參加 IC 競賽，一起切磋晶片製作的實力。感謝李得瑋、郭子筠、吳秈璟同學，跟你們一起整合研究一顆完整的編碼晶片，是一個難得的過程，在我們一同解決問題的過程中，我學習到許多，這也是一段刻苦銘心的回憶。感謝蔡宗憲、曾宇晟、詹景竹、張瑋城、戴瑋呈學弟們，有你們的陪伴，我的碩士班生涯充滿了歡笑。謝謝實驗室的所有成員們，和你們一同奮鬥、流汗與歡樂的過程，都是我在交大寶貴的回憶。

謝謝我的女友，謝謝你不斷的支持與鼓勵，也讓我對未來的學習之路有了全新的轉變。也感謝社團朋友的支持，跟你們一同出遊爬山是我減輕壓力的最好方式，再創我精神與創意的高峰。

最後要感謝默默支持我的家人們，我的爸媽、弟弟，你們的溫暖是我努力最大的支柱。

在此，把本論文獻給所有愛我與所有我愛的人。

Contents

List of Figure

List of Table

1. Introduction

1.1. Motivation

The emerging multimedia technology such as digital television, mobile phone and DVD play indispensible roles in our daily life. These products become our main way to acquire information from the world, to communicate to each other and to entertain ourselves. However, the multimedia information is too large to transmit or record without effective compress them. Therefore, the issue of how to effectively compress the data becomes an important part of multimedia research nowadays. In short, video compression is a technology to transform video signals and try to maintain original quality under a number of constraints such as storage constraint, real time constraint or computation power constraint. It needs effectively exploiting the redundancy within or between frames to reduce the data rates with minimum video quality loss. Thus, the design of data compression systems normally involves a tradeoff between quality, speed, resource utilization and power consumption.

The compression technique in a video scene includes removing data redundancy of spatial, temporal and statistical correlation between frames. The main concept to remove such redundancy is because our human eye and brain (Human Visual System) are more sensitive to lower frequencies and thus enables us to diminish the information of higher frequency to decrease total bit rates. Thus, by removing different types of redundancy, it is possible to compress the data significantly at the expense of a certain amount of information loss (distortion) and further compression can be achieved by encoding the processed data using an entropy coding.

Within these compression techniques, motion estimation and motion compensation are widely used in video compression to reduce the temporal redundancy in video contents.

It is a very efficient and practical way to predict the motion of adjacent frame by using few bit rates; whereas, it occupies very high computational complexity in whole encoding process. Besides, in the recently standard—H.264/AVC, variable block size motion estimation (VBSME) consisting of integer ME (IME) and fractional ME (FME) is adapted to fit different details of video sequences. However, the long coding time and large power consumption of motion estimation becomes the main problem eager to solve in the encoder.

1.2. Contribution of the Thesis

Figure 1 shows main contributions of this thesis within the basic flow of motion estimation. In the mode decision phase, we detect the skip mode and do mode filtering to decrease the complexity of mode combination. Afterward, we estimate several the motion vectors in the integer motion estimation phase; however, the further refinement step for fractional motion vectors will not be included in the thesis. This thesis presents a number of integer motion estimation algorithms and its architectures for variable block size motion estimations. The following novel contributions result from this work.

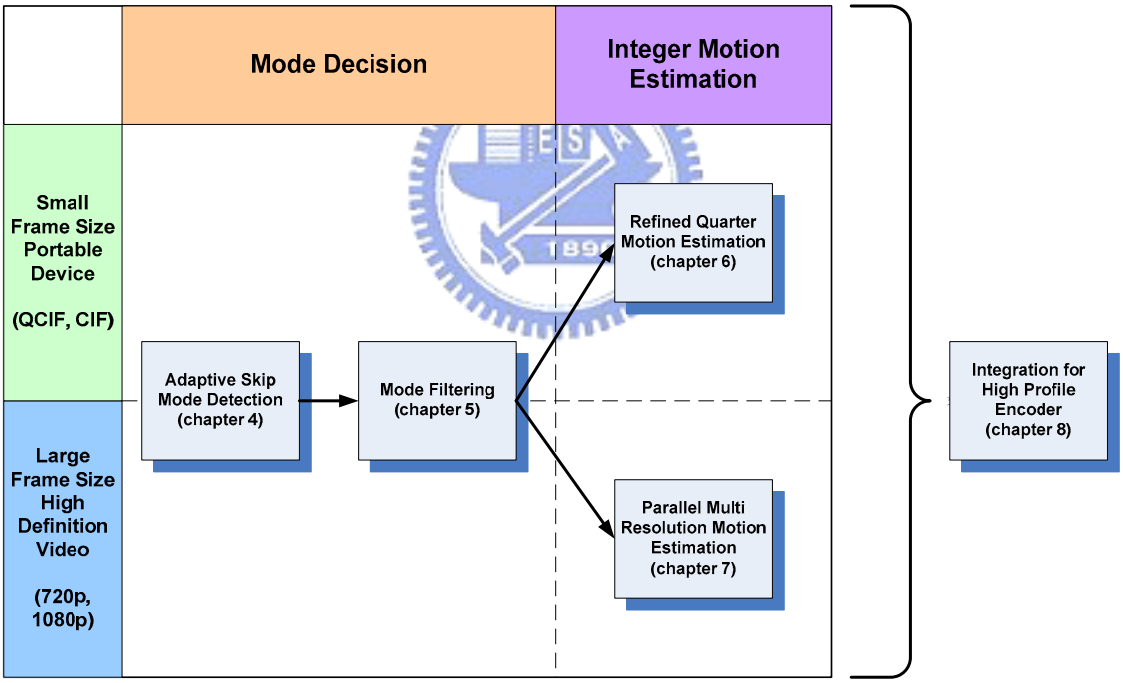

Figure 1 the contribution of the thesis

- ✓ Adaptive skip mode detection: we propose low cost adaptive skip detection hardware algorithm and its architecture to save the computation power for static macroblock. The hardware costs are 0.63K gate counts and small size memory for look-up table. In some low motion and highly quantized sequences, our method can accurately skip 82.39% macroblocks.

- ✓ Mode Filtering (MF): it is a fast algorithm that can speed up the overall motion estimation process with the reduction of fraction motion estimation modes. With different mechanism, the algorithm can be applied to 2-D array design or 1-D array designs. By using the algorithm, the hardware implementation can be pipelined with higher efficiency with slightly performance loss.

- ✓ Refined Quarter Motion Estimation (RQME): it uses the MF to reduce the computational load of fractional motion estimation and the quarter pixel method to enable four times of parallel processing with low computational complexity and low quality loss. Besides, the proposed hardware architecture only needs half the number of process elements and less latency than the general 2-D architectures. This design is very suitable for portable device which has the characters of low cost and small frame size.

- ✓ Parallel Multi Resolution Motion Estimation (PMRME): the method applies parallel multi resolution motion estimation, MF and bit truncation to support large search range [-128, 127] within 256 cycles for p-frame (one-direction) motion estimation. Because of the fast searching mechanism, the design also can support b-frame (bi-directional) motion estimation with only 512 cycles. In addition, this design can save at least 91.91% of memory buffer and 55.1% of bandwidth. The resulted hardware also save up to 48.9% of area cost and 62.1% of memory cost compared to previous approach for 1080P processing. With above features, the proposed design is suitable for larger search range application such as HDTV.

- ✓ A 1080p high profile encoder chip for H.264: we integrate our design into a high performance H.264 high profile encoder that can support 1080p resolution under 145MHz with smaller area. In this integration, we optimize the algorithm and architecture of the motion estimation component as well as the memory organization and pipeline schedule of the whole design to achieve a high throughput and low hardware cost design.

1.3. Organization of the Thesis

The main theme of the thesis is to study different implementation methods for motion estimation in the standard of H.264/AVC [1]. In chapter 2, we briefly introduce the background and basic tools of H.264/AVC standard. In chapter 3, we give an overview of the basic concepts and several algorithms of motion estimation. In chapter 4, we propose an adaptive skip mode detection to lower the computation overhead for static macroblocks. In chapter 5, we propose the algorithm of mode filtering and its performance to simplify the mode combination and enhance throughput of motion estimation. In chapter 6, we present the refined quarter motion estimation design to fit some low cost, small frame size application of H.264. In chapter 7, a large search range PMRME design is proposed to deal with the large frame size application. Then, in chapter 8, we integrate the method mentioned above to implement an encoder which supports 1080p high profile of H.264/AVC. Finally, a conclusion is given in chapter 9.

2. Overview of H.264/AVC Standard

2.1. Overview

Image and video compression has been a very active field of research and development for over twenty years. Many different systems and algorithms for compression and decompression have been proposed and developed. In order to achieve inter-working, industrial competition and possibility of popularity, it is necessary to define standard methods for decoding to allow products from different manufacturers to communicate to each other effectively. Therefore, the standardization process has contributed to the prevalence of broadcast television and home entertainment nowadays. Recently, the ISO (International Standard Organization) MPEG4 standard is enabling a new generation of internet-based video applications while the ITU-T (Telecommunication Standardization Sector) H.263 standard for video compression is now widely used in videoconference systems.

MPEG4 and H.263 are standards that are based on video compression technology start with about 1995. The two groups responsible for these standards: the one is Motion Picture Experts Group (MPEG) and the other is Video Coding Experts Group (VCEG), both of them are in the final stages of developing a new standard that promises to significantly outperform MPEG4 and H.263. It provides better compression of video images by properly adopting a variety of tools to supporting high-quality and low bit rate streaming video. In the VCEG side, after finishing the original H.263 standard 1995, the VCEG started work on two further development areas: a short-term effort to add extra features to H.263 and a long-term effort to develop a new standard for low bit rate visual communications. The long-term effort led to the draft “H.26L” standard, offering significantly better video compression efficiency than previous ITU-T standards. In 2001, the MPEG recognized the potential benefits of H.26L; therefore the Joint Video

Team (JVT) was formed, including experts from MPEG and VCEG. JVT's main task is to develop the draft H.26L model into a full international standard. In fact, the outcome will be two identical standards: ISO MPEG4 Part 10 of MPEG4 and ITU-T H.264. The official title of the new standard is Advanced Video Coding (AVC); however, it is widely known by its old working title, H.26L and by its ITU document number, H.264 [1].

H.264 consists of numerous of tools. Compared to the prior video coding standards, many important and new techniques are employed and bring significant improvement on coding performance. Some details of these techniques can be found in [2]. Here, we would like to give a brief introduction of the basic concepts of these tools, which have existed for some time but nicely tuned and well integrated together to form a good compression scheme in H.264.

2.2. Coding Structure

In common with earlier standards, the H.264 standard does not explicitly define a CODEC (encoder / decoder pair). Instead, the standard defines the syntax of an encoded video bit stream together with the method of decoding. Actually, a compliant encoder and decoder are likely to include the functional elements shown in Figure 2 and Figure 3; besides, the functions shown in these figures are likely to be necessary for compliance. In these figure, we can find that the decoder system is a part of the encoder, whereas there are a certain range for considerable variation in the structure.

In general, most of the video coding systems are based on the motion estimation and motion compensation mechanism along with some other tools to reduce the neighboring frame redundancy. The basic functional elements (prediction, transform, quantization, entropy encoding) are little different from previous standards (MPEG1, MPEG2, MPEG4, H.261, H.263, etc.).

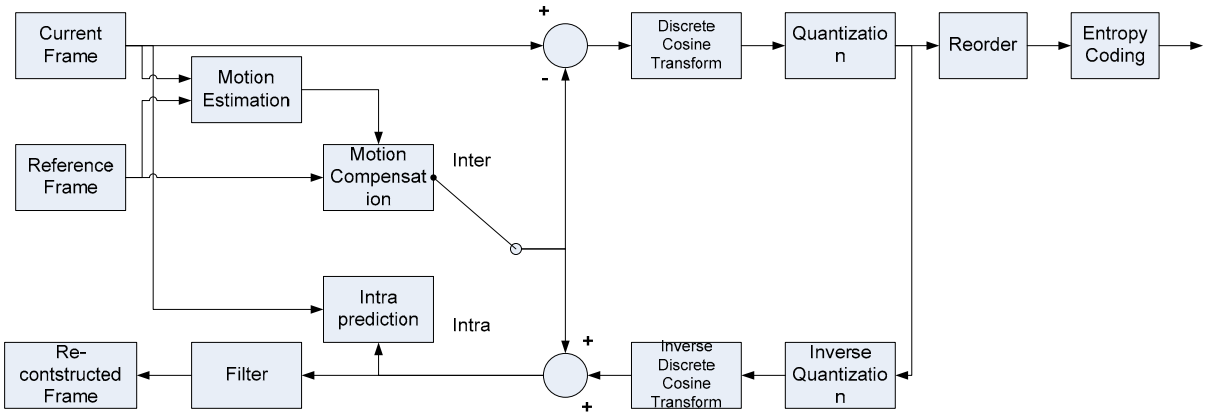

Figure 2 the basic structure of encoder

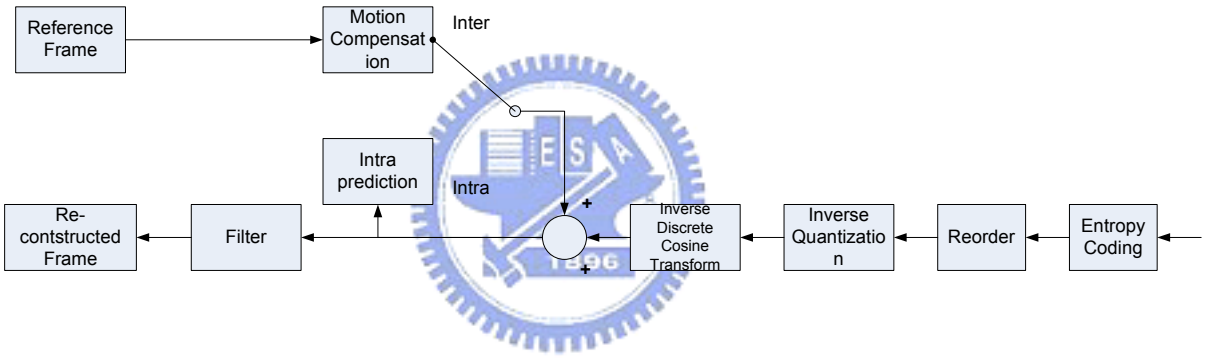

Figure 3 the basic structure of decoder

2.3. Intra Prediction

If a block or macroblock is encoded in intra mode, a prediction block is formed based on previously encoded and reconstructed blocks. This prediction block is subtracted from the current block prior to encoding. In H.264 [1], for the luminance (luma) block, it may be formed for each 4x4 subblock or for a 16x16 macroblock. There are a total of 9 optional prediction modes for each 4x4 luma block and 4 optional modes for a 16x16 luma block and one mode that is always applied to each 4x4 chrominance (chroma) block.

2.4. Inter Prediction

Inter prediction creates a prediction model from one or more previously encoded video frames. The model is formed by shifting samples in the reference frame(s) (motion compensated prediction). The AVC CODEC uses block-based motion compensation, the same principle adopted by every major coding standard since H.261. Important differences from earlier standards include that the H.264 supports for a variety of range of block sizes (down to 4x4) and fine sub-pixel motion vectors (1/4 pixel in the luma component).

2.5. In-loop Filter

In H.264, a filter is applied to every decoded macroblock in order to reduce blocking distortion caused by block-based transformation. In the encoder, the deblocking filter is applied after the inverse transform and before reconstructing and storing the macroblock for future predictions. In the decoder, it is applied before reconstructing and displaying the macroblock. The filter has two benefits: in the first place, block edges are smoothed, improving the appearance of decoded images, especially at higher compression ratios. In the second place, the filtered macroblock is used for motion-compensated prediction of further frames in the encoder, resulting in a smaller residual after prediction.

2.6. Context-based Adaptive Binary Arithmetic Coding (CABAC)

An arithmetic coding system is used to encode and decode H.264 syntax elements. The arithmetic coding scheme selected for H.264, Context-based adaptive binary arithmetic coding (CABAC) achieves good compression performance through for two reasons: first, it selects probability models for each syntax element according to the element's context. Second, it adapts probability estimates based on local statistics by using arithmetic coding.

3. Overview of Block Matching Motion Estimation

For video compression, consecutive frames in a video sequence can be regarded as a set of object appropriately displaced from frame to frame. If the motion trajectory of every object in current frame could be predicted from the previous frame, we only have to record and transmit the trajectory information to the decoder. In this way, we encode the trajectory of the object instead of the pixel information; thus we can diminish the required bits a lot for the video sequence. The trajectory of the object, which we call it motion vector (MV) is needed for decoder to do motion compensation to reconstruct the frame. The process of determining the motion vector in the encoder is called motion estimation (ME) and the maximum value of motion vector is determined by its search range

3.1. Block-based Motion Estimation

There are several ways to do motion estimation. One is object-based motion estimation that detects the outline of the object first and then estimates its motion vector [3]. The other way is block-based motion estimation, which is the most widely used motion estimation method for video coding since most of the pictures are normally rectangular in shape and block-division can be easily done. Besides, the block-base method can significantly reduce complexity compared with the object-based method.

In H.264 [1], the standard defines the standard block sizes for motion estimation. As illustrated in Figure 4 , one frame consists of several macroblocks (MB), which are “16 by 16” pixels square. In one macroblock, it can be divided into four “8 by 8” pixels 8x8 blocks, and within one 8x8 block, it can be further cut into four “4 by 4” pixels 4x4 blocks. The standard of H.264 defines several block type (mode) and its corresponding block size for motion estimation as listed in TABLE I and Figure 5**.**

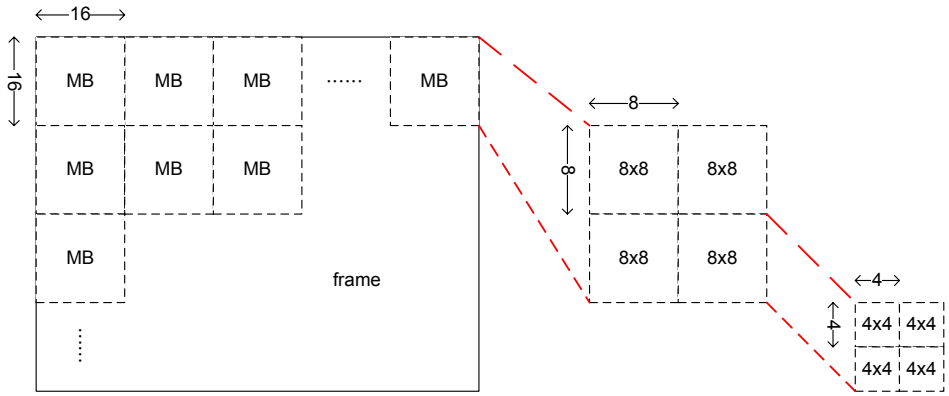

Figure 4 the hierarchy of a macroblock

TABLE I the mode type and its block size for H.264

Mode	Block size
Mode 1	16x16
Mode 2	16x8
Mode 3	8x16
Mode 4	8x8
Mode 5	8x4
Mode 6	4x8
Mode 7	4x4

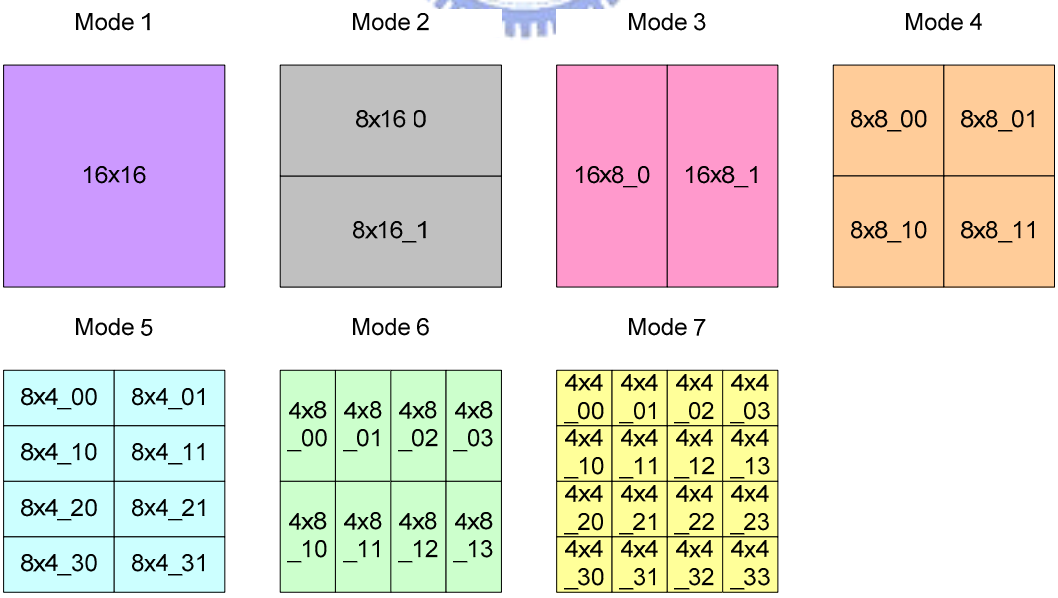

Figure 5 different modes and its block size

The goal of motion estimation is to accurately predict the motion vector inside the search range of previous frame. However, not only the MV but also the block size will determine the quality of prediction and accuracy. Figure 6 shows the relation of motion vector, search range and the distribution of block sizes within a picture. It is easy to see that the detailed region is associated with small blocks whereas the large uniform region is associated with large blocks. Hence, a macroblock can be composed by variable block size, and this method is known as variable block size motion estimation (VBSME).

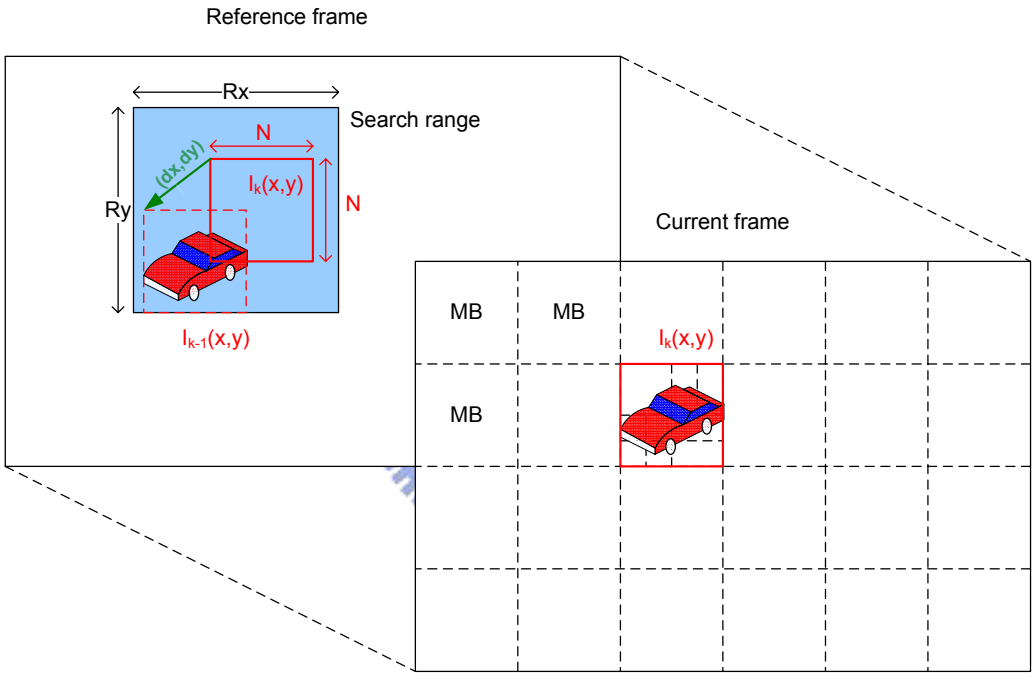

Figure 6 the motion vector and the search range

3.2. The Matching Criteria

Block-based motion estimation obtains the best match by minimizing a cost function. Although there are several cost functions [4], the common used criterion is sum of absolute difference (SAD). It is because of its low complexity, good performance and ease of hardware implementation. The cost function is defined as:

$$\mathrm{SAD}(\mathrm{dx},\mathrm{dy}) = \sum_{\mathrm{m=x}}^{\mathrm{x+N-1}} \sum_{\mathrm{n=y}}^{\mathrm{y+N-1}} |\mathrm{I_K}(\mathrm{m},\mathrm{n}) - \mathrm{I_{k-1}}(\mathrm{m}+\mathrm{dx},\mathrm{n}+\mathrm{dy})| + \lambda * \mathrm{R}\big((\mathrm{dx},\mathrm{dy}) - \mathrm{MVP}\big)$$

$$\overrightarrow{\mathrm{MV}} = (\mathrm{MVx},\mathrm{MVy}) = \min_{(\mathrm{dx},\mathrm{dy})\epsilon \mathrm{A}} \mathrm{SAD}(dx, dy)$$

$I_k(x, y)$: pixel intensity at location (x, y) in k-th frame (current frame)

$I_{k-1}(x, y)$: pixel intensity at location (x, y) in k-1-th frame (reference frame)

λ: Lagrangian multiplier

MVP: the predicted motion vector

R: number of bits to code the motion vector difference (MVD) = (dx, dy)-MVP

A: the region of search range

MV: motion vector

The former term of the function means the residual cost of the search point, and the latter term is the cost of the motion vector difference (MVD). We can find that the Lagrangian multiplier will influence the weighting of the motion vector cost. Therefore, when the Lagrangian is larger, the motion estimation mechanism will prone to choose larger block type because less motion vectors are needed and vice versa. We designers need to make balance between these two costs. Since the introduction of variable block size motion estimation in H.264/AVC, one macroblock can produce more than one motion vector due to the existence of different kinds of blocks. In H.264, 41 motion vectors and their corresponding costs should be produced in one macroblock to choose the best combination and this is known as mode selection.

3.3. Quality Judgment

The quality of a video sequence can be determined by using both objective and subjective approaches. The most widely used objective measure is the peak-signal-to-noise-ratio (PSNR) which is defined as:

$$\mathrm{PSNR} = 10 \log_{10} \frac{255^2}{\mathrm{MSE}}$$

MES: the mean-square-error of decoded frame and original frame

The peak value is 255 since the pixel value is 8 bits in depth (0~255). The higher the PSNR means the higher the quality of video. Although PSNR can objectively represent the quality of coding, it does not equal the subjective quality. Subjective quality is determined by a number of human testers and a conclusion is drawn based on their opinions. In some cases high PSNR results in low subjective quality. However, in most cases, PSNR provides a good approximation to the subjective measure and we use this measure in the rest of the thesis.

The PSNR and bit-rate are usually conflicting. According to the rate-distortion theory, low bit-rate always accompanies low quality (larger distortion) and vice versa.

3.4. Review of Motion Estimation Algorithm

3.4.1. Full Search Algorithm (FSA)

It is conspicuous that the most accurate strategy to find best motion vector is the full search algorithm (FSA) which exhaustively searches all possible search points within a predetermined search range to find the best motion vector. Although this method has heavy burden computation, the method has the characteristic of regular search flow and this feature enables it very suitable for hardware implementation. The data of the search range can be fully reused, and it really diminishes the enormous memory required during motion estimation. In addition, the computation of FSA can be decreased by using some technique to predict the skip macroblock.

3.4.2. Three Steps Search Algorithm (3SS)

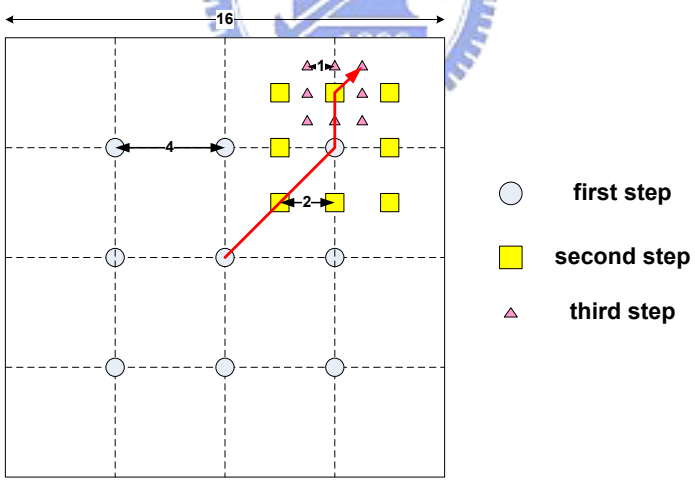

Figure 7 the search steps of three steps search algorithm

A representative work of the fast search algorithm is three-step search (3SS) algorithm [5]. 3SS is widely used because of its good performance and simplicity. It relies on a monotonically increasing match criterion around the location of the optimal motion vector to iteratively determine that location. Figure 7 shows an example to illustrate the

3SS algorithm. For search range equals 16x16, 3SS requires 25 (9+8+8) search points per macroblock, leading to a speedup of 9 when compared with 225 search points per macroblock for the FSA algorithm. However, the main disadvantage of 3SS is that it is inefficiency to estimate small motions, since the points forming the search pattern in the first step are positioned uniformly at relatively large distance around the center of the search window. Nevertheless, most of the motions in real world have a center-biased motion vector distribution [6]. In addition, in view of hardware design, we have to consider the branch condition of 3SS, which will cause the bauble effect in a pipelined hardware design and lower the average throughput. Therefore, we recommend using FSA in hardware design to enhance our performance. On the other side, in order to reduce the computation power (which is also the main concern in hardware design), we would like to use hardware oriented skip algorithm to predict the skip mode to lower the computation power and main high quality performance.

3.4.3. Quarter Pixel Motion Estimation (QME)

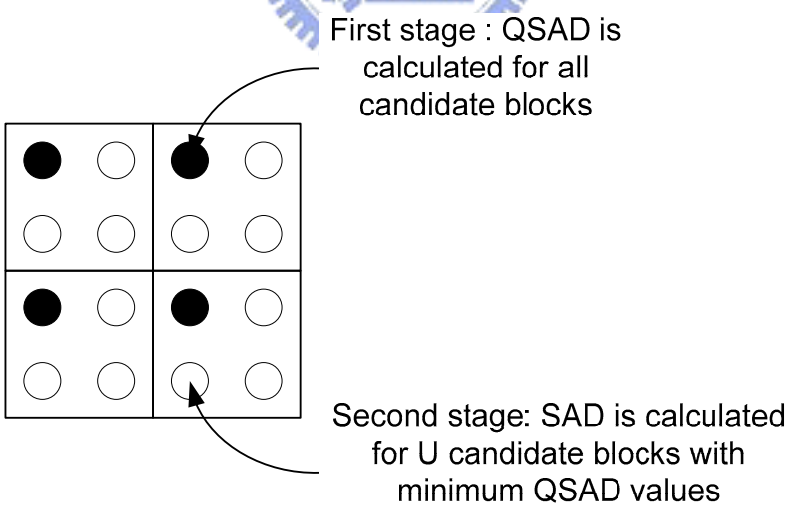

Figure 8 the QME algorithm

The Quarter Motion Estimation (QME) [7] is an algorithm of our previous design. As in Figure 8, the SAD is the sum of absolute difference as mentioned earlier, whereas the

QSAD is quarter pixel (the black dot) absolute difference. In [7], the algorithm can be divided into two stages. The first stage is to perform full search with QSAD as the matching criterion, and keep U candidates that have smallest QSAD values. The second stage is to calculate the SAD values for these U candidates and select the candidate which has the smallest SAD as the final result.

This is a quite efficient algorithm; however, in the design; it does not support VBSME. Therefore, base on this algorithm, we make several modifications and further propose another refined quarter motion estimation to support the VBSME.

3.4.4. Multi Resolution Motion Estimation (MRME)

A conventional MRME [8] is shown Figure 9. It uses three hierarchical levels for search and refines the motion vector from the coarse level to the finest level. At first, it searches two minimum cost motion vector in level 2, which has coarsest resolution. In the second, it refines these two motion vectors along with the predicted motion vector (MVP) in level 1 and selects motion vector which has the minimum cost of them. At last, it further refines this motion vector to get the final result.

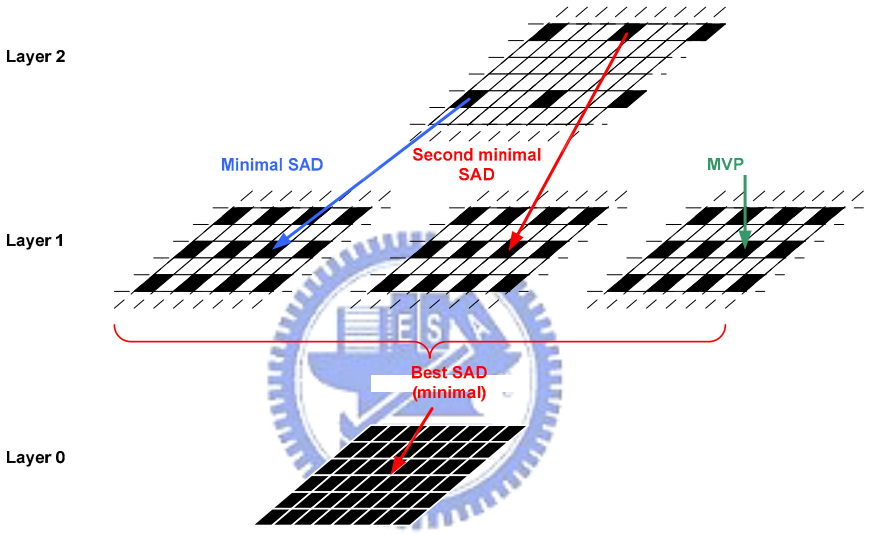

Figure 9 the conventional multi resolution algorithm

It seems to be an effective algorithm to reduce the search timing; however, this approach has several disadvantages for hardware implementation. First, the motion vector found in the higher level needs to be further refined in the lower level. It means the search is a sequential process that will increase the cycle counts, and decrease the hardware utilization and throughput. Besides, a full search range sized buffer is still needed because the dependency between the three hierarchical levels. It will greatly increase the hardware costs if we directly adapt the algorithm for large search range design. Last but not the least; the required bandwidth is still quite large because of poor data reuse in the refinement process.

In this thesis, we propose a highly parallelized MRME not only solve the problems but further supports a large search range motion estimation to meet the high requirement for HDTV applications.

There are still a lot of fast search algorithms such as cross search (CS) [9], one-dimensional gradient descent search (1DGDS) [10], the block-based gradient descent search (BBGDS) [11], the four-step search (4SS) [12], the diamond search (DS) [13], the cross-diamond search (CDS) [14] and the hexagon-based search (HEXBS)[15] and etc. ; however, most of them have several drawbacks in common. In one hand, they may be trapped into a local minimum search point and cannot find the best motion vectors. On the other hand, in view of hardware design, because the search flows of them are not regular, it will decrease the hardware utility and throughput. Last but not the least, these irregular search flow will result in low data reusability, low hardware throughput and raise the memory bandwidth required.

3.5. Review of Skip Mode Detection

In MPEG-4 AVC/H.264 video coding, IME and FME contributes a lot for coding efficiency due to new techniques such as variable block size and six-tap interpolation filter. However, these new complex techniques make ME dominate the computational loading and power of the whole encoding process, up to 96% [16]. The most efficient way to lower the complexity and power of ME is to directly skip the MB encoding and simply denote it with skip mode if the encoding situation is allowed. Therefore, if we can predict the skip mode before ME, we can skip the whole coding stage and save encoding power of these skipped MBs.

In H.264 /AVC, if the following conditions are matched, the MB will be skipped without encoding the motion vector and residuals and just is denoted as skip mode:

1. The chosen block type is 16x16.
2. The best motion vector equals the predicted motion vector (MVP).
3. The chosen reference frame is the previous frame.
4. All coefficients are zero after transform and quantization.

3.5.1. Lagrangian Cost Motion Estimation

In [17], it proposes a skip prediction through Lagrangian cost estimation. The paper use a Lagrangian rate-distortion cost function which incorporates and adaptive model for the Lagrangian multiplier parameter base on local sequence statistics. However, the model is non-linear; therefore it is not suitable for hardware implementation in an efficient way.

3.5.2. All Zero DCT Blocks Detection

In [18] and [19], they perform a comprehensive analysis of the dynamic properties of the DCT and quantization in H.264. They use several partial SADs in a 4x4 block to predict the zero blocks in variety of conditions. Although it is quite precisely, it is not suitable for hardware implementation because these partial SADs cannot acquire in an efficient way and the algorithm has too much different condition branches.

Base on these problems, in this thesis, we will introduce low cost adaptive skip mode detection and accurately predict the skip MB to save the computation complexity and energy of the motion estimation. Besides, we also propose a fast mode decision algorithm to decrease the computation loading of fractional motion estimation which will lower the pipeline efficiency of hardware design.

4. Low Cost Hardware Friendly Adaptive Skip Mode Detection

In this chapter we present a low cost skip mode algorithm and its architecture by detecting 4x4-zero block numbers in a macroblock (MB). With this simple zero block detection and an adaptive threshold, the proposed algorithm can pre-skip 82.39% of total MB encoding and saves 77.13% of coding time for low motion CIF sized sequences with QP=36. Compared with the reference software [20], we can achieve similar quality because of the high accuracy in the skip mode detection. Due to the simplicity of this hardware friendly algorithm, the hardware cost is just 0.63K gate counts for 100MHz clock frequency. With this algorithm, we can efficiently skip the power hungry motion estimation and intra prediction and thus it can be applied to the power constrained mobile devices.

4.1. Introduction

In MPEG-4 AVC/H.264 video coding, a lot of new complex techniques make ME dominate the computational loading and power of the whole encoding process, up to 96% [16]. Thus, speedup with VLSI circuits or fast algorithms is necessary. In which, the most efficient way is to directly skip the MB encoding and simply denote it with skip mode if the encoding situation is allowed. Therefore, if we can predict the skip mode before ME, we can skip the whole coding stage and save encoding power of these skipped MBs. This situation could often happen in many low motion sequences, and thus we can save over 80% of ME for these low motion MBs.

However, the conventional flow as in reference software JM9.0 [20] will still do the MB encoding to detect if it can skip them. In reference software, the skip mode will be decided after the FME finds the best motion vector and finishes the transform and

quantization. All these conditions require encoding for decision and thus will waste power once the MBs are skipped. Thus, if the skipped MB can be predicted before ME, the MB can directly go to entropy coding.

Several approaches have been proposed. In [17], it uses an adaptive threshold for different types of MB. In [18], they use the relationship of 4x4 integer discrete cosine transform (DCT) and partial sum of SAD (sum of absolute difference) to predict the zero block. In [19], they derive a more conservative and thus accurate threshold to predict the SAD of one 4x4 zero block. However, the condition of the threshold is too conservative, and thus could miss many opportunities to skip them. In [21], it uses the total SAD of one MB to predict skip MB, but it is not accurate because it will miss a lot of possible zero MBs in our analysis. In [22], they use an open-loop adjustable threshold that is not robust to have consistent performance. Besides, most of these approaches are not regular and thus are not easy and efficient for hardware design.

To overcome these disadvantages mentioned above, we propose a low cost hardware friendly close-loop algorithm and its architecture. The concept of our approach is that probability of a zero MB is highly depended on its contained zero 4x4-block numbers. Thus, we use a SAD threshold to detect a 4x4 zero block, and an adaptive threshold of the number of 4x4 zero block in a MB to decide a zero MB. Furthermore, we remove the exceptional cases with a spike threshold. With this, we can achieve higher detection with accurate prediction.

4.2. The Fast Skip Algorithm

The whole algorithm is illustrated in Figure 10. We first detect whether a 4x4-block is zero or not by a **4x4-block-SAD-threshold**. We count the number of zero 4x4-blocks in a MB. If the number is larger than an adaptive **MB-zero-block-threshold**, we will denote this MB as a zero MB and skip its encoding. To avoid above SAD threshold affected by local large variations, we adopt a **Spike-threshold** to remove such cases for more accurate detection.

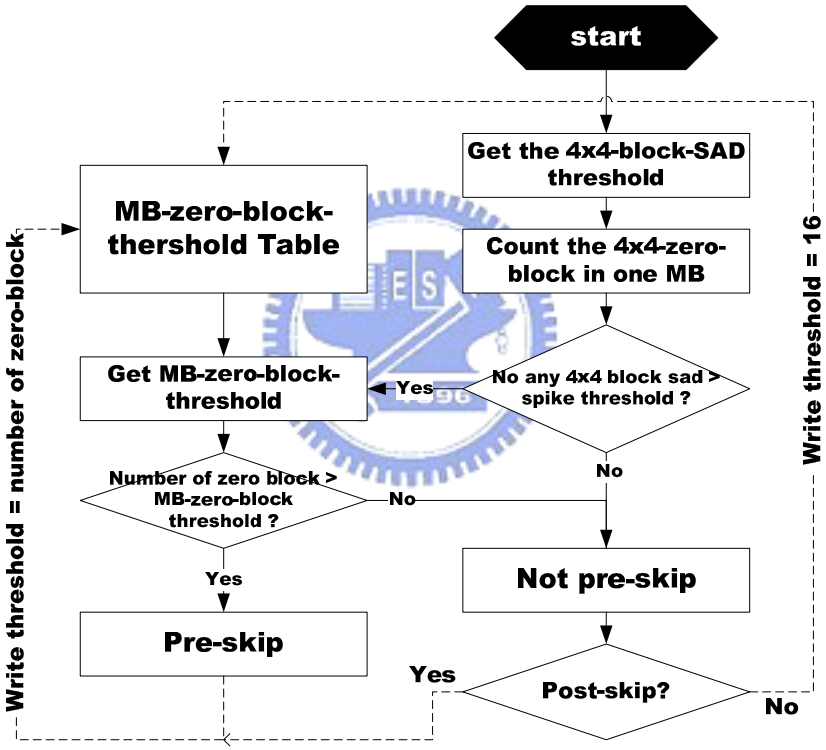

Figure 10 the skip detection flow

4.2.1. 4x4-block-SAD-threshold

This threshold is used to decide if a 4x4-block is zero. We determine this by analyzing the distribution of the 4x4-block SADs higher than the must-be-zero-block-threshold [19] -- we call it T0-- but also quantized to zero block in skipped MBs. We use five 100-frame CIF

sized test sequences to determine this threshold as shown in TABLE II. In which, the “mean”, ”variance”, ”maxima” stand for the average, standard deviation and maxima values of these 4x4-blocks whose SADs are higher than T0. The boundary of the 4x4-block-SAD-threshold is the summation of mean and variance.

From this table, we can find that almost 85.9% in average of the 4x4-block SADs in one skip MB is lower than the boundary. When the SAD of the 4x4-block is less than the boundary, we consider the 4x4-block as a zero block. Therefore, we choose the minimum one of the five sequences as the *4x4-block-SAD-threshold* to prevent from large prediction error. The *4x4-block-SAD-threshold* under different QPs is shown in TABLE III.

TABLE II boundary determination of QP28
(Mean, variance, boundary and maxima for the 4x4-block SAD distribution which higher than T0 when QP 28)

	akiyo	mother	foreman	football	silence
Mean	43	45	45	48	55
Variance	10	9	10	10	12
Boundary	53	54	55	58	67
Maxima	100	**97**	117	**97**	111

TABLE III the 4x4-block-SAD-threshold and Spike-threshold under different QP

	QP20	QP24	QP28	QP32	QP36
4x4-block-SAD-threshold	21	34	53	82	125
Spike-threshold	36	63	97	160	231

4.2.2. MB-zero-block-threshold

In the reference software, we can only decide the MB as a skip MB when the MB has sixteen 4x4-zero-blocks. However, we should consider more about the characteristic of the skip MB. The skipped MB always has spatial and temporal correlations. For example, as in Figure 11, when the MB belongs to the background such as the wall, it is likely that

the neighboring MBs are also background due to spatial correlation. Therefore, we record the zero block numbers as the *MB-zero-block-threshold* if the MB is skipped; otherwise we record the *MB-zero-block-threshold* as 16.

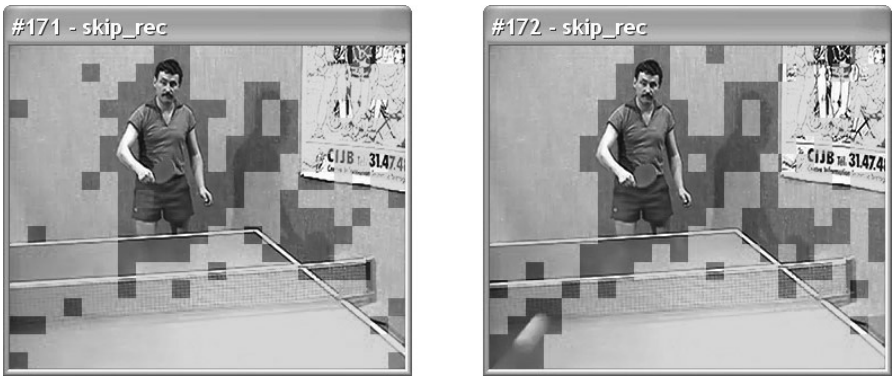

Figure 11 the skipped MB in the sequence "table" in frames 171 and 172 (Light-colored blocks are skipped MB)

For the hardware pipeline consideration, the current(C) *MB-zero-block-threshold* is determined by the minimum threshold of upper (U) and upper-left (L) and upper-right (R), as depicted in Figure 12. The use of (L) is to avoid the "read-after-write" data hazard in the pipelined hardware design.

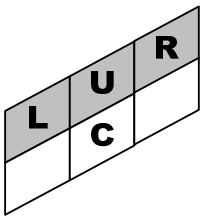

Figure 12 the MB-zero-block-threshold prediction C=min {U, L, R}.

4.2.3. Spike-threshold

Due to its adaptive adjustment, the *MB-zero-block-threshold* could be decreased by the neighboring MB, such as to 13. However, this could lead to a case that the number of zero blocks exceeds the threshold but also contains blocks with large SAD values. In such case, the MB should not be skipped but will be detected as the skipped. For example, in Figure 11, there is one ping-pong ball flying from the left bottom corner in the 172 frame. For this case, the *MB-zero-block-threshold* could be decreased to 13 and this MB has 14 zero blocks that exceeds the threshold. However, it also has two 4x4-blocks with large SADs. In this situation, this MB will be skipped by examining the threshold only, and thus cause error prediction. Thus, we must set up a threshold to detect if there are any large 4x4-block SAD -- we call such 4x4-block as a 'spike'. The *spike-threshold* is determined by the minimum one among the six 'maxima' values in TABLE II. When QP is equal to 28, the spike-threshold is 97. The *spike-threshold* under different QPs is illustrated in TABLE III.

4.3. The Fast Skip Architecture

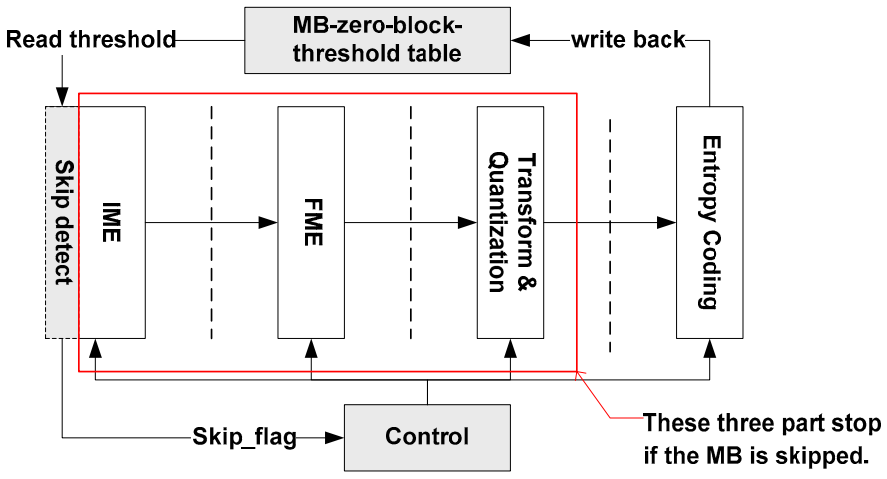

Figure 13 the system architecture

Figure 13 shows the system architecture of our design. The gray colored block is the added hardware by our pre-skip design. The skip detection module is at the beginning stage and combined with the IME module. At each time, the IME starts searching at the MVP and gets the sixteen 4x4-block SADs for the skip detection module. The detection module then compares these SADs and 4x4 zero block count with the *4x4-block-SAD-threshold, MB-zero-block-threshold* and *Spike-threshold* to decide whether the MVP is a skip MB or not. If it does, the skip detection module raises the "skip_flag" signal to the control logic and it will abort the rest of encoding steps. At last, the entropy coding module encodes the MB as a skipped MB and writes back the number of zero block of the skipped MB. On the other hand, if the MB at the MVP is not detected as skipped MB in the first stage, the IME continues searching the best MV in the search range and goes through the normal procedure. However, the entropy coding also has to write back the MB-zero-block-threshold table. If the MB is decided as a skipped MB in the entropy coding stage, it writes back the number of zero blocks. Otherwise, it writes back the threshold as 16.

Figure 14 is the skip detection module architecture. It consists of sixteen comparators to decide whether the SADs are bigger than the *4x4-block-SAD-threshold* and *spike-threshold*. It also include one adder to sum up the number of zero block and one "OR gate" to decide if the MB has spike. In addition, another comparator is used to select the minimum of *MB-zero-block-threshold* in the neighbor MB. At last, an "AND gate" will check whether the MB is to be skipped.

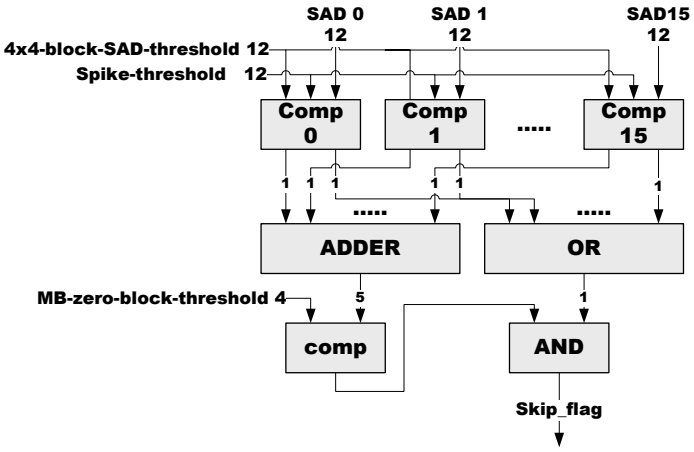

Figure 14 the skip detect architecture

4.4. Experimental Result

The simulation environment is described below. First, we will show the performance under various small size test sequences. Since the number of skipped MB is highly depended on the sequence contents, we roughly partition eight 300-frame CIF sized test sequences into three categories, low motion, medium motion and high motion sequences for more accurate evaluation. The low motion sequence are: 'akiyo', 'mother_daughter', 'news' and the medium motion sequences are 'container', 'table', 'foreman'. The high motion sequences are: 'stephan' and 'mobile'. This algorithm is included into the reference software. Second, we show the performance of large frame size sequences, which are 300-frame 720p sized 'Stockholm' and 'mobcal'.

The test environments are: baseline profile, no rate distortion optimization, one reference frame, search range is equal to 16 and 32 for CIF and 720p sized sequences respectively. The average performances of pre-skip detection are listed in TABLE IV - TABLE VII, with high QP, our design will save more bit rate with some PSNR drop. The "skip hit rate" is the ratio of pre-skip MBs divides the number of skip MBs. All these following simulations are compared with the reference software. All these following simulations are compared with the reference software.

TABLE IV performance of pre-skip detection for low motion CIF sequences

	PSNR (dB)	Bit-rate (%)	Skip hit rate (%)
QP20	-0.02	-1.79	92.10
QP24	-0.10	-1.56	93.44
QP28	-0.13	-2.86	94.82
QP32	-0.21	-5.70	94.53
QP36	-0.28	-8.65	96.56

TABLE V performance of pre-skip detection for medium motion CIF sequences

	PSNR (dB)	Bit-rate (%)	Skip hit rate (%)
QP20	0.00	-0.02	85.71
QP24	-0.09	-0.66	85.86
QP28	-0.13	-1.95	88.96
QP32	-0.21	-4.68	91.13
QP36	-0.36	-7.12	94.77

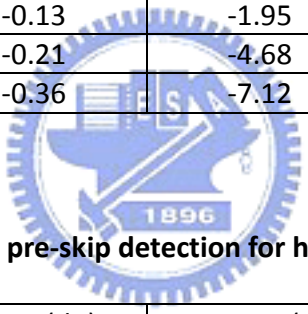

TABLE VI performance of pre-skip detection for high motion CIF sequences

	PSNR (dB)	Bit-rate (%)	Skip hit rate (%)
QP20	0.01	0.17	86.15
QP24	0.01	0.37	92.82
QP28	0.01	1.08	91.70
QP32	0.03	1.27	86.09
QP36	-0.02	0.01	86.93

TABLE VII performance of pre-skip detection for 720p sequences

	PSNR (dB)	Bit-rate (%)	Skip hit rate (%)
QP24	0.03	2.20	86.85
QP28	0.05	4.47	89.75
QP32	-0.06	-0.32	89.31
QP36	-0.20	-3.38	90.72

Figure 15 shows the RD curve of the low motion sequence. The curve of our design is almost the same as the original curve, and sometimes is slightly better than the original curve under high QP because we skip more MBs than JM9.0. These skipped MBs are nearly to be skipped so it does not degrade performance significantly. Figure 17 shows the results of medium motion sequences. Our design has almost the same curve as original curve. Figure 19 reveals the result for high motion sequence; because there are few MBs which are skipped, the performance is almost the same as the original curve. Figure 21 is the RD curve of 720p sequences, it shows slight quality drop because the high definition characteristic of large size sequences are much difficult to correctly predict and an error predicted skip MB will also cause more serious quality drop.

Figure 16, Figure 18, and Figure 20 are the average distribution of not skipped MB, and the skipped MB for different categories CIF sized sequences. Besides, the distributions of 720p sequences are depicted in Figure 22. The portion of skipped MB consists of three types: error predict, under skip and correct skip. “Bad skip” means the MB is pre-skipped but it should not be skipped. “Miss skip” means the MB should be pre-skipped but it is not detected in the pre-skip stage by our design. “Correct skip” means we can accurately pre-skip the MB. We can see that the higher QP case will skip more MBs, up to 82.39 % for low motion and 65.88% for medium motion sequence in average. On the other hand, we find that the “error predict” does not degrade the performance a lot because these error predicted MBs are nearly to be skipped.

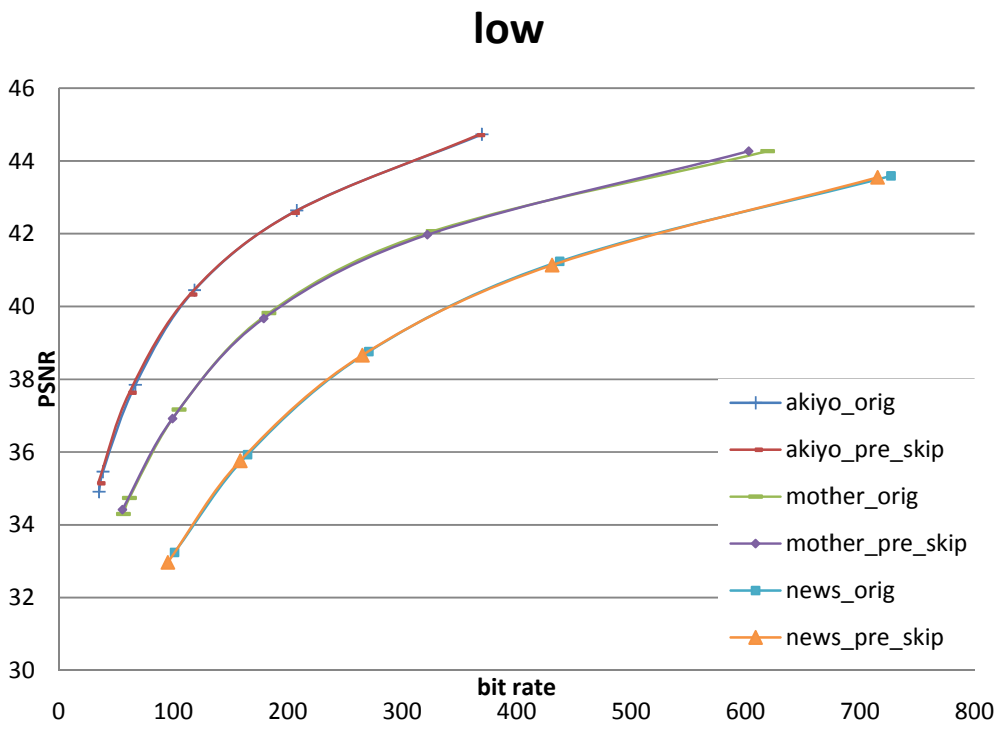

Figure 15 the RD curve of low motion sequences

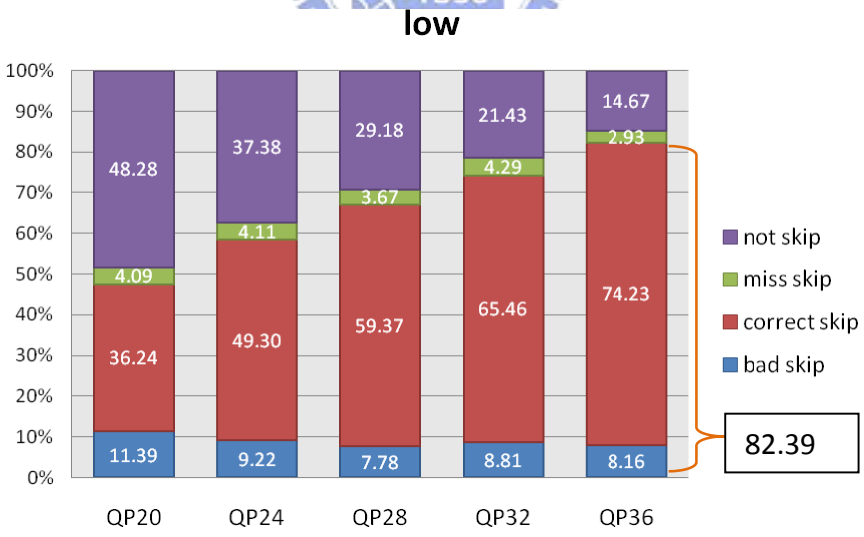

Figure 16 the distribution of MB for low motion sequences

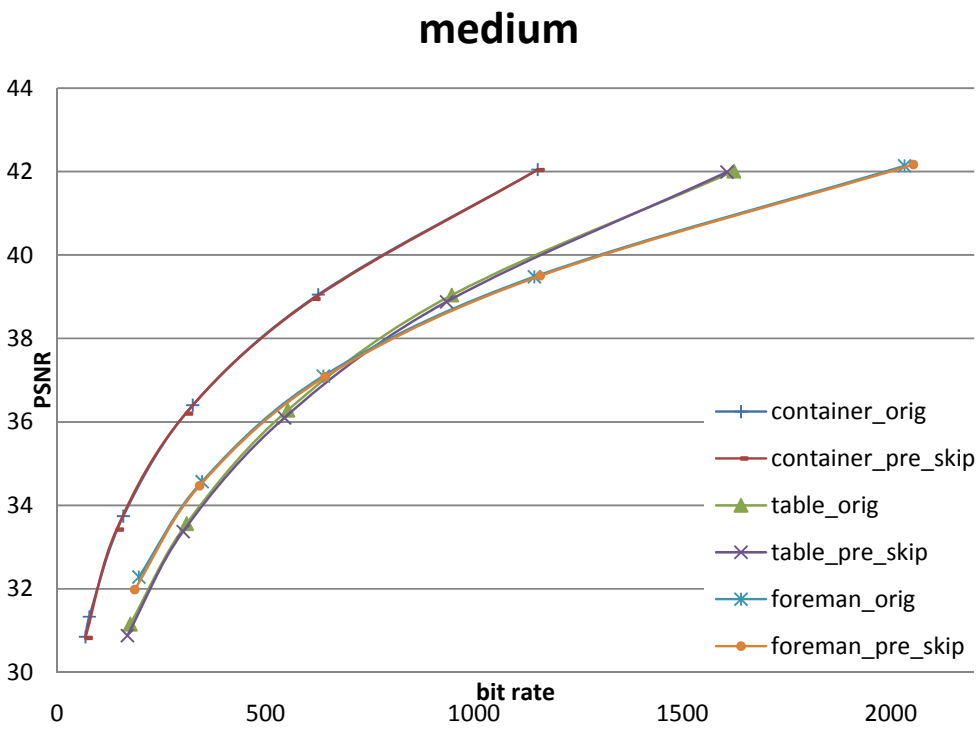

Figure 17 the RD curve of medium motion sequences

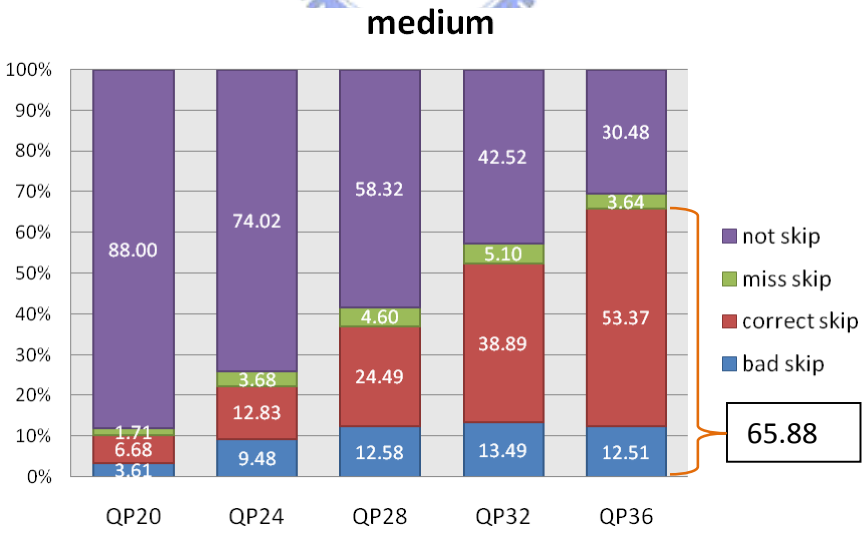

Figure 18 the distribution of MB for medium motion sequences

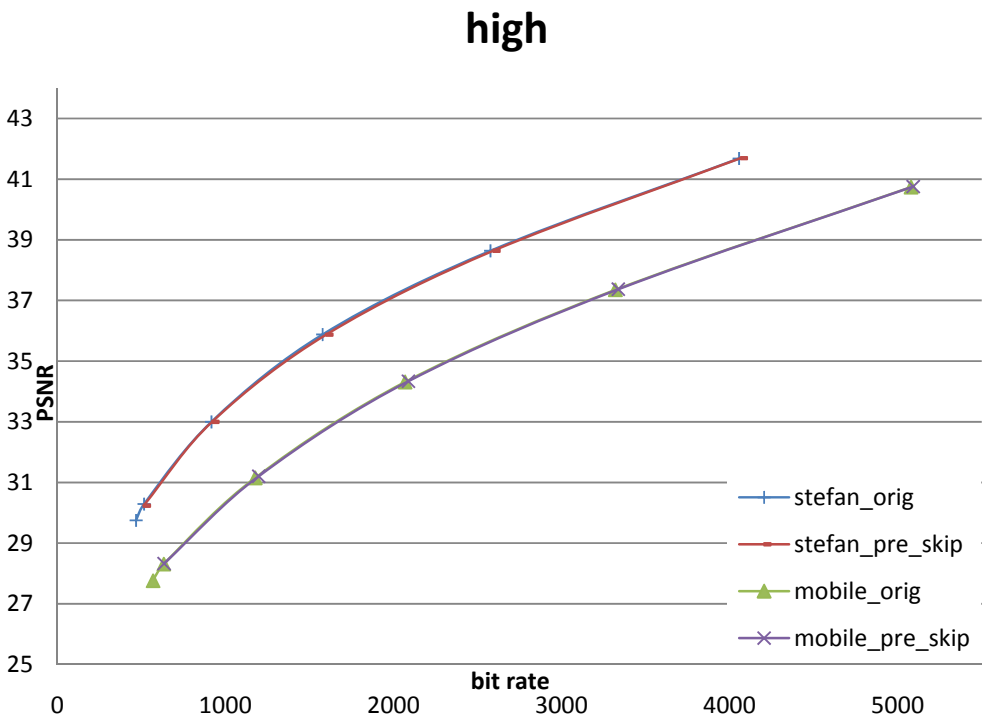

Figure 19 the RD curve of high motion sequences

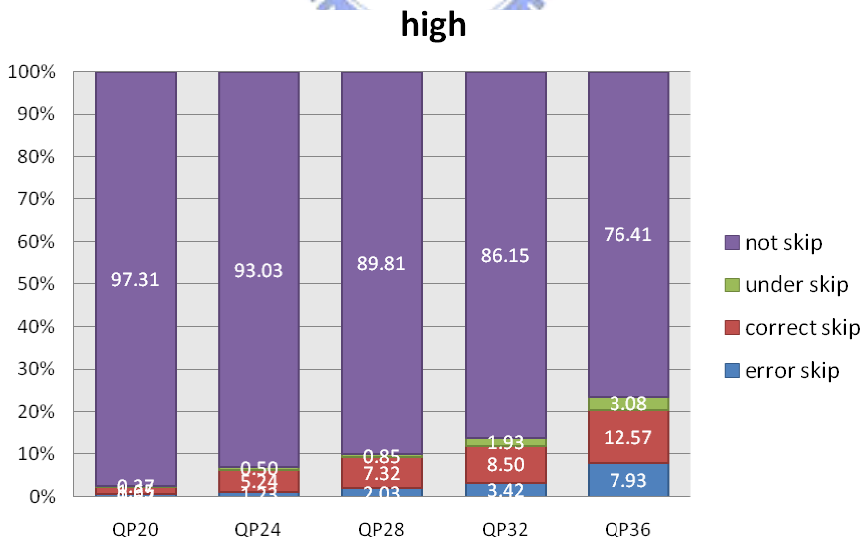

Figure 20 the distribution of MB for high motion sequences

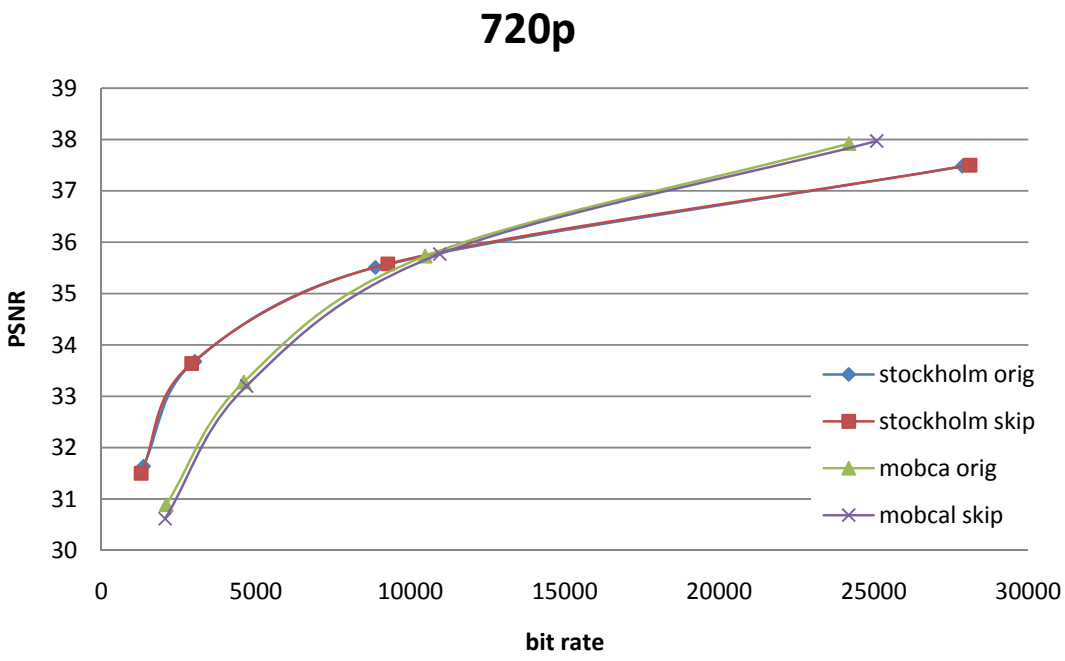

Figure 21 the RD curve of 720p sequences

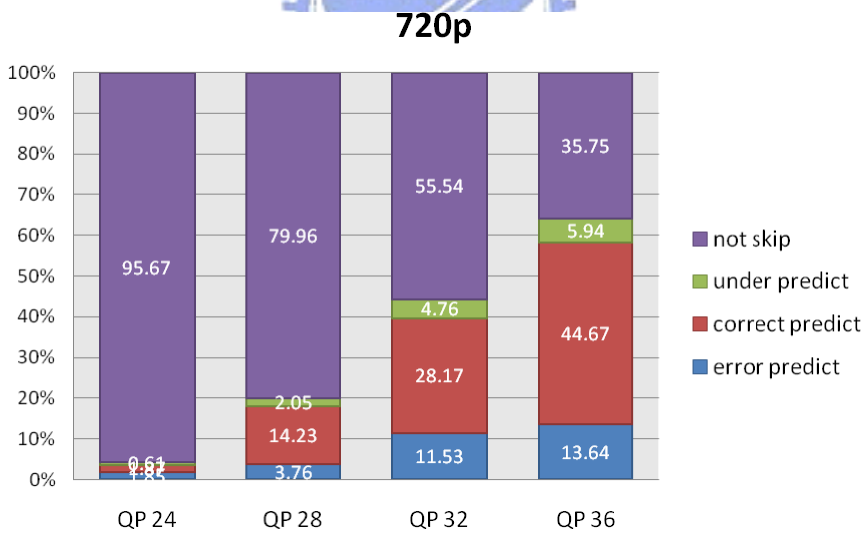

Figure 22 the distribution of MB for 720p sequences

Figure 23 and Figure 24 are the coding time of CIF and 720p sequences, both of them reveal that the coding time is roughly proportion to QP, whereas the coding time decrease rate for 720p sequences are less than CIF sequences, it is because high definition frame can be hardly skipped.

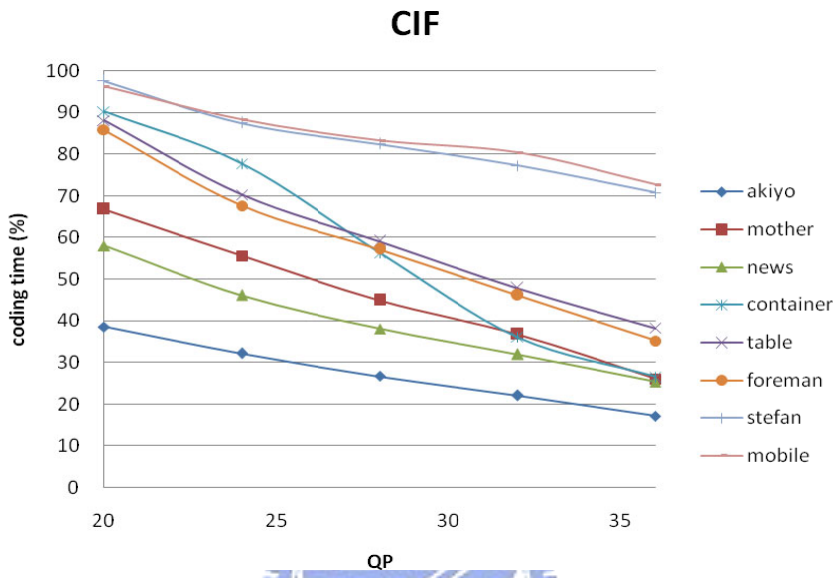

Figure 23 coding time (%) of CIF sequences

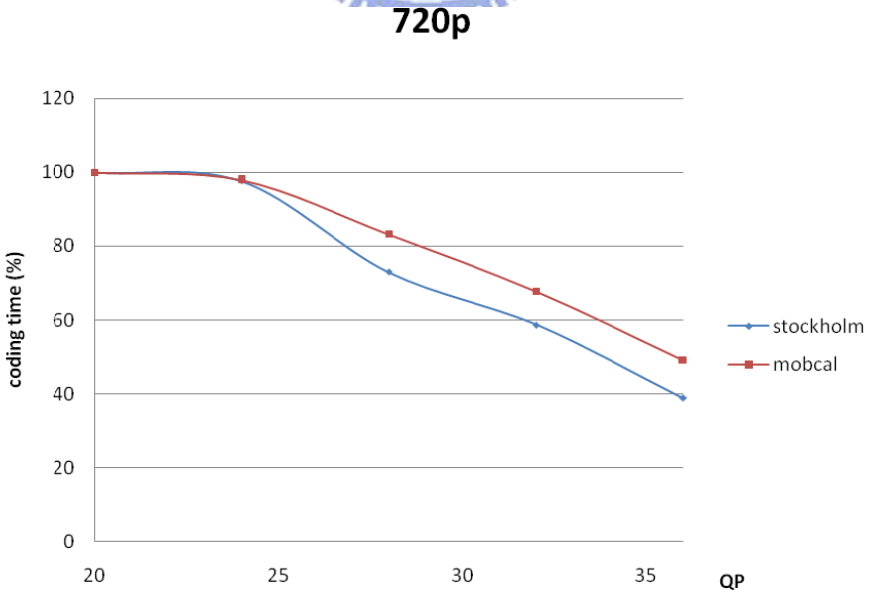

Figure 24 coding time (%) of 720p sequences

TABLE VIII shows our average coding time compared with the JM9.0. For QP=36, our design only needs 22.87% and 29.63% coding timing for low motion and medium motion sequences. However, for high motion sequences, the performance is limited. For 720p sequences, our design also perform well; however, the coding time saved is not as much as CIF sized sequences because of its high definition quality and the characteristics of the sequences itself. Comparisons to other approaches [17] - [19], [21], [22] are not included since their simulation conditions are quite different than the proposed one, and thus are hard to compare to them.

TABLE IX shows the hardware cost synthesized by UMC 0.13 CMOS technology for frequency 100 MHz The memory table size comes from the horizontal number of MB in one line, and each MB takes 0.5 bytes memory.

TABLE VIII the average coding time (%) for categorized CIF and 720p sequences (Compared with JM9.0)

Our design compare with JM9.0				
Time (%)	Low	Medium	High	720p
QP20	54.48	88.38	96.89	
QP24	44.65	73.91	87.91	98.72
QP28	36.58	49.16	82.86	78.12
QP32	30.27	41.63	78.81	63.17
QP36	22.87	29.63	71.72	43.98

TABLE IX the hardware cost of the skip design

Skip detection hardware cost	0.63K gate counts
MB-zero-block-threshold memory table	11 bytes (CIF) / 40 bytes (720p)

4.5. summary

In this chapter, we propose low cost skip detection hardware algorithm and architecture. The hardware costs are 0.63K gate counts, 11 bytes and 40 bytes memory for CIF size and 720p size video respectively. The video performance of our design is quite similar the same as the JM9.0 and our design can save 77.13% to 11.62% of coding time in difference sequences and QPs.

5. Hardware Efficient Fast Mode Decision Algorithm

In this chapter, we present a hardware friendly fast algorithm for motion estimation (ME) in H.264 video coding. The algorithm can save hardware computing cycles by separating the integer-pixel ME and fractional-pixel ME phase with hardware oriented mode filtering. The simulation results show that it can significantly reduce the cycles with small PSNR drop and bit rate increase.

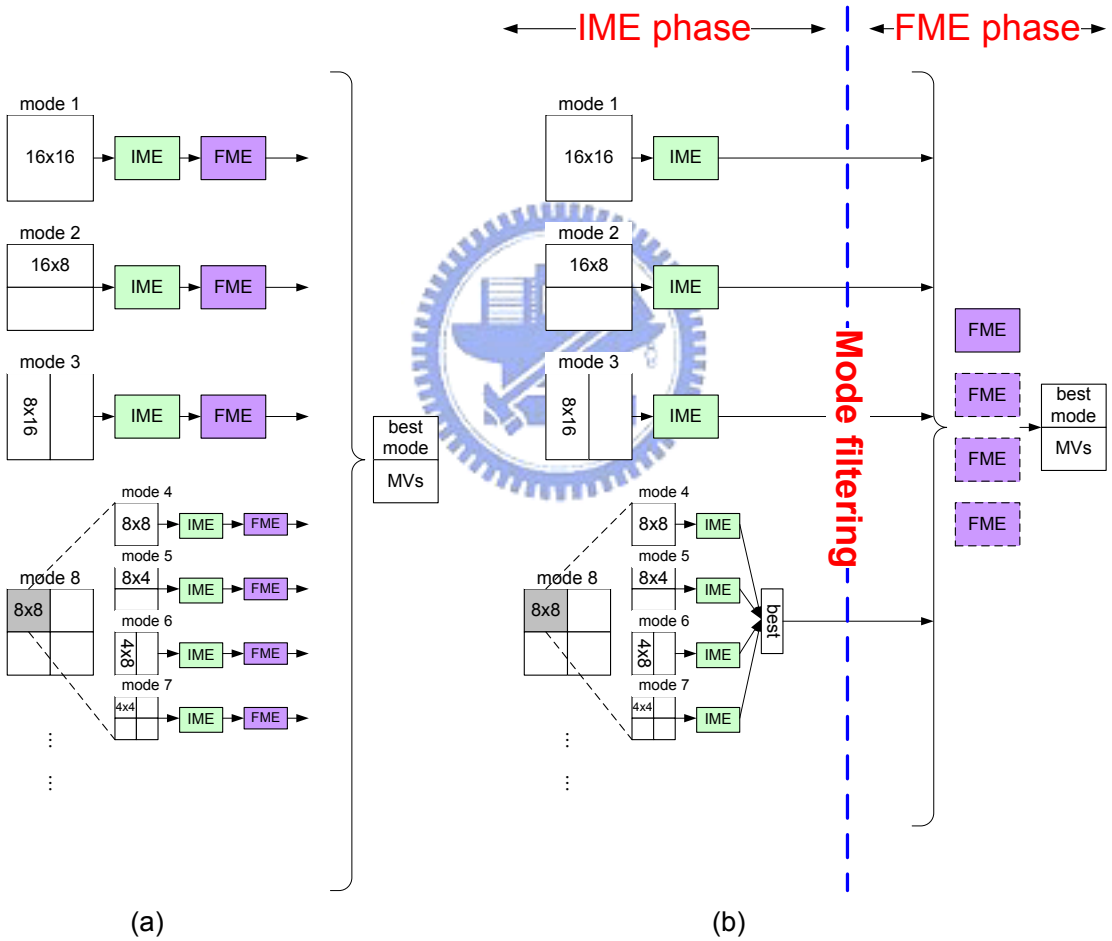

Figure 25 algorithm of mode filtering
(a) is the original reference software algorithm (b) is our proposed mode filtering algorithm

5.1. Introduction

In the standard of MPEG-4 AVC/H.264 video coding [1], variable block size motion estimation (VBSME) consists of integer ME (IME) and fractional ME (FME), which are adapted to fit different details of video sequences. Thus, one 16x16 macroblock (MB) can be partitioned into two 16x8 blocks, two 8x16 blocks or four 8x8 blocks. Furthermore, one 8x8 block can be further partitioned into two 8x4 blocks, two 4x8 blocks or four 4x4 blocks, as shown in Figure 25 (a). With all these combinations, IME could generate totally 41 motion vectors (MVs) for the following FME and this full search flow is widely adopted in current VBSME designs [23], [24], [25] .

However, such high computational complexity not only dominates the computational loading of the whole encoding process, but also results in an unbalanced hardware designs. In [23], for [-16~15] search range, the cycle count of IME is about 1089 cycles per macroblock (MB) by data reuse and data parallelism like the 2-D array based designs. On the other hand, the cycle count of FME is even larger, 1648 cycles for full 41 MVs per MB [25]. Thus, at least 2737 cycles are needed to find a best motion vectors (MVs). Though further pipelining IME and FME can reduce the long latency, it is still unbalanced and results in bad hardware utilization. These kinds of problems result in the difficulty to realize the real-time encoder of H.264.

Thus, in this chapter, we propose a fast algorithm that selects only the most possible candidate modes after IME (called mode filtering) such that FME can have shorter computational cycles. The proposed algorithm can be regarded as a more general approach of the concept in [25]. However, our method explores larger design space and achieves better performance than that in [25].

Figure 25 (a) depicts the original algorithm used in the H.264 reference software [20]. In each mode, IME is followed by FME immediately, and only one best mode and its MVs will be decided finally. However, in order to save the hardware computing cycles, we separate the IME and FME into different parts instead of directly coupled. With this, we transmit one to four (1~4) numbers of IME mode and MV candidates to FME module. Intuitively, with the larger the numbers of IME candidates to be transmitted, the larger loading of FME is need and better performance can be achieved. This method is called "Mode Filtering."

5.2. Mode Filtering Algorithm

As illustrated in Figure 25 (b), after computing the cost of all possible block sizes, we select the most possible few modes for FME (mode filtering) instead of all modes for FME. The most possible few modes are selected from one of 16x16, 16x8, 8x16 and the best of the 8x8 and subblock. For the case of 8x8 and subblock size, each 8x8 block will select its own best mode (8x8, 4x8, 8x4 or 4x4 mode). In the worst case, every subblock will have a 4x4 mode and the four subblocks consist of 16 MVs totally. Therefore, with different number of mode candidates transmitted, different numbers of FME calculation are need and the relationship is shown in TABLE X. For example, if only 2 mode candidates are transmitted, only 3 to 18 MVs instead of 41 MVs are calculated in FME, which can save a lot of computing cycles.

TABLE X the relationship of candidates and motion vectors

Number of candidates	Number of motion vectors
1	1 ~ 16
2	3 ~ 18
3	5 ~ 20
4	9 ~ 21

5.3. The Simulation Result of Mode Filtering

As the previous chapter, we test the algorithm in two different kinds of video: QCIF, CIF (small frame size) and 720p (large frame size) test sequences to see the performances of mode filtering under small frame size and large frame size conditions. The contents of QCIF sized sequence, includes ‘akiyo’, ‘foreman’, and ‘mobile’ (QCIF and CIF), all of three which are low motion, medium motion, and high motion sequences respectively. The contents of CIF sequences are the same as QCIF. For 720p sequences, the test sequences are ‘Stockholm’, and ‘park_run’. The simulation environment is set up as follows: The search range 8, 16 and 32 for QCIF, CIF and 720p sequences respectively. The reference software is JM 9.0 [20] with Rate-Distortion Optimization (RDO) off.

5.3.1. Performance of QCIF/CIF Sequences

Figure 26 - Figure 28 show the result of different candidates’ mode filtering algorithm for the three QCIF sequences and Figure 29 - Figure 31 show the result of different candidates’ mode filtering algorithm for the three CIF sequences. Clearly, in small size sequences, we can see that the 2~4 candidates’ method have similar performance as reference software, whereas the 1 candidate’s method performs not so well.

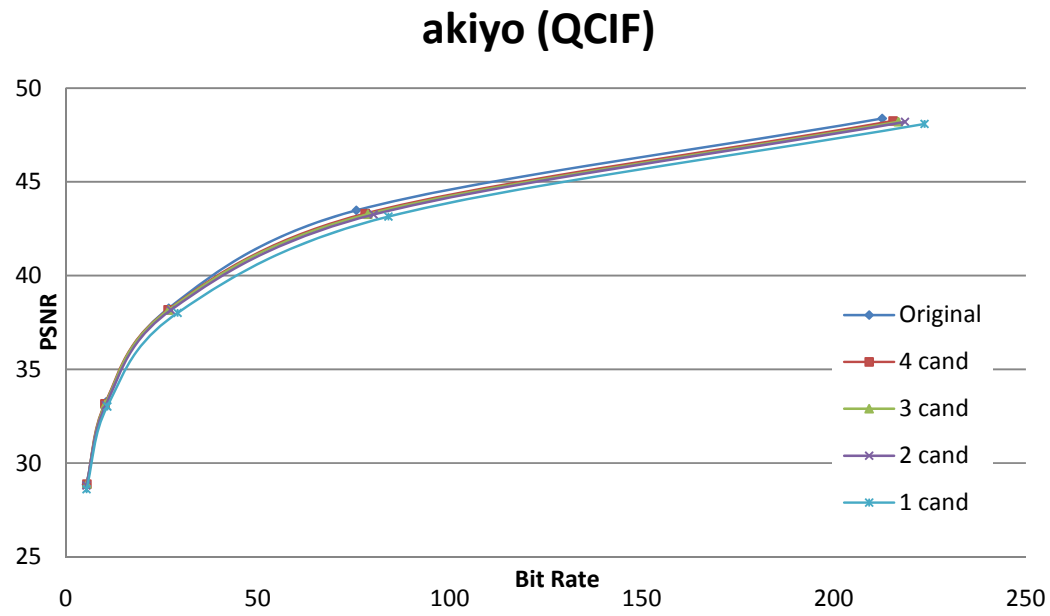

Figure 26 the mode filtering simulation result for akiyo sequence (QCIF)

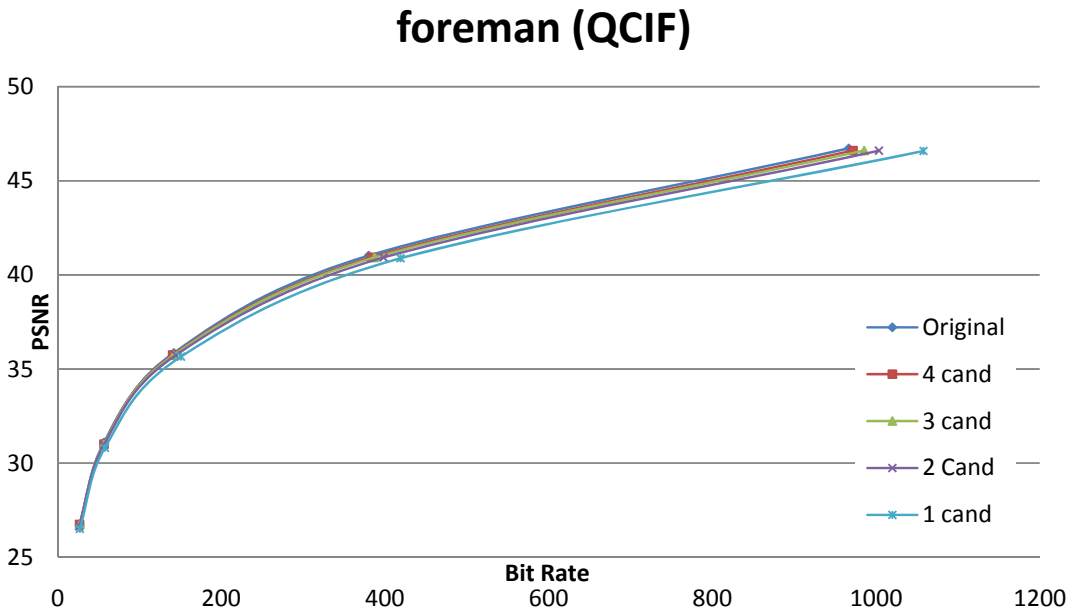

Figure 27 the mode filtering simulation result for foreman sequence (QCIF)

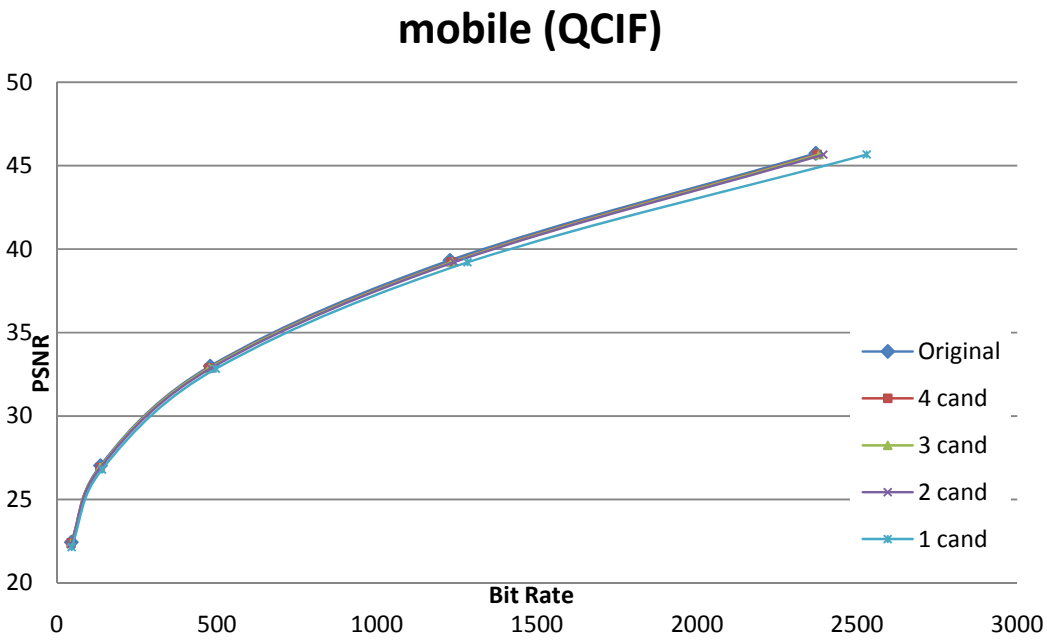

Figure 28 the mode filtering simulation result for mobile sequence (QCIF)

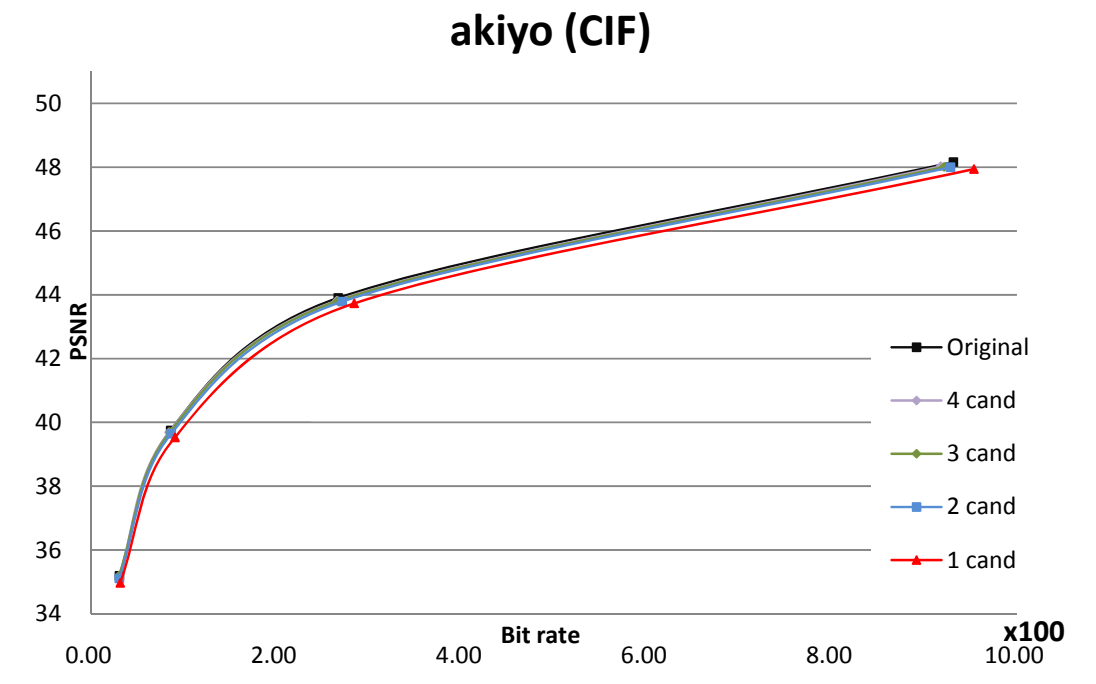

Figure 29 the mode filtering simulation result for akiyo sequence (CIF)

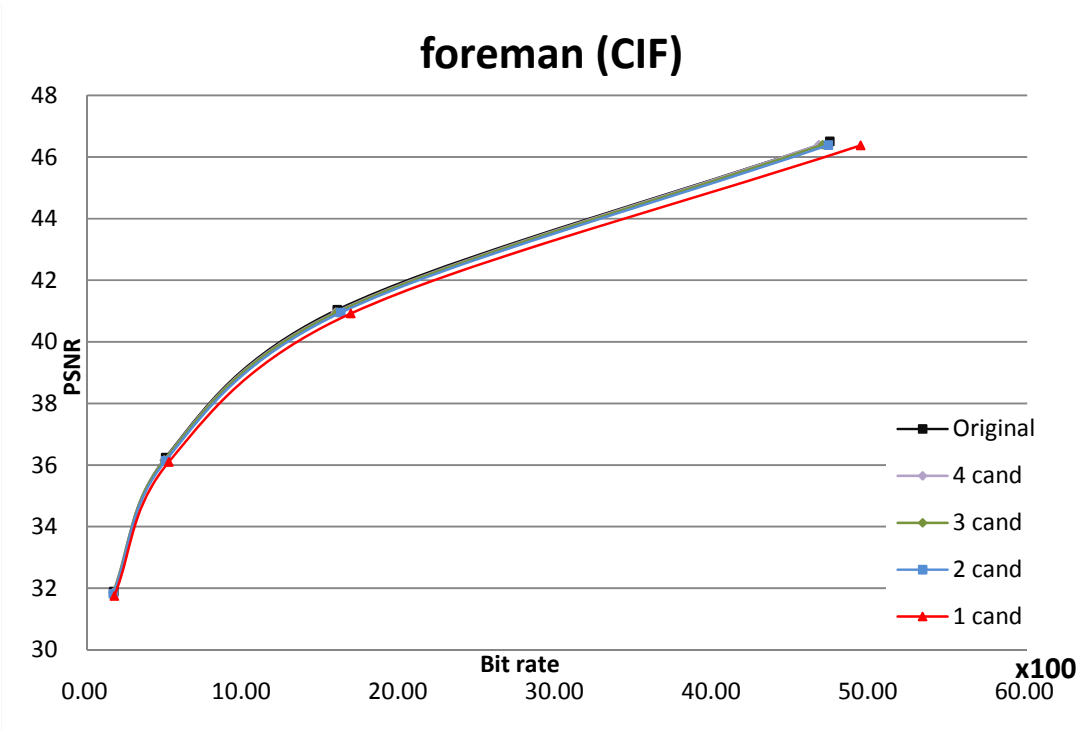

Figure 30 the mode filtering simulation result for foreman (CIF)

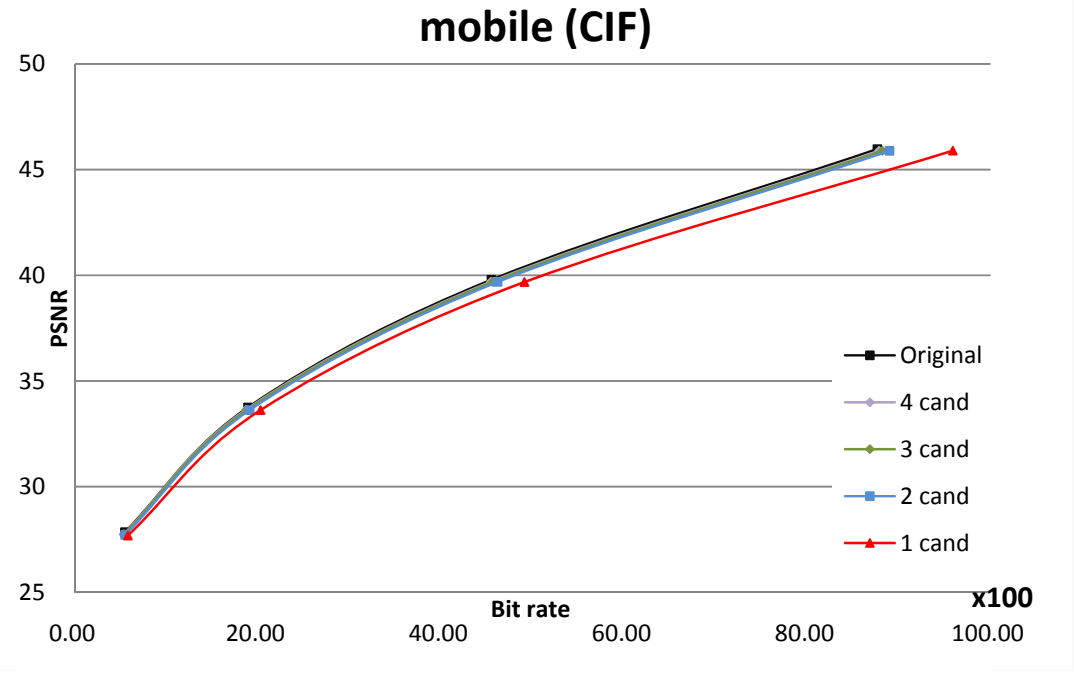

Figure 31 the mode filtering simulation result for mobile (CIF)

TABLE XI and TABLE XII list the average performance of this algorithm of QCIF and CIF sequences respectively. In these result, we find out that the one candidate method has larger bit rate increase and PSNR drop. However, when the 2 candidates' method is chosen, the average bit-rate increasing can be only 0.54%/1.30% and the PSNR degradation is only 0.11db for QCIF/CIF. This is a very significant improvement. When 3 and 4 candidates' methods are used, the performance will be improved further but not much.

The performances for CIF sequence are better than QCIF, because mode filtering will filter much of the FME processing of small block size while small block sizes are more preferable in small size sequences. Therefore, Mode filtering does better in CIF sequences, because CIF sequences are smoother than QCIF and large block sizes can deal with CIF sequences well and save bit rate.

TABLE XI the mode filtering performance for QCIF sequence

		4 candidates	3 candidates	2 candidates	1 candidate
QP14	*PSNR (dB)*	-0.10	-0.12	-0.13	-0.16
	Bit-rate (%)	0.68	1.46	2.52	7.11
QP21	*PSNR (dB)*	-0.12	-0.14	-0.14	-0.20
	Bit-rate (%)	1.41	2.11	3.96	8.59
QP28	*PSNR (dB)*	-0.10	-0.11	-0.12	-0.20
	Bit-rate (%)	-0.83	-0.34	1.45	6.11
QP35	*PSNR (dB)*	-0.10	-0.11	-0.15	-0.26
	Bit-rate (%)	-1.49	-1.46	-0.67	2.74
Average	*PSNR (dB)*	-0.11	-0.12	-0.11	-0.21
	Bit-rate (%)	-0.06	0.44	1.30	6.14

TABLE XII the mode filtering performance for CIF sequence

		4 candidates	3 candidates	2 candidates	1 candidate
QP14	*PSNR (dB)*	-0.11	-0.11	-0.12	-0.14
	Bit-rate (%)	-0.92	-0.48	0.50	5.50
QP21	*PSNR (dB)*	-0.08	-0.09	-0.10	-0.13
	Bit-rate (%)	-0.42	0.03	1.59	6.69
QP28	*PSNR (dB)*	-0.07	-0.09	-0.12	-0.16
	Bit-rate (%)	-1.61	-0.99	0.68	5.59
QP35	*PSNR (dB)*	-0.04	-0.05	-0.10	-0.17
	Bit-rate (%)	-1.86	-1.53	-0.63	3.52
Average	*PSNR (dB)*	-0.08	-0.09	-0.11	-0.15
	Bit-rate (%)	-1.20	-0.74	0.54	5.33

5.3.2. Performance of 720p Sequences

Figure 32 and Figure 33 show the result of different candidates' mode filtering algorithm for 720p sequences. TABLE XIII lists the average performance. The results of 720p are very far from the performance of CIF sequences. For 720p sequences, we find that under the condition of low QPs, the 1 candidate's method provides very good performance, because it saves a lot of bit rate (7.16% under QP12) by dropping a little PSNR. However, as the QP become larger, the bit rate increases rapidly and reaches 5.26% increasing under QP36. On the other side, the 2 candidates' method maintains stable performance under different QPs rather than 1 candidate's method. The performance of 3 candidates' and 4 candidates' are quite the same as 2 candidates' method, and both of them cannot effectively improve the performance.

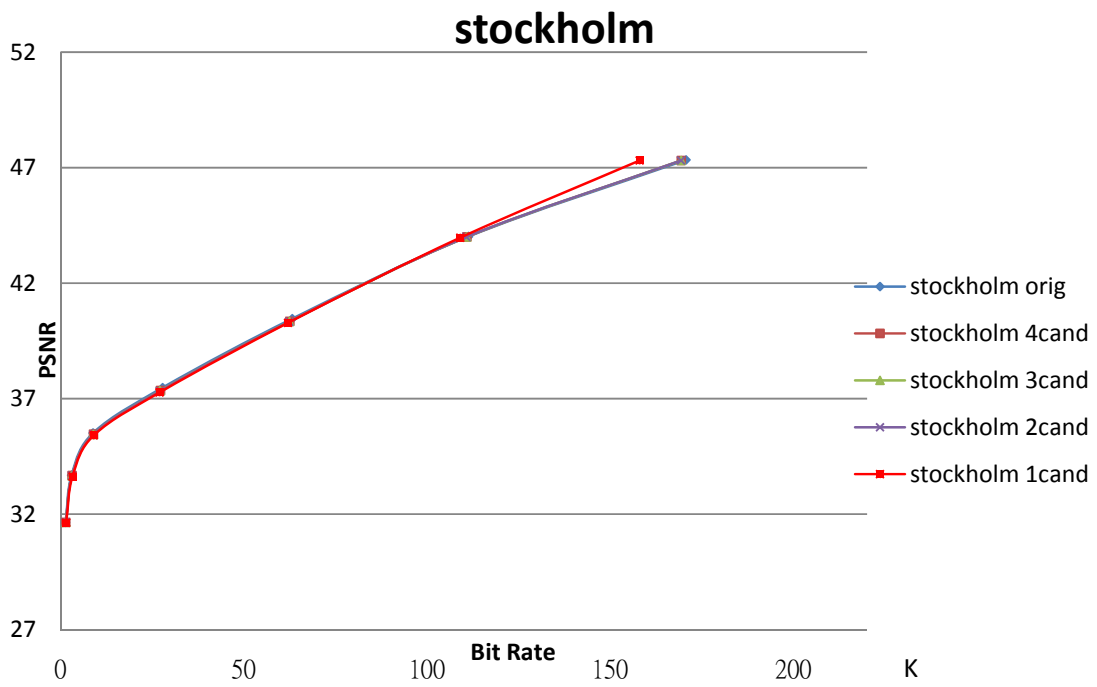

Figure 32 the mode filtering simulation result for Stockholm (720p)

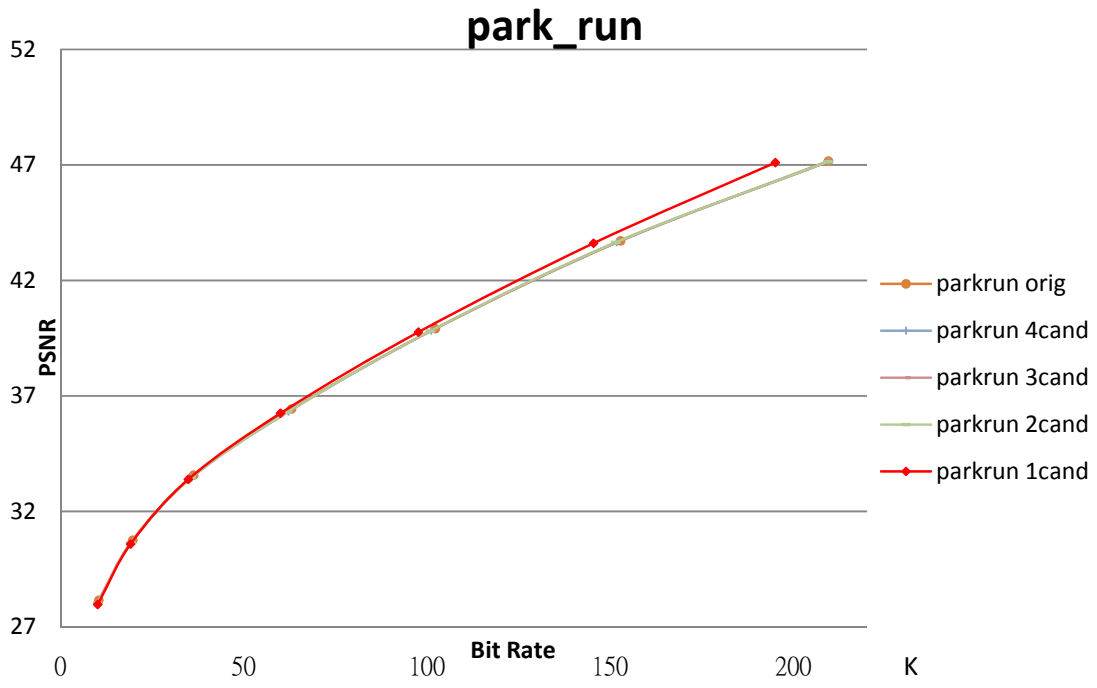

Figure 33 the mode filtering simulation result for park_run (720p)

TABLE XIII the mode filtering performance for 720p sequence

		4 candidates	3 candidates	2 candidates	1 candidate
QP12	*PSNR (dB)*	-0.03	-0.03	-0.03	-0.05
	Bit-rate (%)	-0.41	-0.42	-0.45	-7.16
QP16	*PSNR (dB)*	-0.04	-0.04	-0.04	-0.08
	Bit-rate (%)	-0.57	-0.65	-0.71	-3.44
QP20	*PSNR (dB)*	-0.10	-0.11	-0.11	-0.16
	Bit-rate (%)	-1.10	-1.27	-1.37	-3.21
QP24	*PSNR (dB)*	-0.12	-0.13	-0.15	-0.19
	Bit-rate (%)	-1.73	-2.10	-2.30	-3.89
QP28	*PSNR (dB)*	-0.07	-0.08	-0.09	-0.14
	Bit-rate (%)	-0.05	-0.39	-0.42	-0.91
QP32	*PSNR (dB)*	-0.05	-0.06	-0.07	-0.11
	Bit-rate (%)	1.28	1.12	1.23	2.78
QP36	*PSNR (dB)*	-0.04	-0.05	-0.06	-0.10
	Bit-rate (%)	2.14	1.85	2.19	5.26
Average	*PSNR (dB)*	-0.07	-0.07	-0.08	-0.12
	Bit-rate (%)	-0.06	-0.27	-0.26	-1.51

For small size sequences (QCIF and CIF), the bit-rate overhead is smaller for high QP case (the low bit-rate condition), it is because the mode filtering algorithm prefers to select larger block size, and larger block size is also preferred under the low bit-rate condition because of fewer MV bit. Because QCIF sequences are coarser and smaller than CIF sequences, QCIF will prefer to choose small block size. Therefore, mode filtering performs better in CIF than QCIF sequences. While for 720p sequences, the frame contents are smoother because of the characteristics of high definition; thus, large block size is preferred in 720p sequences and result in the general better performance of mode filtering. This phenomenon proves the point that the IME and FME partition mechanism has an inclination to consume bit-rate to trade off the PSNR performance, so the impact on bit-rate is larger than on PSNR.

5.4. summary

In this chapter, we propose a fast algorithm that can speed up the overall ME process with the reduction of FME modes. With the algorithm, the hardware implementation can be pipelined with higher efficiency and slightly performance loss.

6. Efficient Low Cost Motion Estimation for Portable Devices

For some portable device, in order to achieve the goal of low cost hardware and high efficiency design, we present this hardware friendly fast algorithm and its architecture for motion estimation in H.264 video coding. The fast algorithm adopts the quarter pixel subsampling and mode filtering that reduces the computing complexity of integer ME by 75%, and only two modes instead of a great number of modes are refined for fractional motion estimation. This also can save about 80% fractional ME cycle counts in average. The simulation result shows that the method only increases the bit rate within 1.44% and at most 0.11dB quality degradation; nevertheless, the architecture only costs 41.5% of area cost and requires 48% of cycle counts when compared with the previous designs. Besides, the power consumption for 30fps CIF and QCIF video are just 0.59mW and 0.08mW respectively, which prove that the design is suitable for power constricted portable device.

6.1. Introduction

As mentioned in the previous chapter, an ME design can be partitioned into integer motion estimation (IME) and fractional motion estimation (FME). Most of the current ME designs [23], [26] follow the computing order as in the reference software [20] shown in Figure 34. In this figure, IME will first generate 41 motion vectors (MVs) for various combinations of block sizes, and then FME further refines these 41 MVs into sub-pixel precision. Finally the best MV is selected based on the cost. Though this flow can achieve better coding performance, it has a serious problem for hardware implementation. In hardware implementation, when search range equals 8, the cycle count of IME can be easily reduced to 256 cycles per macroblock (MB) by data reuse and data parallelism like the 2-D array based designs [23]. However, the cycle count of FME is

much larger, 1648 cycles for 41 MVs per MB [25]. The reason for this is that data reuse and data parallelism in FME is hard to achieve to the same level as that in IME because of diverged 41 MVs. This unbalanced problem results in a bad hardware pipelining performance.

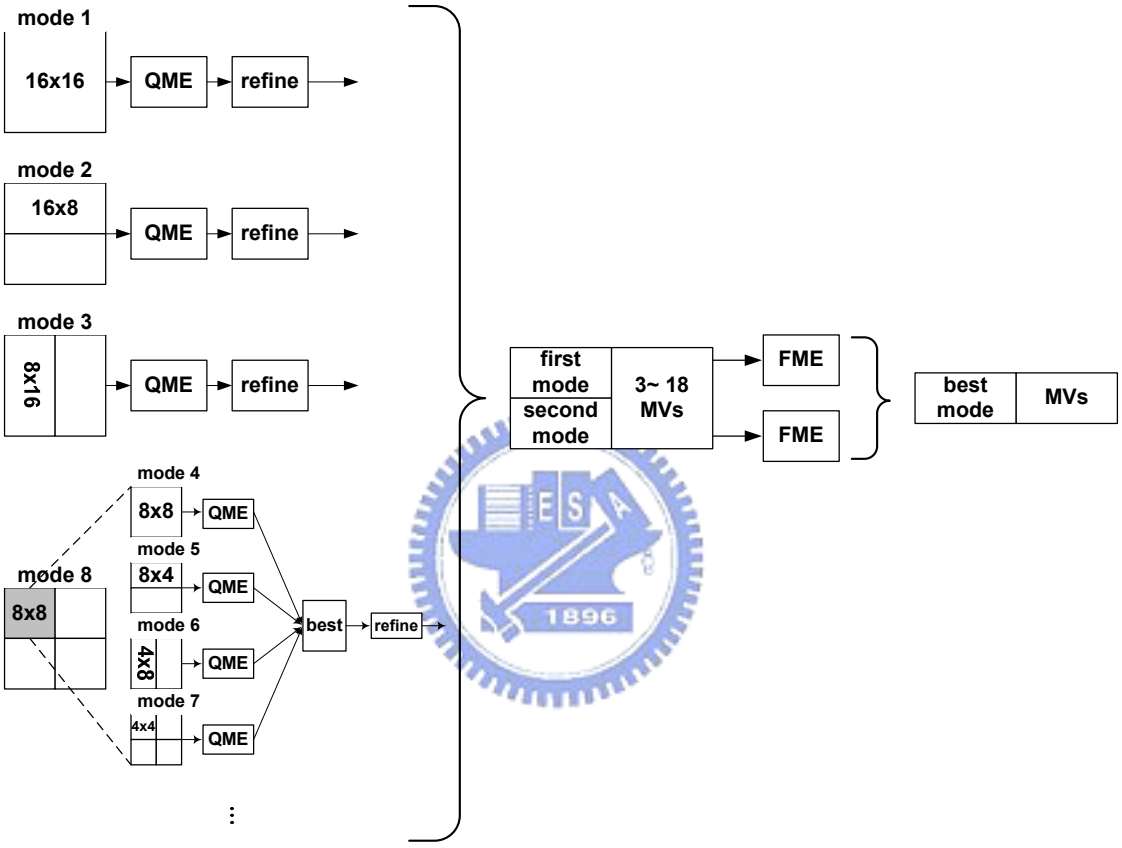

Figure 34 the RQME algorithm

Furthermore, though many ME designs have been proposed, few are proposed for variable block size [23], [26]. Though these designs have different 1-D or 2-D array architectures, they are all based on the full search block matching algorithms. The full search algorithm can provide better performance, but the resulted hardware cost such as the 2-D array based design is large, almost half or more of the total encoder area [26]. On the other hand, if fewer processing elements (PEs) such as the 1-D array are used to save the cost, the cycle counts will increase dramatically. Thus, the hardware cost and performance cannot have a good tradeoff in current designs.

Motivated by above problems, in this chapter, we propose a hardware oriented fast variable block size IME algorithm and its architecture that has lower hardware cost, cycle counts and is also pipelining balanced to FME. This design is based on our previous proposed fast algorithm, Quarterly Motion Estimation (QME) [7]. QME uses the 4:1 subsampling that can reduce the hardware cost and cycle counts significantly but also preserve the regularity of full search like scan pattern. However, direct application of QME to VBSME will result in significant quality degradation since 4:1 subsampling in the small 4x4 block will lead to two fewer samples and thus less accurate result. To avoid such problem, we introduce a refinement process to compensate the quality loss. The hardware architecture is also redesigned to support the variable block size. Besides, the various combinations of block size are reduced to only two before the final FME stage (called mode filtering). Thus, only 3 to 18 MVs instead of 41 MVs are refined in the FME. Using this mechanism we can save about 80% FME in average, and leads to a significant decrease of cycle counts. With above approaches, the area cost and cycle counts of IME is reduced by 58.5% and 52% respectively when compared with the previous design. The resulted IME design (520 cycles) is also more balanced to FME (568 cycles) [29].

6.2. Refined Quarter Motion Estimation (RQME)

QME adopts the 4:1 subsampling as shown in Figure 35. The current block is divided into four quarter fields while the search range is divided into two column fields. QME computes the matching cost on these two column fields (called quarter sum of SAD (QSAD)). When computing the matching cost, the two column fields in the search range are used for computation with the first field of current block (only the Quarter Field 1 is needed). Each field in the search range is for different search point. For example, search point (0, 0) will use the pixels “A” from Even Column Field, (0, 1) will use the pixels “C” from Even Column Field, (1,0) uses the pixels “B” from Odd Column Field, and (1,1) uses the pixels “D” from Odd Column Field. With this, every search point in the search range

buffer will be tested but only one fourth of pixels in the block are used to calculate the matching cost. This algorithm can efficiently reduce the computing complexity with similar performance to that of the full search algorithm. However, when applied to VBSME, 4:1 subsampling will result in too few samples for a 4x4 block and thus has significant quality degradation. Otherwise, the QME method has an inclination to participate larger block size into smaller block size. Thus, we enlarge the motion vector cost during the computation of IME SAD. With this refinement, we overweight the cost for small size block, such that we can effectively balance the error prediction by QME without any other hardware cost.

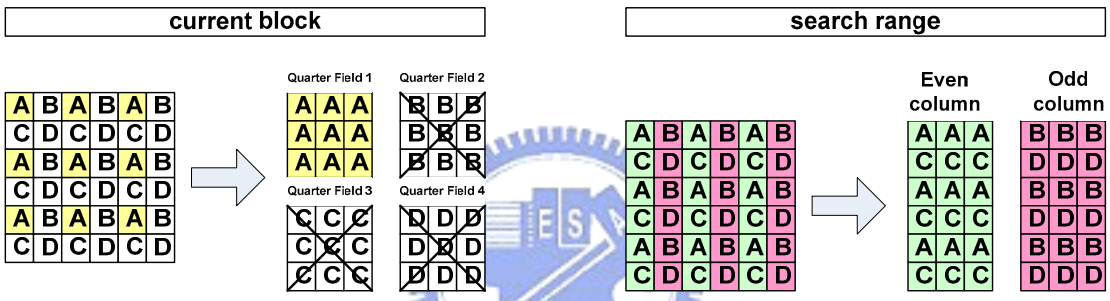

Figure 35 the partition of current block and search range

6.3. Mode Filtering

After computing the cost of all possible block sizes, we select the most possible two modes for FME (mode filtering) instead of all modes for FME, as shown in Figure 34. The reason to choose two modes is that, in the previous chapter [30], the two modes selection has the highest improvement in terms of mode numbers in hardware design. The most possible two modes are selected from one of 16x16, 16x8, 8x16 and the best of the 8x8 and subblock. For the case of 8x8 and subblock size, each 8x8 block will has its own best mode (8x8, 4x8, 8x4 or 4x4) based on QSAD. The cost of the four 8x8 blocks are summed together to compare with other modes. Finally, two best modes are further refined by FME. With this, only 3 to 18 MVs instead of 41 MVs are calculated in FME, which can save about 80% FME computing cycles. In [25] a similar but more complex

procedure has also been proposed. However, our method can achieve better quality and lower cycle count than that in [25]. It is because that we only select two instead of three candidates and only the best candidate for the 8x8 and subblock case is considered in the final best mode selection.

6.4. Performance Analysis

In order to correctly analysis the performance of RQME, we partition the section into two parts. The first one is to analysis only the RQME design, and the other one is to analysis the combined RQME with adaptive skip mode detection we have introduced in Chapter 4.

6.4.1. Performance of MF+RQME

TABLE XIV presents the simulated performance of the proposed algorithm compared with the full search algorithm used in the reference software. The search range is [-15, 16], and the reference software is JM 9.0 [20]. The test sequences are CIF sized, including: Akiyo, flower, football, and table tennis. The average R-D curves of these sequences are shown in Figure 36.

TABLE XIV MF+RQME performance for CIF sequences

QP		Mode Filtering (MF)	RQME	MF+RQME
16	Δ *PSNR (dB)*	-0.08	-0.02	-0.09
	Δ *Bit-rate (%)*	0.07	-0.68	1.63
20	Δ *PSNR (dB)*	-0.08	-0.02	-0.10
	Δ *Bit-rate (%)*	0.31	-0.45	2.15
24	Δ *PSNR (dB)*	-0.12	-0.03	-0.13
	Δ *Bit-rate (%)*	-0.48	-0.27	1.54
28	Δ *PSNR (dB)*	-0.11	-0.03	-0.12
	Δ *Bit-rate (%)*	-0.74	0.15	1.09
32	Δ *PSNR (dB)*	-0.09	-0.03	-0.11
	Δ *Bit-rate (%)*	-0.49	0.45	1.18
36	Δ *PSNR (dB)*	-0.09	-0.03	-0.12
	Δ *Bit-rate (%)*	-0.86	0.82	1.07
Average	Δ *PSNR (dB)*	-0.10	-0.03	-0.11
	Δ *Bit-rate (%)*	-0.36	0.00	1.44

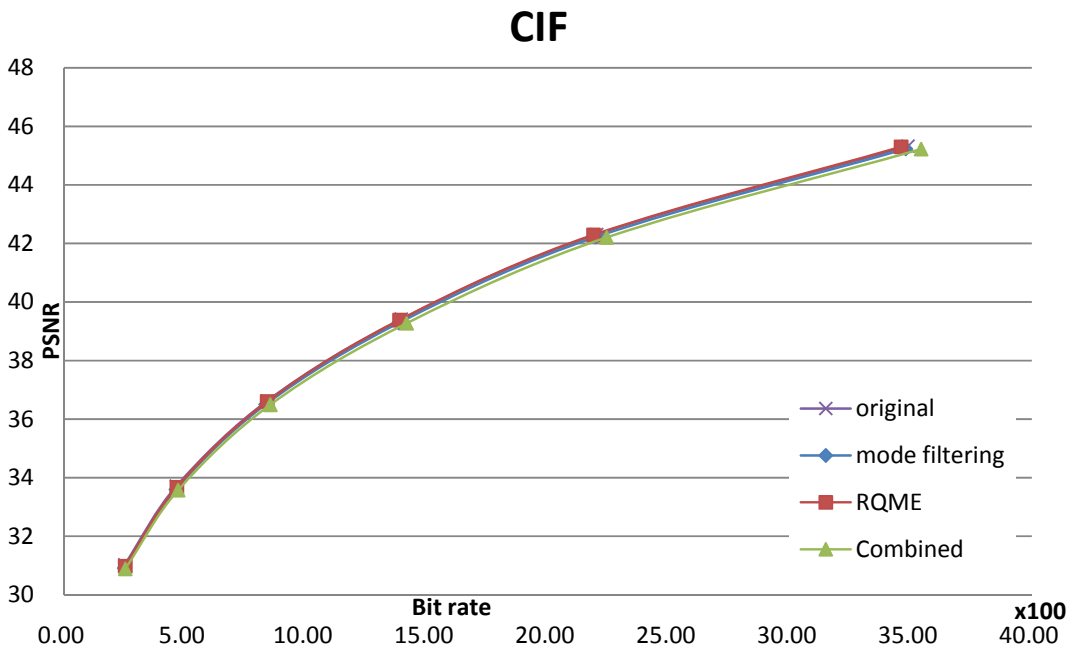

Figure 36 the averaged R-D curve of proposed algorithm

For the method of mode filtering, we can find that the PSNR degradation is small and even the bit-rate is lower than original; however, it is with the penalty of PSNR loss. Besides, the bit-rate overhead for mode filtering is smaller for high QP case which means the low bit-rate condition. The reason for this is that the mode filtering algorithm prefers to select larger block size, and larger block size is also preferred under the low bit-rate condition because of fewer MV bits.

For the refined QME algorithm, the PSNR decrease is smaller than 0.03 dB. If the refinement step is skipped, the bit rate increase will be serious. In terms of bit-rate increase, the refined QME works well under low QP, which is high bit-rate condition. It means the refined QME prefers to select smaller block size which fits the condition of high bit-rate.

From above analysis, these two methods are complementary to each other so that the performance of the combined method is under an average level which means the PSNR degradation is below 0.14dB and the bit-rate increase is below 2.15%. It works well when comparing with other fast motion estimation algorithms for H.264.

6.4.2. Performance of Skip+MF+RQME

Firstly, we test three different types of QCIF sequence to analysis the design which combines with three algorithms we proposed: adaptive skip mode detection, mode filtering and refined quarter motion estimation (Skip+MF+RQME). The test environment are: three 300-frame QCIF sequences (silent: low motion sequence, carphone: medium motion sequence, and mobile: high motion sequence), the search range equals 8, 1 reference frame, coding type: IPPP, no rate-distortion optimization and disable thresholding.

All of the analyses are compared with JM9.0 [20]. The R-D curves of the three sequences are shown in Figure 37 - Figure 39, and TABLE XV **s**hows the average performance, speed-up and pre-skipped MBs. The performances are not very well for low QP conditions; it is because the mode filtering performs badly for small size sequences especially for low QP conditions. While the mode filtering the skip mode detection perform better in high QP, the performance is better for high QP sequences.

TABLE XV Skip+MF+RQME performance for QCIF sequences

	PSNR	Bit rate	Speed up	Pre-Skip rate (%)
QP12	-0.14	2.30	1.28	0.17
QP16	-0.16	2.03	1.45	7.83
QP20	-0.20	2.46	1.56	13.89
QP24	-0.28	0.87	1.76	19.70
QP28	-0.32	-0.59	1.94	22.09
QP32	-0.32	-2.08	2.23	27.19
QP36	-0.30	-3.00	2.57	37.02
average	-0.24	0.28	1.83	17.41

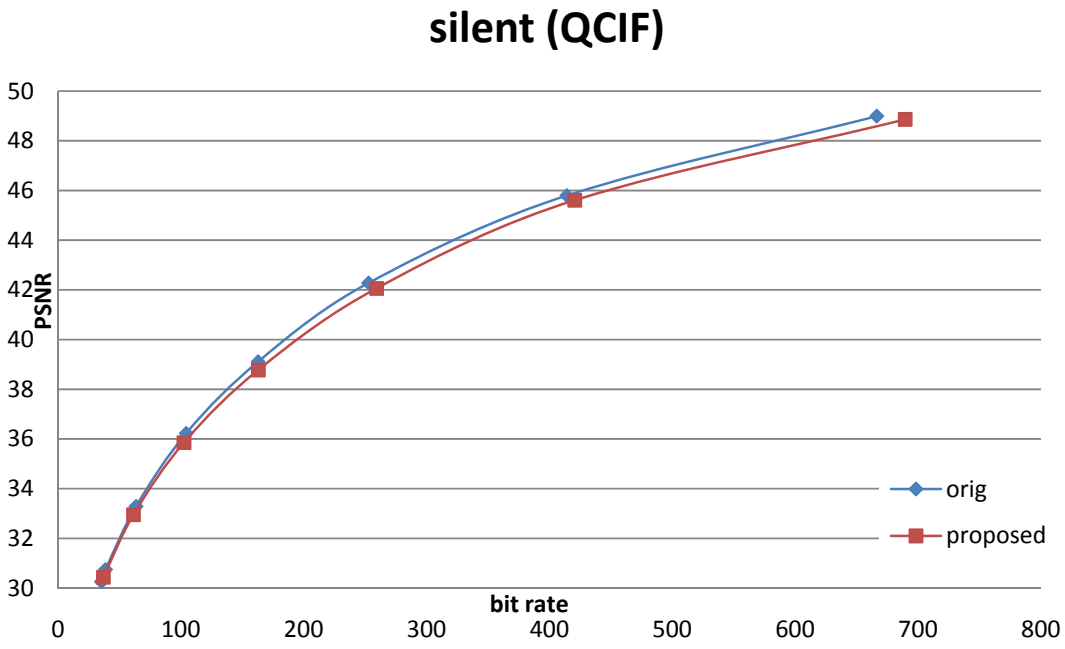

Figure 37 Skip+MF+RQME performance of QCIF silent (low motion)

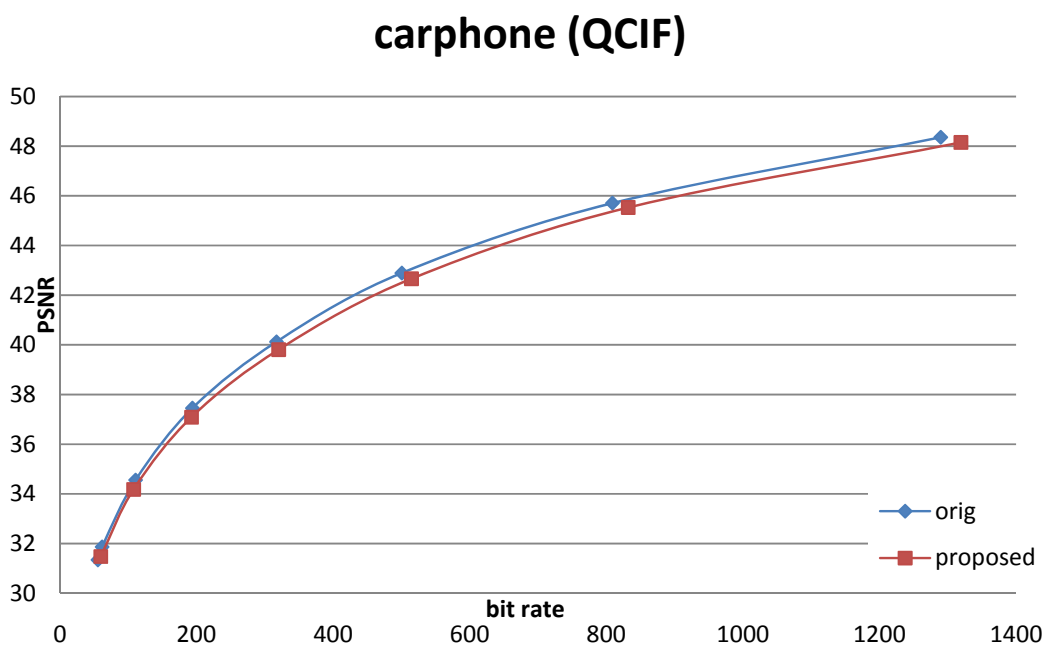

Figure 38 Skip+MF+RQME performance of QCIF carphone (medium motion)

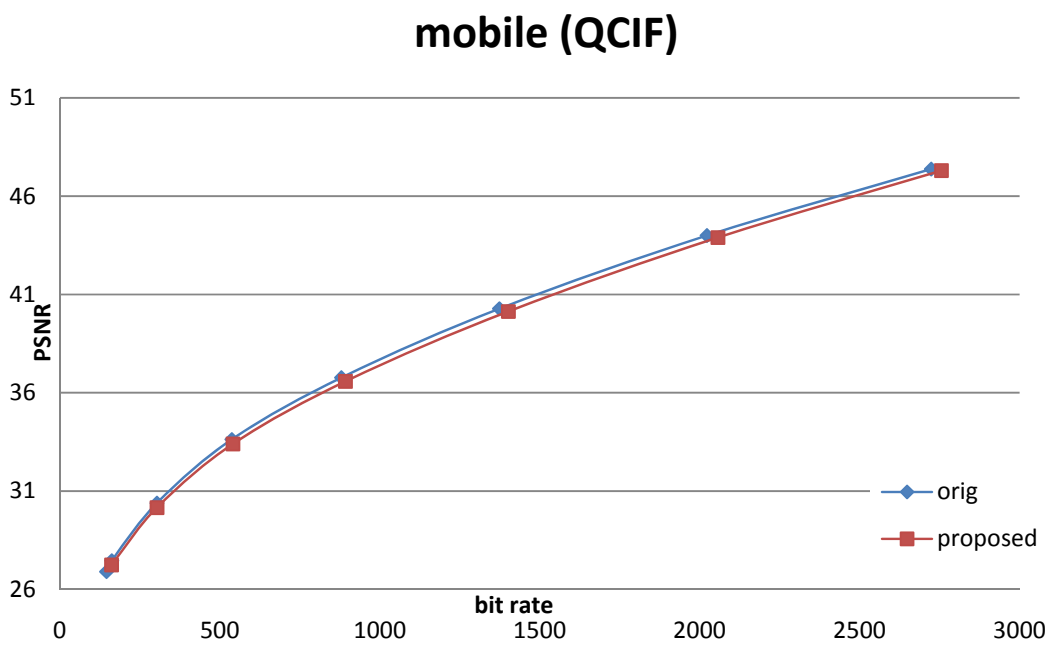

Figure 39 Skip+MF+RQME performance of QCIF mobile (high motion)

Secondly, we also test three CIF sequence to analysis the design. The test environment are: three 300-frame CIF sequences (akiyo: low motion sequence, container: medium motion sequence, and hall: high motion sequence), the search range equals 16, 1 reference frame, coding type: IPPP, no rate-distortion optimization and disable thresholding.

The R-D curve of the three sequences are shown in Figure 40-Figure 42, and TABLE XVI shows the average performance, speed-up and pre-skipped MBs. By seeing the R-D curve, the performances are quite good in CIF sequences; it is because the mode filtering and refined quarter motion estimation do well for CIF sequences. Besides, for high QPs, although the PSNR drop by 0.46dB, the bit rate decreases enormously to 10.22 (%). It is because our adaptive skip mode detection can skip lots of almost skipped MBs and save many bit rates. It is demonstrated by that we pre-skip 82.65(%) MBs in average under QP36. It shows that our design is especially suitable for CIF size video.

TABLE XVI Skip+MF+RQME performance for CIF sequences

	PSNR	Bit rate	Speed up	Pre-Skip rate (%)
QP12	-0.15	-2.90	1.57	25.30
QP16	-0.15	-1.17	1.85	15.51
QP20	-0.14	0.30	2.34	24.49
QP24	-0.19	-2.50	2.75	35.96
QP28	-0.27	-6.28	3.71	58.77
QP32	-0.40	-10.22	5.02	73.97
QP36	-0.46	-8.94	6.37	82.65
average	-0.25	-4.53		

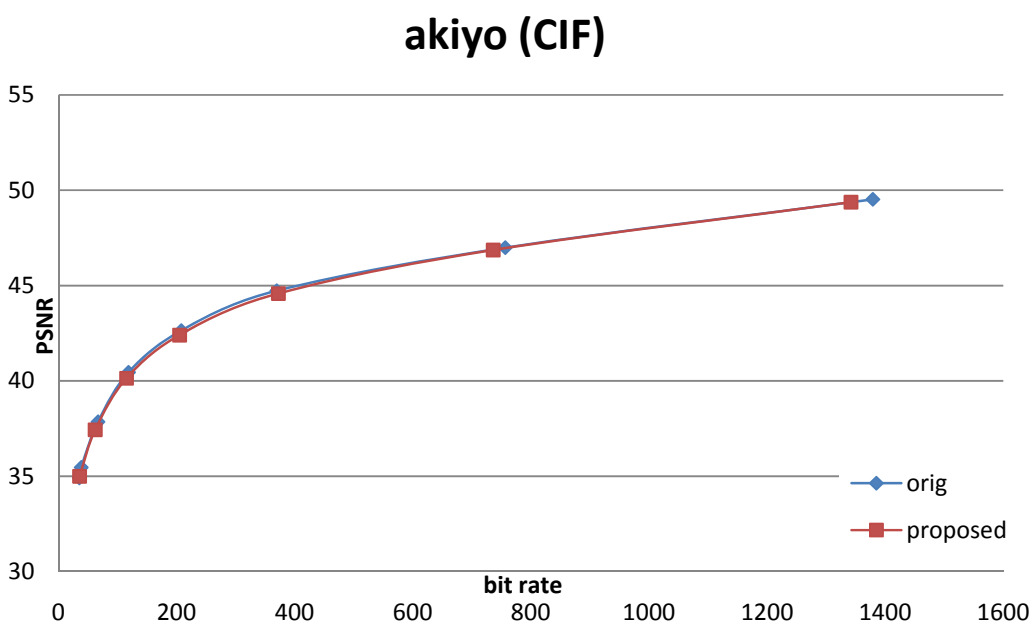

Figure 40 Skip+MF+RQME performance of CIF akiyo (low motion)

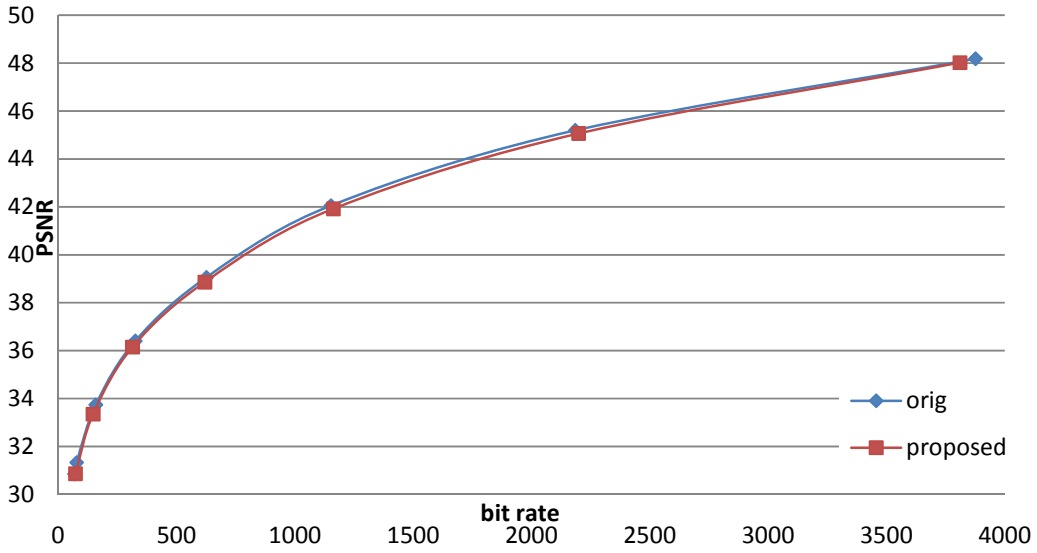

Figure 41 Skip+MF+RQME performance of CIF container (medium motion)

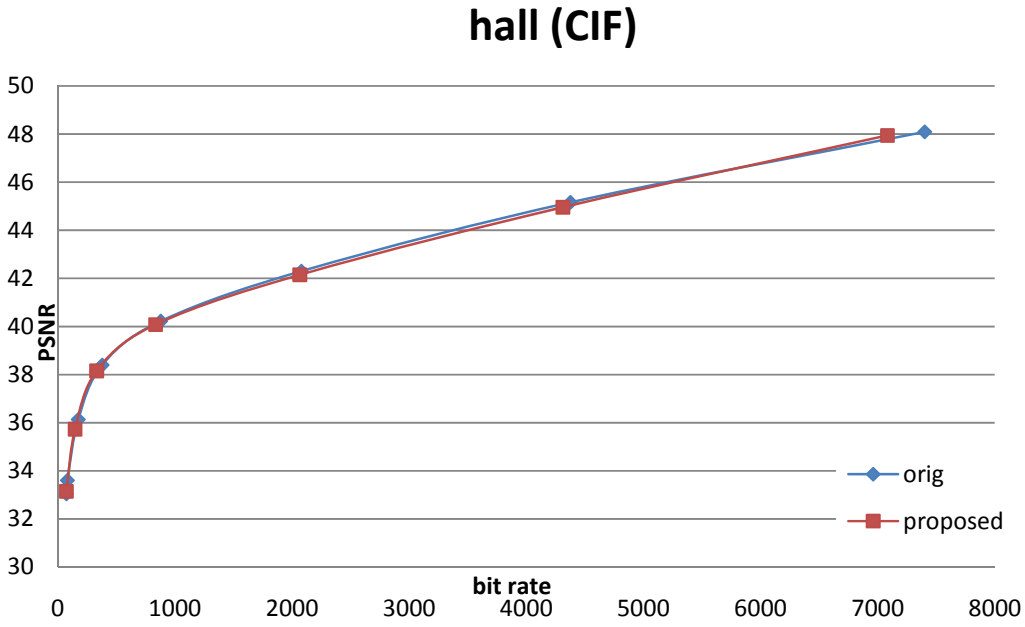

Figure 42 Skip+MF+RQME performance of CIF hall (high motion)

6.5. The RQME Architecture

Figure 43 shows the proposed architecture that follows the algorithm. Thus, the current block is divided into four quarter fields and the reference frame is divided into even column and odd column fields. Both of the two fields of search range are used for computing while only the first field of current block is adopted for QSAD computation. This enables parallel computation by two parallel QME units, one for each column field. With this, the computational complexity can be reduced by 75%. After computing QME for each search point, only the smaller one will be kept to find the best search point.

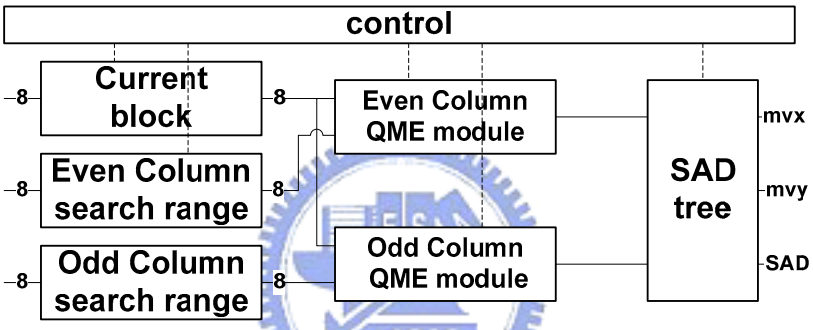

Figure 43 the proposed architecture

For each QME unit in Figure 44, four Row QME modules are included. A 16x16 current block is divided into four row subsets and each subset is computed by a Row QME module. The architecture of a ROW QME module is illustrated in Figure 45. In our design, each process element (PE) unit in the Row SAD module is responsible for SAD computation of two pixels and each PE unit contains two basic units which are used to compute the absolute difference as shown in Figure 46.

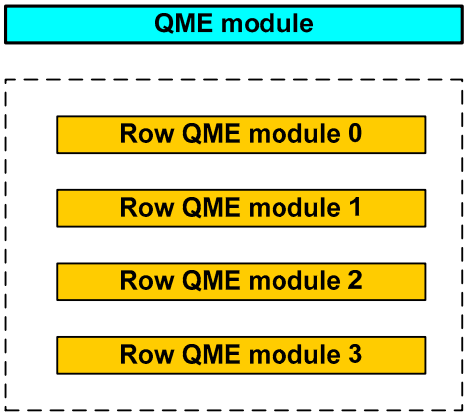

Figure 44 the structure of a QSAD module

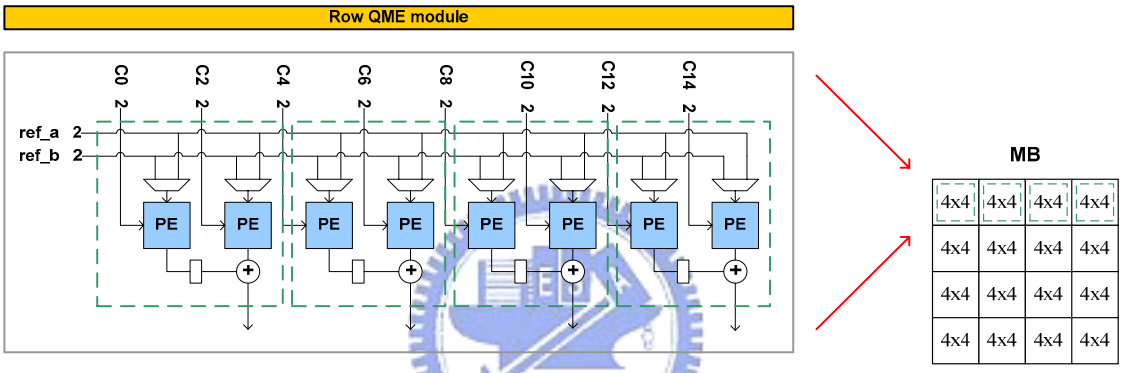

Figure 45 the structure of an Row QME module

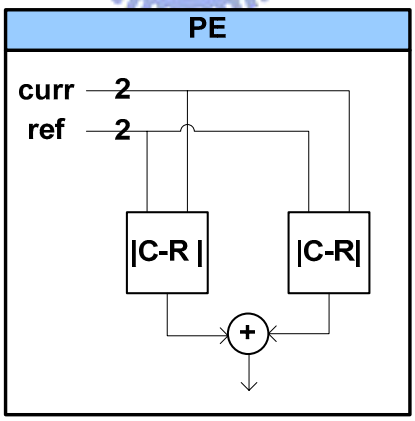

Figure 46 the structure of a PE

In the Row SAD module design, the current block data is directly gotten from the current block buffer, while the two reference data are broadcasted to the eight PE modules in every cycle.

Figure 47 shows the data flow of even column field of the proposed architecture. The timing diagram of odd column field is the same. The dotted line in the figure shows the timing to select different broadcasted reference data to main the fully pipelined data flow.

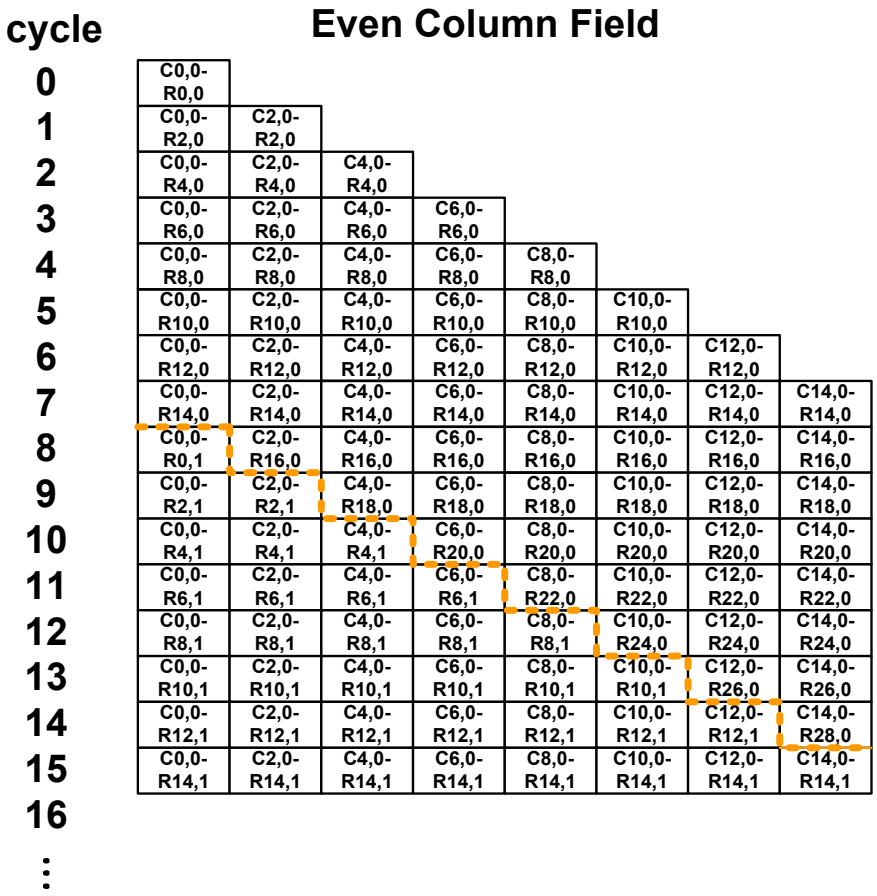

Figure 47 the data flow the even row

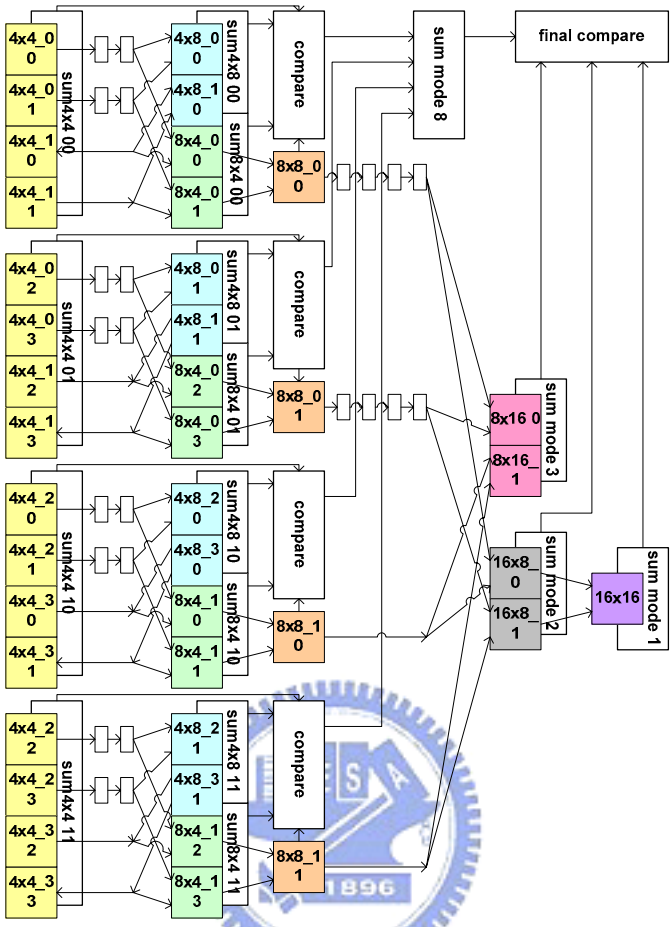

Figure 48 the structure of the SAD tree

The QSAD for block size larger than 4x4 is computed by the SAD tree as shown in Figure 48. This SAD tree is different from the traditional SAD tree as in other VBSME designs. First, this is a skewed tree which can minimize the necessary registers in a delay line. Second, this SAD tree needs to compute the total cost for the four 4x4-blocks, two 8x4-blocks, two 4x8-blocks within one 8x8-block and the sum of the four 8x8 subblock mode, two of 16x8-blocks, two 8x16-blocks within one MB. The reason is that our algorithm needs to decide the best mode of each 8x8 block and the best two modes of an MB. This function is only activated when finishing all the search points for lower power consumption. With above data flow and architecture, our proposed architecture can complete the IME calculation with 520 cycles for search range [-15, 16], which is roughly balanced to the cycle count of FME (568).

6.6. Hardware Implementation Result

The proposed design has been designed by using Verilog HDL and synthesized by 0.13 um CMOS cell library. TABLE XVII shows the implementation result and its comparisons with other designs. The search range of the design is different for CIF and QCIF sequences. Search range [-16, 15] is support to CIF sequences while [-8, 7] is enough to support QCIF sequences. Therefore, the latency is longer (520) for CIF whereas shorter (136) for QCIF sequences.

The design in [26] is based on 1-D array so that the latency is long. The design of [23] is based on the 2-D array. Comparing with these 2-D architectures, the PE number of our architecture is half so that the gate count is almost half compared with the three ones. In term of latency, our method speed up two times than that of in [23] with half of hardware cost. The major contribution for such improvement is the adopted fast algorithm instead of full search as in others.

Besides, we define two parameters:

- ✓ $\text{Throughput} = \frac{\text{search points}}{\text{latency}} * \text{frequency}$
- ✓ $\text{Area efficiency} = \frac{\text{throughput}}{\text{gate counts}}$

'Throughput' means the number of search points generated in every second and 'area efficiency' is the elevation of how much search points are contributed by every unit hardware cost. The throughput of our design is 393.8, and the area efficiency is 8.95. It shows our design outperforms other's design and gets an effective tradeoff between hardware cost and video quality.

TABLE XVII RQME comparisons with previous works

	Ref [23]	Ref [26]	J.M.'s [27]	Chen's [28]	**Ours**
Search Range	32x32	16x16	N/A	64x32	**32x32 / 16x16**
Latency	1089	4096	N/A	N/A	**520 / 136**
Process (μm)	0.35	0.13	0.13	0.18	**0.13**
Voltage (V)	N/A	1.2	1.0	1.3	**1.2**
Gate Count	106K	61K	N/A	63.54K	**44K**
Max Frequency (MHz)	66.67	294	13.5	27	**200**
Throughput	62.66	18.375	N/A	N/A	**393.8**
Area efficiency	0.59	0.30	N/A	N/A	**8.95**
Power (mW)	737.32 @ 66.67MHz for 720x480 30fps	23.76 @ 11 MHz for QCIF 30fps	3.28mW@6.75MHz for CIF 30fps 0.41mW @ 0.85MHz for QCIF 15fps (without subblock)	1.40 @ 13.5 Mhz for CIF 30fps (ultra low power mode)	**55.63mW@200MHz** **0.59mW @1.62MHz for CIF 30fps (low power mode)** **0.08mW @0.2MHz for QCIF 15fps**
Search pattern	Full Search	Full search	Gradient decent	Parallel-VBS Four step search	**Full search Quarter SAD**

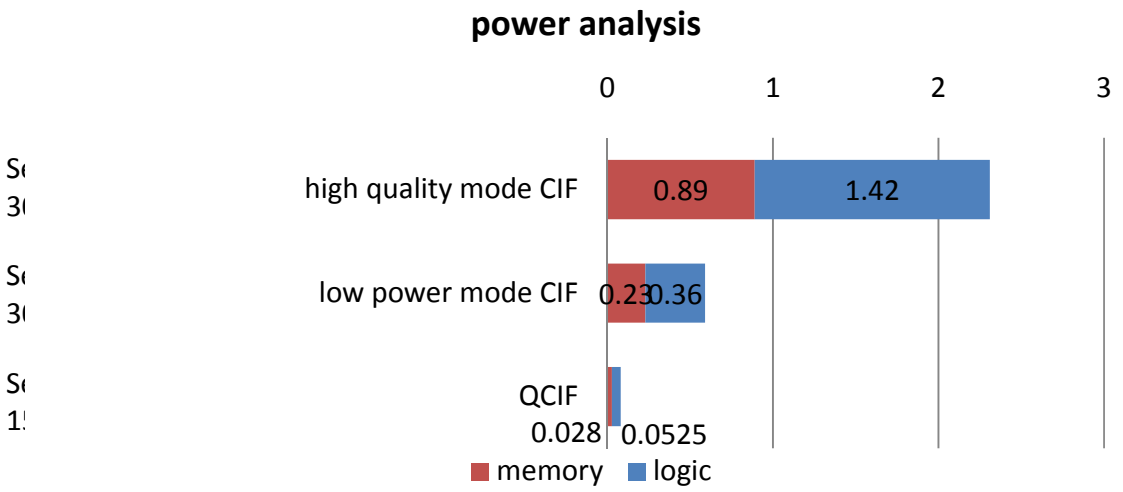

Figure 49 power analysis of RQME design

Figure 49 is the power analysis of our design, we support high quality mode and low

power mode for CIF sequences. For high quality mode, we consume 2.31mW@6.2MHz for search range 32x32. For low power mode, we consume 0.59mW@1.6MHz with search range 16x16. For QCIF sequences, the power consumption is 0.08mW@0.2MHz. Because of our very low operation frequency and quarter pixel algorithms, our design consumes less power than other design such as [27] and [28]. Thus, our design is more suitable for power constricted portable devices.

6.7. Summary

In this chapter [31], we propose a set of algorithm and architecture for search range [-16, 15] integer motion estimation in H.264/AVC. The algorithm uses the mode filtering to reduce the computational load of fractional motion estimation and refined QME to enable four times of parallel processing while lower computational complexity and low quality loss. The proposed architecture only needs half of PE counts and lower latency than the general 2-D architectures. The power consumption of our design in only 0.59mW for CIF and 0.08mW for QCIF; therefore our design is suitable for medium frame size application such as mobile phone, personal digital assistant (PDA) or video camera.

7. Efficient Large Search Range Motion Estimation for High Definition Video Compression

The section presents a hardware-efficient fast algorithm and its architecture for large search range motion estimation (ME) used in HDTV sized H.264 video coding. To solve the high cost and latency in large search range case, the proposed algorithm processes ME in parallel multi-resolution levels instead of serial process in the previous approach. This enables high data reuse for lower bandwidth and low memory cost. Further combining with our previous proposed mode filtering and bit truncation, the algorithm only increases the bit rate within -1.62% and 1.20% and 0.04dB and 0.07dB PSNR degradation in average for 720p and 1080p sequences respectively. The hardware implementation can save up to 48.9% of area cost and 55.1% of memory cost compared to the previous approach for large search range to [-128, 127].

7.1. Introduction

Video compression technique becomes more and more important while the development of mobile video device and HDTV is growing up. H.264/AVC, the latest video standard is well adopted in HDTV and other application since it provides high video quality and excellent coding efficiency. The high efficiency is achieved by several new coding tools such as variable block size integer motion estimation (VBSME). However, so much complex IME become the bottle neck when we want to realize a real-time encoder.

Although many VLSI realizations of VBSME have been widely proposed to speed up the ME process, most of them are only applicable for SDTV size or below. For HDTV size applications that requires large search range up to [-128, 127] or even larger; however, previous approaches will consume too much area cost and computational cycles.

To support large size research range, lots of fast integer ME algorithms have been proposed. For fast IME, various approaches have been proposed such as [8], [16], [32], but few can be readily applicable to large search range as used in HDTV. The large search range requirement will result in longer execution cycles as well as large buffer and high memory access. Previous design [16] with [-63, 64] search range uses the full search method and thus occupies large area cost. Besides, a modified three-step algorithm is used in [33] to decrease the search points for low power, but still consumes large area cost and memory. Nevertheless, most of them are not suitable for hardware implementation due to nearly prohibited memory bandwidth resulted from the poor data reuse flow. To solve this problem, one promising approach is the multi-resolution ME as one proposed in [8], and the conventional multi resolution algorithm is shown in Figure 9. In this design, they use three hierarchical levels for search and refine the motion vector from the coarse level to the finest level. However, this approach has several disadvantages for hardware implementation. First, the motion vector found in the higher level needs to be further refined in the lower level. It means the search is a sequential process that will increase the cycle counts, and decrease the hardware utilization and throughput. Second, a full search range sized buffer is still needed because the dependency between the three hierarchical levels. It greatly increases the hardware costs for large search range design and diminishes the benefits of multi-resolution ME. Third, the required bandwidth is still quite large due to poor data reuse of the refinement process.

To solve above problem, in this chapter, we proposes a parallel multi-Resolution ME (PMRME) algorithm and its architecture. The proposed algorithm uses three independent levels for search. The first two levels with data subsampling cover the large search range to find the rarely occurred large search vector. These two levels have good data reuse by fixing the searching center at (0, 0). On the other hand, the third level without data subsampling covers the search range with the most occurred MV. This level has the search center at the motion vector predictor. The concept behind our algorithm

is the unequal distribution of motion vector that most of them are near the motion vector predictor. Thus, a fine search around the motion vector predictor can find the most of motion vector while the rest of motion vector can be found in the coarse search. With above approaches, we can save at least 92.4% memory buffer compared with the previous method (for the search range 128). Besides, data within two out of three memory buffers are highly reused, and thus can save about 64.8% of memory bandwidth.

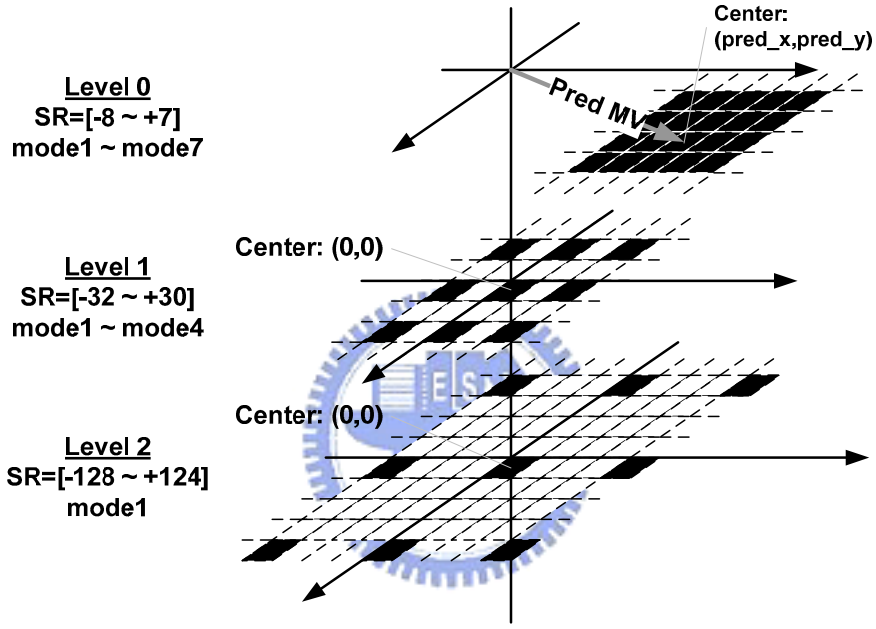

Figure 50 the three level parallel multi resolution motion estimation

7.2. Parallel Multi-Resolution Motion Estimation (PMRME)

PMRME includes three levels and all of them are independent to each other, as illustrated in Figure 50. In the coarsest level 2, the SR is the largest, [-128, 124], and this level is centered on the original point (0, 0). This enables regular memory reuse between successive MB processing. This level uses the 16:1 sampling and thus we only choose the 16x16 mode (mode 1 in TABLE I) since other modes will contain too fewer pixels for SAD calculation.

In level 1, the SR is reduced to [-32, 30] and also centered on (0, 0) for the same reason as level 2. This level uses the 4:1 sampling and thus we only choose the 16x16 to 8x8 mode (mode 1 to 4 in TABLE I) since other modes will contain too fewer pixels for SAD calculation.

In the finest level 0, the SR is set to [-8, 7]. However, unlike the other two levels with (0, 0) center, we choose the predictive motion vector (MVP) as the center due to its higher probability for final MV. Thus, we do not subsample data in this level and thus enable search for all variable block size modes.

The characteristic of the three levels are different and they can properly complement to each other. Level 2 provides a large search range for high motion blocks with coarse precision. It is useful for very high motion blocks, and can find a good enough though approximate motion vector candidate. Also, the level 1 can provide a medium search range but a finer precision. The level 2 and level 1 are complementary to each other. With these two large search levels, the algorithm can rapidly converge for the motion search of the level 0 by effects of MVP as shown in Figure 51, thus undoubtedly level 0 will provides better performance. If only the level 0 is used, it is difficult to trace the high motion blocks because the MVP cannot follow up the real motion effectively in this case.

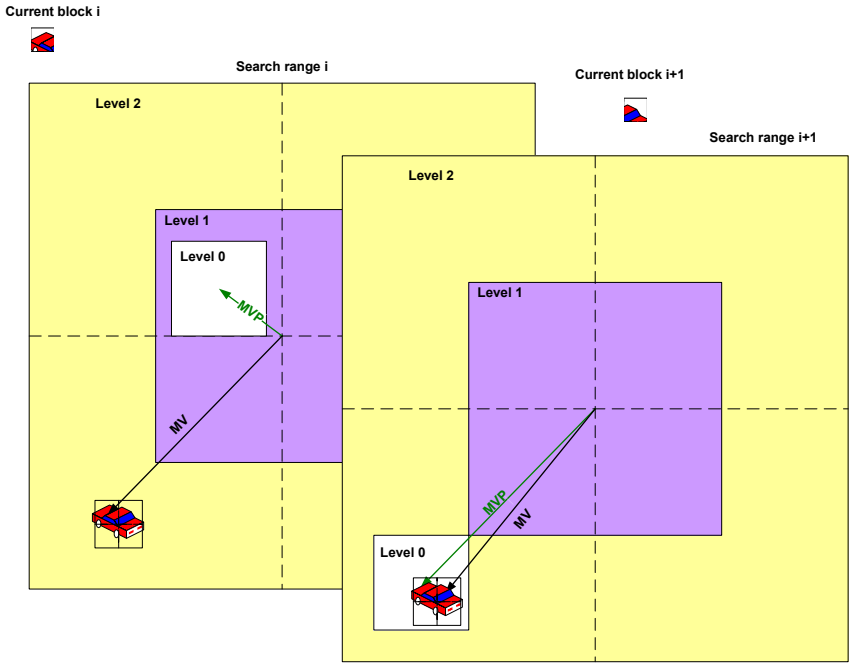

Figure 51 the concept of PMRME
(the motion vector can be rapidly converged)

7.3. Mode Filtering

To further reduce the complexity, we use our previously proposed mode filtering method [30] (see chapter 0). Only two modes in the integer ME stage will be passed to the fraction ME stage to significantly reduce the fractional ME cycle. Besides, the method also increases the overall ME pipelining efficiency.

7.4. Bit Truncation

Two degrees of bit truncation [34] are used in our design. In our analysis (as in TABLE XIX and TABLE XX), five bits precision is enough to provide a good coding efficiency for the 720p sequences. However, at least six bits is needed for the 1080p sequences because of the very high definition characteristics. In our analysis, five bits for 1080p will cause larger bit rate increasing.

By using the bit truncation method, about 38% and 28% hardware cost are saved for “5 bits precision” and “6 bits precision” respectively.

7.5. The PMRME Architecture

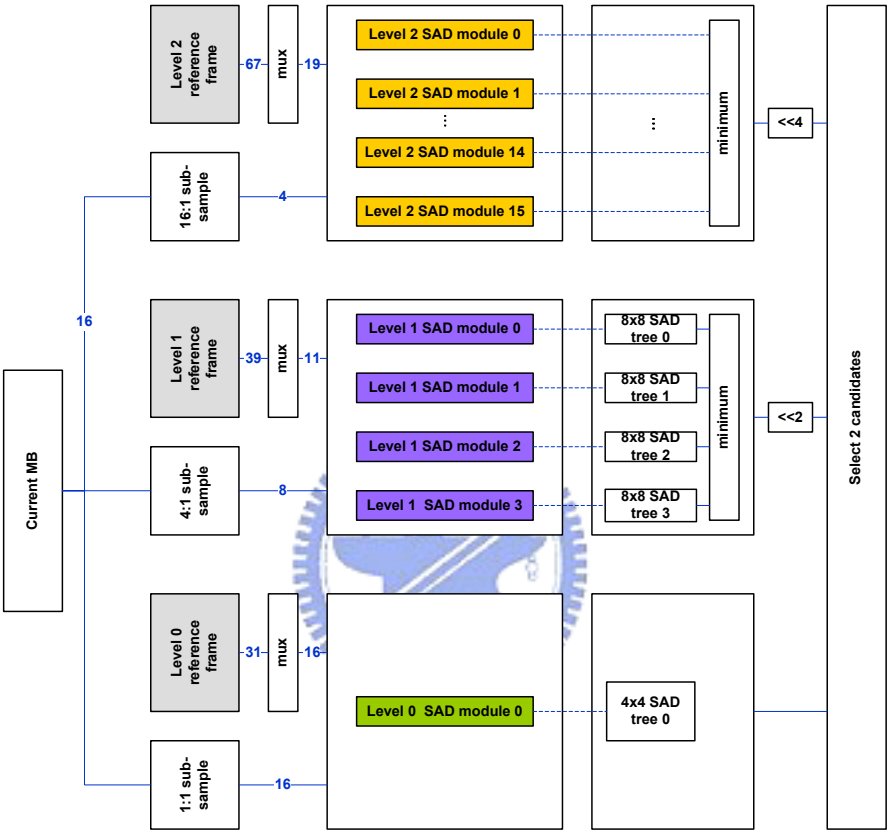

Figure 52 the proposed PMRME architecture
(the number on the line is the number of pixels)

Figure 52 shows the proposed architecture and one 16x16 current block data is shared for the three levels. In this architecture, all computations are decomposed as the combinations of 4x4 blocks, denoted as “primitive module” and this is the basic module to compute the SAD of a 4x4 block as depicted in Figure 53. Each primitive module is capable of processing the data of a 4x4 block within four cycles (four stages pipelined) because there are four SAD modules in a primitive module, and each SAD module is used to sum the SAD of four horizontal pixels and previous SAD. The architecture of a

SAD module is a combinational logic and it is depicted in Figure 54. The reference selection module is for the sake of choosing different broadcasted reference data to raise hardware utility; more detail will be explained in the following section.

With the primitive modules, every level can be easily implemented by regular hierarchal composed module. The hierarchy of motion estimation hardware is “SAD module” > “primitive module” > “Row ME module” > “ME module.” In this way, “Level 0 ME module” contains four “Level 0 Row ME module”, and each “Level 0 Row ME module” has four “primitive modules” and can process the data of a 16x16 macroblock in full resolution (1:1). In brief, level 0 has 16 primitive modules for one macroblock, as in Figure 55.

Because level 1 is 4:1 subsampled, only four primitive modules are needed for one “Level 1 ME module” for one search point, as in Figure 56. In addition, to further speedup the processing, we adopt four “Level 1 ME module” in parallel (see Figure 52). Another reason to parallelize the four “Level 1 ME module” is that we can reuse the overlapped data among these modules. Thus, totally 16 primitive modules are used as that in level 0. Similarly, for level 2 with 16:1 subsampling, only one primitive module is needed for one “Level 2 ME module” for one search point, as in Figure 57. We also speedup the level 2 by using sixteen “Level 2 ME module” in parallel (see Figure 52), thus further highly reuse data in these modules. Therefore, in level 2, we also have the same primitive modules as level 0. Last but not the least, the main reason for such speedup in level 1 and level 2 is to balance the computation cycles for different levels. Thus, computations for all levels can be done in the same 256 cycles with different resolutions and search range.

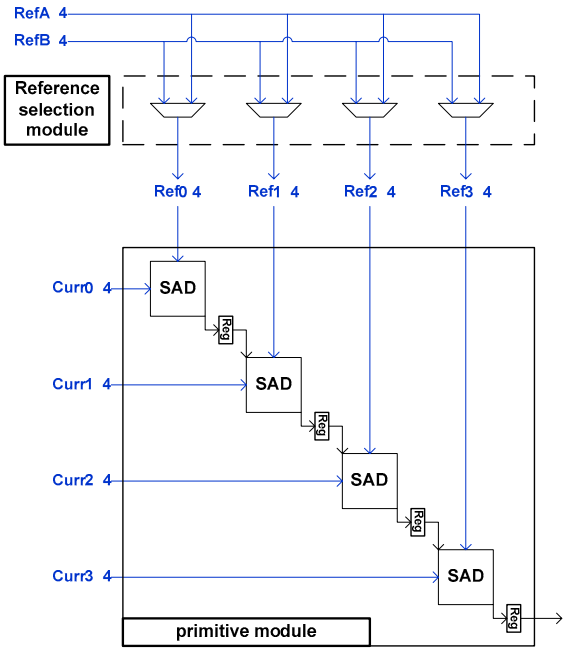

Figure 53 the primitive module

RefA and RefB are input for reference data, Cur0~Cur3 is the input for current block data (the number of on the line means the number of pixels)

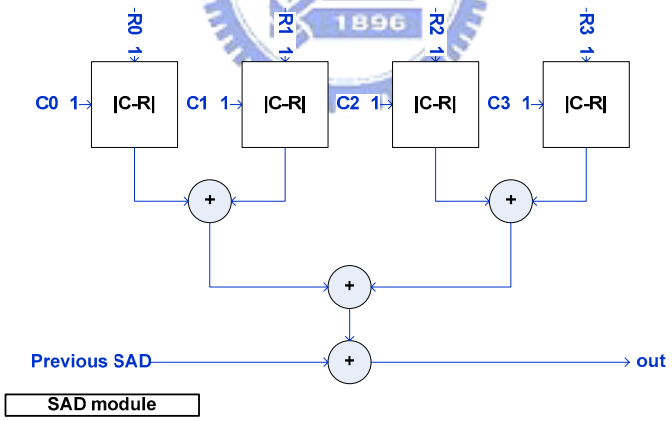

Figure 54 the architecture of SAD module

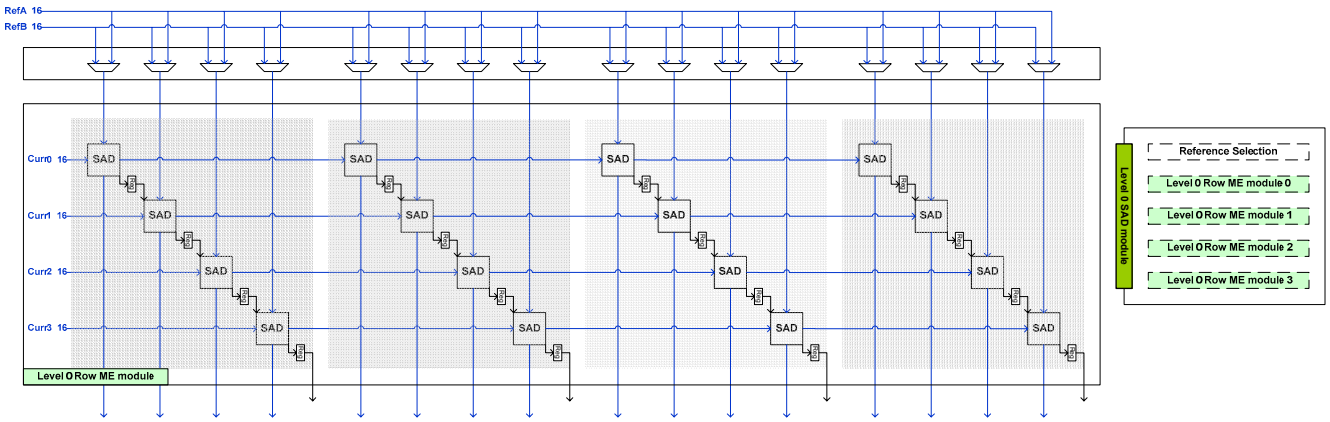

Figure 55 Level 0 ME module

It contains four “Level 0 Row ME modules”, each Row module further contains four “primitive modules” and can process a 16x16 macroblock in full resolution

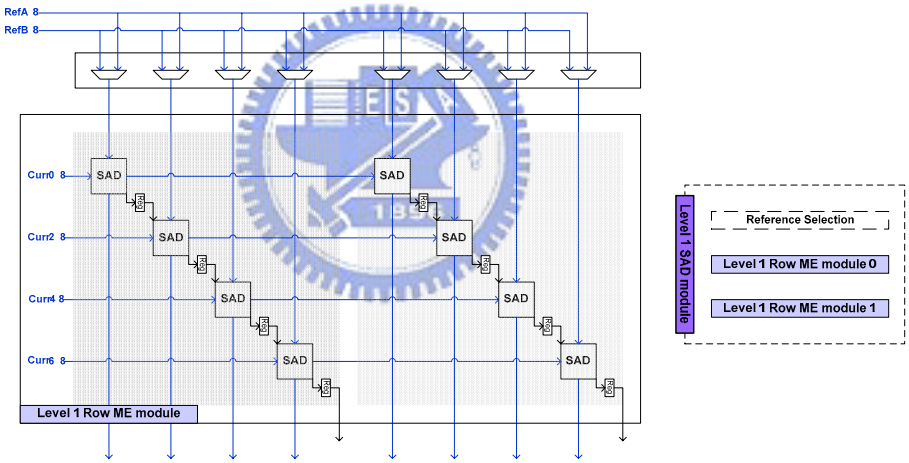

Figure 56 Level 1 ME module

It contains two “Level 1 Row ME modules” and can process one 16x16 macroblock in 4:1 resolution

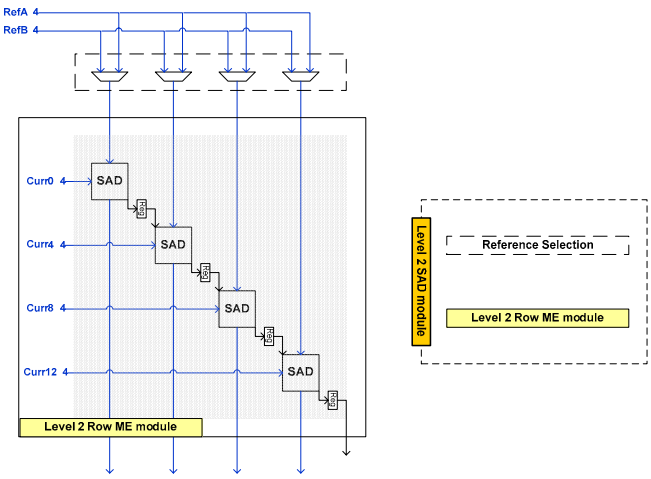

Figure 57 Level 2 ME module

It contains only one “Level 2 Row ME module” and can process one 16x16 macroblock in 16:1 resolution

After getting the SADs form primitive modules of each block, these SADs are further summed up to generate the SADs of different block sizes. For level 0, it has the most complex summation trees for combination of the seven kinds of block types and the architecture of the “4x4 SAD tree” is shown in Figure 58. As for the level 1, four “8x8 SAD tree” are used for combination of the mode 1 to mode 4 block types because of the four times parallelism as illustrated in Figure 59. However, in level 2, only comparators and registers (pipelined to decrease the critical path) are needed to select the minimum SAD cost. Finally, according to mode filtering, the selection module will choose the best two SAD costs from different levels for the fractional ME module.

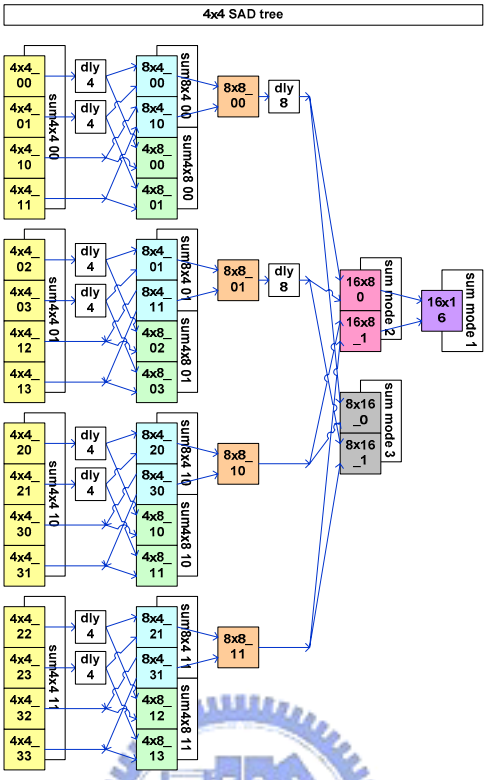

Figure 58 the “4x4 SAD tree”

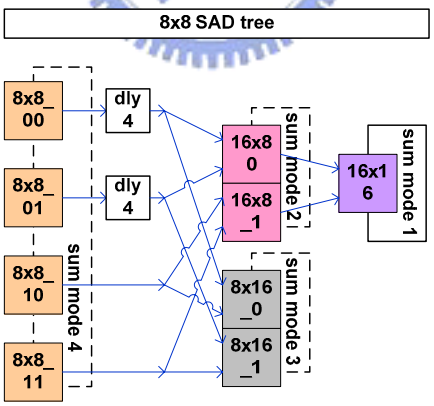

Figure 59 the “8x8 SAD tree”

7.6. Search Scheduling

The whole scheduling is described below. To ease explanation, we first partition the data into several “row packages”. Figure 60 shows the search flow in level 0. For the current MB, we separate it into 16 row packages; while the reference data are cut into many overlapped row packages (16x31, 16 is because of the search range [-8, 7] and 31 comes from the number of words of reference A (16) add the number of words of reference B (15). See the next section).

The search process is done by fed these row packages to motion estimation modules. The search schedule is from top to down and left to right; therefore, in this level, the search point of (-8,-8) will be calculated first, and the (-8,-7) in the next. After finishing the search points of first column (x=-8), it goes the next column (x=-7) until all of the search points are calculated.

The data flow for current and reference row packages are different: for current block the row packages are “stay” and for the reference data the row packages are “broadcasting”, while the results (SAD) are “propagated” as shown in Figure 61.

Figure 62 is the pipelined search schedule of level 0. C0~C15 is the current row packages of a macroblock (see Figure 60), and RefA and RefB stand for reference row package in the search range, while each of them are mapped into the corresponding input ports of “level 0 ME module” (Figure 55). In the first cycle (0^{th} cycle), we get the SAD of [C0, R (-8,-8)]. In the following cycle (1^{st} cycle), we will get the SAD of [C0,R(-8,-7)] and {[C1,R(-8,-7)]+[C0,R(-8,-8)]}. Therefore, in the 4^{th} cycle, we will get {[C3,R(-8,-5)]+[C2,R(-8,-6)]+[C1,R(-8,-7)]+[C0,R(-8,-8)]} which is the SADs of “4x4_00 block, 4x4_01 block, 4x4_02 block, 4x4_03 block” (see Figure 5 and Figure 58) of search point (-8,-8). This four 4x4 SAD results are connected to the “4x4-SAD tree” (Figure 58) to wait to accumulate the SADs for other block types. In the same way, in the 8^{th} cycle,

we get the result of “4x4_10 block, 4x4_11 block, 4x4_12 block, 4x4_13 block”. In the 12th cycle, we get the result of ““4x4_20 block, 4x4_21 block, 4x4_22 block, 4x4_23 block”. At last, in the 16th cycle, the “4x4-SAD tree” can attain all of the sixteen 4x4 SADs, and after a few cycles propagation in the tree, we can have the 41 SADs of various block types of search point (-8,-8).

As shown in Figure 62, to maintain the full pipeline of data, in different cycle counts, different primitive modules should choose different reference row packages from RefA and RefB. The control of this part is determined by reference selection module (Figure 55). Because the search flow is fully pipelined, we can get the result of a search point in each cycle (except for the previous 15 cycles).

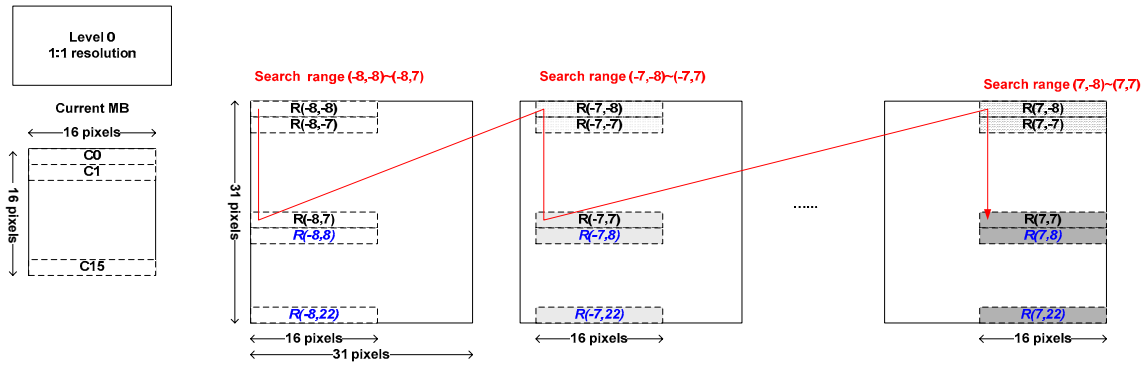

Figure 60 the reference control of level 0

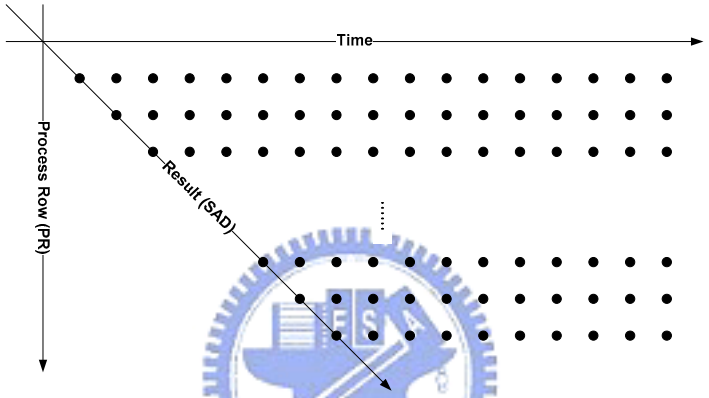

Figure 61 data flow direction of level 0 ME (time-space representation)

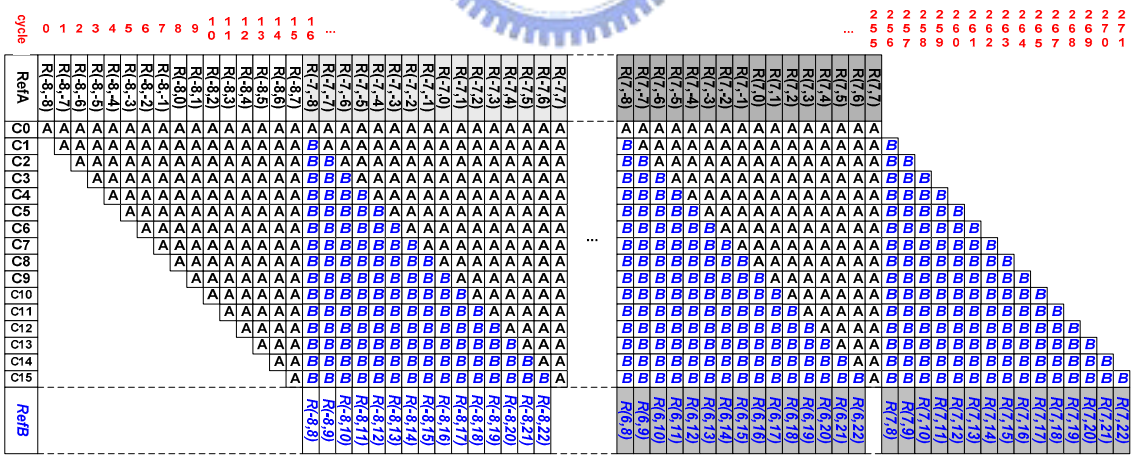

Figure 62 the pipelined search schedule of level 0

Figure 63 is the pipelined search schedule of level 1. The main idea of search flow is alike the same way as level 0 except some other aspects. First, the quantity of cycle counts which is needed to complete one search point is halved (8 cycles instead of 16) because of the 4:1 subsampling. Second, the results of four adjacent search points can be generated at the same time, and it is because we utilize the four times parallelism in this level. With these two features, we can search the motion vector in a region sixteen times larger. Another benefit of the parallelism is macroblock level data reuse, because we can reuse overlapped data in these neighboring search points. For example, [C0, R (-32,-32)] are capable of dealing with the partial SAD of search points (-32,-32), (-30,-32), (-28,-32), (-26,-32) at the same time, as in Figure 64.

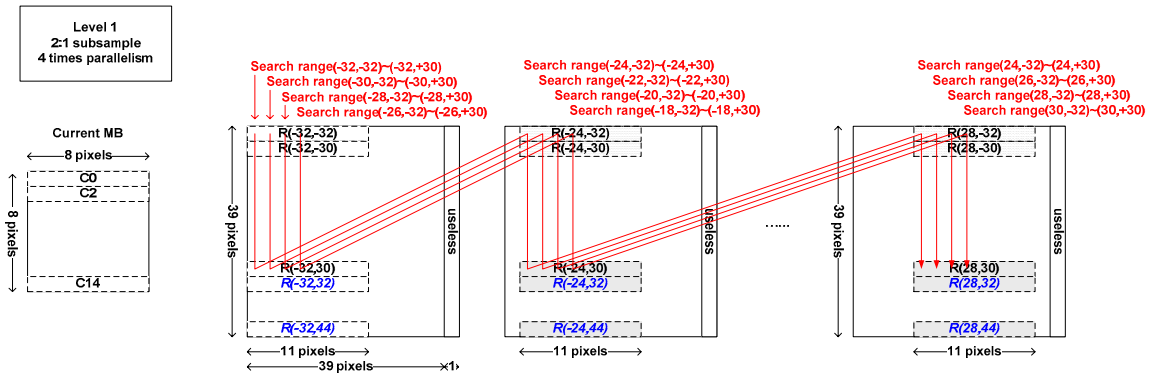

Figure 63 the search flow of level 1

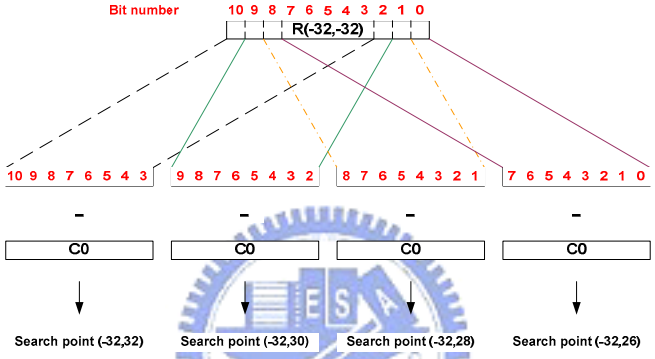

Figure 64 parallel data reuse in level 1

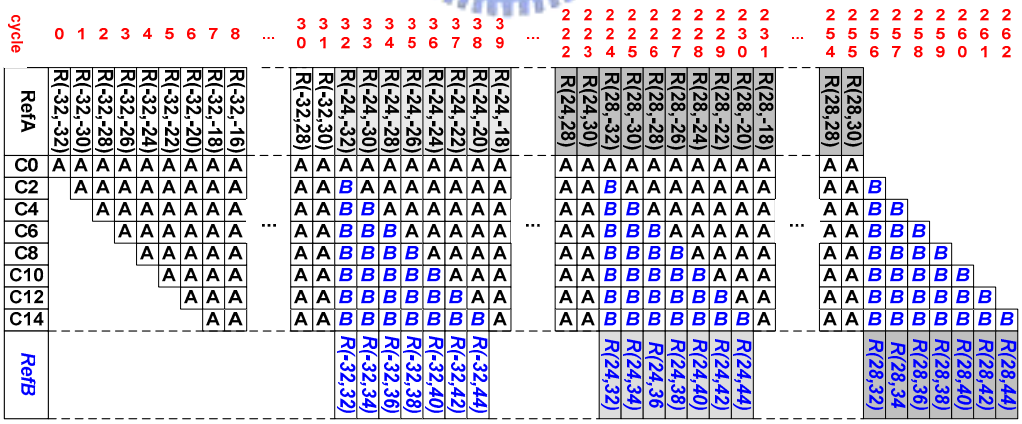

Figure 65 the reference control of level 1

Figure 66 is the pipelined search schedule of level 2. The method of search flow is very likely the same as level 1. The differences are: first, it just needs four cycles to complete one search point. The resolution is 16:1 with 16 times parallelism and the reference data of sixteen neighboring search point are further efficiently reused. These features give the level 2 has the ability to search sixty-four times larger region than level 0 with the same cycles consumed (256).

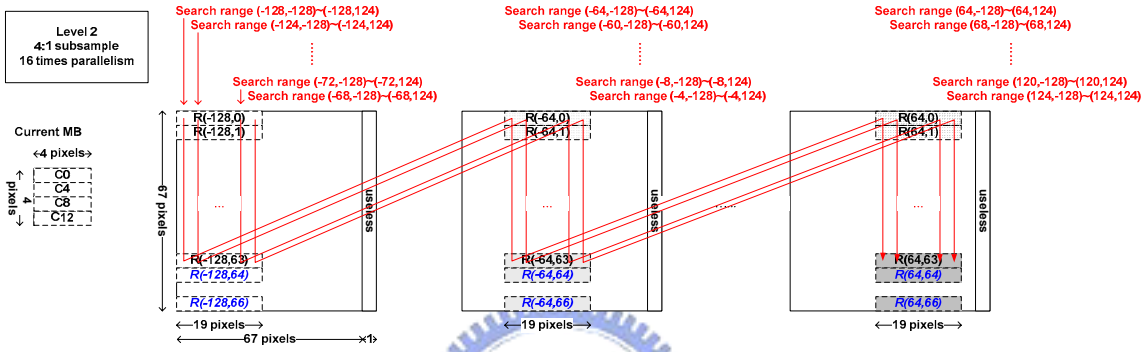

Figure 66 the search flow of level 2

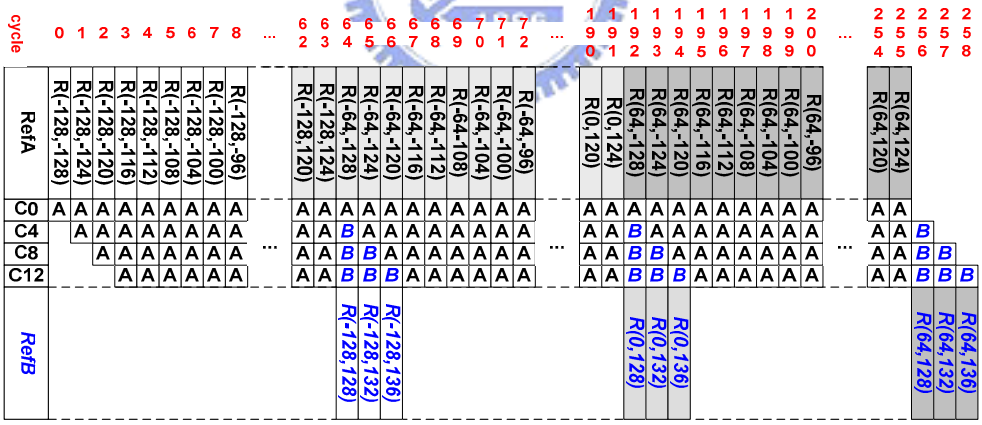

Figure 67 the reference control of level 2

7.7. Memory Allocation

The memory allocation of level 0 is shown in Figure 68. In order to save the memory bandwidth, we propose to share and reuse the reference data of level 0 for IME and FME. In this figure, the 3-pixels stripe surrounds the memory is prepared for fractional ME, while it is not used in integer ME phase. The reason for this is that if the motion vector falls in to the boundary of the search range (-8,-7,-6, 5, 6, 7 out of search range -8~7), the interpolation process of FME needs extra three pixels beyond the search range. Therefore, preventing from the data miss in FME phase, we preload the three-pixel boundary data.

Figure 68 (a) is the reading order of motion estimation computation. It means the order of acquiring reference row package begin with the stripe [0~15] from top to down and the next of the stripe of [1~16] and so on. Figure 68 (b) is the division of bank, the level 0 memory is cut into two parts, and each part contains three banks. The two parts is used for realizing fully pipelined data flow for motion estimation as illustrated in Figure 62. The memory bandwidth in this level equals the memory size in the level, which are 37x37=1369 bytes. It is because that in this level, the search region is around MVP; therefore we have to refresh all the data of search region for every macroblock.

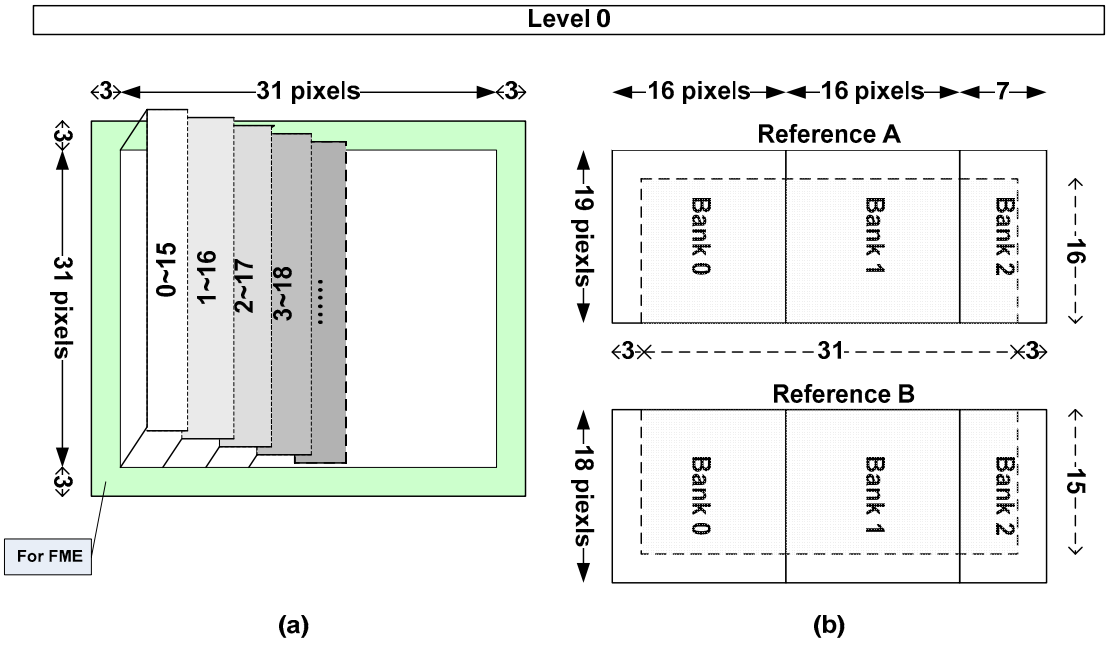

Figure 68 memory allocation of level 0
(a) is the read order of the memory; (b) is the allocation memory bank

The memory allocation of level 1 is shown in Figure 69. The order of reading reference data is shown in Figure 69 (a). The width of the reading strip (11) comes from the width of a macroblock (8 instead of 16 because of 4:1 subsampling) add with the additional three times parallelism (total in 4 times parallelism). More explicitly, the partial data of [0~7], [1~8], [2~9], [3~10] of stripe [0~10] are proceeded at the same time for adjacent search points as illustrated in previous section. Figure 69 (b) is the memory bank allocation of level 1. The width of every bank is the width of one macroblock (8), and we only have to refresh a bank (both refA and refB) for every macroblock. Because the search region is center around (0, 0), the data of other four banks can be reused for the next macroblock.

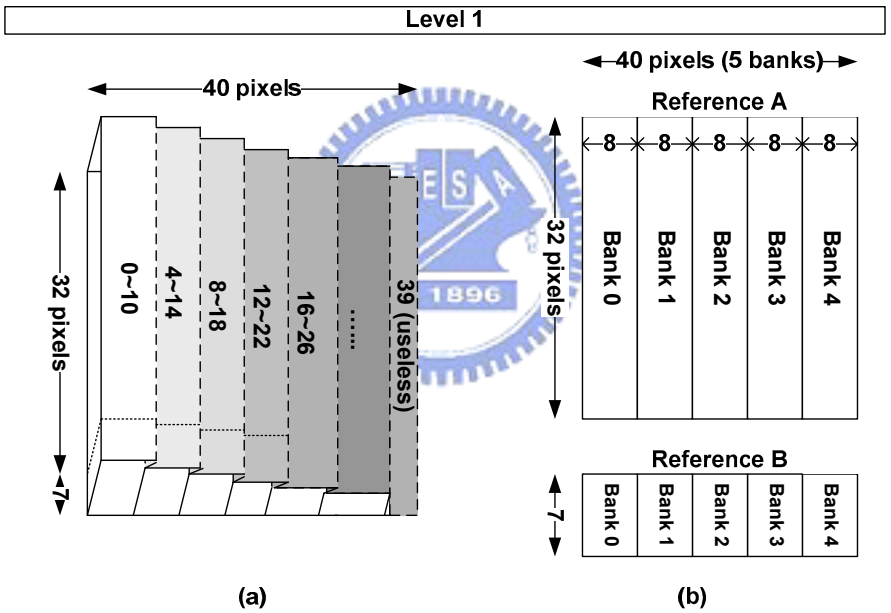

Figure 69 memory allocation of level 1
(a) is the read order of the memory; (b) is the allocation memory bank

The memory allocation of level 2 is shown in Figure 70. The order of reading reference data is shown in Figure 70 (a). The width of the reading strip (19) in level 2 comes from the width of a macroblock (4 instead of 16 because of 16:1 subsampling) add with the additional fifteen times parallelism (total in 16 parallelism). Alike level 1, the partial data of [0~3], [1~4], [2~5], [3~6] …… [15~18] of stripe [0~18] are proceeded at the same time for adjacent sixteen search points. Figure 70 (b) is the memory bank allocation of level 2. The reason for the bank division is the same as level 1, and we only have to update the data of one bank for every macroblock.

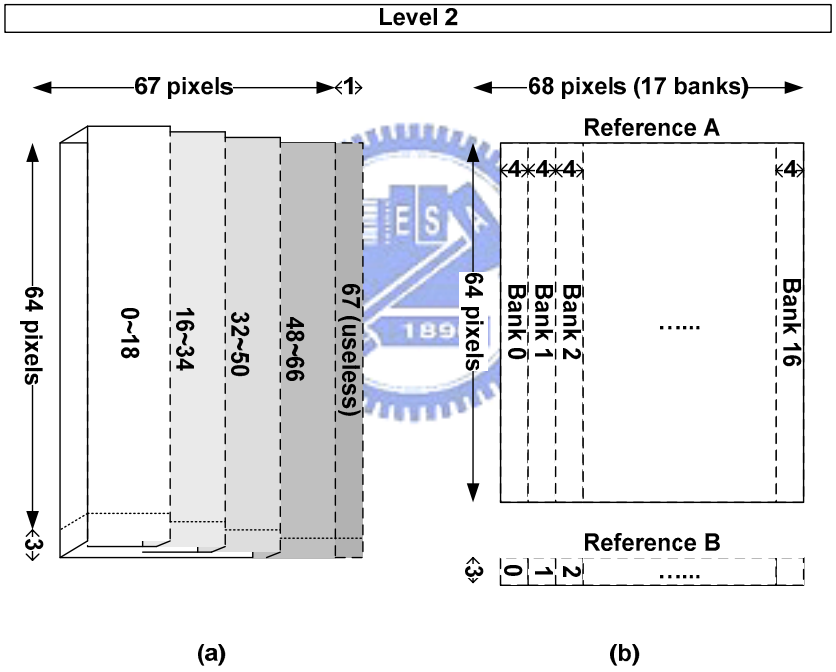

Figure 70 the memory allocation of level 2
(a) is the read order of the memory; (b) is the allocation memory bank

The degree of data reusability is shown in Figure 71, our design is a highly data reuse design and it can significantly diminish the external memory bandwidth and internal memory size. The memory size and bandwidth for three reference frame buffers are listed in TABLE XVIII. The bit width of memory buffer of level 1 and level 2 are also truncated, while that of the level 0 is not. The reason for this is that the level 0 data can be reused by the following fraction ME hardware if the best motion vector falls in the level 0.

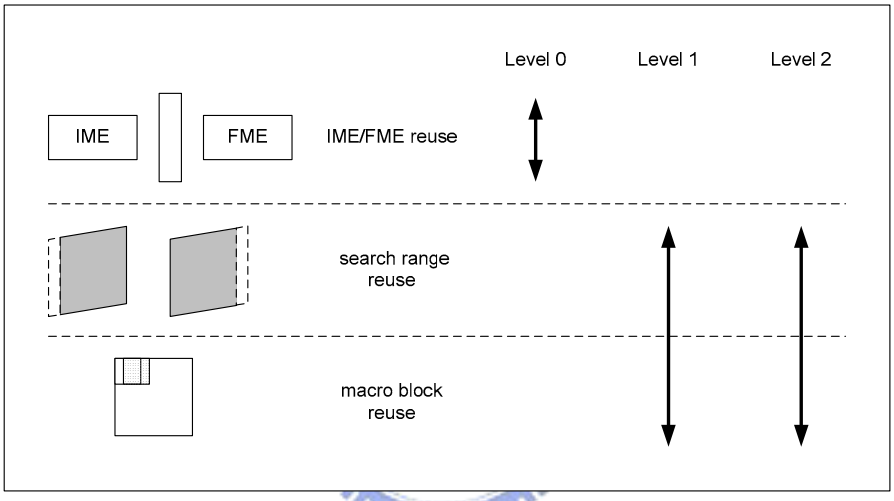

Figure 71 the data reusability degree in different level

TABLE XVIII memory and bandwidth for different frame size

Memory cost	for 720p		for 1080p	
	buffer size	BW(per MB)	buffer size	BW(per MB)
Level 0 (Kbyte)	1.369	1.369	1.369	1.369
Level 1 (Kbyte)	0.975	0.312	1.170	0.312
Level 2 (Kbyte)	2.8475	0.268	3.417	0.268
Total (Kbytes)	**5.1915**	**1.949**	**5.956**	**1.949**
Direct design	**73.712**	**4.336**	**73.712**	**4.336**
Saving (%)	92.95	55.1	91.91	55.1

7.8. Memory Schedule

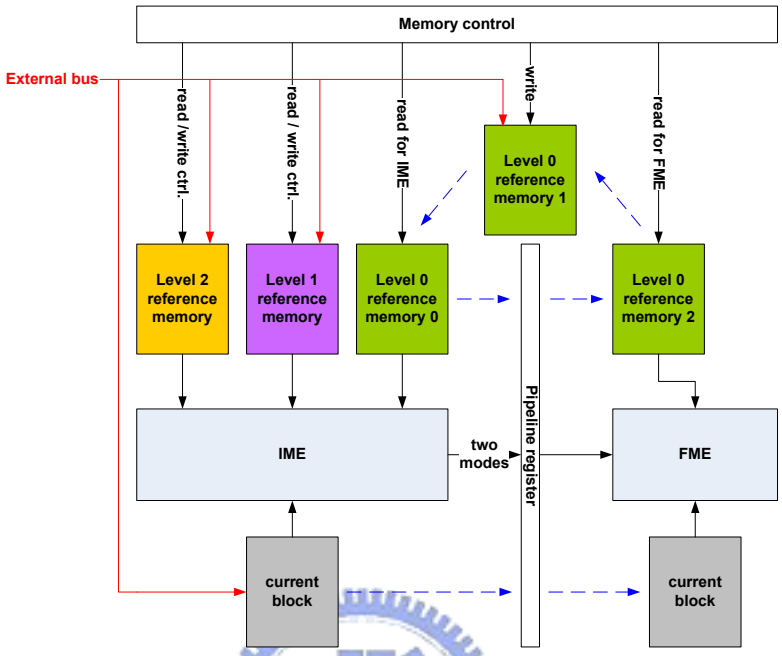

Figure 72 the block diagram of IME and FME

Figure 72 shows the total block diagram of the full ME modules. The IME module has three internal SRAMs for reference pixels storage. Each one is corresponding to one level of reference pixels. When one stage is completed, the current macroblock information in IME is moved to FME in one cycle. The information includes motion vectors, block type, subblock type and the current macroblock pixels. Moreover, the reference pixels in level 0 SRAM needs to be moved to FME. However, instead of moving data, we use three SRAMs as the level 0 buffer and swap them with a ping-pong buffer concept. The three SRAMs includes one for IME level 0 reference, one for FME, and one for loading new data from external memory. Whenever the IME stage completes the coding of the first MB, the SRAM for level 0 reference for the first MB is changed for FME reference in the next stage. At the same time, the SRAM for current FME reference is changed to load the data of the third MB from external memory for further use. The SRAM that is now filled the reference data for the second MB is switched for IME level 0 reference. With above ping-pong buffers, we can share the

level 0 data of IME with the FME, and no additional memory access time is necessary. In our ME design, if the motion vector from IME is around motion vector predictor, FME can use the reference pixels in IME reference SRAM. However, if the motion vector is far away from motion vector predictor, FME has to load the reference pixels from external memory. Thus, to reduce the data loading traffic, the IME stage uses the mode filtering strategy and gives only two block types for FME refinement. To further reduce the loading time for FME, our IME scheme intentionally guarantees that one of the two block types must be around MVP so that FME can access its reference pixel right in IME SRAM.

7.9. Experimental and Implemental Result

We partition the performance analysis into two parts. The first part does not include the adaptive skip detection algorithm (chapter 4) while the second part does.

7.9.1. Performance of PMRME

TABLE XIX and TABLE XX show the simulation result in different parameters, PARME only, PARME with mode filtering (PARME+MF), PARME+MF with bit truncation for 720p and 1080p sequences respectively. In addition, we also test the algorithm performance for high motion sequences by skipping two frames to simulate high motion sequences.

The simulation environments are as following: No rate-distortion optimization (RDO); sequence type IPPP and SR is [-128, 127]. All of the simulation results are compared with that of the Fast-Full-Search (FFS) Algorithm in JM9.0 [20]. The result in this table only shows the average performance under different QPs. For 720p, the test sequence including: Stockholm, park_run, and shields. The frame rate is 50 and 50 frames are coded. For 1080p, the test sequence including: station2, rush_hour, and sunflower. The frame rate is 25 and 100 frames are tested. The 1920x1080p image is truncated to an

image of 1920x1072 to fit the multiples of 16.

The simulation shows the PMRME alone can achieve the similar video quality as the FFS. However, the distortion is larger under high QP because the blocky effect will be more serious for high QP and mislead the subsampling method. Further combining with MF, we can find that the bit-rate for low QP case sometimes lower than the FFS with the penalty of PSNR loss. If the truncation method is combined, the performance is a little worse. For 720p sequences, it has 0.04dB PSNR loss but 1.62% of bite-rate decrease in average. As for 1080p sequences, it has 0.07dB PSNR loss and up to 1.20% of bit-rate increase in average. However, with slightly quality loss, we can save a lot of hardware cost as described in the previous section. For high motion sequences simulated by 2 frame skipping, the proposed design keeps the similar quality. It means the algorithm does well even for high motion sequences.

In these two tables, the term “L0 hit rate” means the percentages of motion vector falls in layer 0. With this, the memory data of level 0 can be directly reused by fraction ME and thus save a lot of bandwidth. In our design, the hit rate is at least 87%, and the higher QP will have higher hit rate and thus can save more power and BW. In our analysis, the hit rate for 720p and 1080p is quite the same, so we conclude the search range 128 is enough for the 720p and 1080p sequence. However, the bit truncation method has stronger impact for 1080p. As we can see, the bit truncation method will lead to 1.20% bit-rate increasing for 1080p sequence. The rate-distortion (RD) curves of 720p and 1080p sequence are shown in Figure 73 and Figure 74, and the RD curves are almost overlapped with that by FFS.

TABLE XIX PMRME performance for 720p sequences

720p						
QP		PMRME	PMRME+MF	PMRME+MF (5 bits)	PMRME+MF (5 bits) Skip 2 frame	
						L0 Hit rate (%)
QP12	PSNR inc.(db)	-0.01	-0.02	-0.02	-0.02	88.62
	Bit rate inc. (%)	-3.53	-4.87	-2.97	-4.10	
QP16	PSNR inc.(db)	0.00	-0.05	-0.03	-0.04	90.41
	Bit rate inc. (%)	-1.33	-2.56	-1.56	-2.38	
QP20	PSNR inc.(db)	0.01	-0.09	-0.07	-0.07	91.67
	Bit rate inc. (%)	-0.94	-2.40	-1.38	-2.13	
QP24	PSNR inc.(db)	0.00	-0.07	-0.06	-0.06	93.72
	Bit rate inc. (%)	-1.26	-2.73	-1.63	-2.69	
QP28	PSNR inc.(db)	0.00	-0.05	-0.04	-0.05	95.14
	Bit rate inc. (%)	-0.86	-2.09	-1.57	-2.91	
QP32	PSNR inc.(db)	-0.01	-0.05	-0.04	-0.05	95.63
	Bit rate inc. (%)	0.27	-0.96	-0.58	-1.75	
Average	PSNR inc.(db)	0.00	-0.06	-0.04	-0.05	92.53
	Bit rate inc. (%)	-1.28	-2.60	-1.62	-2.66	

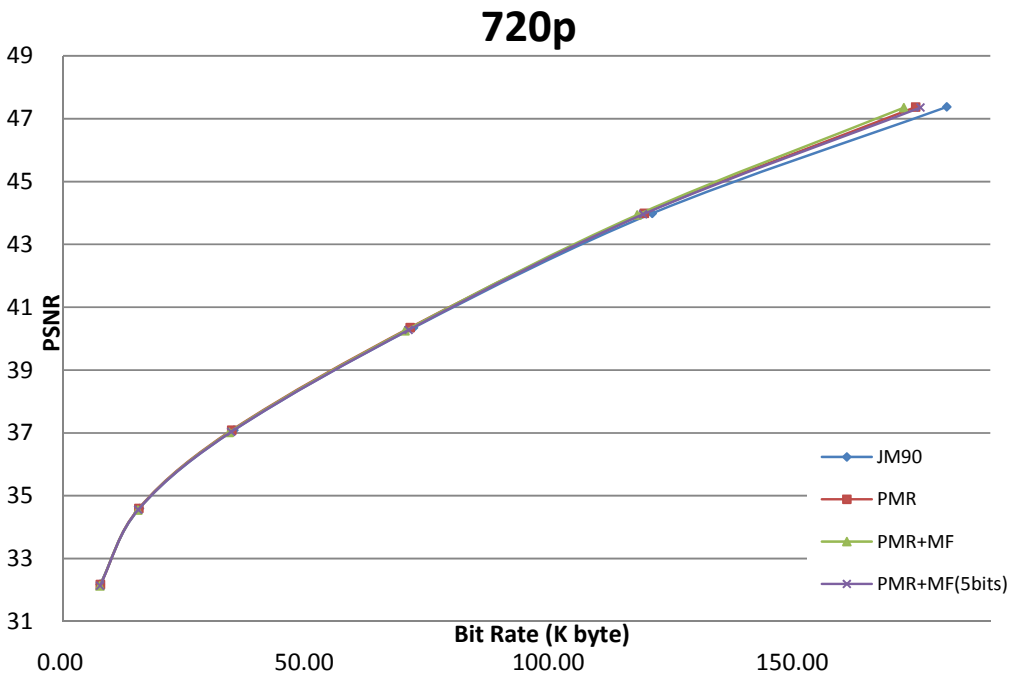

Figure 73 PMRME performance of 720p sequences

TABLE XX PMRME performance for 1080p sequences

1080p						
QP		PMRME	PMRME+MF	PMRME+MF (6 bits)	PMRME+MF (6 bits) Skip 2 frame	
						L0 Hit rate (%)
QP12	PSNR inc.(db)	0.00	-0.08	-0.09	-0.05	87.69
	Bit rate inc. (%)	-2.14	-4.62	-3.00	-3.93	
QP16	PSNR inc.(db)	0.00	-0.07	-0.06	-0.06	89.57
	Bit rate inc. (%)	-0.49	-1.09	0.47	-0.47	
QP20	PSNR inc.(db)	-0.01	-0.04	-0.04	-0.03	92.33
	Bit rate inc. (%)	-0.44	-0.22	2.65	1.89	
QP24	PSNR inc.(db)	-0.03	-0.06	-0.06	-0.07	93.19
	Bit rate inc. (%)	-0.40	-0.94	1.83	1.03	
QP28	PSNR inc.(db)	-0.06	-0.07	-0.08	-0.07	93.47
	Bit rate inc. (%)	0.40	0.19	2.20	1.64	
QP32	PSNR inc.(db)	-0.10	-0.09	-0.10	-0.09	93.39
	Bit rate inc. (%)	1.68	1.59	3.06	2.54	
Average	PSNR inc.(db)	-0.03	-0.07	-0.07	-0.06	92.53
	Bit rate inc. (%)	-0.23	-0.85	1.20	0.45	

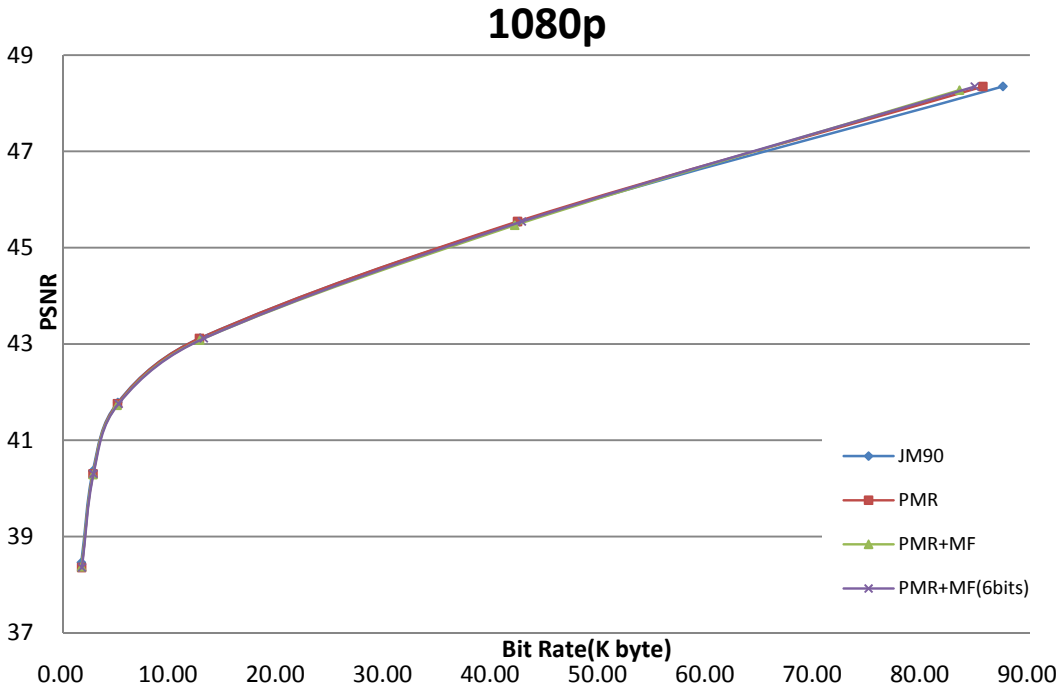

Figure 74 PMRME performance of 1080p sequences

7.9.2. Performance of Skip + PMEME

We further combine the PMRME algorithm with the adaptive skip mode detection. We test three 150-frame 720p sequences. The three sequences are: Stockholm, park_run, and shields. The testing environments are: search range 128, 1 reference frame, coding type: IPPP, no rate-distortion optimization and disable thresholding.

Figure 75 - Figure 77 show the R-D curve of these three test sequences, and TABLE XXI is the average performances. The performance does well for low QP condition, and it is because mode filtering performs well in low QP large frame size. The L0 hit rate is very high (about 98.05% in average) because the MVs are rapidly converged by PMRME. The contribution of speed-up comes from two factors: one factor is that the PMRME has fewer search points than full search algorithm, and the other factor comes from the pre-skip algorithm. We can see that the PMRME has an intrinsic speed-up about 25, and the rest of the speed up will depend on how much MBs are skipped.

TABLE XXI Skip+PMRME performance for 720p sequences

	PSNR (dB)	Bit rate (%)	Speed-up	L0 hit rate (%)	Pre_skip rate (%)
QP12	-0.02	-2.78	25.64	96.87	0.07
QP16	-0.03	-1.10	26.83	96.62	0.10
QP20	-0.10	-1.03	30.33	97.24	0.02
QP24	-0.09	-0.86	32.15	98.45	0.32
QP28	-0.05	0.68	35.53	99.41	9.09
QP32	-0.11	-0.68	48.60	99.70	28.25
average	-0.07	-0.96		98.05	

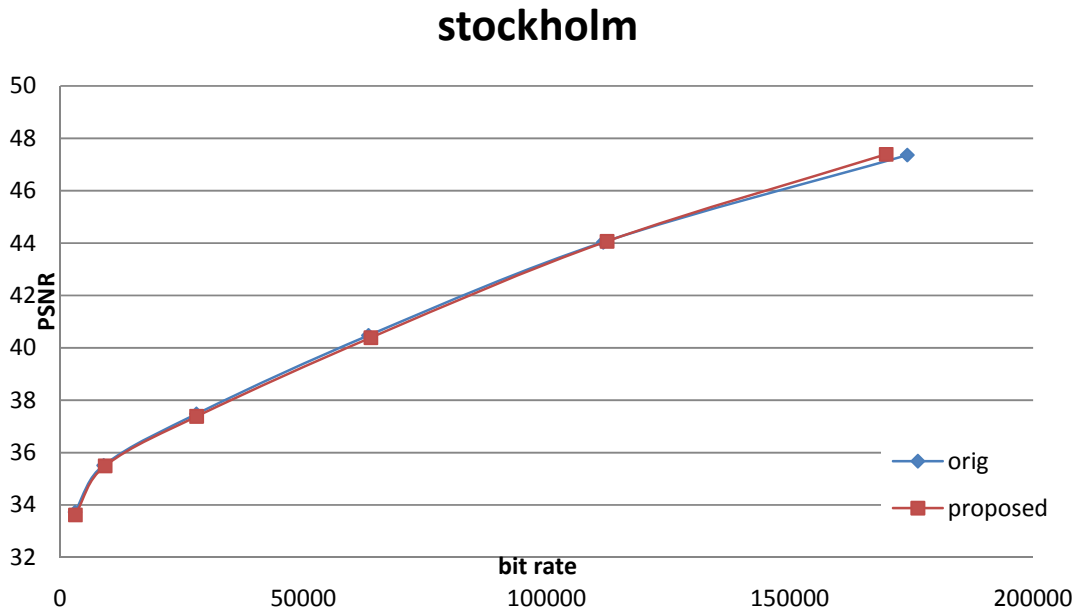

Figure 75 Skip+PMRME performance of 720p stockholm

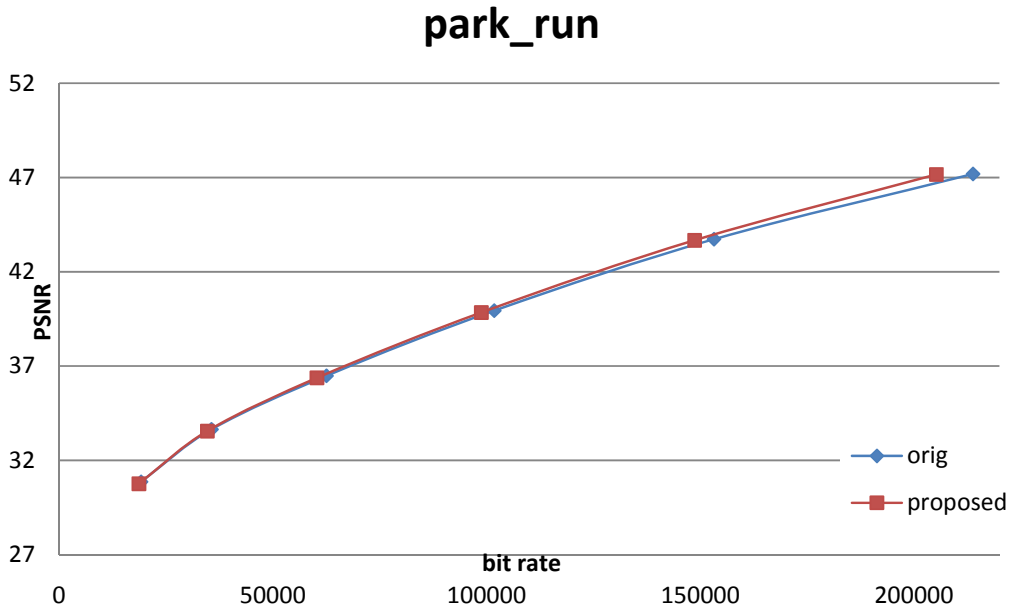

Figure 76 Skip+PMRME performance of 720p park_run

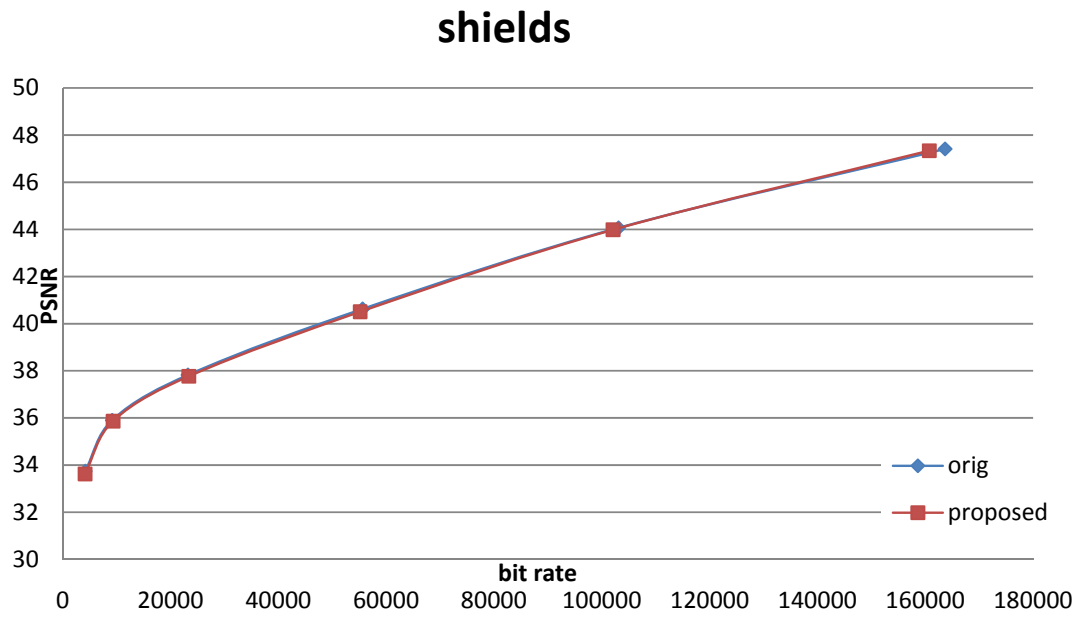

Figure 77 Skip+PMRME performance of 720p shields

Besides, we also test another three 100-frame 1080p sequences. The three sequences are: station2, rush_hour, and sunflower. The testing environments are: search range 128, 1 reference frame, coding type: IPPP, no rate-distortion optimization and disable thresholding.

Figure 78 - Figure 80 show the R-D curve of these three sequences and TABLE XXI shows the average performance. We see that for these three sequences, we pre-skip a lot of MBs for high QP condition with larger PSNR drop and bit rate decreasing. In the extreme case, it drops 0.37dB in PSNR and 7.08 (%) bit rate decreasing. Another point deserves to notice is that 1080p frame is smoother than 720p frame; therefore we can skip more MBs in 1080p sequences.

TABLE XXII Skip+PMRME performance for 1080p sequences

	PSNR (dB)	Bit rate (%)	Speed up	L0 hit rate (%)	Pre_skip rate (%)
QP12	-0.08	-3.02	26.65	94.66	0.00
QP16	-0.09	-0.96	30.02	96.24	0.04
QP20	-0.01	0.61	34.44	98.18	4.86
QP24	-0.06	-1.88	48.10	98.54	28.35
QP28	-0.32	-6.60	75.19	98.31	55.48
QP32	-0.38	-7.08	96.33	98.17	64.60
average	-0.16	-3.15		97.35	

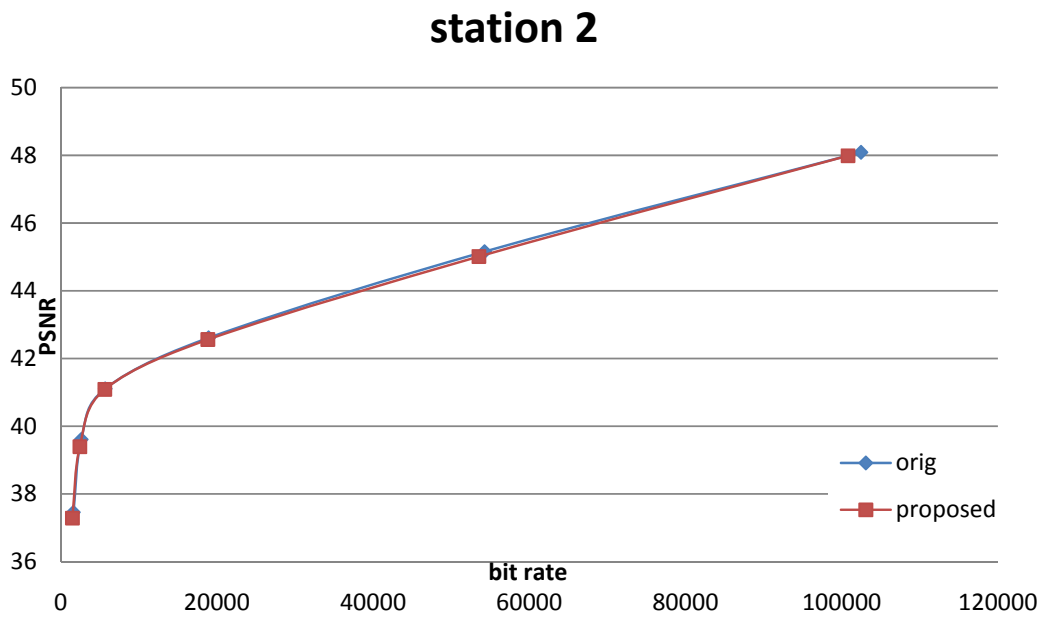

Figure 78 Skip+PMRME performance of 1080p station2

Figure 79 Skip+PMRME performance of 1080p rush_hour

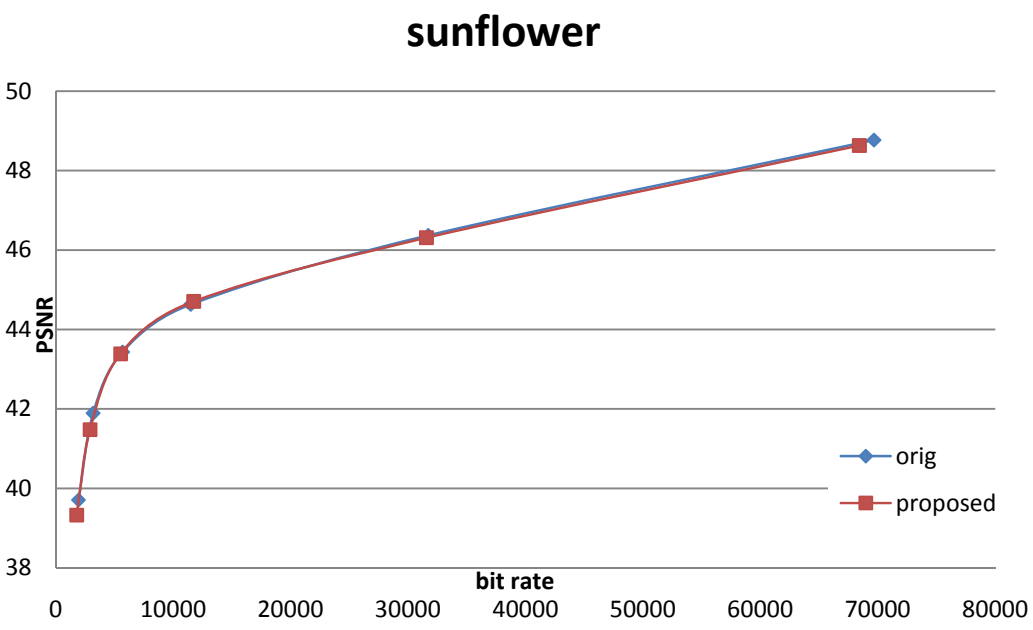

Figure 80 Skip+PMRME performance of 1080p sunflower

7.10. Implementation Result

The proposed design has been implemented and synthesized. TABLE XXIII shows the total hardware cost of our design and comparison to other design. Our design can provide the largest search range capability (High Profile Level 2) but just needs similar hardware cost and smaller buffers. Besides, our design has the shortest latency so that our design can achieve 1080p@60fps specification with only 124MHz operating frequency only. In comparison, designs in [32], [35] has larger area cost and long latency due to the full search architecture. Though designs in [8], [33] use fast algorithms to reduce the latency, they still needs large area cost and buffer. At last, comparing with [16], our design can save at least 48.9% of area costs and 62.1% of memory costs with larger search range. Therefore, the proposed IME design can achieve low latency while low buffer cost and similar area cost, and thus is suitable for HD applications.

TABLE XXIII the PMRME hardware cost comparison

	[8]	[16]	[32]	[33]	[35]	**Ours**
Max. Supporting Resolution	720x480@ 30fps	720p@ 30fps	4CIF@ 15fps	CIF@ 30fps	720p@ 60fps	**1080p@ 60fps**
Search Algorithm	multi resolution	Full	Full	4-Step	Full	**multi resolution**
Max. Search Range	H:±64 V: ±32	H:±64 V: ±32	H:±64 V: ±64	H:±32 V: ±16	H:±16 V: ±16	**H: ±128 V: ±128**
IME Gate Count (K)	N/A	305	154	131.2	176	**155.8 for 720p 213.7 for 1080p**
IME Memory (Kbyte)	N/A	13.71	7.5	64	41.6	**5.19 for 720p 5.95 for 1080p**
Operating Freq.(MHz)	16 / 7.7 for 720x480	108 for 720p	100 for 4CIF	40 (13.3 for CIF)	55.6 for 720p	**27.6 for 720p 124.4for 1080p**
Latency for IME Stage (cycle)	375 / 180	N/A	1024	N/A	258	**256**
CMOS Tech.	N/A	0.18um	0.18um	0.18 um	0.18um	**0.13 um**

7.11. Summary

In this chapter, we propose a parallel multi-resolution algorithm and architecture to integer ME for H.264/AVC. The algorithm uses PMRME, MF and bit truncation to support large search range within 256 cycles. With data reuse and parallel multi-resolution, we can save at least 91.91% of memory buffer and 55.1% of bandwidth. The resulted hardware can save up to 48.9% of area cost and 62.1% of memory cost compared to the previous approach for 720p processing. With above features, the proposed design is very suitable for larger search range application such as HDTV in a more economical way.

8. Integration for 1080P H.264/AVC High Profile Encoder

8.1. introduction

For motion estimation, forward motion estimation (P frame) uses a previous frame as the reference frame. However, it produces poor prediction when the frame contains hidden regions or newly entered objects. Therefore, an effective way to solve the problem is to predict from the next frame. Bi-directional motion estimation (B frame) is supported in high profile H.264 standards [1], and it uses past frames and future frames for prediction as illustrated in Figure 81.

Bi-direction motion estimation provides a number of significant advantages, including:

1. Hidden areas can be predicted by using the future frame
2. Less bit budget can be spent to achieve the same quality of motion compensation
3. It has a better noise suppression quality than the forward frame prediction

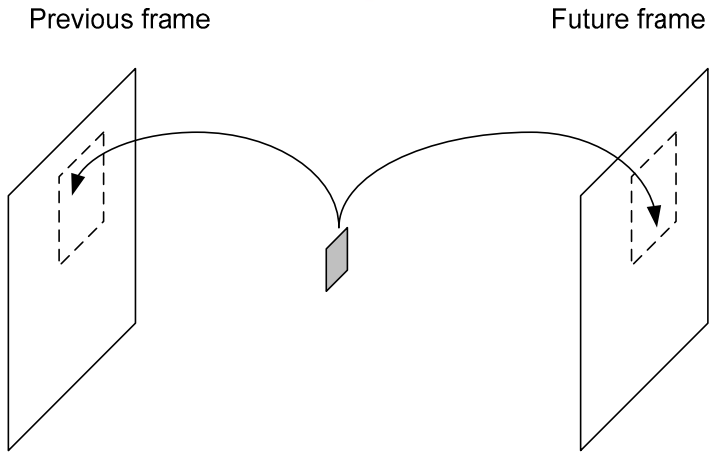

Figure 81 the concept of bi-directional motion estimation

To support the specifications of H.264, we extend PMRME architecture to support

bi-direction motion estimation. As in chapter 0, we need 256 cycles to finish search range [-127,124] in one direction. In this way, we have to consume 512 cycles to complete bi-directional motion estimation, which can be well integrated with our 560 cycle intra coder design [37].

8.2. System Architecture for Bi-direction Motion Estimation

Figure 82 is the system architecture of bi-directional motion estimation. The width of external bus is 128 bits; it is because 128 bits equals the width of a MB which is 16 pixels.

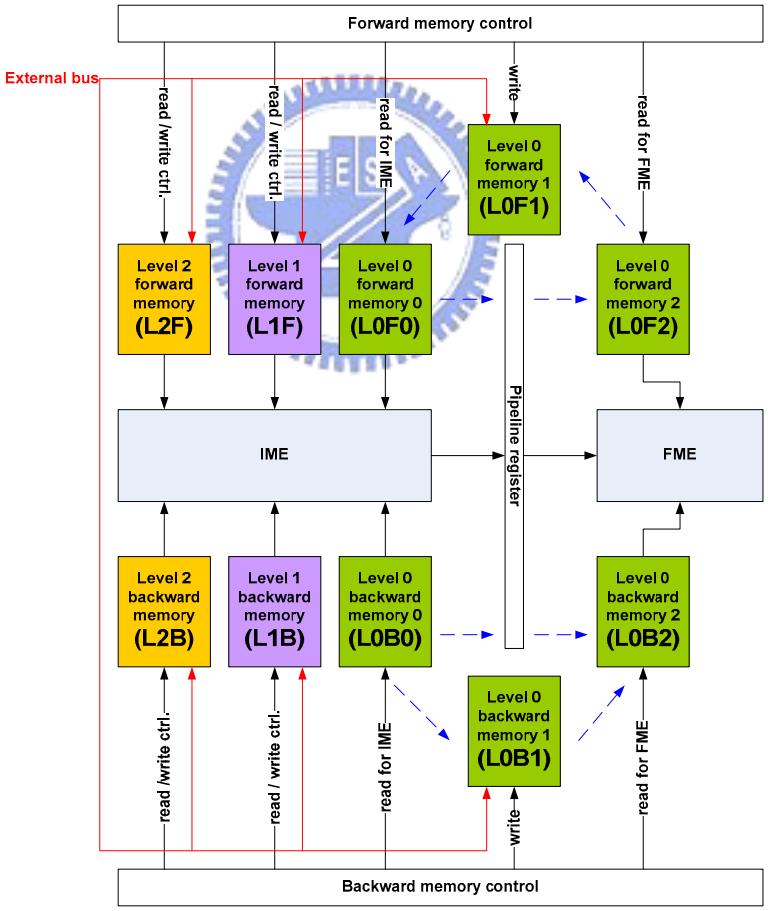

Figure 82 system architecture of bi-directional motion estimation

Roughly speaking, we doubled the memory for reference frame, while the hardware cost

is the same, because the throughput of PMRME (512 in bi-directional) is good enough to meet the requirement of our intra coder (560). As shown in Figure 82, we have 10 memory blocks totally, they are composed of 3 for level 0 forward searching, 3 for level 0 backward searching, 1 for level 1 forward searching, 1 for level 1 backward searching, 1 for level 2 forward searching and 1 for level 2 backward searching. The goal and usage of these 3 memory blocks in level 0 have been explained in section 7.8. With these level 0 ping-pong buffers, we can share the level 0 data of IME with the FME, and no additional memory access time is necessary. In our ME design, if the motion vector from IME is around motion vector predictor, FME can use the reference pixels in IME reference SRAM.

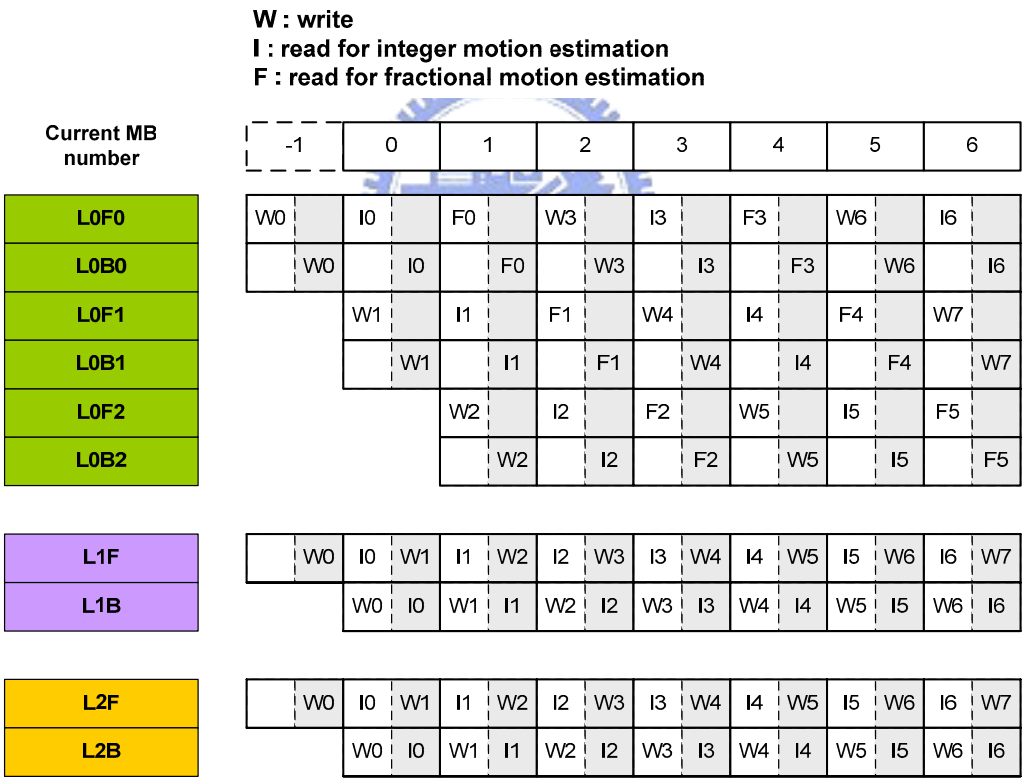

Figure 83 memory schedule of bi-direction ME

Figure 83 is the memory behavior of our design. For each MB processing, it consists of two stages: the former half stage and the latter half stage (gray colored).

For level 0, when processing -1th MB, L0F0 (the numbered 0 of the level 0 forward searching memory) will have to write the forward reference data in the former stage, whereas the L0B0 also has to write the backward reference data in the latter stage, because these data have to be ready before 0th MB to do motion estimation. When processing 0th MB, L0F0 will be read to do forward IME in the former stage and L0B0 will be read to do backward IME in the latter stage. At the same time, L0F1 will write the forward reference data at former stage and L0B1 write the backward reference data at latter stage for 1st MB. When processing 1st MB, the L0F0 and L0B0 will be shifted to FME stage, while L0F0 to do forward FME in the former stage and L0B0 to do backward FME in latter stage. At the same time, L0F1 and L0B1 will do IME for 1st MB and L0F2, L0B2 write the reference data for 2nd MB. In this way, L0F0, L0F1, and L0F2 will perform in a ping-pong buffer and L0B0, L0B1, L0B2 in another ping-pong buffer to continue the motion estimation for every MB.

For level 1 and level2, every former stage will do forward IME and write the reference data for the backward search buffer, and every post stage will do backward IME and write the forward reference data of next MB.

8.3. Frame-Level Memory Scheduling

The processing of MBs in a frame is from left to right and top to down. Take 720p frame as an example, as depicted in Figure 84, after dealing with 80 MBs in a row, the procedure go to the next row until finish total 45 rows in a frame.

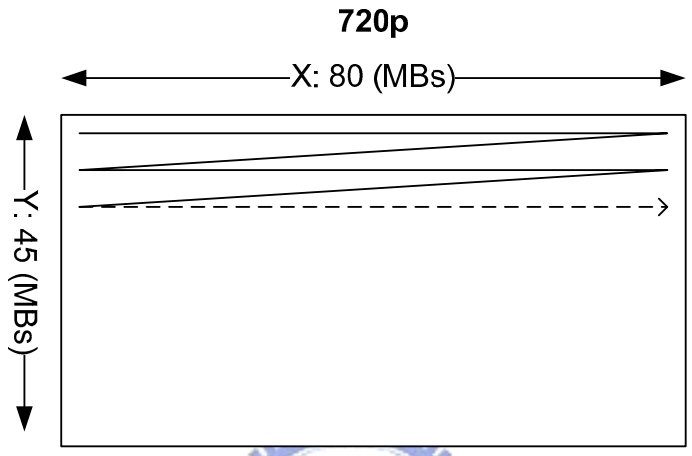

Figure 84 coding order of 720p sequences

The range of accessed reference data is different in different levels. For level 0, because the search region is centered on motion vector predictor (MVP), the search region is irregular for every MB. Therefore, to simplify the data flow and control in an efficient way, we just extend the data from boundary and constrict the motion vectors inside the frame size if the search region is out of the frame boundary.

However, the reference data flow is very regular in level 1 and level 2. Both of their search region are centered on (0, 0), thus we only have to refresh a column of search range (a bank of memory in our design) and it can save a lot of memory bandwidth (BW). The refreshed reference data in level 1 for 720p frame size is shown in Figure 85, and Figure 86. The search range of level 1 is [-32, 30], so we have to load a square search region which contains 5x5 MBs. Nevertheless, if current block is near the boundary of a frame, there are some data outside the boundary, thus we can reduce the memory bandwidth. For example, if the current MB locates on Y=0, we cannot get the upper half

side of data and we only can get a search range of Y=0~2. Besides, for level 1, 4:1 subsampling, we need 8 cycles to acquire the data of one MB, so we need 3x8=24 cycles to refresh a bank of search buffer. In the same manner, we pay 4x8=32 cycles to get the search region of Y=0~3 when current block locates on Y=1. The rest of the search region scheduling is shown in Figure 86. The concept and method of accessed reference scheduling in level 2 are the same as level 1 and it is shown in Figure 87.

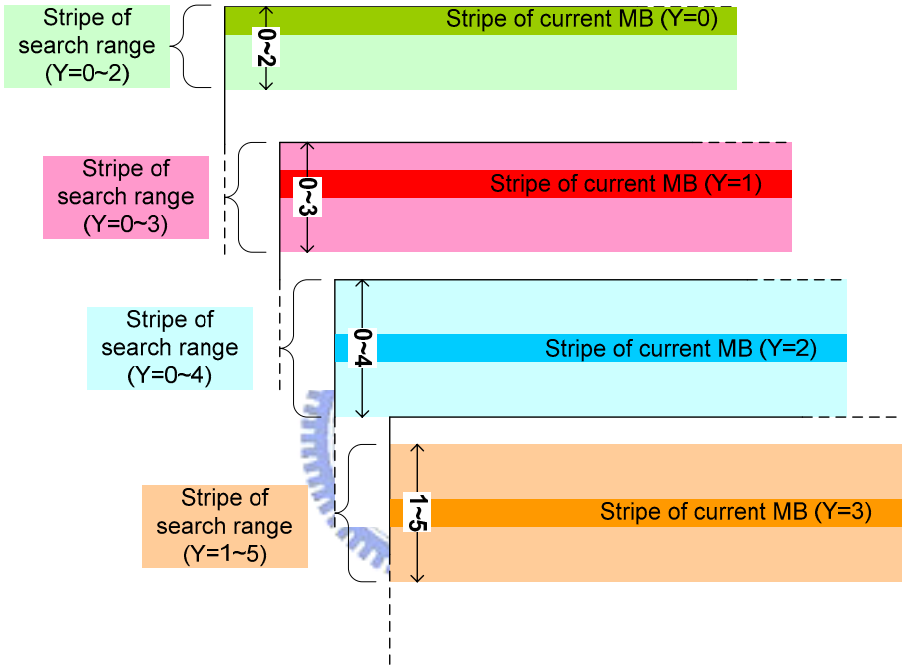

Figure 85 the relationship of stripe of current MB and search range in level 1

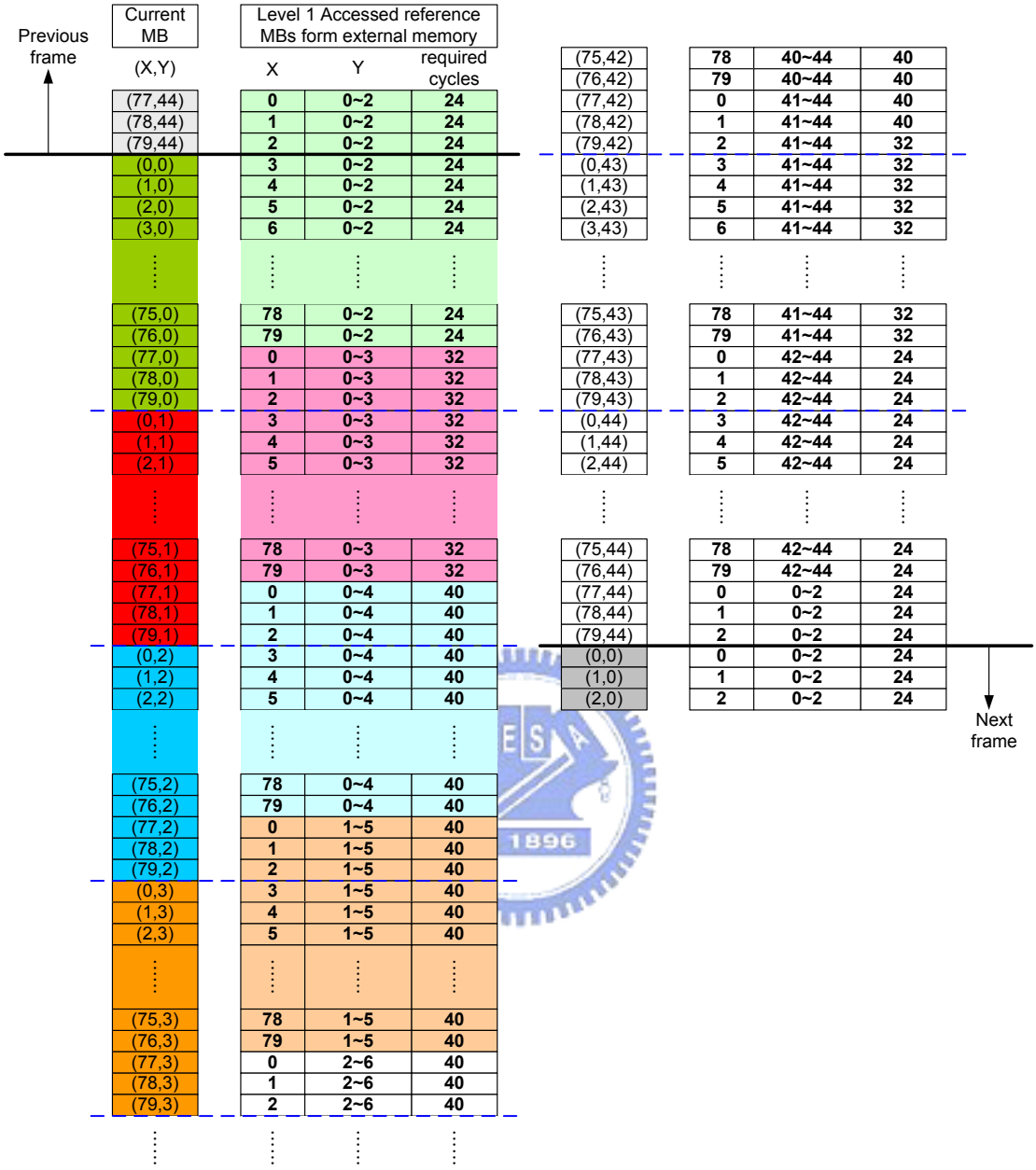

Figure 86 the refreshed reference data of every MB in level 1

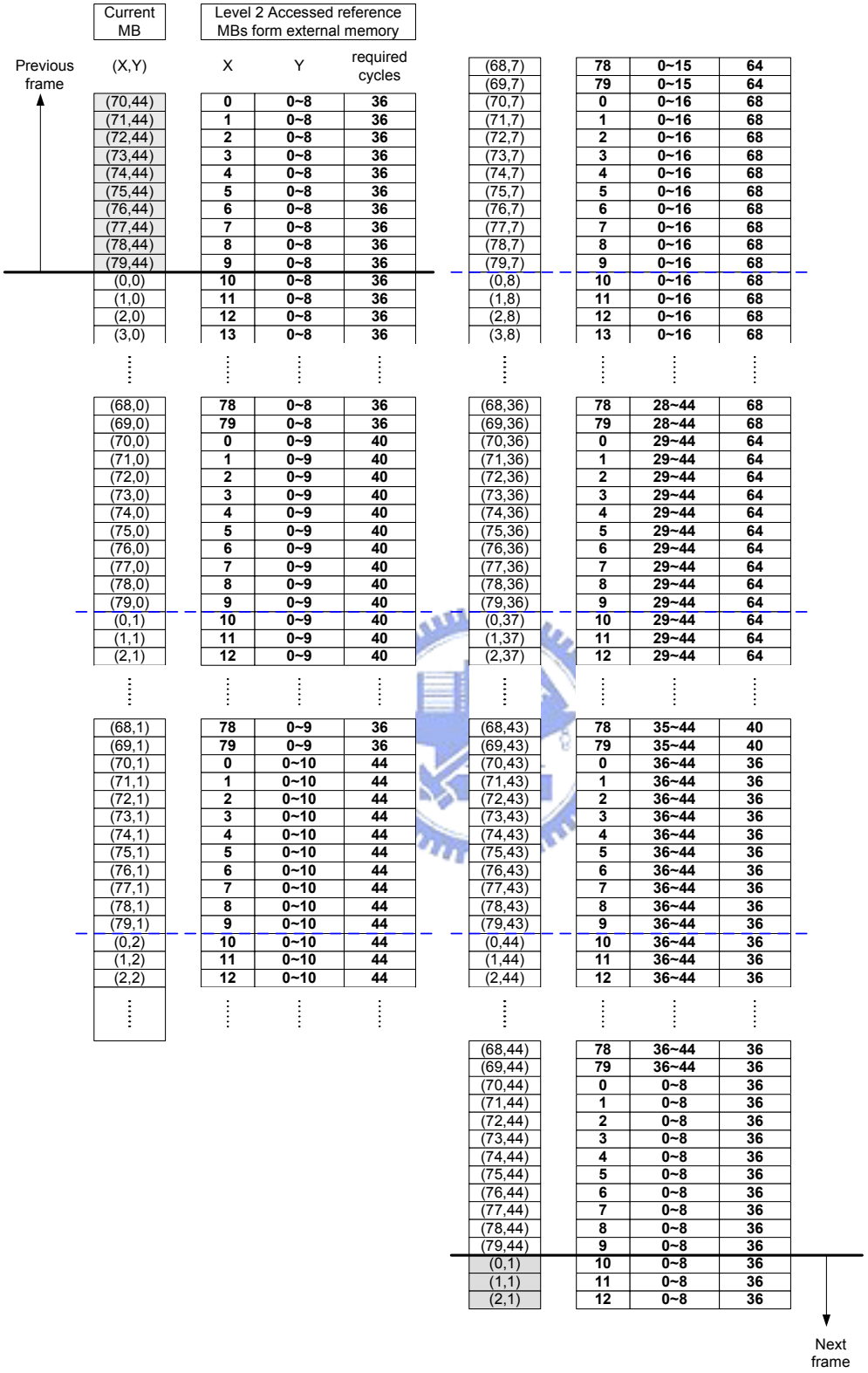

Figure 87 the refreshed reference data of every MB in level 2

8.4. Bandwidth Analysis

The bandwidth required is not consistant because some data outside boundary are unabailable and the bandwidth is depend on the Y-axis of current block. Bandwidth required for every MB is:

$$\text{BandWidth} = \text{required accessing cycle} * 16 \text{ (Bytes)}$$

Required accessing cycle is the cycle needed to refresh the column of serarch range and 16 is the width of the external bus. The bandwidth of level 0, level 1 and level 2 are illustarted in Figure 88 and Figure 90 respectively. At last, the average bandwidth for 720p sequences and 1080p sequences are listed in TABLE XXIV. In one direction, the averaged bandwidth for 720p for is 3.385 bytes per MB, and for 1080p is 3.425 bytes per MB. So, we need 6.77 bytes per MB and 6.85 bytes per MB for bi-direction 720p and 1080p motion estimation respectively.

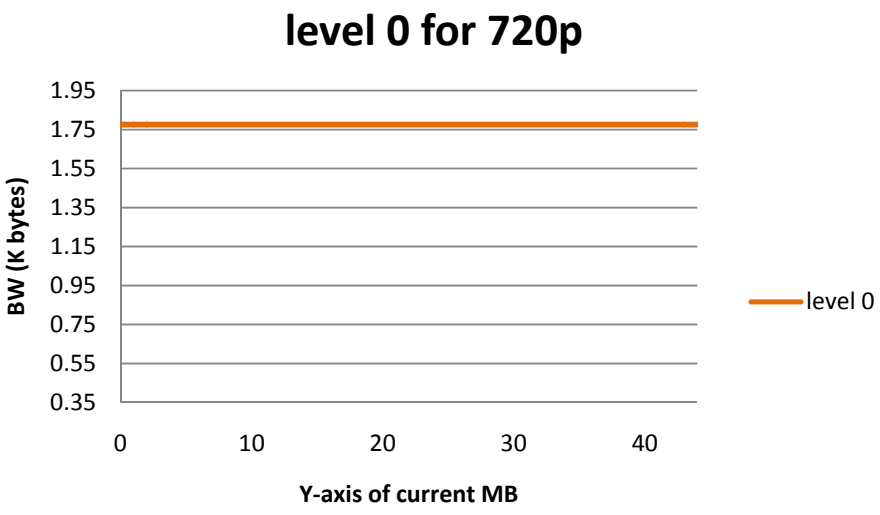

Figure 88 bandwidth of PMRME level 0 (720p)

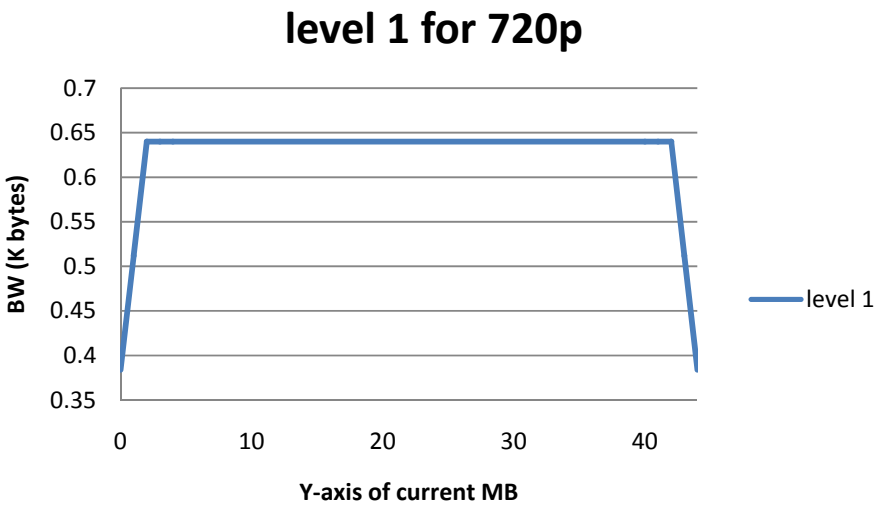

Figure 89 bandwidth of PMRME level 1 (720p)

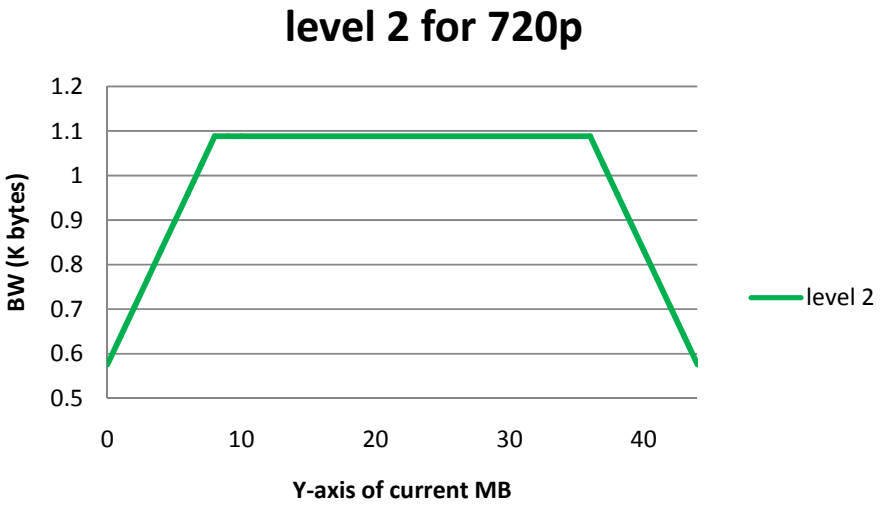

Figure 90 bandwidth of PMRME level 2 (720p)

TABLE XXIV average bandwidth per MB

Averaged BW per MB	720p	1080p
Level 0 (K bytes)	1.776	1.776
Level 1(K bytes)	0.623	0.629
Level 2(K bytes)	0.986	1.020
Total (K bytes)	3.385	3.425

8.5. Conclusion

In this chapter, we integrate the PMRME architecture into a high profile H.264/AVC encoder which supports 1080p bi-directional motion estimation within 512 cycles. The memory size is 11.912 K-bytes and the memory bandwidth is 6.85 K bytes per MB. The design is finally synthesis by UMC 0.13um CMOS process under 145 MHz to support 1080p @ 30fps.

9. Conclusion

In this thesis, we studied and analyzed several algorithms and hardware architectures for motion estimation in the latest video codec standard H.264/AVC. The conclusion can be summarized into five parts:

9.1. Adaptive Skip Mode Detection

The hardware cost of this part is 0.63K gate counts, 11 bytes and 40 bytes memory for CIF size and 720p size video respectively. The video performance of our design is quite similar the same as the JM9.0 and our design can save 77.13% to 11.62% of coding time in difference sequences and QPs.

9.2. Mode filtering

This method can speed up the overall motion estimation process with the reduction of modes for fractional motion estimation. With this algorithm, the hardware implementation can be pipelined with higher efficiency and slightly performance loss.

9.3. Refined Quarter Pixel Motion Estimation

We propose a design for search range [-16, 15] integer motion estimation in H.264/AVC. The algorithm uses the mode filtering to reduce the computational load of fractional motion estimation and refined quarter motion estimation to enable four times of parallel processing while lower computational complexity and low quality loss. The hardware design consumes 44K gate counts, which only needs half of process element counts and lower latency than other design. The power consumption of the design is only 0.59mW for 30fps CIF sequences and 0.08mW for 30fps QCIF sequences. Therefore,

the design is very suitable for medium frame size application such as mobile phone, personal digital assistant (PDA) or video camera.

9.4. Parallel Multi Resolution Motion Estimation

We present a search range [-128, 124] parallel multi-resolution algorithm and architecture of integer motion estimation for H.264/AVC. The algorithm uses PMRME, MF and bit truncation to support large search range within 256 cycles. With data reuse and parallel multi-resolution, we can save at least 91.91% of memory buffer and 55.1% of bandwidth. The resulted hardware can save up to 48.9% of area cost and 62.1% of memory cost compared to the previous approach for 720p processing. With above features, the proposed design is very suitable for larger search range application such as HDTV in a more economical way.

9.5. 1080p High Profile Encoder Chip

We integrate the PMRME architecture into a high profile H.264/AVC encoder which supports 1080p bi-directional motion estimation within 512 cycles. The memory size is 11.912 K-bytes and the memory bandwidth is 6.85 K bytes per MB. The design is finally synthesis by UMC 0.13um CMOS process under 145 MHz to support 1080p @ 30fps.

9.6. Future Work

We have finished several fast motion estimation algorithm and architecture to support H.264/AVC high profile encoder, while the next generation of H.264 – Scalable Video Coding (SVC) [38] has become increasingly important because a variety of end-users with different available bandwidths and processing capabilities may request the same multimedia material. The most important scalability dimensions are for different picture size (spatial), frame rate (temporal), and image quality (signal-to-noise) resolutions. Thus, we have to further modify our algorithm and architecture, and try to find a

trade-off between the performances and cost to support SVC in an efficient way. This is another challenge need to complete in our future.

10. Reference

[1] Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264/ ISO/ IEC14496-10 AVC), Mar. 2003

[2] T. Wiegand, G. J. Sullivan, G. Bjontegaad, and A. Luthra, “Overview of the H.264/AVC video coding standard,” *IEEE Transaction on Circuits System for Video Technology*, vol. 13, pp. 560-575, July 2003.

[3] Peter Kuhn, *Algorithms, complexity analysis and VLSI architectures for mpeg-4 motion estimation*, pp.22-23, Kluwer academic publishers, 1999

[4] M.J, Chen, L. G. Chen, and et al, “A new block-matching criterion for motion estimation and its implementation,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol.5, issue 7, pp.231-236, Jun 1995.

[5] T. Koga, K. Iinurna, A. Hirano, Y. Iijima, and T. Ishiguro, “Motion-compensated interframe coding for video conferencing,” *Proceedings of NTC'81*, pp. G5.3.1-G5.3.5, New Orleans, LA, Dec. 1981.

[6] Reoxiang Li, Bing Zeng, Liou, and M.L., ”A new three-step search algorithm for block motion estimation,” *IEEE Transactions on Circuits System for Video Technolog*, Vol. 4, pp. 438-442, Aug. 1994

[7] H. Y. Chin and et al, “A bandwidth efficient subsampling-based block matching architecture for motion estimation,” *Asia and South Pacific Design Automation Conference*, pp. 18-21, 2005

[8] Jae Hun Lee and Nam Suk Lee, “Variable block size motion estimation algorithm and its hardware architecture for H.264/AVC,” *IEEE conference on International Symposium on Circuits and System,* vol. 3, pp. 741-744, May 2004

[9] M. Ghanbari, “The cross-search algorithm for motion estimation,” *IEEE Transaction on Communication*, vol. 38, issue 7, pp. 950-953, 1990

[10] O.T.-C. Chen, “Motion estimation using a one-dimensional gradient descent search,” *IEEE Transaction on Circuits System for Video Technology*, vol. 10, issue 4, pp. 608-616, 2000

[11] L.-K. Liu and E. Feig, “A block-based gradient descent search algorithm for block motion

estimation in video coding," *IEEE Transaction on Circuits System for Video Technology*, vol. 6, issue 4, pp. 419-422, 1996

[12] L.-M. Po and W.-C. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Transaction on Circuits System for Video Technology*, vol. 6, issue 2, pp. 313-317, 1996

[13] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block matching motion estimation," *IEEE Transaction on Image Processing*, vol. 9, issue 2, pp. 287-290, 2000

[14] C.-H. Cheung and L.-M. Po, "A novel cross-diamond search algorithm for fast block motion estimation," *IEEE Transactions on Circuits System for Video Technology*, vol. 12, issue 12, pp. 1168-1177, 2002

[15] C. Zhu, X. Lin, and L.-P. Chau," Hexagon-based search pattern for fast block motion estimation," *IEEE Transactions on Circuits System for Video Technology*, vol. 12, issue 5, pp. 349-355, 2002

[16] T. C. Chen and et al, "Analysis and Architecture Design of an HDTV720p 30 Frames/s H.264/AVC Encoder," *IEEE Transaction on Circuit System for Video Technology*, vol. 16, no. 6, pp. 673-688, June 2006

[17] C. Sam. Kann. ,and et al, "Low-Complexity Skip Prediction for H.264 Through Lagrangian Cost Estimation," *IEEE Transaction on Circuits System Video Technology*, vol. 16, no. 2, pp. 202-208, 2006.

[18] Hanli Wang and et al, "Efficient prediction algorithm of integer DCT coefficients for H.264/AVC optimization," *IEEE Transactions on Circuits System for Video Technology*, vol. 16, no. 4, pp. 547-552, 2006.

[19] Gyu Yeong Kim and et al, "An early detection of all-zero DCT blocks in H.264" *International Conference on Image Processing*, pp.453 – 456, 24-27 Oct. 2004

[20] Joint Video Team Reference Software JM9.0, ITU-T

[21] Ivanov, Y.V, "Skip Prediction and Early Termination for Fast Mode Decision in H.264/AVC" *International Conference on Telecommunication*, pp.7-7, 29-31 Aug. 2006

[22] Yu-Han Chen, and et al, "Power-scalable algorithm and reconfigurable macro-block pipelining architecture of h.264 encoder for mobile application", *IEEE International*

Conference on Multimedia & Expo, pp. 281-284, July 2006

[23] Y. W. Huang, and et al, "Hardware architecture design for variable block size motion estimation in MPEG-4 AVC/JVT/ITUT-T H.264," *IEEE Conference on International Symposium on Circuits and System*, vol. 2, pp. 196-199, May 2003

[24] Esam A. Al_Qaralleh, Tian-Sheuan Chang, ”Fast Variable Block Size Motion Estimation and Its VLSI Architecture by Adaptive Early Termination,” *IEEE Transaction on Circuits and Systems for Video Technology*, Volume 16, Issue 8, pp.1021-1026, Aug. 2006

[25] T. C. Chen and et al, ”Fully utilized and reusable architecture for fractional motion estimation of H.264/AVC,” *IEEE conference on International Conference on Acoustics, Speech, and Signal Processing*, vol. 5, pp. 9-12, May 2004

[26] S. Y. Yap and J. V. McCanny, “A VLSI architecture for variable block size video motion estimation,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 51, issue 7, pp. 384-389, July 2004

[27] J. Miyakoshi, Y. Kuroda, M. Miyama, K. Imamura, H. Hashimoto, and M. Yoshimoto, “A sub-mW MPEG-4 motion estimation processor core for mobile video application,” *in Proc. IEEE Custom Integration Circuits Conference*, 2003, pp. 181–184.

[28] Tung-Chien Chen, Yu-Han Chen, Sung-Fang Tsai, Liang-Gee Chen,” Architecture Design of Low Power Integer Motion Estimation for H. 264/AVC,” *Conference on International Acoustics, Speech and Signal Processing*, Volume 3, 14-19 May 2006 pp. III 900-4

[29] Y. J. Wang and et al, “A fast Algorithm and its VLSI architecture for fractional motion estimation for H.264/MPEG-4 AVC video coding”, *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, issue 5, pp. 578-583, May 2007

[30] C. C. Lin, and et al, ”Hardware Oriented Algorithms for Motion Estimation in MPEG-4 AVC/H.264 Video Coding”, VLSI Design/CAD Symposium, pp. 505-508, Taiwan, Aug. 2005

[31] C. C. Lin and et al,” A fast algorithm and its architecture for motion estimation in MPEG-4 AVC/H.264 video coding”, *Asia Pacific Conference on Circuit and System*, pp. 1250-1253, Dec. 2006

[32] M. Kim, I. Hwang, and S. I. Chae, “A fast vlsi architecture for full search variable block size motion estimation in MPEG-4 AVC/H.264,” *in Proc. Asia and South Pacific Design*

Automation Conf.(ASPDAC), 2005, vol. 1, pp. 631–634.

[33] T. C. Chen, and et al., “Fast Algorithm and Architecture Design of Low-Power Integer Motion Estimation for H.264/AVC,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, pp. 568-577, May 2007.

[34] Z.-L. He and et al, “Low-power VLSI design for motion estimation using adaptive pixel truncation,” *IEEE Transaction on Circuits System for Video Technology*, vol. 10, no. 5, pp. 669-678, Aug. 2000

[35] Z. Zheng, and et al, “High Data Reuse VLSI Architecture for H.264 Motion Estimation,” *International Conference on Communication Technology, Nov. 2006*

[36] J. H. Lee, K. W. Lim, B. C. Song, and J. B. Ra, “A fast multi-resolution block matching algorithm and its LSI architecture for low bit-rate video coding,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, no. 12, pp. 1289-1301, Dec. 2001.

[37] Chun-Wei Ku, and et al, “*Design of H.264/MPEG-4 AVC Intra Codec for High Definition Size Still Image and Video Applications*”, Master Thesis, Department of Electronics Engineering, Chiao-Tung University, Taiwan, 2006

[38] T. Wiegand, G. Sullivan,J. Reichel, H. Schwarz, M. Wien, "Draft of Scalable Video Coding - Working Draft 5," Joint Video Team of ITU-T VCEG and ISO/IEC MPEG, Doc. JVT-R201, Bangkok, Thailand, 14-20 January, 2006

作　者　簡　歷

姓名： 林嘉俊

籍貫： 台北市

學歷：

台北市立建國高級中學　　(民國 87 年 09 月 ~ 民國 90 年 06 月)

國立交通大學電子工程學系　學士　(民國 90 年 09 月 ~ 民國 94 年 06 月)

國立交通大學電子所系統組　碩士　(民國 94 年 09 月 ~ 民國 96 年 06 月)

著作：

國內會議

[1] Chia-Chun Lin, Yu-Kun Lin, Tian-Sheuan Chang,"Hardware Oriented Algorithms for Motion Estimation in MPEG-4 AVC/H.264 Video Coding", VLSI Design/CAD Symposium, pp. 505-508, Taiwan, Aug. 2005

國際會議

[2] Chia-Chun Lin, Yu-Kun Lin, Tian-Sheuan Chang, " A fast algorithm and its architecture for motion estimation in MPEG-4 AVC/H.264 video coding", *Asia Pacific Conference on Circuit and System*, pp. 1250-1253, Dec. 2006

[3] Chia-Chun Lin, Yu-Kun Lin, Tian-Sheuan Chang, "PMRME: A parallel multi-resolution motion estimation algorithm and architecture for HDTV sized H.264 video coding," in Proc. ICASSP, 2007