國立交通大學

電子工程學系

碩士論文

一個應用於雙向預測之低功率運動估計模組設計

A Low Power Motion Estimation Design for Bi-directional Search

研究生：戴世炘

指導教授：蔣迪豪　博士

中華民國九十五年六月

一個應用於雙向預測之低功率動量估測模組設計

A Low Power Motion Estimation Design for Bi-directional Search

研究生：戴世炘　　Student: Shih-Hsin Tai
指導教授：蔣迪豪　　Advisor: Tihao Chiang

國 立 交 通 大 學
電 子 工 程 學 系
碩 士 論 文

A Thesis
Submitted to Department of Electronics Engineering & Institute of Electronics
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in
Electronics Engineering

June 2006

HsinChu, Taiwan, Republic of China

中華民國九十五年六月

一個應用於雙向預測之低功率動量估測模組設計

研究生: 戴世炘　　指導教授: 蔣迪豪 博士

國立交通大學
電子工程學系 電子研究所碩士班

摘要

本論文提出一個應用於雙向預測之低功率動量估測(motion estimation)模組設計。一個完整的動量估測模組包含整數像素動量估測、模式決定(mode decision)以及分數像素動量估測。在整數像素動量估測部份，我們使用平行二元化搜尋架構以符合低功率應用的需求。此架構能夠同時處理雙方向的動量估測；而對於 P-frame 搜尋，此平行二元化搜尋架構將它分割成兩個部份使得產率加倍。和傳統的搜尋架構相比，此平行二元化搜尋架構在記憶體存取、操作頻率以及硬體需求都佔有相當的優勢。在模式決定部份，本論文使用一個新的一維演算法。和原本的二維演算法相比，新演算法減少了運算延遲同時避免硬體閒置。在分數像素動量估測部份，我們採用一個低功率循序的設計。並提出整個系統平行處理架構，包含整數像素動量估測、模式決定以及分數像素動量估測。本設計需要 130 cycles 完成一個雙向預測的巨方塊(macroblock)搜尋。在 TSMC 0.18um 製程下，本設計需要 131 kilo gate count 及 51 kilo bits 的記憶體使用量。而處理每秒 30 張 CIF 大小的影像所消耗的功率為 11.8 mW。和現存的設計相比，此設計所需的操作頻率最低，而在功率消耗上最多可達到 34 倍的改進。

A Low Power Motion Estimation Design for Bi-directional Search

Student: Shih-Hsin Tai Advisor: Dr. Tihao Chiang

Department of Electronic Engineering &
Institute of Electronics
National Chiao Tung University

Abstract

This thesis proposes a low power motion estimation (ME) design for bi-directional search. A complete ME module contains integer pel ME (IME), sub-pel ME (SME) and sometimes mode decision (MD). For low power applications, our new parallel binary search architecture allows parallel processing of bi-directional search in IME. For P frame search, this parallel search architecture divides the original search into two sub-groups of partial P-frame search to double the processing throughput. Compared to conventional search architecture, this parallel binary search architecture shows advantages of lower memory access bandwidth, working frequency, and hardware design cost requirement. In MD, this work adopts a new one dimensional algorithm to reduce long latency in the original two-dimensional algorithm to avoid hardware idling. In SME, a low power sequential design solution is adopted to balance the system pipelining for IME, MD, and SME. This work completes one bi-directional macroblock search in 147 cycles with 131 kilo gate count and 51 kilo bits on-chip memory using TSMC 0.18μm technology. The power consumption for CIF 30fps is 11.8 mW. Compared to the state-of-the-art designs, this work needs the lowest working frequency and shows 34X power improvement at most.

誌　謝　辭

兩年的碩士生涯即將告一個段落，回首這兩年來的日子，從一開始只懂得學會書上的知識來應付考試，到現在能夠自己分析問題，實作研究進而提出解決方法。這些過程看似簡單，但卻是如何成爲一個獨立思考的研究生最困難的一步。而這兩年來，在老師和學長的諄諄教誨下，加上同學間的互相切磋，使我能有所成長，在此要感謝許多人。

首先感謝我的指導教授蔣迪豪老師，在老師身上不僅學到學術知識，也習得許多待人處事的方法。而王士豪學長在這兩年內不厭其煩地指導我正確的研究的思考模式及實事求是的做事態度，讓我受益良多；儘管我時常無法達到要求，學長總是以正面積極的態度鼓勵我再接再厲；其實學長本身也忙碌於自己的研究工作，卻不吝花費時間精力在我身上，在此致上最深的謝意。亦感謝王俊能學長、彭文孝學長、李志鴻學長和黃項群學長在研究方向上適時的給予意見，對我的研究工作有很大的幫助。

感謝實驗室中一起切磋的同學陳治傑、賴德亘、陳韋霖，許多的研究想法來自於彼此的討論思考。另外感謝Commlab的所有學長、同學及學弟，使實驗室充滿著歡愉溫馨的氣氛，也豐富了研究所多采多姿的生活。

最後要感謝我的父母和家人對我的支持與關懷，讓我毫無後顧之憂地專注於學業上。僅以此文，獻給所有關心我的人。

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Need of a Low Complexity, High Coding Efficiency Video Encoder

Today, the wide interests with real-time video applications on portable devices such as teleconferencing, networked video, etc. grow with time. For the portable video application, a low complexity video encoder is needed to reduce the power consumption and a high coding efficiency video is needed to reduce the requirement of transmission channel bandwidth. Both elements are equally important for the portable video application.

To observe a generic video encoder system, it contains 5 key components, including transformation, quantization, motion estimation (ME), motion compensation and entropy coding. As shown in Figure 1, the generic video system contains 2 coding paths, intra mode and inter mode. For the intra mode, the smallest coding unit, macroblock (MB) is transformed and quantized before entering entropy coding module for video coded bitstream. For inter mode, ME is performed first to reduce the inter-frame correlation and generate the prediction error image. The prediction error image is transformed and quantized, and the entropy coding module encodes the quantized coefficients as coded video stream. The reference frame during ME comes from the reconstruction of previous coded frame in the motion compensation module. The motion compensation module reconstructs the previous coded frame by referring to the motion vectors (MVs) from ME module. Figure 2 shows the computation complexity analysis in a video encoder system. Here, we illustrate MPEG-4 reference software (N4025) [3] as an example to analyze its workload distribution. From Figure 2, we can find ME consumes more than a half of the encoding power. This means reducing the ME complexity can gain the most in the encoder system optimization. Toward a low power and high throughput design, optimization in ME plays the key rule.

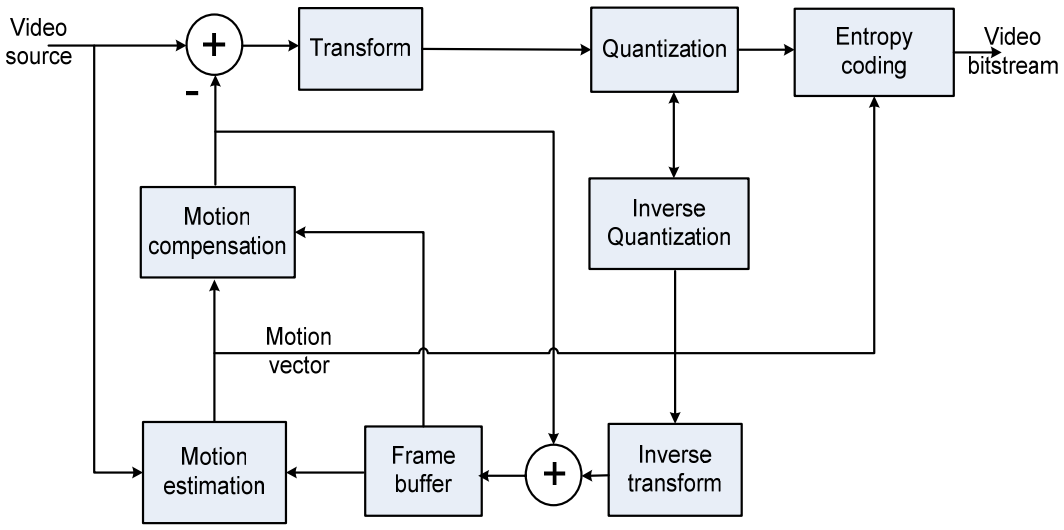


Figure 1. Block diagram of a generic video encoder system.

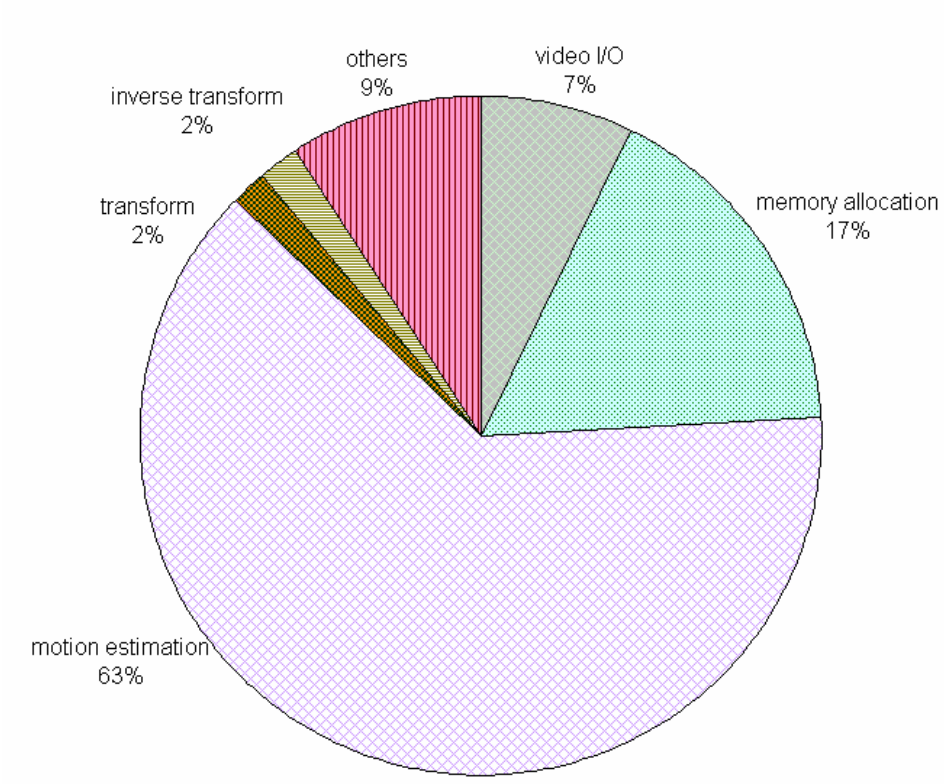


Figure 2. Workload distribution in video encoder [source: MPEG-4 reference software N4025 [6]].

In the advanced profile of video standards such as MPEG-1/2/4 or H.26x [1]-[5], bi-directional inter prediction (also called B-frame) is adopted to improve inter prediction precision and provides better coding efficiency. The B-frame coding scheme allows us to

reduce the prediction errors in ME by using both the forward and backward reference frame. As shown in Table 1, 0.4~2.0 dB PSNR improvement can be observed by replacing one or two P-frames with B-frames. But with the improved coding efficiency, the encoder complexity also increases with the number of frames used for motion search. Around 50%~130% increase of total encoding time are observed when one or two B-frames are used. Thus it poses a challenge to balance the coding complexity and coding efficiency. Meanwhile, the memory access bandwidth is increased proportional to the number of prediction frames. This also means that it raises a challenge for ME to balance the search complexity and search performance under bi-directional prediction scheme.

A few dedicated design solutions are proposed for B-frame search [7] [8]. But most of the existed algorithms are software level solutions. For hardware design, the conventional methods treat B-frame motion search as 2 iterations of P-frame motion search. Such designs sequentially process the forward frame first and then the backward frame (or vise versa).

Table 1. R-D performance for 3 different types of GOP structure with M=1, M=2 and M=3.

GOP	***Bitrate***	***Foreman***	***Akiyo***	***Flower***	***Mobile***	***Football***	***Tempete***
M=1	256	30.84	41.61	23.86	23.38	26.81	26.04
	512	34.18	43.33	26.11	26.18	27.86	28.78
	1024	36.89	44.43	29.3	29.25	31.75	31.55
M=2	256	30.49	41.85	23.88	24.26	26.91	27
	512	34.62	43.47	26.56	27.69	27.38	29.89
	1024	37.37	44.95	29.71	30.65	31.65	32.55
M=3	256	29.86	41.89	23.71	24.55	26.94	27.23
	512	34.44	43.42	26.48	28.15	27.13	30.14
	1024	37.27	45.09	29.58	30.99	31.45	32.76

Table 2. Average encoding time in seconds for 3 types of GOP structures[1].

Sequence	***M=1***	***M=2***	***M=3***
Akiyo	129	163	202
Flower	133	232	388
Football	137	252	424
Foreman	144	228	343
Mobile	166	282	456
Tempete	134	213	328

Considering the B-frame search structure, both forward and backward frames use the same current search blocks. To reduce chip memory access bandwidth and the power consumption, we need to fully utilize each memory access of current blocks for forward and backward frames. But if we break the sequential iterations to be parallel processing of forward and backward frames, it raises some difficulties such as larger size of local memory requirement and more silicon area for parallel processing. To solve the difficulties, an efficient algorithm needs to be applied to balance the tradeoff between higher memory access bandwidth in sequential design and the extra cost of design in parallel architecture.

1.2 Thesis Scope

In this thesis, we focus the bi-directional ME design on low power application. This low power design contains three sub-modules including integer pel ME (IME), mode decision (MD) and sub-pel ME (SME). In IME, we implement a new parallel binary search architecture which can allow parallel processing of forward and backward searches in binary format. Since the memory access is considered as one of the major sources for power consumption, the proposed parallel binary search can greatly reduce on-chip memory access compared to the conventional 8-bit design solutions. In MD, we integrate the MD module into SME to avoid two loops processing of MD and SME to save power. The proposed MD architecture adopts a new one-dimension algorithm to reduce longer latency in the original two-dimension and avoid hardware idling. In SME, a low power sequential design solution is

[1] The simulation is based on MPEG-4 reference software [6] with Pentium 2.8G CPU and 512 MB ram. The average encoding time under three different target bitrate (256, 512, and 1024 kbps) is presented.

adopted to balance the system pipelining for IME, MD, and SME.

The remainder of this thesis is organized as follows. In Chapter 2, we review the algorithms and architectures of bi-directional ME. We also propose a low power solution for bi-directional ME. The proposed algorithm is described in Chapter 3. Chapter 4 describes the architectures of IME, MD and SME modules. Experimental results and analysis are given in Chapter 5 to demonstrate the improved performance in power consumption. Chapter 6 concludes this work.

Chapter 2 Analysis of Bi-directional Motion Estimation

This chapter reviews the previous design solutions for B-frame search. We analyze the design challenge and then propose a low power solution.

2.1 Bi-directional Motion Estimation Algorithms

In this section, we review bi-directional ME algorithms. Two main categories of bi-directional ME algorithms are reviewed. One is area overlap and some relating algorithms, which takes advantage of the relationship between MV of current B-frame and that of future reference frame. The other is hybrid block-matching algorithm, which uses mean absolute error (MAE) between the MB in the current B-frame and the corresponding MB in the reference frame with MV (0, 0) to find the MBs those are not compensable.

2.1.1 Area Overlap and Related Algorithms

If the motion of each block is constant, MV of current B-frame should be proportioned to that of future reference frame. Thus MVs obtained from future P-frame can be reused to omit unnecessary ME for bi-directionally predicted B-frame. Based on this idea, some algorithms are proposed such as area overlap and vector propagation [7], [11].

Firstly, area overlap algorithm is reviewed. Assuming the GOP structure is IPBPB, the first three frames are coded in the following order: $I_0P_2B_1$, where the suffix represents the frame number. After P_2 is coded, MV of each block in frame P_2 (MV_P_2) is available. As the MB moves back along MV_P_2 to a block in frame I_0, it makes a projection on frame B_1. The projected block intersects with one to four MBs in frame B_1 as shown in Figure 3 and Figure 4. For each MB in frame B_1, the area overlap (AO) with projected block can be calculated as followed:

$$AO = \begin{cases} (L_1 - |x-u|)(L_2 - |y-v|), & if \quad |x-u| \le L_1, |y-v| \le L_2 \\ 0 & , otherwise \end{cases} \tag{1}$$

AO is the area overlap between a certain MB in frame B_1 and the projected MB from future reference frame P_2. L_1 is the horizontal dimension of a MB and L_2 is the vertical dimension of

a MB. (x,y) represents the co-ordinates of the top left corner of the MB in frame B_1 and (u,v) represents the co-ordinates of the top left corner of the projected MB. According to the equation, we can find out which projected block from frame P_2 possesses largest area overlap with one certain MB in frame B1. Then its MV is appropriately scaled to give forward and backward MV.

When the motion is not linear, area overlap algorithm yields poor result. To solve this problem, the authors in [12] propose MV interpolation and search method. After applying area overlap method, it further fine tunes the MV by performing search in small search range. The similar technique can be also found in [7].

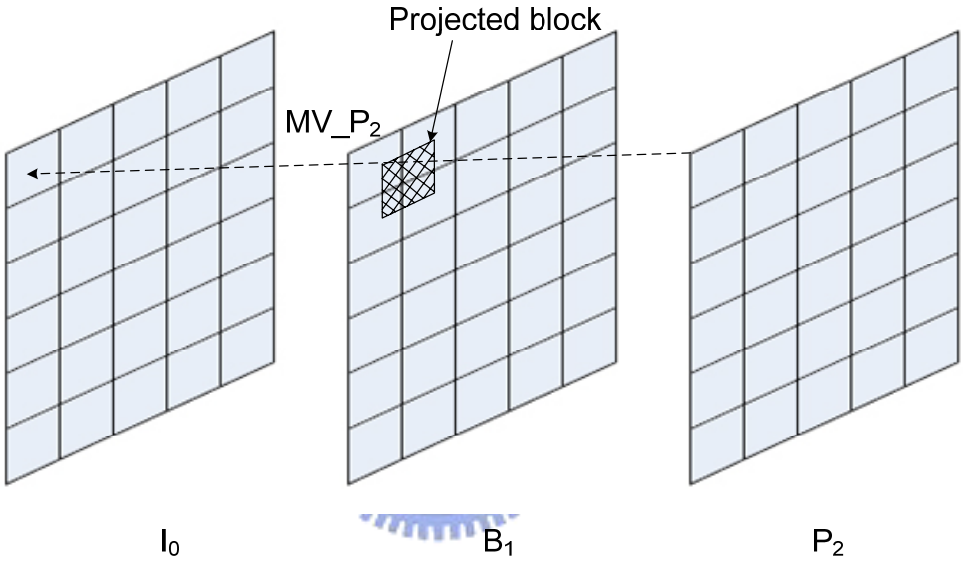


Figure 3. Projected block on frame B_1 when one MB in frame P_2 moves back along MV_P to a block in frame I_0.

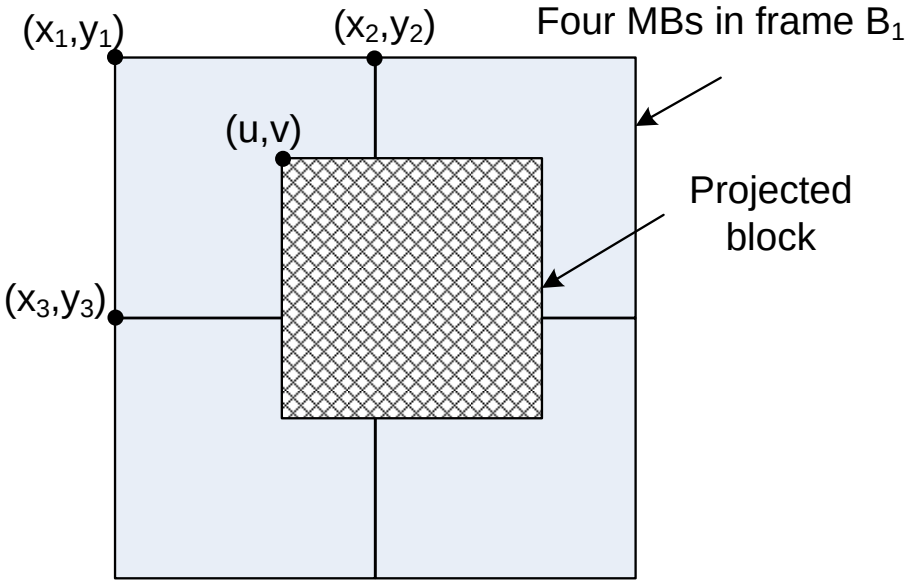


Figure 4. P-frame MB's projection in frame B_1.

2.1.2 Hybrid Block Matching Algorithm

There is about half of the MBs are not compensable in a genetic video sequence. The MBs those are not compensable could be: (a) those corresponding to a stationary background; (b) those corresponding to an uncovered background and new objects where no information can be found from the previous reference frame. Therefore, the main idea of hybrid block-matching method [13] is to find the MBs those are not compensable by using MAE between the MB in the current B-frame and the corresponding MB in the reference frame with MV (0, 0).

The flow of hybrid block-matching algorithm is described as followed. The GOP is set as I-B-B-P-B-B-P-B-B in display order. The I-frame is encoded first and then the next P-frames. After the encoding of P-frame, each MB in B-frame is classified as compensable or not. It assumes that the average MAE for the B-frames will not be much larger than the average MAE for the P-frames. Two thresholds are defined to classify the type of MBs:

$$T_i = R_i \times MAE_{avg}^{P}, i = 1,2 \qquad (2)$$

MAE_{avg}^{P} is the average MB MAE of two reference frames. R_1 and R_2 are two predefined ratios. If the MAE of current MB is smaller than T_1, the MB is defined as a type-A MB which is not compensable. If the MAE_{MB} is large than T_2, then we compare it with the MAE of the corresponding MB in reference frames (past and future frames). If there is no significant

improvement, this MB is identified as a type-B MB. For other MBs, if all four blocks are compensable, the MB is identified as compensable; otherwise it is classified as a type-C MB which is not compensable. Thus ME for those MBs which are not compensable can be skipped. The percentage for the found MBs which are not compensable represents the reduction in computation. And MAE is used not only for the matching criteria, but also for classifying the MBs. Therefore, almost no overhead computation is needed for classifying the MBs which are not compensable. The experiments show the speedup factor for B-frames is about 57 on average compared with full search method.

2.2 Bi-directional Motion Estimation Architectures

In this section, we review conventional architectures for P-frame search. There are many ME architectures such as full search block-matching (FSBM), three step search (TSS), diamond search and so on. FSBM is preferred for hardware implement due to its regular search pattern and simple control overhead. However, it has the highest computation complexity. So besides FSBM, we also review TSS architecture due to its simplicity. These architectures for P-frame search can be extended to B-frame search by processing forward search first and then the backward search (or vise versa) sequentially. Thus B-frame search can be treated as 2 iteration of P-frame search.

2.2.1 Full Search Block-Matching (FSBM)

To overcome the computational costs of FSBM, several different architectures have been proposed over the last few years such as systolic array, adder tree and etc. Among them, systolic array can achieve high hardware utility and be easy for data reuse. So we review the most common FSBM architectures implemented with 1D and 2D systolic array in [14].

Figure 5 and Figure 6 illustrate the FSBM architecture implemented with 1D systolic array, which consists of N processing elements (PEs) for N pixels of a single row in the reference block of size N×N. To achieve parallel processing, the candidate blocks enter serially and are shifted for each clock cycle from one PE to the other. The systolic array calculates the absolute differences in parallel and then sends the partial sums of the absolute differences (SAD) to the parallel adder to compute the SAD of each block matching. Then the differences are fed to controller to determinate the MV with minimum distortion. For a

reference block of size (K×K) and ±P search range, the total number of clock cycles is (2P+1)×(K+2P)×K. With K=16 and P=16, it takes 25344 cycles. The 1D systolic array has less area than other types but the drawback is the increase in the number of clock cycles required to finish the calculations.

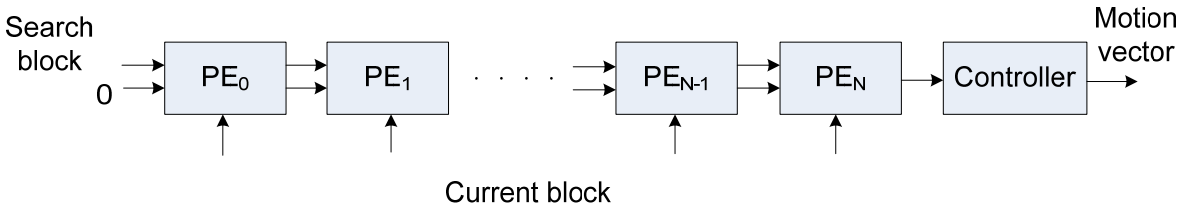


Figure 5. 1D systolic array for FSBM architecture.

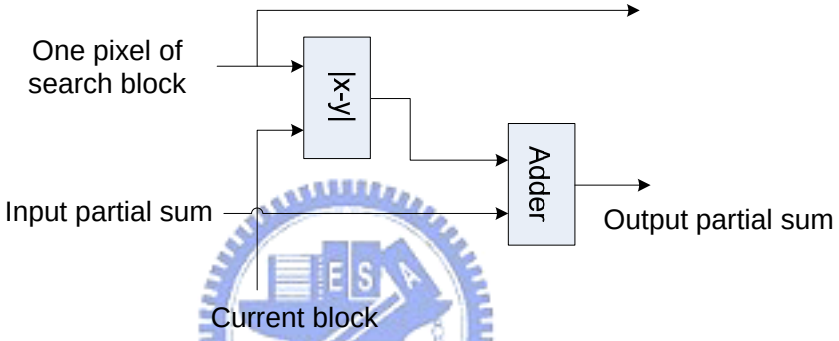


Figure 6. PE structure in 1D systolic array.

For FSBM architecture implemented with 2D systolic array, there are K^2 PE for K×K pixels in the reference block. In general, the PE of 2D systolic array is the same with that of 1D systolic array but there are K^2 PEs. The PE in the systolic array computes the absolute difference between pixels of reference block and current block and then forwards the partial sum of absolute differences from the row below to the row above. For a K×K reference block and a window size of K+2P the total number of clock cycles is $(K+2P)^2$. With K=16 and P=16, it takes 2304 cycles. But it increases the hardware cost and power consumption because of more number of PEs and more interconnect area. The analysis of 1D and 2D systolic array shows the trade-off between the number of PE, processing rate and power dissipation.

2.2.2 Three Step Search (TSS)

The main architectural problems of TSS are the variable distances between candidate locations and the sequential execution between steps. They result in unpredictable data access.

Besides, these problems complicate the control scheme, lower the efficiency of computation kernel, and make the data-reuse difficult. To solve these problems, memory-efficient array architecture with data-rings to implement TSS is proposed in [15]. This architecture not only simplifies the control scheme with a regular raster-scanned data flow, but also shortens the latency by using a comparator-tree structure.

As shown in Figure 7, 9-cells array architecture with data-rings is employed. It can evaluate the 9 candidate locations in parallel and accumulate the absolute difference of each candidate block sequentially. Each basic cell is composed of a memory module and a PE. The PE consists of an absolute difference unit, an accumulator, a final-result latch, and a comparator. The comparators are connected in a tree structure. Figure 8 shows the architecture of one basic cell. The memory modules store the search area pixels for prediction. Memory interleaving technique is used to provide a solution to parallel data access. The search block pixels are interleaved to these 9 memory modules. At each cycle, the current block pixels are sequentially broadcast to all PEs in raster-scanned order. The search block pixels required by each PE are read into the PEs in parallel. Each PE shifts the partial-accumulation to the adjacent PE in horizontal ringed direction to accumulate the next partial-accumulation of the same search position. After every 16 cycles, each partial-accumulation shifts and accumulates in a vertical ringed direction. After 256 clock cycles, the final accumulation-result of each candidate is produced in the final-result latch of each PE. Finally, the 9 latched MADs can then be sent to the comparator-tree to get a minimum MAD. Thus it takes 256 clock cycles for matching at each step and at least 768 cycles to complete TSS.

Because the number of checked positions is much reduced, the required running cycle of TSS architecture is much less than FSBM. But it results in PSNR degradation. So it shows the trade-off between PSNR performance and running cycle.

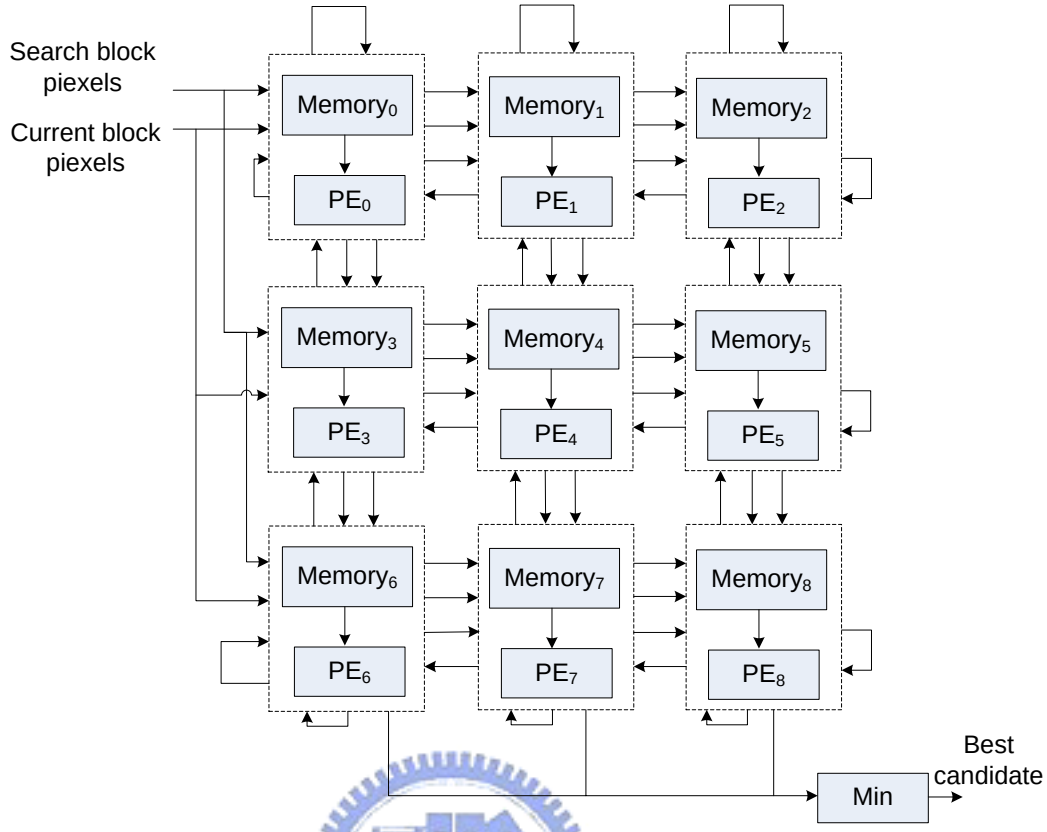


Figure 7. A memory efficient array architecture with data-rings for TSS in [15].

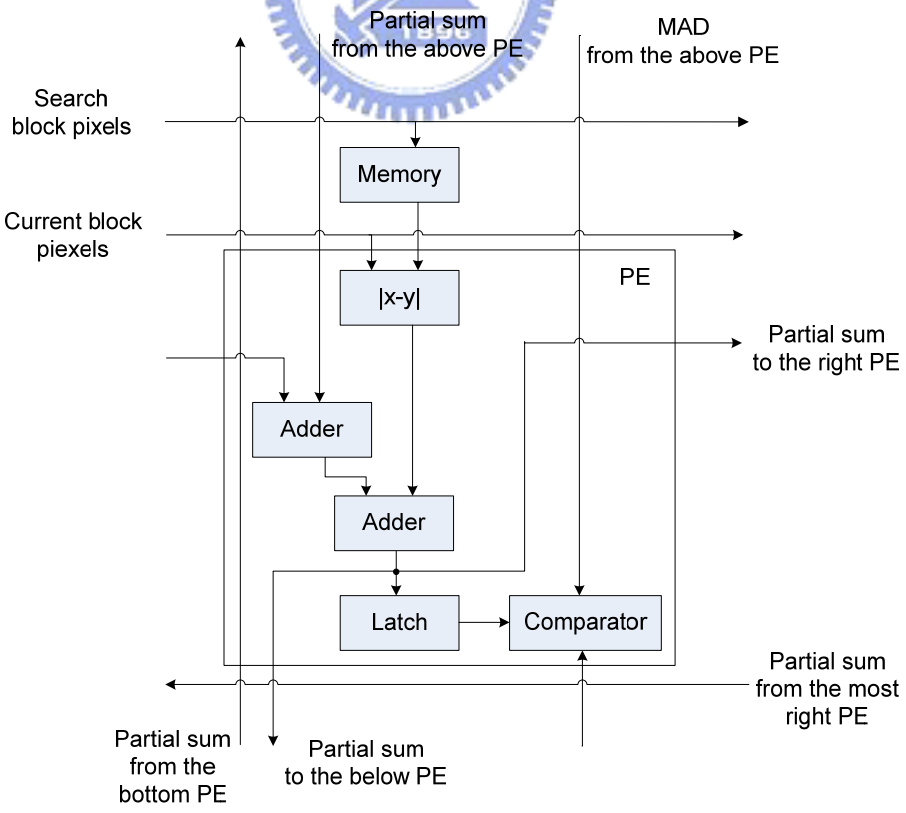


Figure 8. Structure of one basic cell for distortion calculation in TSS architecture.

2.3 Design Challenges and Proposed Solutions

The B-frame search suffers greater design challenges in real-time processing and low power consumption than P-frame search. It needs more computational power in order to meet the same encoding throughput as P-frame only search. A complete ME structure contains 3 sub-modules including IME, MD and SME, and shows different design challenges toward low power design.

A. Integer pel motion estimation (IME)

The IME sub-module dominates the whole power consumption in ME design. More than 60% ME power is consumed in IME. In conventional low power design methodology, it designs a low power IME and applies sequential operation for each prediction frame. But to consider the B-frame search structure, there is still redundancy for forward and backward searches. The sequential search uses the same current search blocks for both searches while one of them can be skipped to reduce on-chip memory access and power consumption. But the conventional parallel processing architecture for multiple frames which trades larger silicon area for higher processing throughput is not suitable for low power design. This thesis proposes a new parallel binary architecture to allow parallel search in binary format. This can efficiently reduce the silicon area and power consumption while still maintaining the advantages of parallel architecture.

B. Mode decision (MD) and sub-pel motion estimation (SME)

The convention ME design treats MD and SME as two separated operation loops. Since MD and SME use the same current and reference search block data, the two loop design methodology is not suitable for low power design. So, the first solution is to merge the MD and SME in single loop processing. The MD compares inter and intra cost to determine the coding mode. The original method for cost calculation is based on 16×16 calculation results which has a longer processing latency. The second solution is to apply an efficient one dimension cost calculation method to reduce the longer latency and processing cycle counts for low power design.

Chapter 3

Algorithm of Bi-directional Binary with Sub-pel Motion Estimation

The proposed bi-directional ME algorithm contains IME loop and merged MD and SME loop. The IME is based on ABME [10] and designs a bi-directional architecture for parallel binary search. The merged MD and SME (referred to as MD-SME) apply a new one dimension cost calculation into the sequential processing SME algorithm.

3.1 Review of All Binary Motion Estimation (ABME)

The ABME is composed of two major modules, including pre-processing engine and motion search engine. The pre-processing engine is to preprocess the full pel image pixel data to generate binary patterns for search. The search engine applies a pyramid search structure to allow three-layer of binary search.

3.1.1 Design Flow of All Binary Motion Estimation

Figure 9 shows the ABME design flow. Firstly, it adopts the frame-based pre-processing method to construct a three-layer binary pyramid in three different resolutions indicated as Level 1 (LV1), Level 2 (LV2), and Level 3 (LV3), which represents the binary representation from the coarsest to the finest resolution respectively. Then the pyramid search performs coarse-to-fine search from LV1 to LV3 sequentially. The LV1 search is a full search with a search range of $\pm$(R/4-1), where R indicates the search range in integer resolution. In LV2 search, six MV predictors from neighboring MBs, including co-located MB in previous frame, top, top right, and left, MV from LV1 and (0, 0) are checked first. If all these motion predictors are equal to zero, a fine tuning operation with a ± 2 cross pattern is performed. If the answer is no, an operation is applied to find the motion predictor with minimum SOD among the six candidates. This is known as the voting process. Then, we select this predictor to perform a ± 1 cross pattern search. LV3 search inherits the LV2 MVs and performs a ± 2

fine tuning search. After the final MV of a 16×16 MB is determined, it is used as the initial center for the 8×8 search with ±2 search range.

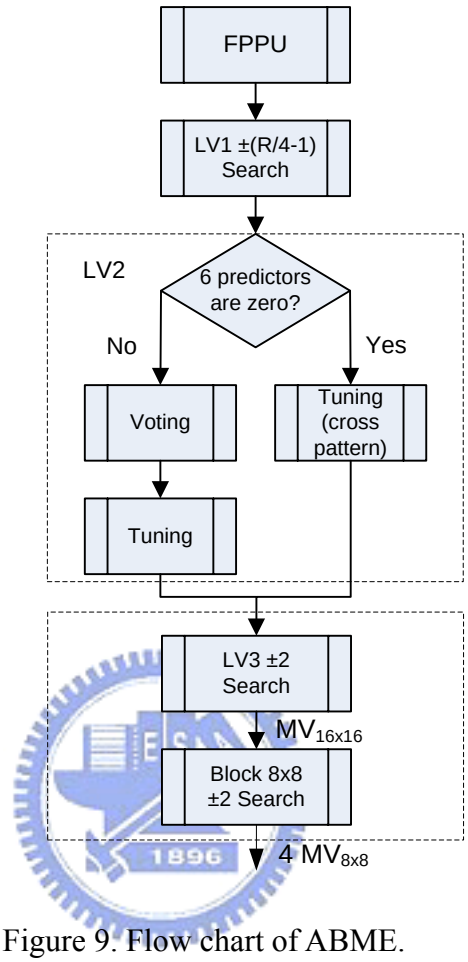


Figure 9. Flow chart of ABME.

3.1.2 Frame-based Pre-processing unit (FPPU)

Similar to the hierarchical ME algorithms, the ABME is also based on the pyramidal structure. The binary pyramid used in ABME is composed of three layers, which is indicated as LV1 to LV3, from the coarsest to the finest resolution. In Table 3, we summarize the resolution at each level. To construct the binary pyramid of three layers, the full pel image pixel data is processed through binarization and sub-sample procedures. Since the binary representation of one pixel is zero or one, a threshold is required for determination. For a pixel ‘A’ and surrounding pixels ‘B, C, D, and E’ as shown in Figure 10, the threshold (TH) is defined as following equation:

$$TH = (B+C+D+E)/4 \quad (3)$$

And the binary value is determined as followed:

$$A' = \begin{cases} 1, & if\ (A \geq TH) \\ 0, & otherwise \end{cases} \tag{4}$$

Where A' denotes the binary value of pixel A. Figure 11 shows an image before and after binarization. It can be seen that binary image preserves the edge of the object. Thus ME can obtain a fair MV with binary image even though the bit depth of one pixel is reduced to one bit.

Table 3. Frame and MB size at each level.

	LV1	***LV2***	***LV3***
fame	$\frac{W}{4} \times \frac{H}{4}$	$\frac{W}{2} \times \frac{H}{2}$	$W \times H$
MB	4×4	8×8	16×16

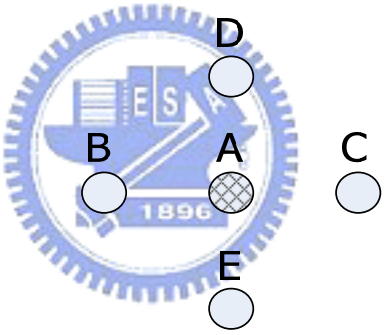


Figure 10. Threshold obtaining for binarization.

(a) (b)

Figure 11. (a) An image in 8-bit representation (b) An image in binary representation.

To construct an image pyramid, the image at coarser resolution (upper level) is obtained through sub-sample procedure. ABME combines sub-sample procedure with binarization to save the computational complexity as illustrated in Figure 12. For binarization, the mean of neighboring pixels is taken as the threshold value, which can be considered as low-pass filtered data. By down-sampling the filtered data, the image at coarser resolution is obtained. By repeating such procedure, we can construct a binary pyramid as shown in Figure 13.

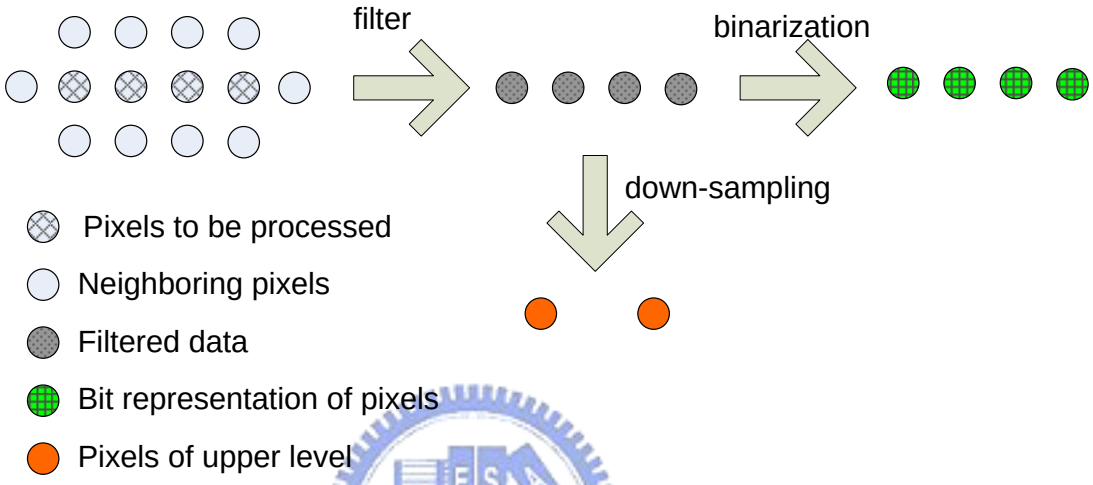


Figure 12. Combination of binarization and sub-sampling.

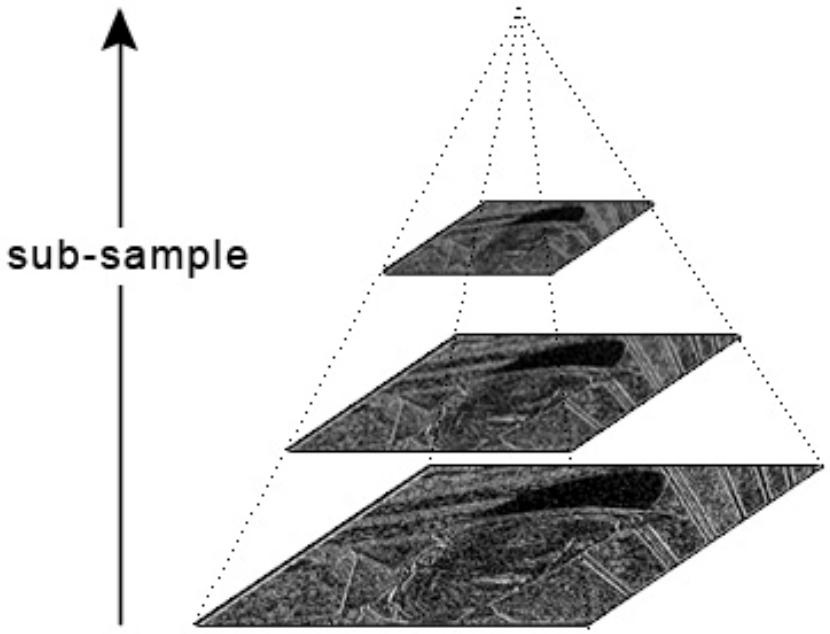


Figure 13. A binary pyramid structure used for ME.

3.1.3 Three Layer Binary Pyramid Search

After constructing a binary pyramid of three layers, ABME performs ME from LV1 to LV3 sequentially. Since the resolution of LV1 image is downsampled by 4, a full search with a search range of ±(R/4-1) is performed at LV1 search, where R is the search range in the original resolution. Then the MV_LV1 is passed to LV2.

In LV2 search of ABME, it defines six MV predictors as followed:

MV_LV1: predictive MV transferred from LV1 full search

MV_UR: MV of upper right MB

MV_U: MV of upper MB

MV_L: MV of left MB

MV_P: MV of the same position but in the previous frame

MV_Z: zero MV (0, 0)

Depending on these predictors, LV2 search in ABME contains two parts: one is search with cross pattern and the other is voting and tuning. If the six candidates are (0, 0), it performs a search with cross pattern. Otherwise, it checks these six positions to see that which candidate results in the smallest SOD. And then it further refines the best candidate by tuning within a small range. The final MV is considered to be the best predictive MV indicating the initial position for LV3 Search. LV3 search inherits the LV2 MVs and performs a ± 2 fine tuning search. After the final MV of a 16×16 MB is determined, it is used as the initial center for the 8×8 search with ±2 search range.

Since the images at three layers are in binary form, the commonly used matching criterion SAD is not suitable for binary representation algorithms. The matching criterion for binary representation is called sum of difference (SOD), which is simply Boolean operation XOR as shown in Equation (5).

$$SOD(u,v) = \sum_{(x,y)\in Block} \left[S_k(x,y) \oplus S_{k+1}(x+u, y+v) \right] \tag{5}$$

SOD : Sum of Difference

x, y: current horizontal/vertical pixel location

$\oplus$: 1-bit XOR operation

$S_k(x,y)$: binary representation at (x, y) on current frame

$S_{k+1}(x+u, y+v)$: binary representation at (x+u, y+v) on reference frame

SOD is similar to SAD but it doesn't have to care about the sign information and be easy for hardware implementation.

3.2 Bi-directional Motion Estimation Algorithm

The proposed bi-directional ME algorithm as shown in Figure 14 contains three processing stages, IME, MD, and SME.

A. Integer pel motion estimation (IME)

In IME, a bi-directional binary ME (BBME) is developed based on our prior work [10], and constructed on MPEG-4 standard. Firstly, a three-layer binary pyramid in three different resolutions is constructed for the three-layer binary pyramid search. For the purpose of MB level pipelining, the pre-processing unit for the three-layer binary pyramid bitplane is designed using MB-based pre-processing (MBPPU). The three binary bitplanes are referred to as LV1 for 1/16 original size, LV2 for 1/4 original size and LV3 in original resolution. After the binary pyramid bitplanes are constructed, three layers of binary search are applied as shown in Figure 14. The design flow is summarized as below.

(1) Apply MBPPU to construct a three layer binary pyramid.

(2) Apply LV1 full search with ±(R/4-1) search range.

(3) Apply a fine-tuning operation using 5 predicted MV candidates with ±1 cross pattern search in LV2. The MV candidate with minimal matching cost is chosen as the search center in LV3.

(4) Apply LV3 full search with ±2 search range for block size 16×16.

(5) Apply LV3 full search with ±2 search range for each 8×8 blocks from the same search center as block size 16×16 so that we can merge these two searches in hardware implement.

B. Mode decision (MD) and sub-pel motion estimation (SME)

The MD calculates the intra and inter R-D cost for MD after IME and before SME, and SME is applied if inter mode is selected. As shown in Figure 14, the MD calculates the intra cost according to Equation (6), which is a form in sum of difference with the mean value in

that MB.

$$\sum_{j=0}^{15}\sum_{i=0}^{15}\left|c_{i,j}-\bar{c}\right|,\quad \bar{c}=\frac{1}{256}\sum_{j=0}^{15}\sum_{i=0}^{15}c_{i,j} \tag{6}$$

The inter cost is from the sum of prediction difference for block 16×16 or 4 block 8×8. If the intra cost is higher than either one of the inter costs for block 16×16 or 4 block 8×8, the inter mode is chosen. Otherwise, intra mode is adopted for encoding. If inter mode is chosen, SME is applied. In P-frame search loop, block 8×8 and block 16×16 are applied. In B-frame search loop, only block 16×16 is applied. The design flow for MD and SME is summarized as follows.

(1) Apply MD using intra cost in Equation (6) and inter costs for block 16×16 and 4 block 8×8. If either one of inter costs is smaller than intra cost, inter mode is chosen. Otherwise, go to step (6).

(2) Apply bi-linear interpolation for SME.

(3) Apply ±1 half-pel search for block size 16×16.

(4) Apply ±1 half-pel search for 4 8×8 blocks if this is a P-frame search loop.

(5) Compare the R-D costs for block 16×16 and 4 block 8×8. The one with smaller cost is chosen as the final MV for inter mode.

(6) Calculate the residue for the decided coding mode in step (1) or step (5).

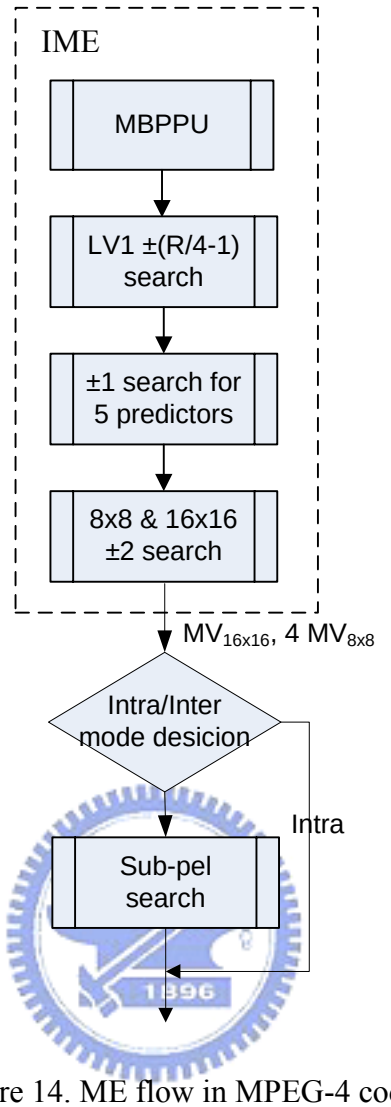


Figure 14. ME flow in MPEG-4 coding.

3.3 Hardware Efficient Design Features

The proposed bi-directional ME algorithm adopts 4 hardware oriented design methods for low power design applications.

A. MB-based pre-processing unit (MBPPU)

The proposed MBPPU is designed for low cost and low data requirement under MB level pipeline hardware architecture. It can reduce the required bus bandwidth and hardware design cost by mirroring the pixels around MB boundary. The FPPU in ABME [10] can cover the global image context information during binarization process but it brings lots of design difficulties in hardware design. For easy pipelining in MB level, the ABME realizes the FPPU in MB basis. Such a design method can maintain the same performance as FPPU. But it suffers bigger problem in memory access bandwidth since extra image data is needed as

shown in Figure 16(a). To construct 4×4 binary block at LV1, one 6×6 block is required. Thus one 14×14 and 30×30 block is needed at LV2 and LV3, respectively. Compared with the original MB size of 16×16, the data is increased about 3.5X.

To solve this problem, we propose a new MBPPU method by using self-padding to reduce the heavy memory bandwidth requirement as shown in Figure 15. At LV3, the block size of K×K is used for binarization. So the downsampled block size at LV2 should be (K/2)×(K/2). If there are p pixels absent for LV2 binary pattern generation, the downsampled block is self-padded by mirroring the pixels around the boundary. And the padded block is used for LV2 binary image and LV1 downsampled image generation. The similar technique is applied to LV1. In this way, the required data is decreased significantly, which accompanies reduction of memory bandwidth and computation complexity. To find out the suitable size of block at LV3, block sizes of 16×16, 18×18, and 20×20 are tested in our simulation as shown in Table 4. Compared with the block size of 30×30 used in ABME, the PSNR drop of 16×16 block is about 0.5dB while those of 18×18 and 20×20 blocks are within 0.2dB. Considering the trade-off between required data and PSNR performance, the 18×18 block size is adopted in our MBPPU.

Figure 16(b) illustrates our proposed MBPPU by self-padding to automatically generate the required MB boundary data for binary image generation. At LV3, we have 18×18 un-padded image data for the current layer binary image generation and LV2 downsampled image generation. For LV2, the downsampled block size is 9×9, which is one pixel absent for LV2 binary pattern generation. So the LV2 block is self-padded by one pixel. Then the padded 10×10 block is used for LV2 binary image and LV1 downsampled image generation. The same rule is applied for LV1 binary pattern generation. As a result, MBPPU can provide 177% data saving compared with FPPU.

Table 4. Comparison of different block size in pre-processing, test sequence is foreman at 512 kbps.

Block size	***16×16***	***18×18***	***20×20***	***30×30***
Required data (bits)	256	324	400	900
PSNR (dB)	33.51	33.87	33.91	34.01

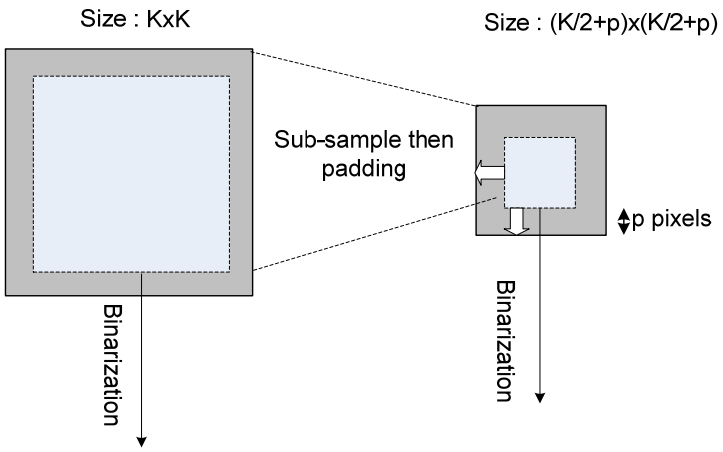


Figure 15. Illustration of self-padding of downsampled block.

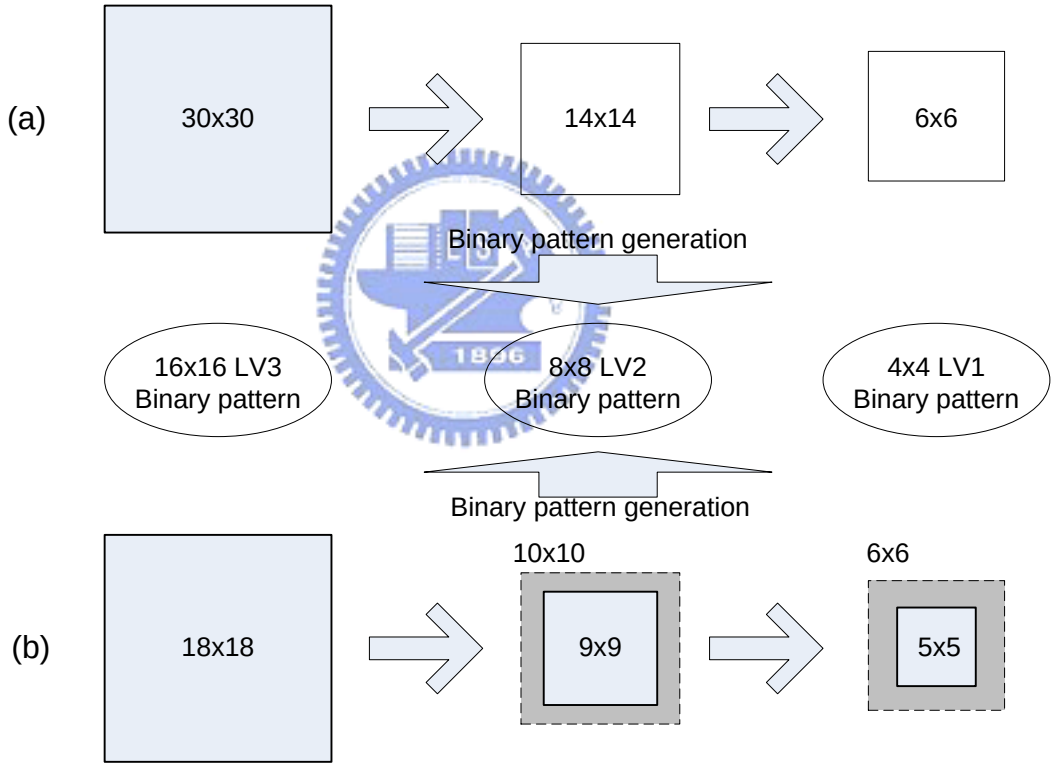


Figure 16. Pre-processing flow for binary pattern generation (a) FPPU with MB realization (b) Proposed MBPPU.

B. Hardware efficient Level 2 design

The proposed LV2 search removes the branch operation to reduce the extra computation and memory access. The LV2 search of ABME contains two conditional paths. The first path is to perform a ±2 cross pattern search if we found six MV candidates are all zero motion.

The second path is to perform a ±1 cross pattern search after the final MV is found from the six MV candidates. For hardware implementation, the original method poses 2 potential problems. Firstly, branch operation needs the first time memory access to check whether six candidates are all (0, 0). Second, if second path is chosen, we need the second time memory access before we perform the final decision with a small range of search. To consider the hardware implementation, the first 2 data access for the conditional check can be saved to avoid the multiple accesses to the same MV candidate. So, in the proposed LV2 search, we remove the branch check and the decision of best MV candidate with minimum distortion. The new flow allows those candidates to be checked sequentially and best candidates are selected among them. One thing needed to be noted is the collocated MV candidate is removed from the LV2 candidate list due to the difficult memory access to that MV information.

C. Integration of 8×8 and 16×16 searches

The two search loops for block 8×8 and 16×16 are merged for power saving in hardware. The MPEG-4 standard supports block sizes of 8×8 and 16×16 search in integer pel resolution. In ABME ±2 full search is performed for 16×16 MB. Then 16×16 MV is used as the initial center of four 8×8 searches with ±2 search range, which implies that total 5 iterations of ME are needed for one 16×16 and four 8×8 blocks. To reduce the computational complexity, the proposed algorithm integrates 8×8 and 16×16 searches into the LV3 search. When performing ±2 full search, the SODs of one 16×16 MB and four 8×8 sub-blocks are calculated simultaneously. Such a modification can simplify the original 5 iterations of search into one combined search.

D. One dimension mode decision with shorter latency

A line-based intra cost calculation method is proposed to reduce the longer latency raised by Equation (6) in hardware design by using a one-dimension calculation solution as shown in Equation (7). For hardware implementation, the original two-dimension calculation in Equation (6) poses two potential problems. One is the mean value of current MB can not be obtained until the whole MB is read, which results in longer latency. The other is that we need two memory accesses. The calculation of mean value needs the first time memory access and the computation of the absolute difference the second one. To reduce the latency and save

memory access, we adopt a line based intra cost calculation as followed:

$$\sum_{j=0}^{15}\left(\sum_{i=0}^{15}\left|c_{i,j}-\left(\frac{1}{16}\sum_{i=0}^{15}c_{i,j}\right)\right|\right) \tag{7}$$

In the proposed line based intra cost calculation, after one row of current MB is read, we can compute the row average, and then the summation of absolute difference between the row average and each pixel of this row. This results in shorter latency and merges two memory accesses into one.

Chapter 4

Architecture of Bi-directional Binary with Sub-pel Motion Estimation

In this chapter, we realize the proposed bi-directional motion search algorithm in hardware. The system architecture and the partitioned modules IME and MD-SME are described in detail.

4.1 System Architecture

The proposed system architecture for the low power bi-directional motion estimator can be partitioned into two main modules, IME and MD-SME which are described respectively as below.

A. Integer pel motion estimation (IME)

The parallel binary search architecture is the key architecture which we used to design a low power bi-directional IME module. This architecture is not just to double the hardware, but to fully use every control or data information which can be shared to achieve the low power performance compared to the sequential search strategy. The parallel search architecture allows parallel processing of forward and backward searches under the control and data information sharing, and fully uses each memory access of the current search blocks to minimize total on-chip memory access. But the parallel architecture suffers double hardware design costs. So, the proposed BBME algorithm is used with the parallel architecture to reduce the design cost and also achieve the ME computation power saving using binary operations.

The IME architecture is shown in Figure 17. It contains a pre-processing unit (PPU) engine and a ME engine. The PPU engine consists of one local memory and three sub-PPU engines, LV1_PPU, LV2_PPU and LV3_PPU, for the generation of three binary bitplane. To reduce the external memory access latency, we adopt the ping-pong buffer as local memory. When reading the data of n_{th} MB from local memory, the data of $(n+1)_{th}$ MB is written in the

same time. The original 8-bit MB data is firstly stored in the local memory (LM_PPU), and then passed to LV3_PPU for LV3 binary MB and 1/4 size downsampled data for LV2_PPU. The 1/4 size downsampled data is passed to LV2_PPU to generate LV2 binary MB and 1/16 size downsampled data is passed to LV1_PPU to output LV1 binary MB. The three binary MBs are passed to ME engine for pattern matching. The ME engine designs three individual search units LV1, LV2 and LV3, and two SOD PEs. The ME engine performs LV1 ME to LV3 ME sequentially. The LV1 binary MB from LV1_PPU is stored in the on-chip memory of LV1 search unit, LM_CUR_LV1, and the forward/backward binary search window (SW) data is stored in separated on-chip memories, LM_SW1_LV1 and LM_SW2_LV1, respectively. For each MB search, the LV1 controller (CTRL_LV1) controls on-chip memories data access and passed to the shared SOD PEs for pattern matching. The matching results are sent back to CTRL_LV1 for the decision of LV1 MV. The LV2 search unit performs ±1 cross pattern fine-tuning from 6 MV candidates and the MV candidate with minimal SOD cost is used as the search center in LV3 search. The LV3 performs block 8×8 and 16×16 parallel search by sharing the same search information, but calculates the SOD separately to decide their individual final SOD costs. The final IME MVs for block 8×8 and 16×16 are generated simultaneously.

B. Mode decision and sub-pel motion estimation (MD-SME)

The core architecture to achieve the low power MD-SME is to merge the two loops of processing of MD and SME in single loop. To achieve the MD and SME concatenation, we design shared PEs for intra cost and inter SAD calculations. The MD-SME architecture is shown in Figure 17. It contains an intra cost engine, a inter cost engine, three shared SAD PEs, a mode determiner, and a MV determiner. For calculation of the intra cost, the 8-bit current MB is stored in the current MB buffer of intra cost engine and sent to average PE (Avg_PE) to calculate the mean value. Then the mean value and current MB are sent to the shared SAD PE for the computation of intra cost. For the calculation of inter SAD cost, 8-bits forward/backward SW data is stored in separate on-chip memories of inter cost engine, LM_SW1_SAD and LM_SW2_SAD, respectively. According to the MVs from IME module, the address generators (AGs) generate correct memory access addresses for the on-chip memories. The on-chip memory outputs data to interpolation PE (Interp_PE) to interpolated sub-pel data on the fly. Then the generated sub-pel data and current MB are sent to the shared

SAD PEs for SAD calculation. The intra and inter costs outputted by SAD PEs are sent to mode determiner for comparison. If the coding mode is determined as inter, MD-SME module continues to perform SME, The data flow of SAD calculation for SME is similar with that of inter cost. But the resulting SADs are sent to the MV determiner for final MV decision. Furthermore, due to the overlapped pixels between adjacent reference blocks in SME, three SAD PEs are designed to process multiple adjacent search locations in parallel for on-chip memory access reduction and power saving.

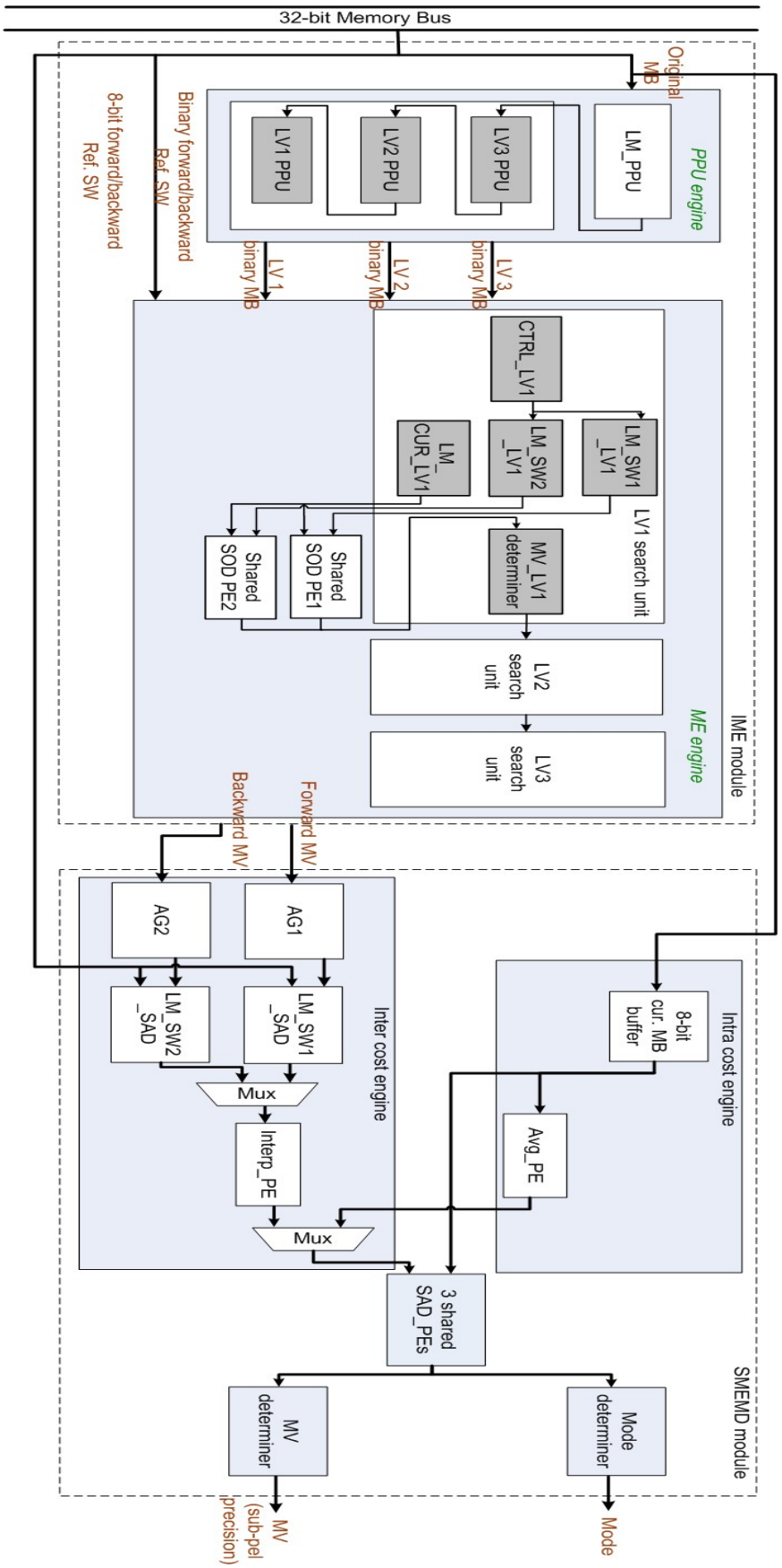


Figure 17. System level architecture consisting of IME module and MD-SME module.

4.2 Integer Pel ME (IME) Module

The IME realizes BBME in the design, and it consists of PPU and ME engines which are detailed below. The architecture design of BBME is based on that of ABME, and we make some modifications to fit our algorithm.

4.2.1 Parallel Binary Architecture

The proposed IME design is based on the parallel binary architecture which allows parallel processing of forward and backward searches to reduce power consumption and enhance the processing throughput. The memory access is considered a major source of power consumption. Reduction of on-chip memory access leads to less power. The parallel architecture for bi-directional search fully uses each memory access of the current search blocks to minimize total on-chip memory access. To efficiently use each access of the current search blocks, two on-chip memories are adopted to allow parallel forward and backward search. Figure 18 shows the functional blocks of the proposed parallel bi-directional architecture. The image data for current search blocks is stored in the local memory of LM_CUR via memory interface (MEM_IF) from the external off-chip memory while the forward/backward SW data is stored in separated on-chip memories, LM_SW0 and LM_SW1, respectively. For each block search, the controller (CTRL) generates correct memory access address for the on-chip memory. The on-chip memories output data to PE for pattern matching. The pattern matching module computes the difference between current and reference search blocks for each search candidate. The matching metrics could be in any form such as SAD, Sum of Square Difference (SSD), SOD, etc. In this thesis, the proposed bi-directional binary motion estimator adopts SOD as the binary pattern matching criterion. The matching results are sent to the comparator for final MV decision. Such a design flow can allow smooth parallel processing of forward and backward search for B-frame.

For the forward only P-frame search, this parallel architecture leaves half the hardware resources idle if no special design is considered. To solve this problem, a parallel P-frame search scheme is proposed. In the parallel P-frame search, the forward search data from LM_SW0 is mirrored to LM_SW1. The original forward search module searches the odd positions of the P-frame search and the original backward search module searches the even

positions (or vice versa). Such a design methodology can make the whole design 100% busy.

Compared with conventional designs, the parallel architecture contains five major advantages including:

- **One-time access of the current search blocks**: This reduces the redundant on-chip memory access and power.
- **Single broadcast of control information**: For the B-frame search, same memory access address and vector information are needed for the processing. The parallel architecture shares these control parameters.
- **Full utilization of PEs**: The design is fully utilized for both P-frame and B-frame searches. The parallel P-search uses both modules to allow another form of parallel processing.
- **Lower working frequency**: The parallel architecture halves the running cycle leading to lower power consumption or doubled throughput.
- **Joint optimization of bi-directional searches**: The parallel processing of bi-directional search allows joint optimization with very minor extra efforts.

Table 5. Summary of 3 design methodologies by adopting parallel or sequential architecture with different pixel bit-depth for B-frame search.

	8-bit sequential	***1-bit sequential***	***1-bit parallel***
On-chip memory capacity	C*8	C	C*2
Peak memory bandwidth	8*(B_1+B_2)	(B_1+B_2)	(B_1+2*B_2)
Execution cycles	$\geq 2K$	$2K$	K
Processing bit-depth	8-bits	1-bit	2 parallel 1-bit

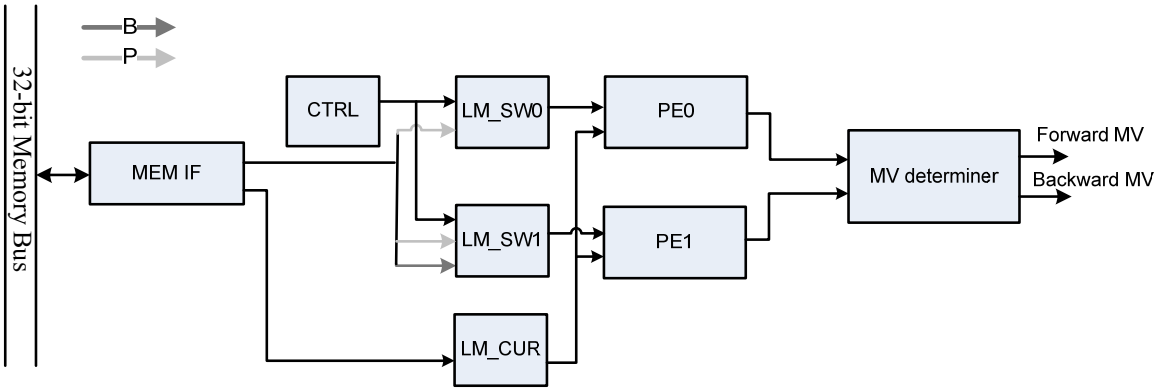


Figure 18. Illustration of parallel bi-direction search.

4.2.2 Pre-processing Unit Engine

Figure 19 shows the architecture of PPU which consists of a local memory and three level PPUs, named LV1, LV2, and LV3 PPU respectively. The structure for the three levels of PPUs is quite similar, except the intermediate register arrays and the processing bit width. The original 8-bit current MB is transmitted from external memory and stored in local memory. To avoid bus transmission issues, we adopt ping-pong buffers. This can allow the parallel processing of current MB preprocessing and next MB data transmission in parallel. After the whole current MB is stored in the local memory, the pixels are written to the register files row by row. In our design, three rows of pixels are needed for binarization, so the PPU_PE waits until the third row of data is ready. The output sub-sampled pixels are sent to next level of PPU. By repeating the same procedure, a three level binary pyramid is constructed.

Each level of PPU contains three main components

1. Register file arrays

Register arrays are designed as the intermediate buffers for PPU operations. Since the pixels in top row and bottom row are needed for preprocessing, three rows of buffer are designed. Smaller buffer size is achieved by using row rotator, which will be described latter.

2. Row Rotator

The row rotator as shown in Figure 20 is designed to rearrange the input data order to fit the PPU_PE processing order. In this design, three rows of register files are used which force the fourth row data to be put in the location of the first row of register

files. Putting the data of next row to the location of the first register file changes the data order not synchronous with processing order. To avoid complex conversion of input data order to hardware processing order, this row rotator is adopted

3. PPU_PE

The PPU_PE calculates the average of the four neighboring pixels, and uses this mean value to compare with the current pixels for binary patterns. The averaged pixels are used as the downsampled data for next level of PPU.

To analyze the design timing for PPU, 2 cycles are used to fill one row of register file at LV3 PPU, and total 36 cycles are used to fetch 18 rows of current blocks. The processing from LV3 to LV1 PPU takes 12 cycles. Total 48 cycles are used in this PPU design. Table 7 shows a comparison with previous work [10]. From this table, we can find the proposed design only needs 50% buffer size, 36% bus bandwidth, and 54% gate counts but achieves 4.75X throughput improvement. The improvement comes from proposed MBPPU design. Due to a smaller number of pixels are needed, we can significantly reduce bandwidth requirement, local buffer size, and design cost.

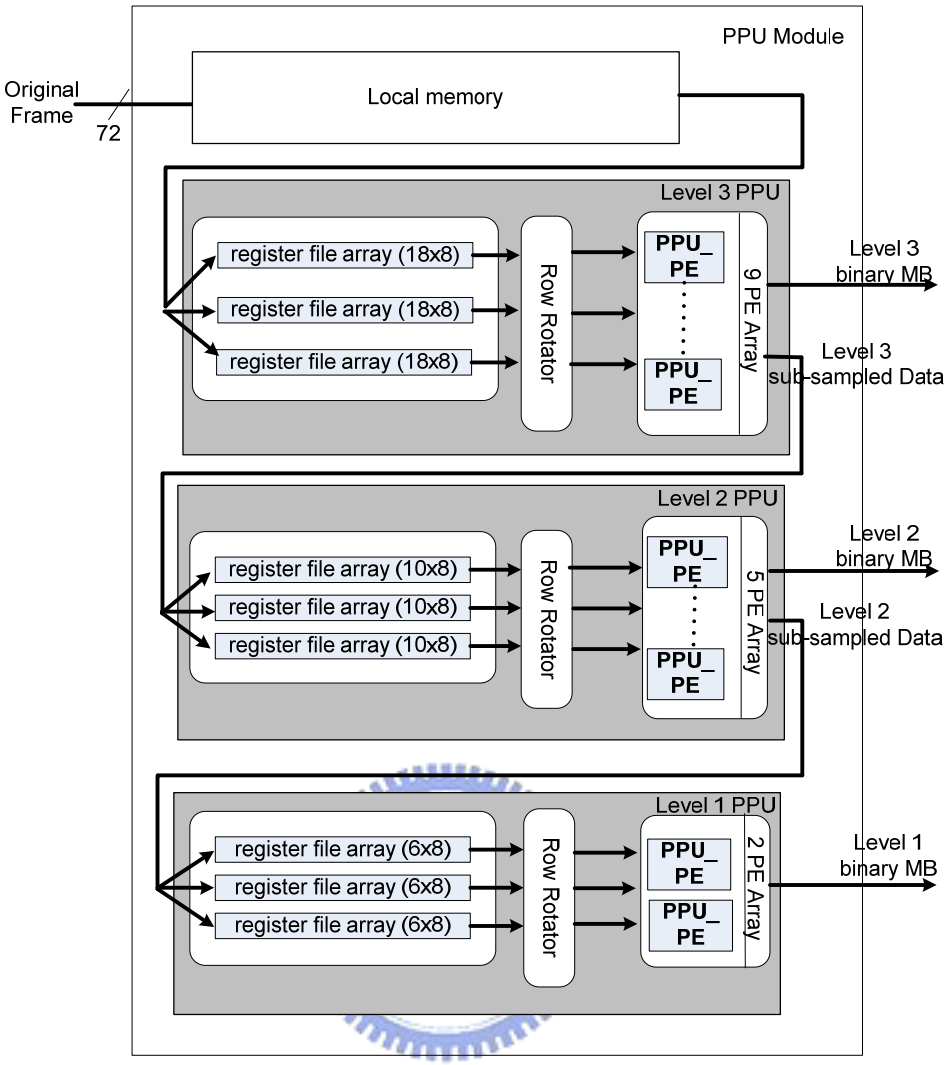


Figure 19. Proposed PPU architecture.

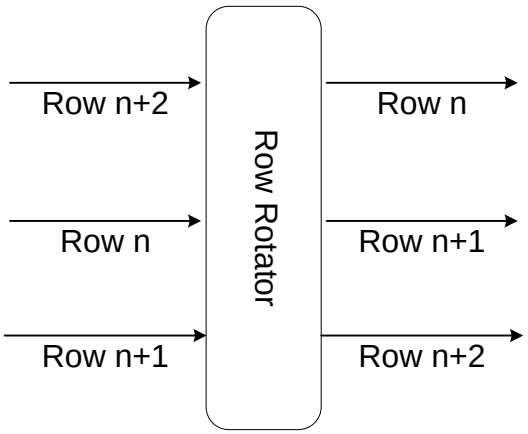


Figure 20. Proposed row rotator to rearrange the order for input rows.

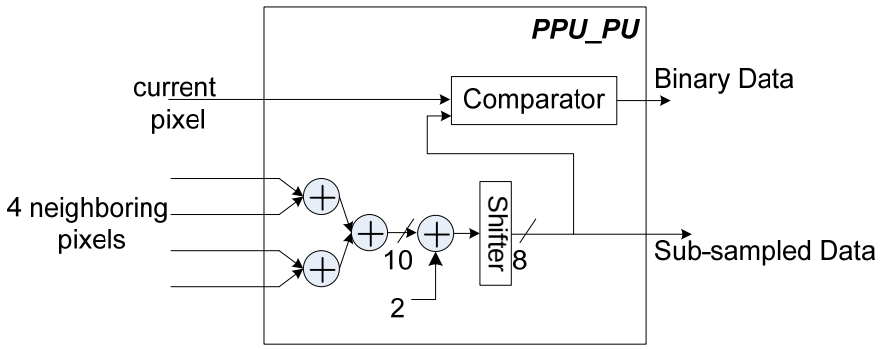


Figure 21. PPU_PE architecture used to generate binary and sub-sampled data.

Table 6. Comparison table of PPU designs between ABME [10] and proposed design.

	PPU in ABME	***Our design***
MB Latency (cycles)	228	48
Local Buffer Size (bits)	1696	816
Bandwidth Requirement (bits/MB)	7200	2592
Gate Count (0.18um process)	26.2 k	14.2 k

4.2.3 Motion Estimation Engine

The ME engine comprises four main components including shared SOD PE, LV1 search unit, LV2 search unit, and LV3 search unit.

A. Shared Sum of Difference Processing element (SOD PE)

In the BBME, there are three search block sizes for each of the three pyramid layers. It uses 4×4, 8×8 and 16×16 block sizes from level one to three respectively. To maximize hardware utilization and minimize hardware cost, a shared PE is designed to compute SOD for different layers with one module. As shown in Figure 22, the SOD is performed in the PE that contains 256 bits XOR operations followed by a 256-bit adder tree. The 256 bits XOR operations are partitioned into 16 blocks of 16-bit XOR operations to provide 16 4×4 SOD

results $S_i^{4x4}\{i=0\sim15\}$. Then, the sixteen 4x4 SODs can be accumulated as four 8x8 SODs $S_i^{8x8}\{i=0\sim3\}$ or one 16x16 SOD S_0^{16x16}.

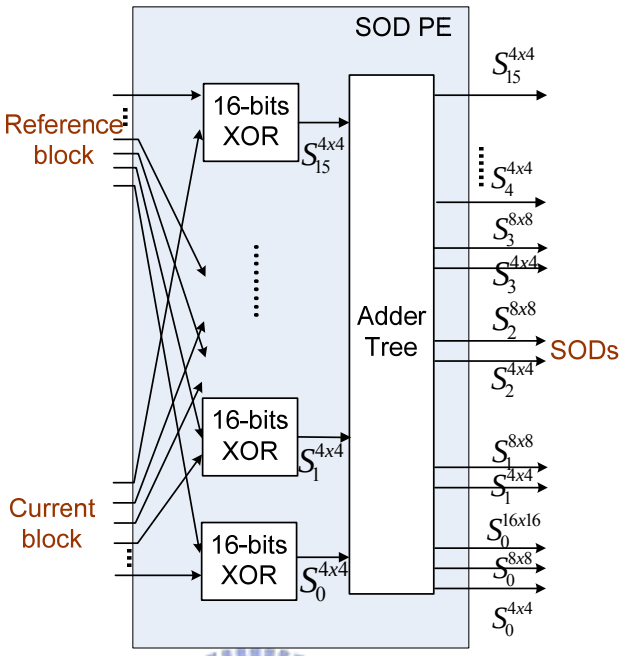


Figure 22. Shared SOD PE design.

B. Level 1 (LV1) Search Unit

The LV1 search unit as shown in Figure 23 is designed based on the proposed parallel architecture to complete the LV1 search with low power consumption. It contains a LV1 controller, a LV1 MV determiner I, and three banks of local memories. To complete the LV1 search, the controller controls the data access from current search data buffer (LM_CUR_LV1) and two reference search data buffers (LM_SW1_LV1 and LM_SW2_LV1) to the shared SOD PEs for SAD calculation. The shared SOD PE is able to compute 16 parallel LV1 search SOD, and returns the results to MV_LV1 determiner for final LV1 MV decision. For B-frame search, the two shared SOD PEs are used to process forward and backward search block in parallel. For P-frame search, the forward search block data is mirrored from LM_SW1_LV1 to LM_SW2_LV1. The controller controls the data flow to be able to let the one SOD PEs to process the one half search locations and the other one to process the other half. Such a design methodology can make the whole design 100% busy.

Figure 24 shows the data processing flow. For search range ± 3, there are 7×7 search locations. To meet the design specification of the shared SOD PE, we calculate fourteen 4×4

SODs in one cycle. In the first cycle, r0 and r1 in SW1 and r5 and r6 in SW2 are checked in parallel. Using this method, we can finish the ±3 search in 4 cycles for B-frame search and 2 cycles for P-frame search. Two more extra cycles are needed due to the control overhead. For search range ± 7, we have to take 10 and 18 cycles for P-search and B-search respectively.

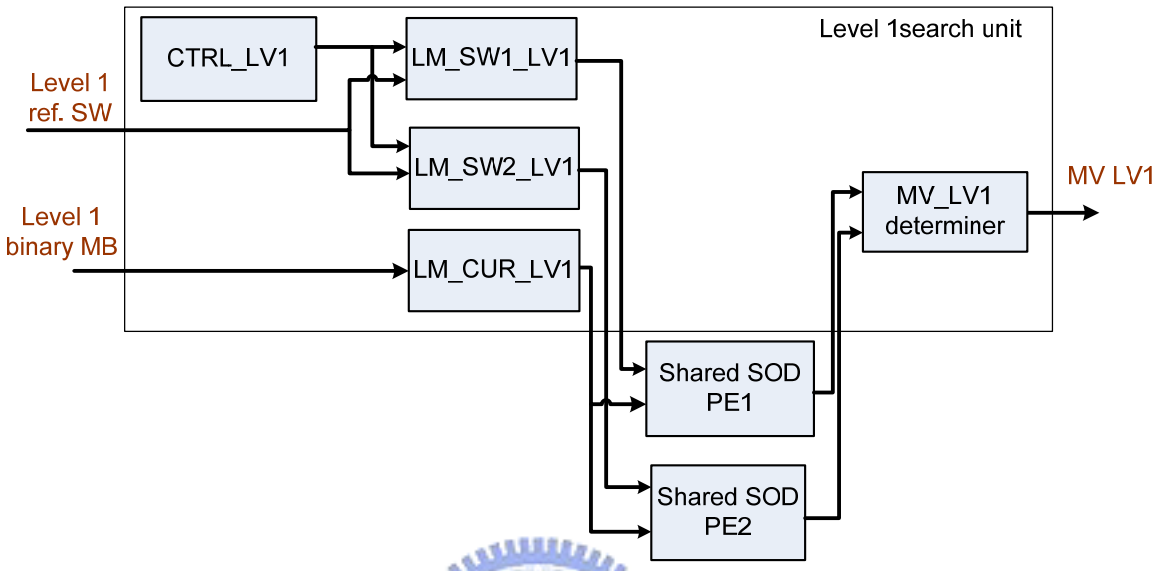


Figure 23. Hardware architecture of LV1 search.

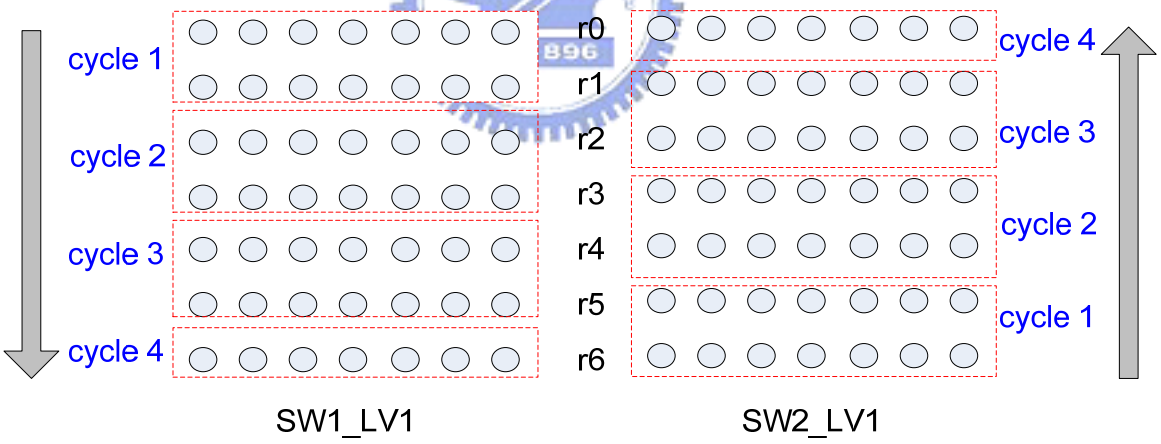


Figure 24. LV1 search order with ± 3 search range. In this design, each SOD PE checks 14 search locations in parallel.

C. Level 2 (LV2) Search Unit

To fully use the shared SOD PEs and remove the latencies in this fine tuning stages for low power, a hardware efficient architecture is used in LV2 design. The original algorithm

shows two hardware design difficulties. The first design difficulty is that two data accesses are needed to allow voting first and then a cross pattern search. Our solution is to check all the candidates sequentially to avoid the branch operations. The second design difficulty is the use of multiplexer (MUX). A 2-D MUX shown in Figure 26(a) is used to access the reference block from SW buffer. In search range of ±16, the SW size is 24×24 in LV2, so a 24×24 to 8×8 MUX is needed for LV2 search. When the search range becomes wider, the hardware cost for 2-D MUX increases significantly. Our solution is to partition the search binary bitplane into several regions to be stored in different register files. So, access of one reference block only needs partial regions. This can fix the MUX size independent of the search range. Figure 26 (b) shows an example to partition a LV2 search range into 9 regions, and only a 16×16 to 10×10 MUX is needed. However, dividing the search binary bitplane into several register files suffers some overhead for the need of extra address decoder. Table 7 shows the comparison of MUX area in work [10] and the proposed LV2 search. It shows the proposed design can achieve at most 34% saving in hardware area. Figure 25 shows the hardware architecture of LV2 search. In B-frame search, the LV2 binary MB from PPU engine is stored in the on-chip memory of LV2 search unit, LM_CUR_LV2, and the forward and backward SWs are stored in the on-chip memories, LM_SW1_LV2 and LM_SW2_LV2, separately. For each MB search, the LV2 controller (CTRL_LV2) controls on-chip memories data access and passed to the shared SOD PEs for pattern matching. At each cycle, one of the five candidates and its neighboring search locations are checked. The resulting SADs are sent back to MV_LV2 determiner and compared with the minimum SOD stored in MV_LV2 determiner. If a smaller SOD is found, the minimal SOD in determiner is updated. After all the candidate search locations are checked, the MV with the minimum SOD for two directions is outputted at the same time. As for P-frame search, the five candidates are separated into two groups. And candidates in these two groups are checked in parallel to reduce the running cycle. In each cycle, we have to check the ±1 cross pattern for one of the five candidates. However, the shared SOD PE can only output four calculate 8×8 SODs each cycle. According to the experiment result, discarding the left position makes least impact on PSNR performance. Thus the left search location in the ±1 cross pattern is discarded in our hardware implement.

To analyze the design timing for LV2 search, the proposed architecture takes one cycle to check each candidate. With the use of two shared SOD PEs, the design takes 5 cycles for the 5 candidates for forward and backward search blocks, and 3 cycles for forward only P-frame

search. Including control overhead, the total cycles are 6 and 8 for P and B-frame, respectively.

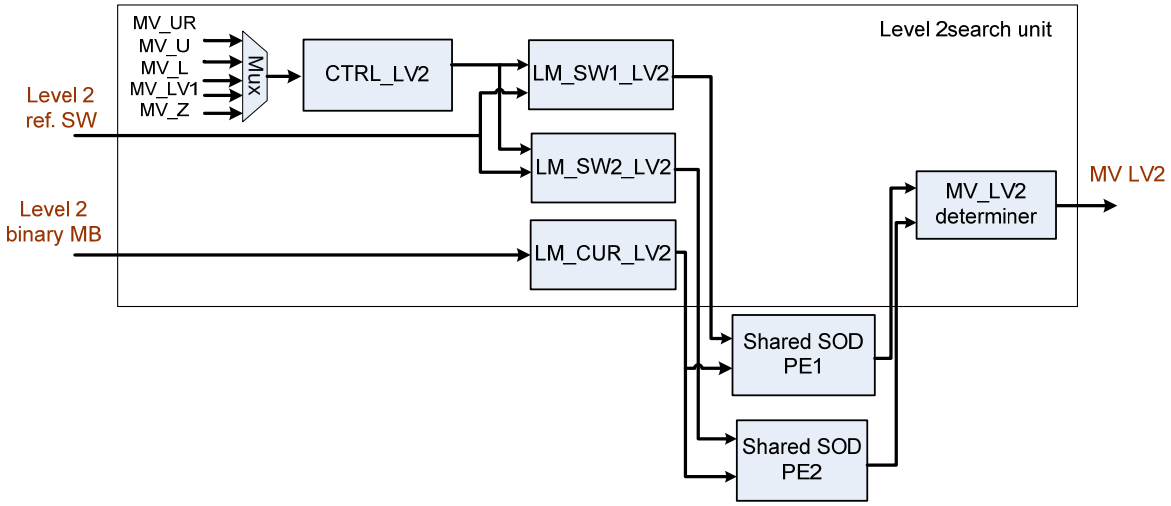


Figure 25. Hardware architecture of LV2 search.

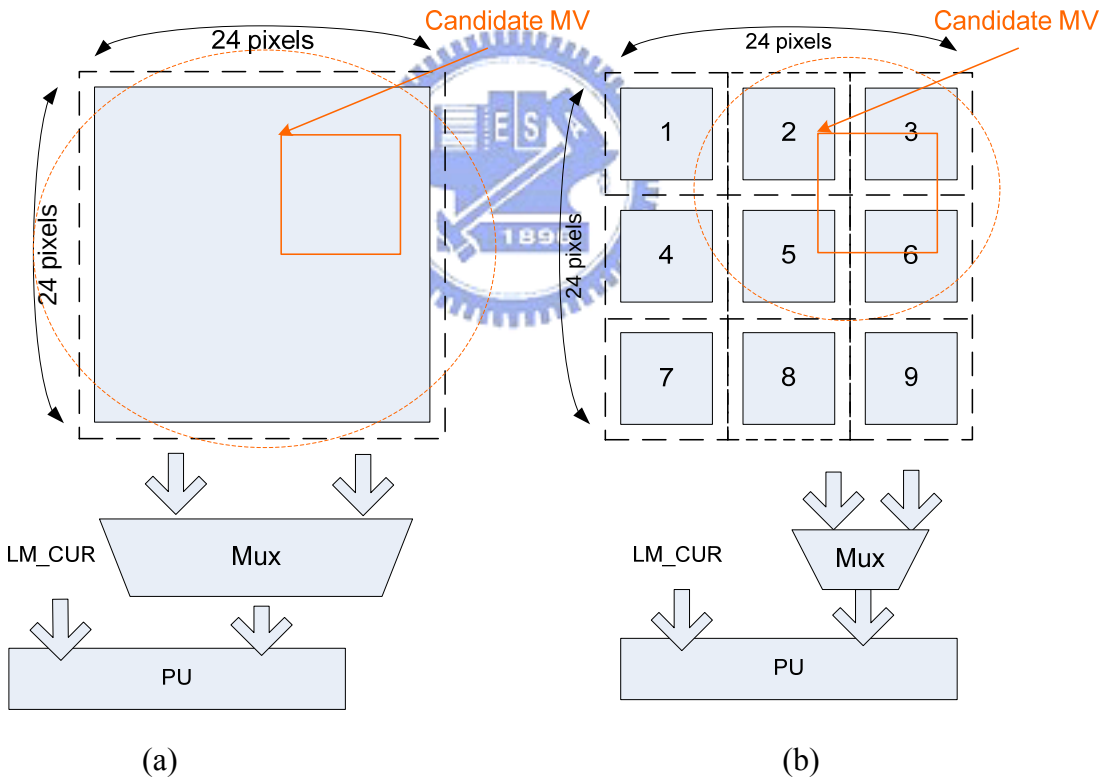


Figure 26. Working flow of LV2 design. (a) Methodology in work [10] which uses one 24×24 to 8×8 MUX (b) Proposed LV2 design which reduces MUX size requirement to be 16×16 to 8×8.

Table 7. Comparison of SW buffer area in work [10] and proposed LV2 search.

Search range	±16	±32	±64
MUX	576 to 100	1600 to 100	5184 to 100
Gate count of MUX in ABME	4.8k	14.2k	43.2k
Gate count of MUX in BBME	4.8k	4.8k	4.8k
Ratio of MUX	100%	33.8%	11%
Gate count of SW buffer in ABME	17.5k	53.1k	166.3k
Gate count of SW buffer BBME	16.7k	37.7k	109.7k
Ratio SW buffer	95.4%	71.0%	66.0%

D. Level 3 (LV3) Search Unit

The LV3 design replaces the MUX with shifter for design cost saving and power reduction. Level 3 performs ±2 full search so the needed SW size is 20×20. If the architecture with 2D MUX is used, this means we need a 20×20 to 16×16 MUX which is a great hardware cost. Our solution is to adopt the shifter register as SW buffer. Figure 27 illustrates the working flow of the shifter register. If the pixels in column 2 and 3 are needed, the entire registers are one column shifted circularly, so that the required data can be moved to the correct position, column 1 and column 2 for next data processing. Figure 28 shows the architecture of LV3 search. In B-frame search, the LV3 binary MB from PPU engine is stored in the on-chip memory of LV3 search unit, LM_CUR_LV3, and the forward and backward SWs are stored in the on-chip memories, LM_SW1_LV3 and LM_SW2_LV3, separately. For each MB search, the LV3 controller (CTRL_LV3) controls on-chip memories data access and passes to the shared SOD PEs for pattern matching. At cycle 0, SOD in search position (-2, -2) is calculated. At cycle 1~4, 20×20 search area pixels are circularly shifted in the left direction column by column as shown in Figure 29 (a). So, the SODs in search positions (-1, -2), (0, -2),

(1, -2), (2, -2) are obtained sequentially. At cycle 5, 20×20 search area pixels are circularly shifted upward with one pixel as shown in Figure 29 (b), and SOD of search position (2, -1) is calculated. In the similar way, all search locations are covered after cycle 25. The search order in SW2_LV3 is reversed as shown in Figure 30. The SODs are sent back to MV_LV3 determiner for comparison with the minimum SOD stored in MV_LV2 determiner. If a smaller SOD is found, the minimal SOD in determiner is updated. After all the candidate search locations are checked, the MVs with the minimum SOD for two directions are outputted at the same time. For P-frame search, the search locations are separated into two sub-groups and are checked in parallel.

To analyze the design timing for LV3 search, the proposed architecture takes one cycle to check each search location. With the use of two shared SOD PEs, the design takes 25 cycles for the 25 search locations for forward and backward search blocks, and 3 cycles to for forward only P-frame search. Including control overhead, the total cycles are 6 and 8 for P and B-frame, respectively. Including the memory access latency for fetching SW_LV3 data and control overhead, the total cycles are 18 and 34 for P and B-frame, respectively.

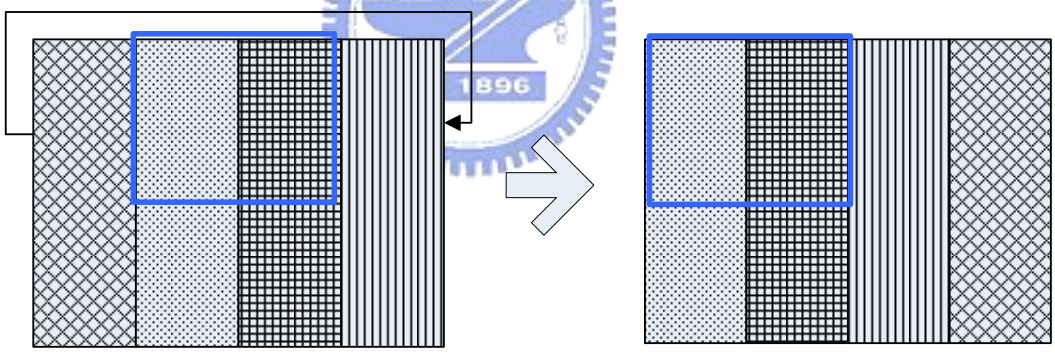


Figure 27. Fixed fetching position by adopting shifter register as SW buffer.

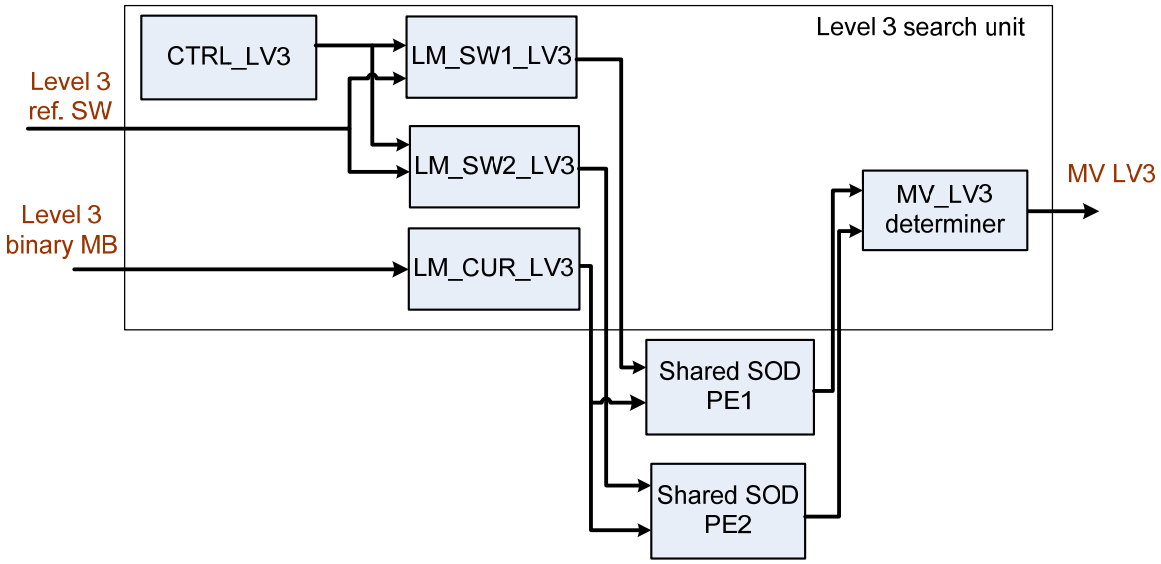
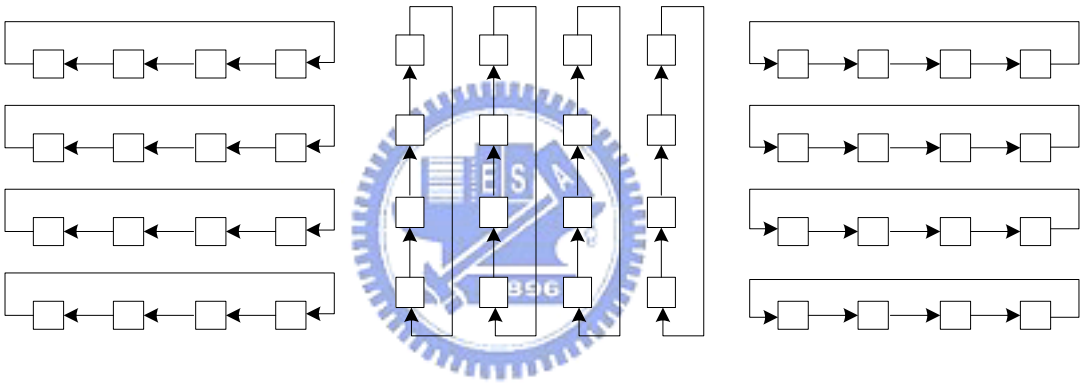
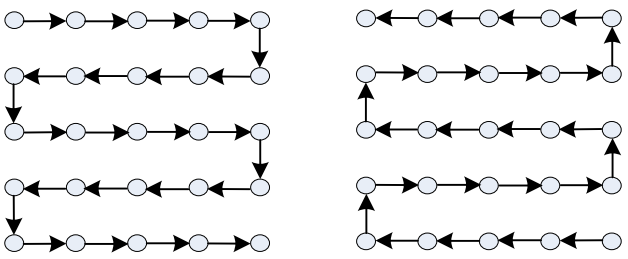


Figure 28. Hardware architecture of LV3 search.



(a) (b) (c)

Figure 29. (a) Whole SW1_LV3 is shifted in left direction at cycle 1~4, 11~14, and 21~24 (b) Whole SW1_LV3 is shifted in left direction at cycle 5, 10, 15 and 20 (c) Whole SW1_LV3 is shifted in left direction at cycle 6~9, and 16~19.



(a) (b)

Figure 30. (a) LV3 ME search order for SW1_LV3 begins from the top-left point and in a snake order (b) LV3 ME search order for SW2_LV3 is the reverse order.

4.3 Mode Decision and Sub-pel ME (MD-SME) Module

In the MD-SME design, we integrate the MD module into SME to avoid two loops processing of MD and SME to save power. Our design shares the PEs for intra cost calculation and SAD operations in SME search so that MD and SME are combined into single loop. Furthermore, considering the overlapped pixels between adjacent reference blocks in SME, we adopt architecture with parallel processing of multiple adjacent search locations to avoid redundant on-chip memory accesses.

Figure 31 shows the architecture of MD-SME. It contains an intra cost engine, an inter cost engine, three shared SAD PEs, a mode determiner, and a MV determiner. To determine the coding mode, MD-SME module begins to calculate the intra cost according to Equation (7) and the inter cost for block 16×16 or 4 block 8×8. For the computation of the intra cost, the 8-bit current MB is stored in the current MB buffer of intra cost engine and sent to average PE (Avg_PE) to calculate the mean value. Then the mean value and current MB are sent to the shared SAD PE for the computation of intra cost. As shown in Figure 32, each SAD_PE consists of 16 absolute differences PEs (AD_PEs) and one accumulator. It can process one row of 16×16 MB or two rows of 8×8 block each cycle. For the computation of the inter cost, 8-bits forward/backward SW data is stored in separate on-chip memories of inter cost engine, LM_SW1_SAD and LM_SW2_SAD, respectively. According to the MVs from IME, the address generator (AG) generates correct memory access addresses for the on-chip memories. The on-chip memory outputs data to interpolation PE (Interp_PE) to interpolate sub-pel data on the fly. Then the generated sub-pel data and current MB are sent to the shared SAD PEs for SAD calculation. The intra and inter costs outputted by SAD PEs are sent to mode determiner for comparison. If the coding mode is determined as inter, MD-SME module continues to perform SME, The data flow of SAD calculation for SME is similar with that of inter cost. Due to the overlapped pixels between adjacent reference blocks in SME as illustrated in Figure 33, 3 SAD PEs are designed to process 3 adjacent search locations in parallel for on-chip memory access reduction and power saving. The resulting SADs are sent to the MV determiner for final MV decision.

To analyze the design timing for MD-SME module, Table 8 and Table 9 illustrate the data flow of SAD_PE for inter and intra cost calculation, respectively. In these two tables, $C_{i,j}$ and $R_{i,j}$ represent the pixels of current and reference MB in location (i, j) and $\overline{C}$ is

the mean value of current MB. Since there are three SAD PEs, the intra and inter cost can be calculated in parallel. So 16 cycles are needed for inter and intra cost calculation. The total running cycle of MD is 18 including memory access and control latency. In SME, we process 3 search locations in parallel and the data flow of SAD_PE array is shown in Table 10, where $Cr_{i,j}$ and $Rr_{i,j}$ represent j_{th} row in i_{th} search block. It takes 16 cycles for every three search locations and 48 cycles for one 16×16 block search. If the current frame is P-frame, we need to perform ME for four 8×8 sub-blocks, which also takes 48 cycles. If the current frame is B-frame, we only have to perform ME for block 16×16 in forward and backward direction sequentially because MPEG-4 does not support sub-pel MV of 8×8 sub-blocks in B-frame. So it takes 96 cycles to compute SADs for both P and B-frame search. However, there is some control latency for an iteration of ME. The P-frame search contains 5 iterations of ME including one 16×16 and four 8×8 searches, so 113 cycles are needed. For B-frame search it only contains 2 iterations of ME, which are 16×16 forward and backward searches. Thus only 101 cycles are needed for B-frame search.

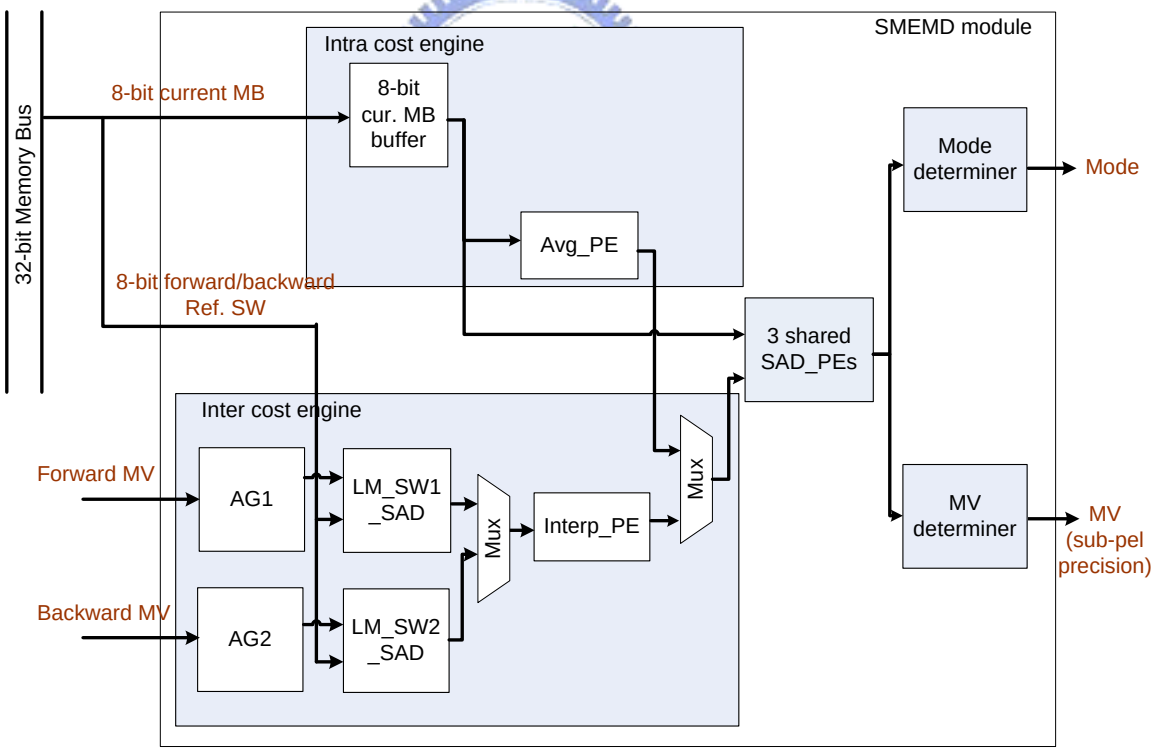


Figure 31. Architecture of MD-SME.

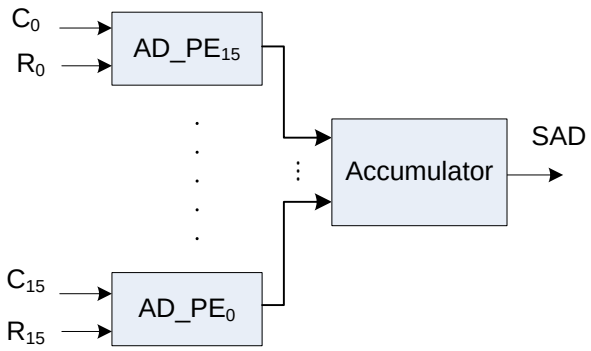


Figure 32. Architecture of SAD_PE in MD-SME.

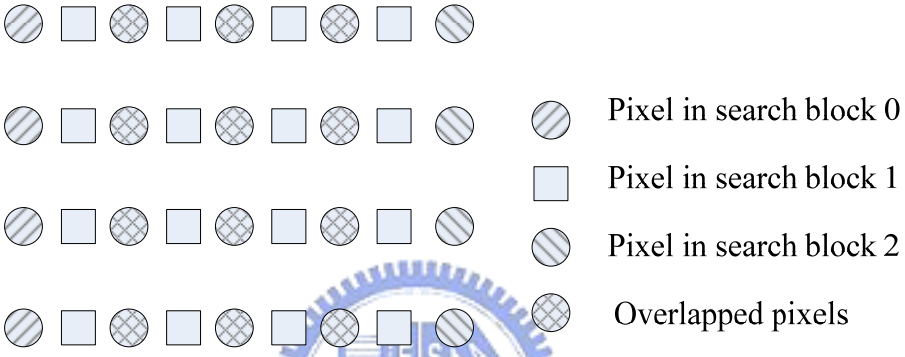


Figure 33. Overlapped pixels between search blocks.

Table 8. Data flow for SAD calculation.

Cycle	*AD_PE0*	*AD_PE1*	*AD_PE2*	*AD_PE3*
0	$C_{0,0}$,$R_{0,0}$	$C_{1,0}$,$R_{1,0}$	$C_{2,0}$,$R_{2,0}$	$C_{3,0}$,$R_{3,0}$
1	$C_{0,1}$,$R_{0,1}$	$C_{1,1}$,$R_{1,1}$	$C_{2,1}$,$R_{2,1}$	$C_{3,1}$,$R_{3,1}$
2	$C_{0,2}$,$R_{0,2}$	$C_{1,2}$,$R_{1,2}$	$C_{2,2}$,$R_{2,2}$	$C_{3,2}$,$R_{3,2}$
3	$C_{0,3}$,$R_{0,3}$	$C_{1,3}$,$R_{1,3}$	$C_{2,3}$,$R_{2,3}$	$C_{3,3}$,$R_{3,3}$
4	$C_{0,4}$,$R_{0,4}$	$C_{1,4}$,$R_{1,4}$	$C_{2,4}$,$R_{2,4}$	$C_{3,4}$,$R_{3,4}$
5	$C_{0,5}$,$R_{0,5}$	$C_{1,5}$,$R_{1,5}$	$C_{2,5}$,$R_{2,5}$	$C_{3,5}$,$R_{3,5}$
6	$C_{0,6}$,$R_{0,6}$	$C_{1,6}$,$R_{1,6}$	$C_{2,6}$,$R_{2,6}$	$C_{3,6}$,$R_{3,6}$
7	$C_{0,7}$,$R_{0,7}$	$C_{1,7}$,$R_{1,7}$	$C_{2,7}$,$R_{2,7}$	$C_{3,7}$,$R_{3,7}$
8	$C_{0,8}$,$R_{0,8}$	$C_{1,8}$,$R_{1,8}$	$C_{2,8}$,$R_{2,8}$	$C_{3,8}$,$R_{3,8}$
9	$C_{0,9}$,$R_{0,9}$	$C_{1,9}$,$R_{1,9}$	$C_{2,9}$,$R_{2,9}$	$C_{3,9}$,$R_{3,9}$
10	$C_{0,10}$,$R_{0,10}$	$C_{1,10}$,$R_{1,10}$	$C_{2,10}$,$R_{2,10}$	$C_{3,10}$,$R_{3,10}$
11	$C_{0,11}$,$R_{0,11}$	$C_{1,11}$,$R_{1,11}$	$C_{2,11}$,$R_{2,11}$	$C_{3,11}$,$R_{3,11}$
12	$C_{0,12}$,$R_{0,12}$	$C_{1,12}$,$R_{1,12}$	$C_{2,12}$,$R_{2,12}$	$C_{3,12}$,$R_{3,12}$
13	$C_{0,13}$,$R_{0,13}$	$C_{1,13}$,$R_{1,13}$	$C_{2,13}$,$R_{2,13}$	$C_{3,13}$,$R_{3,13}$
14	$C_{0,14}$,$R_{0,14}$	$C_{1,14}$,$R_{1,14}$	$C_{2,14}$,$R_{2,14}$	$C_{3,14}$,$R_{3,14}$
15	$C_{0,15}$,$R_{0,15}$	$C_{1,15}$,$R_{1,15}$	$C_{2,15}$,$R_{2,15}$	$C_{3,15}$,$R_{3,15}$

Table 9. Data flow for intra cost calculation.

Cycle	*AD_PE0*	*AD_PE1*	*AD_PE2*	*AD_PE3*
0	$C_{0,0}$, $\overline{C}$	$C_{1,0}$, $\overline{C}$	$C_{2,0}$, $\overline{C}$	$C_{3,0}$, $\overline{C}$
1	$C_{0,1}$, $\overline{C}$	$C_{1,1}$, $\overline{C}$	$C_{2,1}$, $\overline{C}$	$C_{3,1}$, $\overline{C}$
2	$C_{0,2}$, $\overline{C}$	$C_{1,2}$, $\overline{C}$	$C_{2,2}$, $\overline{C}$	$C_{3,2}$, $\overline{C}$
3	$C_{0,3}$, $\overline{C}$	$C_{1,3}$, $\overline{C}$	$C_{2,3}$, $\overline{C}$	$C_{3,3}$, $\overline{C}$
4	$C_{0,4}$, $\overline{C}$	$C_{1,4}$, $\overline{C}$	$C_{2,4}$, $\overline{C}$	$C_{3,4}$, $\overline{C}$
5	$C_{0,5}$, $\overline{C}$	$C_{1,5}$, $\overline{C}$	$C_{2,5}$, $\overline{C}$	$C_{3,5}$, $\overline{C}$
6	$C_{0,6}$, $\overline{C}$	$C_{1,6}$, $\overline{C}$	$C_{2,6}$, $\overline{C}$	$C_{3,6}$, $\overline{C}$
7	$C_{0,7}$, $\overline{C}$	$C_{1,7}$, $\overline{C}$	$C_{2,7}$, $\overline{C}$	$C_{3,7}$, $\overline{C}$
8	$C_{0,8}$, $\overline{C}$	$C_{1,8}$, $\overline{C}$	$C_{2,8}$, $\overline{C}$	$C_{3,8}$, $\overline{C}$
9	$C_{0,9}$, $\overline{C}$	$C_{1,9}$, $\overline{C}$	$C_{2,9}$, $\overline{C}$	$C_{3,9}$, $\overline{C}$
10	$C_{0,10}$, $\overline{C}$	$C_{1,10}$, $\overline{C}$	$C_{2,10}$, $\overline{C}$	$C_{3,10}$, $\overline{C}$
11	$C_{0,11}$, $\overline{C}$	$C_{1,11}$, $\overline{C}$	$C_{2,11}$, $\overline{C}$	$C_{3,11}$, $\overline{C}$
12	$C_{0,12}$, $\overline{C}$	$C_{1,12}$, $\overline{C}$	$C_{2,12}$, $\overline{C}$	$C_{3,12}$, $\overline{C}$
13	$C_{0,13}$, $\overline{C}$	$C_{1,13}$, $\overline{C}$	$C_{2,13}$, $\overline{C}$	$C_{3,13}$, $\overline{C}$
14	$C_{0,14}$, $\overline{C}$	$C_{1,14}$, $\overline{C}$	$C_{2,14}$, $\overline{C}$	$C_{3,14}$, $\overline{C}$
15	$C_{0,15}$, $\overline{C}$	$C_{1,15}$, $\overline{C}$	$C_{2,15}$, $\overline{C}$	$C_{3,15}$, $\overline{C}$

Table 10. Data flow for SME.

Cycle	*SAD_PE0*	*SAD_PE1*	*SAD_PE2*
0	$Cr_{0,0}$,$Rr_{0,0}$	$Cr_{1,0}$,$Rr_{1,0}$	$Cr_{2,0}$,$Rr_{2,0}$
1	$Cr_{0,1}$,$Rr_{0,1}$	$Cr_{1,1}$,$Rr_{1,1}$	$Cr_{2,1}$,$Rr_{2,1}$
2	$Cr_{0,2}$,$Rr_{0,2}$	$Cr_{1,2}$,$Rr_{1,2}$	$Cr_{2,2}$,$Rr_{2,2}$
3	$Cr_{0,3}$,$Rr_{0,3}$	$Cr_{1,3}$,$Rr_{1,3}$	$Cr_{2,3}$,$Rr_{2,3}$
4	$Cr_{0,4}$,$Rr_{0,4}$	$Cr_{1,4}$,$Rr_{1,4}$	$Cr_{2,4}$,$Rr_{2,4}$
5	$Cr_{0,5}$,$Rr_{0,5}$	$Cr_{1,5}$,$Rr_{1,5}$	$Cr_{2,5}$,$Rr_{2,5}$
6	$Cr_{0,6}$,$Rr_{0,6}$	$Cr_{1,6}$,$Rr_{1,6}$	$Cr_{2,6}$,$Rr_{2,6}$
7	$Cr_{0,7}$,$Rr_{0,7}$	$Cr_{1,7}$,$Rr_{1,7}$	$Cr_{2,7}$,$Rr_{2,7}$
8	$Cr_{0,8}$,$Rr_{0,8}$	$Cr_{1,8}$,$Rr_{1,8}$	$Cr_{2,8}$,$Rr_{2,8}$
9	$Cr_{0,9}$,$Rr_{0,9}$	$Cr_{1,9}$,$Rr_{1,9}$	$Cr_{2,9}$,$Rr_{2,9}$
10	$Cr_{0,10}$,$Rr_{0,10}$	$Cr_{1,10}$,$Rr_{1,10}$	$Cr_{2,10}$,$Rr_{2,10}$
11	$Cr_{0,11}$,$Rr_{0,11}$	$Cr_{1,11}$,$Rr_{1,11}$	$Cr_{2,11}$,$Rr_{2,11}$
12	$Cr_{0,12}$,$Rr_{0,12}$	$Cr_{1,12}$,$Rr_{1,12}$	$Cr_{2,12}$,$Rr_{2,12}$
13	$Cr_{0,13}$,$Rr_{0,13}$	$Cr_{1,13}$,$Rr_{1,13}$	$Cr_{2,13}$,$Rr_{2,13}$
14	$Cr_{0,14}$,$Rr_{0,14}$	$Cr_{1,14}$,$Rr_{1,14}$	$Cr_{2,14}$,$Rr_{2,14}$
15	$Cr_{0,15}$,$Rr_{0,15}$	$Cr_{1,15}$,$Rr_{1,15}$	$Cr_{2,15}$,$Rr_{2,15}$

4.4 Timing Analysis and System Pipelining

In data flow of ME flow, IME, transmission of SW data, and MD-SME are performed sequentially. IME is performed firstly to output 16×16 and 8×8 MVs. After IME, 22×22×8 SW data is needed for inter cost and SAD calculation in ME-SME. After transmission is complete, MD-SME module begins to calculate the intra cost according to Equation (7) and the inter cost from the SAD for block 16×16 or 4 block 8×8. The inter cost is from the SAD for block 16×16 or 4 block 8×8. If the mode is decides as inter, 16×16 MV in integer precision is used as the initial center of SME with ±1 search range. In the preceding sections, we detail the hardware design of IME and MD-SME modules. Besides we analyze the running cycle for each sub-module.

To analyze the design timing, Table 11 summarizes the required cycle count for proposed IME under different frame types and different search ranges. The longest latency is 107 cycles for B-frame search with ±32 search range. The transmission of SW data takes 121 cycles for SW of 22×22×8 bits under 32-bit bus bandwidth. The MD_SME operations take at most 131 cycles as shown in Table 12.

The system pipeline architecture is used to enable the parallel processing of IME, sub-pel data transmission, and MD-SME to avoid hardware idling and optimize the overall throughput. The complete ME can be partitioned into three stages including IME, transmission of reference data for MD-SME, and MD-SME. We design a MB level pipelining schedule of ME with three stages as shown in Figure 34. Each stage of ME is processed as follows.

1. IME stage process the n_{th} MB and send the resulting data MV to transmission stage.
2. Transmission stage fetches SW of 22×22×8 bits for n_{th} MB.
3. MD-SME stage process the n_{th} MB and generate the final MV.

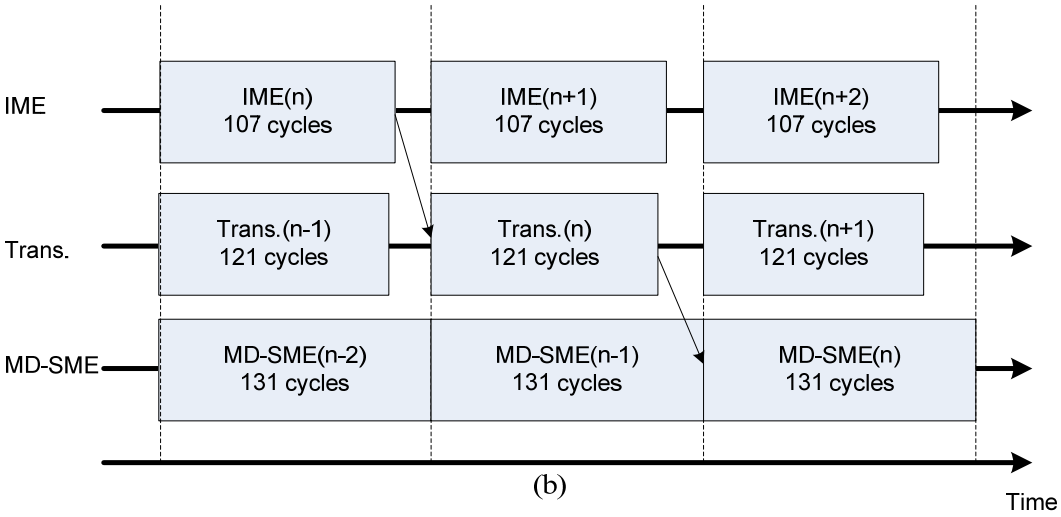
When performing IME for the n_{th} MB, the transmission of SW data for $(n-1)_{th}$ MB and MD-SME for $(n-2)_{th}$ MB are processed at the same time. Hence, all functional modules can be processed in parallel.

Table 11. Cycle count of IME.

	Cycle count for P-frame with SR=±16	***Cycle count for B-frame with SR=±16***	***Cycle count for P-frame with SR=±32***	***Cycle count for B-frame with SR=±32***
Total	76	96	82	107

Table 12. Cycle count of MD-SME.

	Cycle count for P-frame	***Cycle count for B-frame***
MD	18	18
SME	113	101
Total	131	119



IME(n) : IME for n_{th} MB

Trans.(n) : Transmission of SW for n_{th} MB

MD-SME(n) : MD and SME for n_{th} MB

Figure 34. Scheduling approach for MB level pipeline.

Chapter 5

Experimental Results and Analysis

In this chapter, we provide the experimental results and comparisons with prior algorithms and hardware architectures. Detailed experiment environment, test conditions and results are described.

5.1 Algorithm Level Comparison

5.1.1 Test Condition

The proposed bi-directional ME algorithm is integrated in Momusys MPEG-4 reference software. To show the good performance of the proposed algorithm, we do the R-D comparison with full search (FS), diamond search (DS), and ABME [10]. Six common MPEG test sequences are tested. The test bitrate are 256, 512 and 1024kbps. The detailed experiment environment is listed as followed.

- Test Platform: Intel Pentium 2.8Ghz, 512MB Memory
- Test Operation System: Microsoft Windows XP Professional.
- Test Environment: Microsoft Visual C++ 6.0
- Test Software: Momusys MPEG-4 reference video encoder N4025.
- Test Configurations:
  - H.263 quantization mode is selected.
  - Number of coded frame is 300.
  - Initial QP for intra block is 8.
  - Initial QP for inter block in P-VOPs is 8.
  - Initial QP for inter block in B-VOPs is 10.
  - Rounding control is enabled.
  - Initial value for rounding control is 0.
  - Error resilience mode is disabled.
  - Complexity estimation data transmission is disabled.

- Search range per coded frame is 16.
- Quarter pel motion compensation is on.
- No sprite coding.

5.1.2 RD Performance Evaluation

Since the MD-SME design follows the original algorithm in Momusys MPEG-4 reference software, only IME implemented with proposed BBME makes effect on RD performance. To evaluate the PSNR performance of BBME, we employ five common test video sequences, including Foreman, Akiyo, Flower, Mobile and Tempete. For intensive simulation, three target bitrates such as 256, 512, and 1024kbps are tried out. And the distance between two adjacent P-frames is set to 1, 2, and 3, respectively. Table 13 ~ Table 21 show the simulation results. Take Table 13 for example, the PSNR losses of Foreman sequence are 0.95, 0.53 and 0.61 dB for DS, ABME, and BBME, respectively. As for other sequences, the PSNR losses of BBME are 0.02, 0.17, 0.0 and 0.06 dB for Akiyo, Flower, Mobile, and Tempete, respectively. Compared with FS, the PSNR drop of BBME is about 0.6 dB in the worst case. And it has the almost the same PSNR performance in the best case. In Figure 35 to Figure 37, we show the RD curves of Foreman sequence under different ME schemes. Compared with FS, the PSNR drop is about 0.6 dB for BBME while more than 1 dB for DS. In Figure 38 to Figure 40, the RD curves of Mobile sequence are shown. The PSNR drop of BBME is less than 0.2 dB. It can be seen the curve of different ME scheme is quite close.

Table 13. PSNR comparison for BBME, FS, DS, and ABME at target bitrate of 256kbps. (N=300, M=1).

Sequence	*Method*	*Y_PSNR(dB)*	*ΔPSNR(dB)*	*Bitrate(kbps)*
Foreman	FS	30.84		255.91
	DS	29.89	-0.95	253.87
	ABME	30.31	-0.53	255.97
	BBME	30.23	-0.61	255.98
Akiyo	FS	41.61		255.87
	DS	41.6	-0.01	255.99
	ABME	41.50	-0.11	256.04
	BBME	41.59	-0.02	256.01
Flower	FS	23.86		266.77
	DS	23.84	-0.02	275.49
	ABME	23.70	-0.16	290.80
	BBME	23.69	-0.17	309.96
Mobile	FS	23.38		255.35
	DS	23.44	0.06	263.02
	ABME	23.37	-0.01	257.09
	BBME	23.38	0.00	273.15
Tempete	FS	26.04		256.83
	DS	26.07	0.03	256.92
	ABME	25.95	-0.09	256.93
	BBME	25.98	-0.06	259.24

Table 14. PSNR comparison for BBME, FS, DS, and ABME at target bitrate of 512kbps. (N=300, M=1).

Sequence	*Method*	*Y_PSNR(dB)*	*ΔPSNR(dB)*	*Bitrate(kbps)*
Foreman	FS	34.18		511.98
	DS	33.26	-0.92	511.93
	ABME	33.82	-0.36	511.95
	BBME	33.79	-0.39	511.97
Akiyo	FS	43.33		512.02
	DS	43.35	0.02	511.99
	ABME	43.31	-0.02	512.03
	BBME	43.33	0.00	512.01
Flower	FS	26.11		512.07
	DS	25.99	-0.12	512.02
	ABME	25.75	-0.36	512.05
	BBME	25.66	-0.45	511.95
Mobile	FS	26.18		511.84
	DS	26.26	0.08	512.01
	ABME	26.10	-0.08	511.92
	BBME	26.07	-0.11	512.02
Tempete	FS	28.78		512.01
	DS	28.79	0.01	512.14
	ABME	28.78	0.00	512.02
	BBME	28.79	0.01	512.08

Table 15. PSNR comparison for BBME, FS, DS, and ABME at target bitrate of 1024kbps. (N=300, M=1).

Sequence	*Method*	*Y_PSNR(dB)*	*ΔPSNR(dB)*	*Bitrate(kbps)*
Foreman	FS	36.89		1023.98
	DS	36.13	-0.76	1023.97
	ABME	36.66	-0.23	1023.96
	BBME	36.67	-0.22	1024.00
Akiyo	FS	44.43		1024.00
	DS	44.43	0.00	1023.99
	ABME	44.41	-0.02	1024.01
	BBME	44.44	0.01	1024.02
Flower	FS	29.30		1024.09
	DS	29.17	-0.13	1024.13
	ABME	29.05	-0.25	1023.95
	BBME	29.01	-0.29	1023.83
Mobile	FS	29.25		1023.92
	DS	29.30	0.05	1024.06
	ABME	29.16	-0.09	1023.84
	BBME	29.14	-0.11	1023.89
Tempete	FS	31.55		1023.92
	DS	31.56	0.01	1024.06
	ABME	31.60	0.05	1023.84
	BBME	31.60	0.05	1023.89

Table 16. PSNR comparison for BBME, FS, DS, and ABME at target bitrate of 256kbps. (N=300, M=2).

Sequence	*Method*	*Y_PSNR(dB)*	*ΔPSNR(dB)*	*Bitrate(kbps)*
Foreman	FS	30.49		254.77
	DS	30.02	-0.47	264.10
	ABME	30.17	-0.32	254.51
	BBME	29.99	-0.50	251.15
Akiyo	FS	41.85		254.80
	DS	41.96	0.11	255.11
	ABME	41.90	0.05	255.06
	BBME	41.83	-0.02	255.44
Flower	FS	23.88		276.79
	DS	23.66	-0.22	300.23
	ABME	23.72	-0.16	291.17
	BBME	23.72	-0.16	301.19
Mobile	FS	24.26		255.14
	DS	24.43	0.17	255.23
	ABME	24.19	-0.07	255.19
	BBME	23.74	-0.52	256.78
Tempete	FS	27.00		256.65
	DS	27.20	0.20	256.54
	ABME	26.60	-0.40	258.16
	BBME	26.47	-0.53	263.38

Table 17. PSNR comparison for BBME, FS, DS, and ABME at target bitrate of 512kbps. (N=300, M=2).

Sequence	*Method*	*Y_PSNR(dB)*	*ΔPSNR(dB)*	*Bitrate(kbps)*
Foreman	FS	34.62		509.81
	DS	33.50	-1.12	513.33
	ABME	34.01	-0.61	510.80
	BBME	33.87	-0.75	509.94
Akiyo	FS	43.47		513.27
	DS	43.55	0.08	510.17
	ABME	43.39	-0.08	510.22
	BBME	43.52	0.05	509.61
Flower	FS	26.56		511.15
	DS	26.12	-0.44	511.46
	ABME	26.29	-0.27	511.24
	BBME	26.30	-0.26	511.10
Mobile	FS	27.69		510.27
	DS	27.79	0.10	510.15
	ABME	27.59	-0.10	510.28
	BBME	27.33	-0.36	510.31
Tempete	FS	29.89		511.75
	DS	29.92	0.03	511.70
	ABME	29.67	-0.22	511.60
	BBME	29.55	-0.34	511.74

Table 18. PSNR comparison for BBME, FS, DS, and ABME at target bitrate of 1024kbps. (N=300, M=2).

Sequence	*Method*	*Y_PSNR(dB)*	*ΔPSNR(dB)*	*Bitrate(kbps)*
Foreman	FS	37.37		1020.91
	DS	36.43	-0.94	1026.43
	ABME	36.84	-0.53	1019.25
	BBME	36.75	-0.62	1018.86
Akiyo	FS	44.95		1018.24
	DS	44.97	0.02	1018.58
	ABME	44.91	-0.04	1020.60
	BBME	44.94	-0.01	1021.34
Flower	FS	29.71		1021.85
	DS	29.30	-0.41	1022.60
	ABME	29.57	-0.14	1022.49
	BBME	29.60	-0.11	1021.95
Mobile	FS	30.65		1020.55
	DS	30.74	0.09	1020.54
	ABME	30.60	-0.05	1020.99
	BBME	30.38	-0.27	1020.79
Tempete	FS	32.55		1020.55
	DS	32.55	0.00	1020.54
	ABME	32.38	-0.17	1020.99
	BBME	32.34	-0.21	1020.79

Table 19. PSNR comparison for BBME, FS, DS, and ABME at target bitrate of 256kbps. (N=300, M=3).

Sequence	*Method*	*Y_PSNR(dB)*	*ΔPSNR(dB)*	*Bitrate(kbps)*
Foreman	FS	29.86		261.27
	DS	29.70	-0.16	273.77
	ABME	29.84	-0.02	257.59
	BBME	29.73	-0.13	262.52
Akiyo	FS	41.89		255.55
	DS	41.91	0.02	254.10
	ABME	41.74	-0.15	253.41
	BBME	41.70	-0.19	254.43
Flower	FS	23.71		302.27
	DS	23.51	-0.20	385.80
	ABME	23.67	-0.04	315.07
	BBME	23.66	-0.05	325.69
Mobile	FS	24.55		254.25
	DS	24.50	-0.05	254.34
	ABME	24.48	-0.07	254.24
	BBME	23.65	-0.90	265.39
Tempete	FS	27.23		256.99
	DS	27.55	0.32	256.45
	ABME	27.22	-0.01	256.88
	BBME	26.88	-0.35	265.72

Table 20. PSNR comparison for BBME, FS, DS, and ABME at target bitrate of 512kbps. (N=300, M=3).

Sequence	*Method*	*Y_PSNR(dB)*	*ΔPSNR(dB)*	*Bitrate(kbps)*
Foreman	FS	34.44		508.93
	DS	33.17	-1.27	508.88
	ABME	33.67	-0.77	508.99
	BBME	33.57	-0.87	508.80
Akiyo	FS	43.42		508.20
	DS	43.43	0.01	508.52
	ABME	43.45	0.03	508.45
	BBME	43.24	-0.18	508.19
Flower	FS	26.48		511.50
	DS	25.23	-1.25	512.53
	ABME	26.26	-0.22	511.35
	BBME	26.24	-0.24	511.28
Mobile	FS	28.15		508.75
	DS	28.03	-0.12	508.89
	ABME	28.02	-0.13	508.71
	BBME	27.70	-0.45	508.48
Tempete	FS	30.14		511.18
	DS	30.24	0.10	511.23
	ABME	30.13	-0.01	511.39
	BBME	29.90	-0.24	511.33

Table 21. PSNR comparison for BBME, FS, DS, and ABME at target bitrate of 1024kbps. (N=300, M=3).

Sequence	*Method*	*Y_PSNR(dB)*	*ΔPSNR(dB)*	*Bitrate(kbps)*
Foreman	FS	37.27		1017.80
	DS	36.17	-1.10	1018.36
	ABME	36.51	-0.76	1017.14
	BBME	36.53	-0.74	1017.61
Akiyo	FS	45.09		1017.37
	DS	45.01	-0.08	1017.07
	ABME	45.03	-0.06	1016.95
	BBME	44.84	-0.25	1016.56
Flower	FS	29.58		1023.62
	DS	28.47	-1.11	1023.82
	ABME	29.43	-0.15	1023.47
	BBME	29.45	-0.13	1023.76
Mobile	FS	30.99		1017.16
	DS	30.89	-0.10	1017.52
	ABME	30.91	-0.08	1017.57
	BBME	30.67	-0.32	1017.44
Tempete	FS	32.76		1017.16
	DS	32.77	0.01	1017.52
	ABME	32.70	-0.06	1017.57
	BBME	32.55	-0.21	1017.44

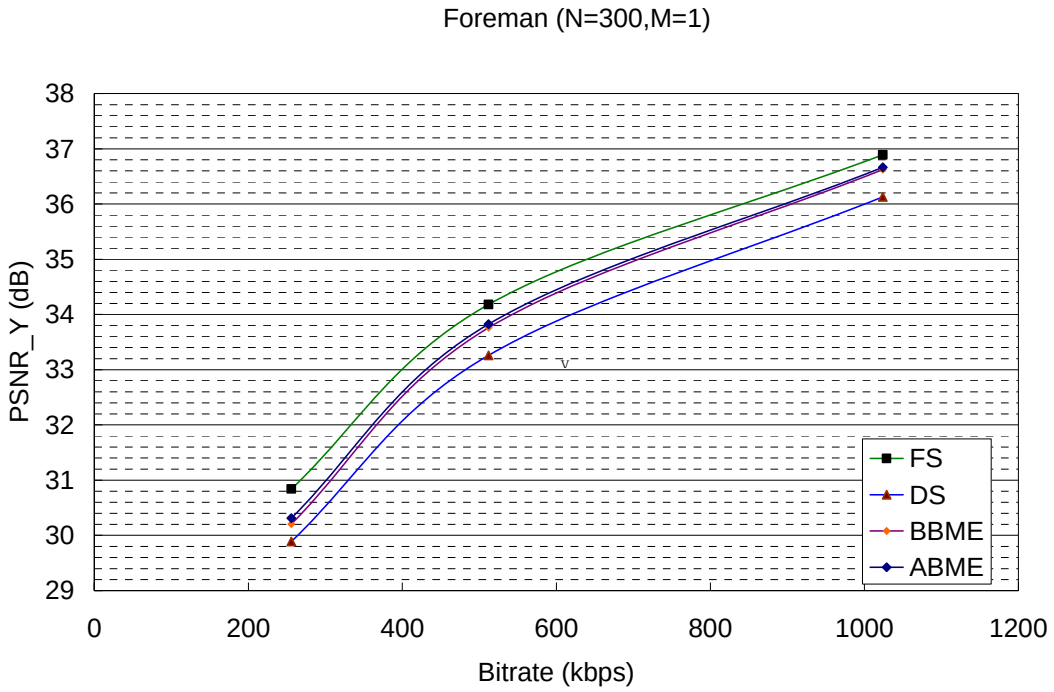


Figure 35. RD curve of BBME, FS, DS, and ABME for Foreman sequence. (N=300, M=1).

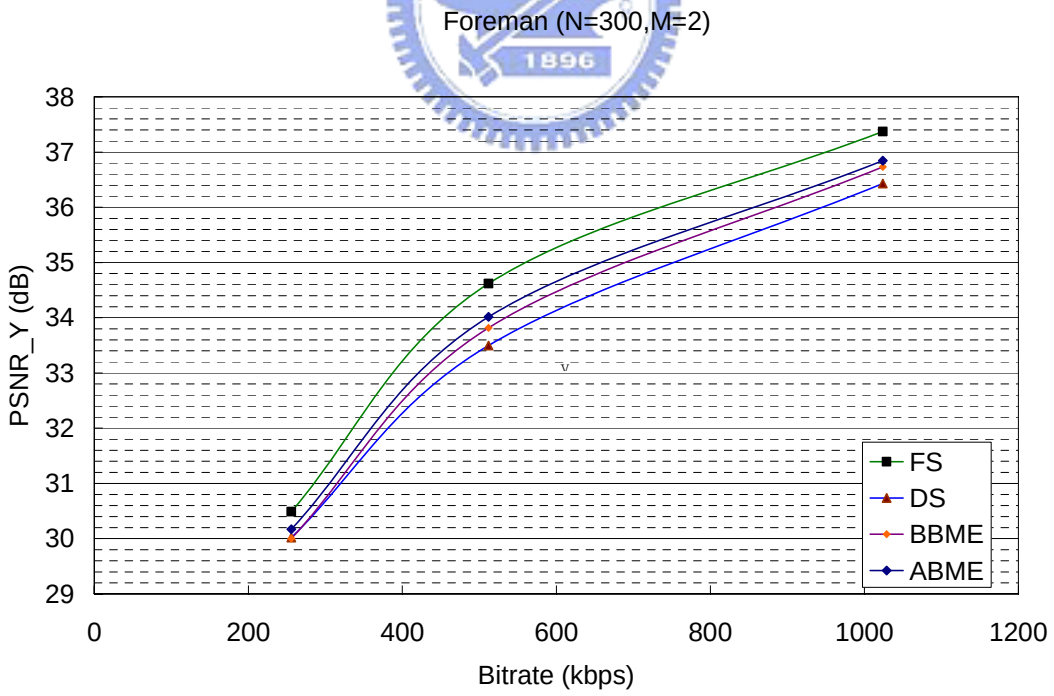
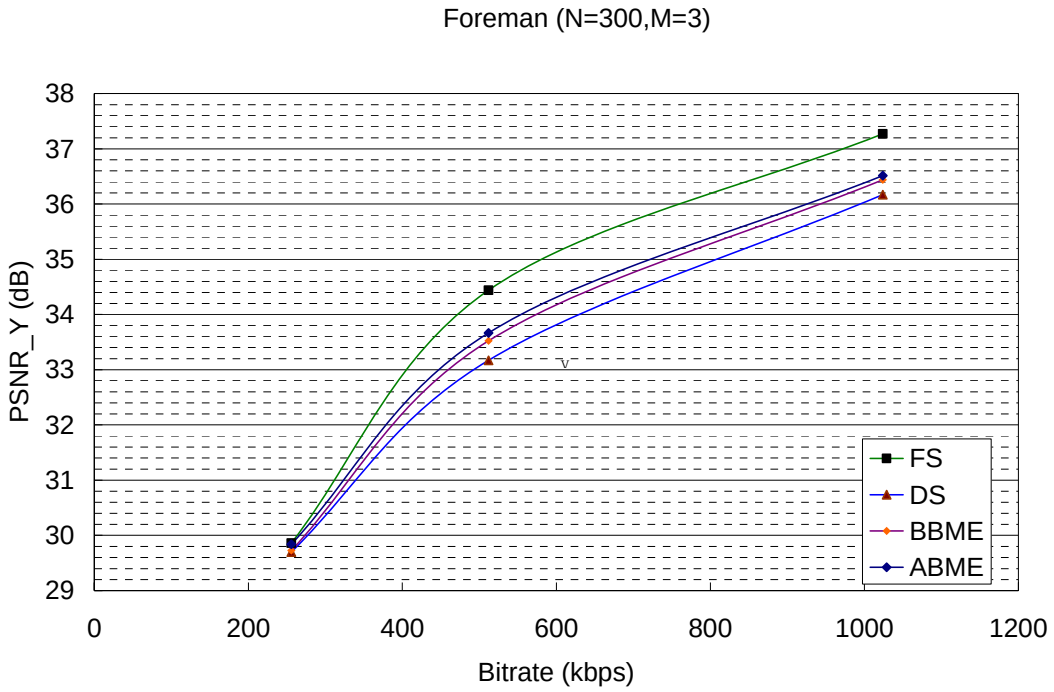
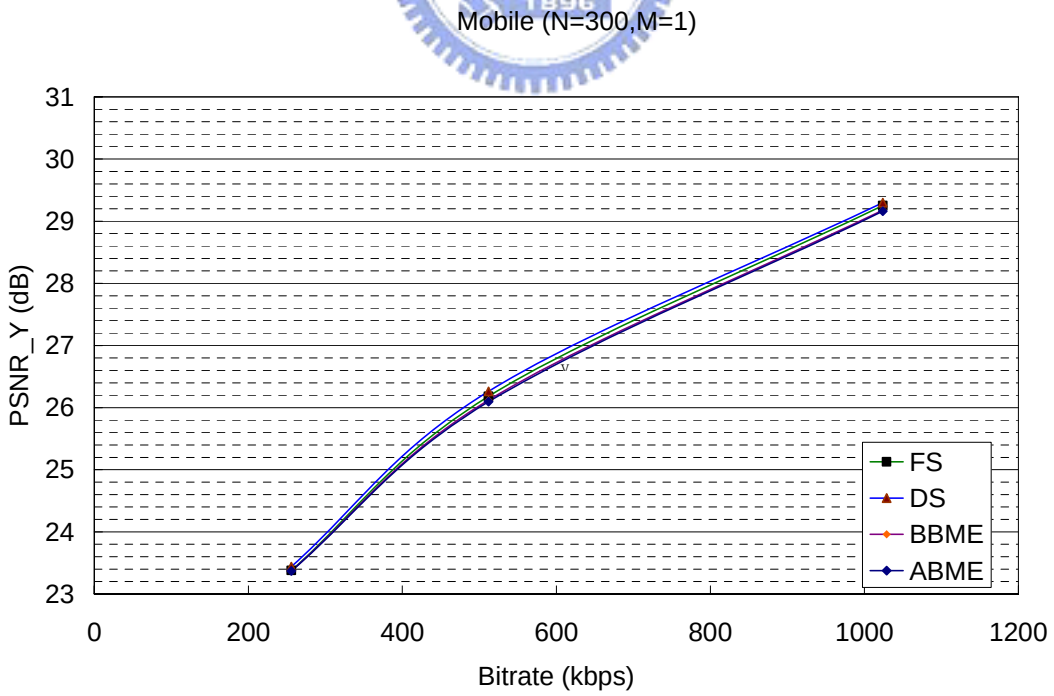


Figure 36. RD curve of BBME, FS, DS, and ABME for Foreman sequence. (N=300, M=2).



(c)

Figure 37. RD curve of BBME, FS, DS, and ABME for Foreman sequence. (N=300, M=3).



(d)

Figure 38. RD curve of BBME, FS, DS, and ABME for Mobile sequence.

(N=300, M=1).

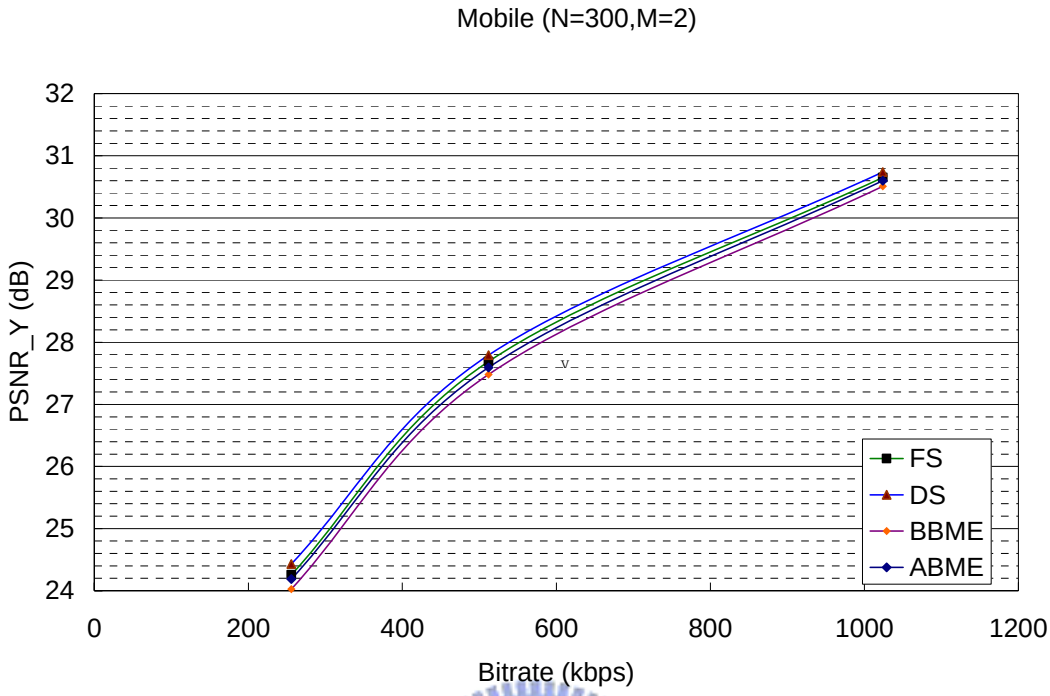


Figure 39. RD curve of BBME, FS, DS, and ABME for Mobile sequence. (N=300, M=2).

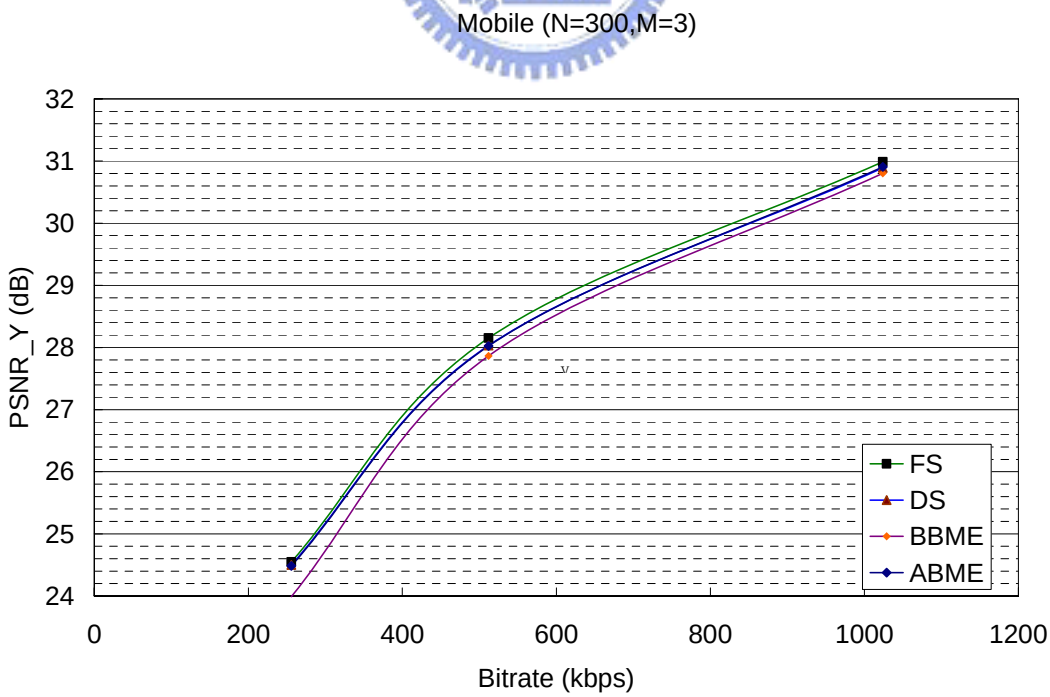


Figure 40. RD curve of BBME, FS, DS, and ABME for Mobile sequence. (N=300, M=3).

5.2 Hardware Design Evaluation

In this section, hardware design specification is presented and compared with the state-of-the-art designs. To evaluate the design functions, the proposed design is emulated on FPGA.

5.2.1 Circuit Design Evaluation

Table 22 summarizes the hardware design specification of proposed bi-directional ME design. The hardware design gate count is about 130 kilo gate count (TSMC 0.18um) with 51 kilo bit SRAM. The proposed design can operate CIF 30fps in only 1.73MHz working frequency and the power consumption is 11.8 mW measured by PrimePower. To compare with the state-of-the-art designs, the proposed algorithm shows the significant throughput improvement over them. Table 23 summaries the design information for [10], [16]-[19]. It shows the proposed design has lowest working frequency for CIF 30fps. The power consumption of our design is 11.8 mW, which mainly comes from MD-SME because we employ 8-bit bit depth in this part. For IME module with BBME, it only takes 0.93 mW for processing CIF 30fps.

Table 22. Hardware design specification of proposed ME for bi-directional search.

Process	TSMC 1P6M 0.18um
Gate count	130.9 k
Memory usage	51 Kbits
Cycles/MB	146
Required freq. for CIF 30fps	1.73 MHz
Power consumption	11.8 mW@1.73MHz
Search range	±16.5 and ±32.5

Table 23. Performance comparisons with state-of-the-art designs.

Design	*Architecture*	*Search range*	*Required freq. for CIF 30fps (MHz)*	*Power for CIF 30fps (mW)*	*On-chip memory (kilo bits)*	*Gate count (kilo)*
FS [16]	2-D systolic	±16.5	48.66	353	N/A	67
GME [17]	Global elimination	±16	61.62	149	24.08	33.32
GDS [18]	Gradient search	±16.5	6.75	2.5	40	250
MRMCS [19]	Multi-resolution search	±16.5	40	N/A	2.3	25
ABME [10]	Binary ME	±16	3.36	2.21	9.80	68.5
Proposed algorithm	Parallel binary IME	±16.5	1.73	11.8	51	130.9

5.2.2 FPGA Based Evaluation Platform

We design a verification platform based on ARM-based platform. Basically, our platform is constructed with the configuration in Figure 41. FPGA based test environment. Our ARM emulation board mainly includes two parts, core module and logic module. In the core module, there are ARM966 CPU, embedded SRAM (1 Mbytes), and external memory interface. On the other hand, the dedicated accelerators are implemented on the logic module which is a FPGA (Filed-programmable Gate Array). Moreover, ARM board employs the AHB bus interfaces to communicate the core module and logic module. Besides, our ARM integrator baseboard employs JTAG (Joint Task Action Group) interface to connect with an ARM MultiICE. The MultiICE connects to a host commuter to conduct the communication between computer and ARM board. In the FPGA environment, we run the codec of MPEG-4 encoder to verify our BBME design. The BBME is implemented on FPGA, and ARM CPU takes charge of the remaining parts of MPEG-4 encoder. The architecture of BBME has passed the verification.

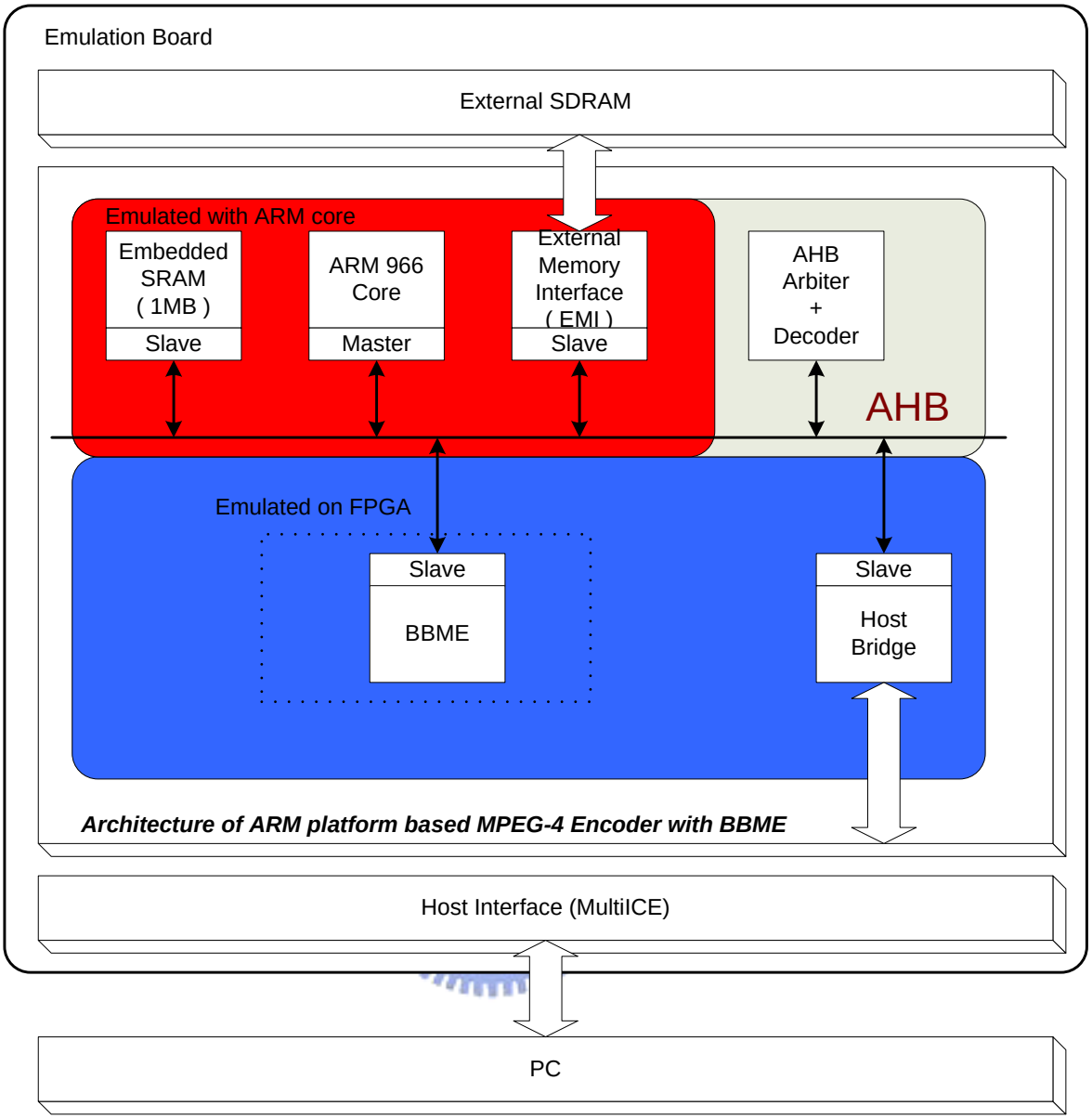


Figure 41. FPGA based test environment

Chapter 6

Conclusion and Future Works

In this thesis, we presented a low power ME design for bi-directional search. The proposed ME design contains two main parts, IME and MD-SME. For low power application, a BBME architecture that can process the forward and backward search in parallel is presented. Such a design can save twice memory access of current frame and then share the operation engine to keep hardware as busy as possible. For P-frame search, this parallel search architecture divides the original search into two sub-groups of partial P-frame search to double the processing throughput. For the hardware design, we proposed three new features to improve the hardware efficiency including MBPPU, hardware efficient LV2 design and integration of 16×16 and 8×8 searches. Besides the optimization of IME, we integrate the MD module into SME to avoid two loops processing of MD and SME to save power. In MD, this work adopts a new line-based algorithm to reduce the longer latency in the original two-dimension and avoid hardware idling. In SME, we adopt an architecture that processes 3 search locations in parallel to reduce memory access and power consumption. To enable the parallel processing of IME and MD_SME, the system pipelining is designed to enhance the throughput and avoid hardware idling. This work completes one bi-directional MB search in 147 cycles with 131 kilo gate count and 51 kilo bits on-chip memory using TSMC 0.18μm technology. The power consumption for CIF 30fps is 11.8 mW.

In the further works, we focus on integrating BBSME with H.264/MPEG-4 AVC standard [20], which supports multiple reference frames. With some minor modifications, this work can be extended to ME with any combination of multiple forward or backward reference frames for throughput improvement

Bibliography

[1] *Information technology - coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbits/s - part 2: video*, ISO/IEC 11172-2, 1993.

[2] *Information technology - generic coding of moving pictures and associated audio information: video*, ISO/IEC 13818-2 and ITU-T Rec. H.262, 1996.

[3] *MPEG-4 overviews*, ISO/IEC JTC1/SC29/WG11 N4668, 2002.

[4] *Video coding for low bit rate communication*, ITU-T Rec. H.263, 1998.

[5] *Video codec for audio visual services at 64 kbit/s*, ITU-T Rec. H.261, 1993.

[6] "*ISO/IEC 14496-5:2001 Final Committee Draft*", MPEG01/N4025.

[7] S. Kalra and M. N. Chong, "Bi-directional motion estimation via vector propagation," *IEEE Trans. on Circuits and Syst. for Video Technol.*, vol.8, pp. 976-987, Dec. 1998.

[8] Y. Keller and A. Averbuch, "Fast motion estimation using bi-directional gradient method," *IEEE Trans. on Image Processing*, vol.13, pp. 1042-1054, Aug. 2004.

[9] J.-H. Luo, *et al.*, "A novel all-binary motion estimation (ABME) with optimized hardware architectures," *IEEE Trans. on Circuits and Syst. for Video Technol.*, vol. 12, pp. 700-712, Aug. 2002.

[10] S.-H. Wang, *et al.*, "Platform based design of all binary motion estimation with bus interleaved architecture," *IEEE International Symposium on VLSI Design, Automation and Test*, pp. 241-244, April 2005.

[11] W. E. Lynch, "Bidirectional motion estimation based on P-frame motion vectors and area

overlap," *IEEE Int'l Conf. on Acoustics, Speech, and Signal Processing*, vol. 3, pp. 445-448, March 1992.

[12] S. Kozu and S. Kulkarni, "A new technique for block-based motion compensation," *IEEE Int'l Conf. on Acoustics, Speech, and Signal Processing*, vol. 5, pp. 217-220, April 1994.

[13] J. Ge and G. Mirchandani, "A new hybrid block-matching motion estimation algorithm," *IEEE Int'l Conf. on Acoustics, Speech, and Signal Processing*, vol. 4, pp. 241-244, May 2002.

[14] M. A. Elgamel, *et al.*, "Systolic array architectures for full-search block matching motion estimation," *third Int'l workshop on Digital and Computational Video*, pp. 108-115, Nov. 2002.

[15] Y.-K. Lai, "A memory efficient motion estimator for three step search block-matching," *IEEE Trans. on Consumer Electronics*, vol. 47, pp. 644-651, Aug. 2001.

[16] J.-F. Shen, *et al.*, "A novel low power full search block matching motion estimation design for H.263+," *IEEE Trans. on Circuits and Syst. for Video Technol.*, vol. 11, pp. 890-897, July 2001.

[17] Y.-W. Huang, *et al.*, "Global elimination algorithm and architecture design for fast block matching motion estimation," *IEEE Trans. on Circuits and Syst. for Video Technol.*, vol. 14, pp. 898-907, June 2004.

[18] M. Miyama, *et al.*, "A sub-mW MPEG-4 motion estimation processor core for mobile video application," *IEEE Journal of Solid State Circuit*, vol. 39, pp. 1562-1570, Sept.

2004.

[19] Jae Hun Lee, *et al.*, "A fast multi-resolution block matching algorithm and its LSI architecture for low bit-rate video coding," *IEEE Trans. on Circuits and Syst. for Video Technol.*, vol. 11, pp. 1289 – 1301, Dec. 2001.

[20] T. Wiegand, *et al.*, "Overview of the H.264/AVC video coding standard," *IEEE Trans. on Circuits Syst. Video Technol.*, vol. 13, pp. 560-576, July. 2003.

簡　歷

戴世炘：民國七十年生於台灣省新竹市。民國九十三年畢業於國立清華大學電機工程學系，之後進入國立交通大學電子工程所攻讀碩士學位。從事多媒體視訊壓縮以及數位電路設計方面的研究。

Shih-Hsin Tai was born in HsinChu in 1981. He received the BS degree in Electronic Engineering, National Tsing Hua University (NTHU), HsinChu, Taiwan in 2004. His current research interests are video compression and digital circuit design.