國　立　交　通　大　學

電子工程學系　電子研究所碩士班

碩　士　論　文

適用於 HEVC 之 270MHz 4Kx2K@60fps 整數像素移動估測設計

A 270MHz 4Kx2K@60fps Integer Pel Motion Estimation Design for High Efficiency Video Coding

研究生：張珊榕

指導教授：張添烜　博士

中華民國　一百零一　年　九　月

適用於 HEVC 之 270MHz 4Kx2K@60fps 整數像素移動估測設計

A 270MHz 4Kx2K@60fps Integer Pel Motion Estimation Design for High Efficiency Video Coding

研 究 生: 張珊榕　　Student: Shan-Jung Chang
指導教授: 張添烜　博士　　Advisor: Tian-Sheuan Chang

國 立 交 通 大 學
電子工程學系　電子研究所碩士班
碩 士 論 文

A Thesis
Submitted to Department of Electronics Engineering & Institute of Electronics
College of Electrical and Computer Engineering
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of Master
In
Electronics Engineering
September 2012
Hsinchu, Taiwan, Republic of China

中華民國　一百零一年 九月

適用於 HEVC 之 270MHz 4Kx2K@60fps 整數像素移動估測設計

研究生: 張珊榕　　　　　　　　　　指導教授: 張添烜 博士

國立交通大學電子研究所碩士班

摘　要

在視訊編碼過程中，整數移動估測(ME)是最複雜的，並且也是即時影像編碼的瓶頸，尤其在最新的影像編碼標準 HEVC 中，因為遞迴式的編碼結構，更大的預測單位大小(PU)，和高等移動向量預測方法(AMVP) ，使 ME 具有相當高的複雜度和大量的記憶體頻寬。

為了要符合即時編碼的需求，本篇論文將會展示一個有效率的整數移動估測積體電路設計。我們的設計首先會省略任何大於16×16非正方形的PU的AMVP，並且採用一個針對 PU 大小為 16×16，16×8，8×16，和 8×8 的五搜尋步驟的預測性強化區域搜尋法(EPZS)，這兩個方法可以大幅降低搜尋點數數量達 78.1%並且維持一定的編碼效果。而硬體架構上則使用交錯不同 PU 大小的 AMVP 和預測性 EPZS 排程，而大於 16×16 的 AMVP 結果則由 16×16 為計算單位組成，這些方法可以提高硬體使用效率並且解決的資料相依性的問題。而為了提高快速演算法的資料重複利用和硬體的簡單性，我們使用兩組 8-way 集合連結快取記憶體特性的暫存器，分別用於 AMVP 和 PEPZS，且使用較小的 tag 位置標示。

從結果可以得到我們提出的演算法與 HEVC 的 HM 6.0 對照的 BDrate 表現，在 YUV 成分分別有 1.3%，1.4%，及 1.6%的降低。我們設計的硬體以 TSMC 90nm 的技術合成，需要 279K 邏輯閘數目量及 8K 位元組的晶片內建記憶體，在工作頻率為 270MHz 的情況下，以處理畫面大小為 4Kx2K，每秒 60 張畫面的影片。

A 270MHz 4Kx2K@60fps Integer Pel Motion Estimation Design for High Efficiency Video Coding

Student: Shan-Jung Chang Advisor: Tian-Sheuan Chang

Institute of Electronics
National Chiao Tung University

Abstract

Motion estimation (ME) processing is the most complex part and the bottle neck of a real time video encoder due to its heavy complexity, and large memory bandwidth, especially for the latest video coding standard, High Efficient Video Coding (HEVC), due to its recursive coding structure, larger prediction unit (PU) size, and advanced motion vector predictors (AMVP).

To meet real time demands, this thesis presents an efficient VLSI ME implementation. This design first skips non-square size AMVP for PU size larger than 16×16 and then adopts a 5-step predictive EPZS (Enhanced Predictive Zonal Search) algorithm only for PU size 16×16, 16×8, 8×16, and 8×8 to reduce the search points significantly by 78.1% while maintain the coding performance. The architecture design uses interlaced AMVP and predictive EPZS scheduling for different PU size and the 16×16 PU based partial AMVP computation for PU size larger than 16×16 to maximize hardware utilization and overcome the data dependency problem. To maximize data reuse while keep design simple for such fast algorithm, the proposed design uses separated 8-way set associative cache based search buffers for AMVP and predictive EPZS with reduced tag address indexing.

The simulation result illustrates the BDrate performance drop by 1.3%, 1.4%, and 1.6% for Y, U, and V component separately, when compared to HEVC reference software HM 6.0. The presented design with 90 nm CMOS process costs 279K logic gates and 8K bytes of on-chip memory and is capable of processing 4Kx2K 60fps video when running at 270 MHz.

誌 謝

首先，要感謝我的指導教授—張添烜博士，不管在大學時期或是研究所時期皆給予我在修課和研究上相當大的建議與協助，讓我學習到很多不同的知識與態度；老師良師益友的形象，深植我心，也讓我學習到更多研究以外的精神。此外，實驗室提供了我們舒適的環境以及充足的設備進行研究，讓我們在努力時可以毫無後顧之憂。

感謝我的口試委員們，交通大學電子工程所郭峻因教授以及資訊工程所彭文孝教授。感謝你們百忙之中抽空前來指導，也因教授們的寶貴意見，讓我的論文更加的充實而完備。

接著我要感謝實驗室的好夥伴們。感謝李國龍和曾宇晟學長的經驗傳承，讓我受用無窮。謝謝王國振學長常常分享一些不同的研究觀點以及知識。謝謝邱亮齊學長的指導和幫助，讓我學習到研究應有的態度及方法。也謝謝陳易群、溫孟勳、曹克嘉、吳英佑學長們的知識分享及照顧。謝謝我的同學們張輔仁、鄭兆傑與許碩文，感謝你們在修課和研究上給我的幫助及對我的包容與關心。謝謝學弟奕晴、效瑜、沛澤、炫谷、瀚萩，你們都是一起討論研究的好夥伴。

謝謝我各個時期的室友，有你們的陪伴，讓我不孤單。也謝謝每位包容我的好朋友及同學們，讓我的學生生活多采多姿。

感謝永遠支持我的家人們：親愛的媽咪、哥哥還有河東，你們的溫暖和包容是我努力的支柱。也謝謝新竹阿姨一家人對我的照顧。

在此，謹將本論文獻給所有愛我以及我愛的人。

Contents

List of Figures

List of Tables

Chapter 1. Introduction

1.1. Motivation

The growing digital media technology such as digital television, mobile phone, internet video streaming, and home entertainment equipment has become an important role in our daily life. Most of us get the information all around the world through these media and devices. However, the multimedia data is too large to transmit or record without compression. The video codec is targeted to compress or decompress digital video effectively; an encoder converts video into a compressed format and a decoder converts compressed video back into an uncompressed format. With the video codec and compression, the multimedia information could be transmitted and stored as digital signal. In order to keep the good quality of the video for the users, the video encoder should effectively compress the data. It is quite complex to keep the video quality the quantity of the video data needed to represent. Because the multimedia data is transmitted or stored under various constraints such as storage size, real time encoding, and power consumption. Therefore, the video compression process needs to exploit the redundancy within or between each frame to reduce the bitrate with minimum video quality loss.

The encoder exploits the subjective, spatial, temporal, and statistical redundancy of the video. Because moving videos contain significant temporal redundancy, for example, successive frames are very similar, it is useful for compression. The amount of data to be coded can be reduced significantly if the previous frame is subtracted from the current frame, and that is why motion estimation and compensation are widely applied to video compression. It is very useful and efficient; however, it occupies very high computational complexity and energy consumption in the encoding process. Many technologies have emerged to get the balance between the coding efficiency and the complexity consumption.

H.264/ AVC has been widely used for the application from the media broadcasting to the personal consumer electronics product nowadays. However, with the development technology of shooting and the economic growth, the demand for higher resolution video becomes larger and larger. High Efficiency Video Coding (HEVC) is a new Standard under the development by the ISO and ITU-T, and it is expected to be more efficient than its predecessor, H.264/ AVC. As well as its improved compression performance, HEVC has greater computational complex and needs longer coding time. Consequently, this becomes a crucial problem in the encoder development and is what we are going to discuss.

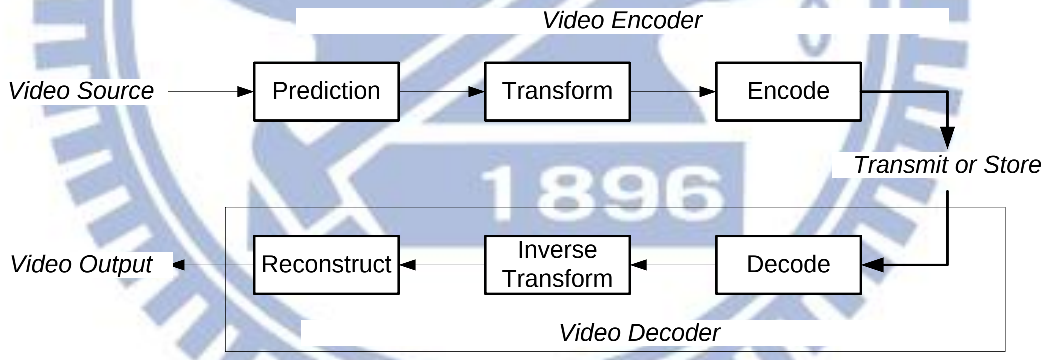


Figure 1-1: Video codec flow

1.2. Thesis Organization

The organization of this thesis is as follows: in chapter 2, we introduce the new video codec standard – HEVC. In chapter 3, we first review some related works and then we will propose a fast hardware-friendly algorithm for our IME architecture. In chapter 4, we propose our architecture of the proposed fast algorithm. Then, in chapter 5, we list the final simulation results to demonstrate our proposed algorithm and some encoded sequence comparison. Hardware implementation results of our motion estimation are also listed in chapter5. In the end, a conclusion is given in chapter 6.

Chapter 2. Overview of HEVC Standards

2.1 Overview of HEVC

High efficiency Video Coding (HEVC) is a new standard by the Joint Collaborative Team on Video Coding (JCT-VC). HEVC has the similar individual building blocks of the hybrid coding to H.264/ AVC, however, the flexibility of the block partitioning for prediction and transform coding is much higher. HEVC improves the coding efficiency compared to H.264/ AVC but it also increases the computational complexity and memory usage because of encoding high resolution video. HEVC is viewed as next-generation video coding standard for HDTV. It provides a bit rate savings for equal PSNR of about 39% for random access applications, 44% for low-delay use, and 25% for all-intra use [1][2].

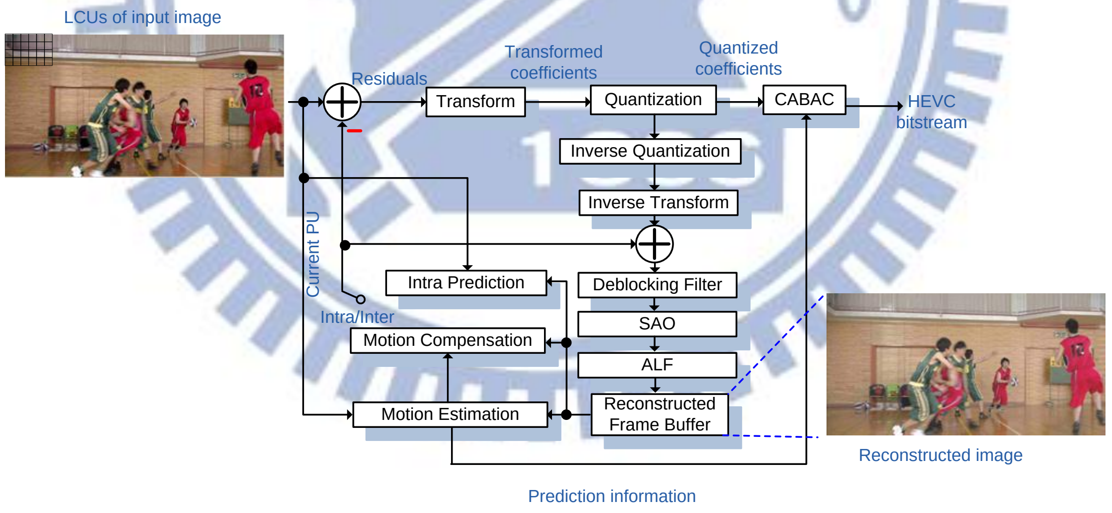


Figure 2-1: HEVC encoding flow.

2.2 Features in HEVC

The video codec concept of HEVC is very similar to the H.264/ AVC, it can be viewed as a generalization of H.264/ AVC. Even though, HEVC has some different characters in this new video coding standard. It has larger prediction and transform blocks and flexible partitioning in those blocks. The spatial intra prediction has much more direction mode. The inter prediction has a new motion vector prediction method for better IME performance and the FME also has two new interpolation filters. HEVC also has a new concept of adaptive loop filter for reconstruction signal after the SAO process [3]. All the improvement in the HEVC helps it gain high coding efficiency and low bit-rate for high resolution video coding application.

2.3 Coding Structure in HEVC

The HEVC standard uses block-based hybrid coding scheme that relies on motion-compensated prediction. Pictures to be encoded will be partitioned into largest coding units (LCUs). The LCU concept is like the macroblock in the standard H.264/ AVC. The maximum allowed size of LCU in HEVC is 64×64.

A LCU consists of different sizes coding units. The Coding Unit (CU) is the basic unit of region splitting used for inter/intra coding. It is square and it may take a size from 8×8 up to the size of the LCU.

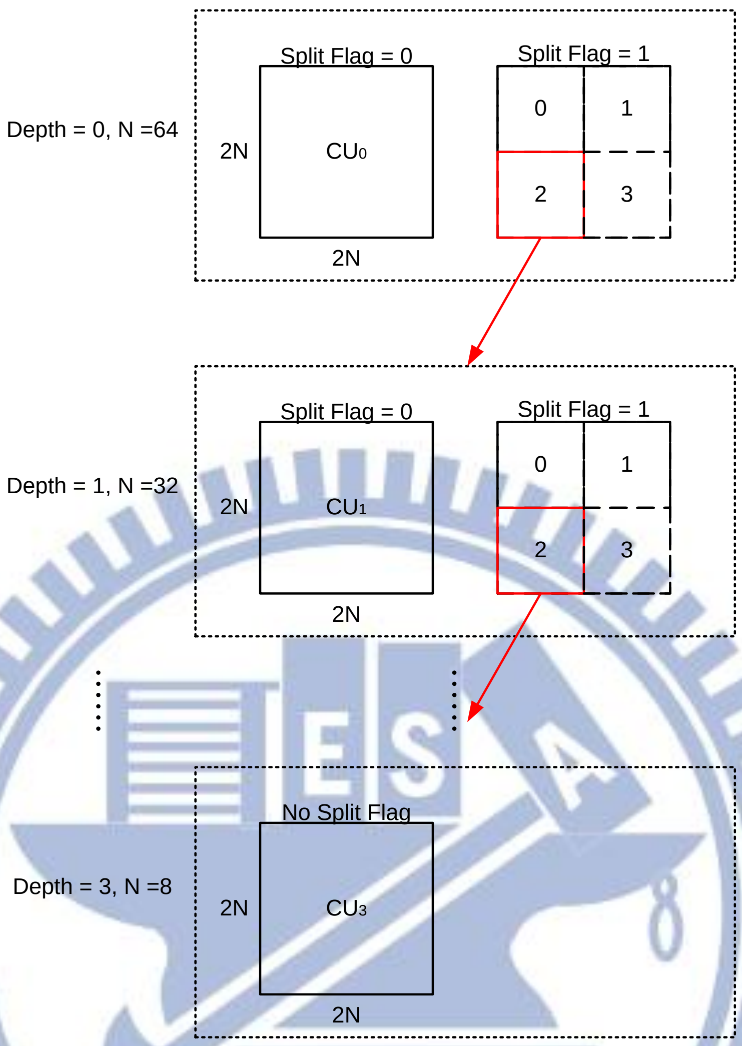


Figure 2-2: Coding Unit recursive partitioning

The CU is recursive split into four equally sized blocks, starting from the LCU. A 2N×2N LCU will be split into 4 N×N CUs with the split flag, and the process goes further to the next depth. This will build up a quad-tree structure of CU blocks. The CUs are sized from LCU to 8×8, and the depth is from 0 to 3.

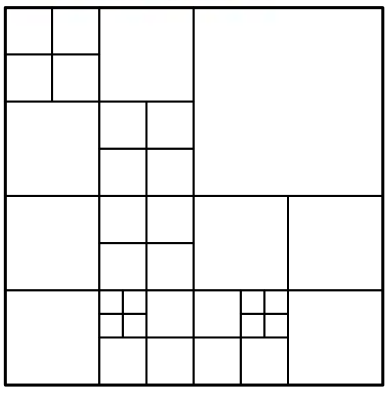


Figure 2-3: Example of Coding Unit structure [3]

The Prediction Unit (PU) is the basic unit used for prediction processes. The PU carries needed information for the prediction and brings the information from top to bottom depth through the prediction flow. The PU is not restricted to being square in shape. A CU could have one or more PUs, and the size for each PU could be as large as CU or as small as 8×4 or 4×8.

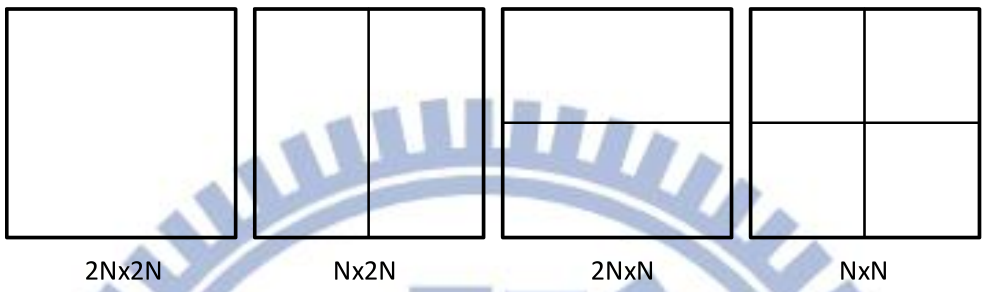


Figure 2-4: Different PU partitions. [3]

The Transform Unit (TU) is the basic unit used for the transform and quantization processes. TU shape depends on PU partitioning mode. TU will be square as PU is square, and it sized from 4×4 up to 32×32. TU is non-square when PU has non-square shape, it may be sized as 32x8, 8x32, 16x4, and 4x16. The TU size is as shown in Figure 2-5. Each CU may contain one or more TUs depend on the PU size, transform and quantization results.

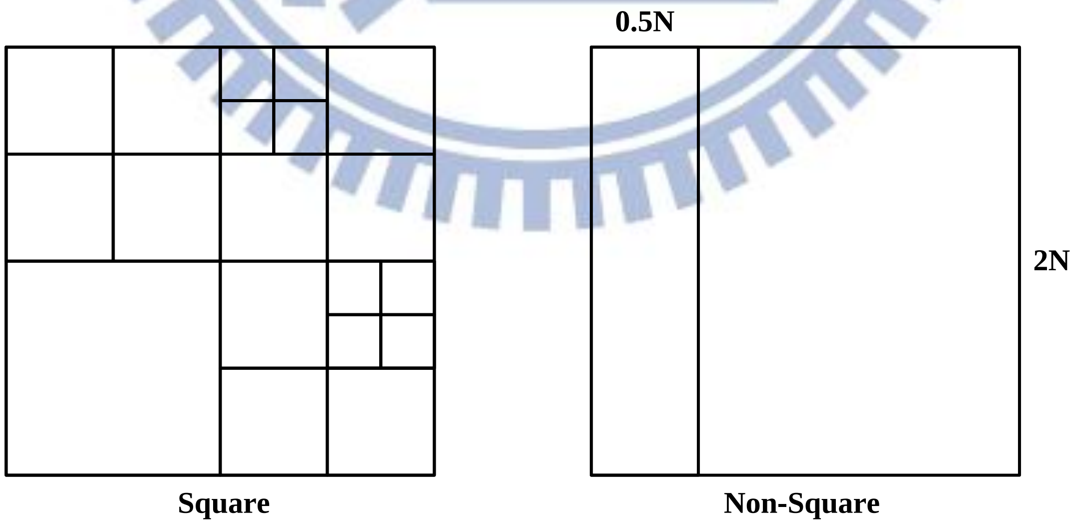


Figure 2-5: TU shape in HEVC

2.4 Advanced Motion Vector Prediction in HEVC

The AMVP is a technique to find the best MVP by the spatio-temporal correlation of motion vector with neighboring PUs. AMVP builds its own motion vector candidate list by firstly checking availability of left, top, and temporal PU positions. It starts checking with A0, A1, B0, B1, B2 (as shown in Figure 2-6), and then the temporary PU. It will remove redundant candidates if it already gets enough candidates for the calculation. The encoder will choose the best predictor from the candidate list by the calculation and then sends the corresponding index of the chosen candidate.

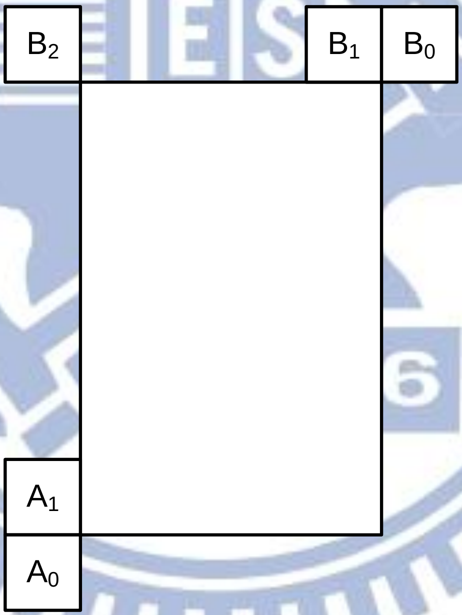


Figure 2-6: Motion vector candidates [3]

2.5 High Efficiency and Low Complexity in HEVC

HEVC encoder supports two encoder configurations, which are High Efficiency (HE) and Low Complexity (LC). HE coding is designed to obtain high compression performance. It supports a bit depth increase up to 10 bit (Internal Bit Depth Increase, IBDI), full capability of loop-filtering process including ALF and CABAC as entropy coder. LC is designed to obtain as high compression performance as it could while keeping the complexity to be low. It will not support IBDI and adaptive loop filtering.

2.6 Challenges in HEVC Implementation

To implement HEVC encoder is a huge challenge. HEVC for HDTV application results in complex video coding. To implement HEVC in hardware, however, will leads to high computational complexity, a large external memory bandwidth, and large size on-chip memory consumption. The computational complexity and memory usage will be our main design challenge if we want to implement HEVC on hardware. The following sections will develop an algorithm and architecture based on the concern of complexity and bandwidth cost.

Chapter 3. Proposed Integer Motion Estimation Fast Algorithm

3.1 Related Works

Block-based motion estimation is adopted by the HEVC. In block-based motion estimation, a block-matching algorithm searches for the best matching block for the current block. During the encoding process, motion estimation is the most important part. The integer motion estimation contributes a lot for the encoding performance but also dominates the complexity and bandwidth consumption because it uses block-matching strategy. To get the good performance while keep the balance with complexity and bandwidth consumption, many motion estimation algorithms have be proposed. Those algorithms focus on reducing computational complexity, and memory bandwidth loading, and that is quite an important issue for the hardware-friendly design. In this section, we will review some related motion estimation algorithms.

3.1.1 Full search algorithm

The well-known full search algorithm (FSA) is the most accurate and simplest method to find the best motion vector. It searches all search points within the search range. However, this approach gets it good performance by its heavy computational complexity since it will calculate every SAD result of all possible search points.

As the full search algorithm searches all the possible point to find the best motion vector in a regular search flow, it is friendly for hardware design. The data within the search range could be fully reused, this decrease the huge memory access amount that motion estimation usually faced.

Because FSS needs to calculate all the SAD results of every search point in the search area, there are some other algorithms have been proposed to decide the checking point to reduce the computation amount.

3.1.2 Three-step search and new three-step search

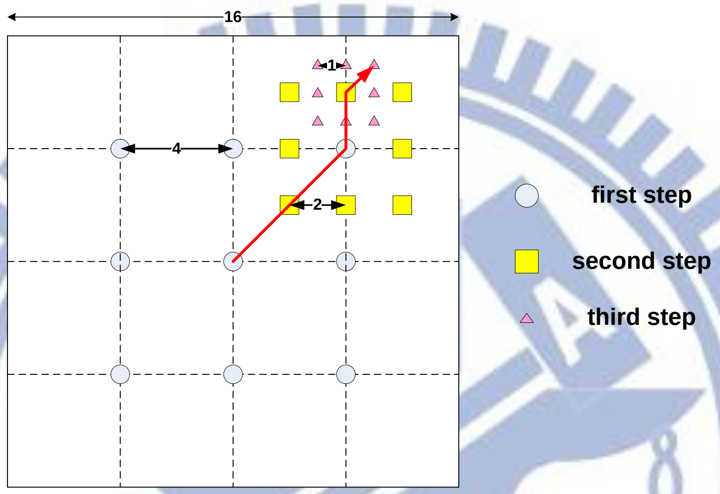


Figure 3-1: TSS search pattern

Three-step search (TSS) [4] is a very popular fast search algorithm because its simplicity and good performance. As the initial step size is picked, 8 search points at a distance of the step size from the center are picked for calculation. The second step center will be moved to the point with the minimum distortion. And the step size for 2nd step and 3rd step is halved and further. From the Figure 3-1, we can see the search points needed for TSS is 25 per macroblock, which is much less than FSS needs.

However, the distribution of the global minimum points in real-world video sequences is centered at the position of zero motion (i.e., search window center) [5], the center-biased new three-step search algorithm (NTSS) has been proposed. It is an improved version of TSS; it adds 8 checking point at the first step, and targeted to achieve better performance with fewer number of search points on average. While it improves the search performance compared to TSS, it somehow loses the simplicity and regularity of TSS [5] [6].

3.1.3 Diamond search

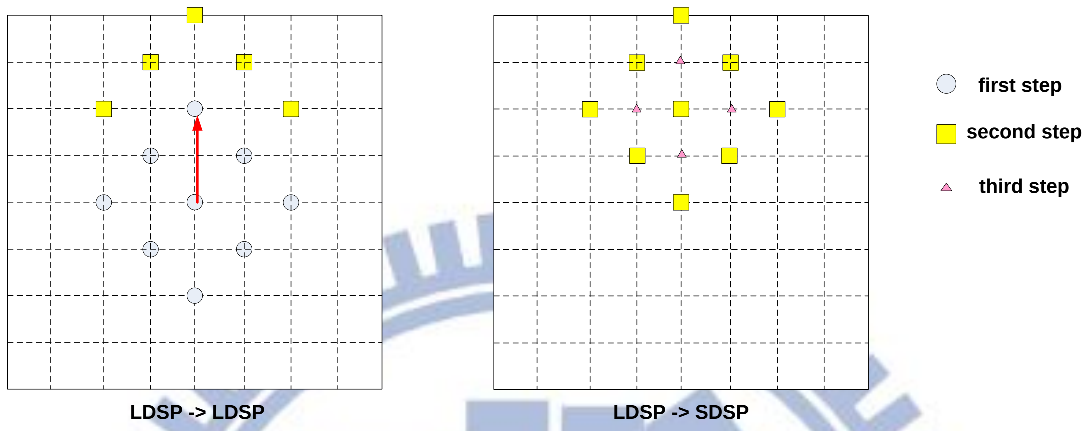


Figure 3-2: DS search pattern

The diamond search algorithm assumes most of motions have a center-biased motion vector distribution [6] [7]. This search algorithm begins with the center. It picks 9 checking points, one is at the center and the other eight one surrounded to compose a diamond shape, to calculate the difference. The diamond search pattern center will be moved to the point with the minimum distortion, which is known as LDSR to LDSR. If the minimum distortion point is at the center, the search pattern will turn into SDSP to check the neighboring four search points. The diamond search step is shown as Figure 3-2.

Diamond search reduces the number of checking point efficiently. The average search point for diamond search algorithm is about 12 to 19. However, the reduced checking points affect considerably the encoding video [8][9]. And sometimes, DS has unrestricted number of steps when it is trapped by local minimum [10].

3.2 EPZS in HM

The HM adopts the EPZS method for the default IME fast search algorithm. This algorithm will check different searching points at each step [9] [11]. The distance between each searching point is doubled when the search is going to next step. For example, the distance between each search point in 1st step is 1, whereas it is 2 and 4 in 2nd and 3rd step separately. The default searching range in HM for the motion estimation is +/- 64 so the search steps for EPZS is seven.

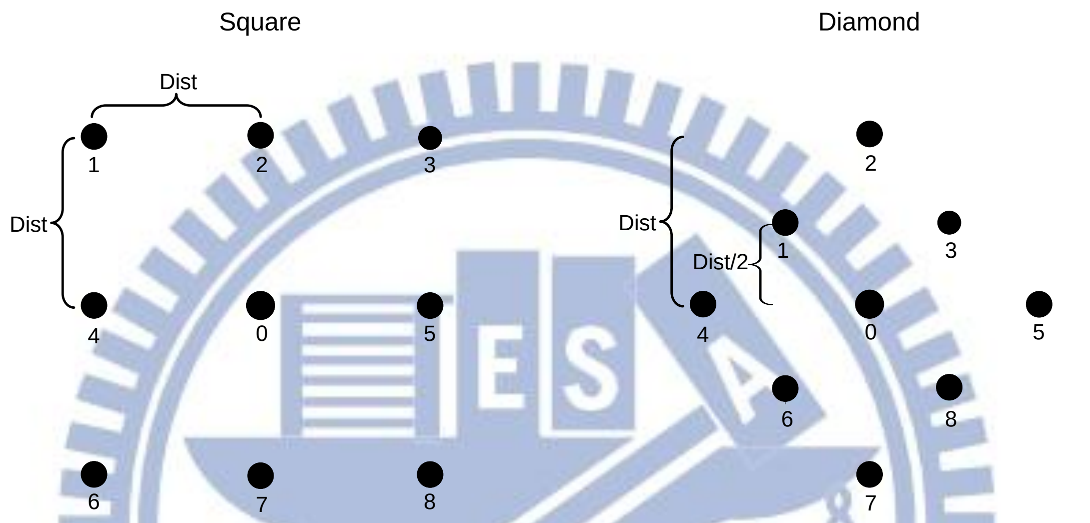


Figure 3-3: The original search point for EPZS square search and diamond search.

There are two different EPZS search algorithm in the HM, one is square and the other is diamond. These two search algorithm have the same search points from 1st step to 4th step, the search point position is like the square search shown in Figure 3-3. While when the step number becomes larger than 4, the search point position is a little bit different for these two search method. The diamond search method still keeps 9 search points while the distance for top-left, top-right, bottom-left, and bottom-right is only half distance which it should be in corresponding step. These two different search methods both have good performance but different data bandwidth. The diamond search method has smaller data bandwidth because it will have more data reuse rate according to the search point position. We adopt diamond search method in EPZS for our later IME fast algorithm development.

3.3 Predictive EPZS in Proposed IME Fast Algorithm

The HM adopts EPZS as its motion estimation algorithm. It has good performance; however, the good performance comes from the dispersive search point. Even though the search points needed for EPZS is less than full search, the total EPZS search point for all PU size is still a large number.

The HM will do EPZS for each PU size, which means that it has to do 57 search point calculation for each PU sized 2N×2N, 2N×N, N×2N, and N×N at each depth. The total number of search point is 23800 for one CU_64×64. This will lead to large computational complexity.

In order to make the motion estimation faster, we need to cut down the number of search points as many as possible. There are many factors that dominate the number of search points. The first one is the search direction. The second one is number of search steps. And the last one is the different PU size do its own search separately.

The later sub sections will talk about the analysis of EPZS search point and its relation with the three factors mentioned above. And we will follow those analyses to develop our fast motion estimation algorithm.

3.3.1 Relationship of direction between each step

The good performance of EPZS comes from the 8 different searching directions at each step. The 8 diverse searching directions results in high computational complexity; however, it can be reduced if we can predict the next direction. In order to predict the next direction, we need to find the relation between two consecutive steps. The relation may be traced by the content of the PU in the test sequence; the final searching point decision at each step would possibly follow the similar direction or route. It is important for us to find the direction difference between the two continuously steps.

From the two following plot, Figure 3-4 and Figure 3-5, it has higher probability to change the direction before step3; this means that the direction at later steps will follow the direction which has been chosen at previous step. This behavior concludes that the search at first 3 steps is much more important than the later ones because the searching direction in the first 3 steps may change more than the later ones, whereas the later ones has higher probability to follow the same direction as the one at previous step.

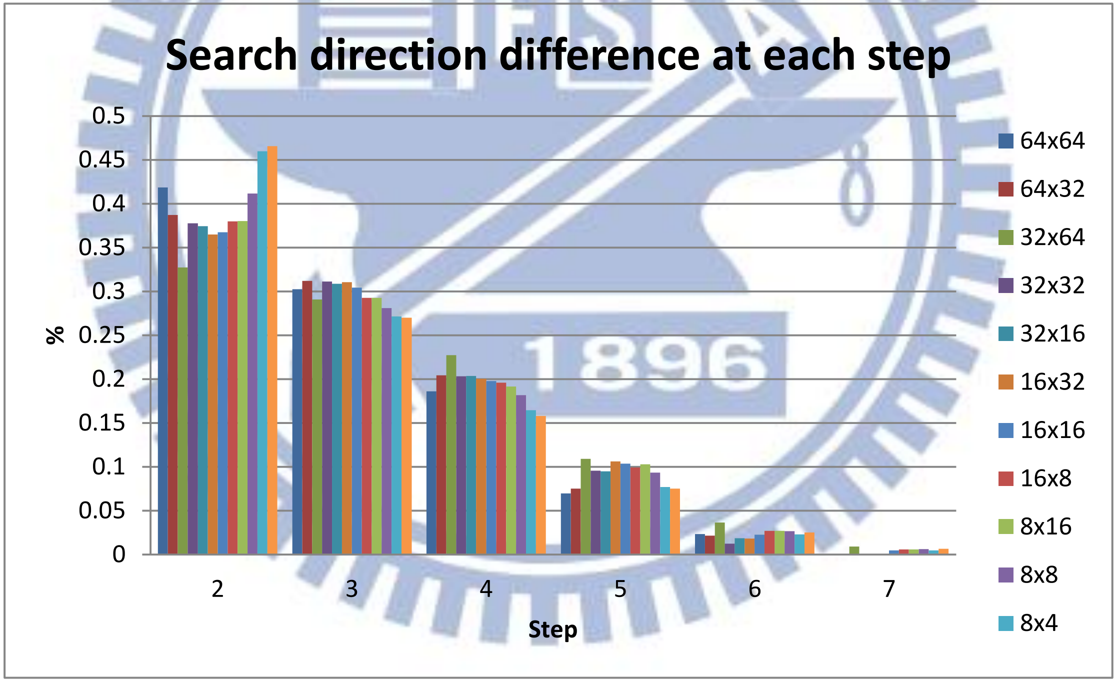


Figure 3-4: Results for Different direction relation with step for RaceHorses_416x240.

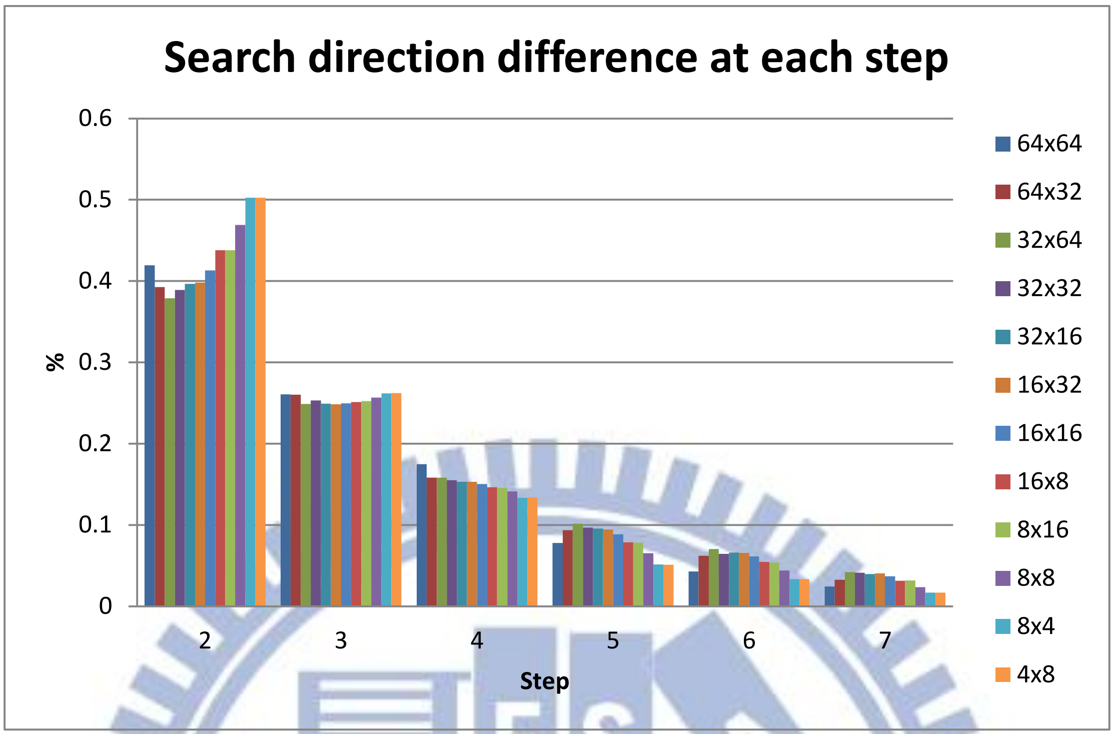


Figure 3-5: Results for Different direction relation with step for BasketballDrive_1920x1080.

3.3.2 Selection of search direction

There are 8 searching points each step for EPZS algorithm, which means that for each PU size, the total searching points will be 8 SPs × 7 steps = 56 searching points. With 56 searching points for each PU sizes, the computational complexity will be very large. To reduce the number of search points and the computational complexity, the decision of search direction for EPZS at each step would be very important.

The simulation results reveal the relationshidip of direction between the previous step and the current one, as shown in Figure 3-6 and Figure 3-7. There are 8 searching points each step so the degree between the previous and current step would be 0^0, +/- 45^0, +/- 90^0, or +/- 135^0. The probability for EPZS to choose the same direction as the previous step is very high whereas is very close to zero when the degree difference is more than 45^0.

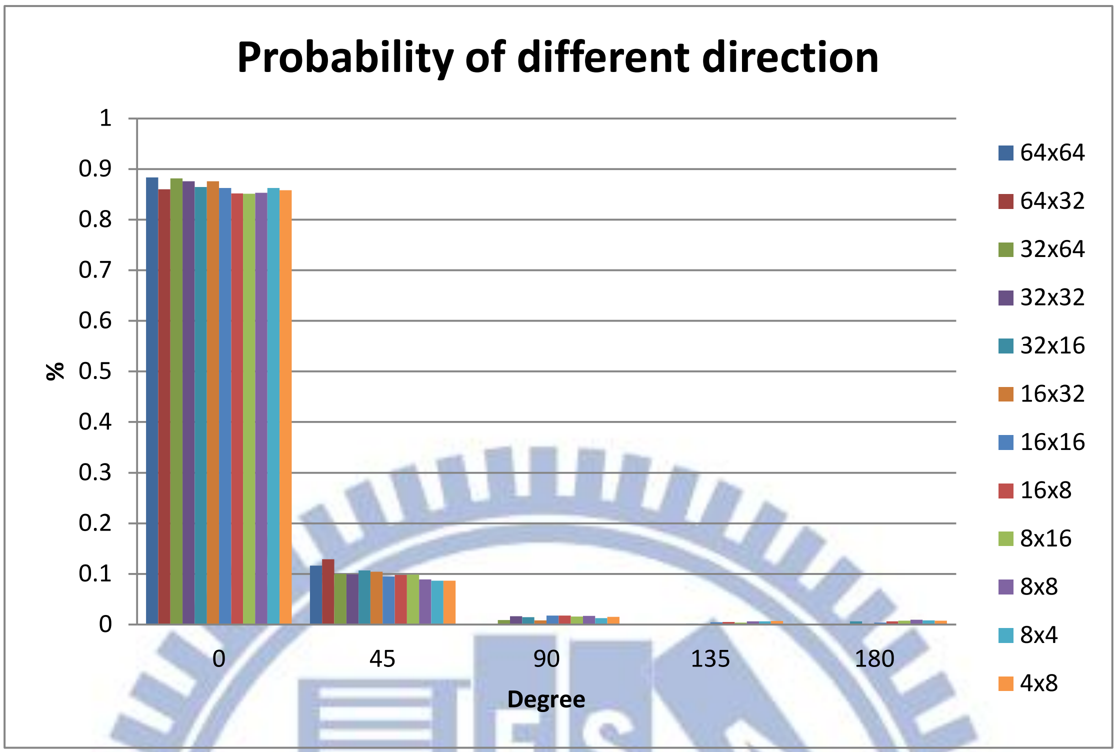


Figure 3-6: Probability of different direction at next step for RaceHorses_416x240.

Table 3-1: Probability of different direction at next step for RaceHorses_416x240

	0^0	$+/-45^0$	$+/-90^0$	$+/-135^0$	$+/-180^0$
Average	0.865065	0.100924	0.012062	0.003022	0.004166

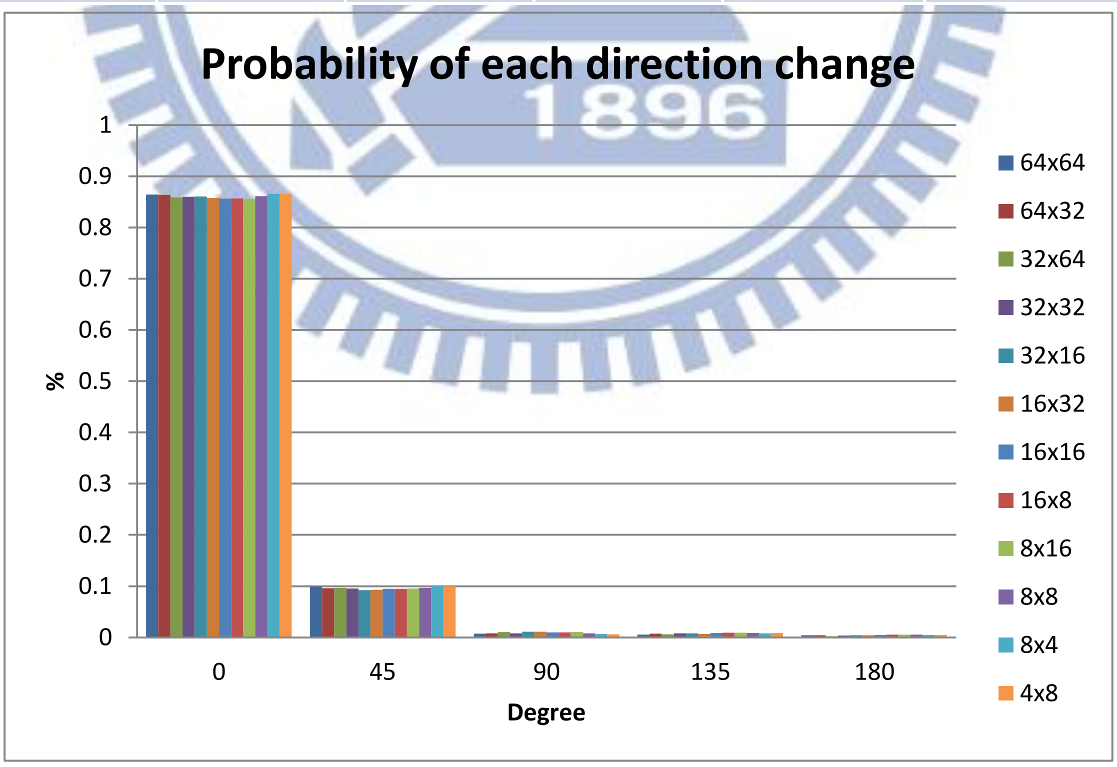


Figure 3-7: Probability of different direction at next step for BasketballDrive_1920x1080.

Table 3-2: Probability of different direction at next step for BasketballDrive_1920x1080

	0^0	$+/-45^0$	$+/-90^0$	$+/-135^0$	$+/-180^0$
Average	0.860406	0.096112	0.008742	0.007685	0.004454

As shown in Figure 3-6 and Figure 3-7, the two difference sequence results, one is Class B with high motion (BasketballDrive_1920x1080) and the other one is Class D with high motion (RaceHorses_416x240). Both of these two sequences have the same behavior that the probability for direction degree of 0^0 or 45^0 is much higher than other degree. According to their behavior, EPZS may only need to check the same direction as the previous step or the two neighboring directions. This predictive EPZS (PEPZS) is shown in Figure 3-8.

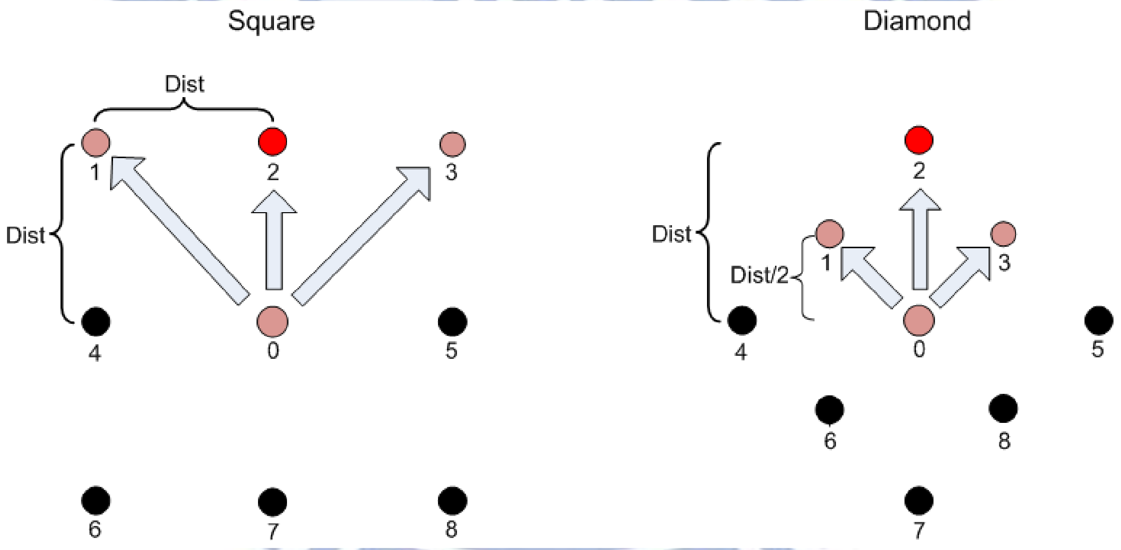


Figure 3-8: Modified searching point at next step.

With the modification mentioned above, we can get the simulation results as shown in Table 3-3. For low and high QP, the bitrate will increase at most 1.02% and the vaule of PSNR decreasing will under 0.03.

Table 3-3: Results for PEPZS.

		Original		Modified		Difference	
	QP	Bitrate	PSNR	Bitrate	PSNR	Bitrate %	PSNR
BasketballPass 416x240	22	1511.172	39.5281	1526.602	39.523	1.02%	-0.0051
	37	148.8864	28.7357	149.3064	28.7121	0.28%	-0.0236

BlowingBubbles	22	1858.804	38.1771	1862.908	38.1744	0.22%	-0.0027
416x240	37	163.208	28.2231	163.656	28.2281	0.27%	0.005
BasketballDrive	22	26190.18	39.3694	26256.25	39.3699	0.25%	0.0004
1920x1080	37	4230.6	35.425	4245.1	35.4284	0.34%	0.0034

3.3.3 EPZS search steps and bandwidth analysis

There is an early termination condition for EPZS. The early termination is on when the EPZS have the same SAD results for 3 steps in a row. It will stop the further EPZS and use the result at the current step.

With the early termination, the computational complexity and bandwidth would be reduced a lot. In the Figure 3-9, the original consumption of memory bandwidth is very high to the modified EPZS in Figure 3-8. As the step goes larger, the increasing amount of memory bandwidth consumption grows quickly for the original EPZS, while it increases not that much for the EPZS modification.

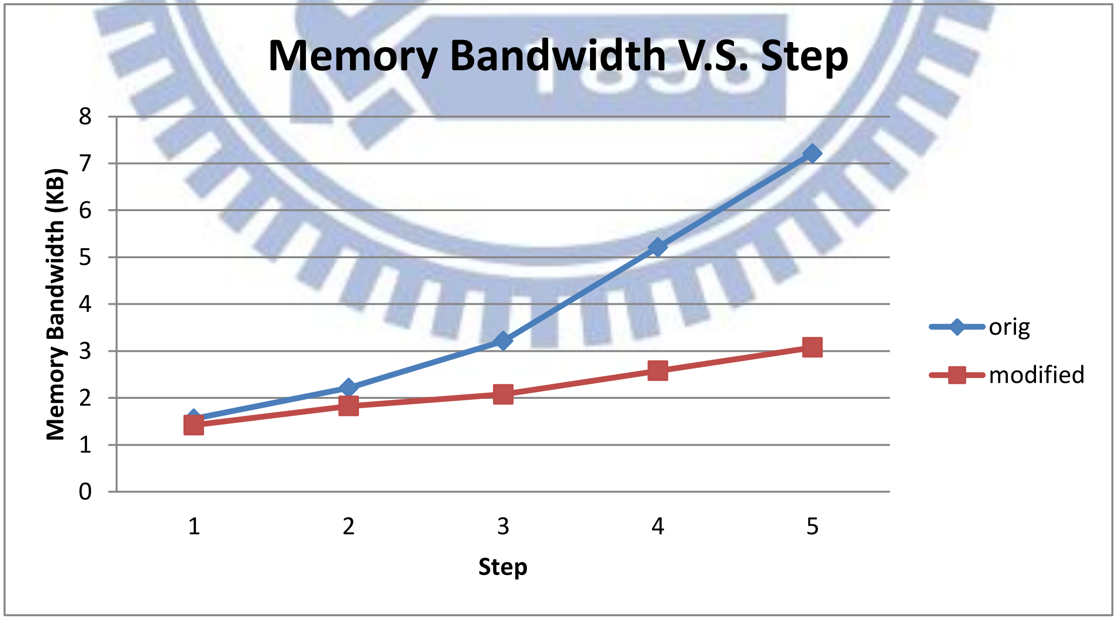


Figure 3-9: The memory bandwidth reduction with early termination.

Table 3-4: The memory bandwidth reduction at each step and with early termination.

Step Count	Step 3	Step 4	Step 5	Step 6	Step 7
Original	1.55 KB	2.21 KB	3.21 KB	5.21 KB	7.21 KB
Modified	1.42 KB	1.83 KB	2.08 KB	2.58 KB	3.08 KB

From the simulation results in Figure 3-10 and in Table 3-5, we can find that most of the searching will be ended at step.3 if it meets the early termination condition, the next searching SAD calculation will be cut off and the system do not to need to load any more for next searching point.

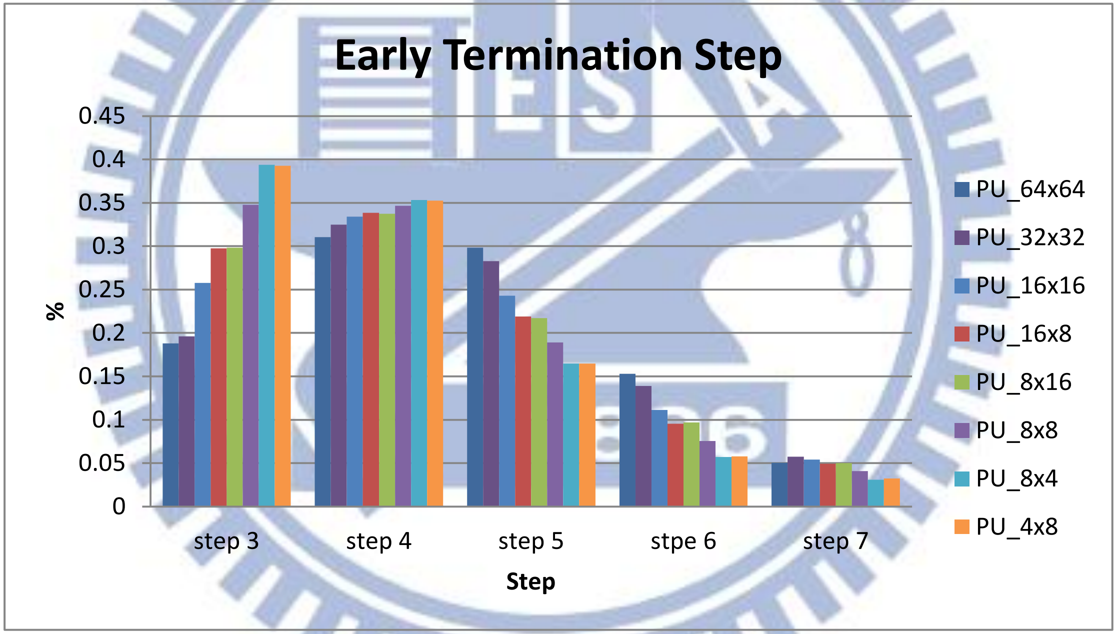


Figure 3-10: Early termination probability at each step.

Table 3-5: Early termination probability for each PU size at every step for RaceHorses_416x240.

	step 3	step 4	step 5	step 6	step 7
PU_64×64	0.188005	0.310596	0.298128	0.152905	0.050368
PU_64×32	0.04785	0.209133	0.247056	0.320096	0.175864
PU_32×64	0.044018	0.174289	0.218883	0.331507	0.231303
PU_32×32	0.196024	0.324827	0.282737	0.139079	0.057333
PU_32×16	0.068425	0.219711	0.2431	0.264324	0.20444
PU_16×32	0.060552	0.210402	0.233748	0.281853	0.213445

PU_16×16	0.257562	0.334026	0.242902	0.11147	0.054039
PU_16×8	0.297505	0.338639	0.218877	0.095603	0.049376
PU_8×16	0.298328	0.337522	0.217305	0.096868	0.049977
PU_8×8	0.347569	0.34672	0.189285	0.07547	0.040957
PU_8×4	0.393838	0.35341	0.16489	0.056984	0.030878
PU_4×8	0.392855	0.352398	0.164719	0.057835	0.032193

The result in Table 3-6 shows the different probability at 4 different QP simulations. From the average value, the probability to early terminate at step 3 is up to 35% to 73%, and more than 50% that EPZS will be terminated at or before step 4. This could help us to decide how to add an early termination constraint on our design if the EPZS can not be finished in specific time.

Table 3-6: Probability of early termination at each step

		Step 3	**Step 4**	**Step 5**	**Step 6**	**Step 7**
RaceHorses 416x240	22	24.2501	17.3408	21.4232	21.0376	15.9484
	27	39.4271	13.7942	17.2538	16.8337	12.6912
	32	56.6103	9.5758	12.1253	12.2618	9.4268
	37	69.9798	7.1739	7.6239	8.538	6.6844
RaceHorsesC 832x480	22	32.3331	12.68	18.8522	19.0209	17.1137
	27	48.4898	10.4471	14.2276	14.0963	12.7392
	32	63.053	8.701	9.7851	9.5555	8.9055
	37	74.6346	7.2277	6.1866	6.0817	5.8694
BasketballPass 416x240	22	46.079	17.3307	15.057	12.3487	9.1845
	27	56.8994	13.6769	11.8859	9.9168	7.621
	32	67.0048	10.4172	8.9616	7.6673	5.9492
	37	76.3703	7.2471	6.4241	5.5786	4.3798
Average	22	**34.2207**	**15.7838**	18.4441	17.469	14.0822
	27	**48.2721**	**12.6394**	14.4557	13.6156	11.0171
	32	**62.2227**	9.5646	10.2906	9.8282	8.0938
	37	**73.6615**	7.2162	6.7448	6.7327	5.6445

With the result and the consideration of the limit working cycle for our hardware design, the performance of early termination should be acceptable if we cut off EPZS after step 5 even after step 3.

3.3.4 Bandwidth reduction by PEPZS

For the EPZS part, the original calculation complexity for each CU_16×16 is very high, and the total data needed to be loaded is 20.75K bytes. From the above experiment results, we can reduced the searching points number from 56 to 29 for each PU size, what is more, with the early termination condition, the search points number will be reduced further. And with the reduced number of the searching points, the pixels needed to be loaded from the off-chip memory will be much lower than the original EPZS algorithm needs. The comparison result is as shown in Table 3-7.

Table 3-7: Memory comparison between different ME algorithm. PU size is 16×16.

	Full Search	**Original EPZS Square**	**Original EPZS Diamond**	**PEPZS**
Memory Access/PU	20.75 Kbytes	7.25 Kbytes	6 Kbytes	2.35 Kbytes
Memory loaded from	MVP blocks + (W+2SR)(H+2SR)	MVP blocks + SP blocks	MVP blocks + SP blocks	MVP blocks + reduced SP blocks

3.3.5 PEPZS base PU selection for search direction

Because of the design specification (will be described in Chapter4) we need to finish the overall IME process in 2220 cycles. Part of high computational complexity comes from the AMVP decision and others come from the PEPZS calculation; for this reason, we try to find a method to cost down the number of SAD calculations in PEPZS.

In order to reduce the computational complexity, we need to lower the search points in the overall PEPZS flow. The original EPZS flow tests all search points for every sized PU; obviously, this will dominate the number of SAD calculation. If we can only check the search points for specific sizes PU, the complexity will be reduced a lot. Therefore, we have to find the specific size unit as the base PU and use the direction of the base PU for the PEPZS. Table 3-8 shows the candidate for the base PU that we tested.

Table 3-8: Tested cases for base PU selection

	Description
Case 1	Use CU_16×16 as base PU for PU size is smaller than 16×16.
Case 2	Use CU_16×16 as base PU for PU size is 16×16, 16×8, or 8×16. Use CU_8×8 as base PU for PU size is 8×8, 8×4, or 4×8.
Case 3	Do its own PEPZS when PU size is smaller or equal to 16×16. Others use CU_64×64 as base PU.
Case 4	Do its own PEPZS when PU size is larger or equal to 16×16. Others use CU_16×16 as base PU.

First, we only use the CU_16×16 as the base PU, and PU size smaller than 16×16 will do the PEPZS with the same direction as CU_16×16. (both reference and modified HMs' AMVP decisions follows the CU_64×64)

Table 3-9: Use only the CU_16×16 as the base PU for search direction (AMVP_64×64)

Case 1	QP	kbps	Y psnr	U psnr	V psnr	
RaceHorses	22	1518.09	39.99	41.36	42.4	13.00%
	27	725.8	35.75	38.42	39.56	
	32	333.31	32.01	36.36	37.46	
	37	157.99	29.15	34.88	35.9	
RaceHorses	22	6343.38	40.1	41.62	42.93	13.20%
	27	2606.95	36.22	38.9	40.45	
	32	1139.65	32.86	36.94	38.61	
	37	513.67	29.92	35.6	37.27	
BasketballPass	22	1845.84	41.07	43.76	43.18	5.50%
	27	922.17	37.06	40.85	39.88	
	32	446.51	33.49	38.72	37.46	
	37	224.6	30.51	37.25	35.74	

In the second case, we use the CU_16×16 and CU_8×8 as the base PU, and PU size between CU_16×16 and CU_8×8 will do the PEPZS with the same direction as CU_16×16, other PU size smaller than CU_8×8 will follow the direction of CU_8×8. (both reference and modified HMs' AMVP decisions follows the CU_64×64)

Table 3-10: Use CU_16×16 and CU_8×8 as base PU for search direction (AMVP_64×64)

Case 2	QP	kbps	Y psnr	U psnr	V psnr	BDrate
RaceHorses	22	1462.7	40.01	41.37	42.4	9.10%
	27	701.15	35.79	38.45	39.58	
	32	327.29	32.06	36.36	37.46	
	37	156.59	29.17	34.89	35.94	
RaceHorses	22	6231.95	40.09	41.62	42.93	10.70%
	27	2548.61	36.23	38.9	40.47	
	32	1120.69	32.89	36.96	38.63	
	37	509.02	29.94	35.61	37.26	
BasketballPass	22	1807.63	41.08	43.77	43.2	3.60%
	27	903.28	37.08	40.86	39.91	
	32	442.1	33.5	38.75	37.5	
	37	223.03	30.52	37.27	35.77	

From the above two tables, we can find that the performance with base PU does not reach the requirement, we try the third case: do PEPZS when PU size is smaller than CU_16×16. The results is better than the previous two cases, it means that the direction for each PU should been chosen by its own PEPZS but not by the base PU.

Table 3-11: Use its own search direction when PU size is smaller or equal to CU_16×16 (AMVP_64×64)

Case 3	QP	kbps	Y psnr	U psnr	V psnr	BDrate
RaceHorses	22	1428.31	40.02	41.39	42.42	6.10%
	27	683.97	35.81	38.47	39.62	
	32	321.11	32.1	36.41	37.5	
	37	155.15	29.2	34.93	35.99	
RaceHorses	22	6144.21	40.09	41.63	42.94	8.70%
	27	2500.58	36.24	38.93	40.49	
	32	1102.82	32.92	36.99	38.66	

	37	504.16	29.97	35.63	37.3	
BasketballPass	22	1788.14	41.08	43.78	43.21	2.20%
	27	893.05	37.09	40.9	39.91	
	32	437.41	33.52	38.8	37.52	
	37	222.2	30.54	37.3	35.8	
BasketballDrill	22	3754.03	40.37	43.01	43.66	2.70%
	27	1788.61	37.12	40.42	40.76	
	32	863.75	34.26	38.3	38.45	
	37	438.91	31.82	36.41	36.27	

Below shows the Table 3-12, the BDrate result of all modification for the base PU chosen and PEPZS. (Both reference and modified HMs' AMVP decisions follow the best candidate of CU_64×64)

Table 3-12: Results of 3 different cases modification for PU chosen, with AMVP_64×64

Modification	Original EPZS	PEPZS	CU_16×16 As base PU	CU_16×16,CU_8×8 As base PU	PEPZS >=CU_16×16	PEPZS <=CU_16×16
RaceHorses 416x240	2.70%	5.50%	13.00%	9.10%	6.80%	6.10%
RaceHorses 832x480	3.00%	4.70%	13.20%	10.70%	6.80%	8.70%
BasketballPass 416x240		1.60%	5.50%	3.60%	2.80%	2.20%

The two cases, doing PEPZS for PU sizes larger than CU_16×16 and doing PEPZS for PU sizes smaller than CU_16×16, give us very similar BDrate performance. In the concern of processing period for IME, the modification doing PEPZS for PU sizes smaller than CU_16×16 is much more appropriate for the hardware design.

From the Table 3-11 we can find the BD-rate performance is not very good, the BD-rate is up to 8.7%, which is too high for the video codec. The modification in Table 3-9 to Table 3-11 is combining the modification of PEPZS and uses the AMVP result of CU_64×64 for each PU. We can see that the performance is too low when it only with one AMVP for every PU. Table 3-13 shows the result of PEPZS with the original AMVP process, and the BD-rate performance is very good for the two test sequence. This shows the importance of the AMVP process, and that is what we are going to discuss in section 3.3.

Table 3-13: Result of original AMVP and PEPZS.

HM_6.0		**Original AMVP + Do all PEPZS under CU_16×16**				**BDrate**		
		kbps	Y psnr	U psnr	V psnr	Y	U	V
BasketballPass	22	1757.49	41.11	43.82	43.22	0.30%	1.20%	0.50%
	27	874.24	37.13	40.94	39.95			
	32	428.01	33.6	38.83	37.59			
	37	218.07	30.65	37.38	35.88			
RaceHorses	22	1364.76	40.07	41.43	42.45	0.80%	1.10%	1.60%
	27	652.12	35.93	38.51	39.67			
	32	307.77	32.26	36.45	37.56			
	37	149.56	29.37	35.01	36.04			

3.4 Modified AMVP in Proposed IME Fast Algorithm

AMVP is a new concept in HEVC and this brings the good performance for HEVC encoding. The AMVP is very important as it is one process that will results in large computational complexity and cost a high memory bandwidth.

The HM does the AMVP for each PU to get the best MVP. This help HM gain the good performance. However, the AMVP calculation for every PU leads to high complexity. What is more, the data for the calculation of MVP candidates are usually very diverse from the other. All these conditions are important issues that we need to concern when we develop the IME fast algorithm. The later sub sections will show the AMVP analysis and our result of modified AMVP.

3.4.1 CU_64×64 as AMVP base PU for LCU_64×64

In HM 6.0, the original IME would do AMVP calculation for each PU size; each AMVP processes two candidates SAD calculations. There are total 401 PUs in each CU_64×64 and this leads to high computational complexity and cost much data bandwidth. For the hardware design, we need to concern the complexity and bandwidth and that is why we want to reduce the calculation of AMVP. What is more, AMVP will decide which searching point is the starting point for PEPZS so the AMVP is very important for the both IME and PEPZS performance.

The simplest way to reduce the AMVP complexity is use the same MVP candidate for every PU in a CU_64×64. After the AMVP of CU_64×64, the candidate will be transmitted to the next depth and all of the rest sized PUs will use the same candidate as their PEPZS starting searching point. This could definitely reduce the complexity a lot; however, as shown in Table 3-14, the BD-rate compared to the original HM which allows every PU to choose its own best MVP is very bad. The BD-rate is 6.6% for luma component; this is not a good performance if we want to apply this on the high resolution video encoding.

Table 3-14: Result of AMVP_64×64 and original EPZS.

HM_6.0		**AMVP_64×64 + Original EPZS**				**BD-rate**		
		kbps	Y psnr	U psnr	V psnr	Y	U	V
RaceHorses 416x240	22	1386.83	39.80	41.07	42.12	6.60%	5.80%	5.30%
	27	665.54	35.68	38.17	39.34			
	32	311.84	32.04	36.15	37.30			
	37	149.67	29.17	34.69	35.79			

3.4.2 Different AMVP base PU analysis and results

Because the experience from the section 3.3.1, we can find that the performance will not match requirement if HEVC only supports only one best MVP for every PU in a LCU sized as 64×64. Here we try some combination of PEPZS and AMVP modification, as shown in Table 3-15, to find the best method for the PUs' AMVP chosen. These combinations are being considered because of the importance of the AMVP size that tested.

Table 3-15: Tested cases for AMVP base PU selection.

	Description
Case 1	Only do AMVP for PU_64×64 and PU_8×8. Do PEPZS for PU size smaller than PU_16×16.
Case 2	Do AMVP for PU_64×64 and PU size smaller than PU_8×8. Do PEPZS for PU size smaller than PU_16×16.
Case 3	Only do AMVP for PU_64×64 and PU_16×16. Do PEPZS for PU size smaller than PU_16×16.
Case 4	Do AMVP for PU_64×64 and PU size smaller than PU_16×16. Do PEPZS for PU size smaller than PU_16×16.

Because the small size plays an important role in the motion estimation if the sequence content has too complex texture, Case 1 tries to support one more AMVP than the test in 3.4.1 for the motion estimation. This one more AMVP should be supported for the smallest LCU size, CU_8×8, to improve the performance. As shown in Table 3-16, the performance is a little bit better than that in Table 3-14. This one more AMVP shows how the starting point is important for the small sized PUs.

Table 3-16: Result of case1 AMVP base PU selection.

Case 1		**AMVP_64×64 + AMVP 8×8 PEPZS under16×16**				**BDrate**		
	QPISlice	kbps	Y psnr	U psnr	V psnr	Y	U	V
RaceHorses	22	6051.97	40.0894	41.5927	42.917	6.10%	6.60%	7.10%
	27	2387.09	36.2596	38.9254	40.4929			

	32	1049.2	33.0326	37.0085	38.6836			
	37	482.197	30.1269	35.6694	37.3525			
RaceHorses	22	1389.13	40.0117	41.3806	42.4057	3.40%	3.60%	4.50%
	27	660.031	35.8699	38.4712	39.6141			
	32	312.311	32.2093	36.4489	37.5347			
	37	152.09	29.3088	34.9522	36.0048			

According to the results in Case 1, it may be improved by adding more AMVP to the rest small sized PUs. Case 2 supports more AMVP to the motion estimations, for PUs which sized smaller than 8×8 will choose its own best MVP for the PEPZS calculation. As shown in Table 3-17, the BD-rate is improved by about 1% for Class C sequence, RaceHorses_832x480, and about 0.4% for Class D.

Table 3-17: Result of case2 AMVP base PU selection.

Case 2		**AMVP_64×64+AMVP<=8×8 PEPZS<=16×16**				**BDrate**		
	QPISlice	kbps	Y psnr	U psnr	V psnr	Y	U	V
RaceHorses	22	5966.92	40.0816	41.6326	42.943	5.10%	4.90%	4.80%
	27	2378.21	36.2768	38.947	40.5182			
	32	1040.7	33.025	37.0202	38.7193			
	37	472.734	30.1327	35.6867	37.3919			
RaceHorses	22	1404.12	40.0003	41.3657	42.3959	3.80%	3.80%	4.60%
	27	664.011	35.8531	38.4745	39.6267			
	32	311.26	32.2085	36.4509	37.5436			
	37	150.973	29.321	34.9704	35.9981			

Because HEVC is targeted for high resolution video codec, such as 3840x2160, the small sized PUs for complex texture block matching may be larger than 8×8. Case 3 tries to support only one more AMVP for CU_16×16 to find which the second best AMVP base is from AMVP by CU_64×64.

As shown in Table 3-18, the result in Case 3 is not as good as Case 1. If AMVP is only supported for the CU_16×16, the performance will not be good enough for our needs.

Table 3-18: Result of case3 AMVP base PU selection.

Case 3		**AMVP_64×64+AMVP_16×16 PEPZS<=16×16**				**BDrate**		
	QPISlice	kbps	Y psnr	U psnr	V psnr	Y	U	V
RaceHorses	22	5989.69	40.0781	41.6304	42.9481	7.00%	6.80%	7.60%
	27	2406.9	36.2769	38.942	40.4997			
	32	1060.58	33.008	36.9997	38.6722			
	37	487.302	30.092	35.6705	37.3545			
RaceHorses	22	1405.53	40.0317	41.3919	42.4173	5.60%	5.00%	5.80%
	27	671.524	35.8416	38.4912	39.6498			
	32	315.717	32.1471	36.4235	37.5117			
	37	152.565	29.2599	34.9383	35.9536			

Case 4 tries to improve the performance in Case 2, which allows PUs to get its own best MVP through AMVP if it sized smaller than 8×8. Case 4 supports more MVP chosen opportunity for PUs; it allows PUs to choose its starting point for motion estimation if it takes size smaller than 16×16. Every PU which is smaller than 16×16 can do AMVP to get the best MVP. It performs much better than the Case 2 does, the results is in Table 3-19, the BD-rate is now under 3.0% for both Class C and Class D sequence.

Table 3-19: Result of case4 AMVP base PU selection.

Case 4		**AMVP_64×64+AMVP<=16×16 EPZS<=16×16**				**BDrate**		
	QPISlice	kbps	Y psnr	U psnr	V psnr	Y	U	V
RaceHorses	22	5839.1	40.0797	41.6419	42.9618	2.40%	2.60%	3.00%
	27	2324.09	36.3227	38.9606	40.5236			
	32	1031.34	33.093	37.0314	38.7121			
	37	479.434	30.1706	35.7055	37.3935			
RaceHorses	22	1366.98	40.0673	41.4097	42.455	1.50%	1.50%	2.40%
	27	653.342	35.9128	38.5038	39.654			
	32	309.836	32.2465	36.4663	37.5515			
	37	151.425	29.347	34.9973	36.0134			

3.5 Combination of PEPZS and AMVP Modification for Proposed IME Fast Algorithm

In section 3.3 and section 0, we have different modification for PEPZS and AMVP separately. IME is composed of MV prediction and motion estimation. Both of them are important that we cannot ignore neither MVP nor ME calculation. We will try to combine the modification is section 3.3 and section 0 to get the best performance as the method for the IME in our algorithm.

3.5.1 Different combination of modified fast algorithm

Because of the limited working cycles (will be described in section 4.1), it is important for the modification to concern the number of SAD calculations at each PU size and steps. The following combinations will be based on those we have done in section 3.2 and 3.3. We can get the importance of AMVP and PEPZS for each PU size as we have the performance from the combinations as shown in Table 3-20. Each combination is targeted to decide the best PU size for AMVP and PEPZS calculation.

Table 3-20: Tested combinations for fast algorithm.

	Description
Combination 1	Do AMVP for PU_64×64. Do AMVP and PEPZS for PU sized 16×16, 16×8, 8×16, and 8×8.
Combination 2	Do AMVP for PU_64×64 and PU size smaller than 16×16. Do PEPZS for PU sized 16×16, 16×8, 8×16, and 8×8.
Combination 3	Do AMVP and PEPZS 1st step for PU_64×64, 32×32, and PU smaller than 16×16. Do rest PEPZS steps for PU sized 16×16, 16×8, 8×16, and 8×8.
Combination 4	Do AMVP and PEPZS 1st step for PU_64×64, 32×32, 8×4, 4×8. Do AMVP and PEPZS for PU 16×16 to 8×8. Search range for PU larger than 8×8 is +/-16, for 8×8 is +/-4

The Combination 1 tries to reduce the number of SAD calculation in the IME so it cuts off the AMVP and PEPZS if the PU size is smaller than 8×8. As shown in Table 3-21, the results is not as good as Table 3-19. It may because the PUs smaller than 8×8 do important block matching when the content is complex in a sequence.

Table 3-21: Result of tested combination 1.

Combination 1		**amvp_64×64+amvp_pepzs_under16×16 8×8 cut off pepzs+amvp**				**BDrate**		
	QP	kbps	Y psnr	U psnr	V psnr	Y	U	V
RaceHorses	22	5902.06	40.0788	41.6368	42.9501	4.30%	4.50%	5.10%
	27	2363.48	36.3117	38.9552	40.5176			
	32	1044.01	33.0624	37.0171	38.6952			
	37	485.198	30.1442	35.6776	37.3617			
RaceHorses	22	1394.57	40.0492	41.4011	42.4278	4.40%	4.50%	4.90%
	27	666.513	35.8795	38.4684	39.6412			
	32	313.367	32.2005	36.4206	37.5233			
	37	152.914	29.306	34.9797	35.9801			

The Combination 2, in the other way, tries to get the balance between reducing the SAD calculations and BD-rate performance. It cuts off the PEPZS calculation if the PU size is smaller 8×8. The result is shown in Table 3-22, which is a little bit better than Combination 1.

Table 3-22: Result of tested combination 2.

Combination 2		**amvp_64×64+amvp_pepzs_under16×16 8×8 cut off pepzs**				**BDrate**		
	QP	kbps	Y psnr	U psnr	V psnr	Y	U	V
RaceHorses	22	5853.14	40.0816	41.6377	42.9605	3.10%	3.10%	3.90%
	27	2337.44	36.3224	38.9604	40.5221			
	32	1038.32	33.0876	37.0381	38.6949			
	37	483.408	30.1545	35.7019	37.388			
RaceHorses	22	1370.32	40.0569	41.4147	42.455	2.30%	2.10%	2.70%
	27	656.953	35.9102	38.5077	39.6552			
	32	311.706	32.2306	36.4583	37.5614			
	37	152.117	29.3197	34.9968	36.0259			

The Combination 3 is based on the Combination 2 and is targeted to improve the BD-rate performance. It adds one more AMVP for CU_32×32 in the IME, the AMVP in Combination 3 now are AMVP_64×64, AMVP_32×32, and AMVP_under_16×16, which is also viewed as skip the non-square AMVP if the PU size is larger than 16×16. With one more AMVP, the BD-rate performs about 0.4% to 0.6% better, as shown in Table 3-23.

Table 3-23: Result of tested combination 3.

Combination 3		**amvp_64×64+32×32+under16×16 pepzs_16×16~8×8**				**BDrate**		
	QP	kbps	Y psnr	U psnr	V psnr	Y	U	V
RaceHorses	22	5844.38	40.0807	41.6372	42.9601	2.50%	2.80%	3.70%
	27	2330.64	36.3274	38.9604	40.5174			
	32	1033.79	33.1003	37.0287	38.6958			
	37	479.079	30.1745	35.7103	37.3734			
RaceHorses	22	1370.15	40.0697	41.4221	42.4593	1.90%	2.00%	3.00%
	27	655.322	35.9086	38.4947	39.6426			
	32	310.88	32.2454	36.4603	37.5382			
	37	151.412	29.3324	35.0007	36.0151			

Targeted for HDTV application, our algorithm needs be further modified to get a better BD-rate performance than the Combination 3. The Combination 4 adds part of PEPZS for the PU size is smaller than 8×8. If the PU size is 8×8, it will have 3 steps PEPZS calculation with the searching range by +/- 4. The BD-rate performances of Y, U, and V are now all under 3.0%, as shown in Table 3-24, and for Class D RaceHorses, it could be about 1.5% for Y-component, which is much better than the Combination 1, 2, and 3.

Table 3-24: Result of tested combination 4.

Combination 4		**amvp_64×64+32×32+<=16×16 pepzs_16×16~8×8 16×16_SR16+8×8_SR4**				**BDrate**		
	QP	kbps	Y psnr	U psnr	V psnr	Y	U	V
RaceHorses	22	5840.6	40.0794	41.6383	42.9657	2.50%	2.40%	3.00%
	27	2328.48	36.3223	38.9677	40.528			
	32	1032.2	33.0943	37.0388	38.7151			
	37	480.134	30.1744	35.7191	37.3948			
RaceHorses	22	1366.82	40.0642	41.4066	42.4396	1.50%	1.90%	3.00%
	27	653.495	35.9148	38.5	39.6446			
	32	309.956	32.2452	36.4439	37.5333			
	37	151.334	29.3448	34.9888	35.9594			

From the above combinations, it is obvious that AVMP and PEPZS are as important as each other. While finding the balance between the computational complexity and bandwidth, it is also important to find the balance number of AMVP and PEPZS calculation. Each AMVP and PEPZS for PU should be chosen carefully in IME.

3.5.2 Bandwidth and complexity reduction

The Combination 4 is designed to reduce the high computational complexity and memory bandwidth that the original IME needs. The Table 3-25 shows the memory bandwidth cost comparison between different IME supported by the original HM and our modification, Combination 4. We can see the memory access amount for combination 4 is much smaller than the original HM needs but it still keeps a good video compression performance. What is more, it needs at most 20 search points per CU, which has much lower complexity compared to other search method. In the Table 3-26, we can find the memory access amount is reduced by 60.6% and the search point number is decreased by 78.1% compared with HM which adopts original AMVP and diamond EPZS.

Table 3-25: Memory bandwidth and search point number comparison. CU size is 64×64.

	Original AMVP Full Search	**Original AMVP EPZS Square**	**Original AMVP EPZS Diamond**	**Combination 4**
Memory Access/CU	375 Kbytes	About 218 Kbytes	About 188 Kbytes	About 74 Kbytes
Memory loaded for	MVP blocks + (W+2SR)(H+2SR)	MVP blocks + SP blocks	MVP blocks + SP blocks	Reduced MVP blocks + Reduced SP blocks
SP #/ CU_16×16	16384	57	57	20

Table 3-26: Memory access and complexity comparison per CU_64×64.

	Original AMVP EPZS Diamond	**Proposed (Combination 4)**	**Saving %**
Memory Access	About 188 Kbytes	About 74 Kbytes	60.6%
SP # for AMVP	850	810	4.7%
SP # for search	23800	4584	80.7%
Total SP#	24650	5394	78.1%

3.5.3 Final decision of fast IME algorithm

The Combination 4 in section 3.5.1 has the good BD-rate performance and a large amount of decreased computational complexity and memory bandwidth usage. It should be a hardware friendly motion estimation algorithm and is also the final fast algorithm for our IME architecture design. The IME fast algorithm we proposed is shown as Figure 3-11. The encoding flow for one CU_64×64 starts from PU_64×64. It will first do the AMVP of PU_64×64 and PU_32×32. Then, the flow goes into further depth and does AMVP for PU_16×16, PU_16×8, and PU_8×16. It is followed by the corresponding PEPZS with search range +/- 16. After finishing AMVP and PEPZS at this depth, it continues to do AMVP for PU_8×8, PU_8×4, and PU_4×8. At this depth, the encoder will only do the PEPZS for PU_8×8 with search range +/-8.

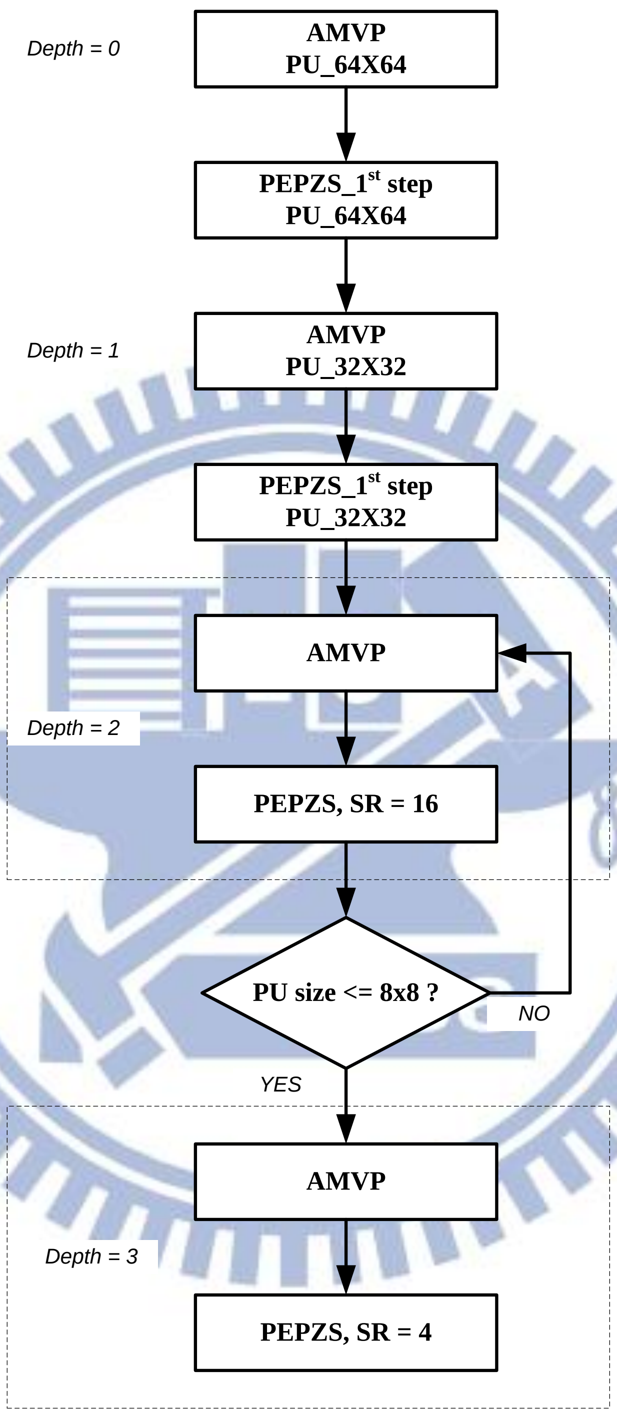


Figure 3-11: Proposed fast IME algorithm.

Chapter 4. Hardware Architecture Design

4.1 Design Specification

The desired system specification is described as follows: an HEVC encoder works under 270 MHz operating frequency with frame size 4kx2k (3840x2160) and frame rate 60fps. According to the specification mentioned above, we can get the needed time as calculated below:

$$3840 \times 2160 \div (64 \times 64) = 2025\ LCUs \quad \text{(Equation 4-1)}$$

With the frame rate 60 fps and the operating frequency 270 MHz, the cycles for encoding a LCU sized 64×64 will be:

$$270M\ Hz \div (2025 \times 60) = 2222\ cycles \quad \text{(Equation 4-2)}$$

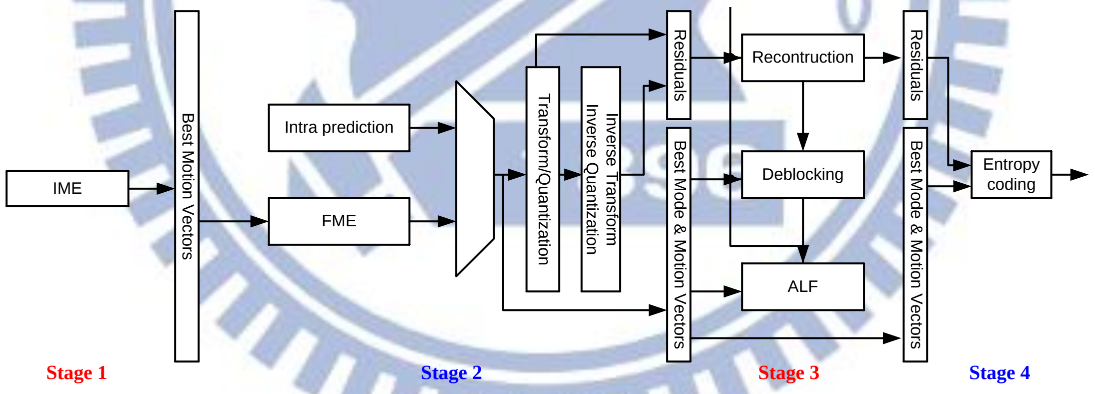


Figure 4-1: The pipelined architecture of HEVC encoder.

In order to make the hardware work as efficiently as possible, we need to figure out how many search points should be calculated at one cycle. From the algorithm proposed in Chapter 3, we can get the total pixels that needed to be computed for the SAD calculation, which will dominate the processing element (PE) number. There are total 425984 pixels needed to be calculated during IME for one CU_64×64.

$$64 \times 64 \times 8(PU\ sizes) \times 2 + 64 \times 64 \times 8 \times 2(PU_{64\times64}, PU_{32\times32}) + 64\ \times$$

$$64 \times 20(search\,point) \times 4(PU\ sizes) = 425984\,pixels$$

(Equation 4-3)

Since the overall calculation should be finished in 2222 cycles, we can find that the hardware should at least get the 192 pixels SAD results per cycle. In short, if the hardware calculates SAD every cycle, it only needs a SAD PE for 192 pixels, which could be covered by a PE sized 16×16.

$$425984 \div 2222\,cycles\ = 192\,\frac{pixels}{cycle}$$

(Equation 4-4)

To consider the time for the data loading in the hardware, we need to get more SAD results per cycle. Because the data dependency caused by the algorithm, the SAD calculation will be followed by data loading, which means that it needs a larger size PE. To solve the problem of data dependency, the data loading and SAD calculation will not be in the same cycle, it will need at least 2 cycles for data loading and SAD calculation. Based on the PE size from (Equation 4-4, our hardware should have a twice larger sized PE. The hardware will have two PE_16×16 for the overall SAD calculation.

4.2 Search Scheduling and Architecture

4.2.1 Overview of IME architecture design

From the section 3.1 we can see there are many block matching algorithms (BMAs) have been proposed. And numerous VLSI architectures have been introduced for motion estimation, such as typical works in [12] and [13]. In [12], Chao proposed an architecture that implements a specific set of BMAs for fast FS and DS. In [13], Li presented an architecture with 9 PEs for PMVFAST and EPZS. These architectures support more than one BMA in the hardware design. And as the video resolution becomes higher recently, there are many efficient works for ME architecture design have been developed, shown in [14] - [25].

From the section 4.1, it is necessary to have 2 processing element so that we can encode a CU_64×64 in 2222 cycles. There will be two PE_16×16 in the EPZS module and work simultaneously according to the scheduling. Every reference pixel needed by PEPZS module will be prepared by the Cache and current unit pixels are kept in PEPZS module. The PEPZS module calculates SAD of AMVP and decides the best candidate for motion estimation to start. It also calculates SAD for EPZS from each PU and keeps the results for the motion estimation to decide what the best partition is for a CU_64×64. The address generator is the most important part of this hardware design; it gets the decision from the PEPZS module and searches the cache module to find the required pixels. The generator also asks for data from the off-chip memory if the needed data is not in the cache. The overall architecture is shown as Figure 4-2.

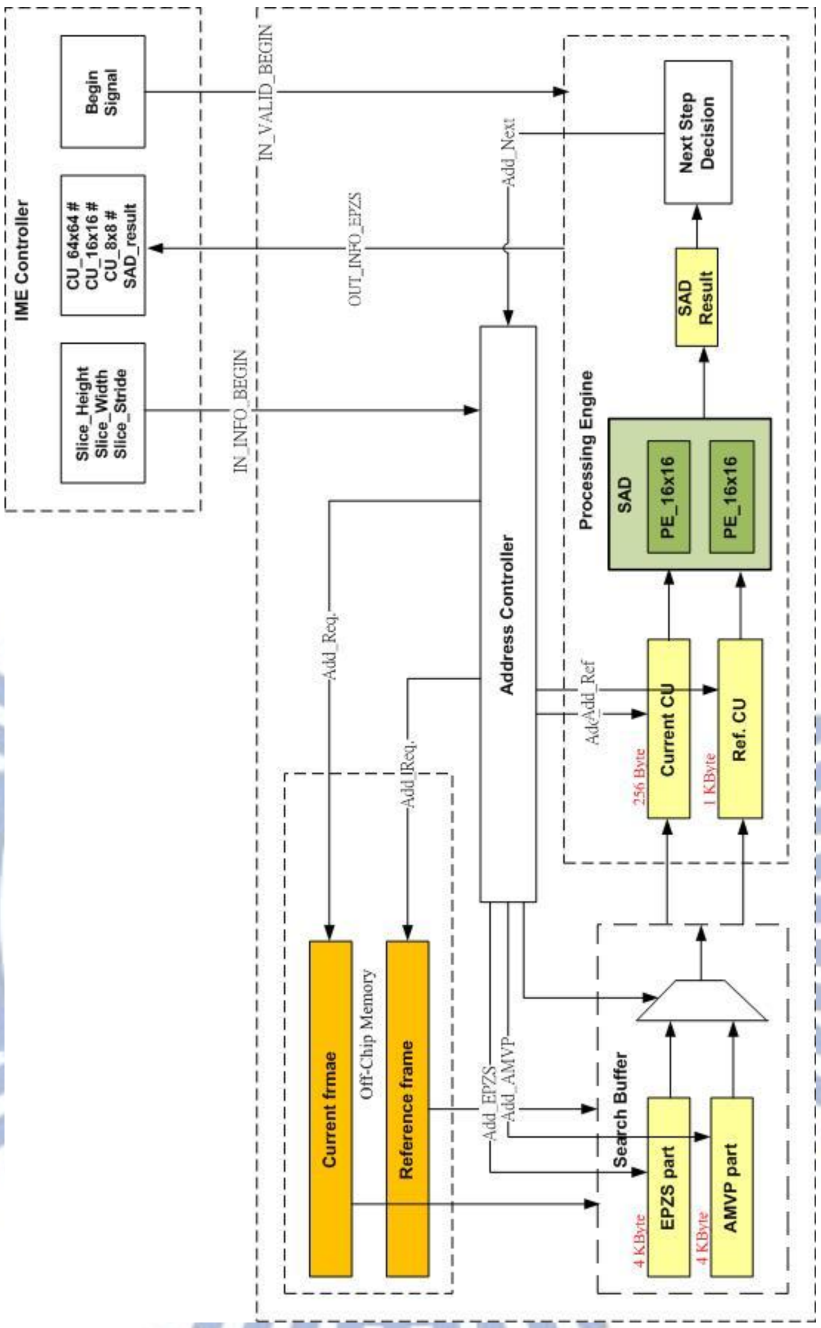


Figure 4-2: Architecture of IME

4.2.2 Searching flow of IME hardware

In the original HEVC flow, the encoding begins at LCU 2N×2N and then the smaller partitions. Because of the dependency of the AMVP calculation, we need to finish the motion estimation from top and left, so that the current PU can get its own AMVP candidates.

In our hardware design, we start from the top-left corner to the bottom-right one. Begin with the first 16×16 unit, process simultaneously with top-left part of CU_32×32 and CU_64×64 and then finally finish all PUs in a CU_64×64. Because the processing element size is 16×16 so the CU size larger than 16×16 (such as CU_64×64 and CU_32×32) would be processed by the flow of CU_16×16, as shown in Figure 4-3. Whenever we do motion estimation for CU_16×16, the AMVP for corresponding blocks of CU_64×64 and CU_32×32 would be finished. When the MVPs of CU_64×64 and CU_32×32 are finally decided, the first step of PEPZS would be calculated and then find the best partition size of IME.

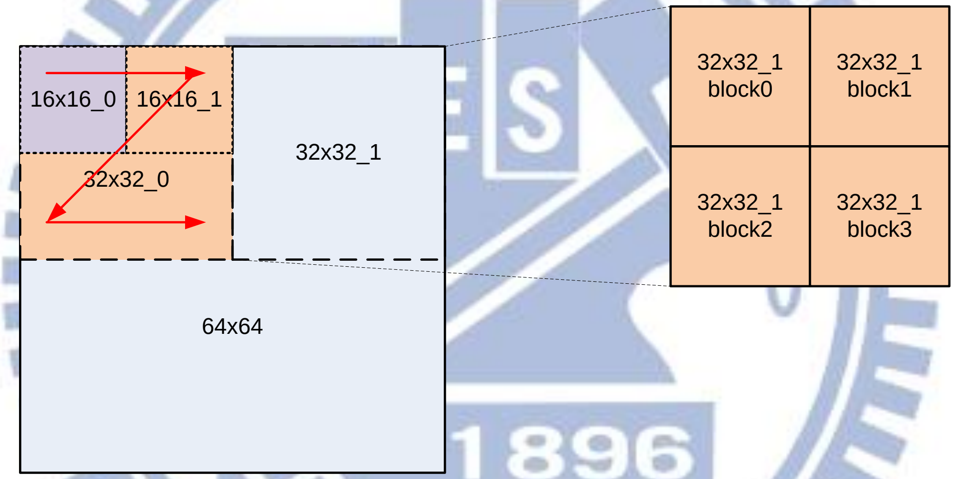
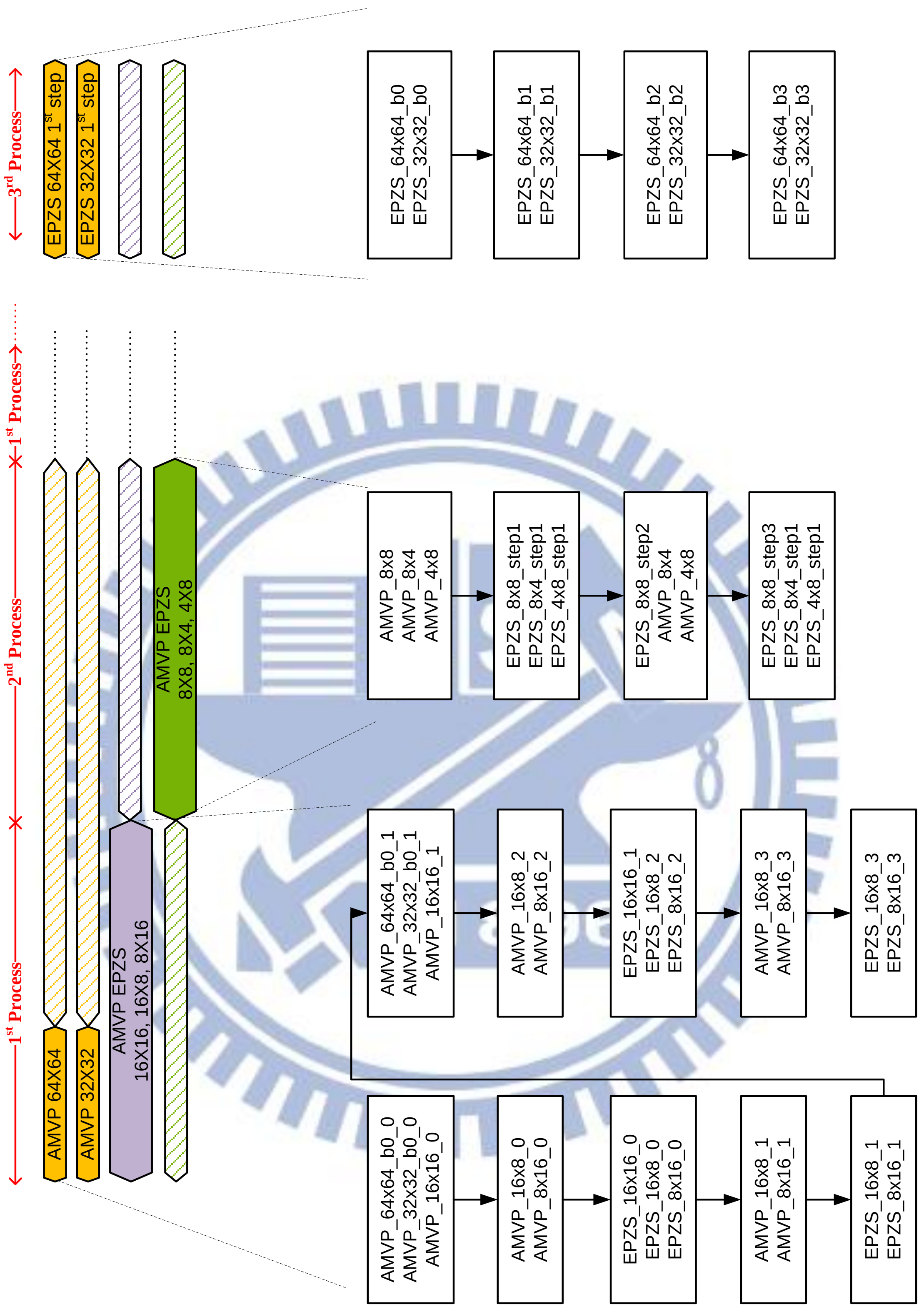


Figure 4-3: Search flow in one CU_32×32 and in CU_64×64.

The CU_64×64 has 16 sets of 16×16 blocks, and CU_32×32 has 4 sets. According to the scheduling, AMVP of CU_64×64 and CU_32×32 would be finished after all CU_16×16s finish their motion estimation. After motion estimation of smaller PUs are finished, the 8 searching points SAD of first step PEPZS for CU_64×64 and CU_32×32 will start. The overall flow for each sized PU is as shown in Figure 4-4.



3rd Process
1st Process
2nd Process
1st Process
EPZS 64X64 1st step
EPZS 32X32 1st step
AMVP 64X64
AMVP 32X32
AMVP EPZS 16X16, 16X8, 8X16
AMVP EPZS 8X8, 8X4, 4X8
EPZS_64x64_b0 EPZS_32x32_b0
EPZS_64x64_b1 EPZS_32x32_b1
EPZS_64x64_b2 EPZS_32x32_b2
EPZS_64x64_b3 EPZS_32x32_b3
AMVP_8x8 AMVP_8x4 AMVP_4x8
EPZS_8x8_step1 EPZS_8x4_step1 EPZS_4x8_step1
EPZS_8x8_step2 AMVP_8x4 AMVP_4x8
EPZS_8x8_step3 EPZS_8x4_step1 EPZS_4x8_step1
AMVP_64x64_b0_1 AMVP_32x32_b0_1 AMVP_16x16_1
AMVP_16x8_2 AMVP_8x16_2
EPZS_16x16_1 EPZS_16x8_2 EPZS_8x16_2
AMVP_16x8_3 AMVP_8x16_3
EPZS_16x8_3 EPZS_8x16_3
AMVP_64x64_b0_0 AMVP_32x32_b0_0 AMVP_16x16_0
AMVP_16x8_0 AMVP_8x16_0
EPZS_16x16_0 EPZS_16x8_0 EPZS_8x16_0
AMVP_16x8_1 AMVP_8x16_1
EPZS_16x8_1 EPZS_8x16_1

Figure 4-4 Flow of hardware encoder process

4.2.3 Interlaced Scheduling of IME hardware

The target encoding time for each CU_64×64 is 2222 cycles. For this limited time, we pipeline the loading data and SAD calculation. Based on the flow in section 4.2.2 and Figure 4-4, we can have an interlaced scheduling shown as Figure 4-5 and we will schedule our hardware work by process separately.

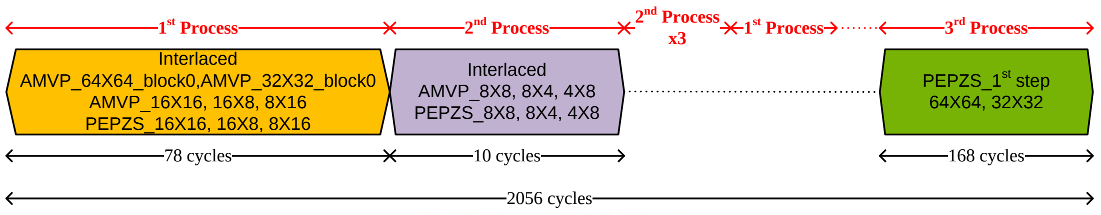

1st Process
2nd Process
2nd Process x3
1st Process
3rd Process
Interlaced AMVP_64X64_block0,AMVP_32X32_block0 AMVP_16X16, 16X8, 8X16 PEPZS_16X16, 16X8, 8X16
Interlaced AMVP_8X8, 8X4, 4X8 PEPZS_8X8, 8X4, 4X8
PEPZS_1st step 64X64, 32X32
78 cycles
10 cycles
168 cycles
2056 cycles

Figure 4-5: Overall encoding interlaced scheduling for proposed fast IME algorithm.

Because of the dependency, sometimes the SAD of searching points needed be calculated first and then the system can get the desired data to do the next step. This is very important in first and second process. For this reason, we process different sizes PU simultaneously, such as PU_16×16, PU_16×8, and PU_8×16 at the same time, when processing elements are working on the PU_16×16, the system can load the desired data for PU_16×8 and PU_8×16, as shown in Figure 4-6.

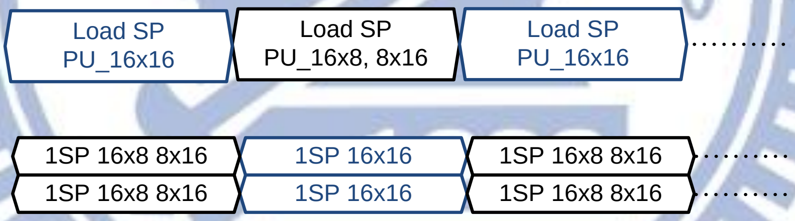
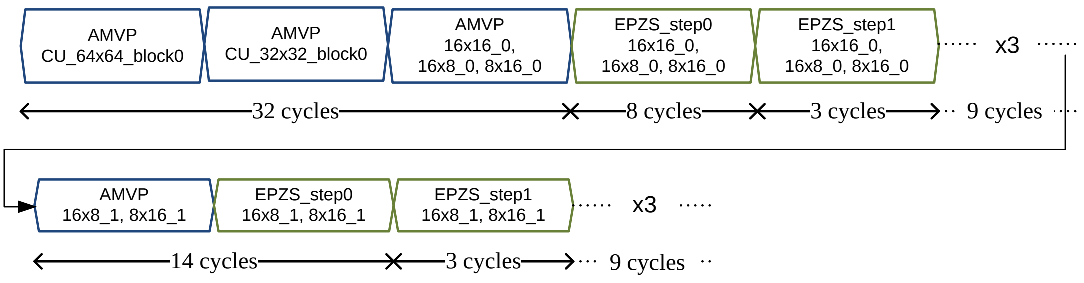


Figure 4-6: Interlace loading data and calculating SAD for PU_16×16, PU_16×8, and PU_8×16.

The system loads data from cache by 64 pixels/cycles, so it needs 4 cycles to load a PU_16×16 candidate, 256 bytes, for AMVP. As a result, the loading time for AMVP in the first process is 32 cycles for each PU_16×16. And because there are two process units 16×16, the SAD calculation time for PEPZS step 0 with 8 searching points is 8 cycles and for the rest steps is 3 cycles with 3 different searching direction points.



AMVP CU_64x64_block0
AMVP CU_32x32_block0
AMVP 16x16_0, 16x8_0, 8x16_0
EPZS_step0 16x16_0, 16x8_0, 8x16_0
EPZS_step1 16x16_0, 16x8_0, 8x16_0
x3
32 cycles
8 cycles
3 cycles
9 cycles
AMVP 16x8_1, 8x16_1
EPZS_step0 16x8_1, 8x16_1
EPZS_step1 16x8_1, 8x16_1
x3
14 cycles
3 cycles
9 cycles

Figure 4-7: The encoding time for 1st process each CU_16×16.

In Figure 4-7, we can find that for the first process, it needs 78 cycles for each CU_16×16. For the overall 1st process, it costs 1248 cycles in total to finish a CU_64×64.

$$(32 + 8 + 3 * 4 + 14 + 3 * 4) \times 16 = 1248\ cycles$$

For the second process, because the CU_8×8 need to do 5 steps PEPZS but the CU_8×4 and CU_4×8 only need to do the first step with 8 different searching point SADs, the encoding time is shorter than first process needs. It costs 10 cycles for CU_8×8, CU_8×4, and CU_4×8 in second process.

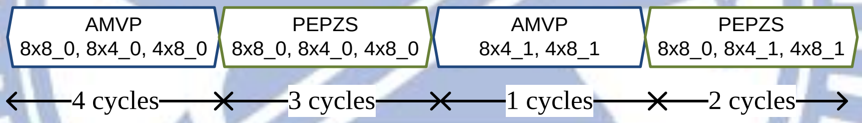


Figure 4-8: The encoding time for 2nd process each CU_8×8

There are total 64 sets of CU_8×8 in a LCU sized 64×64, therefore, the total time for the 2nd process encoding is 640 cycles.

$$10 \times 64 = 640\ cycles$$

The third process only deals with the first step of PEPZS for CU_64×64 and CU_32×32; the encoding time for this process is 168 cycles, as shown in Figure 4-9.

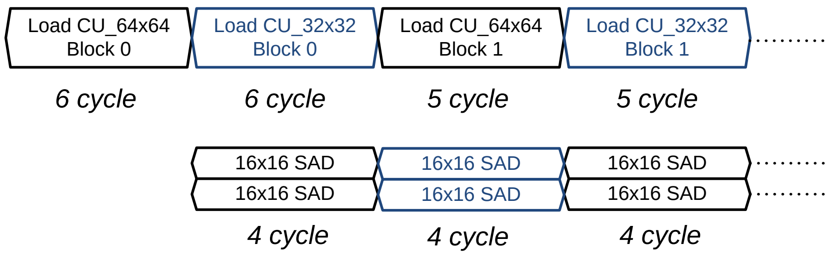


Figure 4-9: The encoding time for 3rd process for CU_64×64.

From the above calculation of encoding cycles for each process, the overall encoding time for this interlaced scheduling design is 2056 cycles for each CU_64×64. The overall interlaced scheduling for those 3 different processes in the hardware is shown as Figure 4-5.This is faster than the one without interlaced idea, shown as Figure 4-10. If the hardware follows the scheduling shown in Figure 4-10, it needs 130 cycles for 1st process, shown as Figure 4-11. The comparison is shown as Table 4-1.

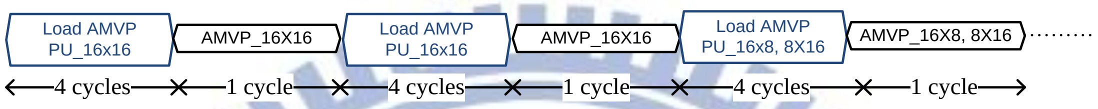
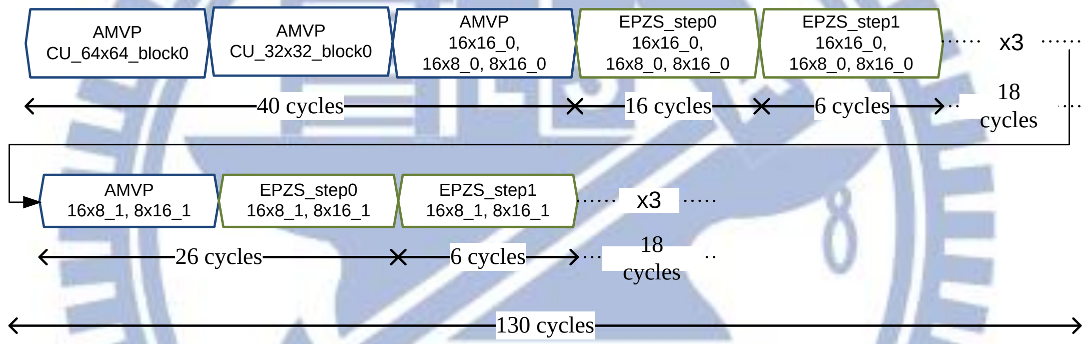


Figure 4-10: Non-interlaced scheduling for AMVP_16×16, 16×8, and 8×16.



AMVP CU_64x64_block0
AMVP CU_32x32_block0
AMVP 16x16_0, 16x8_0, 8x16_0
EPZS_step0 16x16_0, 16x8_0, 8x16_0
EPZS_step1 16x16_0, 16x8_0, 8x16_0
x3
40 cycles
16 cycles
6 cycles
18 cycles
AMVP 16x8_1, 8x16_1
EPZS_step0 16x8_1, 8x16_1
EPZS_step1 16x8_1, 8x16_1
x3
26 cycles
6 cycles
18 cycles
130 cycles

Figure 4-11: Non-interlaced scheduling for 1st process.

Table 4-1: Working cycles needed for IME for two different scheduling.

	Without interlaced (cycles)	Interlaced (cycles)	Saving
1st Process	130	78	40.0 %
2nd Process	18	10	44.4 %
3rd Process	296	168	43.2 %
Total scheduling	3528	2056	41.7 %

4.3 Architecture of PEPZS

4.3.1 Overview of PEPZS architecture design

The most computational part is the PEPZS part, in our hardware design, it includes processing elements PE_16×16, so it can deal with both the AMVP and PEPZS calculations. The PEPZS module is communicated with the IME controller, address controller, and the cache. The control signal comes from the IME controller and address controller whereas the needed data all comes from the cache.

The IME controller sends crucial information for the calculation, such as begin signal, AMVP signal, PEPZS signal, and PU size signal, into the module. After the SAD calculations, it sends the SAD results and best MVP candidate for AMVP or three searching points for next PEPZS step. The architecture of PEPZS module is shown in Figure 4-12.

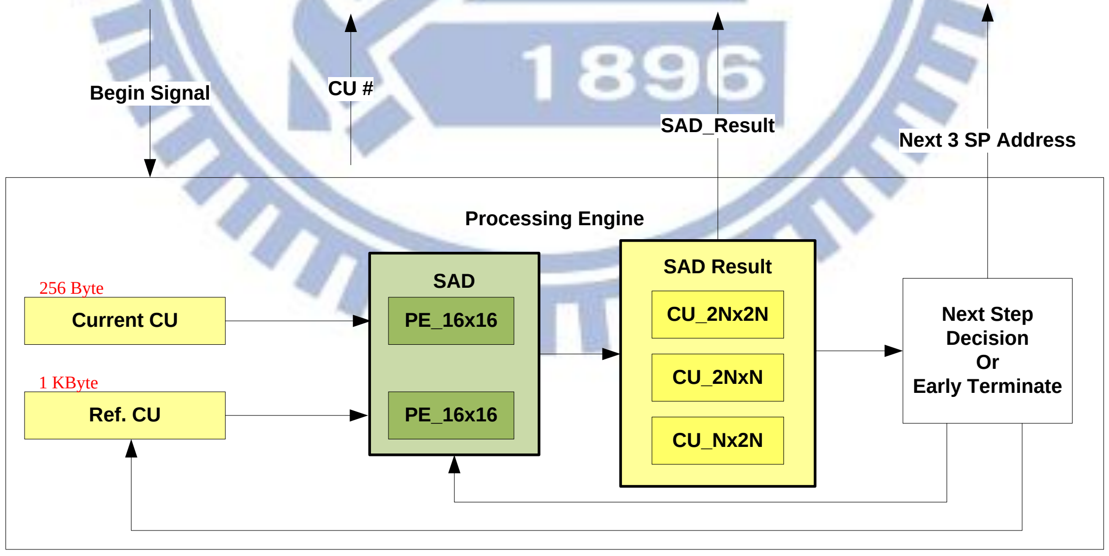


Figure 4-12: Overall architecture of PEPZS module

The PEPZS module keeps the current CU and some reference pixels for the PE_16×16 inside the module, the registers could be updated if the current CU is changed or the new reference pixels come. According to the encoding flow, the decision of best MVP candidate is made after the SAD calculation in AMVP process, what is more, the SAD result will be kept as the (0,0) searching points result for the PEPZS first step in order to reduce the computational number. When the system is processing PEPZS calculation, it will send the next step decision or early termination signal to IME controller, PE_16×16, and data registers to stop the calculation. The early termination will stop the data loading from the cache and also stop asking data from the off-chip memory. When the motion estimation is finished for one PU, the results are kept inside the module and the system will decide the best partition size through these kept SAD results.

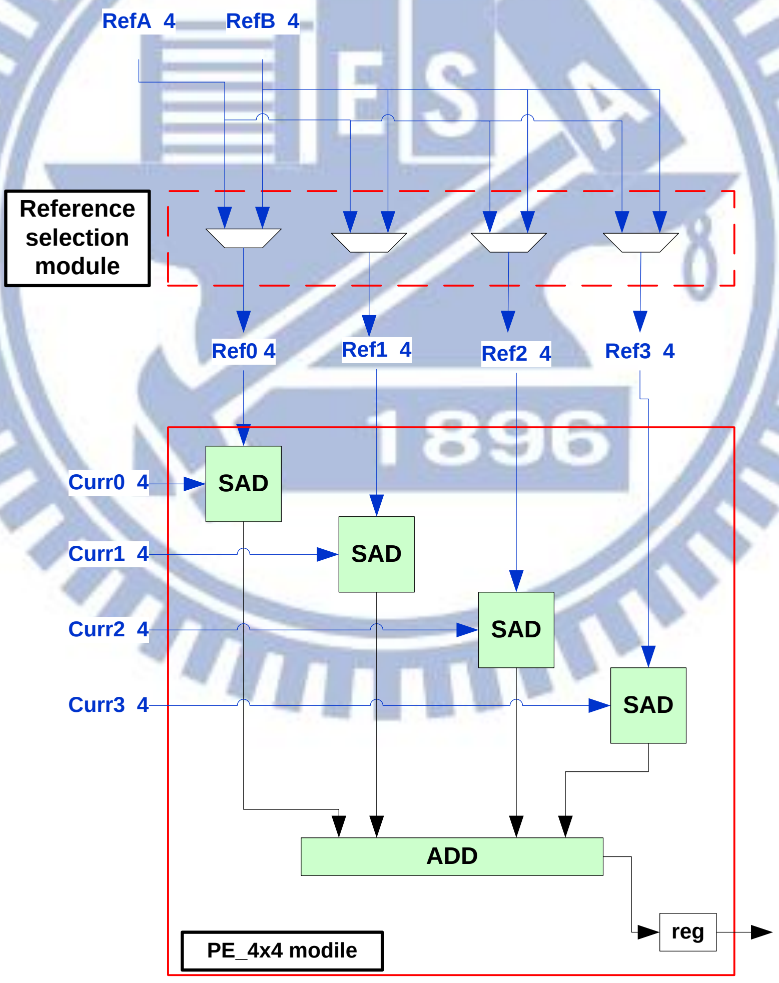


Figure 4-13: PE_4×4

The crucial part in the PEPZS module is the processing element. The processing element we use in our design is sized as 16×16, which can calculate SAD of unit

16×16 . The PE_16×16 is composed by 16 sets of PE_4×4 and the outputs are 16 results of PE_4×4. The system sums these SAD results according to the PU size and uses the summation to make the decision. The PE_16×16 and PE_4×4 can select the reference data by the system desire.

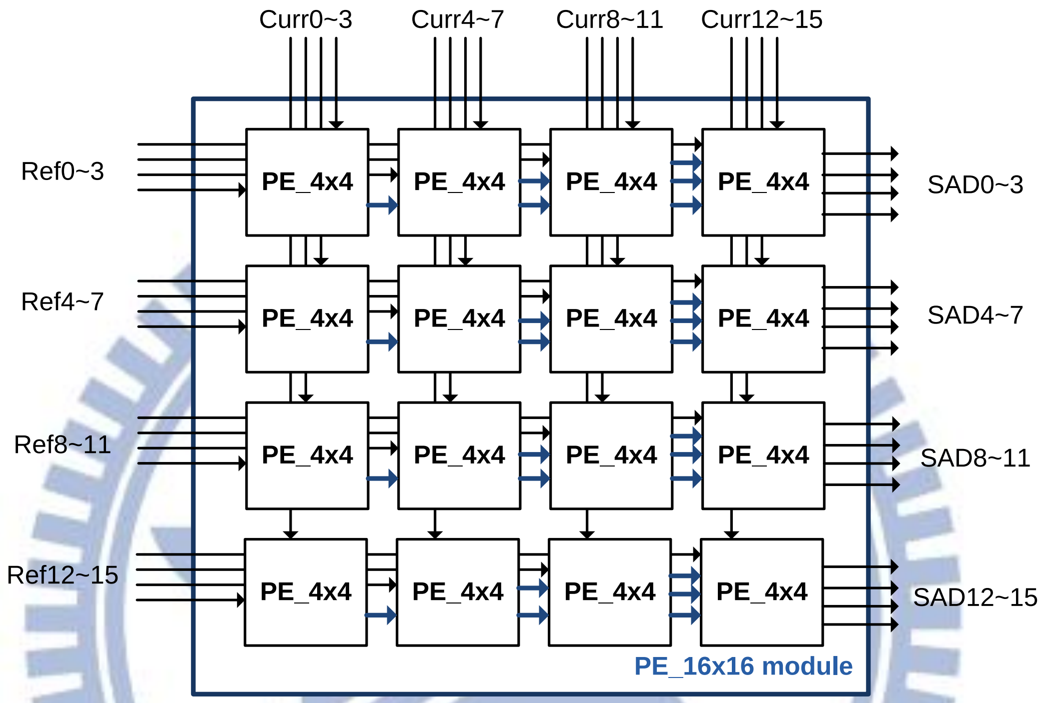


Figure 4-14: PE_16×16

In the PEPZS module, the outputs from PE_16×16 are kept in the register temporally, after the comparison and decision, the results will be updated into the SAD Result register, as shown in Figure 4-15.

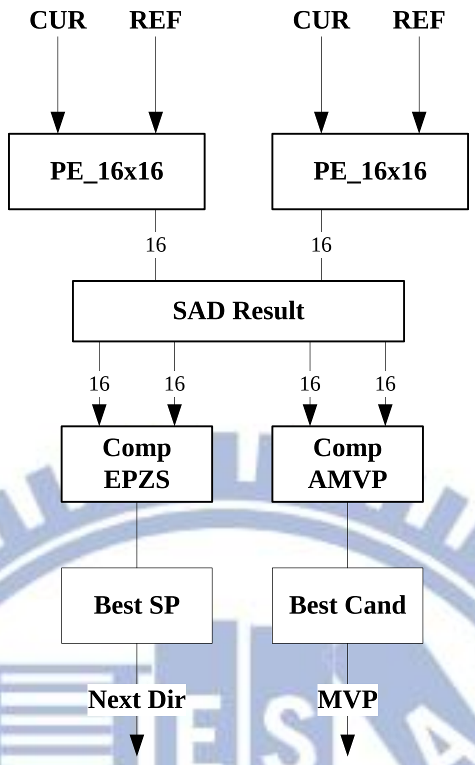


Figure 4-15: PEPZS comparison and updating of SAD results

For the calculation, the PEPZS system controls the PE_16×16 to work or not, as shown in Figure 4-16. According to section 4.2.2, each process stage includes processing of 2N×2N, N×2N, and 2N×N, so the system need to figure out which PU size is under the calculation, and with the PU size information, the system can get the SAD results from the register and do the comparison for AMVP and PEPZS. These comparisons can choose the best MVP candidate, best searching point for next step, or early termination.

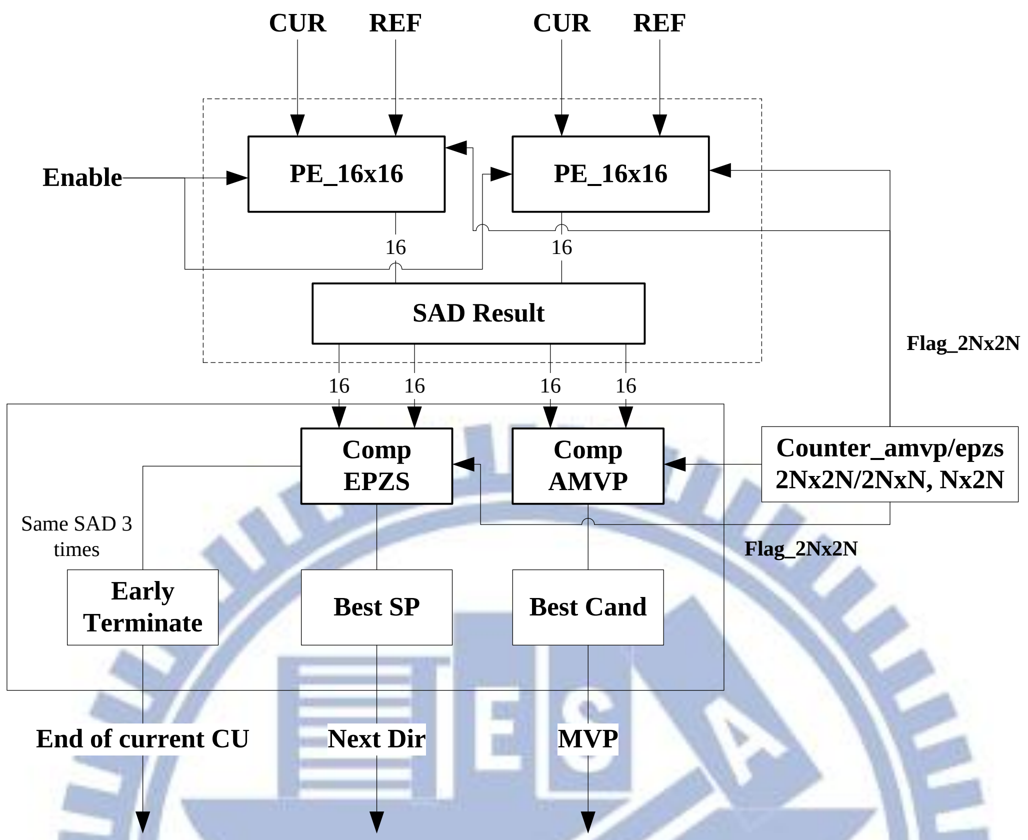


Figure 4-16: PEPZS controller and calculation

4.3.2 PEPZS hardware flow

The calculation flow is following the scheduling in section 4.2. When the IME controller sends the begin signal, the PEPZS system starts the AMVP of PU_2N×2N, PU_N×2N, and PU_2N×N. After finishing AMVP, the PEPZS calculation is in progress then. It can decide whether to do the next searching points or have an early termination. After 5 steps PEPZS or early termination for one PU_2N×2N, the system then continues to do AMVP and PEPZS for the rest PU_N×2N and PU_2N×N. The overall flow is shown as Figure 4-17.

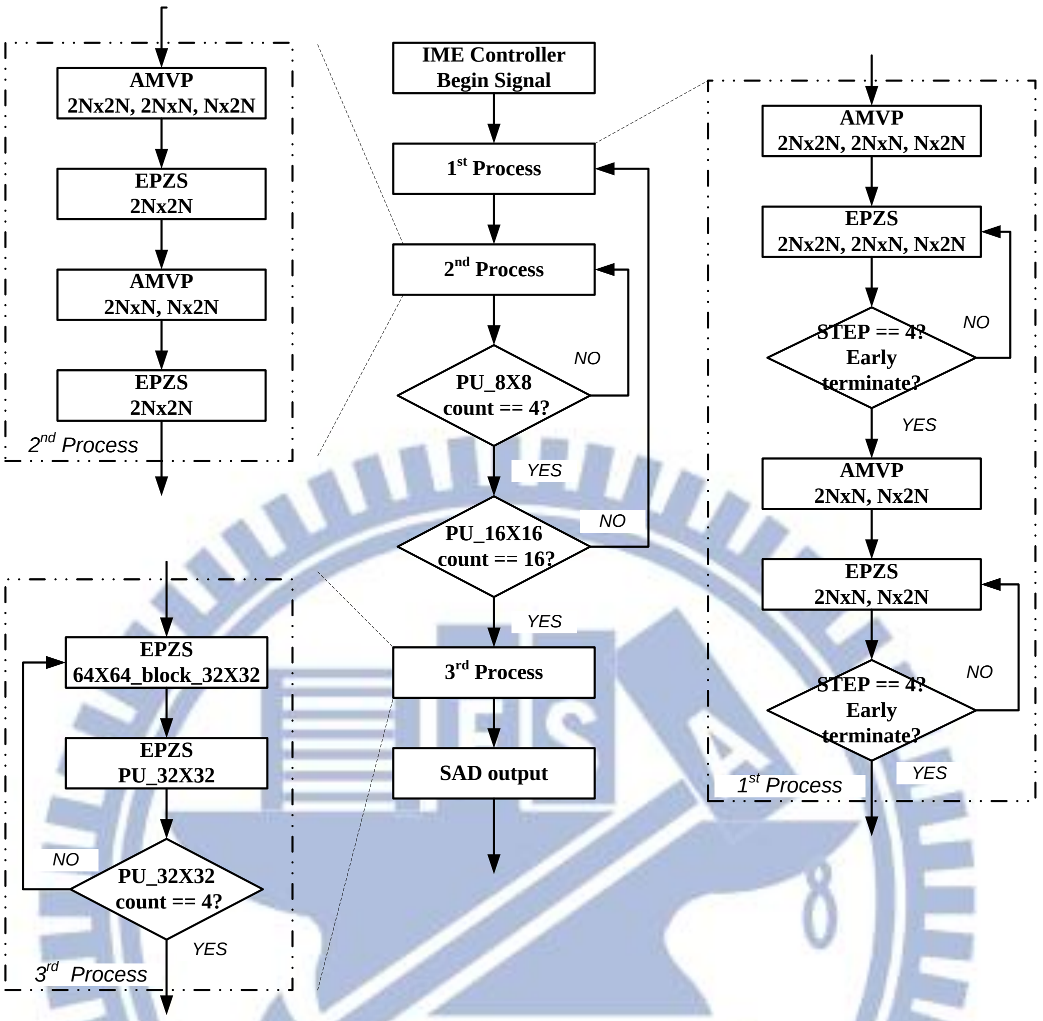


Figure 4-17: PEPZS hardware progress flow

4.4 Cache Based Buffer Design

As same as the huge computational complexity by AMVP and PEPZS, the amount of pixels needed for the SAD calculation is also very large. For those data needed for calculation, the easiest way is loading the pixels whenever the system requests; obviously, this would leads to a large data access bandwidth consumption and costs a long time to load data from the off-chip memory. For the hardware architecture, the data reuse should be an important issue and taken carefully.

Many works are proposed to reduce the memory access. It is efficient to use Level C data reuse algorithm when implementing FS in ME hardware. As for fast ME algorithm, there may be some redundant SR memory access. To get better data reuse rate, many cache-based architectures have been presented [26]-[30]. Some architecture supports multilevel data reuse [26] ; it makes local buffer and cache working simultaneously.

A simple way to do the data reuse is to keep the data in the temporary register by the size according to the PU under operating. If the current PU is 16×16, the pixels would be partitioned into 16 sets of 4×4 blocks and kept in the temporary register, as shown in Figure 4-18, it is very like the cache implementation in [27]. When the system requests the data, it can get the pixels according to its PE size. Most of the previous cache works adopts this kind of word block to store data in cache [26]-[30]. This is good for the hardware design whose motion estimation uses the same MVP for every different sized blocks or units, but not for our proposed motion estimation fast algorithm. Because the MVP for each different PU is not the same, the pixels needed by different PUs may not be the same, even the data could be reused, keeping data in 4×4 blocks is not an easy way for data fetching.

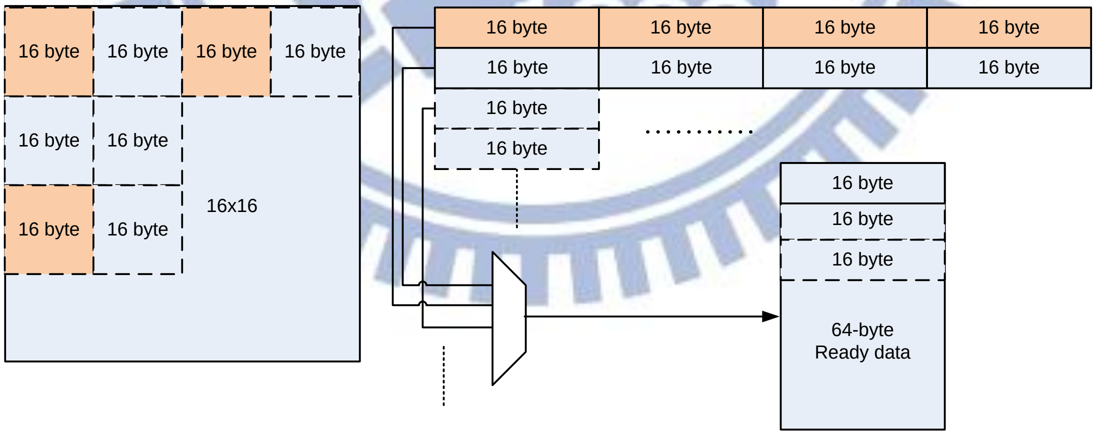
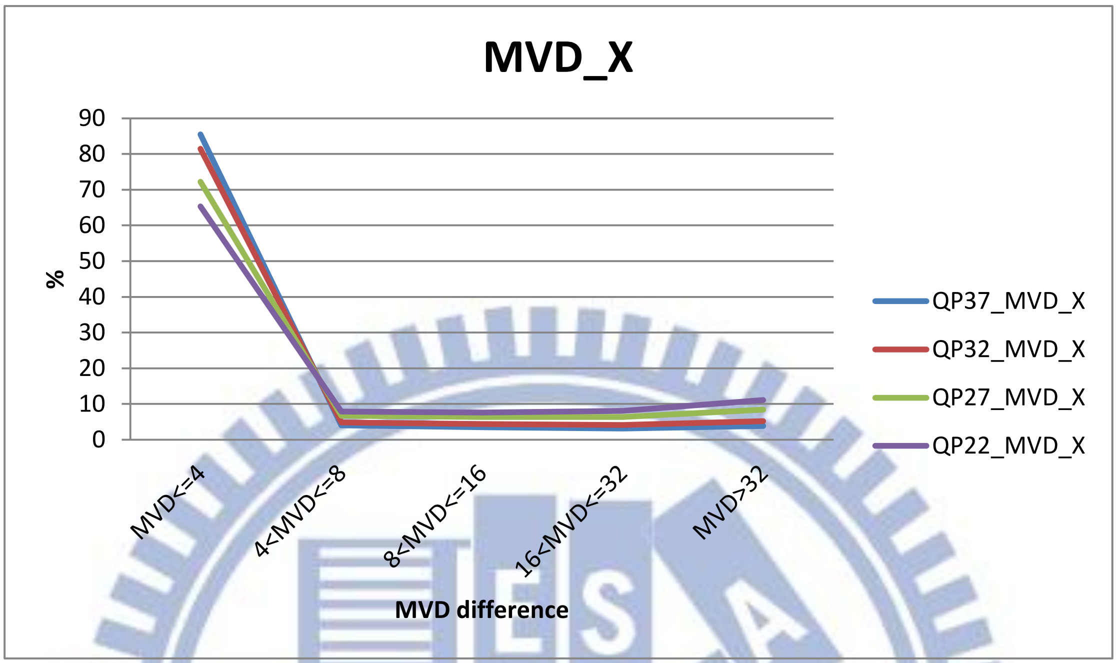


Figure 4-18: Simple data reuse with register

4.4.1 Load all needed blocks into cache based buffer

To design good hardware architecture, we prefer to use the cache to improve the rate of reuse rather than use registers. The first key point to use cache is to figure out which size to load our needed data can benefit most. For the AMVP, we can find the suitable loading size by the MVD value, which is the final MV difference from MVP.



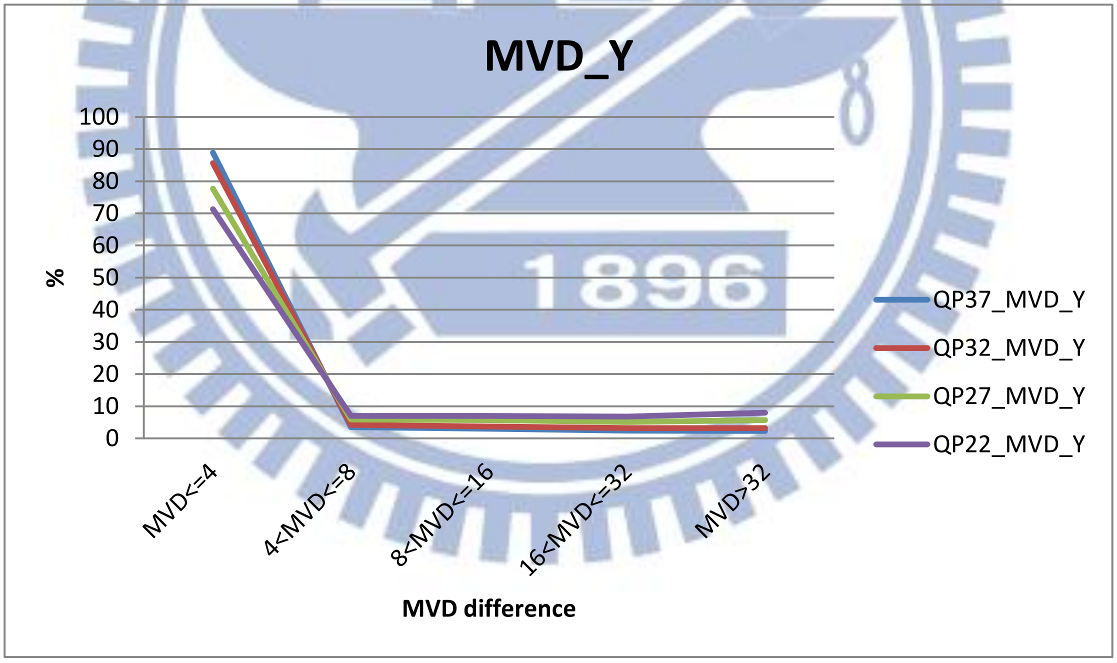


Figure 4-19 MVD distribution of X-axis and Y-axis

From the MVD distribution analysis, as shown in Figure 4-19, most MVD is within +/- 4, which means that for different PU, the MVP candidates have high probability be overlapped with others within distance +/- 4. With the high probability, it is appropriate to load a bigger block. What is more, PEPZS will do at least 3 searching steps in our algorithm, so if we preload a bigger block in AMVP calculation, the data could also be used for 3-step search in PEPZS. For example, if the current PU is 16×16, the system will load a 24×24 block per MVP candidate into the cache buffer for both the AMPV and PEPZS calculation, as shown in Figure 4-20.

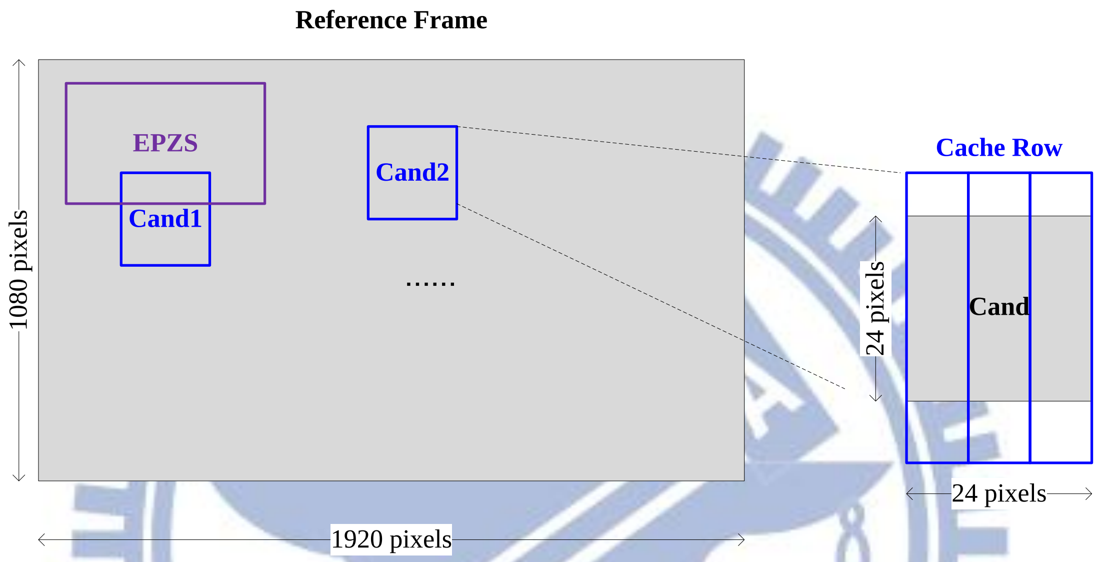


Figure 4-20 The Cache loaded blocks. It will load two candidates blocks and PEPZS block separately.

To keep the data in the cache based buffer, the system employs a cache design with a number of memory rows. Each memory row consists of eight words, which is 64 bits and contains 8 pixels data, as shown in Figure 4-21. However, this storage method does not perform well if it does not have a good searching and comparing function and updating mechanism. The following sub-sections will discuss some cache mechanisms and architectures.

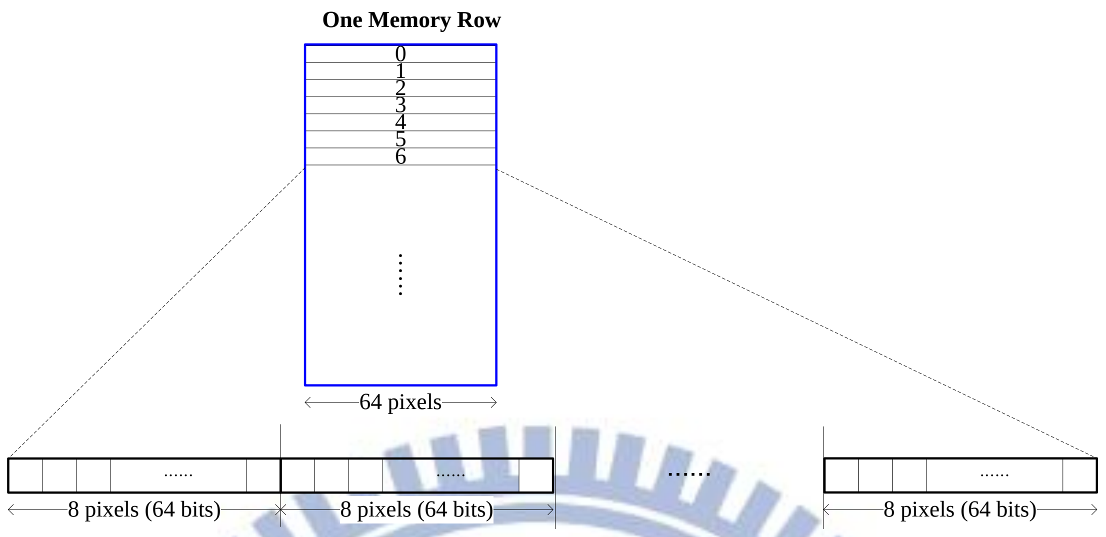


Figure 4-21 One memory row contains 8 words, each word is 8 bytes.

4.4.2 Fully associative cache based buffer

For the cache based buffer design, we first try the fully associative cache idea. When the buffer is empty, the data would be loaded in directly. Every time the system checks the buffer to see if the data needed is valid or not, when there is a miss, the cache will be updated.

For this design, the updating mechanism is loading data inside when the buffer still has empty space for new data. The new data will be added next to the data if they share the same Y position in a frame, otherwise, it will be added randomly if there is an empty space. When the cache is full, the data which is not be used recently will be replaced with the new data. The mechanism is as shown in Figure 4-22.

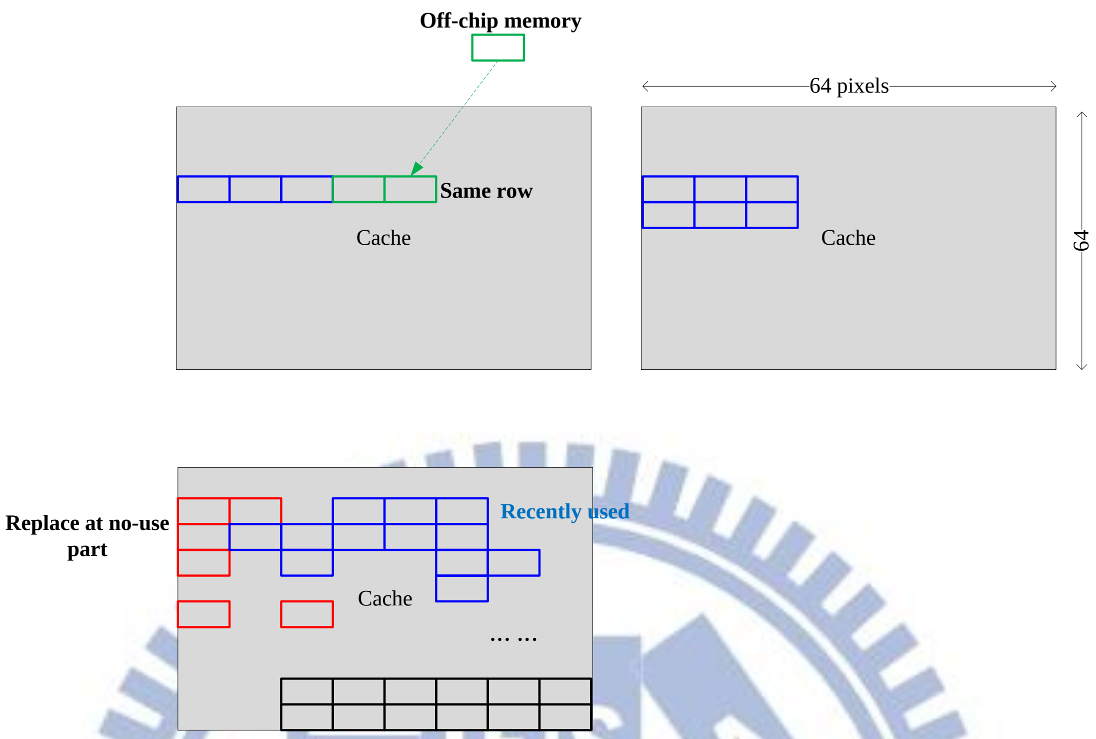


Figure 4-22: Fully associative cache updating mechanism.

4.4.3 Cache based buffer size analysis

The cache based buffer performance depends on the hit rate and miss rate, and these depend on the updating mechanism and cache based buffer size. The size could affect the hit rate a lot, larger size makes the hit rate could be higher. With the fully associative cache method mentioned in 4.4.2, we try some different buffer sizes to find an appropriate one for our cache based buffer design.

The sizes tested are 24×64 bytes to 96×64 bytes. The determinants of cache based buffer size are the value of maximum reloaded byte, average reloaded byte, and hit rate. The maximum value is the number of reloaded bytes per CU_64×64, the average one is the average number of reloaded bytes per CU_64×64 in a test sequence, and the hit rate is percentage of finding data successfully in the cache based buffer.

As shown in Figure 4-23, the maximum value is very large when the size is too small, such as 24×64 bytes, and it becomes lower when the size is bigger than 64×64 bytes. The hit rate behavior is, however, contrast to the max reloaded value. The average reloaded value, however, keeps almost the same for every cache based buffer size. From the results of maximum value, average value, and hit rate, it is obvious that the maximum value is much more related to the size. The maximum reloaded value is what we want to reduce by the cache mechanism. Between the size 64×64 and 96×64, concerning the cache size and the cache performance, the cache based buffer size 64×64 is a better for our design.

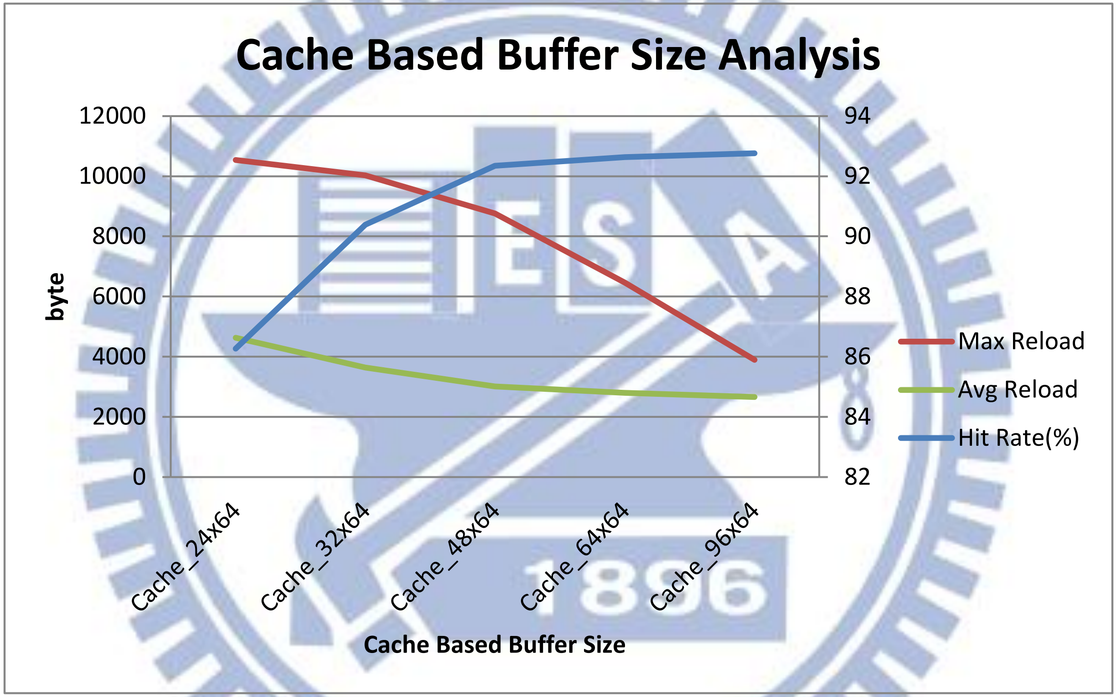


Figure 4-23 Cache based buffer size relationship with the maximum and average reload byte and hit rate.

4.4.4 N-way associative cache based buffer

Because we want to improve the cache performance, we try N-way associative cache mechanism to replace fully-associative cache. There will be two cases, 4-way and 8-way, and different cache based buffer sizes to find the best N-way associative cache based buffer and corresponding size.

When starting to encode a new CU_64×64, the buffer could be refreshed or not, and this decision will results in different data loading value. The first case shows the results that system loads (W+8)×(H+8) bytes in buffer per MVP candidate and refreshes whenever another CU_64×64 starts to encode. As shown in Table 4-2, the 8-way associative cache based buffer both performs better in the hit rate and maximum reloaded value.

Table 4-2: N-way cache based buffer, Load 24x24 each MVP. Refresh buffer if a new CU_64×64 starts

4-way	**Hit Rate (%)**	**Max load (byte)**	**Average load (byte)**
64×64	87.8233	12176	6957.214352
32×64	87.6364	12256	6979.570924
24x64	87.1571	12560	7182.692694

8-way	**Hit Rate (%)**	**Max load (byte)**	**Average load (byte)**
64×64	94.5497	9048	5545.914110
32×64	93.5684	10888	6855.1733080
24x64	87.153	11536	7300.229782

The second case, the cache based buffer will not refresh when the next CU_64×64 starts to encode. From the Table 4-3, 8-way associative cache based buffer is still better than 4-way one, and the cache buffer in second case performs better than in first case slightly.

Table 4-3: N-way cache based buffer, Load 24x24 each MVP. Do not refresh buffer if a new CU_64×64 starts

4-way	**Hit Rate (%)**	**Max load (byte)**	**Average load (byte)**
64×64	87.8896	13120	7021.131065
32×64	87.701	13224	7233.610708
24x64	83.1832	13328	7471.692136

8-way	Hit Rate (%)	Max load (byte)	Average load (byte)
64×64	95.1466	8832	6322.919502
32×64	93.8596	9264	7027.929355
24x64	87.2431	9640	7325.913367

Because the maximum value from the above two cases is still a little bit high for the hardware design, the system needs to try different load mechanism for the cache based buffer. In third case, the system loads W×H bytes in cache buffer per MVP candidate and loads the rest needed data later when the MVP is decided. And the buffer refreshes whenever the next CU_64×64 starts encoding. As shown in Table 4-4, the performance is better than both first and second cases, the third case is a good for our hardware design.

Table 4-4: N-way cache based buffer, Load 16×16 each MVP. Load rest 320 byte if MVP is found. Refresh buffer if a new CU_64×64 starts.

4-way	Hit Rate (%)	Max load (byte)	Average load (byte)
64×64	88.9422	6656	4385.88664
32×64	88.8228	8768	6307.480201
24x64	86.096	8824	6485.419595

8-way	Hit Rate (%)	Max load (byte)	Average load (byte)
64×64	93.2805	5720	4215.33544
32×64	90.859	7856	6294.738799
24x64	88.3407	9368	6587.254136

The third case performs well than other two cases. With the third cache design, we can compute the reloaded byte for each PU size. There are two N-way associative cache based buffer, 4-way and 8-way, and the PU sizes are 16×16, 16×8, 8×16, 8×8, 8×4, and 4×8. As shown in Table 4-5, the average load and maximum load value are about 2000 to 5000. If we load data into cache for every size PU separately, the overall value of loaded byte will be very high.

Table 4-5: Memory load for each PU separately by 4-way (left) and 8-way (right) cache based buffer.

4-way	Average Load	Max Load
PU_16×16	4385.88664	6656
PU_16×8	2793.71864	5040
PU_8×16	2626.979372	5208
PU_8×8	4361.483739	6208
PU_8×4	2883.073035	4672
PU_4×8	4786.360899	8928

8-way	Average Load	Max Load
PU_16×16	4215.33544	5720
PU_16×8	2683.651366	3960
PU_8×16	2582.611411	4224
PU_8×8	4099.505668	5136
PU_8×4	2712.346404	4016
PU_4×8	4182.862665	5952

From the scheduling, our design will process PU_16×16, PU_16×8, and PU_8×16 simultaneously and encodes PU_8×8, PU_8×4, and PU_4×8 together in other process. This means that the loaded data for PU_16×16 may be shared with PU_16×8 and PU_8×16, and PU_8×8 could share data with PU_8×4 and PU_4×8. As shown in Table 4-6, the value for average load and maximum load are much lower than the sum of Table 4-5. For example, for 8-way CU_16×16, the average load value is 4224.99 bytes, whereas for 8-way PU_16×16, PU_16×8, and PU_8×16, the average load value is 9481.60, which is much higher than the value needed by CU_16×16.

Table 4-6: Memory load for CU_16×16, which covers CU_16×8 and CU_8×16, and for CU_8×8, which covers CU_8×4 and CU_4×8. Left: 4-way. Right: 8-way.

4-way	average load	max load
CU_16×16	4395.963576	6656
CU_8×8	4368.188812	6208

8-way	average load	max load
CU_16×16	4224.987177	5720
CU_8×8	4105.682959	5136

Even with the sharing mechanism, data shared between PU_16×16, PU_16×8, and PU_8×16 and between PU_8×8, PU_8×4, and PU_4×8, the value of data needed to be load is still high, for example, the 8-way cache needs to load about 8331 bytes in average and 10856 bytes if needed. To reduce the high value of loading data, the best way is make the data loaded for CU_16×16 be shared with CU_8×8. From the Table 4-7, almost 97% of MVP candidates for CU_16×16 are the same as candidates for CU_8×8. The high overlap percentage means that CU_16×16 and CU_8×8 has high possibility that they need the same data. As shown in Table 4-7, if the loaded data for CU_16×16 can be shared with CU_8×8, the average loaded value will be about 6500 bytes, which is much higher than 8331 bytes from the previous design, and the maximum value will be 7656 bytes, not 10856 bytes.

Table 4-7: Overall memory loaded.

CU_16×16 MVP and CU_8×8 MVP overlap %
0.967461
Average byte reloaded
6497.6
Maximum byte reloaded
7656

4.5 Architecture of Cache Based Buffer

Since the 8-Way associative cache based buffer with 64×64 bytes has been selected in our design we now need to design the hardware of our cache based buffer. The overall architecture is mentioned in section 4.2, Figure 4-24 shows simply the cache relation with other element in the proposed architecture.

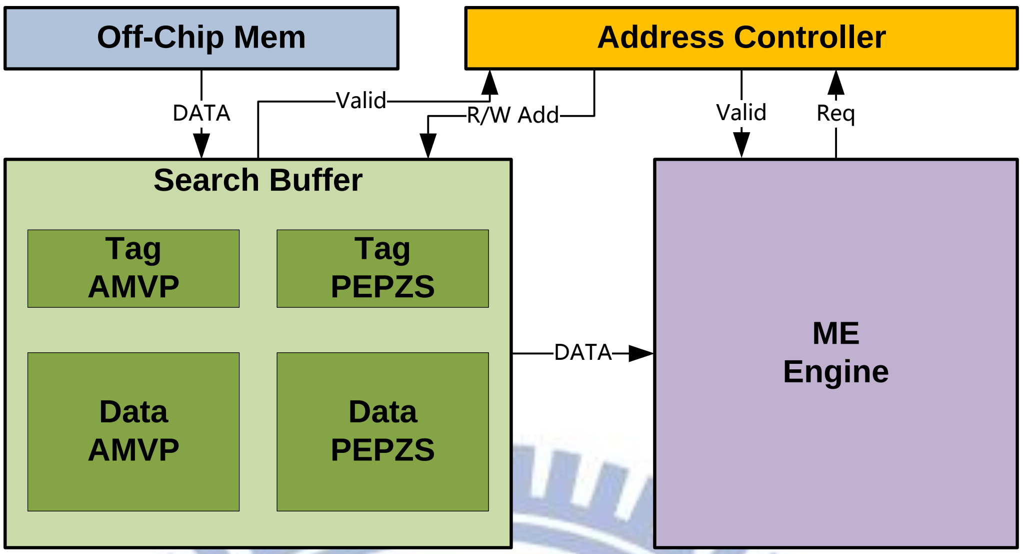


Figure 4-24: Cache module relationship in the overall proposed architecture.

We use two cache based buffer for AMVP and PEPZS separately in our design. The cache mechanism for AMVP part will pre-fetch the first 3-step data for PEPZS while the buffer for PEPZS only load the 4th or 5th step needed data. Since the PEPZS goes beyond the 3rd step, the data for the rest PEPZS step will be very divergent and if the cache buffer loads data for AMVP and PEPZS simultaneously, the hit rate will be decreased. If the cache buffer loads data for AMVP and PEPZS separately, the hit rate will perform better, shows as Table 4-8.

Table 4-8: Hit rate in two different cache based buffer, which are AMVP+PEPZS and PEPZS.

Motion	Class	Test sequence	AMVP Hit rate (%)	AMVP+PEPZS Hit rate (%)	PEPZS Hit rate (%)
High	B	Cactus	93.9885	63.0576	91.5577
		Kimono	94.6786	60.4704	92.1785
	C	BasketballDrill	97.7362	67.5685	95.3984
		RaceHorsesC	91.4237	47.3322	87.8504
	D	RaceHorses	94.9561	58.3367	91.0088
Medium	B	BasketballDrive	96.8091	65.3569	93.7051
	C	BQMall	79.9803	64.7391	85.7357
	D	BasketballPass	97.5183	66.8536	95.3559
Low	B	BQTerrace	96.7301	95.3331	95.4299
		ParkScene	97.4778	75.2201	83.7295

	C	PartyScene	98.4607	86.4885	98.1048
	D	BlowingBubbles	96.1304	94.1446	96.2425
		BQSquare	99.8427	99.1383	99.842
Average hit rate (%)			82.38217	72.61843	92.77994

4.5.1 Overview of cache based buffer

For the N-way cache design, the address controller is the most important core of the cache. The address controller maps the address of a frame and the cache address. It controls the loaded data should be put into which cache position.

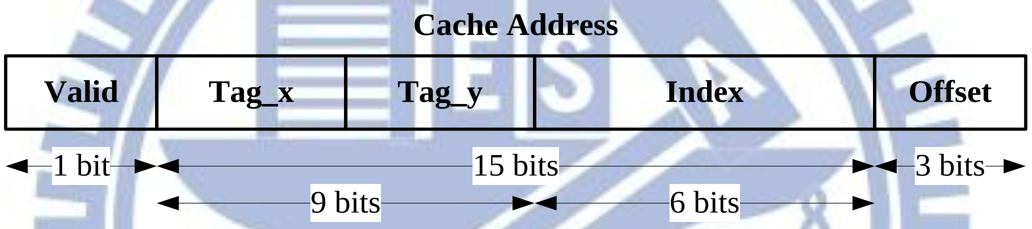


Figure 4-25 Cache address for our hardware design

In our design, the address controller will convert the frames address into the cache address. As shown in Figure 4-25, the cache address can be departed into offset, index, tags, and valid bit. The cache address is mapped from the frame address. We use 9 bits to identify the x-address and y-address in a frame and both address will be mapped with a CU_64×64 starting position into the cache address. The offset in our cache address is 3 bits because each word in the cache is 8 bytes. The index is calculated from the Y-position in a frame, as shown in

(Equation 4-5, it is 6 bits. The tags are calculated from the X-position and Y-position in a frame separately. The valid bit shows 1 if this cache word has valid data, 0 for non-valid data. Every value in a cache address is calculated by the address controller.

$$Y\ address\ in\ a\ frame\%(64 \times 64 \div (8 \times 8)) = Index$$

(Equation 4-5)

4.5.2 Address controller design

The address controller is the most import part of our cache design [27] [31]. As an 8-way cache address controller, it will compare the information with the requested data and then sends signal to the cache or to the off-chip memory.

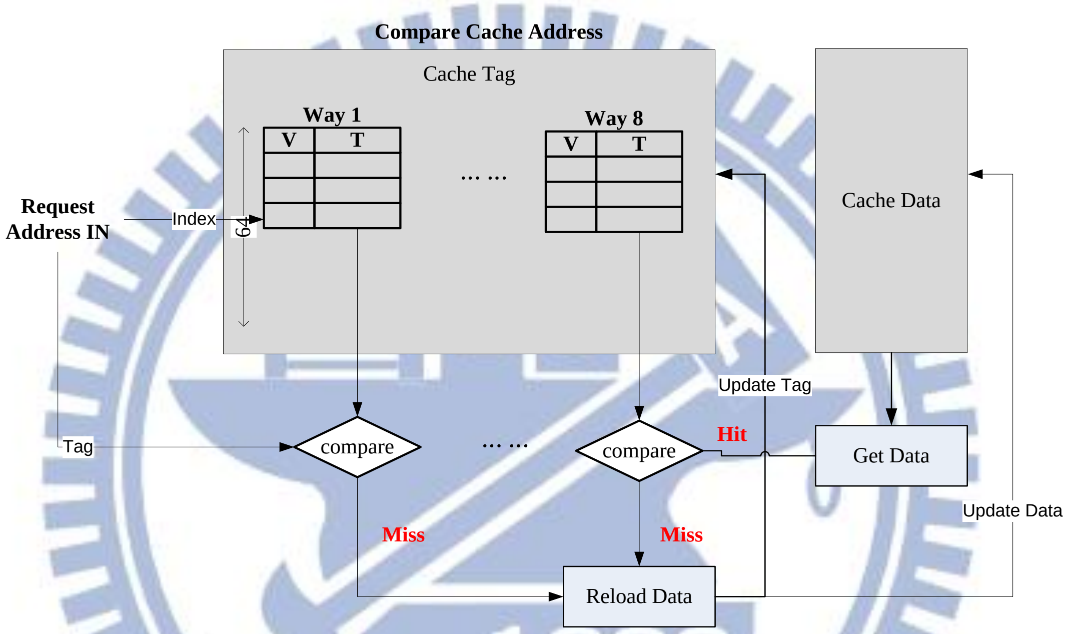


Figure 4-26 Address controller architecture

As shown in Figure 4-26, the address controller compares the requested data address with the index and tags in our cache. It first calculates the index of the requested data and finds the corresponding cache row to find the data. The controller then compare from the first way tag to the eighth way tag to find if the cache word if valid or not, and if the data information is the same. If the comparison result is the same, then the controller will send a hit signal to the cache and the system can get the required data. If the comparison result is not the same, it sends a miss signal to the cache and off-chip memory and updating the cache data.

4.5.3 Cache based buffer architecture design

The index, tags, and valid bit, they will be kept as register in the address controller whereas the data will be kept in 8-way associative cache. To implement an 8-way associative cache, we depart the cache into 8 parts. As shown in Figure 4-27, there will be 8 8×64 bytes register files to build up an 8-way associative cache. With the hit signal from the address controller, the cache can prepare the data requested by the system and send data into the PEPZS module. When the address controller sends miss and updating signal, the cache updates the data with corresponding X and Y address values coming along.

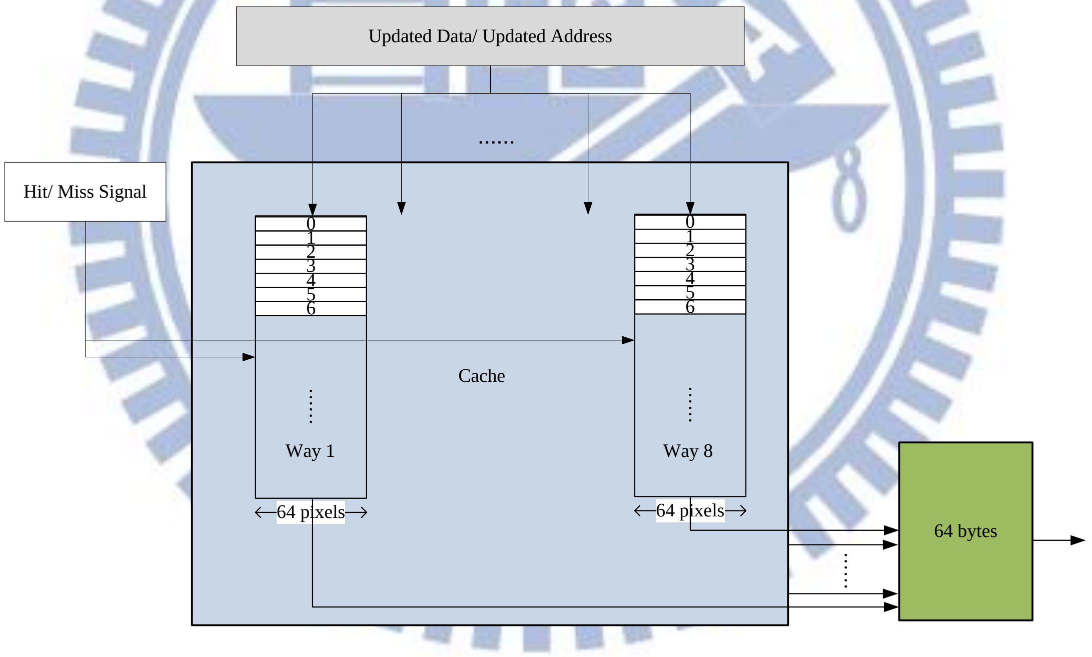


Figure 4-27 Cache architecture

4.5.4 Cache based buffer updating mechanism

The cache updating behavior is controlled by the address controller. The address controller sends request address in the cache to get the desired data for the PEPZS calculation. The desired data will be collected as OUT_DATA, as shown in Figure 4-28, and send to the AMVP or PEPZS calculator. And the controller also sends updating address in the cache to put the updating data in the correct position. The updating data comes from the off-chip memory requested by the address controller. The address controller will update the tag and data if needed when the AMVP or PEPZS calculation request.

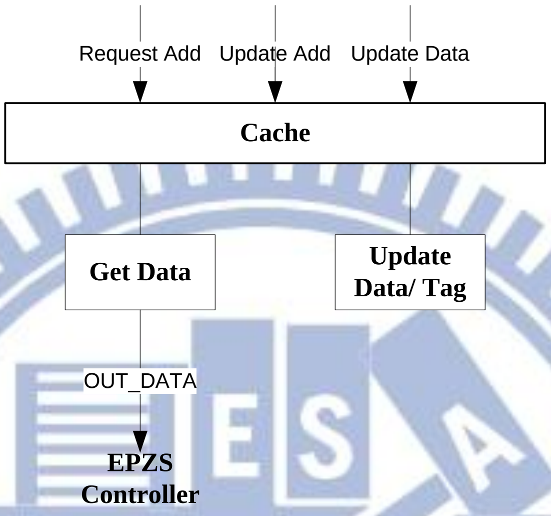


Figure 4-28 Cache fetching and updating.

Chapter 5. Results

5.1 Design Flow

Figure 5-1 shows the design flow in this work. After the defining system target specification, the corresponding C-model is developed. In order to meet the requirement of the system, we exploit the encoding algorithm by software-based approach. When the algorithm is confirmed, the hardware architecture is proposed to implement in Verilog. The RTL functional behavior verification is simulated with the pattern from the C-model.

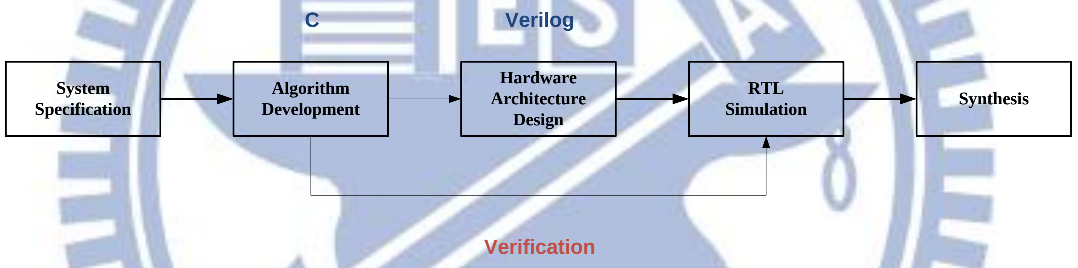


Figure 5-1: Design flow.

5.2 Simulation Results

The proposed IME fast algorithm is implemented on HM 6.0, and the test sequences are listed as Table 5-1:

Table 5-1: Tested sequence description.[2]

Class	**Description**	**Video Name**
Class A	Cropped "Ultra-HD" areas of size 2560x1600 taken from the sequence (30 fps)	Traffic (4096x2048p), PeopleOnStreet(3840x2160p)
Class B	1920x1080p	24 fps: ParkScene, Kimono 50-60 fps: Cactus,

		BasketballDrive, BQTerrace
Class C	832x480p 30-60 fps WVGA	BasketballDrill, BQMall, PartyScene, RaceHorses
Class D	416x240p 30-60 fps WQVGA	BasketballPass, BQSquare, BlowingBubbles, RaceHorses
Class E	1280x720p 60fps video conferencing scenes	FourPeople, Johnny, KristenAndSara

An experiment is conducted under the low-delay B-slice main condition, which allows HM to use fast motion estimation algorithm. HM 6.0 reference software is used as base software. Table 5-2 summarizes BD-rate result in this experiment. The overall BD-rate for our proposed IME fast algorithm is 1.3% for Y-component, 1.4% for U-component, and 1.6% for V-component separately.

Table 5-2: Simulation result of proposed fast IME algorithm under the low delay B-slice main condition.

	Low delay B Main		
	Y	U	V
Class A	1.0%	2.3%	2.1%
Class B	1.6%	0.9%	1.5%
Class C	1.3%	1.7%	1.5%
Class D	1.1%	1.2%	2.0%
Class E	1.7%	1.1%	0.8%
Overall	1.3%	1.4%	1.6%

We pick the worst results from the Class B, C, and D to make the PSNR versus bitrate plots. The worst case is Class C— RaceHorsesC sequence; its BD-rate is 2.2%, 2.4%, and 2.5% for Y, U, and V component separately. But from the Figure 5-2 - Figure 5-4, we can see that the PSNR is only slightly smaller than the reference software HM 6.0.

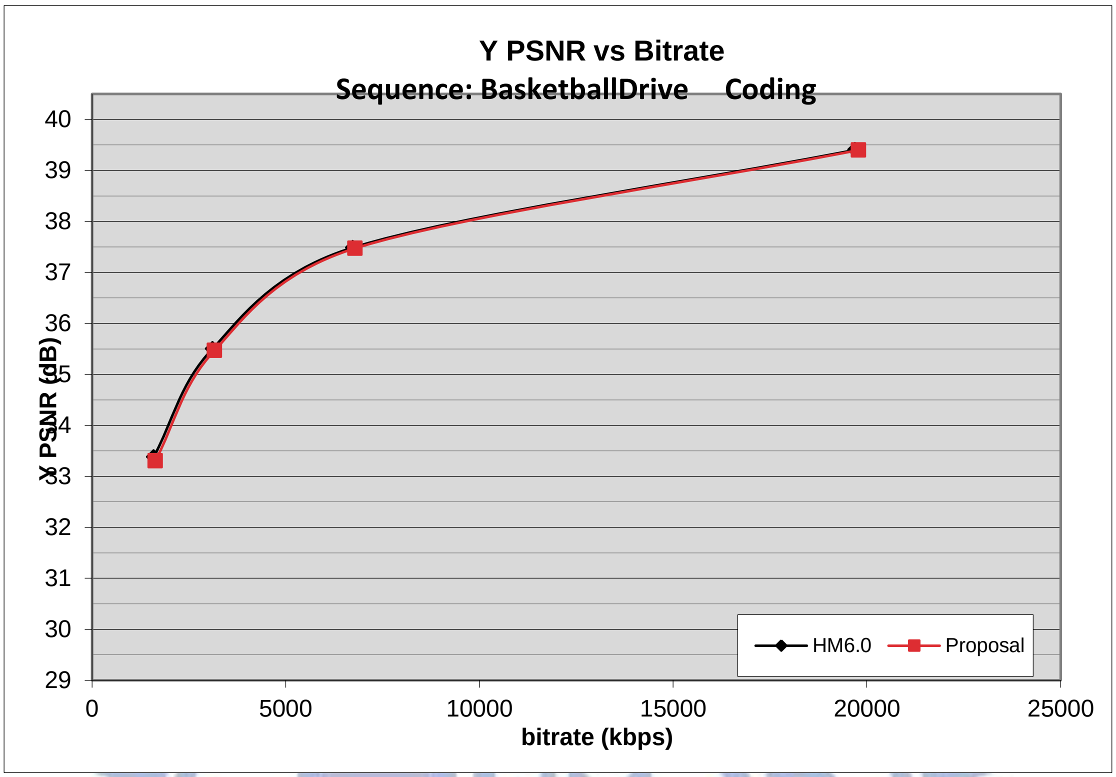


Figure 5-2: Y PSNR v.s. Bitrate, Class B, BasketballDrive

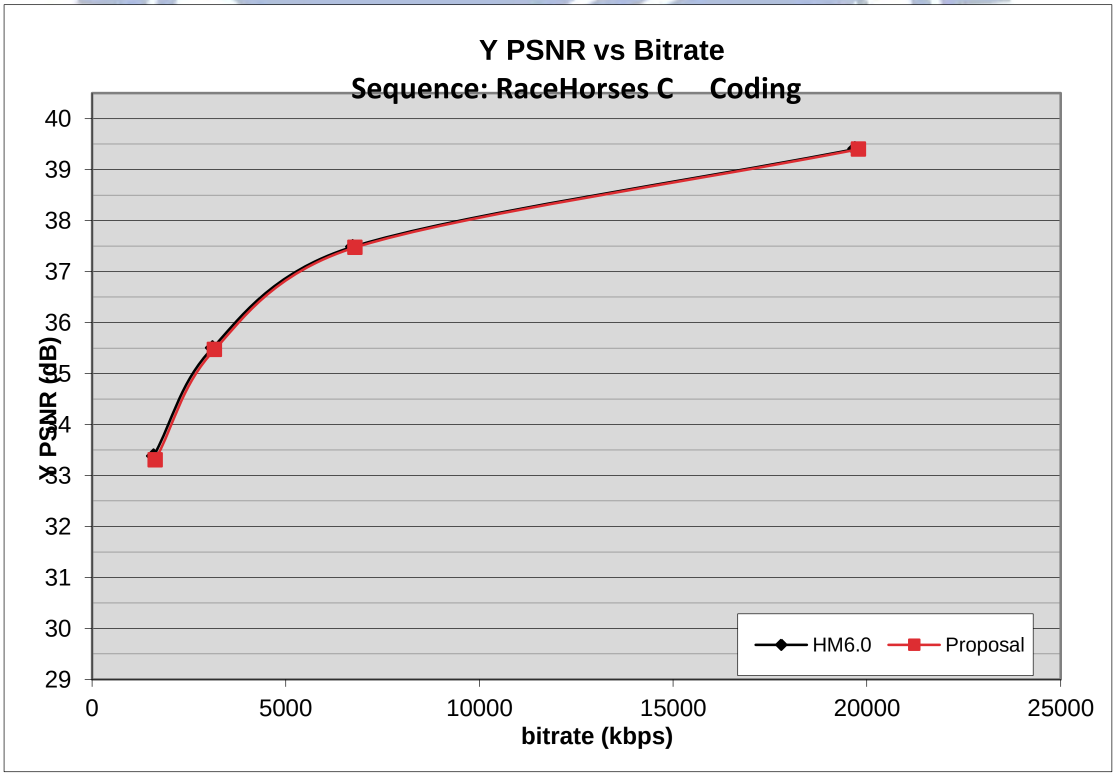


Figure 5-3: Y PSNR v.s. Bitrate, Class C, RaceHorsesC

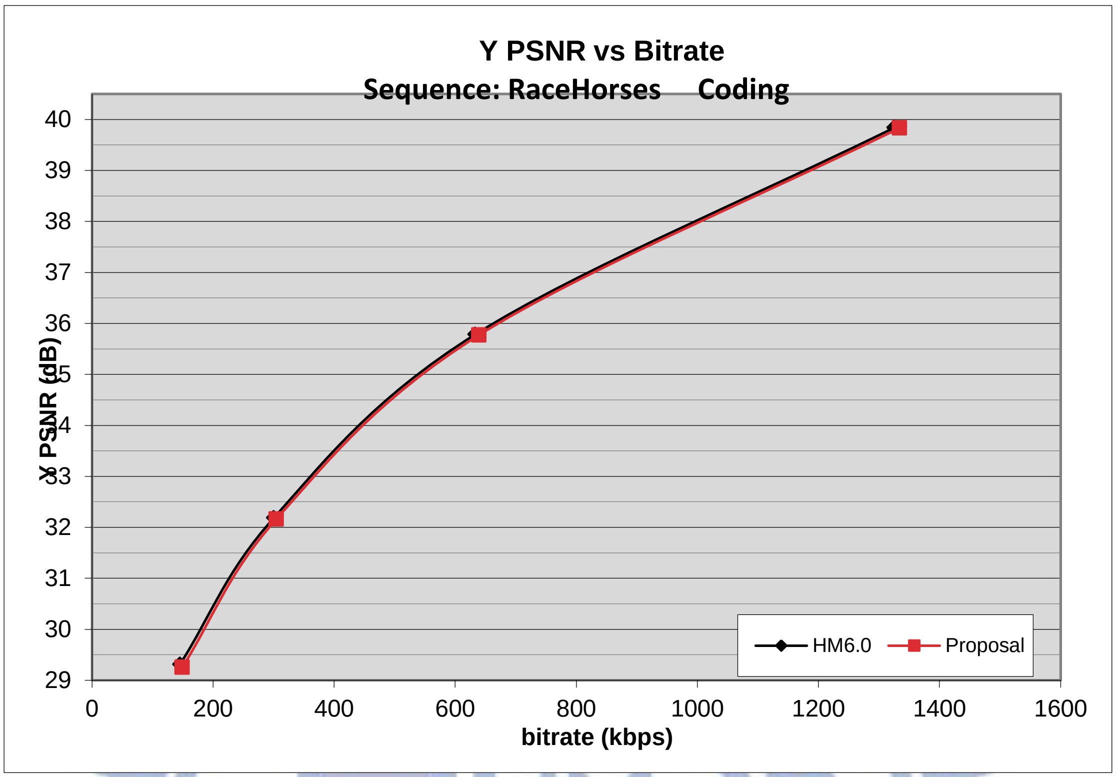


Figure 5-4: Y PSNR v.s. Bitrate, Class D, RaceHorses

Figure 5-5 to Figure 5-8 show the encoded result by two different method, proposed fast IME algorithm and HM 6.0 separately. The red unit is the inter coded blocks. The percentage of each different size PU distribution of the proposed algorithm is very close to the ones of the original HM. The distribution results are identical with the PSNR results, which is also very similar to the PSNR value of original HM.

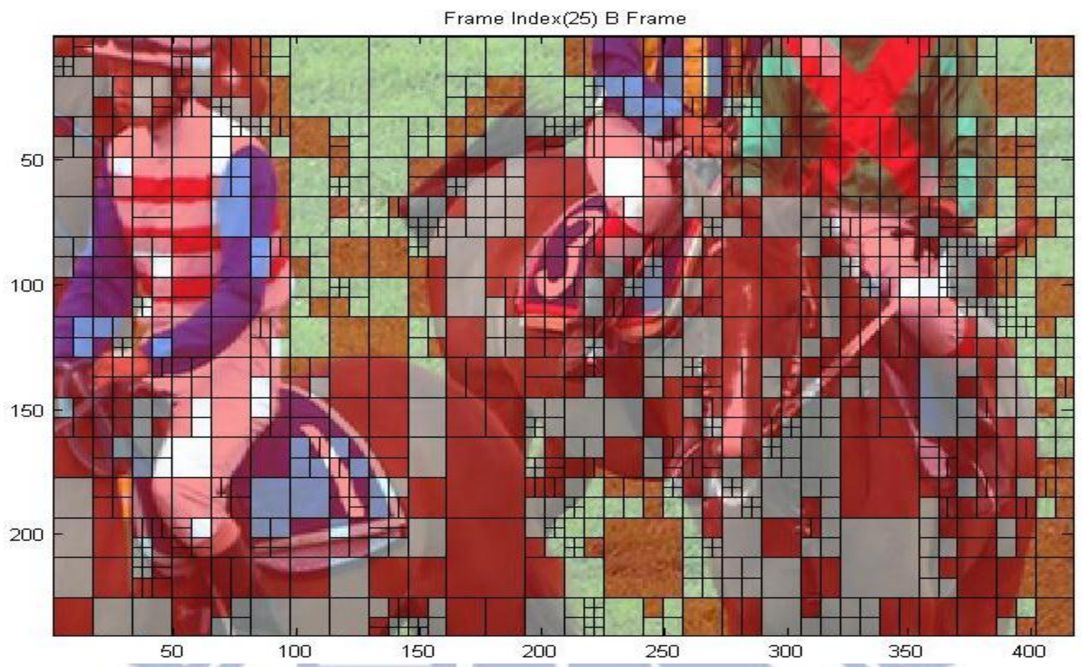


Figure 5-5: Result of proposed IME, RaceHorses_416x240

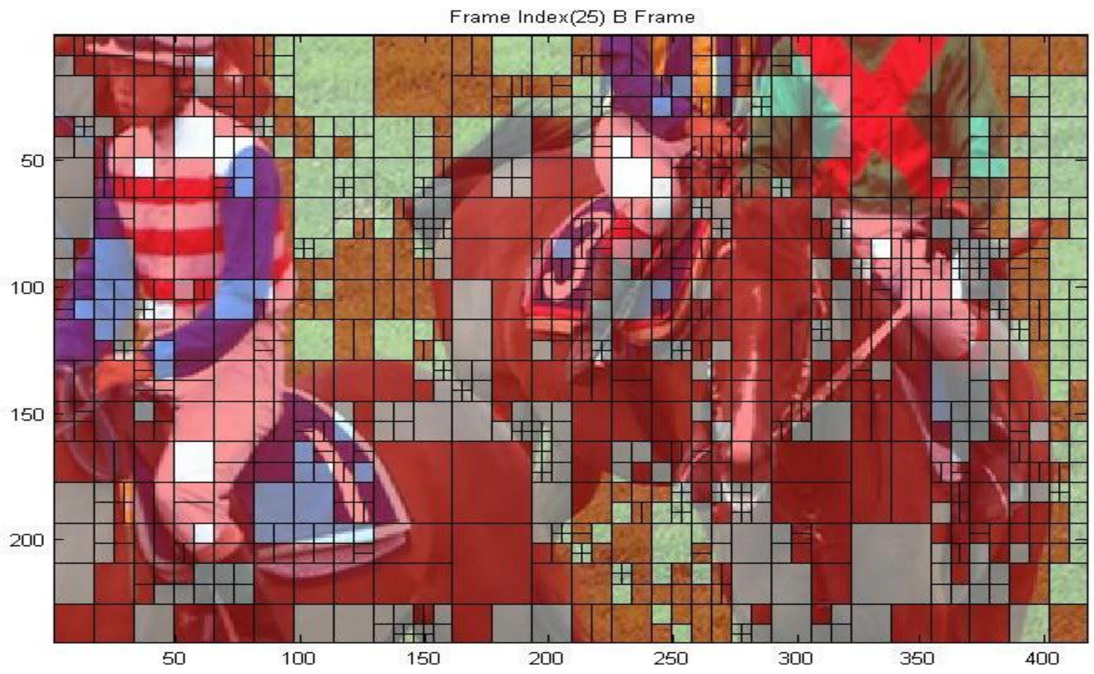


Figure 5-6: Result of original IME, RaceHorses_416x240

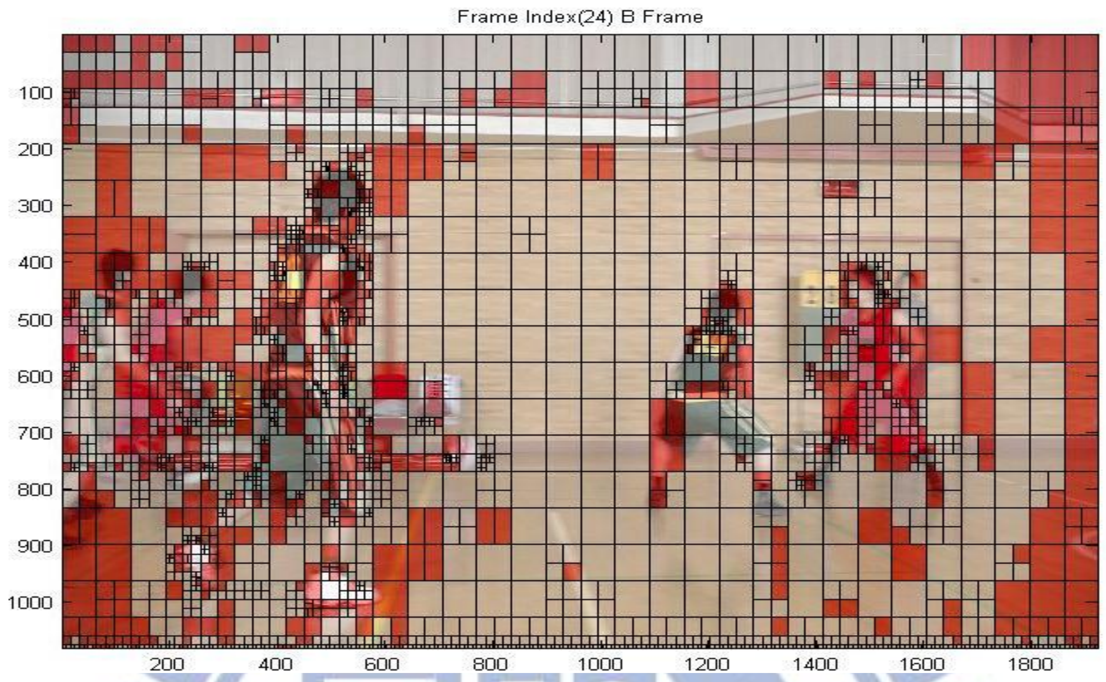


Figure 5-7: Result of proposed IME, BasketballDrive_1920x1080

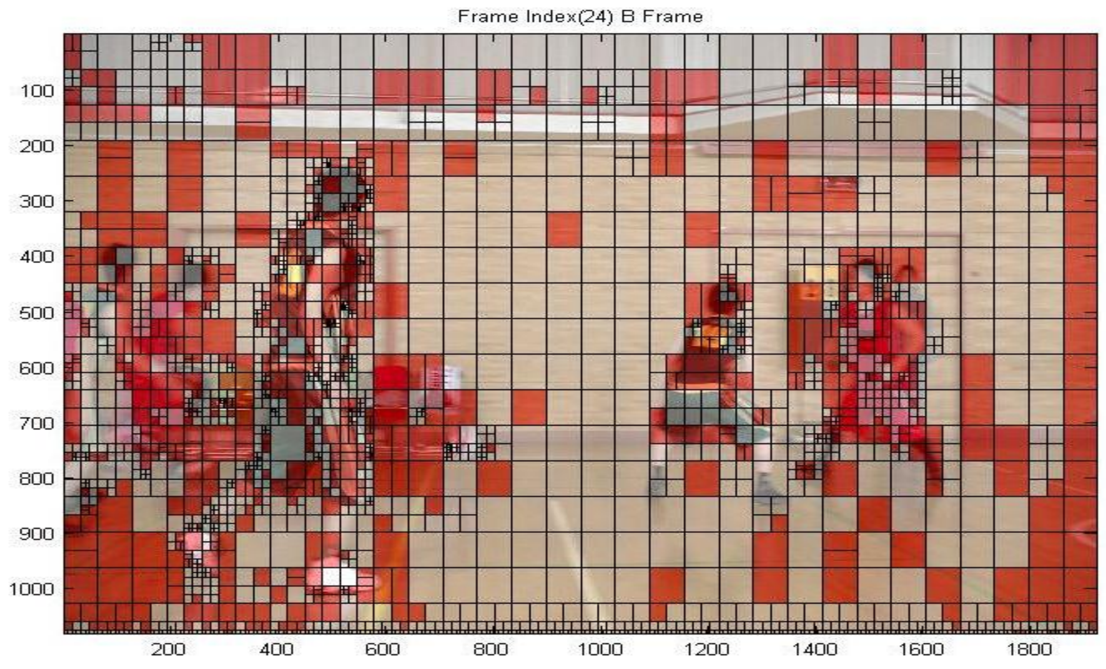


Figure 5-8: Result of original IME, BasketballDrive_1920x1080

5.3 Memory Reduction Results

We can have the memory access comparison between the proposed architecture and other algorithm without data reuse mechanisms. The result of the memory access of AMVP and PEPZS is shown as Table 5-3.

Table 5-3: Memory access comparison. LCU size is 64×64.

	Original AMVP Full Search	**Original AMVP EPZS Diamond**	**Combination 4 at section 3.5**	**Proposed architecture**
Memory Access/CU	375 Kbytes	About 188 Kbytes	About 74 Kbytes	About 8KB
Memory loaded for	MVP blocks + (W+2SR)(H+2SR)	MVP blocks + SP blocks	Reduced MVP blocks + Reduced SP blocks	Cache mechanism

5.4 Synthesis Results

The proposed architecture is implemented by Verilog and synthesis in TSMC 90nm technology at operating frequency 270MHz.

Table 5-4: Synthesis results of proposed architecture.

Module Name	**Gate Count in 270 MHz**
PEPZS module	111,873
Address Controller module	164,914
Cache	2,786
Memory	8K bytes
Total	279,573

As shown in Table 5-4, the memory consumes a large amount of hardware. Since the tag information need to be stored in the address controller that it can be used for cache updating mechanism, the gate count for address controller is a little bit high.

We have our result compared with two other exited architectures, as shown in Table 5-5.

Table 5-5: Comparison of IME architecture.

	Grellert[26]	Lin[19]	Tsai[20]	Tsung[30]	This work
CU	4×4	16×16	16×16	16×16	64×64
Search range	19×19	256×256	256×256	33×33	33×33
Search algorithm	Full search	PMRME	EIMD+PHS	Predictor-centered search	PEPZS
Reference frame #	4	3	1	1	1
Targeted resolution	1280×720 @56fps	1920×1080 @60fps	4k×2k @30fps	4k×2k @24fps	4k×2k @60fps
Clock frequency	265.2MHz	100MHz	125MHz	300MHz	270MHz
Logic Gate	127.83K	180.1K	300K	230K	280K
Memory (Kbytes)	3.96	5.6	12.6	7.81	8
Technology	TSMC 0.18um	UMC 0.13um	UMC 0.13um	TSMC 90nm	TSMC 90nm
Standard	H.264	H.264	H.264	H.264	HEVC

The proposed architecture can supports a CU sized 64×64, while the other two only support 4×4 and 16×16 separately. Since our hardware deals with different PU motion estimation, the logic gate is a little bit higher than [30]. The proposed architecture has the most memory consumption among the 3 different hardware design, however, this work is capable of processing larger frame size than [26] and higher fps than [30]. In other words, if [26] [30] follow our specification, the memory consumption they need may be much more than it does now.

We also can find the hardware cost reduction from the algorithm comparison. For example, if the hardware adopts the algorithm in [25], which have +/-16 search range around MVP, it will need to calculate at least 1024 SADs of PU_64×64. This has to use at least 8 PE_16×16 to meet the specification. Compared with [25], the proposed algorithm only needs 2 PE_16×16 to implement in hardware, which is much lower than [25].

Chapter 6. Conclusion and Future Work

6.1 Conclusion

In this thesis, we have the overall IME design from the algorithm to the hardware. We have discussed the important issues in PEPZS and AMVP when developing the fast IME algorithm and also the high cost and memory use problem in hardware design. The contributions of this thesis can be summarized as follows.

First, we observe and analyze the search direction relation between each step in PEPZS and propose some method to make efficient direction selection, which can be concluded as predictive PEPZS (PEPZS). We also find the AMVP importance in HEVC, the performance degrades if some specific sized AMVP is skipped. Here we analyze the affection of each sized PU to HEVC and skip non-square PUs larger than 16×16 for AMVP calculation while keeps the good compression quality.

Second, we propose an architecture that adopts our fast IME algorithm. We have a scheduling that SAD calculation and data loading are interlaced in order to meet the design specification. We also make the cost down by lowering the PE number. When facing the high memory bandwidth issue, we present a cache design that can fit our fast IME algorithm needs. The overall algorithm and hardware design is implemented with 279K logic gates and 8 KB on-chip memory witch can support the 4Kx2K 60fps video encoding at 270 MHz operation frequency.

6.2 Future work

Although we have the cache design to lower the memory access, the size of cache and local buffer is still an issue for design. In addition, the fast algorithm may be improved by doing some mode selection to reduce the complexity further. This will help the hardware design; it will lower the cost or have flexible time for loading data.

Reference

[1] B. Li, G. J. Sullivan, and J. Xu, "Comparison of Compression Performance of HEVC Working Draft 4 with AVC High Profile", JCTVC-G399, Geneva, CH, 21-30 November, 2011.

[2] G. J. Sullivan and J.-R. Ohm, "Meeting report of the first meeting of the Joint Collaborative Team on Video Coding", JCTVC-A200, Dresden, DE, 15-23 April, 2010.

[3] K. McCann, B. Bross, S.-I. Sekiguchi, and W.-J. Han, "HM4: High Efficiency Video Coding (HEVC) Test Model 4 Encoder Description", JCTVC-F802, Torino, IT, 14-22 July, 2011.

[4] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion compensated interframe coding for video conferencing," in Proc. Nat. Telecommun. Conf., New Orleans, LA, Nov. 29–Dec. 3 1981, pp. G5.3.1–5.3.5.

[5] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," IEEE Trans. Circuits Syst. Video Technol., vol. 4, pp. 438–442, Aug. 1994.

[6] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast blockmatching motion estimation," IEEE Trans. Image Processing, vol. 9, pp.287–290, Feb. 2000.

[7] J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, "A novel unrestricted center-biased diamond search algorithm for block motion estimation," IEEE Trans. Circuits Syst. Video Technol., vol. 8, pp. 369–377, Aug. 1998.

[8] C. Zhu, X. Lin, and L. P. Chau, "Hexagon-based search pattern for fast block motion estimation," IEEE Trans. Circuits Syst. Video Technol, vol. 12, pp. 349–355, May 2002.

[9] A. M. Tourapis, "Enhanced predictive zonal search for single and multiple frame motion estimation," in Proc. VCIP, Jan. 2002, pp. 1069–1079.

[10] C. H. Cheung and L. M. Po, "A novel cross-diamond search algorithm for fast block motion estimation," IEEE Trans. Circuits Syst. Video Technol., vol. 12, no. 12, pp. 1168–1177, Dec. 2002.

[11] H. Y. C. Tourapis and A. M. Tourapis, "Fast motion estimation within the H.264 codec," in Proc. IEEE Int. Conf. Multimedia Expo, 2003, pp. 517–520.

[12] W.-M. Chao, C.-W. Hsu, Y.-C. Chang, and L.-G. Chen, “A novel hybrid motion estimator supporting diamond search and fast full search,” in Proc. IEEE Int. Symp. Circuits Syst., May 2002, vol. 2, pp. II-492–II-495.

[13] T. Li, S. Li, and C. Shen, “A novel configurable motion estimation architecture for high-efficiency MPEG-4/H.264 encoding,” in Proc. Asia and South Pacific Design Automation Conf., vol. 2, Shanghai, China, Jan. 2005, pp. 1264–1267.

[14] T.-C. Chen, S.-Y. Chien, Y.-W. Huang, C.-H. Tsai, C.-Y. Chen, T.-W. Chen, and L.-G. Chen, “Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder,” IEEE Trans. Circuits Syst. Video Technol., vol. 16, no. 6, pp. 673–688, Jun. 2006.

[15] Xiong, X. X., Y. Song, and A. Akoglu, “Architecture Design of Variable Block Size Motion Estimation for Full and Fast Search Algorithms in H.264/AVC,” Computers Electrical Engineering, vol. 37, Issue 3, pp. 285–299, May 2011.

[16] O. Ndili and T. Ogunfunmi, "Algorithm and Architecture Co-Design of Hardware-Oriented, Modified Diamond Search for Fast Motion Estimation in H.264/AVC," IEEE Trans. Circuits Syst. Video Technol., vol.21, no.9, pp. 1214-1227, Sept. 2011.

[17] D.-X. Li, W. Zheng, and M. Zhang, “Architecture Design for H.264/AVC Integer Motion Estimation with Minimum Memory Bandwidth,” IEEE Trans. Consumer Electron., vol.53, no.3, pp.1053-1060, Aug. 2007.

[18] J. Vanne, E. Aho, K. Kuusilinna, and T. D. Hämäläinen, “A configurable motion estimation architecture for block-matching algorithms,” IEEE Trans. Circuits Syst. Video Technol., vol. 19, no. 4, pp. 74–86, Apr. 2009.

[19] C.-C. Lin, Y.-K. Lin, T.-S. Chang, “PMRME: A parallel multi-resolution motion estimation algorithm and architecture for HDTV sized H.264 video coding,” in Proc. ICASSP, 2007.

[20] T.-H. Tsai and Y.-N. Pan, “ High Efficiency Architecture Design of Real-Time QFHD for H.264/AVC Fast Block Motion Estimation,” IEEE Trans. Circuits Syst. Video Technol., vol. 21, no. 11, pp. 1646–1658, Apr. 2011.

[21] Y.-H. Chen et al. “An H.264/AVC Scalable Extension and High Profile HDTV 1080p Encoder Chip,” 2008 Symposium on VLSI Circuits Digest of Technical Papers, pp.104-105, Jun. 2008.

[22] W. Cao, H. Hui, J. Tong, J. Lai, and H. Min, “A high-performance reconfigurable VLSI architecture for VBSME in H.264,” IEEE Trans. Consumer Electron., vol. 54, no. 3, pp. 1338–1345, Aug. 2008.

[23] L.F. Chen et al. “Hardware Efficient Coarse-to-Fine Fast Algorithm for H.264/AVC Variable Block Size Motion Estimation,” IEEE International Symposium on Circuits and Systems, pp.1657-1660, May 2009.

[24] X. Wen, Oscar C. Au, Jiang Xu, Lu Fang, Run Cha, and Jiali Li, “Novel RD-optimized VBSME with matching highly data re-usable hardware architecture, ” IEEE Trans. on Circuits and Systems for Video Technology, vol. 21, no 2, pp. 206-219, Feb. 2011.

[25] L.-F. Ding, W.-Y. Chen, et al, “A 212 MPixels/s 4096 2160p Multiview Video Encoder Chip for 3D/Quad Full HDTV Applications,” IEEE Journal of Solid-State Circuits, Vol. 45, No. 1,pp. 46-58,Jan, 2010.

[26] M. Grellert et al. “A multilevel data reuse scheme for Motion Estimation and its VLSI design”. In: IEEE ISCAS, pp. 583-586, 2011.

[27] C. Y. Tsai, C. H. Chung, Y. H. Chen, T. C. Chen, L. G. Chen, "Low Power Cache Algorithm and Architecture Design for Fast Motion Estimation in H.264/AVC Encoder System," ICASSP 2007. IEEE International Conference, vol.2, no., pp.II-97-II-100, 15-20 April 2007.

[28] L.-F. Ding, W.-Y. Chen, P.-K. Tsung, T.-D. Chuang, P.-H. Hsiao, Y.-H. Chen, H.-K. Chiu, S.-Y. Chien and L.-G. Chen, “A 212 MPixels/s 4096x2160p Multiview Video Encoder Chip for 3D/Quad Full HDTV Applications,” IEEE Journal of solid-state circuits, vol. 45, no. 1, Jan. 2010.

[29] W.-Y. Chen et al, “Algorithm and Architecture Design of Cache System for Motion Estimation in High Definition H.264/AVC,” Proceedings of ICASSP 2008, pp. 2193-2196.

[30] P.-K. Tsung, W.-Y. Chen, L.-F. Ding, S.-Y. Chien, and L.-G. Chen,“Cache-based integer motion/disparity estimation for quad-HD H.264/AVC and HD multiview video coding,” in Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing, 2009, pp. 2013–2016.

[31] J. Vanne, E. Aho, T. D. Hämäläinen, and K. Kuusilinna, “A parallel memory system for variable block-size motion estimation algorithms,” IEEE Trans. Circuits Syst. Video Technol., vol. 18, no. 4, pp. 538–543, Apr. 2008.