

國立交通大學

電子工程學系電子研究所

博士論文

針對高畫質視訊之H.264/MPEG-4 AVC視訊編碼
器設計

Design of H.264/MPEG-4 AVC Video Encoder for
High Definition Video

研究生：林佑昆

指導教授：張添烜 教授

任建葳 教授

中華民國 九十七年 六月

針對高畫質視訊之H.264/MPEG-4 AVC視訊編碼
器設計

Design of H.264/MPEG-4 AVC Video Encoder for
High Definition Video

研究生：林佑昆

Student：Yu-Kun Lin

指導教授：張添烜博士

Advisor：Dr. Tian-Sheuan Chang

任建葳博士

Dr. Chein-Wei Jen



A Dissertation

Submitted to Department of Electronics Engineering and
Institute of Electronics

College of Electrical and Computer Engineering
National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Electronics Engineering

June 2008

Hsinchu, Taiwan, Republic of China

中華民國 九十七年六月

針對高畫質視訊之 H.264/MPEG-4 AVC 視訊編碼器設計

學生：林佑昆

指導教授：張添烜教授

任建葳教授

國立交通大學

電子工程學系 電子研究所

摘要

H.264 因為其具備的高壓縮率與高畫質，已是目前最被廣泛採用的視訊壓縮標準。但是其主要的問題是需要極高的運算量，特別是要支援到 1920x1080 (1080p)，所謂的高清畫質解析度時，其所需即時處理的資料量更達到以往 1280x720 (720p) 解析度的四倍以上，所需要支援的功能也更多，很難使用軟體架構進行即時編碼。所以使用單晶片架構來設計 H.264 編碼器，已被廣泛採用於業界與學界。但如果使用硬體架構進行即時的 H.264 編碼，不論是在硬體面積、記憶體的数量與頻寬等方面，仍需要極高的成本。此外 H.264 所需的高運算量會導致低資料輸出率與高操作頻率。總和以上因素，巨大的功率消耗也是不可避免的。因此本論文提出了學術界第一個可以即時編碼 1080p 解析度之視訊，並且支援 H.264 高級規範的單晶片，此晶片中使用多種演算法與架構上的最佳化技術，將其硬體的成木與消耗功率降到最低，並且幾乎對其畫質與壓縮率沒有影響。

本論文共包含三大部分。首先，本論文針對 H.264 編碼器中最消耗硬體資源與運算量的移動偵測模組，進行討論與分析。因應 H.264 特有的可變區塊尺寸移動偵測技術，我們提出了模式濾波技術，在所有可能的區塊尺寸組合中，只挑出兩

組最好的組合進行微調，藉此節省了 73.2% 的運算量。在整數移動偵測部分，為了達到影像品質與硬體成本之間的最佳平衡，本論文採用了多層次的平行化移動偵測的技術，此技術可以減少 91.7% 的運算量與 30% 的硬體面積。此外本論文也使用 C 層級的資料重複採用技術，以減少記憶體存取量，藉此減少 88% 的內部記憶體與 46% 的記憶體頻寬。接著在分數移動偵測部分，本論文採用一次遞迴的技術，使資料處理速度變成以往所有採用二次遞迴技術之設計的兩倍，同時也節省了 68% 的硬體。綜合了以上的技術之後，本論文提出了一個能夠支援 1080p 解析度，並且搜尋範圍能夠達到 ± 128 的 H.264 移動偵測器。相較於之前的研究，我們的設計可減少 60% 的硬體面積與 68.9% 的內部記憶體。

論文的第二部分是 H.264 框內編碼器的架構設計。H.264 規格中的框內編碼，提供了比過去的影像壓縮技術如 JPEG2000 等，更高的壓縮率，可是又不需像移動偵測如此巨大的運算量與系統資源，因此是影像處理或低功耗視訊壓縮的一個新選擇，但其硬體設計的主要缺點是因為其可選擇的預測模式過多而導致的低資料輸出率。因此本論文提出了一個高資料輸出率與小面積的 H.264 框內編碼器。首先，本論文採用了一個修改過的三步快速演算法，在確保影像品質不下降時，減少運算所需要的時間。此外，此編碼器採用可變平行度的設計概念，在運算量較高的部分採用較高平行度架構，但在非瓶頸區域，則採用較低平行度架構，以減少硬體需求。此設計同樣能夠即時處理 1080p 解析度的視訊，並減少 23.5% 的硬體面積。此外因為操作頻率可以減少 48%，並且也採用了多項低功率技術，故能夠達到低功耗的效果。

本論文的最後一部分是一個完整的 H.264 高級規範編碼器，因為許多支援高清解析度的應用採用 H.264 標準中的高級規範，所以我們將論文前半部提出的移動偵測器與框內編碼器，再結合了高級規範裡的新工具，整合成一個完整可支援 1080p 解析度的 H.264 高級規範編碼器。因為比起基礎規範編碼器，高級規範編碼器的設計有更大的挑戰在資料傳輸率、硬體資源與功率消耗上。此外，移動偵測模組與框內編碼模組在三級平行化系統架構當中，其重建模組會有時間上的衝

突，因此在系統層面上，我們提出了跨平行化階層的硬體共享技術，以除去這項時間衝突與減少重複的硬體。此外我們採用全八點平行處理的技術，更進一步的加快資料處理速度，以免新增的高級規範工具變成系統瓶頸。在移動偵測的部分，我們讓新的雙向移動偵測共用同一組硬體，以減少面積；此外整數移動偵測與分數移動偵測硬體間也共享內部記憶體，以減少記憶體面積與所需頻寬。總之，這個學術界第一個發表的高級規範編碼器，在 145MHz 下便可支援 1080p 解析度，使用 0.13 微米製程時，其面積只要 3.17x3.17 平方毫米，只占過去類似設計的 54%。支援 1080p 解析度時的功率消耗只要 242 毫瓦，而支援 720p 解析度時，功率消耗只需要過去類似設計的 46.3%。而此小面積、低功率但高資料處理速度的設計也證明了本論文的研究成果確實適用在高畫質的視訊處理之上。





Design of H.264/MPEG-4 AVC Video Encoder for High Definition Video

Student: Yu-Kun Lin

Advisor: Dr. Tian-Sheuan Chang

Dr. Chein-Wei Jen

Department of Electronics Engineering

Institute of Electronics

National Chiao Tung University

Abstract

H.264 video standard has been widely adopted in high definition video applications because of its high compression efficiency and video quality. However, the major bottlenecks of H.264 implementation are its high computational loading and large memory bandwidth, especially for encoding 1920x1080 (1080p) high definition video in real time. Therefore, this dissertation proposes the first chip in academia which can both support H.264 high profile and encode 1080p video in real time.

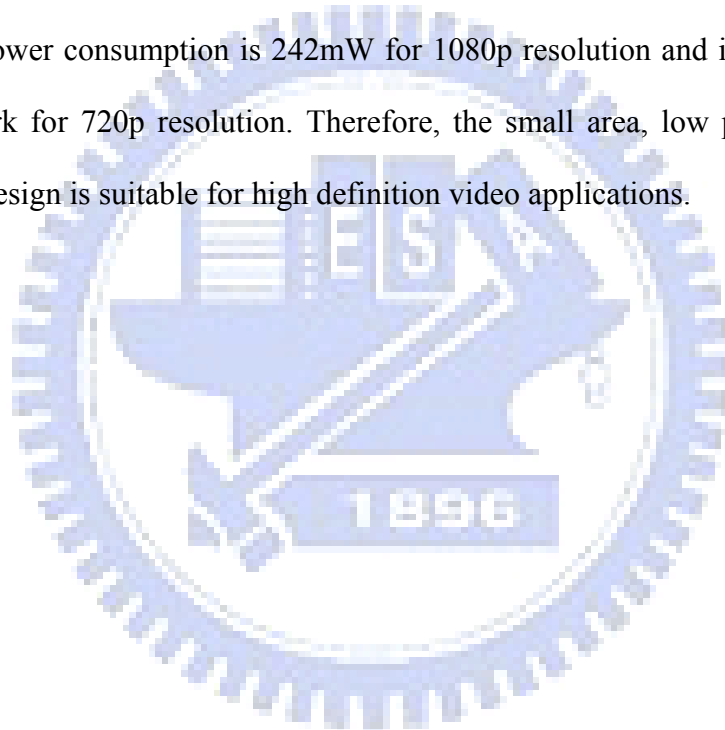
This dissertation contains three parts. First, we discuss and analyze the inter prediction modules which occupy the most memory bandwidth and hardware cost in H.264 encoder. To overcome these problems, we present a low complexity and hardware efficient motion estimation design with several design techniques. The first low complexity technique, mode filtering, selects the best two candidates of all possible block size combinations for refinement, and reduces the computations of fractional refinement by 73.2%. To further reduce the complexity and hardware cost,

we propose a multi-level parallel processing technique in integer motion estimation stage. By this technique, 91.7% of complexity and 30% of gate count can be reduced. Furthermore, 88% of local memory size and 46% of external memory bandwidth can be reduced by the level C data reuse technique. Finally, our proposed single iteration technique can remove 68% of gate count and double the throughput of fractional motion estimation stage, which is a bottleneck in the inter prediction modules. In summary, the proposed H.264 inter prediction engine not only can support 1080p resolution and ± 128 search range but also can reduce 60% of hardware and 68.9% of internal SRAM than previous work.

The second part of the dissertation is the architecture design of H.264 intra encoder. The intra encoder in H.264 standard provides comparable coding efficiency with JPEG 2000 standards. To achieve high throughput and low area cost, we apply the modified three-step fast intra prediction to reduce the cycle count while keeping the quality as close as full search. Then, we further adopt the variable pixel parallelism to speed up performance on the critical intra prediction part while keeping other parts with low area cost. The achieved design supports 1080p video encoding and reduces 23.5% of gate count cost compared to the previous design. In addition, this design can achieve low power consumption by reducing 48% of operating frequency and several low power techniques.

The final part of this dissertation is a complete H.264 high profile encoder. Because several high definition applications apply H.264 high profile, we integrate our motion estimation engine, intra encoder, and the new coding tools of H.264 high profile into a complete H.264 high profile encoder supporting 1080p video. These 1080p high profile applications present a series of new design challenges in throughput, cost and power. Furthermore, in system level, a timing conflict happens in the reconstruction stage of inter and intra prediction due to the three pipelined stages architecture.

Therefore, we first propose the crossing stage hardware sharing technique to remove the conflict and repeated hardware. To solve the high throughput demands and structural hazards, this design adopts full eight-pixel parallelism. In motion estimation part, the bi-directional motion estimation modules share the hardware, and the integer and fractional motion estimation modules also share the local SRAM to reduce the internal memory size and bandwidth. In summary, we propose the first H.264 high profile encoder in academia which supports 1080p resolution under only 145MHz. The core area is $3.17 \times 3.17 \text{mm}^2$ under $0.13 \mu\text{m}$ process, which is only 54% of previous work. The power consumption is 242mW for 1080p resolution and is only 46.3% of previous work for 720p resolution. Therefore, the small area, low power, and high throughput design is suitable for high definition video applications.





誌 謝

一轉眼，在交大就度過了六年時光。其實這幾年的博士生涯，並不是一帆風順的，途中也經歷了尋找研究主題時的迷惘、論文被拒絕時的打擊等種種困難，所以最後能夠順利的得到這個學位，其實得到過很多人的幫助。首先要感謝博士班引領我入門的任建葳教授，提供我良好的研究環境與可自由揮灑的研究空間。此外，我也致最高的謝意給我的指導教授—張添烜教授。張教授在這幾年間在研究方向、論文撰寫等方面，不厭其煩的給我指導與鼓勵，讓我最後能夠克服難關，完成學業。當然，我也要感謝我的口試委員：李鎮宜教授、蔣迪豪教授、方偉騏教授、楊家輝教授、林永隆教授、陳永昌教授、吳炳飛教授、蔡宗漢教授，在百忙當中抽空來參加我的論文口試，並且給了我精闢的建議，讓我獲益良多。

除了諸位師長，我更要感謝我的父母，一直給我全力的支持與協助；也要感謝他們在我遇到挫折時，能夠容忍我的壞脾氣並給我鼓勵。此外也要謝謝老弟這幾年來的相互協助。還要感謝阿姨與姨丈對我在新竹這幾年的照顧，讓我可以專心於研究。

接著要謝謝這幾年與我一起共度的交大學長、同學與學弟妹們。首先要感謝李坤儉學長在我剛進入博士班時的悉心指導，還有李元仲學長在我剛接手實驗室工作站時給我的協助。再來要謝謝我的好同學 Nelson 張彥中，這六年裡一起奮鬥，相互砥礪，讓我獲益匪淺。接著要謝謝和我共同完成 H.264 Encoder Chip 的學弟們：嘉俊、得瑋、子筠、私璟、瑋呈、瑋城，論文能夠被 ISSCC 接受，是大家共同努力的成果。還有其他 H.264 戰隊的學弟們：朝鐘、君偉、裕仁、國亘、旻奇、錦木，與你們的教學相長也讓我成長。也要感謝所有張教授實驗室的學弟妹們：Esam、浩雲、昕儀、惠錚、彥芪、英澤、國龍、宇晟、宗憲、景竹、筱珊、之悠、孟維、博淵、政君，謝謝你們讓我這幾年的研究生涯充滿歡笑。還要謝謝子明，與所上所有幫助過我的學長學弟們。

最後謹將這本論文，獻給所有關心我的人們。



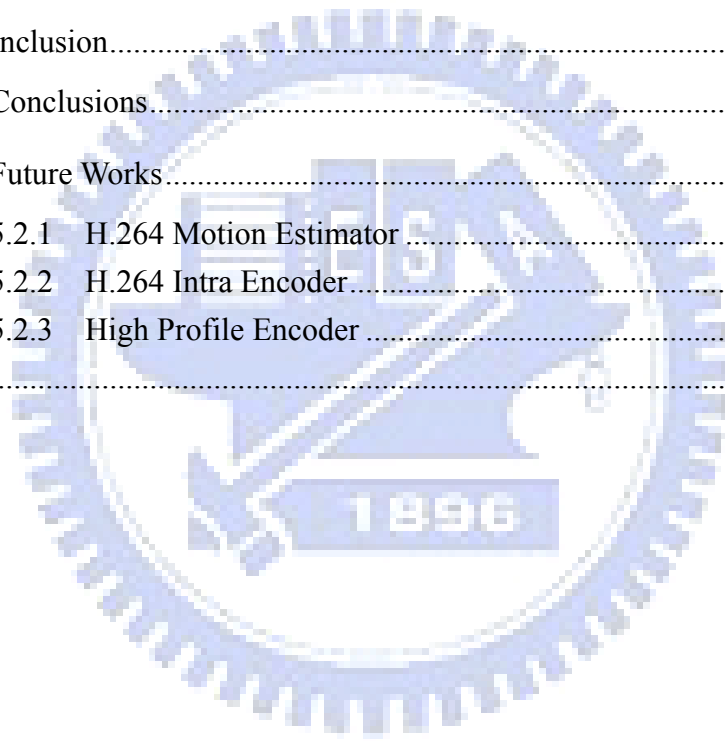
Contents

| | |
|--|----|
| Chapter 1 Introduction | 1 |
| 1.1 Overview of H.264..... | 1 |
| 1.1.1 History of H.264 | 1 |
| 1.1.2 Introduction of H.264 encoder and decoder | 2 |
| 1.1.3 Profiles and levels of H.264 specification | 5 |
| 1.2 Motivation of Thesis | 7 |
| 1.3 Organization and Contribution of Thesis..... | 8 |
| Chapter 2 High Performance H.264 Motion Estimator for HDTV | 11 |
| 2.1 Introduction to H.264 Motion Estimation..... | 12 |
| 2.1.1 System overview for H.264 motion estimation | 12 |
| 2.1.2 Variable block size motion estimation (VBSME)..... | 12 |
| 2.1.3 Quarter-pel fractional motion estimation..... | 14 |
| 2.1.4 Multiple reference frames..... | 15 |
| 2.1.5 Skip mode | 15 |
| 2.2 Design Challenges and Paper Survey | 16 |
| 2.2.1 Design challenges | 16 |
| 2.2.2 Paper survey..... | 17 |
| 2.3 Mode Filtering Algorithm..... | 18 |
| 2.3.1 Introduction to mode filtering..... | 18 |
| 2.3.2 Simulation result of mode filtering..... | 21 |
| 2.3.2.1 Performance of QCIF/CIF sequences | 21 |
| 2.3.2.2 Performance of 720p sequences..... | 23 |
| 2.4 Integer Motion Estimation Module : Parallel Multi-Resolution Motion Estimation (PMRME) [35] | 25 |
| 2.4.1 Algorithm of PMRME | 25 |
| 2.4.2 Performance of PMRME | 26 |
| 2.4.3 Architecture of PMRME..... | 28 |
| 2.4.4 Implementation result and comparisons | 35 |
| 2.5 Fractional Motion Estimation Module: Single Iteration Fractional Motion Estimation (SIFME) [40] | 37 |
| 2.5.1 Algorithm of SIFME..... | 37 |

| | | |
|--|--|----|
| 2.5.2 | Performance of SIFME | 40 |
| 2.5.3 | Architecture of SIFME | 44 |
| 2.5.4 | Implementation result and comparisons of SIFME | 47 |
| 2.6 | Integrated Design | 49 |
| 2.6.1 | Integrated video quality analysis | 49 |
| 2.6.2 | Integrated architecture | 52 |
| 2.6.3 | Implementation results and comparisons..... | 53 |
| 2.7 | Summary | 54 |
| Chapter 3 Design of H.264 1080p Intra-only Encoder | | 57 |
| 3.1 | Introduction of H.264 intra-only encoder | 58 |
| 3.1.1 | Overview of H.264 Intra-only encoder | 58 |
| 3.1.2 | Intra prediction..... | 59 |
| 3.1.3 | 4x4 integer DCT/IDCT | 59 |
| 3.1.4 | Quantization/Inverse quantization | 62 |
| 3.1.5 | CAVLC | 62 |
| 3.2 | Design Challenges and Paper Survey | 63 |
| 3.2.1 | Design challenges | 63 |
| 3.2.2 | Paper survey..... | 64 |
| 3.3 | Fast and Hardware-Efficient Intra Prediction Algorithms | 64 |
| 3.3.1 | Modified three step algorithm [52]..... | 64 |
| 3.3.2 | Enhanced SATD algorithm [42]..... | 68 |
| 3.3.3 | Plane mode removal technique [42] | 70 |
| 3.3.4 | Performance comparison | 71 |
| 3.4 | Architecture of Intra-only Encoder | 74 |
| 3.4.1 | Overview of intra-only encoder with variable pixel parallelism | 74 |
| 3.4.2 | Schedule of encoder | 75 |
| 3.4.3 | Architecture of eight-pixel parallelism modules..... | 79 |
| 3.4.3.1 | Eight-pixel intra predictor | 79 |
| 3.4.3.2 | Eight-pixel DCT..... | 81 |
| 3.4.4 | Architecture of four-pixel parallelism modules..... | 82 |
| 3.4.4.1 | Four-pixel IDCT | 82 |
| 3.4.4.2 | Q/IQ | 83 |
| 3.4.5 | Architecture of CAVLC module | 83 |
| 3.5 | Implementation Results and Comparison | 86 |
| 3.5.1 | Implementation results..... | 86 |

| | | |
|-----------|--|-----|
| 3.5.2 | Comparison with previous works | 88 |
| 3.6 | Summary | 89 |
| Chapter 4 | H.264 HD1080p High Profile Encoder Chip | 93 |
| 4.1 | Overview of H.264/AVC High Profile | 94 |
| 4.1.1 | History of H.264/AVC high profile | 94 |
| 4.1.2 | Introduction of the coding tools of H.264 high profiles and levels 94 | |
| 4.1.3 | Introduction to new tools of H.264/AVC high profile encoder ... | 95 |
| 4.1.3.1 | 8x8 intra prediction | 95 |
| 4.1.3.2 | 8x8 transform | 96 |
| 4.1.3.3 | Weighted bi-directional motion estimation | 97 |
| 4.1.3.4 | Context adaptive binary arithmetic coding (CABAC) | 97 |
| 4.1.3.5 | Deblocking | 100 |
| 4.2 | Design Challenges and Paper Survey | 101 |
| 4.2.1 | Design challenges | 101 |
| 4.2.2 | Paper survey | 102 |
| 4.3 | System Overview | 103 |
| 4.4 | Schedule of H.264 High Profile Encoder | 104 |
| 4.5 | System Level Hardware Sharing Techniques | 105 |
| 4.5.1 | Reconstruction sharing | 105 |
| 4.5.2 | Hardware-shared bi-directional motion estimation | 106 |
| 4.6 | Full eight-pixel intra encoder | 107 |
| 4.6.1 | Intra predictor | 110 |
| 4.6.2 | Interlaced schedule with intra 8x8 prediction | 111 |
| 4.6.3 | 8x8 transform unit | 113 |
| 4.6.4 | Shared 8x8 inverse transform unit | 114 |
| 4.6.5 | 8-pixel quantization and inverse quantization unit | 118 |
| 4.7 | Bi-directional Inter Predictor Module | 119 |
| 4.7.1 | Techniques for inter prediction | 119 |
| 4.7.2 | 4x4 SATD cost function | 121 |
| 4.8 | Architecture of CABAC [73] | 123 |
| 4.8.1 | The proposed algorithm flow and architecture of CABAC | 123 |
| 4.8.2 | Architecture of binarization | 124 |
| 4.8.3 | Architecture of context modeling | 124 |

| | | |
|------------|--------------------------------------|-----|
| 4.8.4 | Architecture of AC | 124 |
| 4.8.5 | Interval maintainer in AC..... | 125 |
| 4.8.6 | Renormalization in AC | 125 |
| 4.9 | Deblocking Filter | 129 |
| 4.10 | Implementation Result | 131 |
| 4.10.1 | Chip specification | 131 |
| 4.10.2 | Power measurement result | 131 |
| 4.10.3 | Comparisons with previous work | 131 |
| 4.11 | System Integration | 135 |
| 4.12 | Summary | 135 |
| Chapter 5 | Conclusion..... | 137 |
| 5.1 | Conclusions..... | 137 |
| 5.2 | Future Works..... | 138 |
| 5.2.1 | H.264 Motion Estimator | 138 |
| 5.2.2 | H.264 Intra Encoder..... | 139 |
| 5.2.3 | High Profile Encoder | 139 |
| References | | 141 |



List of Figures

| | |
|---|----|
| Fig. 1-1 The basic structure of encoder..... | 3 |
| Fig. 1-2 The basic structure of decoder..... | 3 |
| Fig. 1-3 Organization of this thesis..... | 9 |
| Fig. 2-1 Block diagram of H.264 motion estimator..... | 13 |
| Fig. 2-2 Block sizes and hierarchy for H.264 motion estimation..... | 13 |
| Fig. 2-3 Integer samples and fractional sample positions for (a) luma and (b) chroma interpolation..... | 14 |
| Fig. 2-4 Multiple references in motion estimation..... | 15 |
| Fig. 2-5 (a) The original coding flow between IME and FME (b) Mode filtering algorithm..... | 20 |
| Fig. 2-6 The rate-distortion curves of QCIF sequences..... | 22 |
| Fig. 2-7. The rate-distortion curves of CIF sequences..... | 22 |
| Fig. 2-8 The rate-distortion curves of 720p sequences..... | 24 |
| Fig. 2-9. The three-level new multi-resolution algorithm..... | 26 |
| Fig. 2-10 The rate-distortion curves of 720p sequences..... | 27 |
| Fig. 2-11 The rate-distortion curves of 1080p sequences..... | 28 |
| Fig. 2-12. The proposed architecture of IME stage..... | 31 |
| Fig. 2-13. Basic 4p-SAD unit can accumulate the SAD of four pixels..... | 31 |
| Fig. 2-14. The SAD calculation unit used for different levels. The modules can process a search point of a 16x16 MB within one cycle. (a) The L0 (Level 0) search point module (b) The L1 (Level 1) search point module (c) The L2 (Level 2) search point module..... | 32 |
| Fig. 2-15. (a) The 4x4 SAD Tree used in level 0. (b) The 8x8 SAD Tree used in level 1..... | 34 |
| Fig. 2-16 The search algorithm of reference software [27]..... | 39 |
| Fig. 2-17. The proposed SIFME on two square points, (0, 0) and $frac_pred_mv$, and four triangle point around $frac_pred_mv$ in one quarter-pel distance..... | 39 |
| Fig. 2-18. The proposed hardware architecture of FME..... | 45 |
| Fig. 2-19 Interpolation unit..... | 46 |
| Fig. 2-20. 6-tap 1-D FIR filter..... | 46 |
| Fig. 2-21. The block diagram of IME and FME..... | 54 |
| Fig. 3-1 Block diagram of intra-only encoder..... | 59 |
| Fig. 3-2 Nine modes for intra luma 4x4 and 8x8 prediction..... | 61 |

| | |
|---|-----|
| Fig. 3-3 Four modes for intra luma 16x16 and chroma 8x8 prediction..... | 61 |
| Fig. 3-4. Transmission order of all coefficients in a macroblock predicted by 16x16 intra mode. | 61 |
| Fig. 3-5 The scan order and the syntax symbols of a non-zero 4x4 block... | 62 |
| Fig. 3-6 Decision flow of (a) original three-step algorithm (b) modified three-step algorithm. | 66 |
| Fig. 3-7 Proposed timing schedule for the modified three-step algorithm. . | 67 |
| Fig. 3-8 Proposed architecture of encoder with variable pixel parallelism. . | 75 |
| Fig. 3-9 Pipelined schedule for fast encoder (a) best luma mode is 16x16 (b) best luma mode is 4x4..... | 78 |
| Fig. 3-10 (a) Eight-pixel parallelism intra prediction generator (b) Examples of operations for intra 16x16 DC mode. | 80 |
| Fig. 3-11 Eight-pixel parallelism transform unit. | 80 |
| Fig. 3-12 Inverse transform Unit | 82 |
| Fig. 3-13 (a) Quantization and (b) inverse quantization unit..... | 83 |
| Fig. 3-14 Overall architecture of entropy encoder in H.264 baseline encoder. | 84 |
| Fig. 3-15 The overall architecture of CAVLC encoder..... | 85 |
| Fig. 3-16 An example for nonzero index table: (a) Original 4x4 block and zig-zag scan (b) the initial table after all coefficients are loaded and (c) the updated table after first iteration of leading one detection. | 85 |
| Fig. 3-17 The cycle reduction by adopted techniques. | 87 |
| Fig. 3-18 The layout and its design specification. | 88 |
| Fig. 4-1 Profiles of H.264/AVC | 95 |
| Fig. 4-2 Nine modes for intra 8x8 prediction. | 96 |
| Fig. 4-3 Bi-directional motion estimation..... | 97 |
| Fig. 4-4 Block diagram of CABAC | 98 |
| Fig. 4-5 Flow diagram of arithmetic coding. | 101 |
| Fig. 4-6 Filtering boundary of a macroblock. | 101 |
| Fig. 4-7. System overview of H.264 high profile encoder..... | 104 |
| Fig. 4-8. The scheduling of H.264 high profile encoder..... | 104 |
| Fig. 4-9. The schedule of reconstruction module..... | 106 |
| Fig. 4-10 System architecture of bi-directional motion estimator for H.264 high profile..... | 107 |
| Fig. 4-11. (a)The architecture of intra encoder part. (b)The gate count reduction of intra encoder by proposed techniques. | 109 |
| Fig. 4-12. Intra prediction generator used for intra luma 8x8 modes..... | 111 |

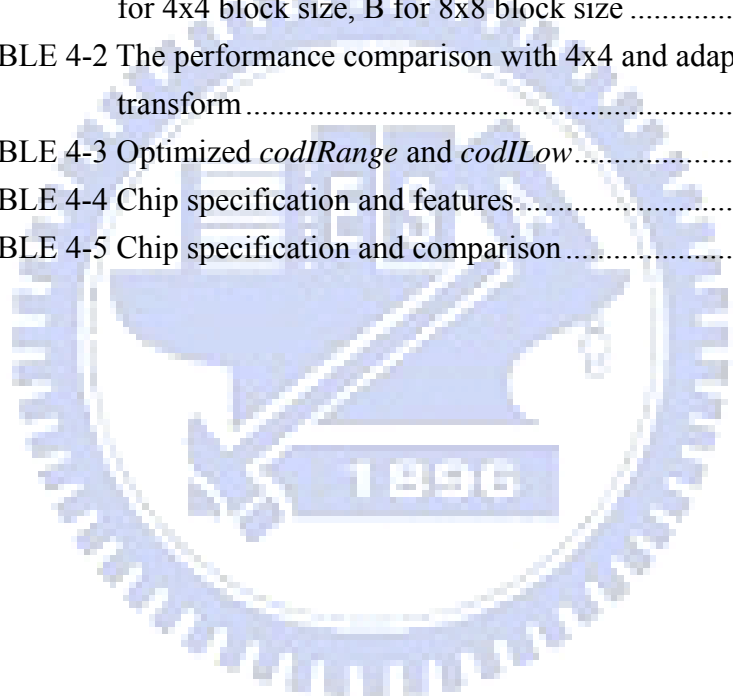
| | |
|---|-----|
| Fig. 4-13 Pipelined schedule of proposed intra prediction generator | 113 |
| Fig. 4-14 Hardware architecture of transform unit | 113 |
| Fig. 4-15 Block diagram architecture of inverse transform unit..... | 115 |
| Fig. 4-16 The architecture of 1-D transform unit | 116 |
| Fig. 4-17 The 4x4 IDCT transform datapath in inverse transform unit. | 116 |
| Fig. 4-18 The 8x8 IDCT transform datapath in inverse transform unit..... | 117 |
| Fig. 4-19 The inverse Hadamard transform datapath in inverse transform unit | 117 |
| Fig. 4-20 Block algorithm of quantization circuits..... | 118 |
| Fig. 4-21. The architecture of motion estimation part and the proposed algorithms. | 121 |
| Fig. 4-22.(a) The memory access reduction of ME (b) the gate count reduction of ME (c) the internal SRAM buffer reduction of ME (d) The trade-off between the number of search point and quality loss. | 121 |
| Fig. 4-23 (a) Original serial chedule of CABAC. (b) Modified parallel algorithm for CABAC..... | 126 |
| Fig. 4-24 Pipelined CABAC encoding flow | 126 |
| Fig. 4-25 Architecture of Binarization | 127 |
| Fig. 4-26 Architecture of Context Modeling..... | 127 |
| Fig. 4-27 Architecture of AC | 127 |
| Fig. 4-28 Architecture of Interval Maintainer..... | 128 |
| Fig. 4-29 Architecture of Renormalization..... | 128 |
| Fig. 4-30. Architecture design of deblocking filter. | 130 |
| Fig. 4-31. Edge processing order for (A) luma edge, and (B) chroma edge 130 | |
| Fig. 4-32. Chip micrograph..... | 133 |
| Fig. 4-33. The power of proposed design and previous works..... | 135 |



List of Tables

| | |
|---|----|
| TABLE 1-1 Profiles of H.264 Specification..... | 6 |
| TABLE 1-2 Levels of H.264 Specification..... | 7 |
| TABLE 2-1 The average mode filtering performance for QCIF and CIF sequences | 23 |
| TABLE 2-2. The average mode filtering performance for 720p sequences | 24 |
| TABLE 2-3. Performance of PMRME for 720p and 1080p sequences..... | 27 |
| TABLE 2-4. Memory and bandwidth requirement equation for each level. The MB_{size} is 16. Besides, SR_{L0} , SR_{L1} , and SR_{L2} are 16, 64, and 256 in respect | 32 |
| TABLE 2-5. Memory and bandwidth requirement is for different frame size. The saving is compared to the direct design [25]. The maximum search range is [-128, 127]..... | 33 |
| TABLE 2-6 Comparison of the IME part with previous designs. | 36 |
| TABLE 2-7 Prediction accuracy of motion vector (mvx and mvy) compared to the full search FME algorithm | 40 |
| TABLE 2-8 Search point comparisons for different algorithms..... | 40 |
| TABLE 2-9 Simulation results of SIFME for different CIF sequences and QPs when compared to the reference software [27] | 42 |
| TABLE 2-10 PSNR and bit rate comparison for different 720p sequences and QPs. Speed up is only the performance in fractional ME part | 42 |
| TABLE 2-11 PSNR & bit rate comparison for different 1080p sequences and QP..... | 43 |
| TABLE 2-12 Simulation comparison with previous works..... | 43 |
| TABLE 2-13 comparisons of number of processing unit (PU) and number of iterative search steps..... | 45 |
| TABLE 2-14 Comparison of the FME part with previous designs..... | 48 |
| TABLE 2-15 PSNR and bitrate change for proposed algorithms compared with full search for 720p sequences..... | 51 |
| TABLE 2-16 PSNR and bitrate change for proposed algorithms compared with full search for 1080p sequences..... | 52 |
| TABLE 2-17 hardware cost comparison for complete H.264 ME accelerator with previous works | 55 |
| TABLE 3-1 H.264/AVC quantization coefficients | 70 |

| | |
|---|-----|
| TABLE 3-2 H.264/AVC de-quantization coefficients | 70 |
| TABLE 3-3 Probability Distribution of All 16x16 Modes in 720p Sequences with 300 I-frames when QP=28 | 72 |
| TABLE 3-4 The performance of modified 3-step algorithm and combined algorithm for 720p video sequences. | 73 |
| TABLE 3-5 The performance of modified 3-step algorithm and combined algorithm for 1080p video sequences. | 74 |
| TABLE 3-6 Zero-block Codeword Table | 84 |
| TABLE 3-7 Gate count table for the encoder for HD1080p at 140MHz..... | 87 |
| TABLE 3-8 Comparison with previous intra encoders..... | 90 |
| TABLE 3-9 Comparison of intra predictor part with the state-of-the-art.... | 91 |
| TABLE 4-1 Quantization parameter table when QP equals twenty-eight: A for 4x4 block size, B for 8x8 block size | 118 |
| TABLE 4-2 The performance comparison with 4x4 and adaptive Hadamard transform | 123 |
| TABLE 4-3 Optimized <i>codIRange</i> and <i>codILow</i> | 128 |
| TABLE 4-4 Chip specification and features..... | 133 |
| TABLE 4-5 Chip specification and comparison..... | 134 |



Chapter 1

Introduction

The video applications exist in our life in every corner such as the analog/digital broadcast TV, the DVD/Blu-ray video disk, and the streaming video through mobile phone or computer. The video applications provide us a lot of fun and convenience. However, the data amount of the video is very huge. If without compression, no storage device can process these data. Therefore, efficient video compression technique has been proposed to reduce the data size and the bandwidth when transmitting these video signals. The H.264 standard is the latest and the most powerful video compression standard, and many applications adopt this standard. In this chapter, we will review the trends of video coding stand and overview H.264 specification. And then, the motivation of the thesis is proposed and followed by the organization and contribution of the thesis.

1.1 Overview of H.264

1.1.1 History of H.264

In 1990s, The ISO (International Standard Organization) MPEG4 standard was proposed for new internet-based video applications while the ITU-T (Telecommunication Standardization Sector) H.263 standard for video compression was widely used in videoconference systems.

MPEG4 and H.263 are standards based on video compression technology, which are developed by two groups. The one is Motion Picture Experts Group (MPEG) and the other is Video Coding Experts Group (VCEG). In 21th century, both groups were

in the final stages of developing a new standard that promises to significantly outperform MPEG4 and H.263. The VCEG group started work on two further development areas: a short-term effort to add extra features to H.263 and a long-term effort to develop a new standard for low bit rate video communications. The long-term effort led to the draft “H.26L” standard, offering significantly better video compression efficiency than previous ITU-T standards. Due to the similarity of the groups, in 2001, the Joint Video Team (JVT) was formed by the experts from MPEG and VCEG group. The major task of JVT is to develop the draft H.26L to be a full international standard. Finally, the two identical standards, ISO MPEG4 Part 10 of MPEG4 and ITU-T H.264, were developed. The official title of the new technique is Advanced Video Coding (AVC); however, it is well known by the ITU document number, H.264 [1].

1.1.2 Introduction of H.264 encoder and decoder

Compared to prior video coding standards, many new techniques are employed in H.264 standard and result in significant improvement on coding performance. The details of these techniques can be found in [2]. Here, we would like to give a brief introduction of the basic concepts of the H.264 encoder and decoder.

In common with earlier standards, the H.264 standard does not explicitly define a CODEC (encoder / decoder pair). Instead, the standard defines the syntax of an encoded video bit stream together with the method of decoding. Therefore, some variations in encoder is allowed as long as the format of encoded bit-stream is correct. Actually, a compliant H.264 encoder and decoder include the functional modules shown in Fig. 1-1 and Fig. 1-2. In these figures, we can find that the decoder system is

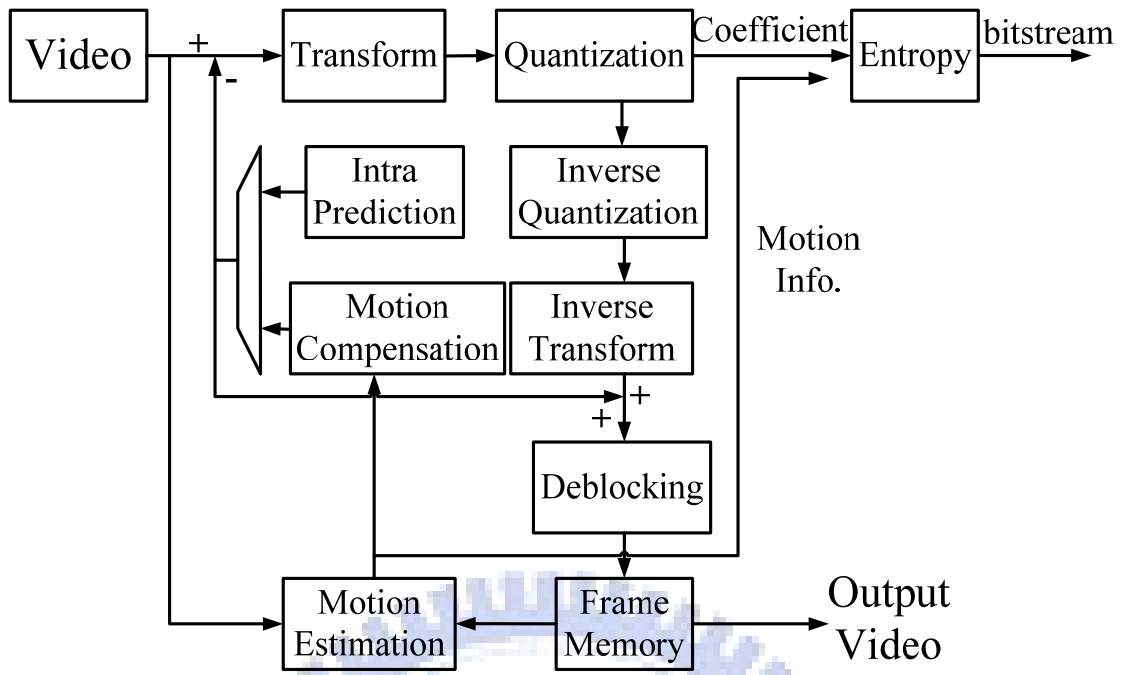


Fig. 1-1 The basic structure of encoder.

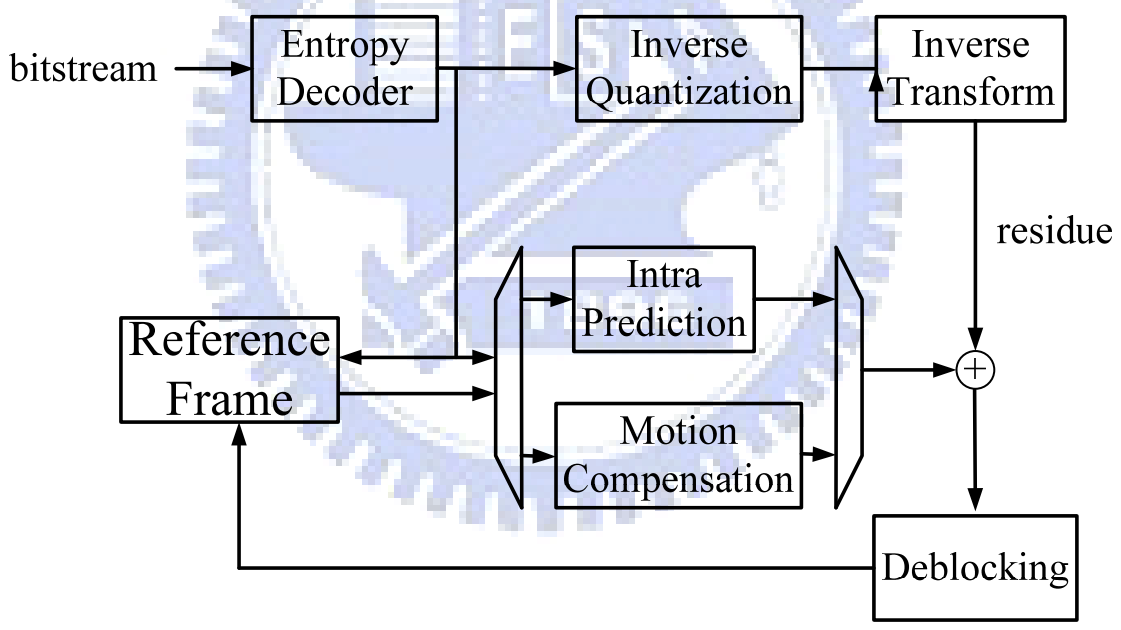


Fig. 1-2 The basic structure of decoder.

a part of the encoder, whereas there is a certain range for considerable variation in the structure.

H.264/AVC also adopts the hybrid video coding scheme which is the same with MPEG 1/2/4. The input video is divided into macroblocks. A macroblock consists of three components, luma and two chroma components. The luma component presents

the brightness and the chroma components show the color information. The input macroblocks are predicted by motion estimation (i.e. inter prediction) or intra prediction. If using inter prediction, the macroblock is predicted by the blocks in encoded frames. For intra prediction, the macroblock is predicted by the pixels from neighbor coded macroblocks. The prediction error, which is the difference between the original and predicted pixels, will be transformed and quantized to reduce the value. Finally, the processed predicted error is sent to entropy coding module to generate the final bit-stream. At the same time, the quantized coefficients are reconstructed by inverse quantization and inverse transform and added by the predicted values. The reconstructed image is filtered and stored in the memory as the reference of next macroblock or next frame.

Comparing with previous standards, the H.264/AVC standard has these changes:

1. H.264/AVC uses in-loop deblocking filter to replace the post-loop filter in previous standards.
2. H.264/AVC supports multiple references frames.
3. The intra prediction provides higher coding efficiency than previous MPEG-4 standard.
4. The Discrete Cosine Transform (DCT) used in previous standards is replaced by the integer transform.

Fig. 1-2 shows the diagram of H.264/AVC decoder. The entropy decoder decodes the quantized coefficients and the motion data. As in the encoder, the prediction pixels are obtained by intra or inter prediction, which is added to the inverse transformed coefficients.

The details of the important modules will be introduced in the next chapters.

1.1.3 Profiles and levels of H.264 specification

H.264/AVC has many applications; however, different applications have different requirements both in terms of functionalities and complexity. In order to satisfy the requirement of all applications as possible, the H.264/AVC specification defines profiles and levels. A profile is a subset of the coding tools. All decoders compliant to a certain profile must support all the tools of the profile and the syntax format. Now, H.264/AVC standard contains seven profiles, whose supporting tools are listed in TABLE 1-1.

However, for many applications, the major difference of requirement between them is the format constrain in resolution and bit-rate, not the supporting tools. Therefore, a level which defines a set of constraints on values of the syntax elements in the bit stream was created for each profile. Each level specifies upper bounds for the bit stream or lower bounds for the decoder capabilities. The difference of all levels is listed in TABLE 1-2. The detailed information on the H.264/AVC profiles and levels can be found in Annex A of [1].

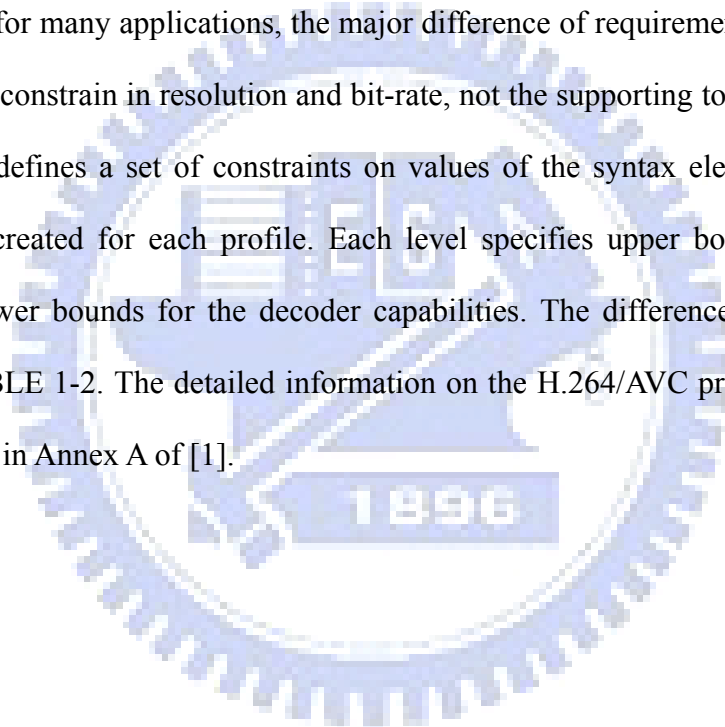


TABLE 1-1 Profiles of H.264 Specification

| Profiles | Baseline | Extended | Main | High | High 10 | High 4:2:2 | High 4:4:4 |
|-------------------------------|----------|----------|------|------|---------|------------|------------|
| I and P Slices | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| B Slices | No | Yes | Yes | Yes | Yes | Yes | Yes |
| SI and SP Slices | No | Yes | No | No | No | No | No |
| Multiple Reference | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Deblocking Filter | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| CAVLC | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| CABAC | No | No | Yes | Yes | Yes | Yes | Yes |
| FMO | Yes | Yes | No | No | No | No | No |
| ASO | Yes | Yes | No | No | No | No | No |
| RS | Yes | Yes | No | No | No | No | No |
| Data Partitioning | No | Yes | No | No | No | No | No |
| Interlaced Coding | No | Yes | Yes | Yes | Yes | Yes | Yes |
| 4:2:0 Format | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| 4:0:0 Format | No | No | No | Yes | Yes | Yes | Yes |
| 4:2:2 Format | No | No | No | No | No | Yes | Yes |
| 4:4:4 Format | No | No | No | No | No | No | Yes |
| 8 Bit Sample Depth | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| 9 and 10 Bit Sample Depth | No | No | No | No | Yes | Yes | Yes |
| 11 to 14 Bit Sample Depth | No | No | No | No | No | No | Yes |
| 8x8 Transform | No | No | No | Yes | Yes | Yes | Yes |
| Quantization Scaling Metrics | No | No | No | Yes | Yes | Yes | Yes |
| Separate Cb and Cr QP control | No | No | No | Yes | Yes | Yes | Yes |
| Separate Color Plane Coding | No | No | No | No | No | No | Yes |
| Predictive Lossless Coding | No | No | No | No | No | No | Yes |

TABLE 1-2 Levels of H.264 Specification

| Level Number | Max macroblocks per second | Max frame size (macroblocks) | Max video bit rate (VCL) for all profiles | Vertical MV component range |
|--------------|----------------------------|------------------------------|---|-----------------------------|
| 1 | 1485 | 99 | 64-256 kbits/s | [-64,+63.75] |
| 1b | 1485 | 99 | 128-512 kbits/s | [-64,+63.75] |
| 1.1 | 3000 | 396 | 192-768 kbits/s | [-128,+127.75] |
| 1.2 | 6000 | 396 | 384-1536 kbits/s | [-128,+127.75] |
| 1.3 | 11880 | 396 | 768-3072 kbits/s | [-128,+127.75] |
| 2 | 11880 | 396 | 2-8Mbits/s | [-128,+127.75] |
| 2.1 | 19800 | 792 | 4-16Mbit/s | [-256,+255.75] |
| 2.2 | 20250 | 1620 | 4-16Mbits/s | [-256,+255.75] |
| 3 | 40500 | 1620 | 10-40Mbits/s | [-256,+255.75] |
| 3.1 | 108000 | 3600 | 14-56Mbits/s | [-512,+511.75] |
| 3.2 | 216000 | 5120 | 20-80Mbits/s | [-512,+511.75] |
| 4 | 245760 | 8192 | 20-80Mbits | [-512,+511.75] |
| 4.1 | 245760 | 8192 | 50-200Mbits | [-512,+511.75] |
| 4.2 | 522240 | 8704 | 50-200Mbits | [-512,+511.75] |
| 5 | 589824 | 22080 | 135-540Mbits | [-512,+511.75] |
| 5.1 | 983040 | 36864 | 240-960Mbits | [-512,+511.75] |

1.2 Motivation of Thesis

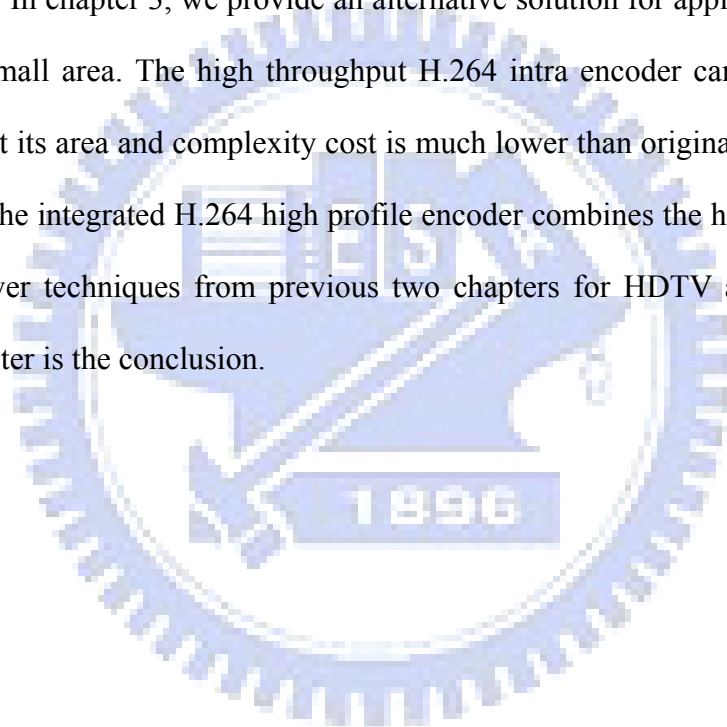
H.264 has been adopted as the major coding standard in recently popular high definition video because of its excellent coding efficiency. Due to its high computational loading, ASIC implementation of H.264 encoder is preferred. Therefore, several implementations have been developed [3]-[5], but their performance is limited to baseline 720p [3][4] or SDTV [5]. The main stream 1080p application presents a series of new design challenges in throughput, cost and power because of at least a 4X higher complexity than in the 720p baseline.

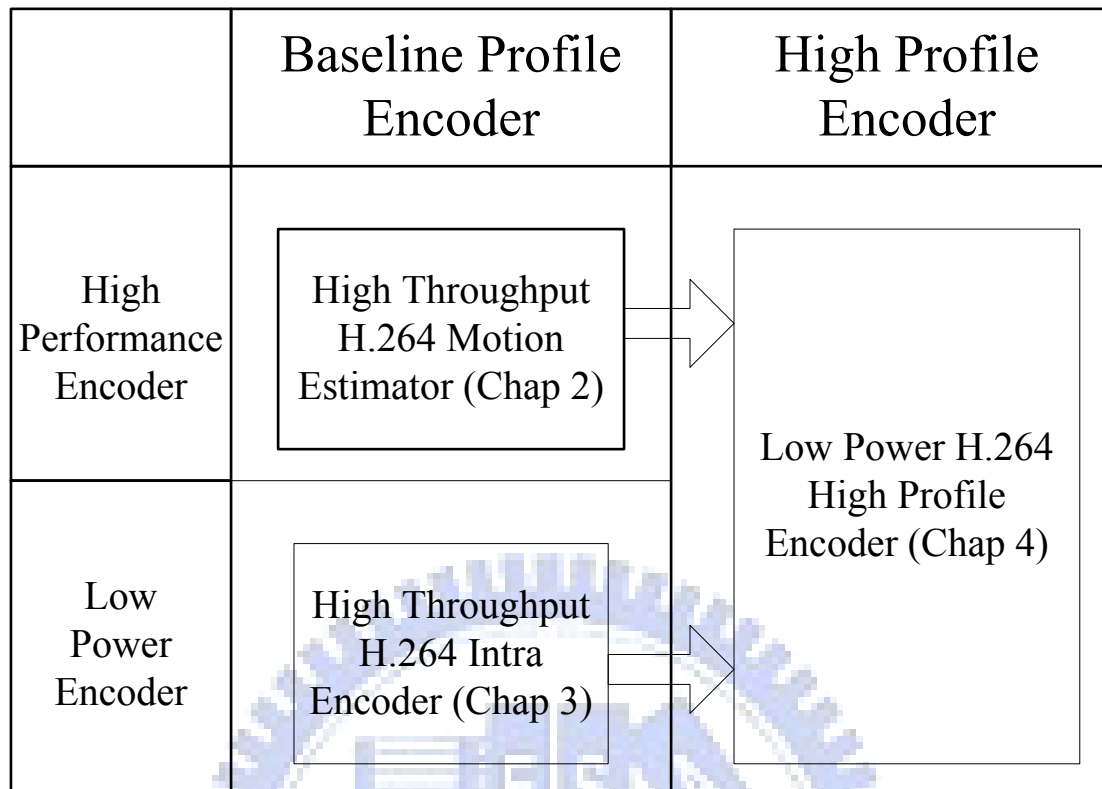
Thus, in this thesis, we first propose algorithms and architectures for our H.264 encoder to support 1080p resolution without significant quality loss and hardware

overhead. And then, we integrate the whole design with the new coding tools of high profile to support the latest high definition video applications.

1.3 Organization and Contribution of Thesis

Fig. 1-3 shows the organization and contribution of the dissertation. In chapter 2, we propose a high performance H.264 motion estimator which can support 1080p video with 60fps and the search range up to ± 128 without area and throughput overhead [6]. In chapter 3, we provide an alternative solution for application with low power and small area. The high throughput H.264 intra encoder can support 1080p resolution but its area and complexity cost is much lower than original H.264 encoder [7]. Finally, the integrated H.264 high profile encoder combines the high performance and low power techniques from previous two chapters for HDTV applications [8]. The last chapter is the conclusion.





All modules and encoder support 1080p video encoding in real time

Fig. 1-3 Organization of this thesis.



Chapter 2

High Performance H.264 Motion Estimator for HDTV

Motion estimation (ME) part is the most important component in H.264 encoder. In which, the variable block size integer-pel motion estimation (IME) and its improved fractional-pel ME (FME) not only contributes a lot for coding efficiency but also dominates the computational loading of the whole encoding process. Thus, various VLSI realizations of ME have been proposed to speed up the process. In this chapter, we first introduce the motion estimation algorithms of H.264. Besides, we review the previous works and define the problems when extending the supporting resolution to full high definition (HD) size and large search range (SR). And then, we introduce our proposed algorithms and architectures in H.264 motion estimation part for high definition (HD) applications: The first technique, mode filtering (MF), is used to speed up the throughput of whole system. And then the parallel multi-resolution motion estimation (PMRME) reduces the most complexity and memory bandwidth for variable block size motion estimation. Finally, the single iteration fractional motion estimation (SIFME) minimizes the hardware cost and latency for one-quarter fractional motion estimation.

2.1 Introduction to H.264 Motion Estimation

2.1.1 System overview for H.264 motion estimation

The H.264 standard adopts the general block-based motion estimation algorithm, which compares the block-based coding data with the reference data to find the best motion vectors. The motion estimation flow of H.264 is illustrated in Fig. 2-1. The current block is compared with the reference data in the search range of previous frames, and the best integer motion vectors are decided by integer motion estimation module. And then the related data is processed by fractional motion estimation modules for refinement. Finally, the residue data which is the difference between current block and best reference block is generated for further coding.

Although H.264 standard uses the common block matching algorithm, it has some new features which differ from previous video standards and will be introduced in next sections:

- Variable block size motion estimation
- Quarter-pel fractional motion estimation
- Multiple reference frames

2.1.2 Variable block size motion estimation (VBSME)

The H.264 standard adopts hierarchical variable block size (VBS) motion estimation technique to improve the prediction accuracy. Fig. 2-2 shows the block selection procedure and seven possible block size modes for VBSME. In the first step, the best block size is chosen from mode 1 to mode 4 as shown in Fig. 2-2. If the 8x8 mode is preferred, all blocks are split into smaller blocks from mode 4 to mode 7 in the second step. Therefore, there are many combinations of chosen modes in a macroblock. Unlike the previous MPEG 1/2/4 standards which only support 16x16 or 8x8 block matching units, the VBS technique provides flexibility for different

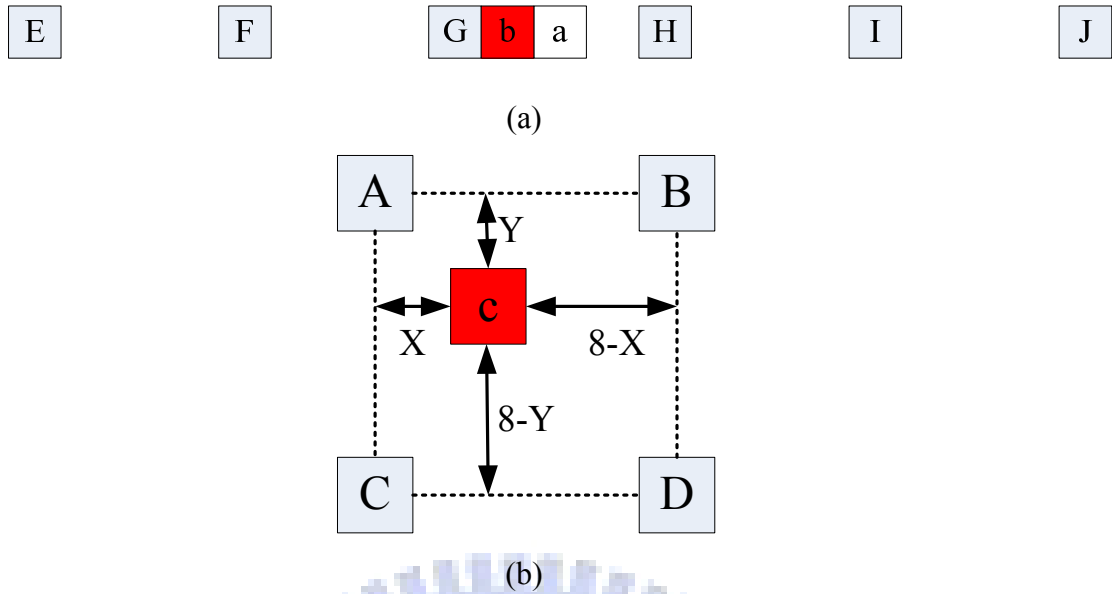


Fig. 2-3 Integer samples and fractional sample positions for (a) luma and (b) chroma interpolation.

2.1.3 Quarter-pel fractional motion estimation

H.264 standard supports the motion estimation resolution to quarter pixel in luma components and one-eighth pixel in chroma parts. The sub-pixel motion estimation technique can raise the prediction accuracy and reduce bit-rate. The interpolation schedules to generate the half and quarter pixels are presented in Fig. 2-3 (a), The half luma pixel is generated by the 6-tap filter:

$$a = E-5*F+20*G-20*H+5I-J+16/32 \quad (1)$$

The quarter pixel is generated by the average of integer and half pixels:

$$b = (E+F+1)/2 \quad (2)$$

As for the chroma part, the sub-pixel in Fig. 2-3 (b) is calculated by the interpolation equation:

$$c = ((8-x)*(8-y)*A+x*(8-y)*B+(8-x)*y*C+x*y*D+32)/32 \quad (3)$$

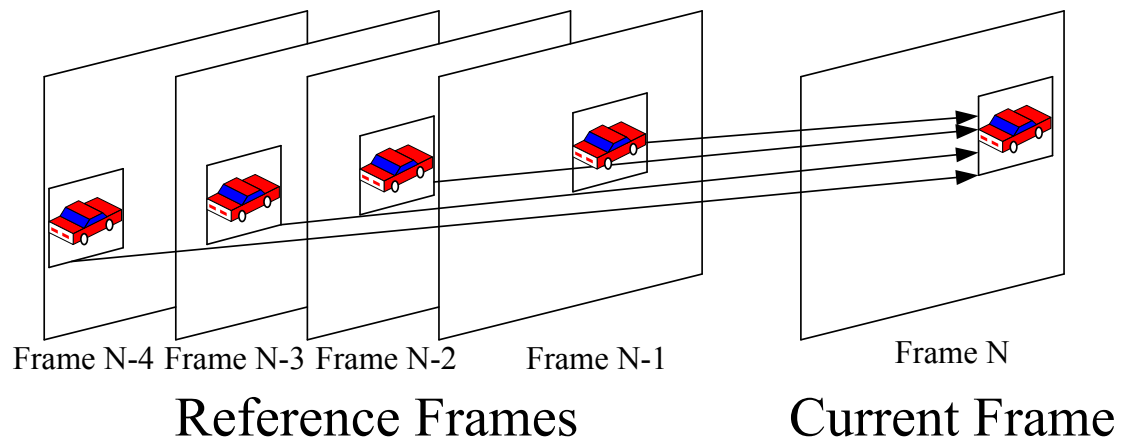


Fig. 2-4 Multiple references in motion estimation.

2.1.4 Multiple reference frames

H.264/AVC standard supports multiple reference frames in motion estimation as shown in Fig. 2-4. At most five frames can be used to predict current block. By this technique, the coding efficiency and prediction accuracy can be further improved. However, the computational complexity is proportional to the number of reference frame.

2.1.5 Skip mode

Because these new techniques in motion estimation stage dominate the computational loading and power of the H.264 encoding process, the most efficient way to lower the complexity and power of H.264 encoder is to skip the prediction procedure of a macroblock and simply use the information of coded macroblock directly.

In H.264/AVC, if the following conditions are matched, the macroblock will be skipped and encoded as skip mode:

1. The chosen block type is 16x16.
2. The best motion vector equals the predicted motion vector (MVP).
3. The chosen reference frame is the previous frame.

4. All coefficients are zero after transform and quantization.

2.2 Design Challenges and Paper Survey

2.2.1 Design challenges

As mentioned above, the VBS integer-pel motion estimation (IME) and its improved fractional-pel ME (FME) modules require the most computational resources in the whole encoding process of H.264 standard. Thus many VLSI realizations of ME have been proposed to speed up the process [9]-[18]. However, most of them are only applicable for standard definition (SD) size or below. For high definition (HD) video applications that requires large search range up to $[-128, 127]$ or even larger, direct extension with previous approaches will consume too large area cost, buffers, bandwidth and cycles. To support large search range, many fast integer ME algorithms have been proposed [19]-[24]. However, most of them are not suitable for hardware implementation because of its irregular data flow. Besides, most of these approaches only consider IME or FME only without exploiting their relationship, which may result in extra computation cost.

To solve the above problems, we present an efficient ME architecture suitable for HD videos by various design techniques, including a mode filtering algorithm to jointly reduce the IME and FME computations, a parallel multi-resolution ME (PMRME) for large search range IME, and a single iteration FME (SIFME) to achieve the lower cycle count. The cycles are reduced by hardware parallelism and algorithm modification (PMRME and SIFME). Furthermore, we lower requirements of bandwidth and buffer by reusing data within IME as well as between IME and FME. The video quality loss is low by exploiting unequal distribution of motion vectors. With these approaches, we can save more than half of area, bandwidth and

buffer cost when compared to previous designs.

2.2.2 Paper survey

For fast IME of H.264 standard, various approaches have been proposed [14]-[24] but few can be readily applicable to large search range as used in HDTV. The large search range requirement will result in longer execution cycles as well as large buffer and high memory access. Previous designs with $[-63, +64]$ search range [25][26] use the full search method and thus occupy large area cost. To solve these problems, one promising approach is the multi-resolution ME [14]. In [14], they use three hierarchical levels for search and refine the motion vectors from the coarse level to the finest level. However, the motion vector found in the higher level needs to be further refined in the lower level. It implies the search is a sequential process that will increase the cycle counts, and thus decrease the hardware utilization and throughput. Besides, a full search range sized buffer is still needed because of the dependency between the three hierarchical levels. And then, the required bandwidth is still too large because of poor data reuse of the refinement process. In [15], a modified three-step algorithm is used to decrease the search points for low power, but still consumes large area cost and memory. [16] also uses the subsampling techniques to reduce the hardware cost; however, the two-stage architecture results in longer cycle counts. In [18], the two-stage flow and the irregular search range cause the difficulty of external data transfer.

For fast FME, most of them follows the two-step approaches as in reference software [27] which needs total 17 search points for fractional ME. Although this algorithm is suitable for hardware [28], it has two drawbacks. First, the nine search points in each step result in area-costly nine processing units (PUs) for hardware implementation. The second drawback is that it needs two iterative search loops of

interpolation and Hadamard transform to calculate the SATD cost.

To speedup FME, many fast FME [29]-[34] algorithms are proposed to speed up the process. However, these algorithms [29]-[32] are software-oriented, with irregular data flow and thus are not suitable for hardware design. Our previous work [33] is more suitable for hardware and can reduce the processing unit from nine to five to save hardware cost. But all these algorithms suffer from long computation cycles due to the two iterative search loops, one for half-pels and one for quarter-pels. On the other hand, single iteration algorithms like [36][37] have bad performance due to poor interpolation accuracy. The design in [38] increases the throughput by the cost of large area and memory bandwidth. In summary, the hardware implementations of these fast algorithms only reduce the processing element but degrade the quality a lot or do not reduce the total cycle count. These problems will pose strict limits on the HD video applications since FME will take more cycles than IME and thus will dominate the whole pipelining cycle time.

2.3 Mode Filtering Algorithm

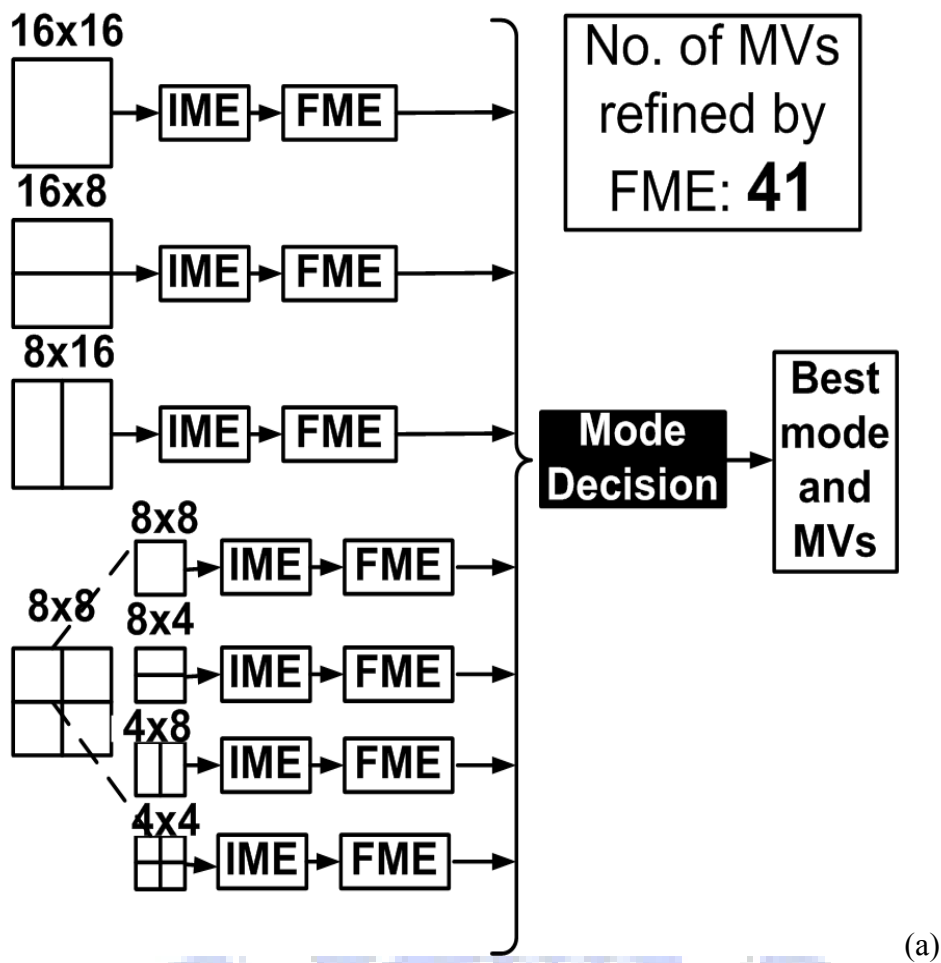
2.3.1 Introduction to mode filtering

Fig. 2-5 (a) presents the general flow of IME and FME in the reference software [27] that IME sends the motion vector to FME for refinement. After all possible modes and motion vectors are generated, the best mode and its motion vectors are chosen in the final step of FME. Thus, the IME and FME module both process 41 MVs.

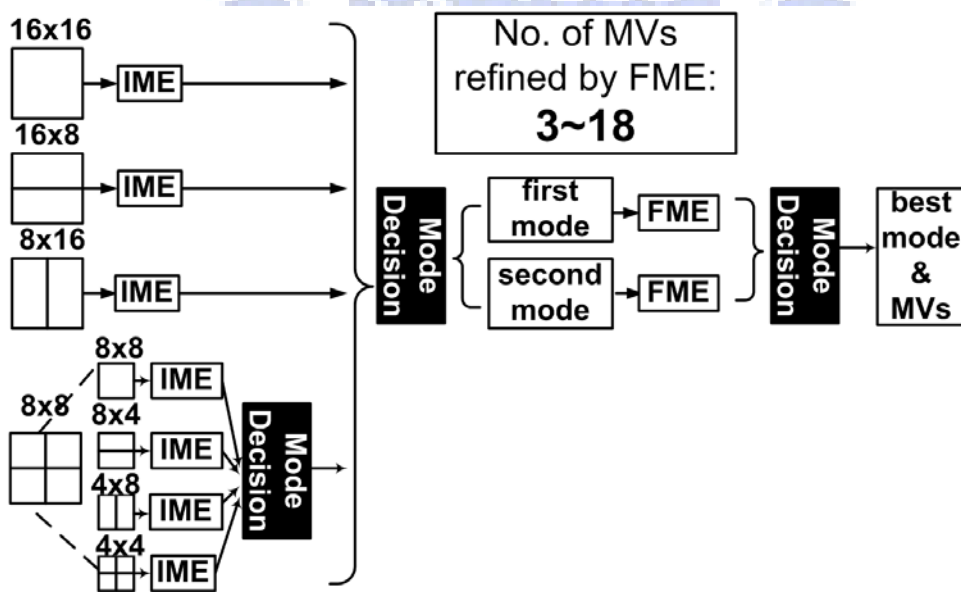
To reduce the complexity, we select the two best modes instead of all modes for FME refinement as shown in Fig. 2-5 (b). One mode is chosen from mode 1 to mode 3 in Fig. 2-2, and the other mode is selected from mode 1 to mode 7. With this, only 3 to 18 MVs instead of 41 MVs are computed in FME, which saves 60% to 70%

computing cycles. In [28], a similar concept but more complex procedure has been proposed. Our method can achieve better quality and lower cycle count than that in [28] because we only select two instead of three candidates and only the best candidate for the 8x8 and smaller subblock case is considered in the final best mode selection. Besides, the method also increases the overall ME pipelining efficiency because it can reduce the cycle count of FME to be similar to that of the IME stage.





(a)



(b)

Fig. 2-5 (a) The original coding flow between IME and FME (b) Mode filtering algorithm.

2.3.2 Simulation result of mode filtering

We test the mode filtering algorithm in three different sizes of video: QCIF, CIF and 720p test sequences to see the performances under different conditions. The test sequences with QCIF and CIF resolution are ‘akiyo’, ‘foreman’, and ‘mobile’, which are low motion, medium motion, and high motion sequences respectively. For 720p resolution, the test sequences are ‘Stockholm’, and ‘park_run’. The search ranges are 8, 16 and 32 for QCIF, CIF and 720p sequences respectively. The reference software is JM 9.0 [27] without rate-distortion optimization (RDO).

2.3.2.1 Performance of QCIF/CIF sequences

Fig. 2-6 and Fig. 2-7 show the result of mode filtering algorithm for the QCIF and CIF sequences. For small size sequences, we can observe that mode filtering method has similar performance as reference software.

TABLE 2-1 presents the average performance of this algorithm of QCIF and CIF sequences respectively. In these results, we find out that the average bit-rate increasing can be only 0.54% and 1.30% and the PSNR degradation is only 0.11db, which performs well.

The performances for CIF sequences are better than that for QCIF because the mode filtering technique will filter most of the small block modes while these small block modes are more preferable in small size sequences. Therefore, mode filtering has better performance for CIF sequences because CIF sequences more prefer larger block sizes than QCIF.

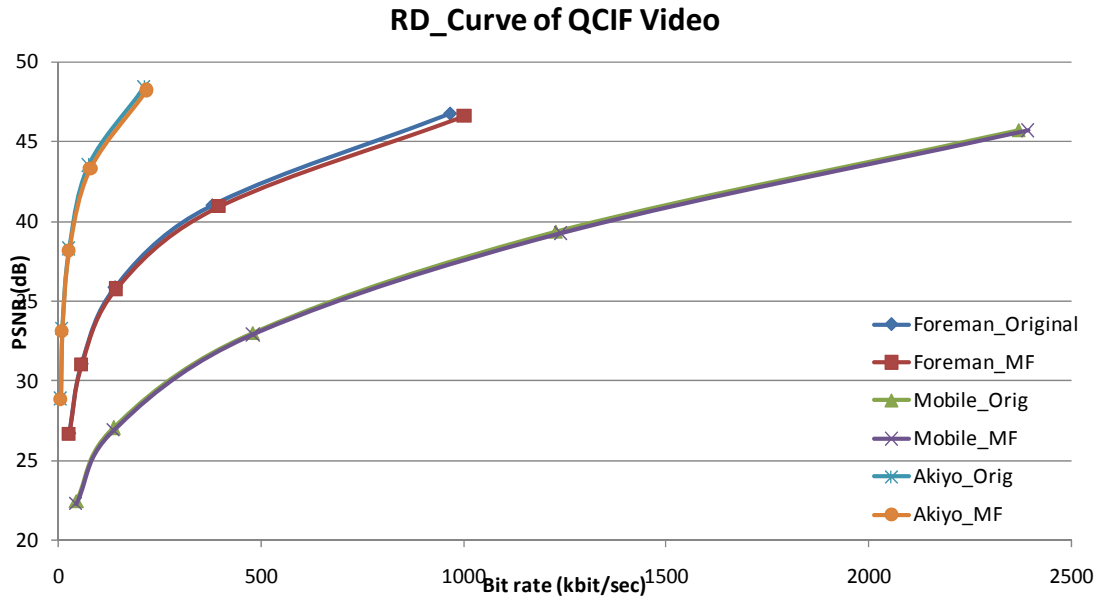


Fig. 2-6 The rate-distortion curves of QCIF sequences.

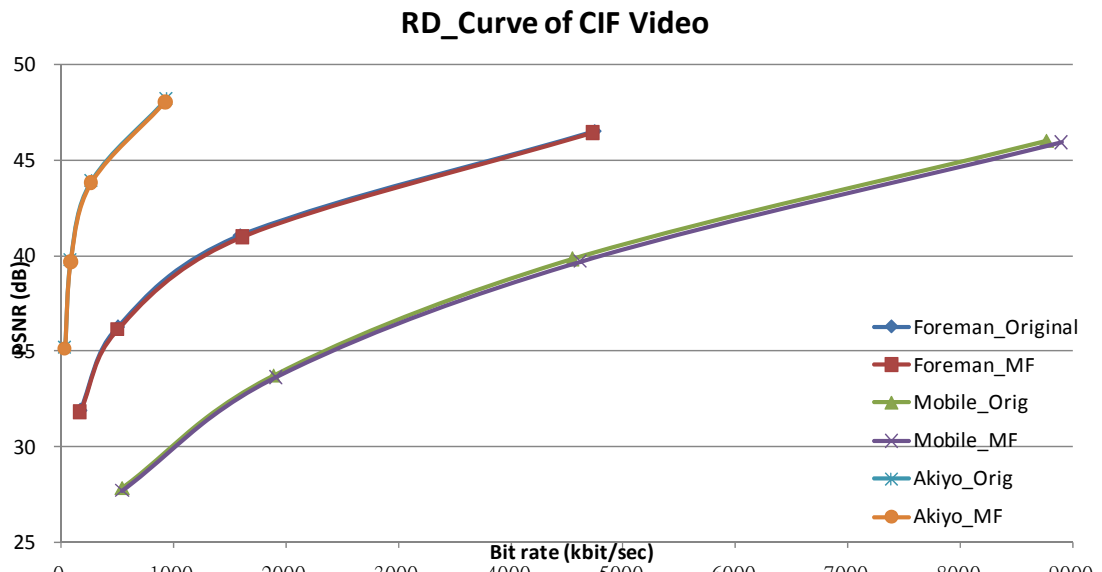
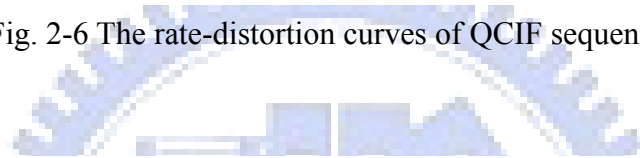


Fig. 2-7. The rate-distortion curves of CIF sequences.

TABLE 2-1 The average mode filtering performance for QCIF and CIF sequences

| | | QCIF | CIF |
|---------|---------------------|-------|-------|
| QP14 | <i>PSNR (dB)</i> | -0.13 | -0.12 |
| | <i>Bit-rate (%)</i> | 2.52 | 0.50 |
| QP21 | <i>PSNR (dB)</i> | -0.14 | -0.10 |
| | <i>Bit-rate (%)</i> | 3.96 | 1.59 |
| QP28 | <i>PSNR (dB)</i> | -0.12 | -0.12 |
| | <i>Bit-rate (%)</i> | 1.45 | 0.68 |
| QP35 | <i>PSNR (dB)</i> | -0.15 | -0.10 |
| | <i>Bit-rate (%)</i> | -0.67 | -0.63 |
| Average | <i>PSNR (dB)</i> | -0.11 | -0.11 |
| | <i>Bit-rate (%)</i> | 1.30 | 0.54 |

2.3.2.2 Performance of 720p sequences

Fig. 2-8 shows the rate-distortion curves of 720p sequences. TABLE 2-2 lists the average performance. For 720p sequences, we find that mode filtering provides very good performance with very little PSNR dropping for low QP situation. However, as the QP increases, the bit rate also increases rapidly and reaches 2.19% increasing under QP36.

For small size (QCIF and CIF) video, the bit-rate overhead is smaller for high QP case (the low bit-rate condition). It is because the mode filtering algorithm prefers larger block size which is the better choice for the low bit-rate condition by reduced MV bit. For 720p sequences, the frame contents are smoother because of the characteristics of high definition; thus, large block size is preferred in 720p sequences and results in better performance of mode filtering than that in QCIF and CIF video.

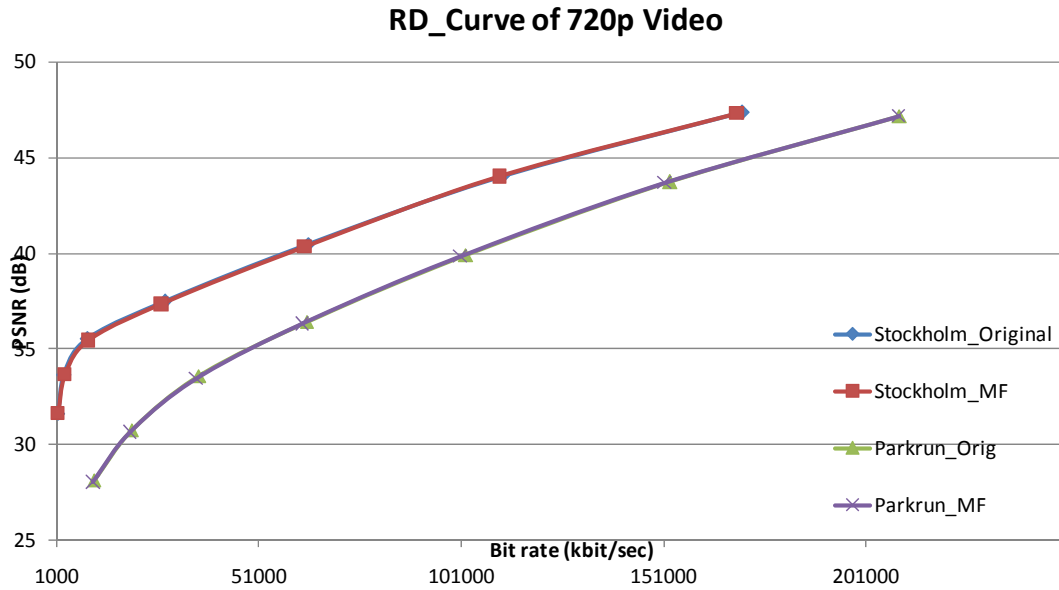


Fig. 2-8 The rate-distortion curves of 720p sequences.

TABLE 2-2. The average mode filtering performance for 720p sequences

| | | 720p |
|---------|---------------------|-------|
| QP12 | <i>PSNR (dB)</i> | -0.03 |
| | <i>Bit-rate (%)</i> | -0.45 |
| QP16 | <i>PSNR (dB)</i> | -0.04 |
| | <i>Bit-rate (%)</i> | -0.71 |
| QP20 | <i>PSNR (dB)</i> | -0.11 |
| | <i>Bit-rate (%)</i> | -1.37 |
| QP24 | <i>PSNR (dB)</i> | -0.15 |
| | <i>Bit-rate (%)</i> | -2.30 |
| QP28 | <i>PSNR (dB)</i> | -0.09 |
| | <i>Bit-rate (%)</i> | -0.42 |
| QP32 | <i>PSNR (dB)</i> | -0.07 |
| | <i>Bit-rate (%)</i> | 1.23 |
| QP36 | <i>PSNR (dB)</i> | -0.06 |
| | <i>Bit-rate (%)</i> | 2.19 |
| Average | <i>PSNR (dB)</i> | -0.08 |
| | <i>Bit-rate (%)</i> | -0.26 |

2.4 Integer Motion Estimation Module : Parallel

Multi-Resolution Motion Estimation (PMRME) [35]

2.4.1 Algorithm of PMRME

PMRME includes three levels and all of them are independent to each other, as illustrated in Fig. 2-9.

In the coarsest level, level 2, the search range (SR) is the largest, $[-128 \sim 127]$, and centered on the original point $(0,0)$. This enables the regular memory reuse between successive MB processing as used in most of ME designs [39]. This level uses the 16:1 sampling and thus we only choose the 16x16 mode (mode 1 in Fig. 2-2) since other modes will contain too fewer pixels for SAD calculation and may result in poor mode decision.

In level 1, the SR is reduced to $[-32 \sim +31]$ and also centered on $(0,0)$ for memory reuse. This level uses the 4:1 sampling and thus we only choose the 16x16 to 8x8 mode (mode 1 to 4 in Fig. 2-2) for the same reason as level 2.

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

In the three parallel levels, the level 2 provides a large search range for high motion blocks with coarse precision. It is useful for very high motion blocks, and can find a good enough though rough motion vector candidate. Also, the level 1 can provide a

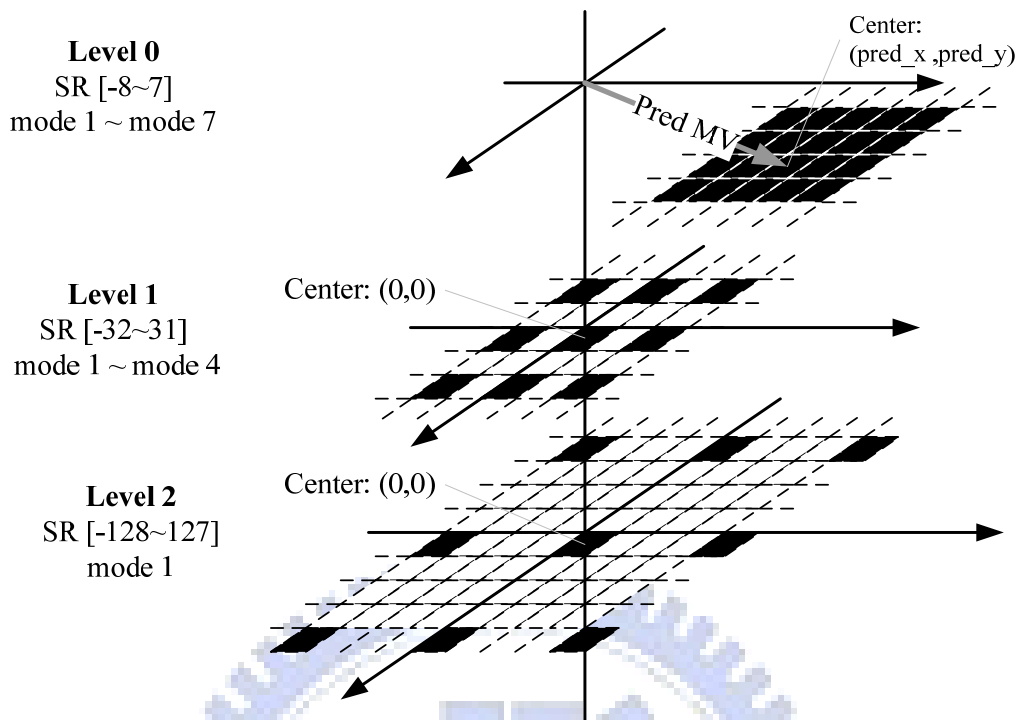


Fig. 2-9. The three-level new multi-resolution algorithm.

medium search range but a finer MV precision. With these two large search levels, the motion search algorithm of level 0 can converge to the true motion vector quickly by effects of MVP. If only the level 0 is used, it is difficult to trace the high motion blocks because the MVP cannot follow up the real motion effectively in this case.

2.4.2 Performance of PMRME

TABLE 2-3 shows the video quality of PMRME algorithm for 720p and 1080p sequences respectively.

Rate-distortion optimization (RDO) is not used, and only the first frame is intra frame. The search range (SR) is [-128, 127]. All simulation results are compared with reference software JM9.0 [27]. TABLE 2-3 shows the average performance under different QPs. For 720p video, four test sequences are used: Stockholm, park_run, and shields. The frame rate is 30 and 300 frames are coded.

TABLE 2-3. Performance of PMRME for 720p and 1080p sequences

| QP | | 720p | 1080p |
|---------|--------------------------|---------|-------|
| QP16 | PSNR inc.(db) | -0.0025 | 0.00 |
| | <i>Bit rate inc. (%)</i> | -1.02 | -0.49 |
| QP20 | PSNR inc.(db) | 0 | -0.01 |
| | <i>Bit rate inc. (%)</i> | -0.49 | -0.44 |
| QP24 | PSNR inc.(db) | -0.0075 | -0.03 |
| | <i>Bit rate inc. (%)</i> | -0.335 | -0.40 |
| QP28 | PSNR inc.(db) | -0.005 | -0.06 |
| | <i>Bit rate inc. (%)</i> | -0.2 | 0.40 |
| QP32 | PSNR inc.(db) | -0.01 | -0.06 |
| | <i>Bit rate inc. (%)</i> | 1.56 | 1.68 |
| Average | PSNR inc.(db) | -0.005 | -0.04 |
| | <i>Bit rate inc. (%)</i> | -0.017 | 0.15 |

RD-Curve for 720p sequences

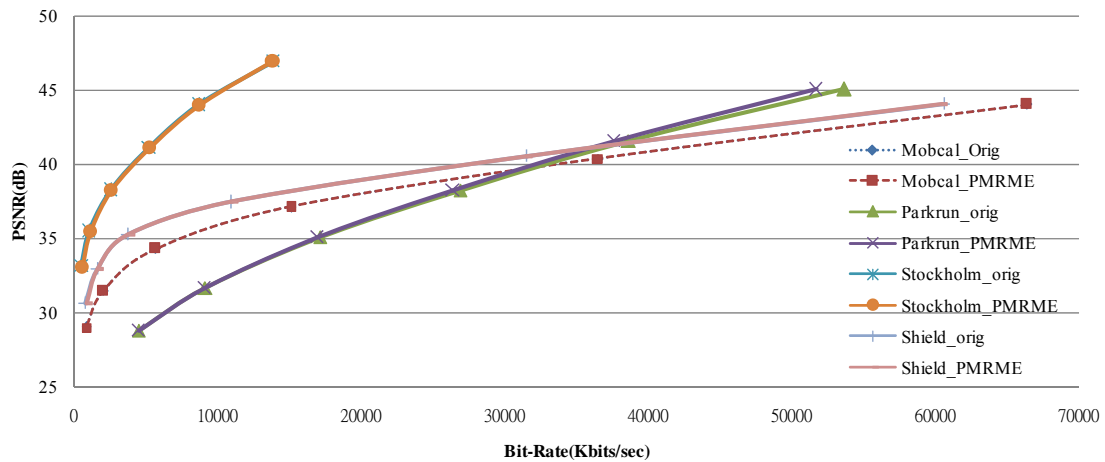


Fig. 2-10 The rate-distortion curves of 720p sequences.

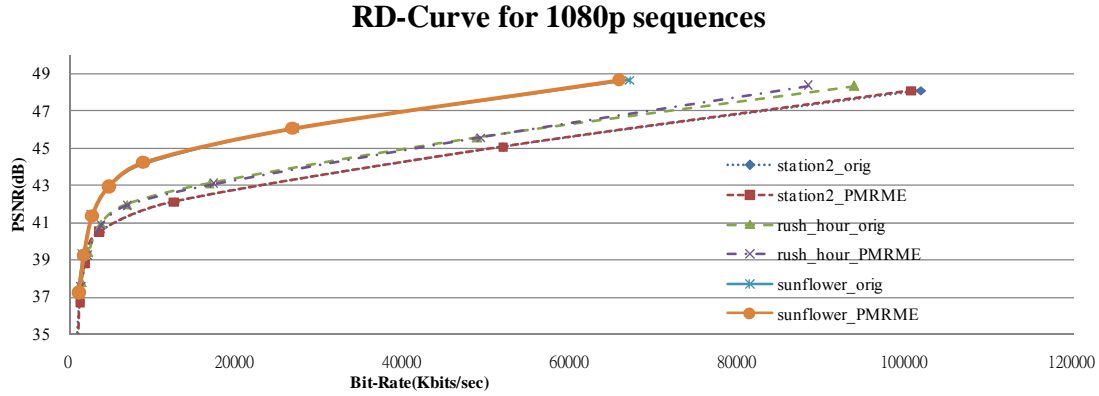


Fig. 2-11 The rate-distortion curves of 1080p sequences.

For 1080p video, three test sequences include: station2, rush_hour, and sunflower. The number of testing frames is 100. We should note that the 1920x1080 image is truncated to 1920x1072 to fit the multiples of 16.

Fig. 2-7 and Fig. 2-8 present the rate-distortion curve of 720p and 1080p video sequences, respectively. The results show the PMRME algorithm can achieve the similar video quality as the full search algorithm. However, the bit-rate overhead is larger under high OP because the video quality of reconstructed video will be worse for high QP and the worse reference will mislead the subsampling method. For 720p sequences, PSNR loss is only 0.005dB and the bit-rate decreasing is 1.28% in average. As for 1080p sequences, it has 0.04dB PSNR loss and up to 0.15% of bit-rate decreasing in average. The average PSNR quality loss is negligible and the bit-rate in some cases is decreasing because PMRME also prefers larger block size.

2.4.3 Architecture of PMRME

Fig. 2-12 shows the proposed IME architecture. All three levels can be computed in parallel. A 16x16 current block is shared by three levels. The memory size and bandwidth for three reference frame buffers are listed in TABLE 2-4 and TABLE 2-5.

The bit width of memory buffer of level 1 and level 2 are truncated while that of the level 0 is not. The reason for this is that the level 0 data can be reused by the following FME hardware if the best MV falls in the level 0. TABLE 2-4 presents the equation of buffer size and the memory access requirement for each level and direct implementation [25]. The MBSIZE in the table is 16. Besides, SRL0, SRL1, and SRL2 are respective 16, 64, and 256. We should note that the buffer size for direct implementation is the search range size. As for level 0, the buffer size is a little larger than the search range because it includes the neighbor pixels for FME interpolation. But in the case of level 1 and level 2, the memory size is only one-fourth and one-sixteen of their search range by the subsampling techniques. Besides, the bit-truncation technique also reduces 25% to 37.5% buffer size if two or three bits are truncated. As for the memory bandwidth, by the level C data-reuse scheme in [39], the direct implementation needs to update $(SR+MBSIZE-1)*16$ pixels. Therefore, the larger search range results in the lower proportion of update rate. Thus, only $16/(64+16) = 20\%$ data in level 1 SRAM should be updated when the coding MB changes with above approach. As for level 2, only $16/(256+16) = 5.88\%$ data should be updated. TABLE 2-5 shows the real buffer size and memory bandwidth requirement for 720p and 1080p video. The proposed algorithm can save over 91.91% buffer in the 720p case and 55% bandwidth in the 1080p case by subsampling and bit-truncation when comparing to [18] that also uses level C data-reuse scheme. If the bus width is 128 bits, it only needs 121 cycles per MB to transfer the required data from external memory to SRAM.

In this architecture, all computations are decomposed as the combinations of 4x4 blocks. The basic processing unit is the 4p-SAD (four-pixel SAD) unit which can process the SAD of four pixels as depicted in Fig. 2-13. With this, every level can be easily implemented by regularly composed SAD units. As Fig. 2-12 presents, L0

(level 0) has one search point module which can process a search point within one cycle so that the level 0 with search range $[-8, +7]$ can finish the full search within 256 cycles. In the same manner, level 1 and level 2 has four and 16 search point modules, which mean the level 1 and level 2 can process four and 16 search points in parallel. Therefore, level 1 and level 2 can process 1024 and 4096 search points within 256 cycles by the parallelization techniques.

Fig. 2-14 shows the detailed architecture of search point modules of each level. Fig. 2-14 (a) shows the “L0 search point module”, which is consisted of four row SAD modules. Each row SAD module contains 16 4p-SAD units. Thus, the L0 search point module includes 64 4p-SAD units in total to generate the total SAD cost of a 16x16 MB. As for level 1 with 2:1 subsampling, the number of search point is 1024. Furthermore, since the current buffer for level 1 is also subsampled, only 64 pixels are compared in current MB. Therefore, L1 (Level 1) search point modules in Fig. 2-14(b) only needs 16 4p-SAD units, which is quarter of that in level 0. In order to keep the cycle count of level 1 as the same as that of level 0, we use four L1 search point modules. Thus, four search points in level 1 can be processed in parallel with the same current block. With above arrangement, the total hardware cost of level 1 is the same as that of level 0, 64 4p-SAD units. Similar design considerations are also applied to level 2. Thus, in level 2, the L2 (Level 2) search point module in Fig. 2-14(c) only needs 4 4p-SAD units so that we use 16 L2 search point modules to compute 16 search points in parallel. In summary, all these levels have 64 4p-SAD units respectively to balance the computation cycle of each level to be the same 256 cycles.

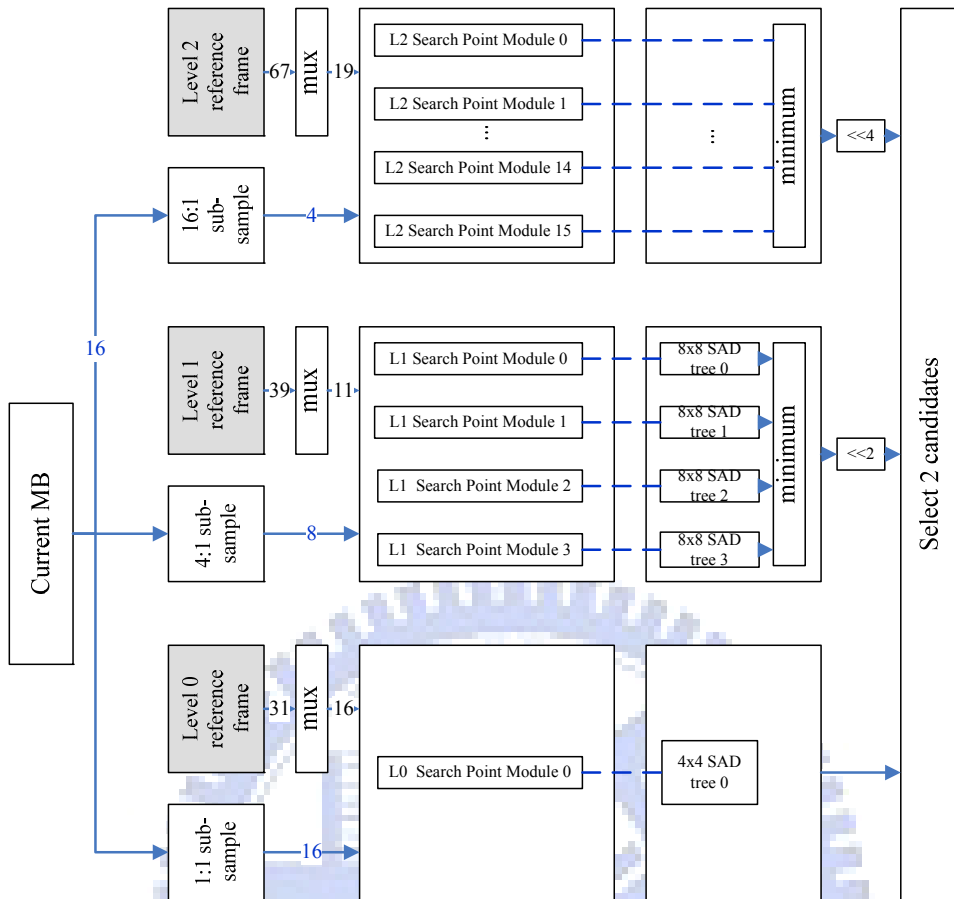


Fig. 2-12. The proposed architecture of IME stage.

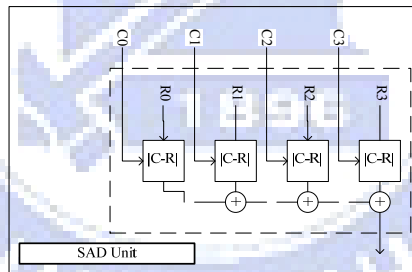


Fig. 2-13. Basic 4p-SAD unit can accumulate the SAD of four pixels.

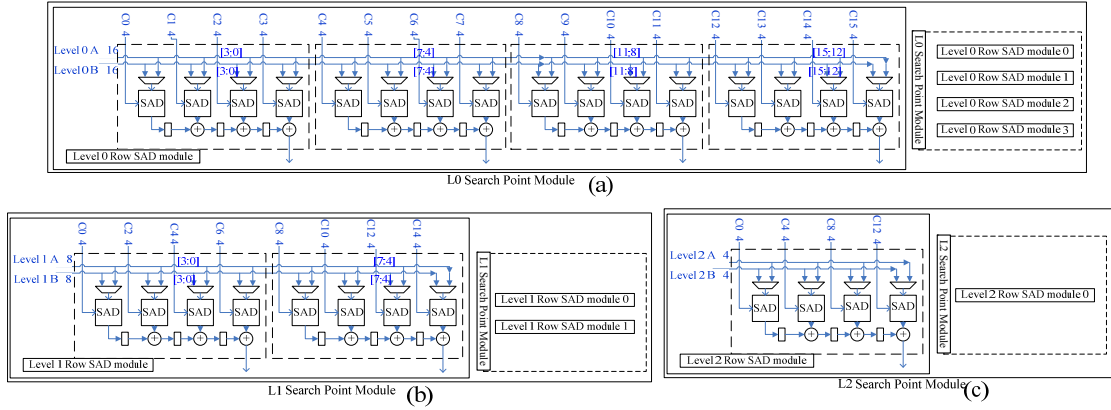


Fig. 2-14. The SAD calculation unit used for different levels. The modules can process a search point of a 16x16 MB within one cycle. (a) The L0 (Level 0) search point module (b) The L1 (Level 1) search point module (c) The L2 (Level 2) search point module.

TABLE 2-4. Memory and bandwidth requirement equation for each level. The MB_{size} is 16. Besides, SR_{L0} , SR_{L1} , and SR_{L2} are 16, 64, and 256 in respect

| Memory cost | buffer size | BW(per MB) |
|---------------|---|---|
| Level 0 | $(SR_{L0} + MB_{size} + 5)$ $* (SR_{L0} + MB_{size} + 5) * 8$ | $(SR_{L0} + MB_{size} + 5)$ $* (SR_{L0} + MB_{size} + 5) * 8$ |
| Level 1 | $(SR_{L1}/2 + MB_{size}/2 - 1)$ $* (SR_{L1}/2 + MB_{size}/2)$ $* (Pixel_Depth_{L1})$ | $(SR_{L1}/2 + MB_{size}/2 - 1)$ $* (SR_{L1}/2 + MB_{size}/2) * 8$ $(16/(64+16)) * 8$ |
| Level 2 | $(SR_{L2}/4 + MB_{size}/4 - 1)$ $* (SR_{L2}/4 + MB_{size}/4)$ $* (Pixel_Depth_{L2})$ | $(SR_{L2}/4 + MB_{size}/4 - 1)$ $* (SR_{L2}/4 + MB_{size}/4) * 8$ $* (16/(256+16)) * 8$ |
| Direct design | $(SR + MB_{size} - 1)$ $(SR + MB_{size}) * 8$ | $(SR + MB_{size} - 1) (SR + MB_{size}) * 8$ $* (16/(256+16)) * 8$ |

TABLE 2-5. Memory and bandwidth requirement is for different frame size. The saving is compared to the direct design [25]. The maximum search range is [-128, 127]

| Memory cost | for 720p | | for 1080p | |
|-----------------|-------------|------------|-------------|------------|
| | buffer size | BW(per MB) | buffer size | BW(per MB) |
| Level 0 (Kbyte) | 1.369 | 1.369 | 1.369 | 1.369 |
| Level 1 (Kbyte) | 0.975 | 0.312 | 1.170 | 0.312 |
| Level 2 (Kbyte) | 2.8475 | 0.268 | 3.417 | 0.268 |
| Total (Kbytes) | 5.1915 | 1.949 | 5.956 | 1.572 |
| Direct design | 73.712 | 4.336 | 73.712 | 4.336 |
| Saving (%) | 92.95 | 55 | 91.91 | 55 |

The SADs generated from the SAD modules are further summed up by the summation trees to generate the SAD of different block size as shown in Fig. 2-15. In Fig. 2-15(a), level 0 has the most complex summation trees for combination of the seven kinds of block types. The SADs of 4x4 blocks 00, 01, 02, and 03 are accumulated in the first step, and then they are saved to registers dly4. When the SADs of the 4x4 blocks 10, 11, 12, and 13 are ready, these SADs are accumulated for 4x8 and 8x4 SADs. Then two 8x4 SADs are used to generate 8x8 SAD. In the same manner, the SAD of 16x8, 8x16, and 16x16 blocks are generated. For the level 1, four “8x8 SAD tree” are used for combination of the mode 1 to mode 4 block types.

Fig. 2-15 (b) presents the 8x8 SAD trees for level 1. However, in level 2, only comparators and registers are needed to select the minimum SAD cost. Finally, the selection module will choose the best two SAD costs from different levels for the fractional ME module.

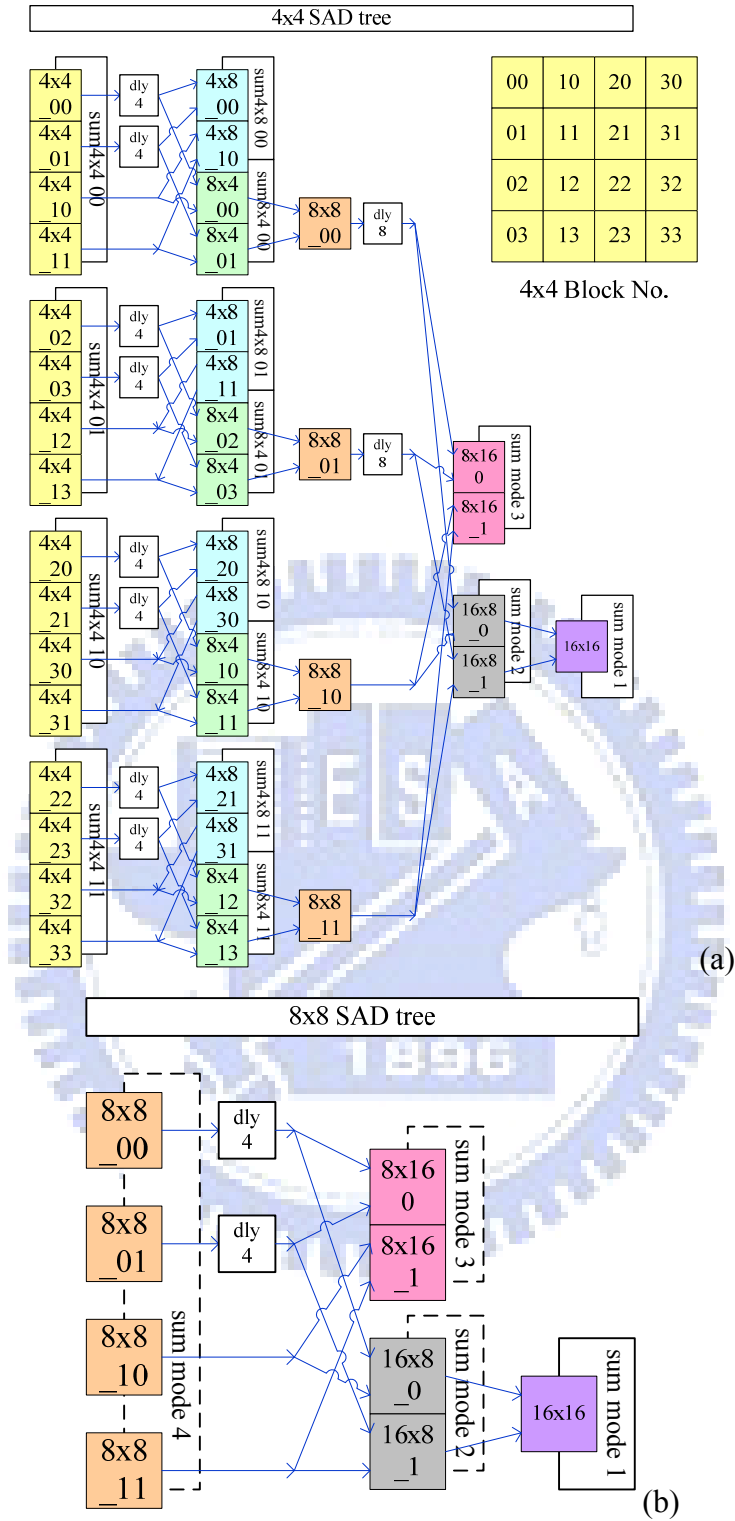


Fig. 2-15. (a) The 4x4 SAD Tree used in level 0. (b) The 8x8 SAD Tree used in level 1.

2.4.4 Implementation result and comparisons

The proposed design has been implemented by Verilog and synthesized by 0.13 μ m CMOS process. TABLE 2-6 shows the total hardware cost of our IME design and comparison to other designs. Our design can provide the largest search range capability (High Profile Level 2) but just needs 213.7K gate count and 5.95KB local SRAM. Besides, our design has the shortest latency so that our design can achieve 1080p@60fps specification with only 124MHz operating frequency only. In comparison, designs in [12][13][26] has larger area cost and long latency due to the full search architecture. Though designs in [14][15] use fast algorithms to reduce the latency, they still needs large area cost and buffer. As for [16], their throughput is only one-fourth of ours though it uses fast algorithm. The proposed IME design can achieve low latency with low buffer cost and similar area cost, and thus is suitable for HD applications.

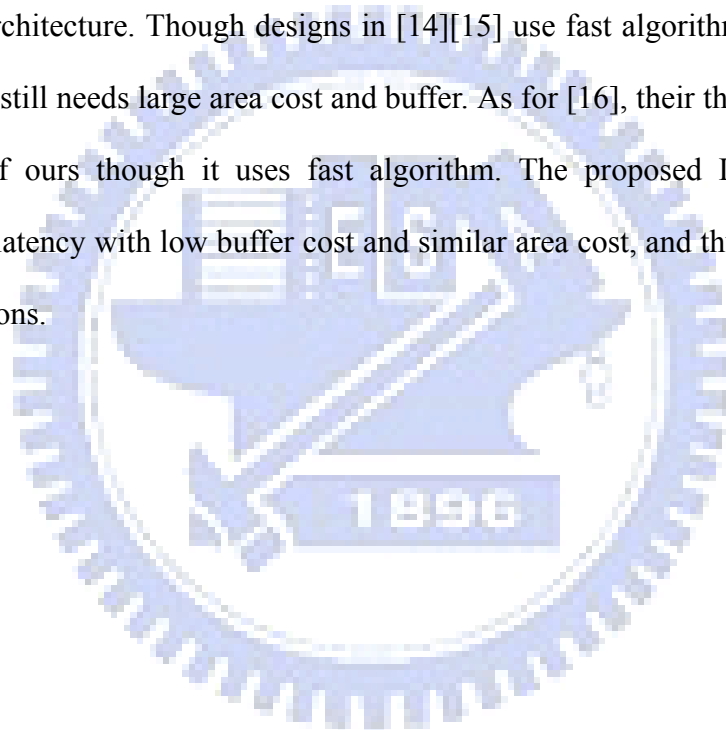


TABLE 2-6 Comparison of the IME part with previous designs.

| | [15] | [9] | [13] | [14] | [26] | [16] | [12] | Ours [35] |
|----------------------|----------------------|-----------------|-----------------|-------------------|-----------------|-----------------------|------------------|-----------------------------------|
| Max. Resolution | CIF@ 30fps | 4CIF@ 15fps | 4CIF@ 15fps | 720x480@ 30fps | 720p@ 30fps | 720p@ 30fps | 720p@ 60fps | 1080p@ 60fps |
| Search Algorithm | 4-Step | Full | Full | Multi-resolution | Full | Sub-sampling | Full | Multi-resolution |
| Quality loss(dB) | About 0.1 | 0 | 0 | 0.4 | 0 | 0.083 | 0 | 0.065 |
| PE (SAD Module) | 256 | 16 | 256 | 64/320 | 1024 | 32 | 256 | 192 |
| Max. Search Range | H:±32 V: ±16 | H:±32 V: ±32 | H:±64 V: ±64 | H:±64 V: ±64 | H:±64 V: ±32 | H:±32 V: ±32 | H:±16 V: ±16 | H: ±128 V: ±128 |
| Gate Count (K) | 131.2 | 61 | 154 | n.a | 330.2 | 47.9+4k bit buffer | 176 | 155.8 for 720p 213.7 for 1080p |
| Memory (Kbyte) | 8 | n.a. | 7.5 | n.a | 26 | 2.75 | 41.6 | 5.19 for 720p 5.95 for 1080p |
| Operating Freq.(MHz) | 40 (13.3 for CIF) | 294 for 4CIF | 100 for 4CIF | 16 for 720x480 | n.a | 105 for 720p | 55.6 for 720p | 27.6 for 720p 124.4for 1080p |
| Latency (Cycle) | n.a. | 4096 | 1024 | 375 | n.a | 972 | 258 | 256 |
| CMOS Tech. | 0.18 um | 0.13um | 0.18um | n.a | 0.18um | 0.18um | 0.18um | 0.13 um |

2.5 Fractional Motion Estimation Module: Single Iteration

Fractional Motion Estimation (SIFME) [40]

2.5.1 Algorithm of SIFME

Fig. 2-16 shows the fractional-pel motion estimation (FME) algorithm in reference software [27]. This search process is divided into two steps. The first step is half-pel motion estimation, where the specific pixels at half-pel spacing are calculated for comparison. The second step is the quarter-pel motion estimation, where the pixels at quarter-pel spacing are obtained for comparison. However, this algorithm searches 17 point totally, and the two step schedule doubles the latency of FME modules. Therefore, we must propose single iteration fractional motion estimation (SIFME) [40] to reduce the latency and hardware cost of FME.

Inspired by the unequal distribution of MVs, we propose SIFME that searches six candidates in only one step without refined search as shown in Fig. 2-17. The candidate with the lowest cost will be selected as the best one.

It first calculates the fractional predicted motion vector ($frac_pred_mv$):

$$frac_pred_mv = (pred_mv - mv) \% \beta \quad (4)$$

where $pred_mv$ here is defined as the fractional pixel unit of MVP. mv is the integer pixel motion vector after IME process, and is also in fractional pel unit. $\%$ is the mode operation. β is 4 in 1/4-pel case and is 8 in 1/8-pel case. $frac_pred_mv$ is the predicted fractional motion vector and indicates only fractional position.

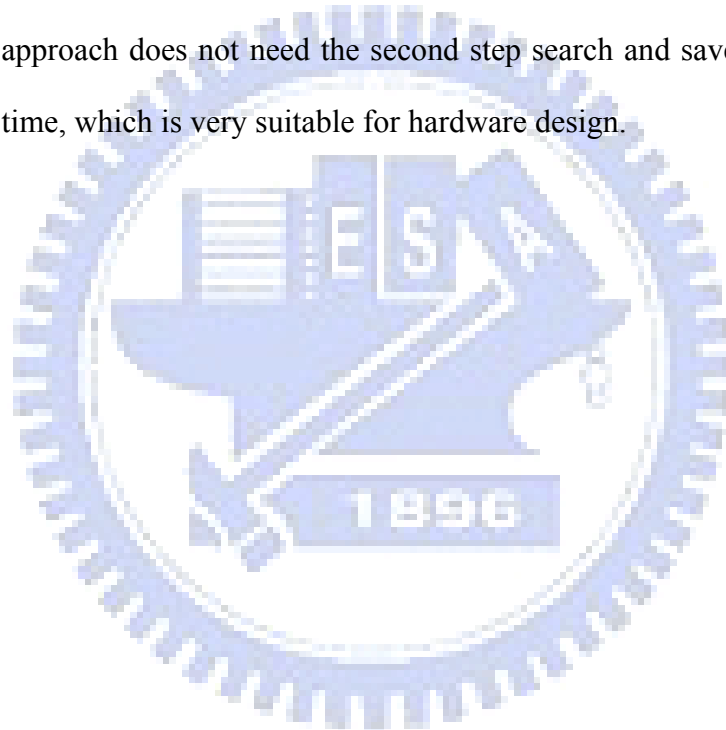
The six candidates includes $(0, 0)$, $frac_pred_mv$ from (4) and four diamond points around $frac_pred_mv$. $(0, 0)$ is included for low texture and low motion sequences. Other search points are placed around $frac_pred_mv$ since the best fractional motion vector is more probable around $frac_pred_mv$ than around $(0, 0)$.

TABLE 2-7 shows the prediction correctness compared with the algorithm in the

reference software. The prediction accuracy is defined as if the fractional MV by the proposed approach is the same as that by the full search algorithm of the reference software. We use four 720p-sized test sequences with 300 frames under different QPs.

The reference software is JM9.0 [27]. This result shows that it has more than 70% prediction accuracy in average though the proposed one has ignored more than 88% search points.

TABLE 2-8 shows the search point comparisons with other algorithms. The proposed algorithm searches the fewest points compared to other search algorithms. Besides, our approach does not need the second step search and saves the additional interpolation time, which is very suitable for hardware design.



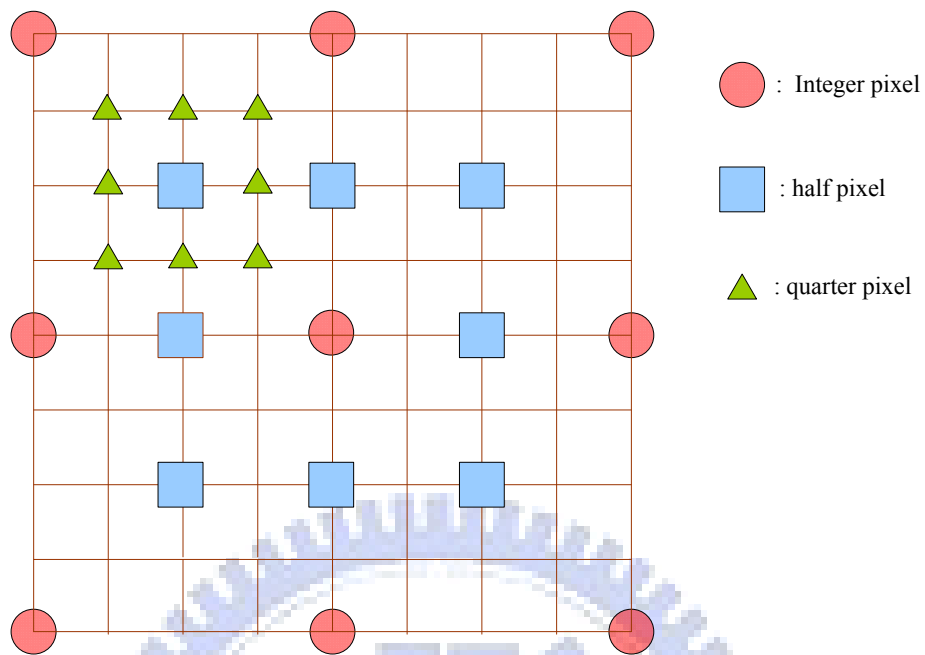


Fig. 2-16 The search algorithm of reference software [27]

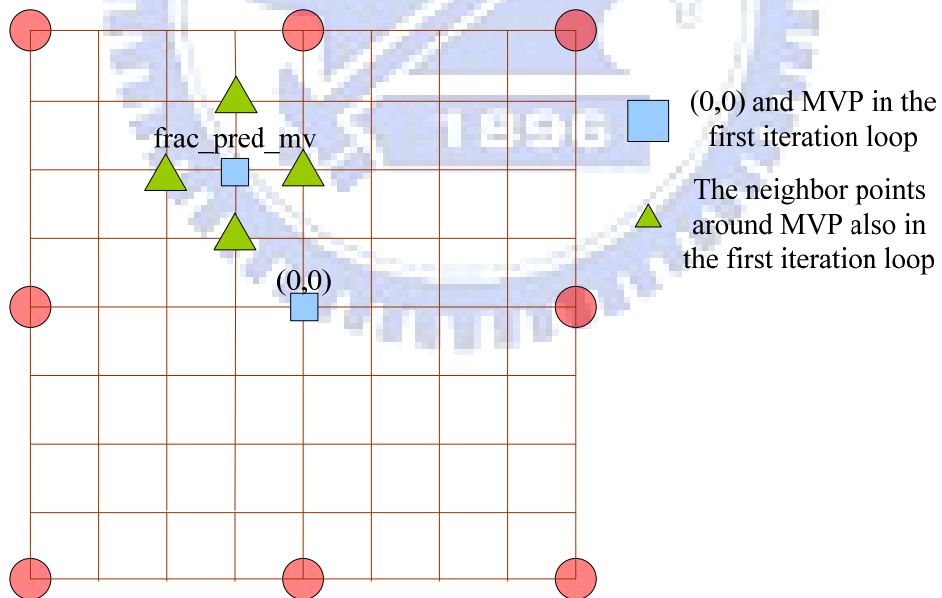


Fig. 2-17. The proposed SIFME on two square points, $(0, 0)$ and $frac_pred_mv$, and four triangle point around $frac_pred_mv$ in one quarter-pel distance.

TABLE 2-7 Prediction accuracy of motion vector (mvx and mvy) compared to the full search FME algorithm

| 720p size, 300 frame, IPPP, RDO off, SR= ± 32 | | | | |
|---|--------------------|---------|----------|-----------|
| QP | mobile calendar | shields | park run | Stockholm |
| 10 | 58.62% | 48.77% | 64.65% | 73.49% |
| 16 | 65.68% | 55.27% | 66.65% | 76.07% |
| 22 | 77.74% | 66.78% | 67% | 78.01% |
| 28 | 87.46% | 87.61% | 72.84% | 84.86% |
| 34 | 91.31% | 92.34% | 80.83% | 91.2% |
| 40 | 92.65% | 93.71% | 85.92% | 94.3% |
| Avg. | 78.91% | 74.08% | 72.94% | 82.9% |

TABLE 2-8 Search point comparisons for different algorithms

| | search point |
|---------------|--|
| JM [27] | 17 |
| [30] | 6+multiple diamond search (Total ≤ 11) |
| [29] | 6 + multiple diamond search |
| [33] | 8~9 |
| Proposed [40] | 6 |

2.5.2 Performance of SIFME

TABLE 2-9 shows the simulation results of SIFME for CIF sequences. Only the first frame is set to I-frame because inserting I-frame periodically will reduce the effect of SIFME. The results shown in TABLE 2-9 are compared with the reference software. The maximum PSNR drop is only 0.13dB and the bit-rate overhead is lower than 2.11% except the cases with QP40. That is quite acceptable since the bit rate at that condition is quite low and any increase will be large in terms of that bit rate.

TABLE 2-10 and TABLE 2-11 show the simulation results of the proposed SIFME for 720p and 1080p sequences. Since our hardware architecture is used for HDTV size video, we care more about the performance on 1080p and 720p size sequences

rather than that on CIF size sequences. For the result on 720p size sequence shown in TABLE 2-10, the PSNR degradation is lower than 0.08dB and the bit-rate increasing is below 4.28%. Moreover, the bit-rate even decreases in most cases. The reason may be that SIFME tends to find the motion vector similar to the motion vector predictor (mvp) and thus saves bits for coding motion vectors. TABLE 2-11 shows that the result of 1080p video. The PSNR degradation is also lower than 0.08dB and the bit-rate increasing is below 4.47%. Under high QP condition, the PSNR performance of SIFME is even better than that of the reference software. The reason is that the correct motion vectors are getting closer to motion vector predictors under high QP condition, and hence the accurate fractional motion vectors are getting closer to frac_mv_pred in eq. (4).

Comparing the results shown in TABLE 2-9 to the results in TABLE 2-10 and TABLE 2-11, we can find that SIFME has better performance on large size sequences than CIF size sequences, which matches our goal. Besides, SIFME greatly reduces computation time of FME. The proposed algorithm can speed up the FME part by up to 4 times compared to the reference software. The major reason is the reduction of search candidates.

Summing the information from TABLE 2-9 to TABLE 2-11, we can conclude that SIFME can reduce 88% of search points and speed up the coding by 4 times with only less than 0.13 dB PSNR degradation and 4.47% of bit rate increase. For some 720p or 1080p sequences, SIFME even has better PSNR quality or less bit rate than that of JM software [27].

TABLE 2-12 shows the performance comparison with previous works. SIFME speeds up more than our previous work [33] with the similar PSNR quality and less bit rate increase. The algorithm in [29] has better video quality than ours but it requires much iteration and hence is not suitable for hardware implementation.

TABLE 2-9 Simulation results of SIFME for different CIF sequences and QPs when compared to the reference software [27]

| CIF size, 300 frame, only first frame is I-frame, ProfileIDC=100, RDO off, Search range = 32 | | | | | | | | |
|---|-----------------------|----------------------|-----------------------|----------------------|----------------------------|----------------------|-----------------------|----------------------|
| SIFME | | | | | | | | |
| | container | | foreman | | mobile&calendar | | stefan | |
| QP | Δ PSNR (dB) | Δ bit rate | Δ PSNR (dB) | Δ bit rate | Δ PSNR (dB) | Δ bit rate | Δ PSNR (dB) | Δ bit rate |
| 10 | -0.03 | -0.75% | -0.05 | 0.04% | -0.04 | -0.24% | -0.04 | 0% |
| 16 | 0 | -0.28% | -0.07 | 1.03% | -0.06 | 0.16% | -0.05 | 0.30% |
| 22 | -0.03 | -0.37% | -0.09 | 0.89% | -0.08 | 0.06% | -0.06 | 0.50% |
| 28 | 0.03 | 0.46% | -0.09 | 1.50% | -0.07 | 0.47% | -0.07 | 1.24% |
| 34 | 0.04 | 2.11% | -0.12 | 1.35% | -0.07 | 1.73% | -0.10 | 1.57% |
| 40 | -0.03 | 4.36% | -0.08 | -0.36% | -0.08 | 2.30% | -0.13 | 1.02% |

TABLE 2-10 PSNR and bit rate comparison for different 720p sequences and QPs.
Speed up is only the performance in fractional ME part

| 720p, 300 frames, only first frame is I-frame, ProfileIDC=100, RDO off, search range=64 | | | | | | | | | | | | |
|--|-----------------------|----------------------|-------------|-----------------------|----------------------|-------------|-----------------------|----------------------|-------------|-----------------------|----------------------|-------------|
| SIFME | | | | | | | | | | | | |
| | mobcal | | | parkrun | | | shields | | | stockholm | | |
| QP | Δ PSN R(dB) | Δ bit rate | speed up | Δ PSN R(dB) | Δ bit rate | speed up | Δ PSN R(dB) | Δ bit rate | speed up | Δ PSN R(dB) | Δ bit rate | speed up |
| 10 | -0.04 | -0.77% | 4.0 | -0.02 | -0.77% | 3.9 | -0.04 | -0.42% | 3.6 | -0.04 | 0.05% | 3.8 |
| 16 | -0.04 | -1.07% | 3.6 | -0.04 | -0.99% | 3.7 | -0.08 | -1.27% | 3.7 | -0.08 | -0.86% | 3.6 |
| 22 | -0.01 | -1.08% | 4.0 | -0.05 | -1.42% | 3.9 | -0.04 | -1.54% | 3.9 | -0.05 | -1.50% | 3.7 |
| 28 | -0.01 | -0.36% | 3.9 | -0.04 | -0.63% | 3.9 | -0.02 | -0.36% | 3.6 | -0.02 | -0.71% | 3.8 |
| 34 | -0.05 | 3.20% | 3.9 | -0.05 | -0.14% | 3.8 | -0.03 | 0.30% | 3.6 | -0.01 | -1.87% | 3.7 |
| 40 | -0.06 | 4.28% | 3.7 | -0.04 | -0.70% | 4.1 | -0.01 | -7.05% | 3.5 | 0 | -8.86% | 3.7 |

In summary, SIFME greatly reduces computational complexity and is suitable for hardware design with only small amount of quality loss.

TABLE 2-11 PSNR & bit rate comparison for different 1080p sequences and QP

| 1080p, 200 frames, only first frame is I-frame, ProfileIDC=100, RDO off, SearchRange=128 | | | | | | | | | | | | | | |
|---|-----------------------|----------------------|-----------------------|----------------------|-----------------------|----------------------|-----------------------|----------------------|-----------------------|----------------------|-----------------------|----------------------|-----------------------|----------------------|
| SIFME | | | | | | | | | | | | | | |
| | blue sky | | pedestrian | | riverbed | | rush hour | | sation2 | | sunflower | | tractor | |
| QP | Δ PSN R(dB) | Δ bit rate | Δ PSN R(dB) | Δ bit rate | Δ PSN R(dB) | Δ bit rate | Δ PSN R(dB) | Δ bit rate | Δ PSN R(dB) | Δ bit rate | Δ PSN R(dB) | Δ bit rate | Δ PSN R(dB) | Δ bit rate |
| 10 | -0.06 | -1.14% | -0.07 | -1.41% | -0.07 | -0.53% | -0.07 | -1.13% | -0.06 | -0.64% | -0.11 | -0.27% | -0.08 | 0.38% |
| 16 | -0.05 | -0.74% | -0.05 | -0.68% | -0.09 | -1.85% | -0.06 | 0.62% | -0.07 | -1.08% | -0.10 | -0.11% | -0.12 | -0.16% |
| 22 | -0.03 | -1.20% | -0.05 | -1.32% | -0.07 | -2.11% | -0.04 | 0.02% | -0.08 | -1.65% | -0.07 | -2.53% | -0.11 | -1.38% |
| 28 | 0 | 0.08% | -0.02 | -1.03% | -0.08 | -1.14% | 0.02 | 0.72% | -0.01 | 4.70% | -0.01 | -1.71% | -0.09 | -0.66% |
| 34 | 0.01 | 2.40% | 0.07 | 0.55% | -0.03 | 0.48% | 0.13 | 1.68% | 0.03 | -2.43% | -0.02 | -3.54% | -0.03 | 1.10% |
| 40 | 0.08 | 4.47% | 0.16 | 0.68% | 0.06 | 1.45% | 0.22 | 1.44% | 0.13 | -7.55% | 0.02 | -5.07% | 0 | 2.36% |

TABLE 2-12 Simulation comparison with previous works.

| QP = 28 | | Stefan | Mobile | Foreman | Coastguard | News | # of iteration |
|------------------------|----------------------|---------------|---------------|----------------|-------------------|-------------|-----------------------|
| JM [27] | bit rate | 1441.14 | 1888.69 | 498.62 | 1127.87 | 223.72 | 1 |
| | PSNR | 35.36 | 33.75 | 36.24 | 34.52 | 38.12 | |
| | time (sec) | 491.604 | 471.993 | 496.974 | 488.039 | 450.37 | |
| Y. J. Wang [33] | Δ bit rate(%) | 2.2843 | 2.36407 | 1.780915 | 1.070159 | 2.23494 | 2 |
| | Δ PSNR(dB) | -0.09 | -0.11 | -0.07 | -0.04 | -0.06 | |
| | speed up | 2.34227 | 2.24167 | 2.361651 | 2.283373 | 2.24787 | |
| CBFPS [29] | Δ bit rate(%) | -0.1524 | -0.0822 | -0.7819 | -0.402 | 0.2294 | > 2 |
| | Δ PSNR(dB) | -0.01 | -0.01 | -0.03 | -0.01 | 0 | |
| | speed up | 2.163 | 2.265 | 2.249 | 2.307 | 2.638 | |
| proposed [40] | Δ bit rate(%) | 1.2408 | 0.4657 | 1.5022 | -0.9468 | 2.3643 | 1 |
| | Δ PSNR(dB) | -0.07 | -0.07 | -0.09 | -0.06 | -0.09 | |
| | speed up | 3.6 | 3.9 | 3.7 | 3.8 | 3.9 | |

2.5.3 Architecture of SIFME

Fig. 2-18 shows the proposed FME hardware architecture. The input data is first interpolated by the interpolation unit for half and quarter pixels of one 4x4 block. Then these data are computed with the current block data by six 4x4 block PUs. Each PU is in charge of residual generation and 4x4 Hadamard transform. All larger sized block are decomposed into 4x4 block for processing. Then the residual cost combining with MV cost is sent to the Compare Unit to find the best one and stored in SB_buffer.

TABLE 2-13 shows the comparisons of the number of PUs and iteration steps. SIFME searches only six candidates and thus only needs six PUs. Besides, with single loop design, our design just takes about only half of cycles when compared to others [28][33].

The interpolation unit is shown in Fig. 2-19. It is composed of two sets of directional (horizontal and vertical) 1-D FIR filters as shown in Fig. 2-20. First, we interpolate the horizontal half pixels by five FIR filters from 10 adjacent integer pixels. These five intermediate values and six integer pixels are stored and shifted cycle by cycle in the interpolation buffer. We use the same way to interpolate the vertical half pixels with 11 FIR filters. In our algorithm, because we don't need to check all possible half pixels, 88% of redundant filters for full search can be removed.

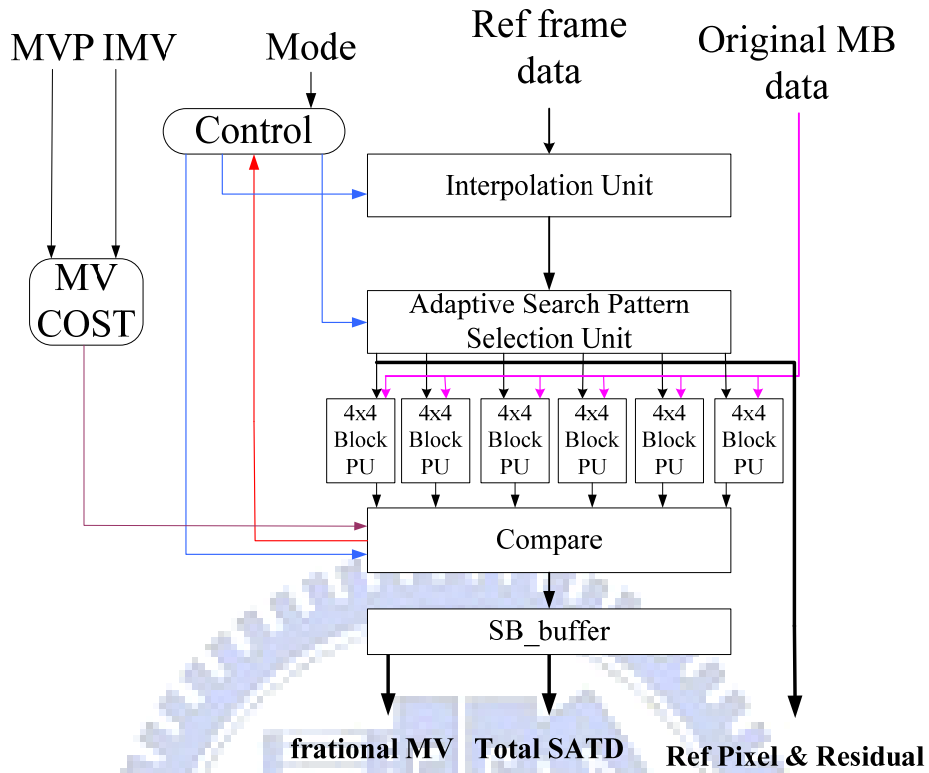


Fig. 2-18. The proposed hardware architecture of FME.

TABLE 2-13 comparisons of number of processing unit (PU) and number of iterative search steps

| | # of PU | # of iterative search step |
|---------------|---------|----------------------------|
| [33] | 5 | 2 |
| [28] | 9 | 2 |
| Proposed [40] | 6 | 1 |

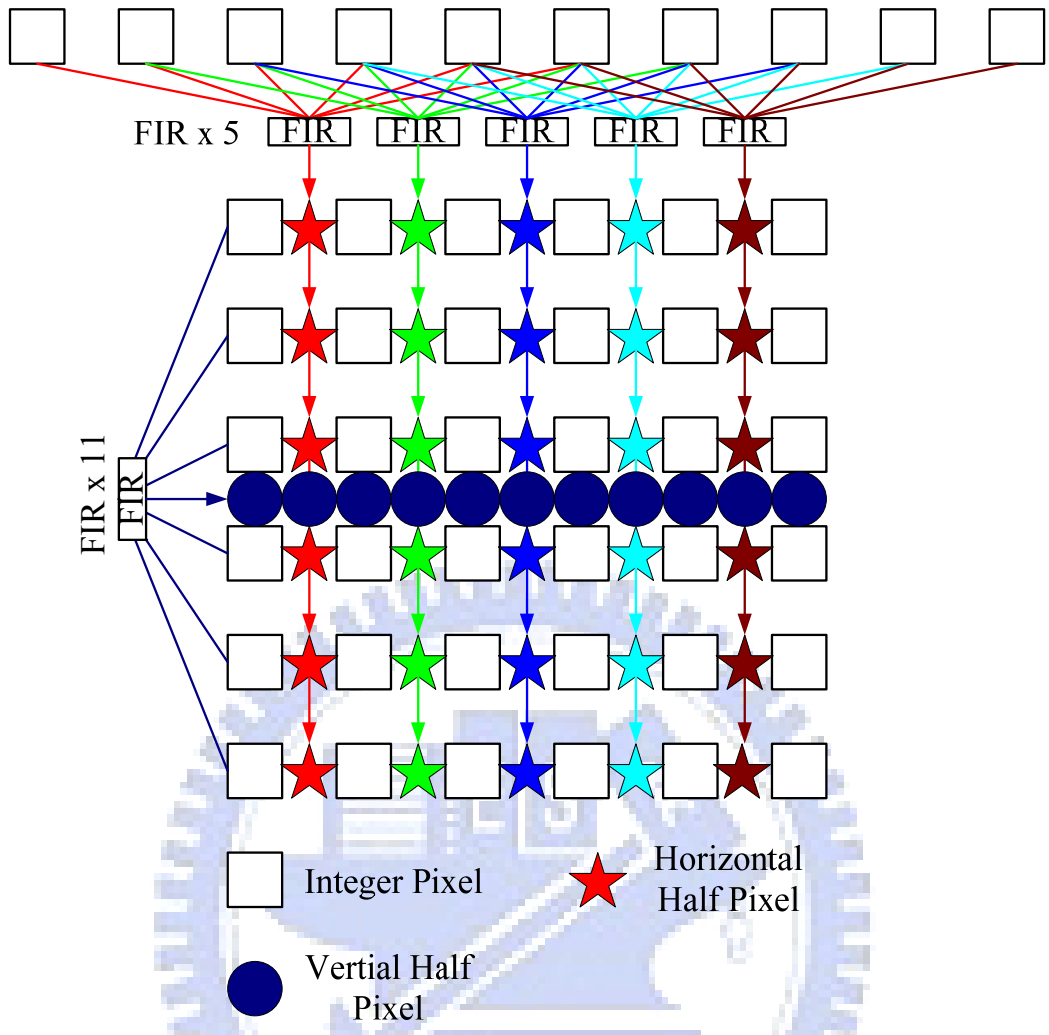


Fig. 2-19 Interpolation unit

$$\text{Out} = A - 5B + 20C + 20D - 5E + F$$

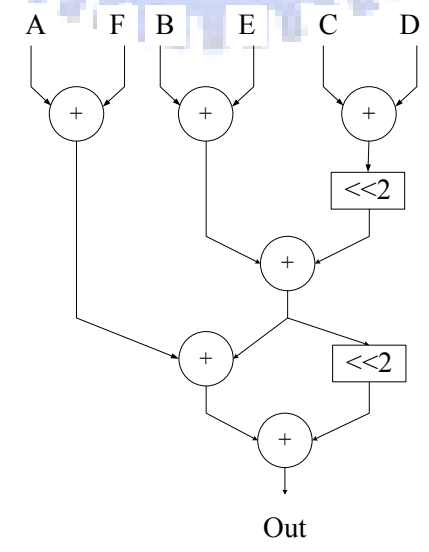


Fig. 2-20. 6-tap 1-D FIR filter

2.5.4 Implementation result and comparisons of SIFME

Our design is implemented by Verilog and synthesized by 0.13 μ m technology. The area gate count is 52.8K for 720p video and 68.9K for 1080p. The latency without mode filtering is 1002 cycles, but it can be reduced to 264 cycles if combining with mode filtering as discussed in section 2.3. Therefore, our design can achieve high throughput with low area cost and negligible quality loss. TABLE 2-14 shows the comparison of our implementation result to previous works. From this table, we can observe that our hardware cost is only slightly larger than our previous work [33], but with six times throughput. As for [34], the area cost is doubled than our work because it needs to search 25 points. In comparison, though design in [36] has higher throughput than ours, their quality drop is more than 0.15dB due to imprecise mathematical modeling. Besides, such mathematical modeling design still needs additional hardware to calculate the final residuals, which is not included in the gate count report. When comparing to other design for 1080p@30fps [38], our FME design only needs 36.5% of gate count and 54.6% of cycles due to our hardware based approaches.

TABLE 2-14 Comparison of the FME part with previous designs

| | [28] | [33] | [34] | [38] | [36] | Ours [40] |
|----------------------------|--|---|--|--|--------------------------------|---|
| Max. Supporting Resolution | 720x576@30fps | 720p@30fps | 720p@30fps | 1080p@30fps | 3200x2400@30fps | 1080p@60fps |
| Algorithm | 17 candidates 2-iteration interpolation | 8 candidates 2-iteration interpolation | 25 candidates 1-iteration interpolation | 17 candidates 2-iteration interpolation | Math-model No interpolation | 6 candidates 1-iteration interpolation |
| Gate Count (K) | 79.3 | 48 | 117.2 | 188.45 | 56.53 | 52.8 for 720p 68.9 for 1080p |
| Latency (Cycle) | 1648 | 2000 | 1000 | 790 | 110 | 1002 264~432 (With mode filtering) |
| Operating Freq.(MHz) | 100 | 100 | 108 for 720p | 285 | 100 | 28.5 for 720p 128.3 for 1080p |
| Throughput (Kilo-MBs/sec) | 49 | 50 | 108 | 250 | 909 | 486 |
| Quality Drop (dB) | 0.1 | 0.09 | 0.012 | n.a. | 0.15 | 0.04 |
| CMOS Tech. | 0.18 μ m | 0.18 μ m | 0.13 μ m | 0.18 μ m | 0.13 μ m | 0.13 μ m |

2.6 Integrated Design

2.6.1 Integrated video quality analysis

TABLE 2-15 presents the simulation results for different algorithms combinations: PMRME, mode filtering, and SIFME. In these results, we also include the bit truncation in this design to reduce the hardware cost. The simulation environments are as following: rate-distortion optimization (RDO) is off. Only the first frame is I-frame and search range is $[-128, 127]$. All of the simulation results are compared to that of the default full search algorithm in JM9.0 [27]. The result in this table only shows the average performance under different QPs. The test sequences are all 720p resolution including Stockholm, parkrun, mobile calendar, and shields. The frame rate is 30 and 300 frames are coded.

TABLE 2-15 shows the PSNR change, bit-rate increasing, and “Sharing Rate of L0 (Level 0) buffer”. The sharing rate of level 0 denotes the percentage that FME can directly reuse level 0 search range buffer for computation to save memory bandwidth. This sharing occurs once the final MV is within level 0 search range. In our design, the sharing rate is at least 90%, and the higher QP will have higher sharing rate and thus can save more power and bandwidth.

In this table, we can find that the performance of PMRME is almost the same with full search. The average PSNR drop is only 0.005dB and the bit-rate is even decreasing when comparing with full search. It is because the PMRME ignores smaller blocks in level 1 and level 2 and prefer larger block which results bit-rate decrease. As for mode filtering, the algorithm also prefers to select larger block size, so the bit-rate decreasing is more obvious. Oppositely, the PSNR drop is a little serious than using PMRME only. But the worst quality drop is only 0.095dB. While considering the bit-truncation technique, the influence on PSNR is only 0.064dB, and the bit-rate increasing is 0.52% in average. Finally, we combine all proposed

techniques, the PSNR quality is almost the same and the bit-rate quality drop is a little increasing to 2.11% in average.

TABLE 2-16 shows the performance of our proposed algorithms for 1080p video sequences. The performance is not as good as the performance for 720p video, especially the bit-rate increasing rate. The average bit-rate increasing rate reaches 3.07% for QP32. This is because the 1080p sequences prefer the larger block size than 720p or other smaller sequences, which agrees the tendency of our proposed algorithms. Therefore, our algorithms don't provide too much reduction in bit-rate which happens in the smaller sequences as shown in TABLE 2-1, TABLE 2-2, TABLE 2-9, and TABLE 2-10. However, the quality loss is still acceptable and the average sharing rate is also higher than 90% for 1080p sequences.

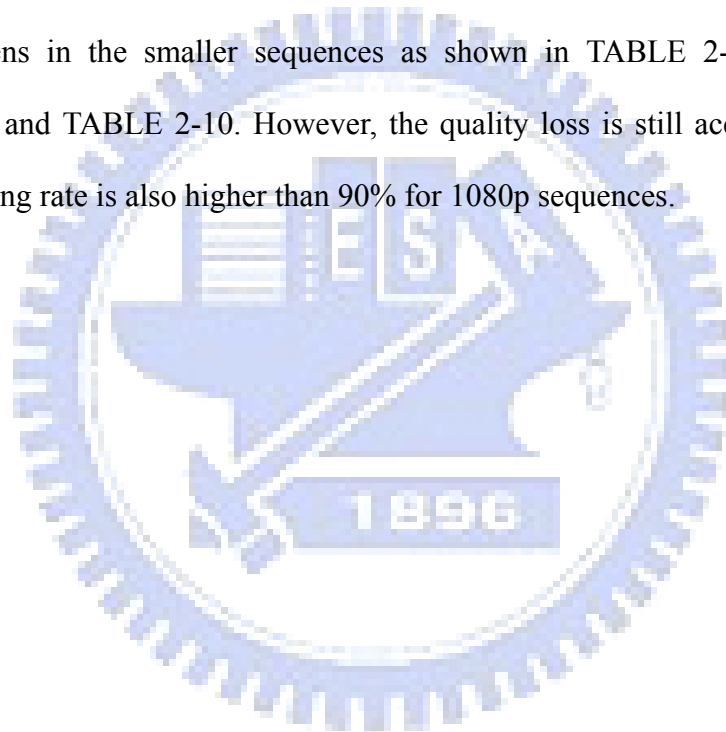


TABLE 2-15 PSNR and bitrate change for proposed algorithms compared with full search for 720p sequences

| Frame size | | 720p | | | |
|------------|------------------------------|---------|-----------------------------|---|--|
| QP | | PMRME | PMRME +MODE FILTERING | PMRME +MODE FILTERING +Bit-Truncation(5 bits) | PMRME +MODE FILTERING +Bit-Truncation (5 bits) +SIFME |
| QP16 | PSNR inc.(db) | -0.0025 | -0.085 | -0.075 | -0.0975 |
| | Bit rate inc. (%) | -1.02 | -1.66 | -0.66 | 1.08 |
| | Sharing Rate of L0 Buffer(%) | n.a. | 96.95 | 96.08 | 96.10 |
| QP20 | PSNR inc.(db) | 0 | -0.095 | -0.0825 | -0.117 |
| | Bit rate inc. (%) | -0.49 | -1.24 | -0.013 | 1.80 |
| | Sharing Rate of L0 Buffer(%) | n.a. | 97.84 | 96.86 | 96.87 |
| QP24 | PSNR inc.(db) | -0.0075 | -0.08 | -0.0675 | -0.1025 |
| | Bit rate inc. (%) | -0.33 | -1.11 | 0.32 | 2.27 |
| | Sharing Rate of L0 Buffer(%) | n.a. | 98.36 | 97.77 | 97.75 |
| QP28 | PSNR inc.(db) | -0.005 | -0.0625 | -0.0525 | -0.0925 |
| | Bit rate inc. (%) | 0.20 | -0.57 | 0.76 | 2.52 |
| | Sharing Rate of L0 Buffer(%) | n.a. | 98.78 | 98.21 | 98.19 |
| QP32 | PSNR inc.(db) | -0.01 | -0.0525 | -0.045 | -0.09 |
| | Bit rate inc. (%) | 1.56 | 1.14 | 2.18 | 2.90 |
| | Sharing Rate of L0 Buffer(%) | n.a. | 99.00 | 98.30 | 98.31 |
| Avg | PSNR inc.(db) | -0.005 | -0.075 | -0.0645 | -0.1 |
| | Bit rate inc. (%) | -0.017 | -0.69 | 0.52 | 2.11 |
| | Sharing Rate of L0 Buffer(%) | n.a. | 98.18 | 97.44 | 97.44 |

TABLE 2-16 PSNR and bitrate change for proposed algorithms compared with full search for 1080p sequences

| QP | Frame size | 1080p | | | |
|------|---------------------------------|-------|-----------------------------|---|--|
| | | PMRME | PMRME +MODE FILTERING | PMRME +MODE FILTERING +Bit-Truncation(6 bits) | PMRME +MODE FILTERING +Bit-Truncation (6 bits) +SIFME |
| QP16 | PSNR inc.(db) | 0 | -0.07 | -0.06 | -0.11 |
| | Bit rate inc. (%) | -0.49 | -1.09 | 0.47 | 2.22 |
| | Sharing Rate of L0 Buffer(%) | n.a. | 93.61 | 90.64 | 92.73 |
| QP20 | PSNR inc.(db) | -0.01 | -0.04 | -0.04 | -0.08 |
| | Bit rate inc. (%) | -0.44 | -0.22 | 2.65 | 5.04 |
| | Sharing Rate of L0 Buffer(%) | n.a. | 95.6 | 94.12 | 94.48 |
| QP24 | PSNR inc.(db) | -0.03 | -0.06 | -0.06 | -0.09 |
| | Bit rate inc. (%) | -0.4 | -0.94 | 1.83 | 3.57 |
| | Sharing Rate of L0 Buffer(%) | n.a. | 95.94 | 95 | 95.08 |
| QP28 | PSNR inc.(db) | -0.06 | -0.07 | -0.08 | -0.1 |
| | Bit rate inc. (%) | 0.4 | 0.19 | 2.20 | 3.7 |
| | Sharing Rate of L0 Buffer(%) | n.a. | 95.97 | 95.17 | 95.19 |
| QP32 | PSNR inc.(db) | 1.68 | -0.09 | -0.1 | -0.08 |
| | Bit rate inc. (%) | 1.56 | 1.59 | 3.06 | 3.44 |
| | Sharing Rate of L0 Buffer(%) | n.a. | 95.65 | 94.79 | 95.05 |
| Avg | PSNR inc.(db) | -0.04 | -0.07 | -0.07 | -0.08 |
| | Bit rate inc. (%) | 0.15 | -0.1 | 2.04 | 3.07 |
| | Sharing Rate of L0 Buffer(%) | n.a. | 95.36 | 93.94 | 94.49 |

2.6.2 Integrated architecture

Fig. 2-21 shows the total block diagram of the full ME modules. It contains IME, FME, several memory buffer and external data access interface. The whole flow is as described in Fig. 2-5(b).

To enable the data reuse between IME and FME, the IME module has three internal SRAMs for reference pixels storage. When the IME search of a MB is completed, its macroblock information is sent to FME. Moreover, the reference pixels in level 0 SRAM is also sent to FME. However, instead of moving data, we use three SRAMs

as the level 0 buffer and swap them with a ping-pong buffer concept. The three level 0 buffers includes one for IME level 0 reference, one for FME, and one for loading new data from external memory. Whenever the IME stage completes the coding of the first MB, the buffer for level 0 reference for the first MB is changed as the FME reference in the next stage. At the same time, the buffer for current FME reference is changed to load the data of the third MB from external memory for further use. The buffer that is now filled the reference data for the second MB is switched for IME level 0 reference. With above ping-pong buffers, we can share the level 0 data of IME with the FME, and no additional memory access time is necessary. Besides, the data in level 0 buffers can be reused by FME for more than 90% of MBs according to the sharing rate in TABLE 2-15. With above arrangement, all these data can be reused as much as possible and reduce the bandwidth a lot.

2.6.3 Implementation results and comparisons

TABLE 2-17 shows the total hardware cost of our ME design and comparison to the integrated designs [18][25]. Comparing to [25], our design can save at least 30% of area costs and 50% of memory costs in IME part. As for FME part, we save 82.8% of area cost due to fewer number of PUs and reduce 81.2% of memory. In summary, the total area and memory saving is 60% and 65.78% respectively. As for the throughput, our design is sufficient for HD video applications. Our design improves throughput by 75% when comparing to that in [25]. If comparing with the other integrated design [11] using fast algorithms in IME, our design still saves 12.3% area. As for the cycle count, our design also has 75.5% of throughput improvement than [18]. By the high throughput, only 28.5 MHz is enough for 720p sequence with 30 frames per second, and 128.8 MHz for 1080p sequences with 60 frames per second. To satisfy high throughput requirement, the external bus width should be 128 bits.

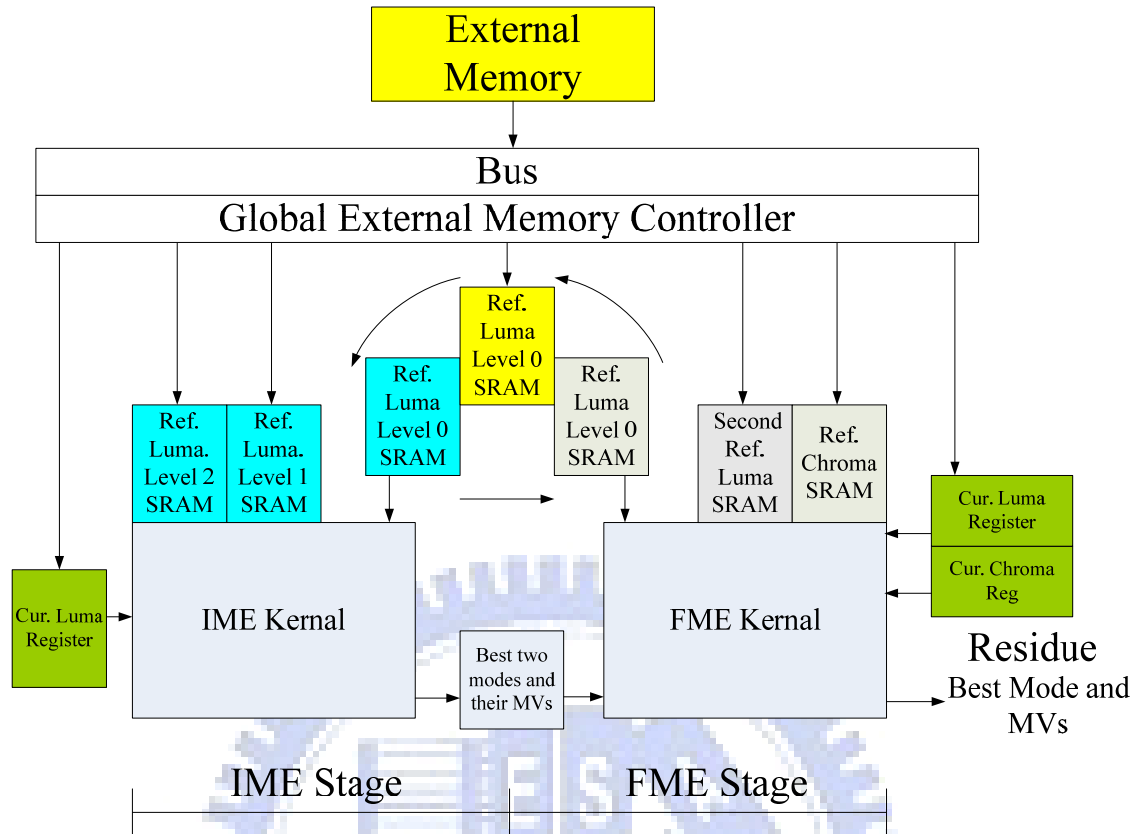


Fig. 2-21. The block diagram of IME and FME.

2.7 Summary

In this chapter, we propose a highly data reused ME design with low cost and latency for high definition video. This design maximizes the most concerned data reuse by sharing data within IME as well as between IME and FME, while minimizes the computation and latency by parallel multi-resolution IME and single iteration FME. The final design can easily support processing for 1080p sequences with just 128.8MHz and 282.6K gates, and saves 60% of gate count, and 68.9% of SRAM buffers compared to the previous design. The presented design also can be easily scaled to other smaller size video with search range adjustment.

TABLE 2-17 hardware cost comparison for complete H.264 ME accelerator with previous works

| | [25] | [18] | Ours [6] | Saving with [25] | Saving with [18] |
|-------------------------------|----------------------------|----------------------------|-----------------------------------|-----------------------------------|------------------|
| Max. Resolution | 720p@30fps | 720p@30fps | 1080p@60fps | | |
| Search Range | H: ± 64 V: ± 32 | H: ± 96 V: ± 96 | H: ± 128 V: ± 128 | | |
| Quality Loss (dB) | 0 | n.a. | 0.1 | | |
| IME Gate Count (K) | 305.2 | n.a | 155.8 for 720p 213.7 for 1080p | 48.9% for 720p 30% for 1080p | |
| FME Gate Count (K) | 401.8 | n.a | 52.8 for 720p 68.9 for 1080p | 86.8% for 720p 82.8% for 1080p | |
| Total Gate Count (K) | 707 | 238 | 208.6 for 720p 282.6 for 1080p | 70.4% for 720p 60% for 1080p | 12.3% for 720p |
| IME Memory (Kbyte) | 13.71 | n.a | 5.19 for 720p 5.95 for 1080p | 62.1% for 720p 56.6% for 720p | n.a |
| FME Memory (Kbyte) | 13.82 | n.a | 2.59 | 81.2% | n.a |
| Total Memory (Kbyte) | 27.53 | n.a | 7.78 for 720p 8.54 for 1080p | 71.7% for 720p 68.9% for 1080p | n.a |
| Latency for IME Stage (Cycle) | 1024 | 1079 | 256 | 75% | 75.5%(Best) |
| Latency for FME Stage (Cycle) | 1648 | | 264(Best) 432(Worst) | 83.9%(Best) | |
| Freq. (MHz) | 120 (108 for 720p) | 117 for 720p | 28.5 for 720p 128.8 for 1080p | 73.6% for 720p | 75.6% for 720p |
| CMOS Tech. | 0.18 μ m | 0.18 μ m | 0.13 μ m | | |



Chapter 3

Design of H.264 1080p Intra-only

Encoder

In H.264 video standard, the intra frame prediction technique can efficiently use the neighboring pixels to predict the current coding block from various directions. With this, the coding efficiency is even competitive with the latest still image coding standard, JPEG2000. Thus, this all intra frame coding is now accepted as the intra only profile [41], and is suitable for applications like digital video recorder and digital still camera that cannot afford the inter prediction computing.

Therefore, we propose a H.264 intra encoder which keeping the high coding efficiency and excellent video quality without the large hardware cost and heavy computing loading from the motion estimation parts in H.264 [7].

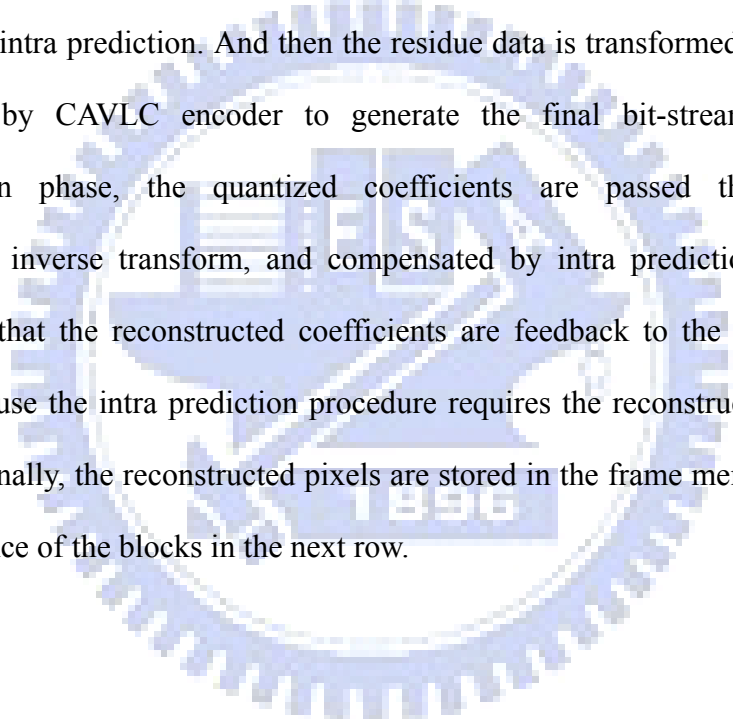
In this chapter, we first introduce the basic of intra prediction and other modules in H.264 intra encoder. In the second section, the design challenges of H.264 intra encoder for high definition video and image applications are described. Besides, the previous works are also reviewed and discussed. In the third section, our proposed algorithms including three step algorithms, enhanced SATD algorithm, and plain mode removal are used to speed up the encoding procedure and reduce hardware cost of intra prediction part. In architecture level, the variable-pixel parallelism architecture is proposed to satisfy the throughput requirement while minimizing the hardware overhead. Some module level optimization techniques are also proposed in integer transform, quantization, and CAVLC designs to increase the throughput and

decrease the hardware cost. Finally, the implementation result and comparisons of the entire H.264 intra encoder are presented.

3.1 Introduction of H.264 intra-only encoder

3.1.1 Overview of H.264 Intra-only encoder

The major difference between general H.264 encoder and intra only encoder is that the inter prediction and deblocking modules are removed from intra-only encoder. Fig. 3-1 shows the block diagram of H.264 intra-only encoder. The input video is predicted by intra prediction. And then the residue data is transformed, quantized and compressed by CAVLC encoder to generate the final bit-stream. As for the reconstruction phase, the quantized coefficients are passed through inverse quantization, inverse transform, and compensated by intra prediction module. We should note that the reconstructed coefficients are feedback to the intra prediction module because the intra prediction procedure requires the reconstructed data of the left block. Finally, the reconstructed pixels are stored in the frame memory and as the upper reference of the blocks in the next row.



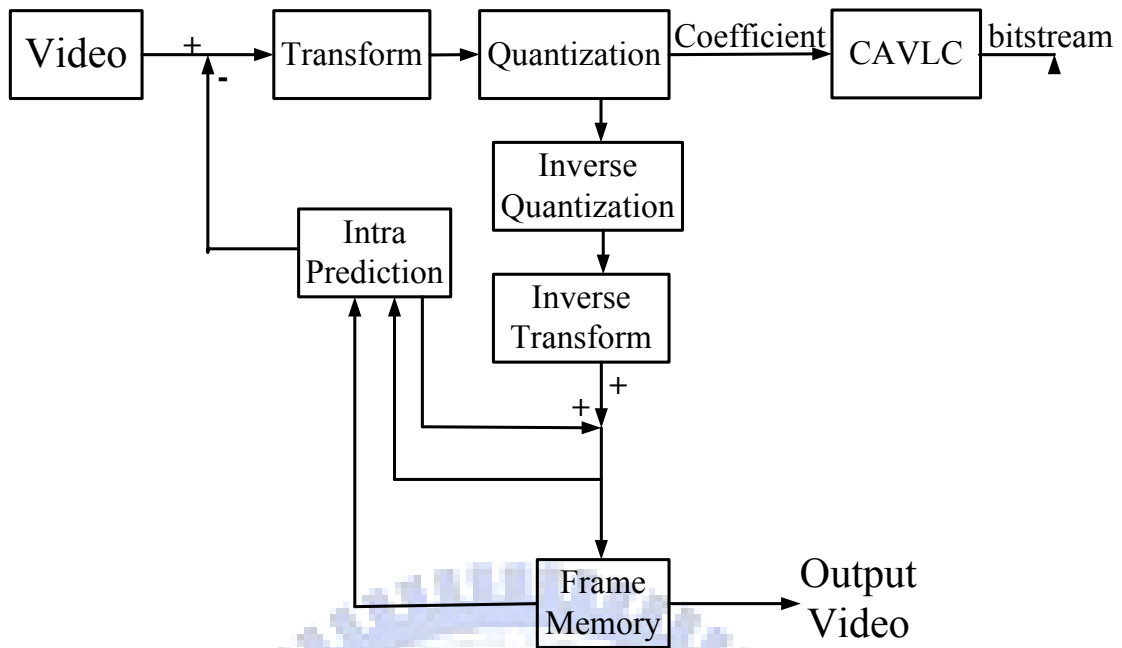


Fig. 3-1 Block diagram of intra-only encoder.

3.1.2 Intra prediction

The intra prediction technique is first proposed in the MPEG-4 standard, which uses neighbor pixels to predict the current blocks in vertical and horizontal directions. In H.264/AVC, two different block sizes of intra prediction are possible for the prediction of the luminance component. These predictions can be one of nine kinds of 4x4 luma prediction modes as in Fig. 3-2 or four kinds of 16x16 luma prediction mode as in Fig. 3-3. In Fig. 3-2 and Fig. 3-3, the number before the name of each mode is the mode number of each mode, and we use the number to replace the name of each mode in this paper for simplicity. For chroma blocks, four kinds of 8x8 chroma prediction mode are the same with the four modes of 16x16 luma prediction modes. Among these modes, the one with the minimum cost value is selected as the best mode.

3.1.3 4x4 integer DCT/IDCT

The transform of residue data is used to reduce the spatial redundancy. All previous

standards such as MPEG-1/2/4 all applied two dimensional Discrete Cosine Transform (DCT) of the size 8x8. In H.264/AVC, however, the size of the transforms is from 2x2 to 4x4 in baseline and main profile. The smaller transform block size is better to fit the smallest block size of motion estimation and to reduce the redundancy of local residue data. Besides, the new transforms with integer coefficients adopted in H.264/AVC standard can reduce the most complexity and hardware cost than traditional DCT transform.

There are three kinds of transforms adopted in H.264/AVC standard. The first type is a 4x4 transform whose matrix coefficients are shown in eq. (5). If the macroblock chooses 16x16 intra prediction mode, a Hadamard transform shown in eq. (6), is applied. This 4x4 matrix transforms all 16 DC terms of luma coefficients of the already transformed 4x4 blocks. The third transform is a 2x2 Hadamard transform which is used for the transform of the 4 DC coefficients of the chrominance component. Its matrix is shown in eq. (7). The three transforms only use integer coefficients and can be easily computed by low complex add, subtract, and shift operations.

Fig. 3-4 shows the transmission order of all coefficients if the macroblock is predicted by intra 16x16 mode. The DC coefficient blocks are first transmitted and the AC coefficient blocks follow no matter in luma or chroma components.

$$M_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \quad (5)$$

$$M_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \quad (6)$$

$$M_3 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (7)$$

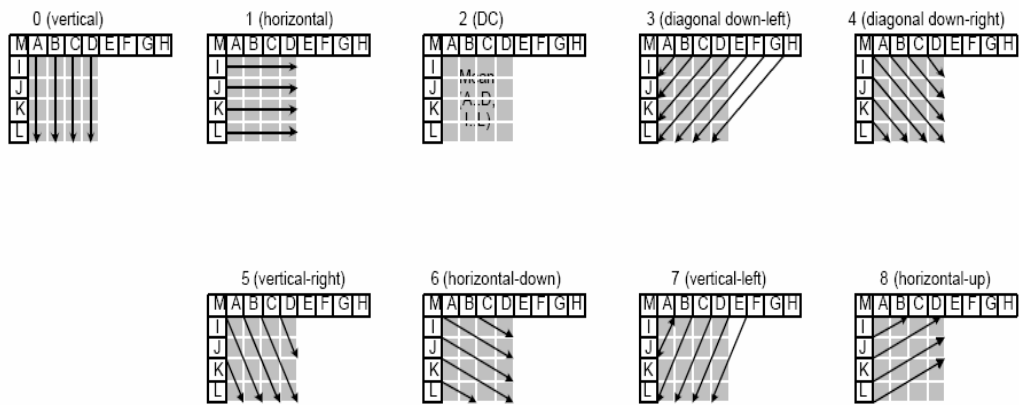


Fig. 3-2 Nine modes for intra luma 4x4 and 8x8 prediction

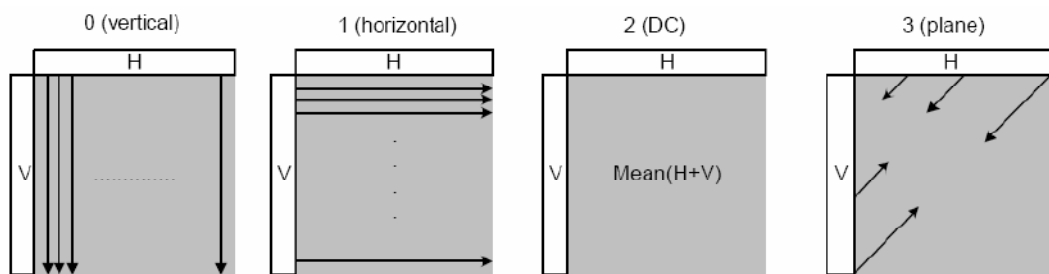


Fig. 3-3 Four modes for intra luma 16x16 and chroma 8x8 prediction

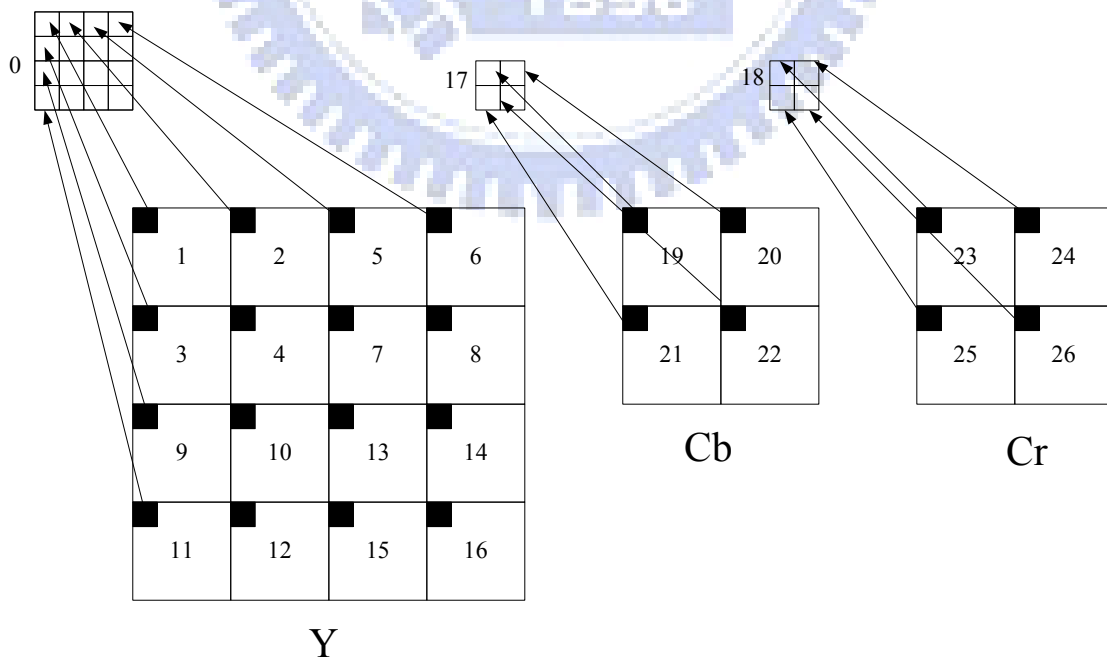
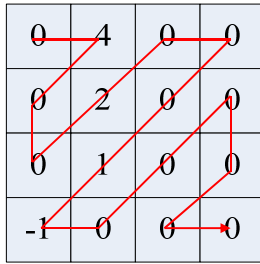


Fig. 3-4. Transmission order of all coefficients in a macroblock predicted by 16x16 intra mode.

CBP NON SKIP!



1. Zigzag reordered block:

0,4,0,0,2,0,0,0,1,-1,...

| Element | Value |
|-----------------------------|------------------------------|
| 2.coeff_token | TotalCoeff=4, TrailingOnes=2 |
| 3.trailing_one_sign_flag(3) | 1 |
| 4.trailing_one_sign_flag(2) | 0 |
| 5.Level (1) | 2 |
| 6.Level (0) | 4 |
| 7.total_zeros | 6 |
| 8.run_before (3) | 0 |
| 9.run_before (2) | 3 |
| 10.run_before (1) | 2 |
| 11.run_before (0) | 1 |

Fig. 3-5 The scan order and the syntax symbols of a non-zero 4x4 block.

3.1.4 Quantization/Inverse quantization

The residue coefficients are processed by a lossy scalar quantizer. The quantization step size is decided by the quantization parameter (QP) which range from 0 to 52. If the QP increases six, the step size will double. It means that an increment of QP by 1 results in approximately 12.5% of increase of the required data rate.

3.1.5 CAVLC

In H.264/AVC, two algorithms of entropy coding are supported. The simpler method called Context-Adaptive Variable Length Coding (CAVLC) is employed in baseline profile. In this algorithm, VLC tables for various syntax elements are switched according to previous syntax elements. Therefore, the coding efficiency of CAVLC is better than traditional VLC coding which uses fixed table.

The CAVLC process has three phases. First, the CAVLC process checks the CBP to decide whether an 8x8 block is all-zero. We can skip an all-zero 8x8 block. If the 8x8 block is not all-zero, the four 4x4 blocks in the 8x8 block are encoded in the next step.

Secondly, the CAVLC process scans a 4x4 block as the steps shown below and gets the syntax symbols as Fig. 3-5 presents:

1. Scan 4x4 coefficients in reverse zigzag scan order.

2. Calculate the number of total nonzero coefficients, TotalCoeffs, and the number of trailing ones, TrailingOnes.
3. Get the sign of each trailing one.
4. Get the level of each nonzero coefficient.
5. After the first nonzero coefficient, get the number of total zeros, TotalZeros,
6. Calculate each run of zeros, RunBefore, between the nonzero coefficients, which will also depend on the number of zeros that have not yet been coded.

Finally, the CAVLC process looks up the codeword of syntax symbol in several tables. Each type of syntax symbol (except each trailing one's sign) has several context-adaptive codeword tables. The selection of codeword tables depends on either the content of the 4x4 block or that of neighbor 4x4 blocks. Then each codeword is concatenated in a specific order (coeff_token, each trailing one's sign, levels, TotalZeros, and RunBefore) to generate bitstream.

3.2 Design Challenges and Paper Survey

3.2.1 Design challenges

The design challenges for H.264 intra frame encoder is the low throughput and huge hardware cost. Several previous H.264 intra frame encoder have been reported in [42]-[44]. However, these designs are still limited to HD720p resolution because of the low parallelism and multi-stage intra prediction algorithms. If extending the parallelism of previous works to support 1080p resolution directly, the area and memory requirement are not affordable because their algorithms include irregular data path and huge required buffer. Therefore, if we want to design an intra frame encoder which can support 1080p resolution with affordable hardware cost, hardware-friendly algorithms for intra prediction, and the balance between parallelism and hardware cost must be considered carefully.

3.2.2 Paper survey

Intra prediction algorithms can be classified as two categories. The first type of fast algorithms adopts the two-step approach [45]-[50]. The first step uses image features to select the possible modes, and the second step computes the selected modes. Thus, in the worst case, it will have to compute all modes [50]. Furthermore, this two-step dependency results in irregular data flow which is not suitable for hardware design. In some cases, the feature calculation in the first step will be too complex for hardware design. For example, the edge-based method [48] uses an arc tangent function and two dividers to calculate the edge for possible mode selections. Thus, these feature calculation circuits introduce extra hardware overhead to the original intra prediction circuit. Even though this arc tangent function can be simplified as shown in [54], this simplification results in an overhead of 15K gates, which is still large when compared to a full mode calculation circuit with 10K gates [55]. The second type of algorithms uses single-step approach [51] [53] [55]. This type of algorithms calculates the modes without extra overhead. However, full search implementations like [53] [55] need large area cost and long computational cycles. Our previous algorithm [51], three step algorithm, divides the mode computation into three steps without complex feature calculation. It just needs to compute constant six modes and is more suitable for hardware design. However, it still introduces pipeline bubble cycles.

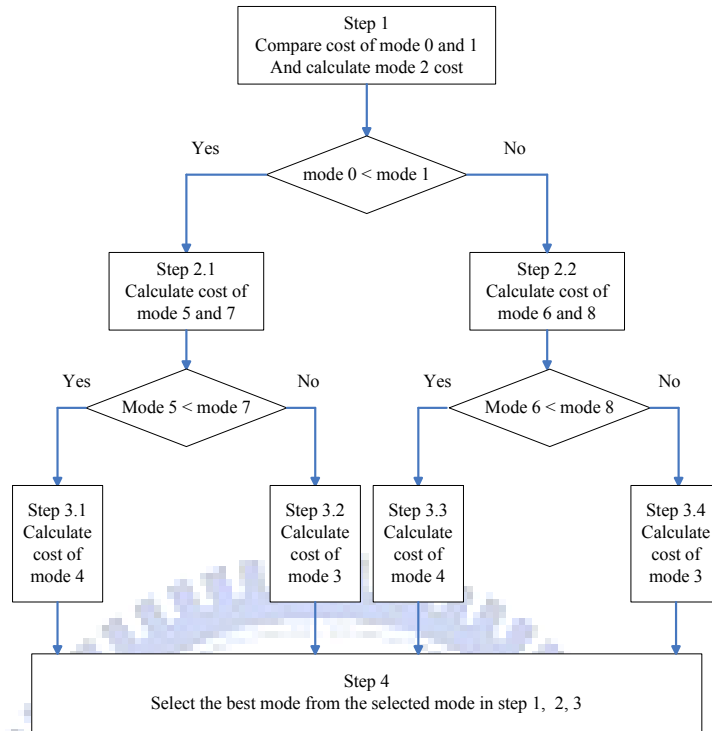
3.3 Fast and Hardware-Efficient Intra Prediction Algorithms

3.3.1 Modified three step algorithm [52]

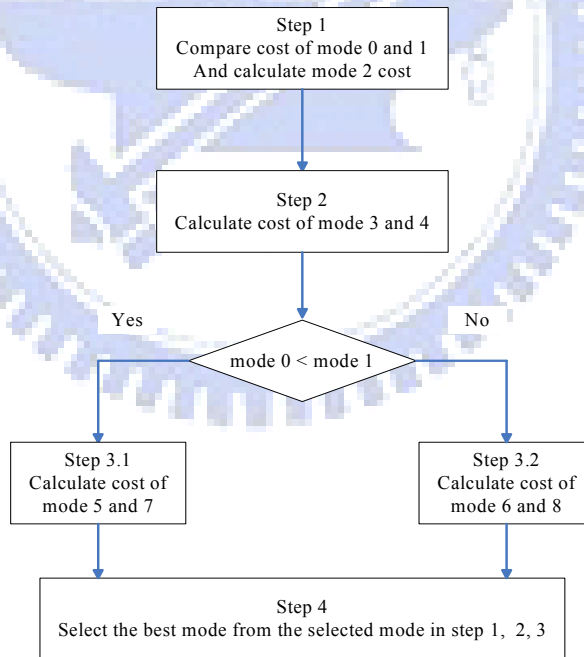
Fig. 3-6 (a) illustrates the original three-step fast intra prediction algorithm [53]. In Fig. 3-6 (a), we first compare the vertical and horizontal modes, and then compare the two neighboring modes around the winner of previous step at the second and third step. Finally, costs of these winners are compared and the minimum one is selected.

This algorithm can save one-third operations with negligible quality loss, only 0.04dB PSNR degradation and 1.58% of bit-rate increase. Further reduction of intra mode test will result in significant quality degradation. Besides, the flow of three step algorithm is regular so that hardware implementation is simpler. Although the three-step algorithm is more suitable for hardware design than the other software-based ones [45]-[50], it still has much room for improvement in practice. Direct applying the algorithm to the hardware will result in pipeline bubble and performance loss.

Fig. 3-7 (a) illustrates a pipeline schedule example when applying the original three-step algorithm to our previous hardware design. For illustration purpose, this example assumes the cost of mode 0 is smaller than that of mode 1, and the cost of mode 5 is larger than that of mode 7 without loss of generality. In the pipeline stage diagram, each block takes eight cycles latency to complete a mode prediction including intra prediction, SATD calculation, and mode decision. In Fig. 3-7(a), the first step will take 12 cycles for three modes, and the second step can be executed immediately in the 11th cycle since the comparison results of mode 0 and mode 1 is finished in the 10th cycle. However, this scheduling leads to four cycle bubbles marked in step 2 of Fig. 3-7(a) because step 2 must wait the comparison results of mode 0 and mode 1. The same situation also occurs between the second step and the third step, and six cycles latency are generated. Therefore, total 28 cycles are needed to predict a block with the original fast algorithm due to the decision flow.



(a)



(b)

Fig. 3-6 Decision flow of (a) original three-step algorithm (b) modified three-step algorithm.

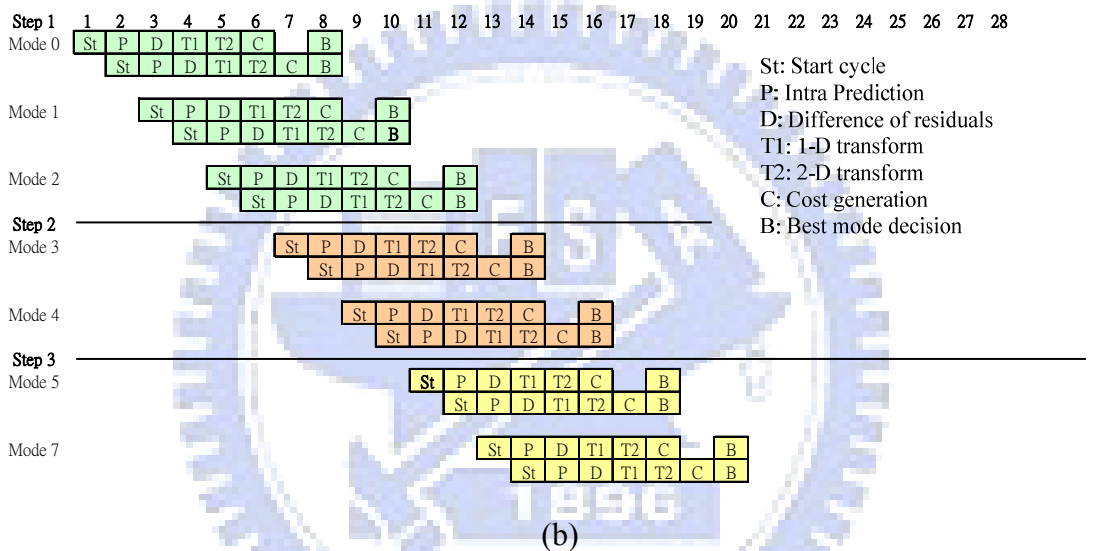
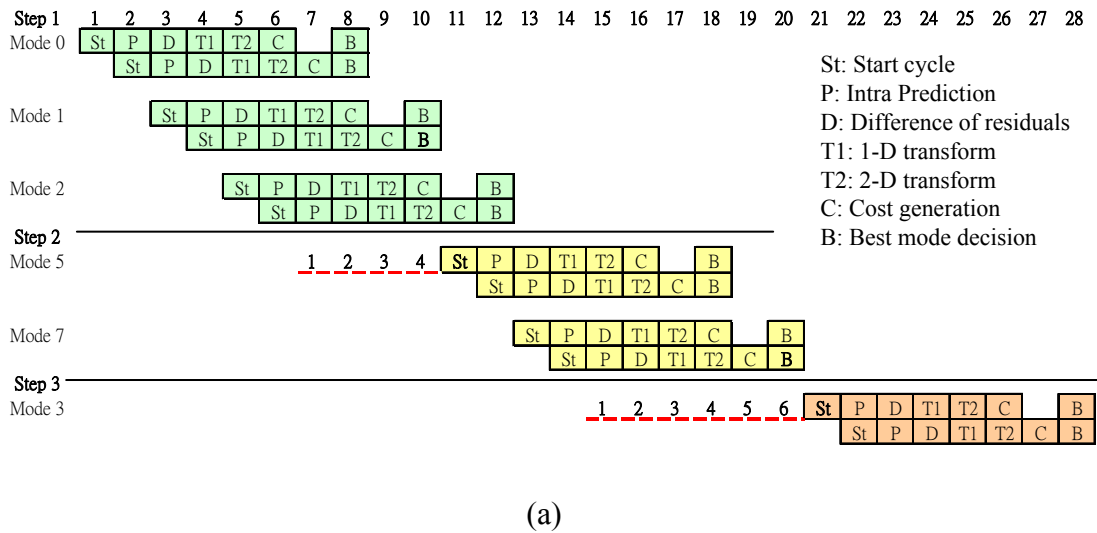


Fig. 3-7 Proposed timing schedule for the modified three-step algorithm.

Therefore, we change the fast three-step algorithm to hide the bubble cycles by adjusting the order of prediction modes in the scheduling. The idea of the modified algorithm [52] is from the analysis of Fig. 3-7 (a). Since the third step in Fig. 3-7 (a) has to predict either mode 3 or mode 4 no matter which branch is chosen, we can move these two modes to the second step and fill the transition bubbles. Fig. 3-6 (b) shows the decision flow of the modified three step algorithm. It is more regular than the original one because we can just calculate the second step, mode 3 and mode 4, without waiting for the comparison result of step 1. Fig. 3-7 (b) shows the pipeline

scheduling for the modified three step algorithm. From the diagram, we can find the total cycles to predict a block are reduced to 20 and no bubble cycle exists, though the number of prediction modes is increased from six to seven. With more prediction modes, the final quality shall be better.

3.3.2 Enhanced SATD algorithm [42]

In intra-only H.264 encoding, the cost function for intra mode decision is an important issue for coding performance. Although adopting rate distortion optimization (RDO) in the cost function can provide the best performance, its corresponding complexity limits its hardware implementation. Thus, the sum of absolute difference (SATD) method is an alternative. The equation for SATD is shown in the below equation:

$$\text{SATD} = \sum_{i=1}^4 \sum_{j=1}^4 |T(s_{ij} - p_{ij})| \quad (8)$$

where s_{ij} and p_{ij} denote the (i, j) th elements of source block and predicted block respectively and the function $T(x)$ in (8) means a 4x4 transform function.

However, the choice of transform $T(x)$ for SATD computation becomes a main issue now. In reference software [27], the Hadamard transform in eq. (6) used for SATD calculation is computationally simple but is much different from the real discrete cosine transform. A better choice of transform matrix for SATD shall consider the effect of transform and quantization used in H.264 encoding to estimate the real bit-rate. Therefore, previous designs [44][45] adopt the 4x4 integer transform shown in eq. (5) as their choice.

Although previous approaches can achieve better performance than Hadamard transform does, they still have space to improve since they do not consider the fractional multiplication factors. A complete transform function for SATD calculation shall include the integer transform and multiplication factors in the quantization formula as shown in (9).

$$Y = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix} \begin{bmatrix} X \\ \end{bmatrix} \begin{pmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{pmatrix} \otimes \begin{bmatrix} a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \\ a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \end{bmatrix} \quad (9)$$

The matrices with factors a and b denote the scalar multiplication.

However, to integrate these factors into the cost function directly will increase a lot of computation because they are not simple numbers for computation. Besides, these factors cannot be directly obtained from (9) because they have already been integrated with the quantization coefficients in H.264/AVC specification. To resolve this problem, we propose a new cost function which combines the integer transform with simplified multiplication factors, which are acquired from quantization and de-quantization coefficients shown in TABLE 3-1 and TABLE 3-2. From these tables, we can derive the factors by exploiting the relationship among the reciprocal of de-quantization coefficients and simplify the equations as below:

$$1/\text{quant_coef}: p(0,0)-1:p(0,1)-1:p(1,1)-1 \sim= 30:19:12 \quad (10)$$

$$1/\text{dequant_coef}: p(0,0)-1:p(0,1)-1:p(1,1)-1 \sim= 30:25:20 \quad (11)$$

where the $p(x,y)$ is the quantization and de-quantization coefficients of different positions in TABLE 3-1 and TABLE 3-2. These simplified scaling factors are shown in (12) which consider the final performance and their hardware cost. In the equation, division by 32 is added to avoid the enlargement of the cost values and can be carried out by simple shifting to reduce computational complexity and hardware cost. Therefore, this cost function can estimate the energy of residuals after the transform more precise than previous methods while keeping computation simple and suitable for hardware implementation.

Besides, this technique can even provide better video quality than the original cost function in the reference software and can compensate the quality loss of other fast intra prediction algorithms discussed in the next subsections.

TABLE 3-1 H.264/AVC quantization coefficients

| QP | Positions (0,0) (2,0) (2,2) (0,2) | Positions (1,1) (1,3), (3,1), (3,3) | Other Position |
|----|--------------------------------------|--|----------------|
| 0 | 13107 | 5243 | 8066 |
| 1 | 11916 | 4660 | 7490 |
| 2 | 10082 | 4194 | 6554 |
| 3 | 9362 | 3647 | 5825 |
| 4 | 8192 | 3355 | 5243 |
| 5 | 7282 | 2893 | 4599 |

TABLE 3-2 H.264/AVC de-quantization coefficients

| QP | Positions (0,0) (2,0) (2,2) (0,2) | Positions (1,1) (1,3), (3,1), (3,3) | Other Position |
|----|--------------------------------------|--|----------------|
| 0 | 10 | 5243 | 8066 |
| 1 | 11 | 4660 | 7490 |
| 2 | 13 | 4194 | 6554 |
| 3 | 9362 | 3647 | 5825 |
| 4 | 8192 | 3355 | 5243 |
| 5 | 7282 | 2893 | 4599 |

$$Y = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -2 \end{bmatrix} [X] \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} 32 & 25 & 32 & 25 \\ 25 & 20 & 25 & 20 \\ 32 & 25 & 32 & 25 \\ 25 & 20 & 25 & 20 \end{bmatrix} / 32 \quad (12)$$

Besides, it can even provide better video quality than the original cost function in the reference software and can compensate the quality loss of plane mode removal discussed in the next subsection.

3.3.3 Plane mode removal technique [42]

The plane mode shown in Fig. 3-3 is derived by the approximation of bilinear function. Though this mode is simplified with only integer arithmetic operations, it is still much more computationally complex than other modes, hard to reuse its results for other modes, and needs almost half of the area in intra prediction unit.

A solution for this problem is to remove the plane mode from the intra prediction,

but this may result in performance loss. TABLE 3-3 shows the probability distribution of all 16x16 prediction modes. The ratio of macroblocks predicted by plane mode is only 4.1% in average and generally not larger than 6.6%. However, the intra prediction without plane mode only increases about 1% of bit-rate than that with plane mode for those video sequences. The loss of 1% bit-rate difference can be easily compensated by the proposed enhanced SATD cost function. By plane mode removal, we can achieve almost the same results as the original one but save lots of computational cycles and area cost.

3.3.4 Performance comparison

TABLE 3-4 shows the simulation results of the modified intra prediction algorithm and combined algorithms for four HD720p sequences when compared with the reference software JM8.6 [56]. The test condition is all I-frames encoding with RDO off. In the simulation, we first consider the effects of the modified three-step algorithm. The proposed one has the negligible quality degradation with at most 1.13% of bit rate increase, which is better than that in the original three-step algorithm. Besides, we combine the effect of previously proposed plane mode removal and enhanced SATD with the modified three-step algorithm as shown in TABLE 3-4. We can find the performance is still good. The bit-rate is increased by no more than 1.18%, and the PSNR degradation does not exceed 0.25 dB in luma part. In some conditions with lower QP, the PSNR is even increased by the improvement of enhanced SATD algorithm. In average, our algorithm can achieve 0.02 dB improvement in Y-PSNR and 0.58% of bit-rate increase.

TABLE 3-3 Probability Distribution of All 16x16 Modes in 720p Sequences with 300 I-frames when QP=28

| Sequence | 16x16 modes | | | | |
|--------------------|-------------|----------|------------|------|-------|
| | Total ratio | Vertical | Horizontal | DC | Plane |
| Mobile Calendar | 25.8% | 11.3% | 6.43% | 4.8% | 3.3% |
| Park run | 7.2% | 2% | 0.8% | 2.7% | 1.5% |
| Shields | 24.1% | 8.6% | 4% | 6.5% | 4.9% |
| Stockholm | 32.8% | 3.9% | 17.1% | 5.2% | 6.6% |
| Average | 22.5% | 6.5% | 7.1% | 4.8% | 4.1% |



TABLE 3-4 The performance of modified 3-step algorithm and combined algorithm for 720p video sequences.

| Sequence | QP | Modified 3-step Algorithm | | | | Combined Algorithm [7] | | | |
|-----------|----|---------------------------|---------------------------|---------------------------|-----------------------------|---------------------------|---------------------------|---------------------------|-----------------------------|
| | | Δ SNR_Y (dB) | Δ SNR_U (dB) | Δ SNR_V (dB) | Δ Bit-Rate (%) | Δ SNR_Y (dB) | Δ SNR_U (dB) | Δ SNR_V (dB) | Δ Bit-Rate (%) |
| Park_run | 16 | 0 | 0 | 0 | 0.23 | 0.04 | 0.1 | 0.13 | 0.16 |
| | 20 | 0 | 0 | 0 | 0.32 | 0.06 | 0.06 | 0.05 | 0.19 |
| | 24 | 0 | 0 | 0 | 0.42 | 0.06 | -0.06 | -0.03 | 0.30 |
| | 28 | -0.01 | 0 | 0 | 0.55 | 0.11 | -0.11 | -0.06 | 0.44 |
| | 32 | -0.01 | 0 | 0 | 0.68 | 0.03 | -0.23 | -0.12 | 0.57 |
| | 36 | -0.01 | 0 | 0 | 0.74 | -0.07 | -0.25 | -0.1 | 0.69 |
| Mobcal | 16 | 0 | 0 | 0 | 0.30 | 0.05 | 0.06 | 0.1 | 0.30 |
| | 20 | 0 | 0 | 0 | 0.41 | 0.12 | 0.14 | 0.05 | 0.39 |
| | 24 | 0 | 0 | 0 | 0.57 | 0.07 | 0.02 | -0.16 | 0.56 |
| | 28 | -0.01 | 0 | 0 | 0.80 | 0.07 | -0.07 | -0.27 | 0.78 |
| | 32 | -0.01 | 0 | 0 | 1.03 | -0.01 | -0.19 | -0.32 | 1.03 |
| | 36 | -0.01 | 0 | 0 | 1.13 | -0.08 | -0.22 | -0.26 | 1.18 |
| Shields | 16 | 0 | 0 | 0 | 0.29 | 0.07 | 0.09 | 0.08 | 0.25 |
| | 20 | 0 | 0 | 0 | 0.42 | 0.17 | 0.04 | -0.04 | 0.39 |
| | 24 | -0.01 | 0 | 0 | 0.60 | 0.08 | -0.08 | -0.16 | 0.63 |
| | 28 | 0 | 0 | 0 | 0.81 | 0.03 | -0.15 | -0.27 | 0.85 |
| | 32 | -0.02 | 0 | 0 | 0.88 | -0.12 | -0.26 | -0.44 | 0.97 |
| | 36 | -0.02 | 0 | 0 | 0.84 | -0.22 | -0.35 | -0.51 | 0.91 |
| Stockholm | 16 | 0 | 0 | 0 | 0.15 | 0.06 | 0.07 | 0.1 | 0.37 |
| | 20 | 0 | 0 | 0 | 0.24 | 0.16 | -0.01 | 0.02 | 0.40 |
| | 24 | -0.01 | 0 | 0 | 0.38 | 0.09 | -0.12 | -0.11 | 0.66 |
| | 28 | -0.01 | 0 | 0 | 0.57 | 0.07 | -0.21 | -0.2 | 0.66 |
| | 32 | -0.01 | 0 | 0 | 0.64 | -0.11 | -0.33 | -0.31 | 0.65 |
| | 36 | -0.01 | 0 | 0 | 0.55 | -0.25 | -0.33 | -0.33 | 0.50 |
| Average | | -0.006 | 0 | 0 | 0.57 | 0.02 | -0.1 | -0.13 | 0.58 |

TABLE 3-5 The performance of modified 3-step algorithm and combined algorithm for 1080p video sequences.

| Sequence | | Station2 | | | | | tractor | | | | | Avg. |
|--------------------|-----------------------|----------|-------|-------|-------|-------|---------|------|-------|-------|-------|--------|
| QP | | 16 | 20 | 24 | 28 | 32 | 16 | 20 | 24 | 28 | 32 | |
| Modified 3-Step | Δ PSNR (dB) | 0 | -0.01 | -0.02 | -0.02 | -0.01 | 0 | 0 | -0.01 | -0.02 | -0.02 | -0.011 |
| | Δ Bit rate (%) | 0.53 | 0.78 | 1 | 0.99 | 0.82 | 0.35 | 0.47 | 0.63 | 0.75 | 0.74 | 0.71 |
| Combined [7] | Δ PSNR (dB) | 0.15 | 0.22 | -0.03 | -0.11 | -0.35 | 0.09 | 0.06 | -0.24 | -0.34 | -0.56 | -0.11 |
| | Δ Bit rate (%) | 0.51 | 0.91 | 1.4 | 1.52 | 1.36 | 0.33 | 0.58 | 1.03 | 1.28 | 1.39 | 1.03 |

TABLE 3-5 shows the simulation results for two HD1080p sequences. The bit rate increasing is within 1% for modified three step algorithm. Besides, even if we also include the plane mode removal and enhanced SATD algorithms, the performance of the combined algorithms only results in 0.11 dB PSNR degradation and 1.03% bit rate increase in average.

3.4 Architecture of Intra-only Encoder

3.4.1 Overview of intra-only encoder with variable pixel parallelism

To further speed up the intra prediction, we adopt some hardware parallelism strategies. However, only the most critical part, intra prediction, adopts the eight-pixel parallelism to double the throughput and thus reduces almost half of computation cycles. Other parts like the quantization and reconstruction use four-pixel parallelism to save the area cost. This architecture with mixed eight-pixel and four-pixel parallelism are thus called *variable pixel parallelism architecture*.

Fig. 3-8 shows the proposed intra frame encoder design with variable pixel parallelism architecture. The whole design works as the data flow in Fig. 3-1. This

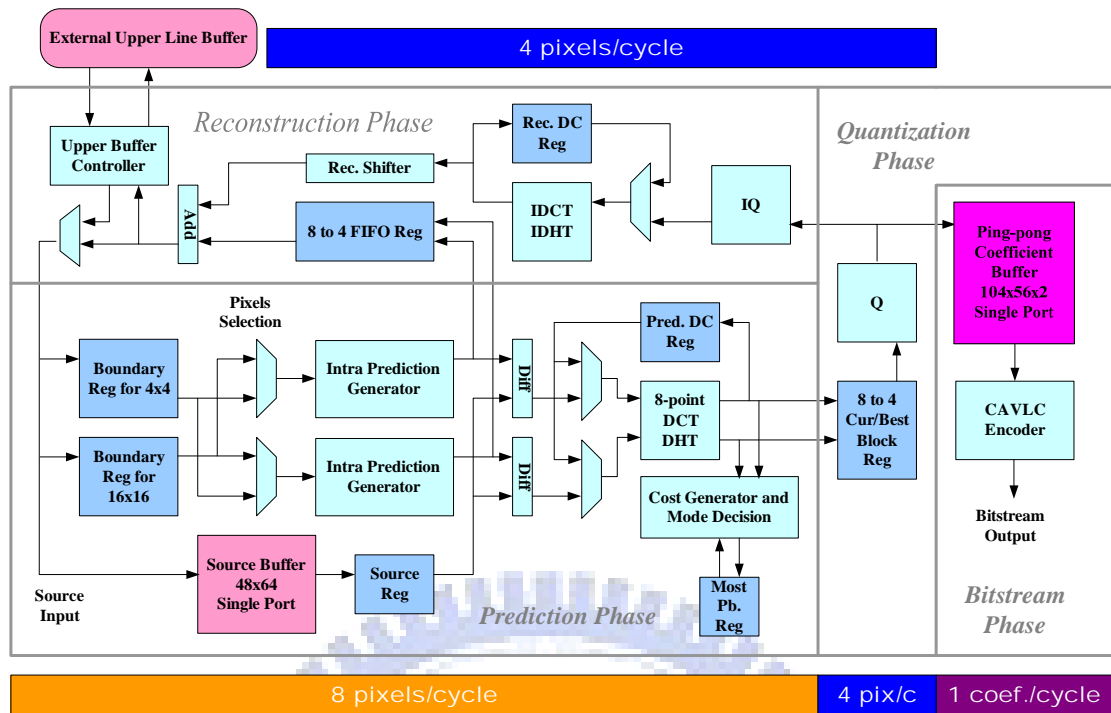


Fig. 3-8 Proposed architecture of encoder with variable pixel parallelism.

architecture is partitioned into four phases: prediction phase, reconstruction phase, quantization phase, and bitstream phase.

This variable pixel parallelism has the benefit of low area overhead and high throughput. This parallelism will not introduce the performance bottleneck at the four-pixel part since only blocks with the best mode will be passed to the quantization phase and reconstruction phase. Data flow between different data parallelism is smoothed by several buffers, including the current block and best block registers in the quantization phase and the FIFO registers in the reconstruction phase. To achieve the eight-pixel parallelism in prediction phase, we only add one more intra prediction engine, two 1-D four-point transform units, and a few small buffers. The gate count overhead of these new components is very little.

3.4.2 Schedule of encoder

Fig. 3-9 shows the scheduling diagram of the proposed encoder. In the intra

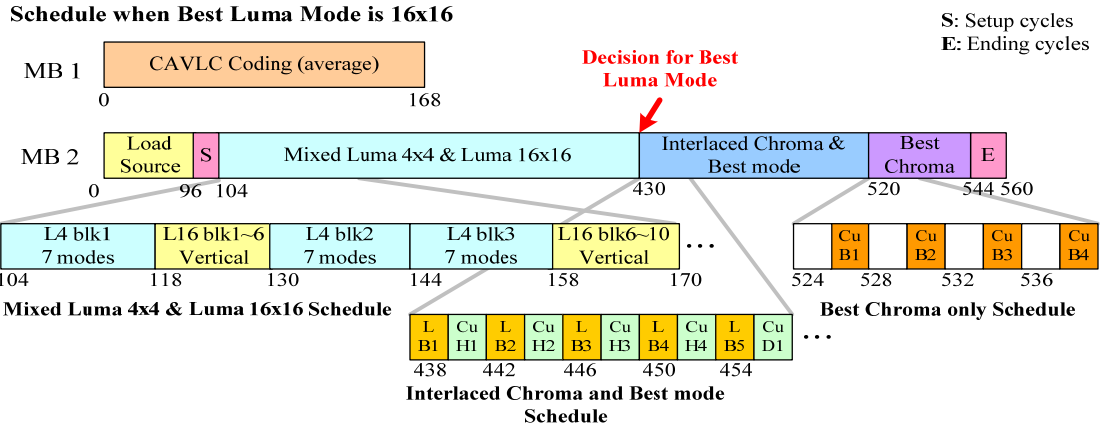
encoder design, the major problem is the data dependency of neighboring blocks since each intra prediction will use the reconstructed data from its left and upper block. During these reconstruction cycles, the intra prediction unit will be idle. Thus, the scheduling challenge is how to hide the reconstruction bubble cycles by keeping the intra prediction unit busy. To maximize performance without pipeline bubble cycles, this design also includes previous techniques proposed in [42], such as insertion of luma 16x16 or chroma 8x8 predictions, early start of next 4x4 block prediction and re-computation of luma 16x16 and chroma 8x8 best modes. The insertion of luma 16x16 or chroma 8x8 predictions techniques will insert luma 16x16 or chroma 8x8 intra prediction into the reconstruction bubble cycles to pre-compute their costs. Thus, utilization of components in the prediction phase is improved. Furthermore, since the 4x4 blocks are processed in the Z-scan order, upper and left boundary samples might not be available at the same time for prediction purpose. Thus, we adopt the early start of next 4x4 block prediction techniques by rearranging the processing order of prediction modes such that prediction modes can be started as early as possible if the needed data is available. For the 4x4 block prediction, we use a small buffer to save the residuals of the best mode. However, when such a strategy applies to 16x16 or 8x8 predictions, a large macroblock-size buffer will be needed. Therefore, we neglect the data generated in the prediction and re-compute them for the best mode of 16x16 and 8x8 macroblocks after the prediction if it is selected as the best mode. This re-computation of luma 16x16 and chroma 8x8 best modes approach may increase the total encoding cycles, but it is still in an acceptable range and can reduce the buffer cost as well.

Besides above previous proposed techniques, the newly adopted variable-pixel parallel architecture also introduces a problem that demands a new scheduling technique. In our variable-pixel parallel architecture, we adopt a block-size buffer at

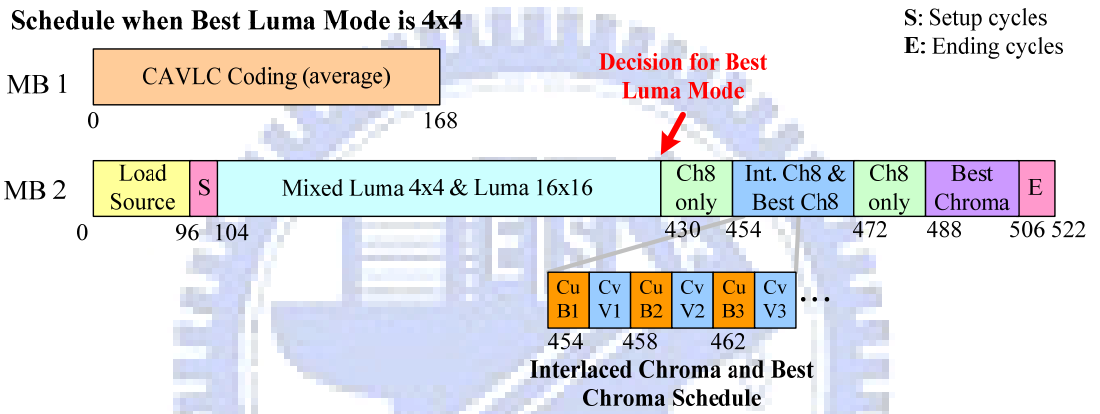
the boundary between four-pixel parallel quantization phase and eight-pixel parallel prediction phase. However, this is not enough since the recomputed coefficients for best luma 16x16 and chroma modes cannot be passed through consecutively because of different parallelism. Thus these coefficients will be blocked. This will result in a larger buffer to store temporarily blocked data or low hardware utilization with empty cycles in the prediction phase.

An efficient solution without utilization loss is to use the interlaced pipelined schedule for best mode as shown in Fig. 3-9. This solution is called *interlaced best mode computation strategy*. We interleave and insert the successively recomputed best modes for luma 16x16 and chroma components into the normal prediction modes that may not pass to quantization phase. With this interlaced scheduling, we will have two different scheduling diagrams for best mode is 4x4 or 16x16, as showed in Fig. 3-9 (a) and (b) respectively. If the best mode is 16x16 as shown in Fig. 3-9 (a), the recomputed 16x16 best modes of luma components are interleaved with modes of normal chroma components. This can improve the hardware utilization in the prediction phase without wasted cycles and keep the data continuity in the quantization phase as well. However, a few empty cycles from cycle 520 to 544 are still needed for the best chroma mode as shown in Fig. 3-9 (a). If the best mode is 4x4 as shown in Fig. 3-9 (b), only the best mode of chroma 8x8 is recomputed and interleaved with normal chroma modes because the best mode of intra 4x4 mode is saved in the buffer and re-computation is not necessary. With the interlaced method, the total cycle count for encoding a macroblock can be reduced to 522 as shown in Fig. 3-9 (b) or 560 in Fig. 3-9 (a), about only 52% when compared to that in previous design [42].

In the previous scheduling, though there are three luma 16x16 modes inserted in the prediction schedule, not all of them are useful for final decision. When the cost of the



(a)



(b)

Fig. 3-9 Pipelined schedule for fast encoder (a) best luma mode is 16x16 (b) best luma mode is 4x4

first-predicted 16x16 mode is obtained, the early termination for second-predicted and third-predicted modes can be asserted. If the currently accumulated cost in the prediction is larger than the previous one, the following operations of this mode will be canceled and the other prediction can be started. This strategy can reduce redundant prediction cycles and extra power consumption.

3.4.3 Architecture of eight-pixel parallelism modules

3.4.3.1 Eight-pixel intra predictor

The eight-pixel parallelism intra prediction generator consists of two four-pixel parallelism units. One is for even row and one is for odd row in a 4x4 block. Fig. 3-10 (a) shows the eight-pixel parallelism intra prediction generator. Its input ports can switch to select any neighboring data in registers for different modes except horizontal and vertical mode. The bypass input ports are used for horizontal and vertical modes because the inputs of these two modes are passed to output directly.

After data input, the computation for each mode is done by selecting the appropriate datapath through multiplexers. Fig. 3-10 (b) shows the example for intra 16x16 DC mode. In DC mode, only the odd-row intra predictor is used to generate the summation of eight neighbor pixels for DC mode. In which, the `dc_reg` is only used for the intra 16x16 DC mode, which needs to sum up 32 neighbor pixels. The sum of eight input data is stored in the `dc_reg`, and then another eight pixels are added with the result in `dc_reg`, and so on. Finally, the total summation for intra 16x16 DC mode is generated after four cycles.

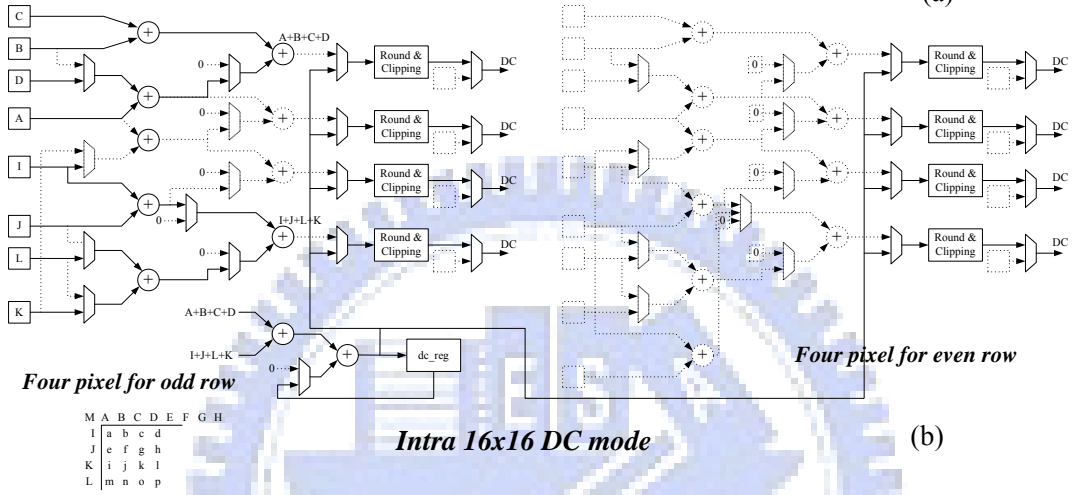
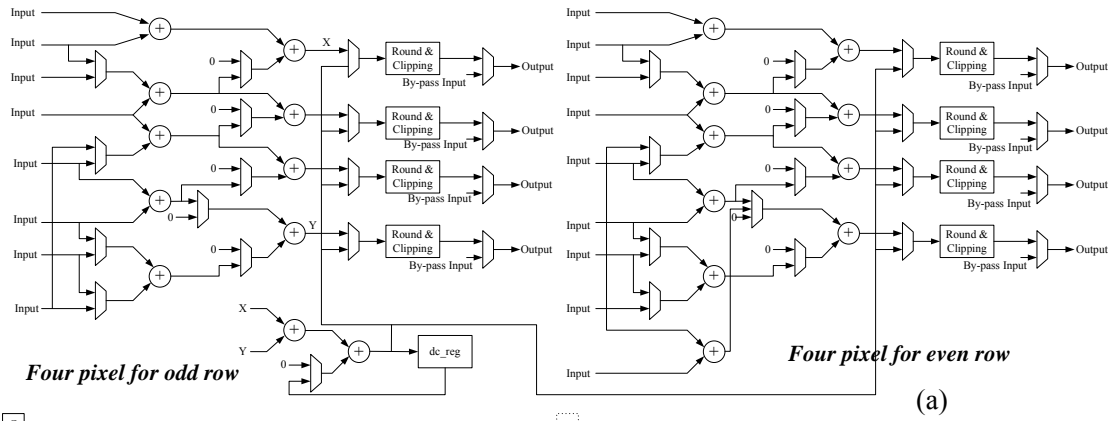


Fig. 3-10 (a) Eight-pixel parallelism intra prediction generator (b) Examples of operations for intra 16x16 DC mode.

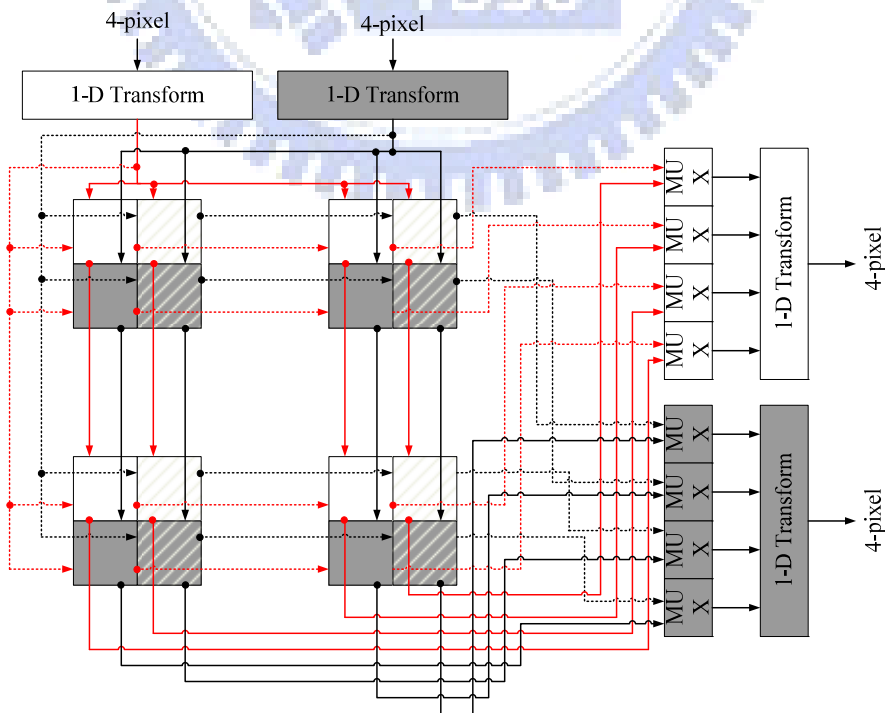
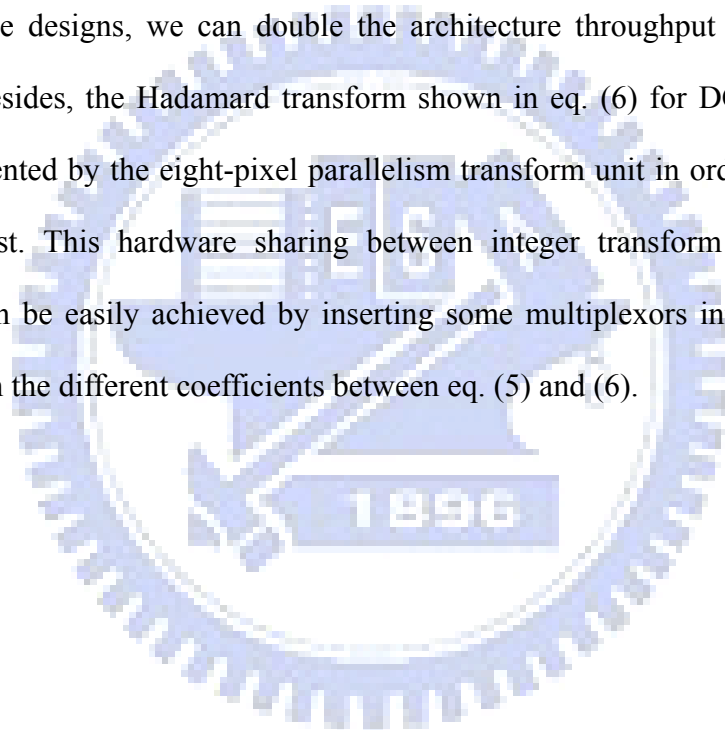


Fig. 3-11 Eight-pixel parallelism transform unit.

3.4.3.2 Eight-pixel DCT

In addition to eight-pixel parallelism intra predictor, the transform unit also adopts eight-pixel parallelism for the same data throughput. The eight-pixel parallelism transform design as shown in Fig. 3-11 includes two 1-D four-point row transform units and two 1-D four-point column transform units for computations. To fit such design, the corresponding transpose registers are decomposed as four 2x2 transpose register array. With such arrangement, we only need two additional 1-D transform units when compared to the design with four-pixel parallelism [42].

With above designs, we can double the architecture throughput with small area overhead. Besides, the Hadamard transform shown in eq. (6) for DC coefficients is also implemented by the eight-pixel parallelism transform unit in order to reduce the hardware cost. This hardware sharing between integer transform and Hadamard transform can be easily achieved by inserting some multiplexors into the transform unit to switch the different coefficients between eq. (5) and (6).



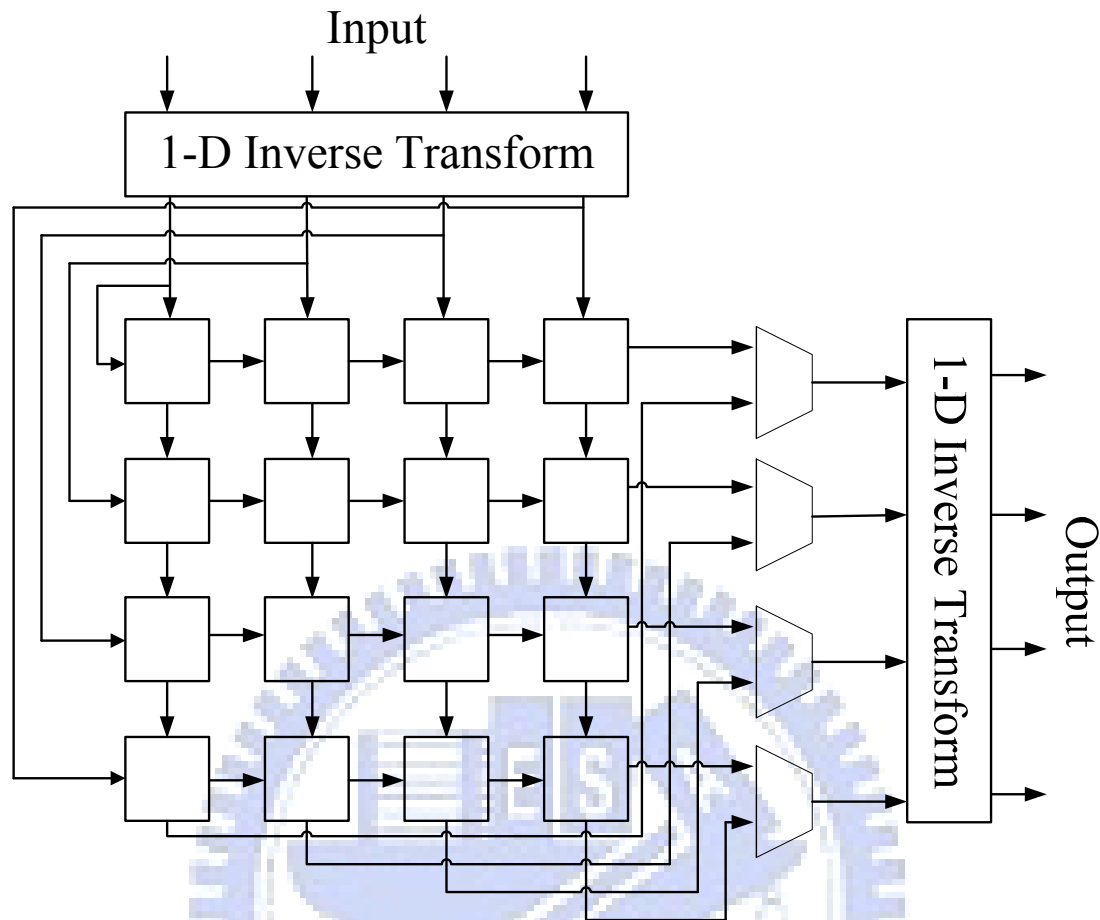


Fig. 3-12 Inverse transform Unit

3.4.4 Architecture of four-pixel parallelism modules

3.4.4.1 Four-pixel IDCT

The transform matrices used in H.264/AVC standard as shown in eq. (5), (6), and (7) are symmetry matrices and can be easily implemented without complex floating point operations. The 2-D inverse transform is generally separated into two 1-D inverse transform with fast algorithm and butterfly architecture [57]. Though there have been several transform designs for H.264 codec [58]-[60], our design adopts the four-pixel architecture presented in [61] to execute inverse integer transform. It is because this architecture is simple and with low hardware cost. Fig. 3-12 shows the architecture of inverse integer transform.

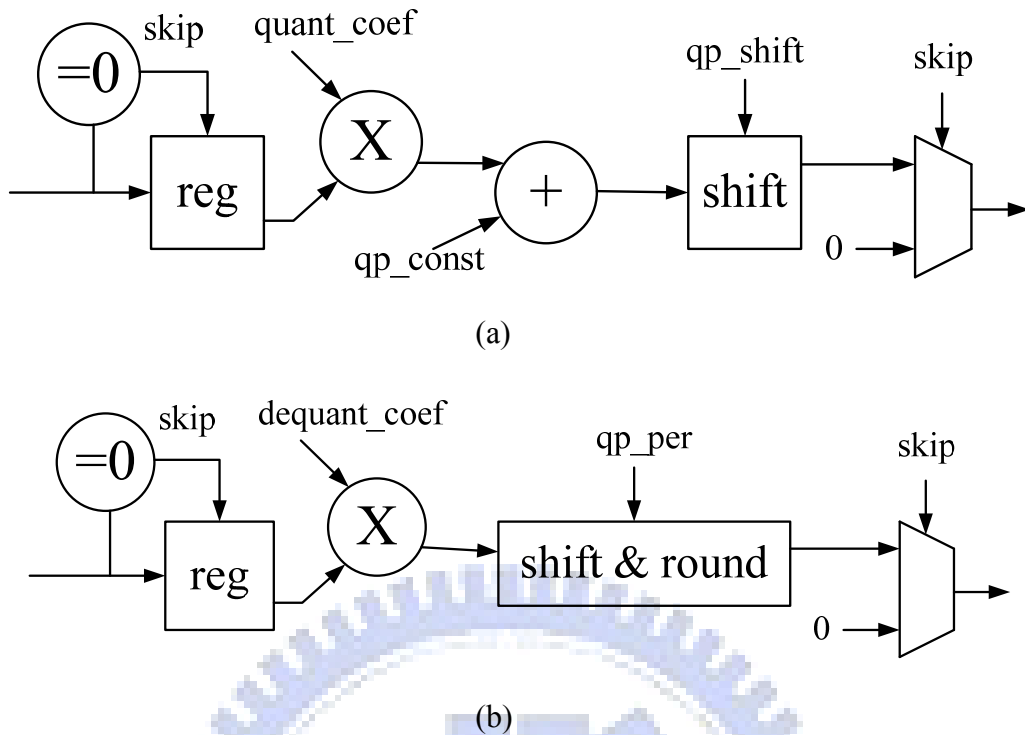


Fig. 3-13 (a) Quantization and (b) inverse quantization unit

3.4.4.2 Q/IQ

Fig. 3-13 shows the quantization and de-quantization units. In which, the `quant_coef`, `dequant_coef`, `qp_const`, `qp_shift`, and `qp_per` denote the quantization parameters (QP). And then, we establish the QP-dependent look-up tables to implement the constant quantization coefficients. The quantized coefficient is derived through a multiplication with `quant_coef`, an addition of `qp_const`, and a shifter. In the de-quantization unit, the data is also passed through a multiplication followed by rounding and shift module.

3.4.5 Architecture of CAVLC module

Fig. 3-14 presents the overall architecture of entropy encoder in a H.264 baseline encoder [62]. The CAVLC module accepts a 4x4 block residue from the residue buffer. When the residue data is sent to the CAVLC encoder, the corresponding CBP also sends to encoder to check the 8x8 block. The output bitstream is packaged with the output of UVLC to generate the final bit stream.

TABLE 3-6 Zero-block Codeword Table

| $0 \leq nC < 2$ | $2 \leq nC < 4$ | $4 \leq nC < 8$ | $8 \leq nC$ | $nC = -1$ |
|-----------------|-----------------|-----------------|-------------|-----------|
| 1 | 11 | 1111 | 000011 | 01 |

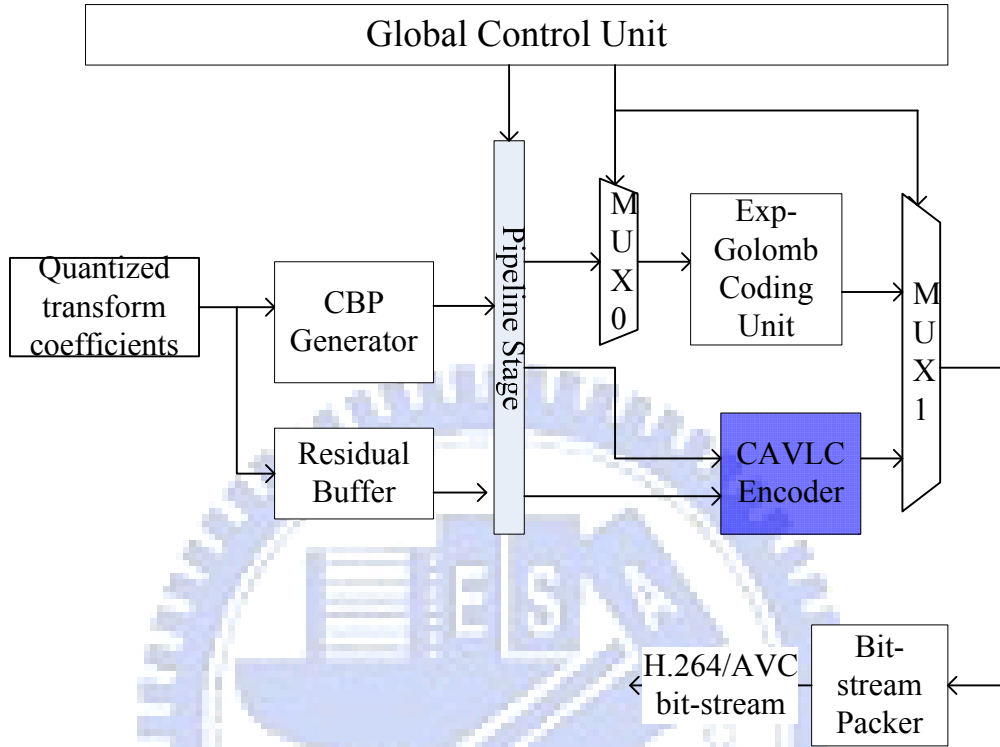


Fig. 3-14 Overall architecture of entropy encoder in H.264 baseline encoder.

Fig. 3-15 presents the detailed architecture of the CAVLC encoder. The first step is to check the CBP. If the CBP of 8x8 block is nonzero, its coefficients is passed to zig-zag scan. At the same time, its nonzero index table is generated, and the 4x4 zero block checking is finished. Then if a zero 4x4 block is detected, the output codeword will be generated directly by the zero block codeword table in TABLE 3-6.

The general CAVLC encoding can be divided into two processes, scanning and coding process. However, unlike the previous approach as in [63]-[66] that requires a large static buffer to process pipelining, the proposed approach directly integrates the two processes together for direct encoding and thus no intermediate buffer is required.

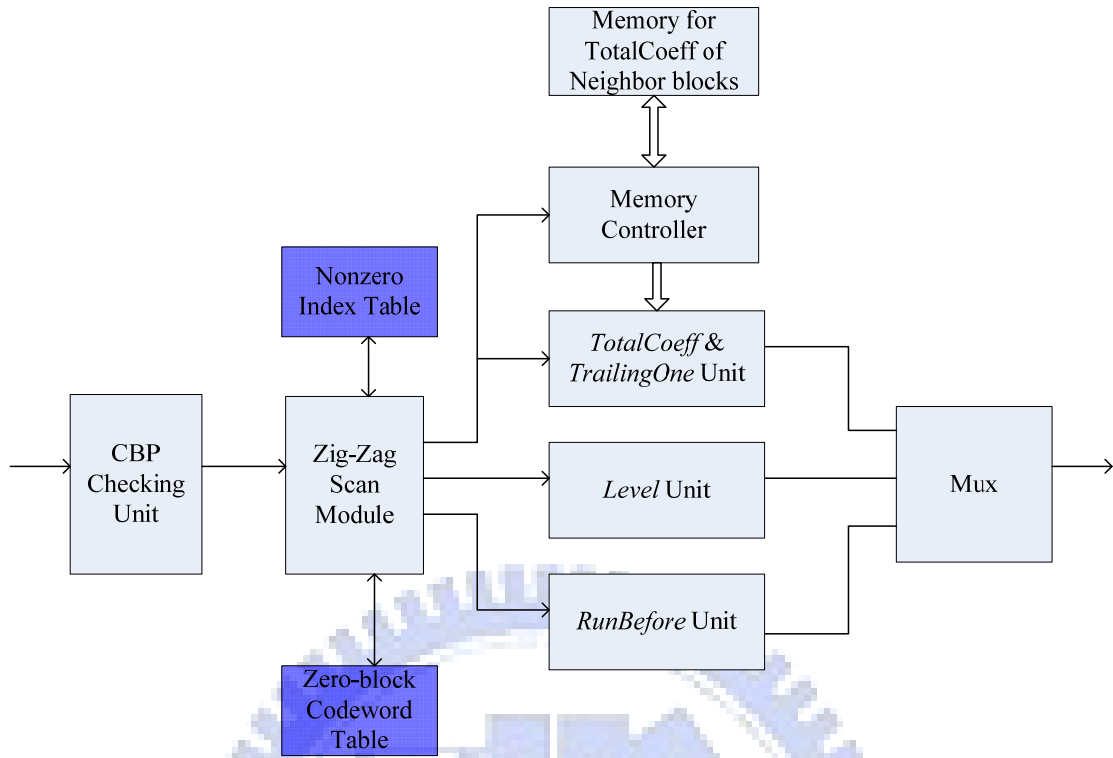


Fig. 3-15 The overall architecture of CAVLC encoder

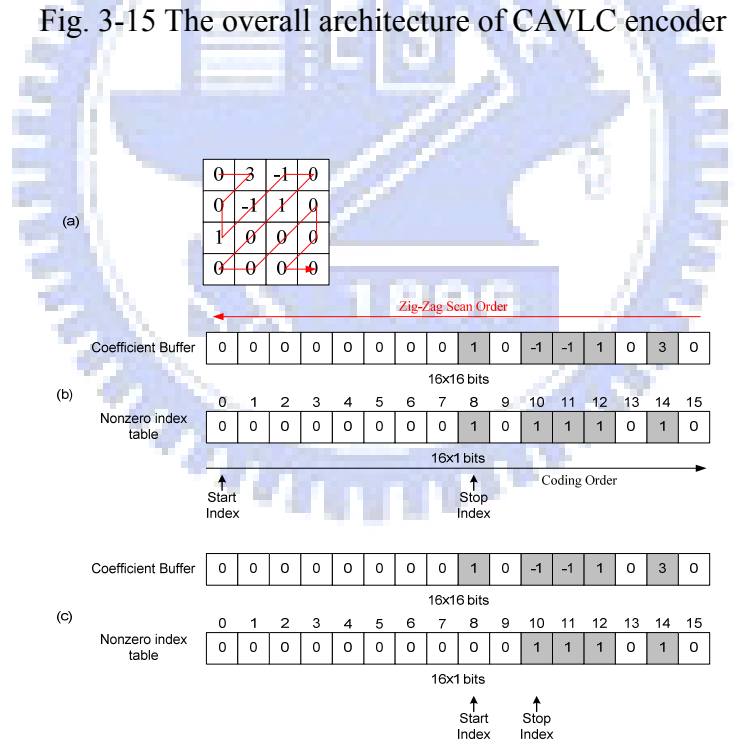


Fig. 3-16 An example for nonzero index table: (a) Original 4x4 block and zig-zag scan (b) the initial table after all coefficients are loaded and (c) the updated table after first iteration of leading one detection.

Besides, each components in Fig. 3-15 can be simplified and speedup by nonzero index table shown in Fig. 3-16.

3.5 Implementation Results and Comparison

3.5.1 Implementation results

The proposed intra frame encoder was designed using Verilog HDL and implemented using 0.13 μ m CMOS technology. Fig. 3-17 shows the effects of the proposed techniques. The variable parallelism architecture and the modified three step algorithm can reduce the cycle count by 39.3% and 10.3% respectively. Although the interlaced scheme only reduces the latency by 4.9%, it can increase the hardware utility and remove bubble cycles. With these three techniques, we can process a macroblock (MB) with 560 cycles. Thus, the final design can achieve HD720p 30 frames/sec encoding at 61MHz and HD1080p 30 frames/sec encoding at 140MHz. For digital still camera applications, our design can process a 4096x2304 image with 6.78 frames per second. The total gate count is 94.7K for HD1080p 30 frames/sec encoding at 140MHz. TABLE 3-7 lists the final results of gate count for each component. Most of the area is spent on boundary prediction buffer, quantization, DCT, and cost generator for mode decision as shown in TABLE 3-7. Fig. 3-17 shows the layout of this design.

TABLE 3-7 Gate count table for the encoder for HD1080p at 140MHz.

| Component | Gate Count |
|-------------------------|------------|
| Boundary Buffer | 12,015 |
| Predictor | 6,005 |
| Cost Generation | 13,865 |
| Schedule Control Unit | 1,532 |
| DCT | 13,970 |
| IDCT | 5,347 |
| DC register | 6,566 |
| Quantization | 23,263 |
| Reconstruction and FIFO | 3,528 |
| CAVLC encoder | 7,474 |
| Total Design Gate Count | 94,729 |

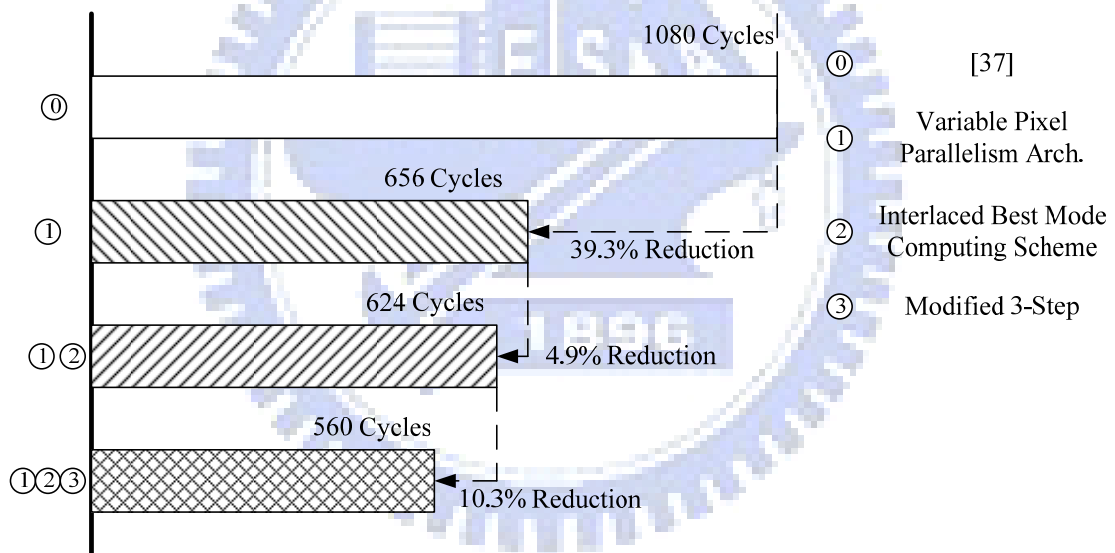
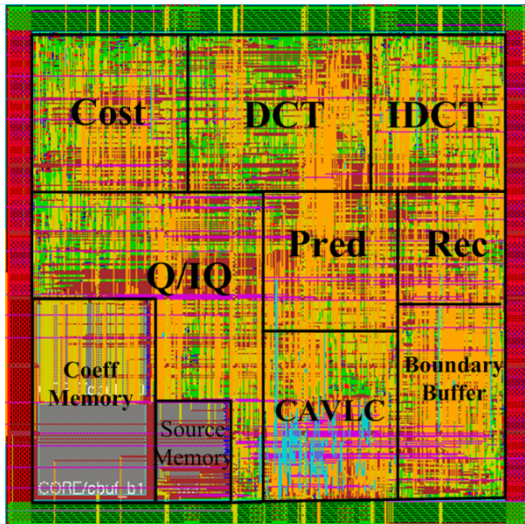


Fig. 3-17 The cycle reduction by adopted techniques.



| | |
|----------------|---|
| Technology | TSMC 0.13um 1p8m CMOS |
| Core Voltage | 1.2V |
| I/O Voltage | 3.3V |
| Core Size | 0.85x0.85mm ² |
| Package | CQFP 144 |
| On-Chip Memory | Single-port 104 x 56 bit x 2 Single-port 48 x 64 bit |

Fig. 3-18 The layout and its design specification.

3.5.2 Comparison with previous works

TABLE 3-8 shows the comparison to other designs. For the same HD720p 30 frames/sec encoding requirement, this design can reduce 48% of operating frequency compared with [42] (encoding part, only) because of lower latency. With lower operating frequency, the critical path timing is thus relaxed and the area cost is 23.5% lower than [42]. Moreover, our design can support HD1080p 30 frame/sec encoding at 140MHz but with similar gate count as [42]. Compared to the standard definition (SD, 720x480) sized encoder in [43], this design reduces the working frequency by 57.6% for SD sized support. Besides, comparing with another HD-sized design [44], our gate count reduction reaches 50.8%.

For comparisons with intra predictor part only, TABLE 3-9 shows that our design needs 19.8K gates (including both intra predictor and cost generation) for HD1080p encoding. Compared to full search design with HD720p intra predictor only [55], our intra predictor design with 6K gate count for HD1080p can save 40% gate count. Compared to [53], which adopts the fast algorithm [48], the design uses more highly parallel hardware and thus its gate count is 122% larger than ours. Moreover, this design does not consider the overhead of feature calculation in the fast algorithm.

Finally, one simplified feature calculation circuit has been shown in [54]. In [54], it only deals with feature-based mode selection without considering the intra predictor and cost generation part. This circuit costs 15K gates, which is a large overhead. Note that in TABLE 3-9, the term “*Not implemented*” means that that item should be included for a complete intra prediction design but is not done in that reference.

3.6 Summary

A high throughput and low cost H.264/AVC intra frame encoder is presented in this chapter with just 94K gate and 0.72mm² core area at 140MHz. We have applied techniques such as fast prediction algorithm, variable pixel parallelism and other scheduling techniques to optimize this design. Compared to previous design for HD720p 30 frames/sec, this work can reduce 23.5% of gate count but only with 52% of working frequency. With these improvements, the new design can support digital video recorder applications with HD1080p 30 frames/sec resolution in real time. Besides, the work also can support digital still camera application with 4096x2304 resolution at 6.78 frames/sec. Further extension to full intra-only profile is straightforward by including 8x8 transform and intra prediction and higher bit width per pixel.

TABLE 3-8 Comparison with previous intra encoders

| design Feature | This Work [7] | [42] | [43] | [44] |
|-----------------------------|--------------------------------------|--------------------------------------|---|-----------------------|
| Max operating Freq. | 140MHz | 125MHz | 55MHz | n.a. |
| Pixel parallelism | 8-pixel/4pixel | 4-pixel | 4-pixel | 4-pixel |
| CMOS technology | TSMC 0.13 μ m | UMC 0.18 μ m | TSMC 0.25 μ m | Hynix 0.35 μ m |
| Chip Core size | 0.85x0.85mm ² | 1.28x1.28mm ² | 1.86x1.86mm ² | n.a. |
| Gate count | 66.2K@61MHz 94.7K@140MHz | 86.6K | 85K | 192K |
| On-Chip memory usage | Single 48x64(x1) Single104x56(x2) | Single 96x32(x1) Single104x64(x2) | Single 96x32(x2) Single 64x32(x1) Dual 96x16(x4) | 27.6K bits |
| Max target Size | HD1920x1080 | HD1280x720 | SD 720x480 | HD1280x720 |
| Freq. for HD 1080p@30fps | 140MHz 61MHz | n.a. 125MHz | n.a. n.a. | n.a. 108MHz |
| Freq. for HD 720p@30fps | 23MHz 6.7MHz | 43MHz 12.8MHz | 54MHz 15.8MHz | |
| Freq. for SD@30fps | | | | |
| Freq. for CIF@30fps | | | | |
| Processing cycles/MB | <560 cycles | <1080 cycles | <1300 cycles | <927 cycles |
| Cost Function | Enhanced DCT-based SATD | Enhanced DCT-based SATD | DCT-based SATD | DCT-based SATD |
| Mode decision method | Modified 3-step | 3-Step | Full search | Full Search |

TABLE 3-9 Comparison of intra predictor part with the state-of-the-art

| Design Feature | This Work [7] | [55] | [53] | [54] |
|---------------------------------------|----------------------------------|-----------------|----------------------|---------------------------|
| Max operating Freq. | 140MHz | 120MHz | n.a. | 200MHz |
| Pixel parallelism | 8-pixel | 4-pixel | 10-pixel | n.a |
| CMOS technology | TSMC 0.13um | UMC 0.18um | TSMC 0.18um | TSMC 0.18um |
| Gate count for intra predictor | 6K for HD1080 3K for HD720 | 10K for HD720 | 28.51K for HD1080 | Not implemented |
| Gate count for cost and mode decision | 13K for HD1080 9.8K for HD720 | Not implemented | | Not implemented |
| Gate count for fast mode decision | 0 | 0 | Not implemented | 15.8K |
| Max target Size | HD1920x1080 | HD1280x720 | HD1920x1080 | HD1920x1080 |
| Processing cycles/MB | <560 cycles | <896 | < 256 | n.a. |
| Mode decision method | Modified 3-step + SATD | Full search | Edge Detection + SAD | Simplified Edge Detection |



Chapter 4

H.264 HD1080p High Profile

Encoder Chip

H.264/AVC high profile standard is the latest extension of H.264/AVC for high resolution video applications. This standard has been adopted in a lot of video applications such as Blu-ray, HD-DVD, and DVB-H. These new coding tools improve a lot of coding efficiency especially for high resolution video. However, these tools also result in huge computation power and require extremely high throughput for high definition (HD) applications. Due to these requirements, ASIC design is the only solution to process H.264/AVC high profile encoding in real time.

Therefore, in this chapter, we propose a H.264 high profile encoder chip which can support 1080p video at 30 frames per second [8]. This design includes the most techniques discussed in previous two chapters no matter in algorithm and architecture level. Besides, the new coding tools of high profile are added to the encoder without resource conflict and large hardware overhead.

This design adopts three stage pipelining schedule. In the first stage, the integer motion estimation module mentioned in Chapter 2 is adopted and extended to support bi-directional motion estimation. The second stage includes the fractional motion estimation module and full eight-pixel parallelism intra predictor which are introduced in Chapter 2 and Chapter 3 respectively. Finally, the third stage consists of the deblocking filter and two entropy tools, CAVLC and CABAC.

Except the individual modules, the system integration and hardware sharing

between pipeline stages are important issues in our design. By the integration, the hardware cost and power can be reduced a lot when comparing with previous similar designs.

4.1 Overview of H.264/AVC High Profile

4.1.1 History of H.264/AVC high profile

After the completion of H.264/AVC standard in May 2003, the JVT group focuses on an extension for coding of high definition video material, especially in application areas like professional film production, video post production, or high-definition TV/DVD. The work on the Fidelity Range Extensions (FRExt) of H.264/MPEG4-AVC was completed in July 2004, and its final draft amendment text was released in September 2004 [67].

4.1.2 Introduction of the coding tools of H.264 high profiles and levels

Fig. 4-1 shows four major four profiles defined in H.264/AVC, which are baseline, main, extended and high profiles. Baseline profile consists of basic coding tools and features, such as intra prediction, forward inter prediction, deblocking filter, and CAVLC. Main profile includes all coding tools of baseline profile and other advanced techniques, such as weighted bi-directional inter prediction, CABAC, and et al. The third profile, extended profile, contains all tools of main profile except CABAC. This profile is designed as the streaming video profile with new tools for robustness to data losses and server stream switching.

Finally, the high profile is the latest and the most complex profile. The new tools such as intra 8x8 prediction types, transform and quantization with 8x8 block size,

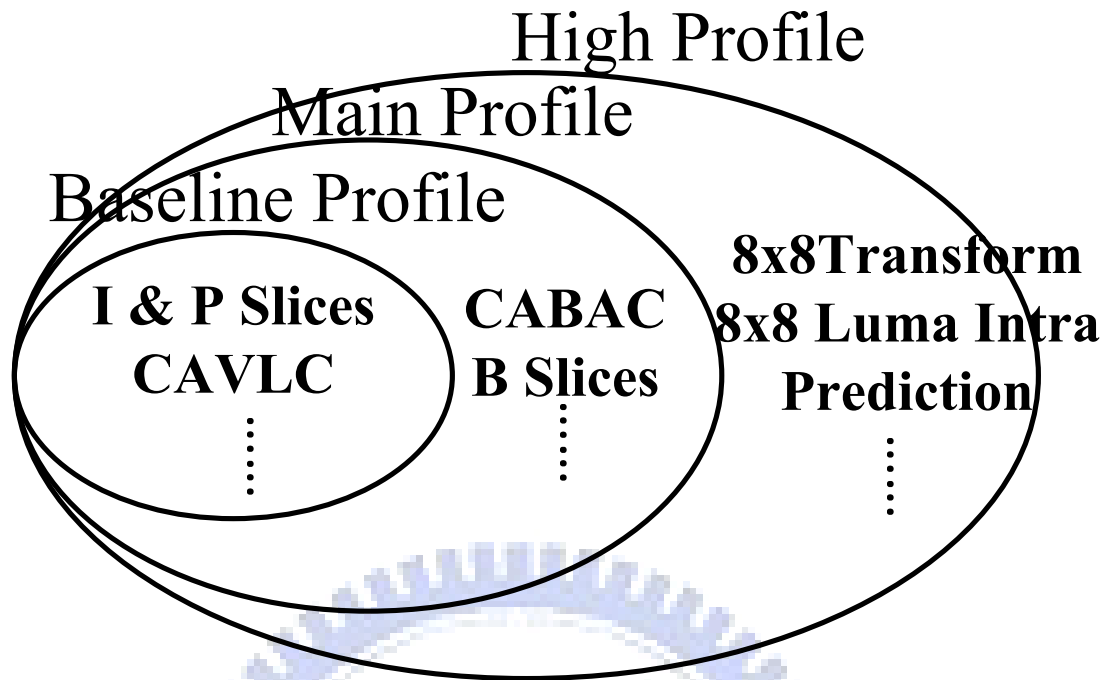


Fig. 4-1 Profiles of H.264/AVC

and others are supported in this profile. The high profile can provide more bit-rate saving and better video quality than baseline and major profiles. However, this new profile requires much more computation efforts. Therefore, H.264/AVC high profile has been widely used in multimedia applications especially for high quality and low bitrate requirements.

4.1.3 Introduction to new tools of H.264/AVC high profile encoder

4.1.3.1 8x8 intra prediction

In the high profile of H.264, a new prediction block size of 8×8 was used for spatial luma prediction by extending the concepts for 4×4 intra prediction in baseline profile.

As shown in Fig. 4-2, the luma 8×8 block is predicted from neighboring reconstructed reference pixels, where nine modes can be selected by the encoder. We should note that the selection of the inter or intra prediction block size (4×4 , 8×8 , or 16×16) also influences the corresponding luma transform size.

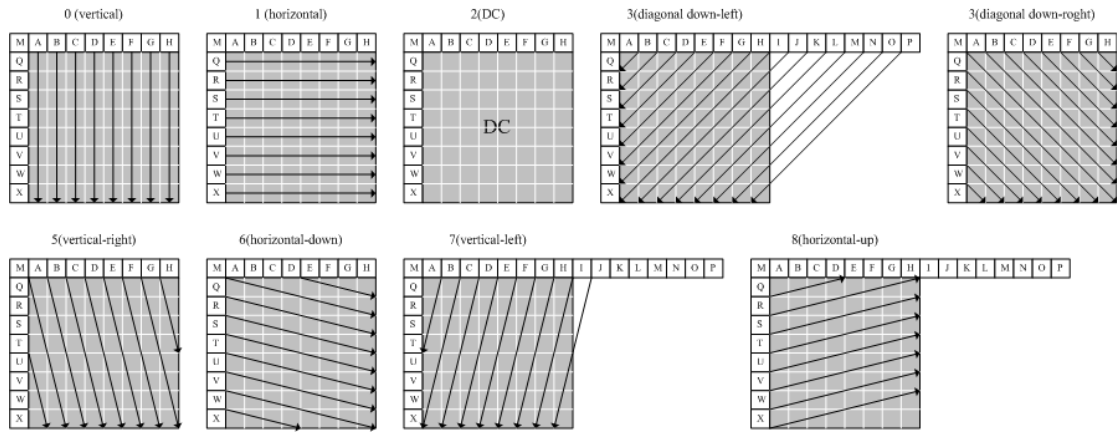


Fig. 4-2 Nine modes for intra 8x8 prediction.

4.1.3.2 8x8 transform

For high resolution video, the details and textures which can be processed by the function with larger basic block unit become more important than that in low resolution video. Therefore, the high profile includes an 8×8 integer transform and the following quantization function and allows the encoder to switch adaptively between the 4×4 and 8×8 transform for luma samples in a macroblock level [68].

The 2-D 8×8 transform also can be executed as a 1-D horizontal transform followed by a 1-D vertical transform, where the 1-D transformation matrix is shown in (13):

$$T_{8 \times 8} = \begin{bmatrix} 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 12 & 10 & 6 & 3 & -3 & -6 & -10 & -12 \\ 8 & 4 & -4 & -8 & -8 & -4 & 4 & 8 \\ 10 & -3 & -12 & -6 & 6 & 12 & 3 & -10 \\ 8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\ 6 & -12 & 3 & 10 & -10 & -3 & 12 & -6 \\ 4 & -8 & 8 & -4 & -4 & 8 & -8 & 4 \\ 3 & -6 & 10 & -12 & 12 & -10 & 6 & -3 \end{bmatrix} \quad (13)$$

Because (13) consists of integer coefficients, both the forward and inverse 8×8 transform can be efficiently implemented by shift and add operations.

After 8x8 transforms, the following processes such as scaling, quantization, and scanning of 8×8 transform coefficients are extended directly from that defined for the 4×4 transform. Besides, two restrictions for the transform size

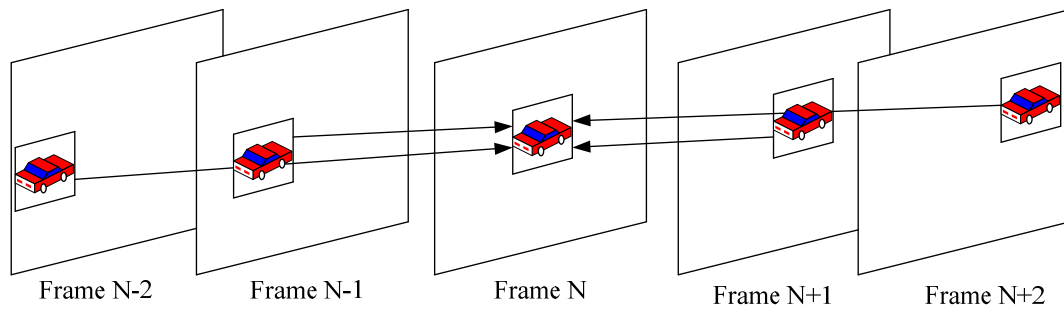


Fig. 4-3 Bi-directional motion estimation

selection are listed:

1. For inter-predicted macroblocks, they adopt 4x4 luma transform if at least one of their sub-blocks is smaller than 8x8.
2. For intra-predicted macroblocks, they choose the 8x8 luma transform if and only if 8x8 luma intra prediction is used.

4.1.3.3 Weighted bi-directional motion estimation

The H.264 motion estimation supports bi-directional motion estimation so that both the backward and forward prediction can be used to improve the coding efficiency as shown in Fig. 4-3. Except the generalized bi-directional motion estimation, the H.264 also supports the weighted prediction so that prediction result of different reference data can be averaged and weighted to optimize the prediction result. Besides, the direct mode can be used to reduce the complexity load overhead from the bi-directional prediction.

4.1.3.4 Context adaptive binary arithmetic coding (CABAC)

CABAC is used as one of the entropy coding method for H.264 video coding that is consisted of three stages: binarization, context modeling and arithmetic coding (AC) as shown in Fig. 4-4. First, a given non-binary value syntax element will pass to

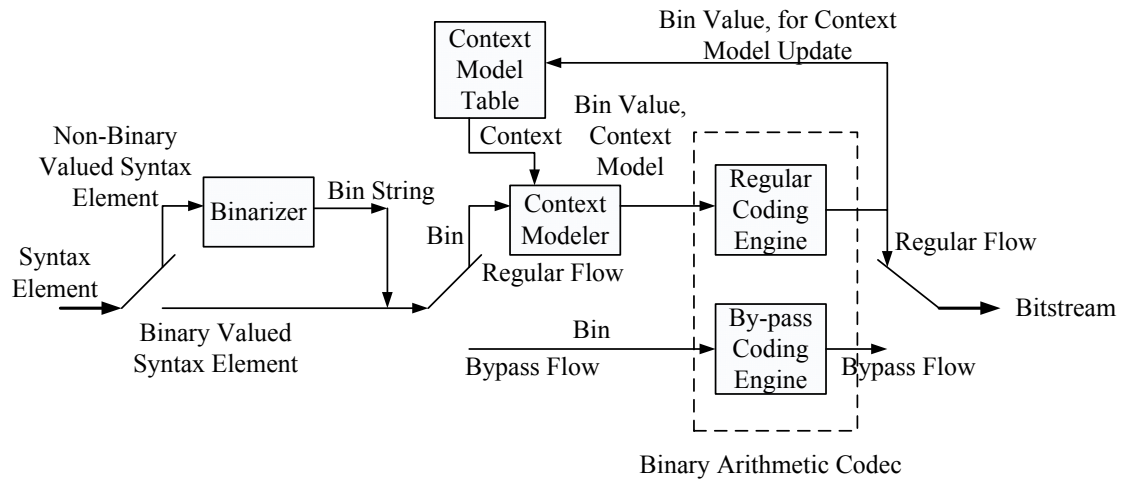


Fig. 4-4 Block diagram of CABAC

binarization to form a uniquely bin-string. Second, except for suffix of syntax element for motion vector and level information, all of bins from binarization will enter into decision mode, and a probability model will be selected to assign context model. The selection of probability models depends on previously encoded syntax elements or bins. After receiving bin and context, AC can encode and output the compressed data directly. AC consists of two sub-engines and is classified into three modes. These two engines are called decision coding engine and bypass coding engine, while the three modes are:

1. “decision” mode” that includes adaptive probability models and interval maintainer.
2. “bypass” mode for fast encoding of symbols.
3. “termination” mode for ending of encoding.

- Binarization

For a given non-binary valued syntax element, H.264/AVC adopts four schemes to do binarization. Such fours schemes are:

1. The unary code word consists of x “1” bits plus a terminating “0” bit for a given unsigned integer x.
2. For truncated unary (TU) code, unary code is used only when $x < cMax$. If $x=cMax$,

the terminating “0” bit is neglected.

3. A unary/K-th order Exp-Golomb (UEG_k) bin-string is a concatenation of a prefix bit string with TU and a suffix bit string with Exp-Golomb code.

4. The fixed length (FL) codeword of x is simply x with a fixed (minimum) number $FL_{bits} = \log_2(c_{Max} + 1)$ of bits.

- Context modeling

In the context modeling, the encoder should calculate context index (ctxIdx) from 0 to 460. With ctxIdx as memory address, it can get probability state (pStateIdx) and Most Probable Symbol (MPS) from context table. The pStateIdx is in range from 0 to 63, and MPS is either 0 or 1. CABAC provides two equations to calculate ctxIdx. Except for syntax element coded_block_flag, last_significant_flag, significant_flag and coeff_abs_level_minus1, eq (14) is used for calculating ctxIdx. Otherwise, eq (15) is used.

$$ctxIdx = ctxIdxOffset + ctxIdxInc \quad (14)$$

$$ctxIdx = ctxIdxOffset + ctxIdxInc + ctxCatOffset \quad (15)$$

In (13) and (14), both of ctxIdxOffset and ctxCatOffset are constant for calculating ctxIdx. The ctxIdxInc is calculated from the information of neighbor macroblock

- Arithmetic coding (AC)

Fig. 4-5 shows the flow diagram of AC encoding for a given bin value, binVal, in the Decision mode. AC is consisted of three parts.

1. Interval Maintainer.
2. Probability Updating
3. Renormalization.

4.1.3.5 Deblocking

Deblocking filter is used to the decoded macroblocks to reduce blocking distortion. This filter is applied after the inverse transform in both the encoder and decoder. The major benefits of adopting this are smoothing the block edges, reducing blocking effect and improving the objective quality of the decoded images.

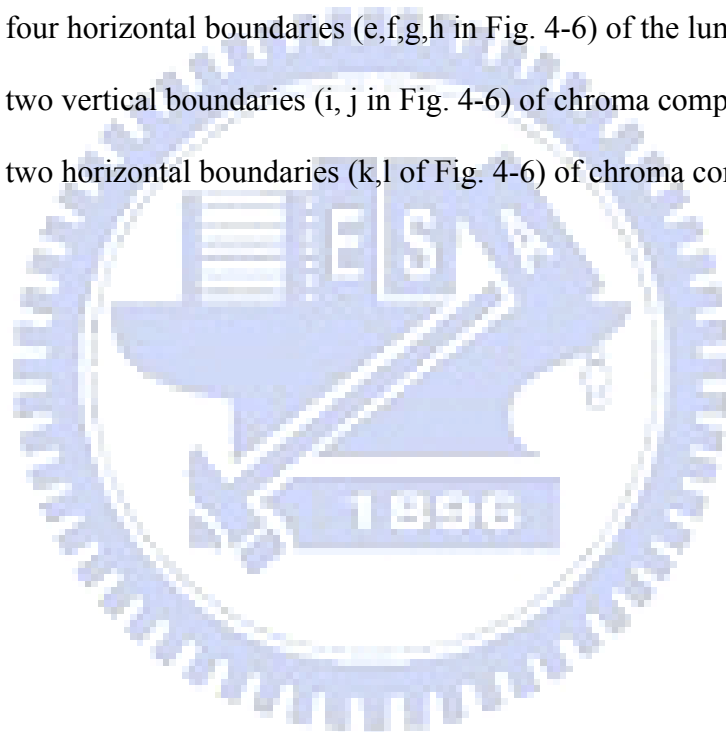
The deblocking filtering is applied to vertical or horizontal edges of 4x4 blocks in a macroblock. The deblocking steps are shown in the following:

Step 1: Filter four vertical boundaries (a,b,c,d in Fig. 4-6) of the luma component.

Step 2: Filter four horizontal boundaries (e,f,g,h in Fig. 4-6) of the luma component.

Step 3: Filter two vertical boundaries (i, j in Fig. 4-6) of chroma components.

Step 4: Filter two horizontal boundaries (k,l of Fig. 4-6) of chroma components.



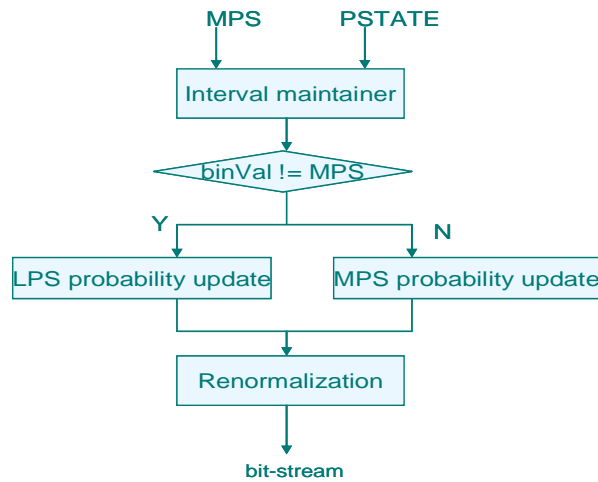


Fig. 4-5 Flow diagram of arithmetic coding.

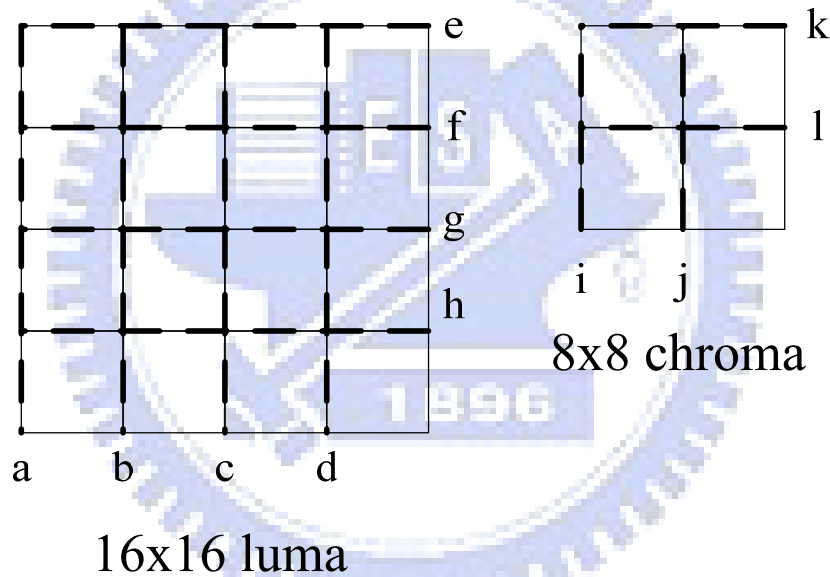


Fig. 4-6 Filtering boundary of a macroblock.

4.2 Design Challenges and Paper Survey

4.2.1 Design challenges

The design challenges of H.264/AVC basic encoder mainly come as follows:

- the high complexity of the encoding algorithms
- large memory requirement and computational loading of motion estimation
- The throughput of intra prediction and motion estimation is not enough for real time coding if adopting the algorithms of reference software

- The deblocking filter requires huge memory access and may become the system bottleneck
- The power and area consumption of the H.264 encoder is too huge.

However, the high resolution applications such as High Definition Television (HDTV), HD-DVD, and BD all adopt 1080p (1920x1080) H.264 high profile for higher compression efficiency and better video quality, which cannot be supported by previous works. Thus, the main stream 1080p high profile application presents a series of new design challenges:

- The new coding tools increase more complexity
- The 1080p high profile application needs at least 4X higher complexity than in the 720p baseline.
- The memory requirement and hardware cost of motion estimation module are double by bi-directional motion estimation.
- The resource conflict between the new coding tools and baseline tools
- CABAC will become the bottleneck due to its data dependency
- The high profile encoder must have the compatibility to support baseline and main profile encoding.

4.2.2 Paper survey

Because of high complexity of H.264 encoder, several VLSI implementations have been presented [3][4][5][69] but their performance is limited to baseline 720p (1280x720) [3][4] or SDTV (640x480) [5]. Although [69] can support 1080p resolution, the SoC design with embedded memory is very large and the power consumption is huge. Besides, the design targets of previous designs are focused on H.264 baseline profile which only provides the basic video quality and compression

efficiency. As for the commercial design, only [70][71] can support H.264/AVC high profile encoding. However, their performances are achieved by high operating frequency and huge power consumption.

4.3 System Overview

Fig. 4-7 presents the system overview [8]. The new high profile coding tools are included as the shaded parts. An important challenge is to add these new coding tools to the system but keep the similar throughput and minimum hardware overhead. As shown in Fig. 4-7, the system architecture of the proposed encoder has three macroblock pipelined stages. The first stage is the integer motion estimation (IME) stage which occupies the most computation and memory resource of the entire H.264 encoder. In the second stage, intra prediction and fractional motion estimation (FME) are placed in the same stage to share the current block buffer and pipelined buffer. Intra prediction uses the neighbor pixels to predict the current block and the FME refines the result of IME stage. The third stage is the entropy coding stage including Context-Adaptive Variable Length Coding (CAVLC) and Context-Adaptive Binary Arithmetic Coding (CABAC), which both provide high compression efficiency to generate the final bit-stream.

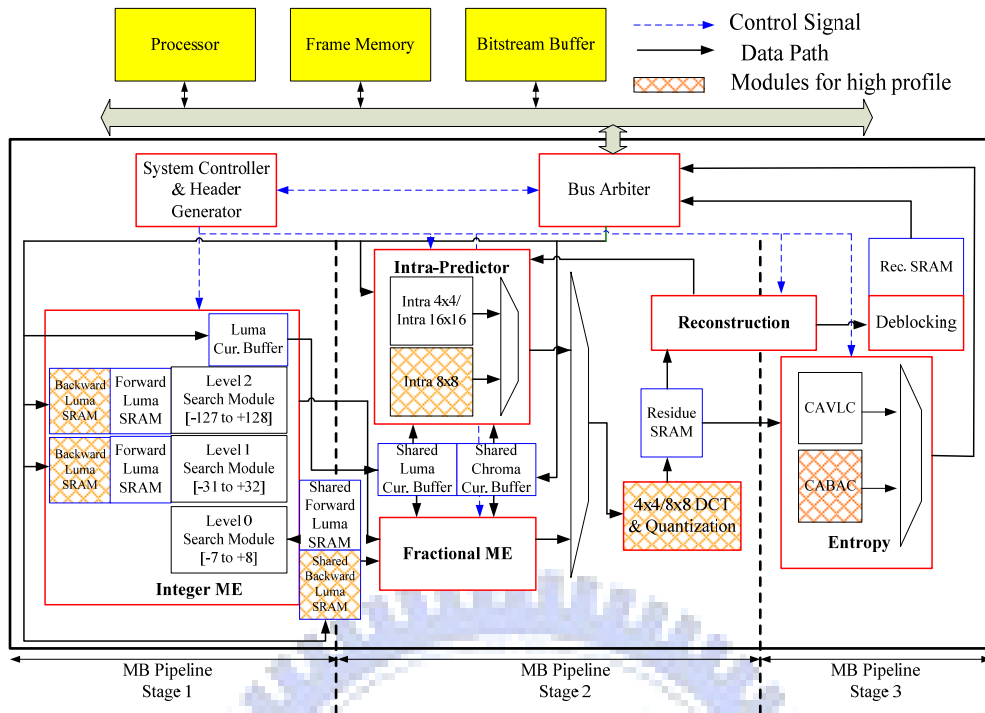


Fig. 4-7. System overview of H.264 high profile encoder.

| | | | | | | | | | | | | | | |
|--------------|-------------------------------|------|----|-----|----------------------------|------|-----------------|-----------------|-----|-----|------|------|------|-----|
| | 0 | 16 | 62 | 170 | 225 | 256 | 282 | 382 | 394 | 506 | 512 | 559 | 568 | 600 |
| IME (MB 2) | IME | | | | | | | | | | | LOAD | | |
| FME (MB 1) | Luma | | | | | IDLE | | Luma and Chroma | | | | LOAD | | |
| Intra (MB 1) | LOAD | Luma | | | | | Luma and Chroma | | | | IDLE | | LOAD | |
| Rec. (MB 0) | Rec. for deblocking and Inter | | | | Reconstruct intra boundary | | | IDLE | | | | | Rec. | |

Fig. 4-8. The scheduling of H.264 high profile encoder

4.4 Schedule of H.264 High Profile Encoder

Fig. 4-8 shows the scheduling of these three stages. There are three features in this scheduling. First, we load the reference data for IME in advanced because the IME requires huge amount of memory access. The second feature is that FME and Intra modules share residual SRAM and reference SRAM. Besides, the two modules adopt special scheduling for loading data from external memory so that their data requirement can be satisfied without confliction. Finally, the reconstruction process is through the second stage and the third stage.

4.5 System Level Hardware Sharing Techniques

4.5.1 Reconstruction sharing

The intra prediction and FME in the same stage could cause timing conflict in the reconstruction of inter and intra prediction and thus reconstruction hardware has to be duplicated. This conflict is that the intra predictor in the second stage needs the reconstructed boundary pixels of previous block immediately for intra mode decision and thus its reconstruction must finish in the second stage. But the reconstruction of inter prediction after the final mode decision must execute after all predictions are done. Besides, the reconstructed data of inter predicted blocks must go through the deblocking filter to remove the blocking effect and should finish in the third stage. To solve the reconstruction hazard, we place the reconstruction stage cross the second and third stages so that the intra and inter predictions can share the same hardware in different time slots. As shown in Fig. 4-9, the non-filtered reconstructed data is feedback to the intra predictor in the end of the second stage, but the deblocking filter processes the reconstructed blocks in the third stage. With the above *reconstruction sharing* technique, we can eliminate one extra reconstruction hardware unit and its power.

Therefore, during cycle 16 to 382 shown in Fig. 4-8, the reconstruction module reconstructs data for intra prediction procedure of MB 1 and filters the reconstructed data of MB 0 and MB 1. Moreover, after residual re-computation of a best mode is finished, the data is quantized and reconstruction begins immediately in the second stage. This work continues to the third stage which finishes the reconstruction of a MB and sends the data into the deblocking engine. By this flow, we can send the necessary quantized residuals to entropy coding module in time and remove bubble cycles in the third stage.

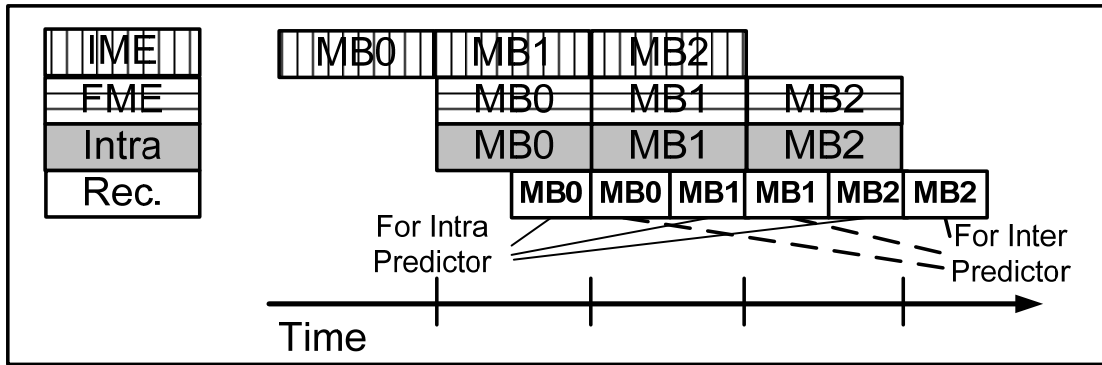


Fig. 4-9. The schedule of reconstruction module

4.5.2 Hardware-shared bi-directional motion estimation

Fig. 4-10 is the system architecture of bi-directional motion estimator for H.264 high profile. The width of external bus is 128 bits because the width of a MB is just 128 bits (i.e. 16 pixels)

To support bi-directional motion estimation, we double the memory for two reference frames but keep the same hardware cost with one directional motion estimator. It is because the throughput of PMRME (i.e. 512 cycles for bi-directional) is good enough to meet the throughput requirement of system (i.e. 600 cycles). As shown in Fig. 4-10, we have 10 memory blocks totally. Among these memories, three memory modules are used for level 0 forward search. The other two forward search modules for level 1 and level 2 both need one memory module. As for backward search, the memory allocation is the same with forward search. The three memories for level 0 are used as ping-pong buffers as presented in Sec. 2.6.2. With these level 0 buffer sharing technique, we can reuse the level 0 data between IME and FME and reduce the memory access time.

For level 1 and level2, the forward and backward search are interlaced. When forward or backward search is executing, the memory access of inverse direction search is also executing at the same time.

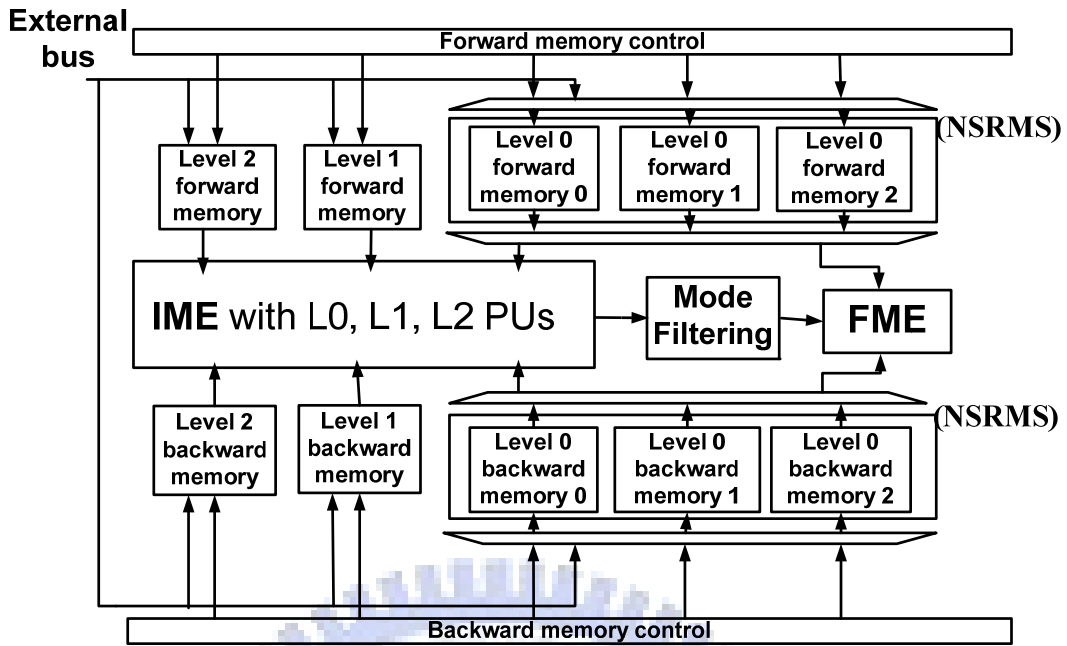


Fig. 4-10 System architecture of bi-directional motion estimator for H.264 high profile

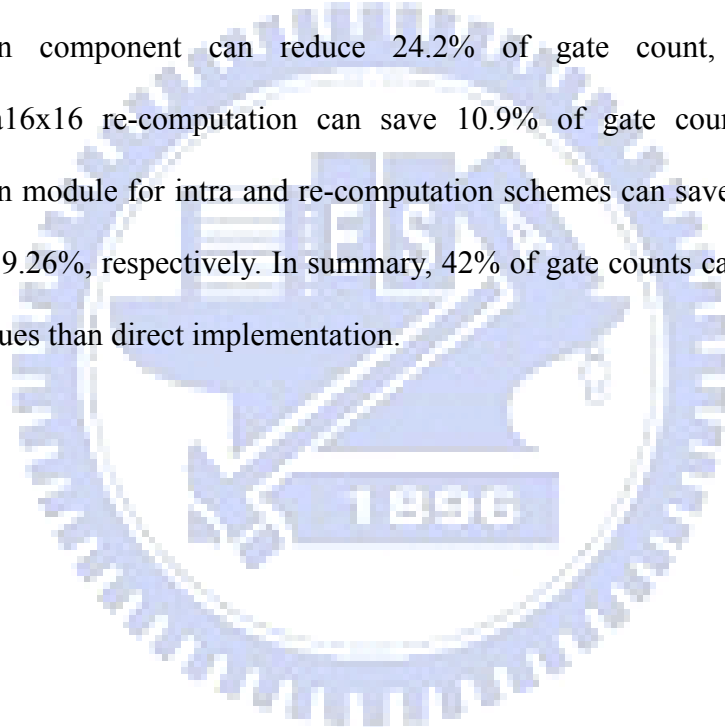
4.6 Full eight-pixel intra encoder

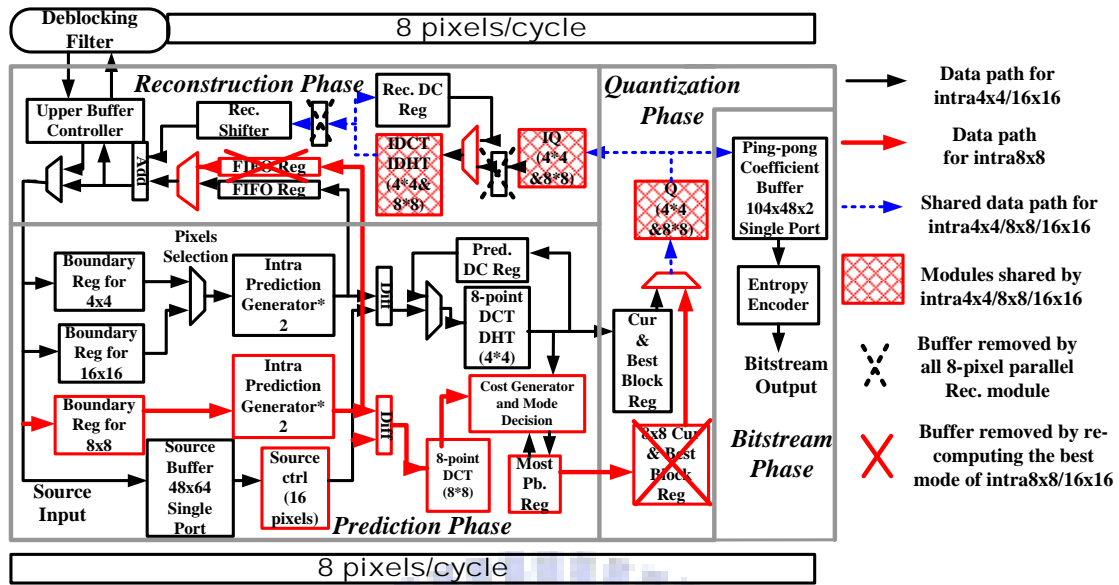
Fig. 4-11 (a) presents the overview of this design except ME and entropy coding. The new 1080p high profile coding tools including intra8x8 prediction and 8x8 integer DCT increase the complexity by 37.5% and the throughput by 2.5X compared to the baseline profile. Besides, the structure and data hazards will occur since the new high profile tools such as intra8x8 predictor need extra reconstruction and different reconstructed boundary data and thus will conflict with intra4x4 modules. For the data hazard, we adopt independent boundary buffer for intra8x8 prediction to eliminate it.

To solve the high throughput request and structure hazard, this design adopts eight-pixel parallelism. To further improve throughput, we parallel process intra8x8 and intra4x4/16x16 and use interlaced scheduling to minimize the stall cycles by data hazards. However, direct implementation will cause high cost due to eight-pixel parallelism. To reduce cost, we merge the reconstruction of intra4x4/16x16 and

intra8x8 into one and further share it with reconstruction of ME as stated above. To further decrease the cost, we adopt intra8x8/16x16 recomputation so that the best mode result and its prediction value of intra8x8/16x16 are not saved and recomputed if being chosen. With this, 2560 bits of memory can be reduced. Besides, using eight-pixel architecture in reconstruction and quantization phases can save the extra buffers between different pixel-parallelism phases. The details of these techniques will be discussed in the next section.

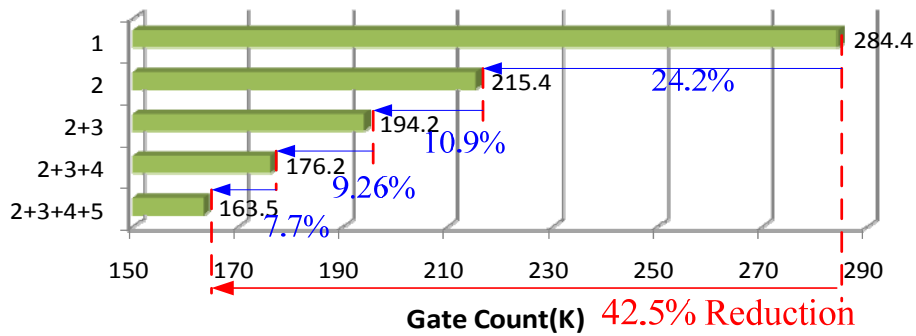
Fig. 4-11 (b) shows the performance of these proposed algorithms. The cross-stage reconstruction component can reduce 24.2% of gate count, and then the intra8x8/intra16x16 re-computation can save 10.9% of gate count. The merged reconstruction module for intra and re-computation schemes can save the gate counts by 7.7% and 9.26%, respectively. In summary, 42% of gate counts can be reduced by these techniques than direct implementation.





(a)

Reduction of Intra and Rec. Hardware Cost



- 1: Direct implementation
- 2: Rec. module to cross 2nd and 3rd stage
- 3: Intra8x8/16x16 recomputation
- 4: Eight-pixel parallelism Rec. module
- 5: Merged intra4x4/8x8/16x16 Rec. module

(b)

Fig. 4-11. (a)The architecture of intra encoder part. (b)The gate count reduction of intra encoder by proposed techniques.

4.6.1 Intra predictor

8x8 prediction generator is modified from 4x4 prediction generator to support more than six input ports. The major difference between 4x4 and 8x8 intra prediction generator is that new multiplexers are added in 8x8 prediction generator to support more inputs. Instead of predicting two rows of 4x4 blocks simultaneously shown in Fig. 3-10, this predictor computes a row in an 8x8 block directly. Its detailed architecture is shown in Fig. 4-12.



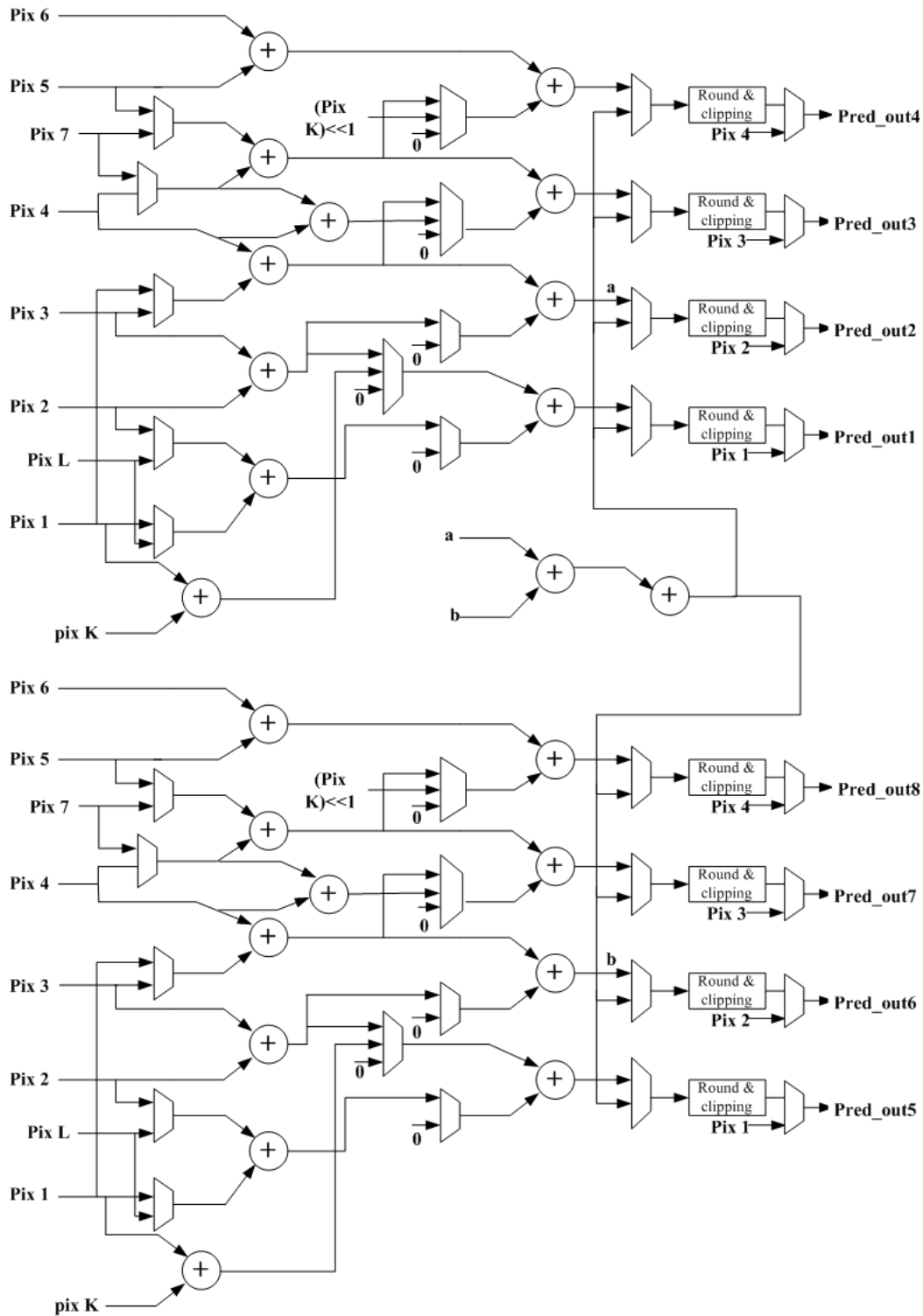


Fig. 4-12. Intra prediction generator used for intra luma 8x8 modes.

4.6.2 Interlaced schedule with intra 8x8 prediction

This scheduling of intra prediction generator as shown in Fig. 4-13 is based on previous work shown in Fig. 3-9. We used three scheduling techniques in this

scheduling, interlaced scheduling, parallel intra 8x8/4x4 computation, and re-computation for intra 8x8 boundary values.

- Interlaced scheduling :

The interlaced scheduling of luma 4x4 and 16x16 intra prediction modes is the same as Fig. 3-9. Because the reconstruction architecture also adopts 8-pixel parallelism, we re-arrange the schedule of some activities: deciding best luma mode, chroma mode prediction, and re-computing chroma best mode.

- Parallel 8x8/4x4 intra computation :

Because the additional path for 8x8 intra mode decision is added in the design, parallel intra 8x8/4x4 computation is used to computing 4x4 and 8x8 intra prediction modes in parallel without structure hazards in reconstruction phase to keep the same throughput as baseline designs.

- Re-computation for intra 8x8/16x16 boundary values :

For additional path for 8x8 intra prediction, we re-compute intra 8x8 boundary values to save hardware cost and increase its utilization. After the best intra mode decision of one 8x8 block, re-computation for intra 8x8 boundary values will process twice. The first re-computation is to re-compute boundary pixels as reference of its right and down blocks and the second is to produce the prediction reference value for reconstruction.

- Remove setup cycles in previous work [42]:

The purpose of setup cycles in Fig. 3-9 is to compute DC value used in luma 16x16 dc mode. However, we remove it and calculate the average value during computing enhanced SATD value of luma 16x16 vertical and horizontal modes in our design to save calculating cycles.

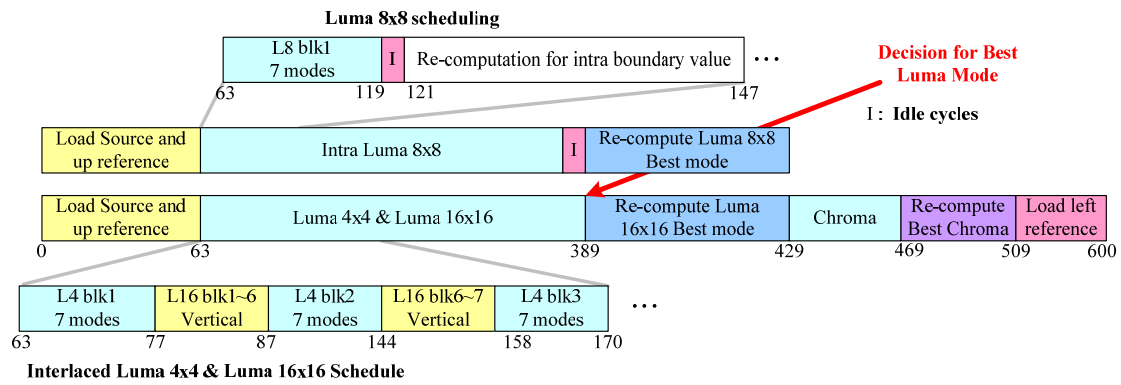


Fig. 4-13 Pipelined schedule of proposed intra prediction generator

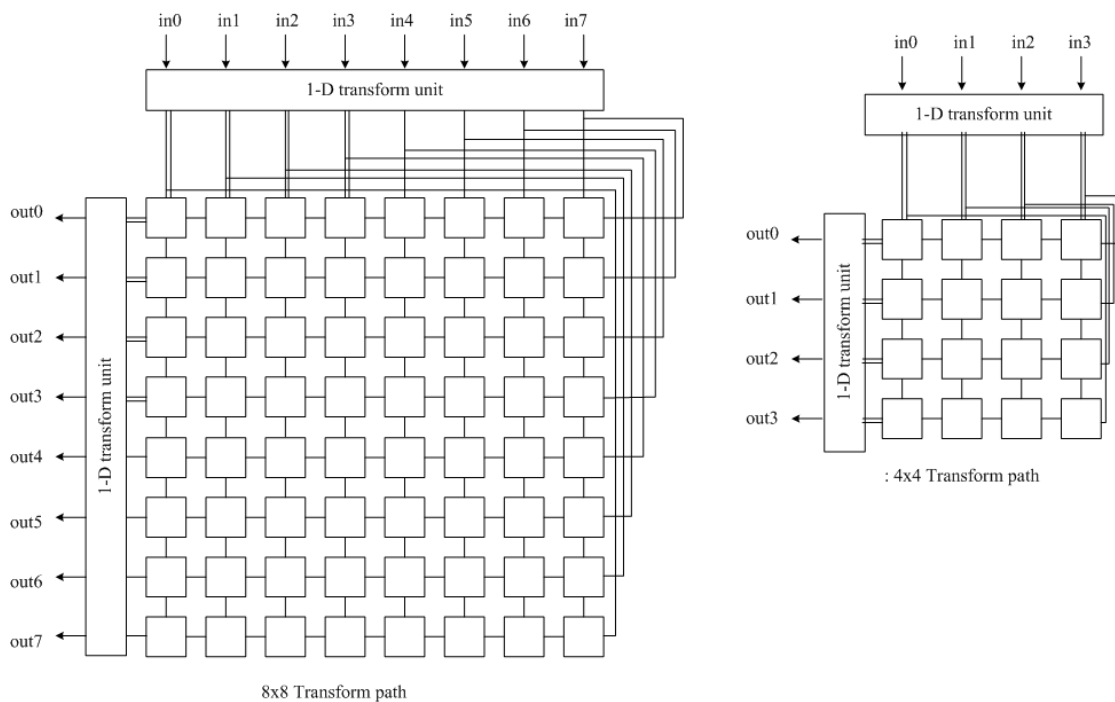


Fig. 4-14 Hardware architecture of transform unit

4.6.3 8x8 transform unit

This work also adopts the butterfly architecture in [61] as shown in Fig. 4-14, to execute integer 4x4 DCT, 8x8 DCT, 4x4 Discrete Hadamard Transform (DHT), and 2x2 DHT. Besides, two 4x4 block registers are used to store the DC coefficients for further DHT computation. The 2-D transform is also executed by two separate 1-D transforms with butterfly architecture [57]. Because 4x4 DCT and DHT have the same butterfly structures and do not operate at the same time in the encoder, they can share the hardware to reduce area.

4.6.4 Shared 8x8 inverse transform unit

Compare with other modules of our design, the reconstruction circuits has much lower hardware utilization. To resolve this problem, every unit of this reconstruction phase focuses on both reducing hardware cost and raising the utilization. For instance, the inverse transform unit can execute inverse 4x4 integer transform, inverse 8x8 integer transform, and inverse 4x4 DHT. This module adopts the design in [72] but it has structure hazards when an 8x8 DCT follows the 4x4 DCT transform immediately. We can avoid all structure hazards by carefully scheduling of intra prediction phase. Fig. 4-15 shows the block diagram of inverse 2-D transform unit.

Fig. 4-16 is the architecture of 1-D transform unit which we only add few multiplexers from general 8-pixel IDCT transform architecture to support two rows of four-pixel inverse integer transform operations in parallel. Fig. 4-17 and Fig. 4-18 show the data path for four-pixel and eight-pixel inverse integer transform respective. Similar to the forward transform hardware, the inverse integer transform and inverse Hadamard transform also share the hardware to reduce area and increase hardware utilization. Fig. 4-19 illustrates data path for inverse Hadamard transform. The steps of switching the hardware to support different types of inverse transform are shown below: Firstly, we decide inputs ports according to which function executed. And then, if the function is inverse Hadamard transform, we avoid all paths with shifters in the figure. If the function is inverse 4x4 DCT transform, we select the second datapath of every multiplexer as shown in Fig. 4-17. Otherwise, the first datapath of every multiplexer will be selected when executing functions is 8x8 DCT transform. To avoid bubble cycles for the DC value of special mode like intra 16x16 mode and intra chroma mode during reconstructing data, we compute inverse DHT transform after the DC value pass through DHT transform, quantization, and inverse quantization circuits immediately.

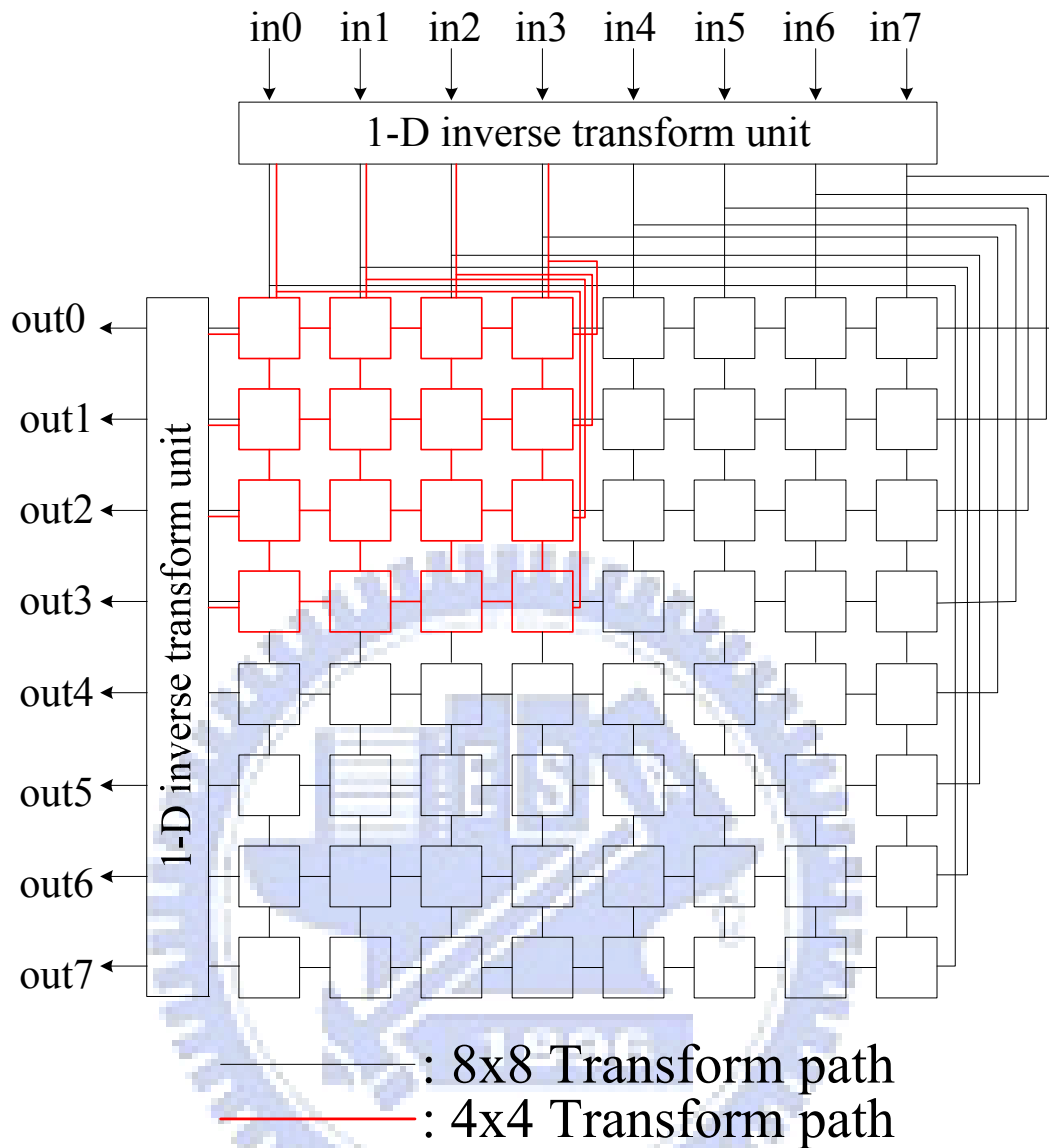


Fig. 4-15 Block diagram architecture of inverse transform unit

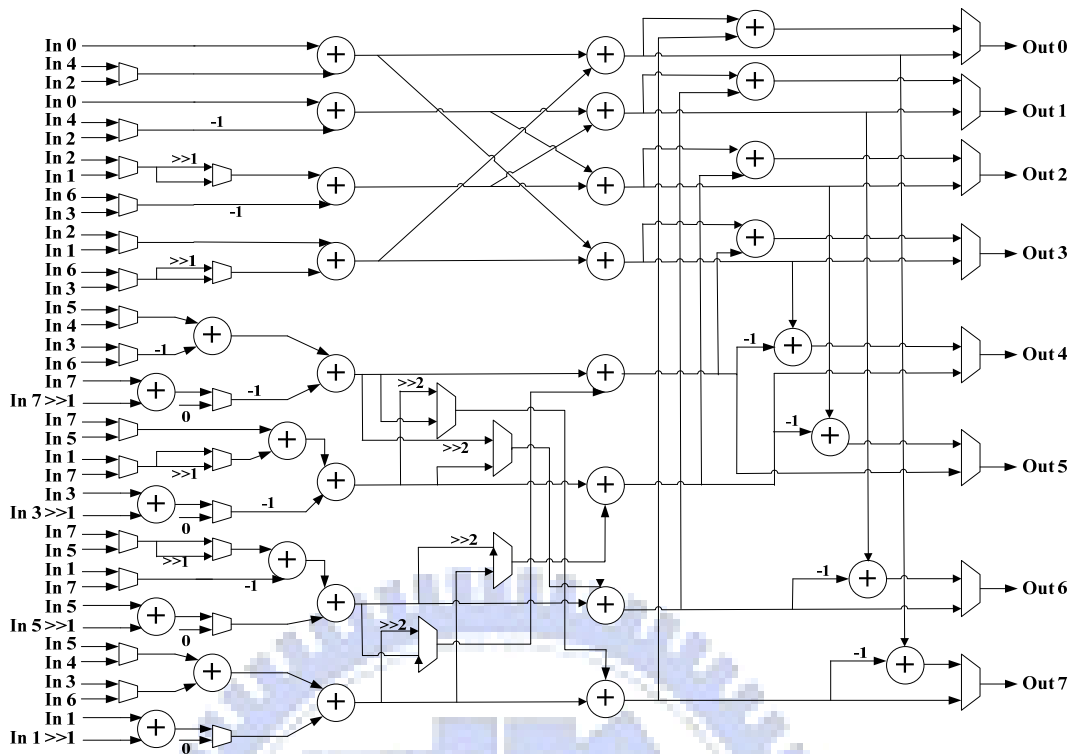


Fig. 4-16 The architecture of 1-D transform unit

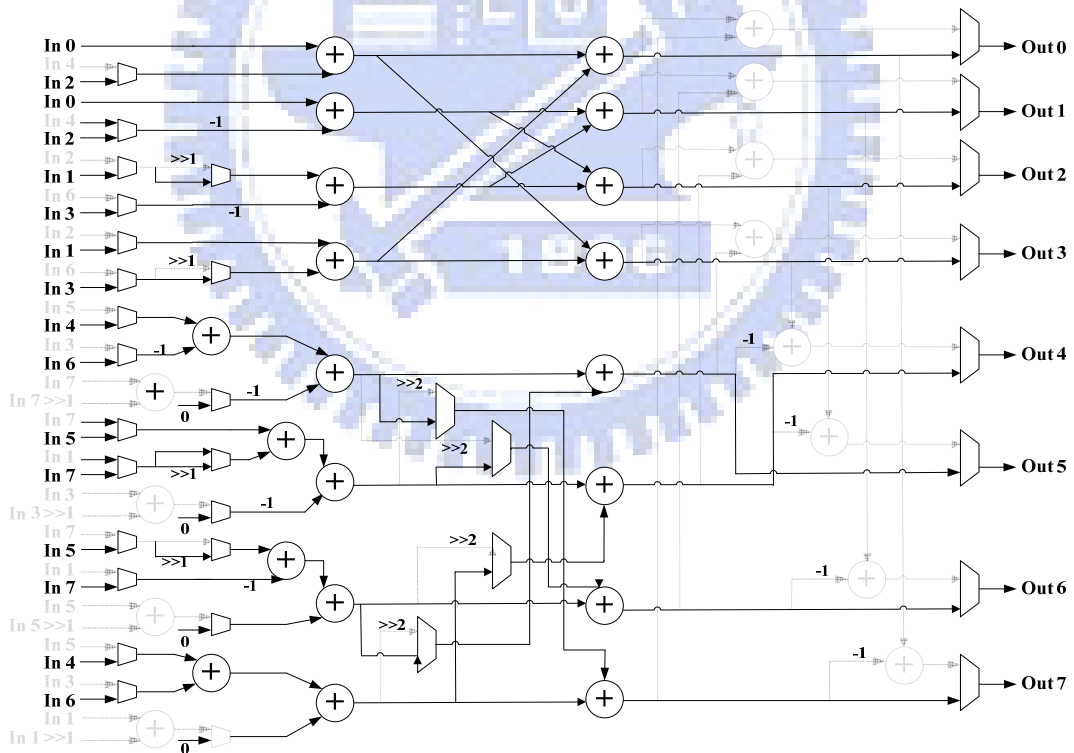


Fig. 4-17 The 4x4 IDCT transform datapath in inverse transform unit.

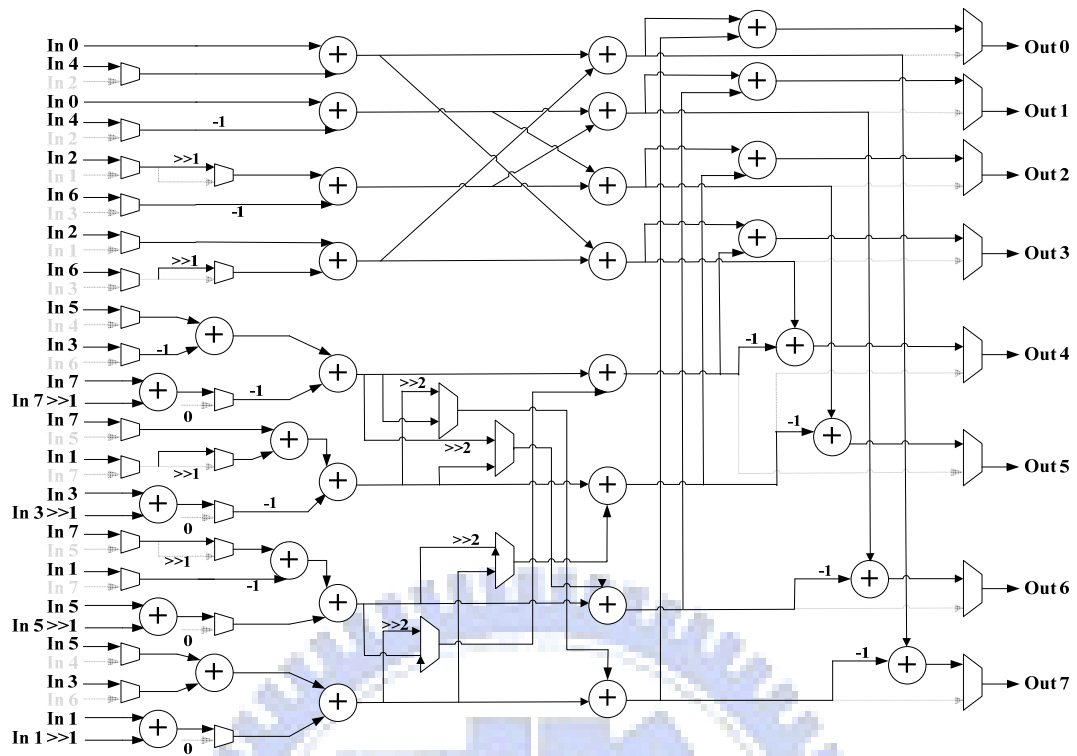


Fig. 4-18 The 8x8 IDCT transform datapath in inverse transform unit

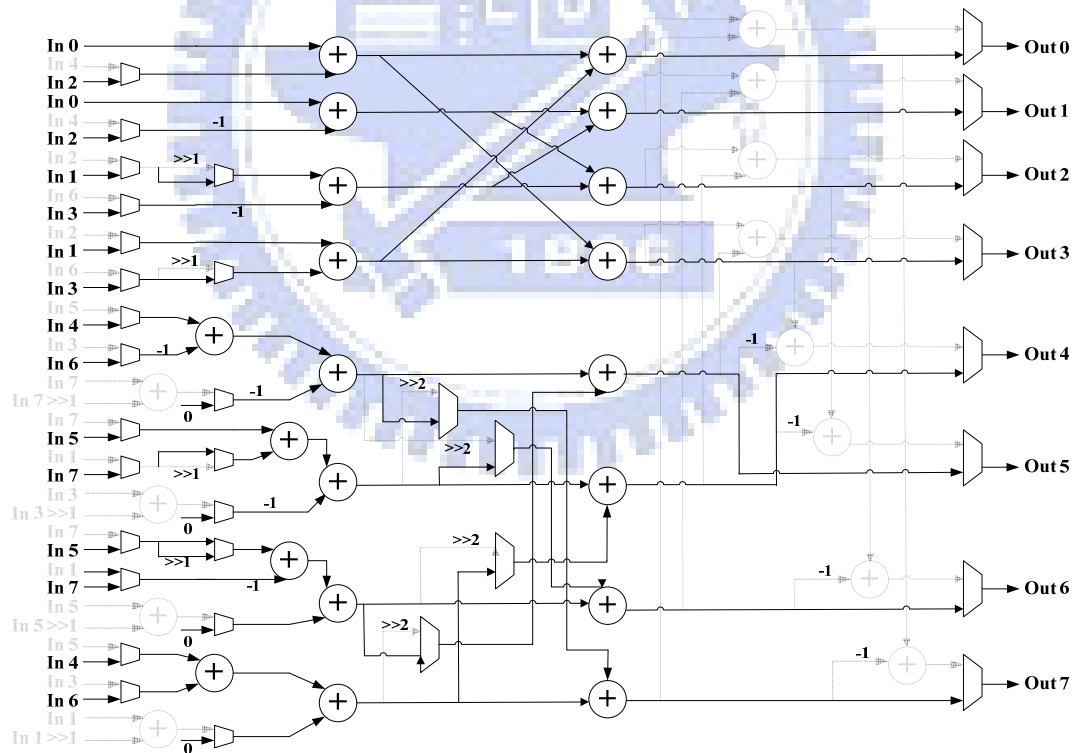


Fig. 4-19 The inverse Hadamard transform datapath in inverse transform unit

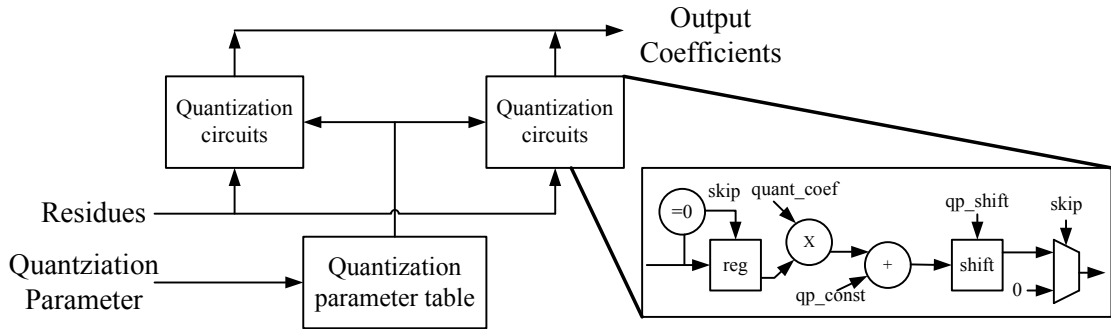


Fig. 4-20 Block algorithm of quantization circuits

TABLE 4-1 Quantization parameter table when QP equals twenty-eight: A for 4x4 block size, B for 8x8 block size

$$A = \begin{bmatrix} 8192 & 5243 & 5243 & 5243 \\ 5243 & 3355 & 5243 & 3355 \\ 8192 & 5243 & 5243 & 5243 \\ 5243 & 3355 & 5243 & 3355 \end{bmatrix}$$

$$B = \begin{bmatrix} 8192 & 7740 & 10486 & 7740 & 8192 & 7740 & 10486 & 7740 \\ 7740 & 7346 & 9777 & 7346 & 7740 & 7346 & 9777 & 7346 \\ 10486 & 9777 & 13159 & 9777 & 10486 & 9777 & 13159 & 9777 \\ 7740 & 7346 & 9777 & 7346 & 7740 & 7346 & 9777 & 7346 \\ 8192 & 7740 & 10486 & 7740 & 8192 & 7740 & 10486 & 7740 \\ 7740 & 7346 & 9777 & 7346 & 7740 & 7346 & 9777 & 7346 \\ 10486 & 9777 & 13159 & 9777 & 10486 & 9777 & 13159 & 9777 \\ 7740 & 7346 & 9777 & 7346 & 7740 & 7346 & 9777 & 7346 \end{bmatrix}$$

4.6.5 8-pixel quantization and inverse quantization unit

To process eight pixels in parallel, we use a pair of quantization circuits shown in Fig. 4-20 which are modified from the circuits presented in 3.4.4.2. However, the dual circuits can share the quantization parameter table shown in TABLE 4-1. For example, if we quantize the coefficients of one 4x4 block, one quantization circuits only needs odd rows of quantization parameter table and the other one only accesses even row parameters of the table. As for 8x8 block, one quantization circuits only needs left four parameters of one row and the other requests right four parameters. Finally, the similar architecture also adopts in de-quantization module due to the similar function behavior between quantization and de-quantization.

4.7 Bi-directional Inter Predictor Module

4.7.1 Techniques for inter prediction

Fig. 4-21 shows the motion estimation (ME) algorithm and its architecture. The basic techniques all come from Chapter 2. To achieve 1080p resolution with bi-directional prediction, we adopt parallel single step processing for ME. Thus, for integer motion estimation (IME), we use a parallelized subsampling algorithm, Parallel Multi-resolution ME (PMRME) as discussed in chapter 2. It searches three subsampling levels of different search ranges in parallel so that all searches are done with 256 cycles in single step. This provides higher throughput than [4] and [5] using two and four steps in IME respectively. With 256 cycles, the high throughput IME can process bi-directional predictions sequentially while still meets 1080p requirement. Thus, a single IME module cost is enough for both directional predictions. Besides, to support search range (SR) ± 128 and reduce the hardware cost within the limited quality loss, level 1 and 2 provide different subsampling ratio according to the search range for large motion vectors, and the search centers of the two levels are at (0,0) for further data reuse. Furthermore, to compensate quality loss of subsampling to meet the 1080p requirement, the level 0 without subsampling is centered at the motion vector predictor (MVP) to cover the most occurred motion vectors.

After IME, we use Mode Filtering (MF) to select only two best modes for FME refinement so that FME tests at most 18 motion vectors instead of 41 motion vectors in [3]. As for the fractional motion estimation part, we use the six pixels only instead of 17 pixels and 25 pixels in [3][5], and reduce at least 64.7% of complexity and 76% of processing units used in [4]. Besides, in order to improve the throughput, we finish the FME stage within a single iteration to double the throughput than previous works. To further reduce the bandwidth of FME, we use the Non-Subsampling Reference Memory Sharing (NSRMS), another cross stage technique, which uses three SRAM

banks to enable sharing of level 0 reference memory of IME and that of FME. FME searches only six candidates in a single step so that only six processing units are needed, which eliminates 76% of processing units used in [4].

Fig. 4-22 (d) shows the trade-off between the video quality loss and the search point reduction for motion estimation. In previous work [4], the fast motion estimation algorithms may result in 0.6dB PSNR loss, which may cause obvious image distortion. In our proposed ME algorithm, only 0.1dB quality loss is required, but the search points (search complexity) can reduce 98.7%. The fast IME (PMRME) can save the complexity by 91.7%. As for mode filtering and fractional motion estimation, they can reduce the complexity by 56% and 64%, respectively. It means the three algorithms all can reduce the complexity a lot.

In addition to the complexity, our proposed architecture also reduces the hardware cost including area, internal SRAM size and memory bandwidth. Compared with [3][25], the proposed gate count for motion estimation can decrease 30% and 62.5% gate count in IME and FME part as shown in Fig. 4-22 (b). These benefits come from the reduction of processing units to calculate the sum of absolute difference (SAD). For local memory reduction shown in Fig. 4-22 (c), the memory size can save 86% because PMRME uses the subsampling techniques and only the sampled pixels are necessary to be stored in local memory. Finally, the memory access can reduce 46% due to the subsampling PMRME technique as Fig. 4-22 (a) presents.

for block size larger than 8x8. Though Hadamard transform is greatly simplified, a 8x8 Hadamard transform unit still consumes about four times area than that of 4x4 one. Because the FME adopts six PUs architecture, six 8x8 SATD transform units will be required and thus cost a lot of area cost. Moreover, the area of interpolation unit will also increase. To solve this area problem, we propose to use 4x4 Hadamard for all SATD calculation disregarding of the block size [40].

TABLE 4-2 shows the comparison results of our algorithm with different SATD strategy for 1080p test sequences. The frame number is 100 and we set only the first frame to be I-frame because inserting I-frame periodically will reduce the impact of our algorithm. All data in TABLE 4-2 are acquired with the reference software [27]. As shown in the table, the results of using 4x4 and adaptive Hadamard transform are similar except for high QP situations. This is quite acceptable since the bit rate under that condition is quite low and any increase will be large in terms of that bit rate. Since the 4x4 transform unit only consumes 25% of area cost of 8x8 one, we calculate SATD by 4x4 Hadamard transform which doesn't influence video quality and saves about 75% of area cost in PU and 60% of area cost in the total FME module.

TABLE 4-2 The performance comparison with 4x4 and adaptive Hadamard transform

| 1080p size, 100 frame, only first frame is I-frame, RDO off, Search range = 16 | | | | | | |
|---|-----------------------|-------------------|-----------------------|-------------------|-----------------------|-------------------|
| | Blue Sky | | Pedestrian | | Riverbed | |
| QP | Δ PSNR (dB) | Δ bit rate | Δ PSNR (dB) | Δ bit rate | Δ PSNR (dB) | Δ bit rate |
| 16 | 0 | -0.48% | 0 | -0.11% | 0 | -0.06% |
| 22 | 0 | -0.67% | 0 | 0.61% | 0 | -0.16% |
| 28 | 0 | -0.4% | 0.01 | 1.38% | -0.01 | -0.02% |
| 34 | 0.01 | 0.83% | 0 | 2.13% | -0.02 | 0.67% |
| 40 | 0.08 | 3% | 0 | 1.81% | 0.01 | 0.96% |

4.8 Architecture of CABAC [73]

4.8.1 The proposed algorithm flow and architecture of CABAC

In [74], they rearrange the overall CABAC flow into four stages as shown in Fig. 4-23 (a). With help of software analysis, first, we find that *ctxIdx* calculation is depended on previous *binVal* not current ones. So, we can process binarization and context generation in parallel. Secondly, to read *pstateIdx* and MPS from context memory and update them at the same time, a dual-port memory is adopted here for increasing encoding speed. With this approach, the encoding iteration can be reduced from 5 to 3-4 cycles as shown in Fig. 4-23 (b) and architecture is easier to be pipelined into three stages as in Fig. 4-24.

In the first stage of CABAC as in Fig. 4-24, the binarization stage will output the bin-string to second stage and context memory will be updated and output *ctxIdx* for arithmetic coding (AC) at the same time. However, if the *ctxIdx* are the same for the successive processing, a stall signal should be added to avoid *pstateIdx* be read out before updating. This is the reason why proposed iteration is 4. The second stage is AC, which takes responsibility for calculating interval and output bit-stream. The last stage is FIFO, which collects data from AC. Because output word length of bit-stream is varying from 0 to 2, a FIFO is needed.

4.8.2 Architecture of binarization

In the binarization stage, although there are four schemes, it can be simply reduced into two types. In which, we classify U, TU and FL schemes into the table based type because they are easier to be realized by combinational logic. On the other hand, the table based UEGk will cost a lot due to the large table. To minimize the table cost, we use the arithmetic method to calculate it by adapting the table partition introduced in [75]. Thus, we use the parameter “base” to find the partition block and a carry save adder (CSA) to calculate it suffix. The proposed architecture is showed in Fig. 4-25.

4.8.3 Architecture of context modeling

The architecture of context modeling is showed as Fig. 4-26. As mention above, we adopted a dual-port memory for context memory. Besides, the probability updating is extracted from AC because it depends on values of pstateIdx and MPS. However, it does not depend on codIRange and codILow. Furthermore, transition table of MPS is reduced into simple one by its regular characteristic.

4.8.4 Architecture of AC

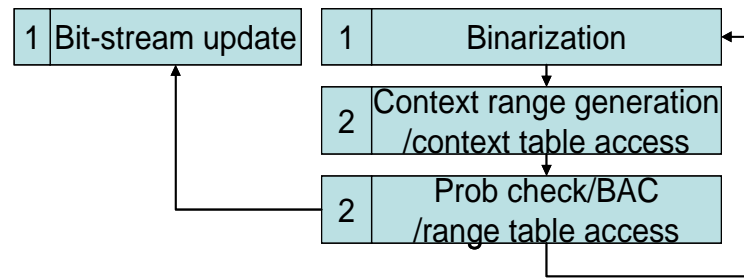
Fig. 4-27 shows the architecture of AC. There are two loops in AC [1]. One is controlled by codIRange and the other one is controlled by bitsOutStanding. To speed up the first loop, we skip the successive one by the Leading-Zero Detector (LZD) and Barrel-shifter to generate new interval. At the same time the output of LZD will sent to FSM to calculate the renormalization. The idea for the second loop speedup is by bit-parallelism as described below.

4.8.5 Interval maintainer in AC

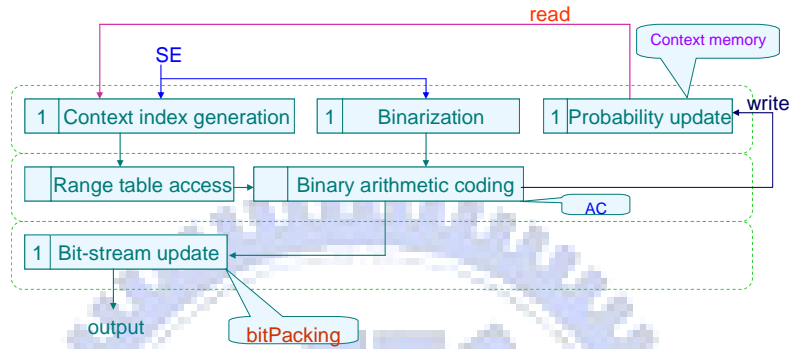
For the sake to maximize hardware sharing, we analyze `codILow`, and `codIRange` between three modes as shown in TABLE 4-3. Here, we can find no matter which mode is selected, `codILow` will involve a three input adder (when `binVal` equals to `MPS` or equals to 0). Thus we use a carry-save adder to compute new `codILow` to save hardware. This adder also helps calculating `codIRange` since `binVal` equals to zero and `binVal` non-equals to zero will not happen at the same time when the mode is on termination. After this calculation, the interval will be sent to renormalization. The proposed architecture is showed in Fig. 4-28.

4.8.6 Renormalization in AC

When renormalization is happened, [1] uses adder for updating `codILow`. However, with a detailed analysis, we can find that `codILow` is just trying to eliminate its MSB when `codIRange` is less than `0x100`. Thus, to minimum the hardware cost, we adopt a FSM instead of adder. After that, `BitsPacking` in the renormalization will receive the bit-stream from AC and pack them in byte. Within `BitsPacking`, `bitsOutstading` has to solve the carry over problem that requires a loop to output data. To break such multi-cycle operations, we use two masks to generate output data in parallel. With such bit-parallelism, we can process this loop in one cycle. Fig. 4-29 shows the architecture of the renormalization stage.



(a)



(b)

Fig. 4-23 (a) Original serial chedule of CABAC. (b) Modified parallel algorithm for CABAC

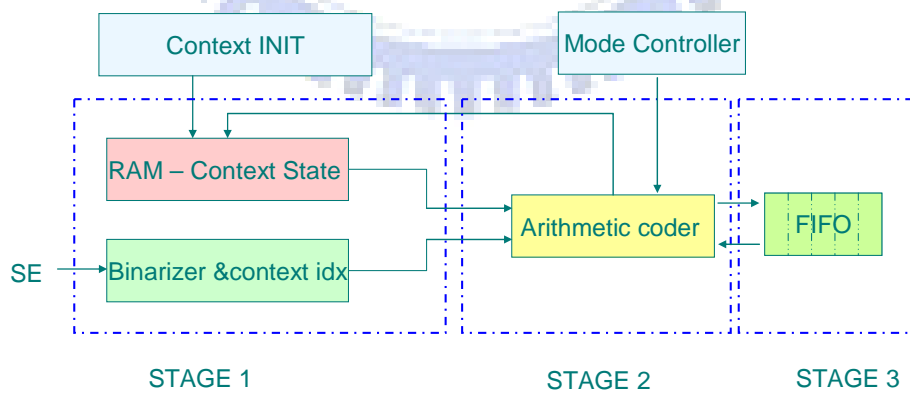


Fig. 4-24 Pipelined CABAC encoding flow

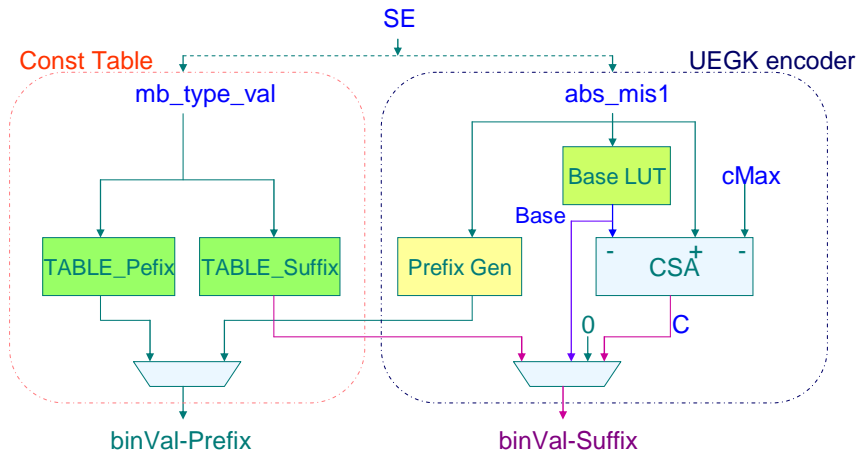


Fig. 4-25 Architecture of Binarization

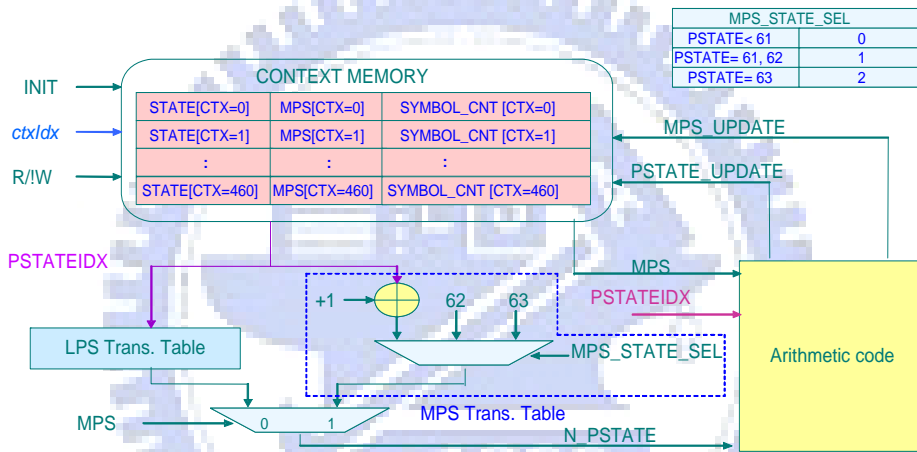


Fig. 4-26 Architecture of Context Modeling

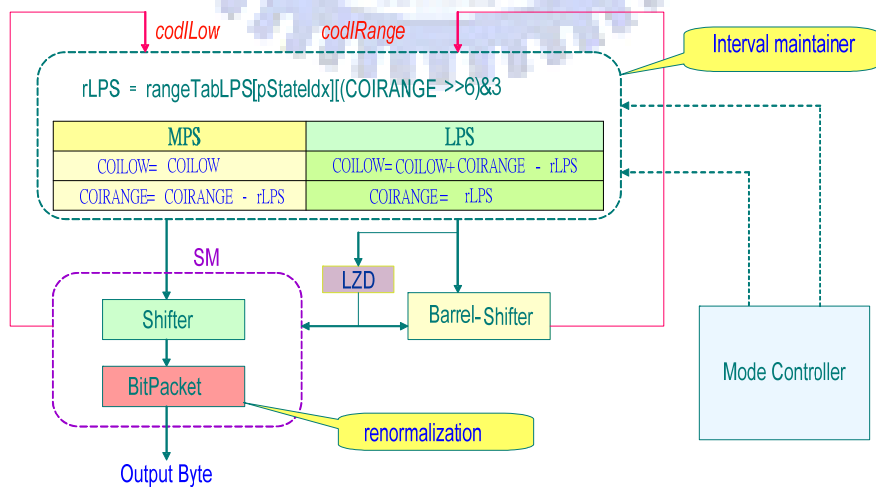


Fig. 4-27 Architecture of AC

TABLE 4-3 Optimized *codIRange* and *codILow*

| codIRange | | |
|-------------|------------------|---|
| DECISION | binVal = MPS(F) | $\text{codIRange} - \text{rangeTabLPS}[\text{pStateIdx}][\text{qCodIRangeIdx}]$ |
| | binVal != MPS(T) | $\text{rangeTabLPS}[\text{pStateIdx}][\text{qCodIRangeIdx}]$ |
| BYPASS | binVal = 0(F) | codIRange |
| | binVal != 0(T) | codIRange |
| TERMINATION | binVal = 0(F) | codIRange-2 |
| | binVal != 0(T) | 2 |

| codILow | | |
|-------------|------------------|--|
| DECISION | binVal = MPS(F) | codILow |
| | binVal != MPS(T) | $\text{codILow} + \text{codIRange} - \text{rangeTabLPS}$ |
| BYPASS | binVal = 0(F) | $\text{codILow} \ll 1$ |
| | binVal != 0(T) | $(\text{codILow} \ll 1) + \text{codIRange}$ |
| TERMINATION | binVal = 0(F) | codILow |
| | binVal != 0(T) | $\text{codILow} + \text{codIRange} - 2$ |

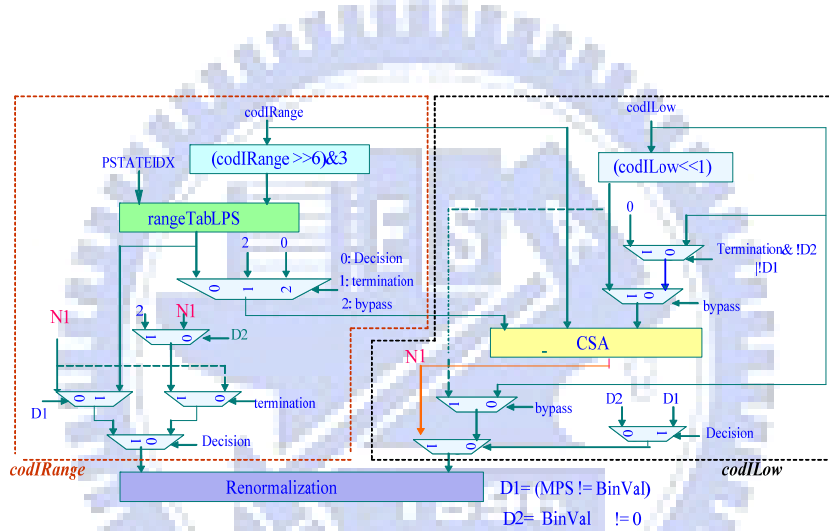


Fig. 4-28 Architecture of Interval Maintainer

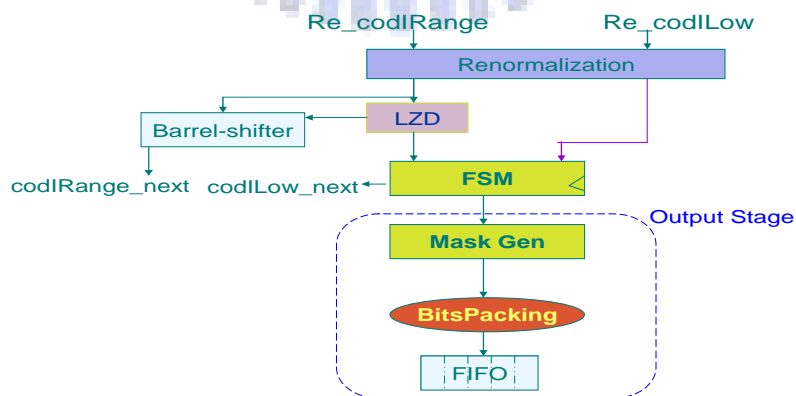


Fig. 4-29 Architecture of Renormalization

4.9 Deblocking Filter

Fig. 4-30 presents the proposed deblocking filter architecture design. The deblocking filter module adopts our previous design [76]. Fig. 4-31 shows the proposed filtering order of block boundary data. In order to speed up the throughput to meet the requirement and reduce buffer size, we change the filtering order shown in Fig. 4-6. The filtering starts from the most left-top block, in which we first filter the pixels on its two vertical edges (edge 0 and edge 1). Then, since all data is available for horizontal edge 2, we can filter this edge immediately. This horizontal- vertical interleaved approach is repeated for each 4x4 block in raster scan order, as the edge number shown in Fig. 4-31. This data flow also improves more data reusability than the original order shown in Fig. 4-6.

Besides, there are two SRAM modules in this filter. The SRAM, `rec_SRAM` in Fig. 4-30 stores the data only filtered in one dimension or not filtered yet does not complete filtering for further data reusing and removing data transferring cycles. The other SRAM module, `Ext_SRAM`, is used to store the filtered data for reconstruction. Finally, we output data to external memory until we complete filtering data of a row of blocks.

However, the filtering flow requires storing a row of blocks as the reference blocks for next row of blocks, which needs 30720 bits local memory for 1080p video. Therefore, it is necessary to send these temporary data to external memory.

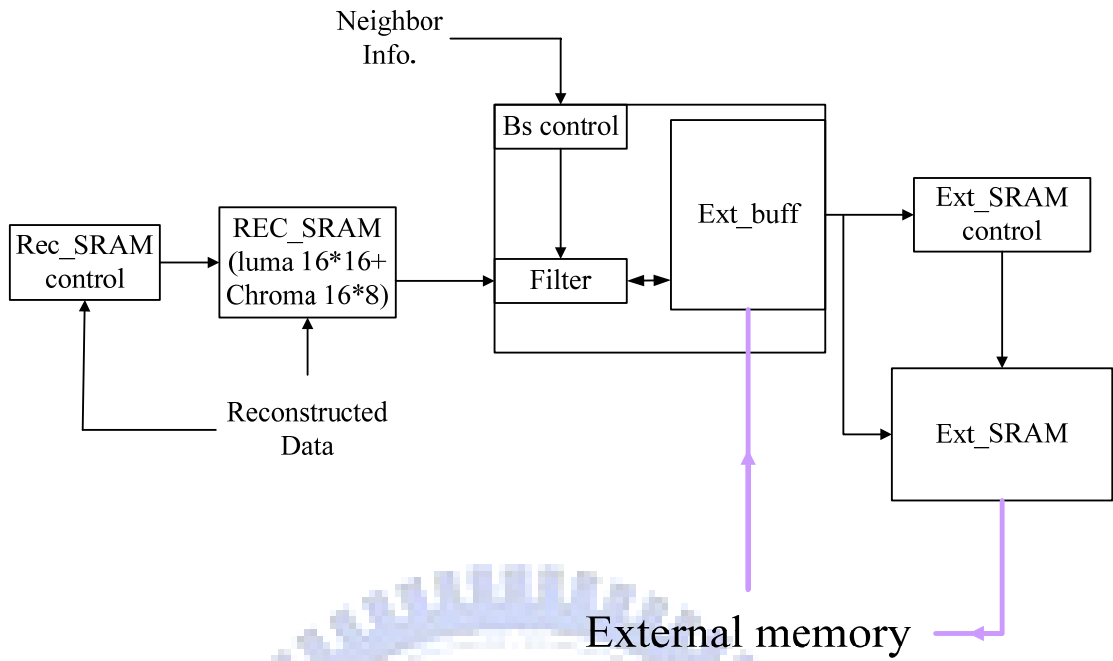


Fig. 4-30. Architecture design of deblocking filter.

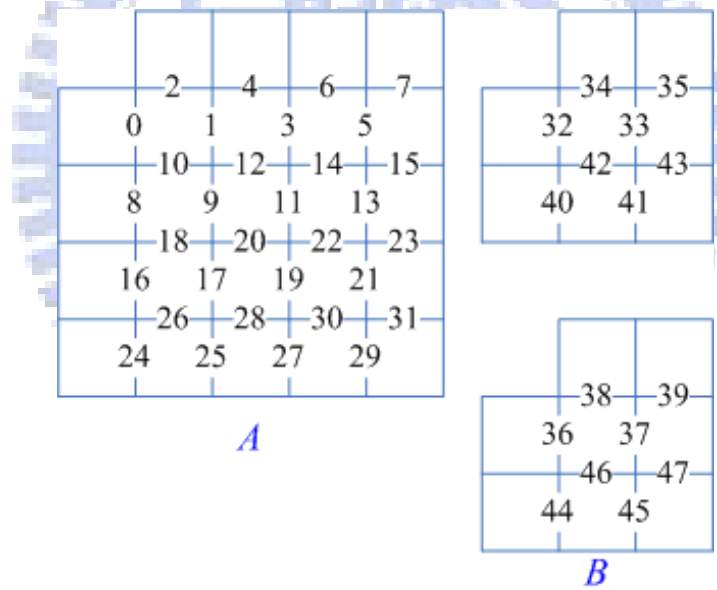


Fig. 4-31. Edge processing order for (A) luma edge, and (B) chroma edge

4.10 Implementation Result

4.10.1 Chip specification

A 0.13 μm 1080p high profile H.264 video encoder is presented with 3.1x3.1mm² core size. The equivalent gate count is 593K and the local memory size is 22 KB. The operating frequency for 720p and 1080p at 30fps is 62.5 and 145MHz respectively. The chip specification and features are summarized in TABLE 4-4, and the chip photo is shown in Fig. 4-32.

4.10.2 Power measurement result

TABLE 4-4 also shows the power consumption of our chip under different supporting resolution. For high profile, the power consumption is 116.61 and 242mW for 720p and 1080p resolution respectively. As for the baseline profile, the power consumption for 720p and 1080p resolution is 84.6 and 176.1mW. If you consider the low resolution video, the required power is 23.61mW for D1, 6.74mW for CIF, and 2.92mW for QCIF. We should note that the core voltage can be lower to 0.9V for CIF and QCIF resolution. Fig. 4-33 shows the power curves of our design and previous design. It is obviously that our design is the lowest power design when comparing with [4][5] no matter which resolution is adopted.

4.10.3 Comparisons with previous work

TABLE 4-5 shows the detailed comparison with other works. The presented design can support 1080p resolution, high profile, and up to ± 128 search range. Compared to the state-of-the-art design [4] targeted at 720p baseline in low power mode, the power consumption and operating frequency are 53.4% and 13% lower than [4] due to our enhanced throughput and complexity reduction. The PSNR loss is only 0.1dB that is better than [4] that has 0.6dB quality loss at the low power mode. For the area, our core size is only 54% of [4]. For small resolution video, the level 1 and level 2 of IME

are turned off when the video size is below CIF, and level 2 is off for D1 video size. Therefore, the power consumption is 6.74mW, only 42.6% of [5] and comparable to the lower power mode in [4] for baseline CIF video. In summary, the design outperforms other works to achieve 1080p processing with the less power consumption and area.



TABLE 4-4 Chip specification and features.

| | |
|-------------------------------|--|
| Name | H.264/AVC High Profile @Level 4 Encoder |
| Process | UMC 0.13 μ m 1P8M Standard CMOS 1.2V core, 3.3V I/O |
| Package | CQFP 208-pin |
| Gate Count | 593K |
| Internal Memory | 22KB |
| Chip Size | 3.76x3.76mm ² |
| Core Size | 3.17x3.17mm ² |
| Maximum Processing Throughput | 62.208 M pixels/sec @145MHz |
| Operating Frequency | 145MHz@1080p/30fps 62.5MHz@ 720p/30fps 28.5MHz@D1/30fps 7.2MHz@CIF/30fps 1.8MHz@QCIF/30fps |
| Core Power consumption | Baseline Profile: 176.1mW@1080p/30fps/1.2V 84.6mW @720p/30fps/1.2V 23.61mW@ D1/30fps/1.2V 6.74mW@CIF/30fps/0.9V 2.92mW@QCIF/30fps/0.9V High Profile: 242.01 mW@1080p/30fps/1.2V 116.61 mW@ 720p/30fps/1.2V |

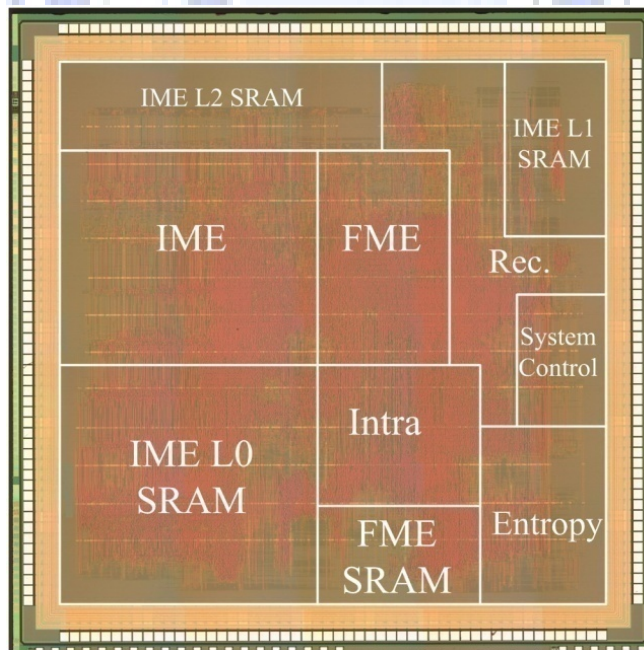


Fig. 4-32. Chip micrograph.

TABLE 4-5 Chip specification and comparison

| | Proposed [8] | [3] | [4] | [5] |
|-------------------------------------|---|------------------------------|---|---|
| CMOS Tech. | UMC 0.13 μ m | TSMC 0.18 μ m | TSMC 0.13 μ m | TSMC 0.18 μ m |
| Core Size | 3.17x3.17 mm ² | 7.68x4.13mm ² | 4.3x4.3mm ² | 3.47x3.7mm ² |
| Chip Size | 3.74x3.74mm ² | N/A | 4.9x4.9mm ² | N/A |
| Package | CQFP 208-pin | N/A | CQFP 160-pin | N/A |
| Profile | H.264 High @Level4 | H.264 Baseline | H.264 Baseline | H.264 Baseline |
| Maximum Support Resolution | 1920x1080 @30fps (1080p) | 1280x720 @30fps (720p) | 1280x720 @30fps (720p) | 640x480@30fps (SDTV) |
| Maximum Search Range | H: -128~+127 V: -128~+127 | H: -64~+63 V: -32~+31 | H: -32~+31 V: -32~+31 | H: -32~+31 V: -16~+15 |
| Quality Loss | 0.1dB | N/A | Up to 0.6dB | N/A |
| Gate Count | 593K | 922.8K | 470K | 458.2K |
| SRAM Size | 22KB /B-frame 14KB /P-frame | 34.72KB | 13.3KB | 16.95KB |
| Operating Frequency (for 30fps) | 145MHz@1080p 62.5MHz@ 720p 28.5MHz@D1 7.2MHz@CIF | 108MHz@720p 81MHz @D1 | 72/108MHz@720p 30~96MHz@D1 10~28MHz@ CIF | 54.5MHz@SDTV 13.5MHz@CIF |
| Power@ baseline profile (for 30fps) | 176.1mW @1080p/1.2V 84.6mW @ 720p/1.2V 23.61mW @ D1/1.08V 6.74mW @CIF/0.9V | 785mW@720p 581mW@D1 | 183mW @720p/1.2V 27mW@D1/0.9V 7mW@CIF/0.7V | 67.2mW @ SDTV/1.8V 15.9mW @ CIF/1.3V |
| Power@ high profile (for 30fps) | 242 mW @ 1080ps/1.2V 116.61 mW @ 720p/1.2V | N/A | N/A | N/A |

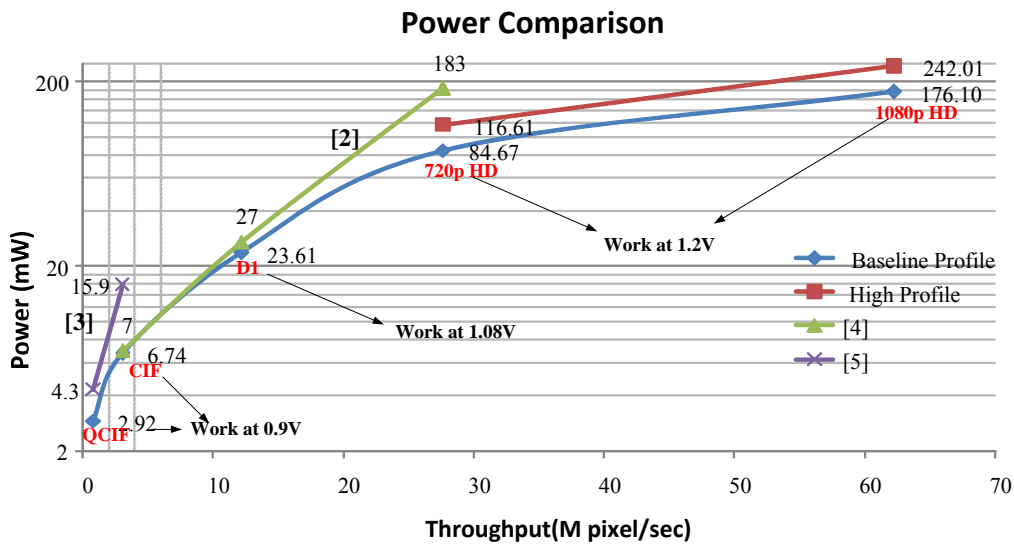


Fig. 4-33. The power of proposed design and previous works.

4.11 System Integration

After the chip implementation, the next step is to integrate our chip into a complete embedded system which consists of bus system, external memory, embedded processor, and memory controller as shown in Fig. 4-7. The proposed bus width of our design is 128 bits, which is four times than general AMBA bus protocol and external memory bandwidth. To satisfy the memory requirement, the clock rate of the external bus and memory should be four times than our chip so that the system with 32 bit bus width can provide the bandwidth equivalent to 128 bits. Our design supports 1080p resolution at 145MHz, so the external bus and memory should work at 580MHz which can be supported by AMBA 2.0 bus protocol and DDR3 SDRAM.

4.12 Summary

In this chapter, we propose a high throughput and low power H.264 encoder. The design not only supports the high profile specification and more coding tools, but also uses smaller area and lower power by parallelism enhanced throughput and cross

stage sharing pipeline. Besides, by the golden model and robust verification and testing strategies, our chip can work correctly and can be used as an IP for further reuse. In summary, this design achieves 46.7% and 54% reduction in area and power respectively.



Chapter 5

Conclusion

In this chapter, we conclude our work and point out some future works which can be the next research topics.

5.1 Conclusions

In this thesis, we propose not only a complete design of H.264 high profile encoder including innovative techniques in algorithm and architecture level but also the first H.264 high profile encoder chip in academia. This chip supports the high profile specification and achieves 46.7% and 54% reduction in area and power consumption by the system level and module level optimization techniques. Therefore, our design can be used in high definition video applications because it can encode 1080p video in real time and only requests acceptable hardware cost in mobile or HDTV devices.

Besides, this thesis also explores the cores of H.264 and provides highly data reused motion estimation engine. First, the proposed techniques such as mode filtering, PMRME, and SIFME make our engine support 1080p video with just 128.8MHz and 282.6K gates, and saves 60% of gate count, and 68.9% of SRAM buffers compared to the previous designs. Another important feature of presented motion estimator is that it can be easily scaled to other smaller size video with search range adjustment.

Secondly, for the applications which needs high resolution but has fewer resources like still camera, the intra-only encoder is the best choice. Therefore, we propose a H.264 intra frame encoder with just 94K gate and 0.72mm^2 core area at 140MHz

which can support digital video recorder applications with HD1080p 30 frames/sec resolution in real time and digital still camera application with 4096x2304 resolution at 6.78 frames/sec. The excellent specification is achieved by the novel optimization techniques such as fast prediction algorithm, variable pixel parallelism and other scheduling. With these techniques, this work can reduce 23.5% of gate count but only with 52% of working frequency.

In summary, the dissertation proposes a series of novelties which can be used in high definition applications without significant overhead and video quality degradation. We hope and believe that these improvements from this thesis can be adopted in commercial products in near future.

5.2 Future Works

5.2.1 H.264 Motion Estimator

- Skip Mode Detection

The skip mode in H.264 motion estimation can reduce a lot of bit-rate and complexity. However, the original flow about skip mode checking is not suitable for pipelined hardware design. Therefore, how to modify the skip mode detection algorithm and integrate it into the hardware is an important issue for low power H.264 encoder design.

- Direct Mode Implementation

To reduce the complexity cost of bi-directional and multiple reference motion estimation of H.264 encoder, direct mode is used to predict motion vectors in B frame encoding. The difficulty of direct mode implementation is the complex memory control and access schedule. For the ultra low power H.264 high profile encoder, the direct mode implementation must be a future work.

- Supporting Multiple Reference

Multiple reference technique is another key feature of H.264 motion estimator which can increase the precision of motion estimation. However, the computational loading and memory requirement of this technique are proportional to the number of reference frames. Therefore, our design does not include this technique yet. In the future work, our design will include the multiple reference technique.

5.2.2 H.264 Intra Encoder

- Multi-standard Encoder

As for the intra encoder, some modules are similar to previous image compression standards. For example, the entropy coding modules are similar to JPEG and JPEG2000 standard. Maybe we can integrate the H.264 intra encoder with JPEG2000 standard.

5.2.3 High Profile Encoder

- Scalable Video Coding Extension

The latest extension of H.264, the scalable video coding, is in the final draft stage. This extension is used to provide scalability of H.264 bit-stream so that the same bit-stream can be broadcasted to difference applications. And then, the different decoders retrieve the necessary data from the bit-stream. However, the foundation of the scalable video coding is the H.264 high profile codec. Therefore, we can use the H.264 high profile encoder as a basic structure and add the new coding tools of scalable video coding.

- System Level Integration

The H.264 high profile encoder chip must be as an accelerator in an integrated

system with general purpose processor and external memory system. Therefore, in addition to the chip performance, we should consider the integration of our design. Therefore, we are implementing a FPGA platform which implements our design into FPGA as an IP in an embedded system. By this platform, we can evaluate the system performance of a complete H.264 encoding system and modify our chip design to fit the requirement of whole system.

- H.264 High Profile Encoder with Rate-Distortion Optimization (RDO)

H.264 standard supports rate-distortion optimization technique to further improve the video quality and reduce the bit-rate. However, the iterative coding flow of rate-distortion optimization results in multiple times of complexity than that of encoder without optimization. Besides, the iterative loops also result in data hazard in pipelined system architecture. Therefore, most of H.264 encoder implementations don't include the rate-distortion optimization technique. However, the benefits of RDO is more obvious for high definition video application which pursuits the best video quality and optimized compression rate. So it is necessary to include the RDO function in future encoder design.

References

- [1] Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264/ ISO/ IEC14496-10 AVC), Mar. 2005.
- [2] T. Wiegand, and et al., "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no.7, pp. 560-575, July 2003.
- [3] Y. W. Huang, and et al., "A 1.3TOPS H.264/AVC Single-Chip Encoder for HDTV applications," in *Proc. Int. Solid State Circuits Conf.*, pp. 128–588, Feb. 2005.
- [4] H. C. Chang, and et al., "A 7mW to 183mW Dynamic Quality-Scalable H.264 Video Encoder Chip," in *Proc. Int. Solid State Circuits Conf.*, pp. 280–281, Feb. 2007.
- [5] T. C. Chen, and et al., "2.8 to 67.2mW Low-Power and Power-Aware H.264 Encoder for Mobile Applications," in *Proc. Symp. on VLSI Circuits*, pp. 222-223, June 2007.
- [6] Y. K. Lin, and et al., "A Hardware Efficient H.264/AVC Motion Estimation Design for High Definition Video," to be published, *IEEE Trans. on Circuits Syst. I, Reg. Papers*.
- [7] Y. K. Lin, and et al., "A 140MHz 94K GATES HD1080P 30 FRAMES/SEC INTRA-ONLY PROFILE H.264 ENCODER," to be published, *IEEE Trans. Circuits Syst. Video Technol.*
- [8] Y. K. Lin, and et al., "A 242mW, 10mm² 1080p H.264/AVC High Profile Encoder Chip," in *Proc. Int. Solid State Circuits Conf.*, pp. 314-315, Feb. 2008.

- [9] S. Y. Yap, and J. V. McCanny, "A VLSI architecture for variable block size video motion estimation," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 51, no. 7, pp. 384-389, July 2004.
- [10] C. M. Ou, C. F. Le, and W. J. Hwang, "An efficient VLSI architecture for H.264 variable block size motion estimation", *IEEE Trans. on Consum. Electron.*, vol. 51, no. 4, pp. 1291-1299, Nov. 2005.
- [11] C. Wei and M. Z. Gang, "A novel VLSI architecture for VBSME in MPEG-4 AVC/H.264", in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2 pp. 1794-1797, May 2005.
- [12] Z. Zheng, and et al., "High Data Reuse VLSI Architecture for H.264 Motion Estimation," in *Proc. Int. Conf. on Comm. Technol.*, pp. 1-4, Nov. 2006.
- [13] M. Kim, I. Hwang, and S. I. Chae, "A fast vlsi architecture for full search variable block size motion estimation in MPEG-4 AVC/H.264," in *Proc. Asia and South Pacific Design Automation Conf.*, vol. 1, pp. 631-634, Jan. 2005.
- [14] J. H. Lee and N. S. Lee, "Variable block size motion estimation algorithm and its hardware architecture for H.264/AVC," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol.3, pp. 741-744, May 2004.
- [15] T. C. Chen, and et al., "Fast Algorithm and Architecture Design of Low-Power Integer Motion Estimation for H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 5, pp. 568-577, May 2007.
- [16] C. L. Su, and et al., "A Low Complexity High Quality Interger Motion Estimation Architecture Design for H.264/AVC," in *Proc IEEE Asia Pacific Conf. on Circuits and Syst.*, pp. 398 - 401, Dec. 2006.
- [17] Y. L. Xi, and et al., "A fast block-matching algorithm based on adaptive search area and its VLSI architecture for H.264/AVC," *Signal Process.: Image Commun.*, vol. 21, no. 8, pp. 626-646, Sep., 2006,

- [18] L. Zhang, and W. Gao, "Reusable Architecture and Complexity-Controllable Algorithm for the Integer/Fractional Motion Estimation of H.264," *IEEE Trans. on Consum. Electron.*, vol. 53, no. 2, pp. 749-756, May 2007.
- [19] T. H. Tsai, and Y. N. Pan, "A Novel 3-D Predict Hexagon Search Algorithm for Fast Block Motion Estimation on H.264 Video Coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 12, pp.1542-1547, Dec. 2006.
- [20] C. H. Kuo, M. Shen, and C. C. J. Kuo, "Fast motion search with efficient inter-prediction mode decision for H.264," *J. Visual Comm. and Image Represent.*, vol. 17, no. 2, pp. 217-242, April, 2006.
- [21] N.A. Khan, S. Masud, A. A. Ahmad, "variable block size motion estimation algorithm for real-time H.264 video encoding," *Signal Process.: Image Commun.*," vol. 21, no. 4, pp. 306-315, April, 2006.
- [22] Y. K. Tu, and et al., "Fast variable-size block motion estimation for efficient H.264/AVC encoding," *Signal Process.: Image Commun.*, vol. 20, no. 7, pp. 595-623, August, 2005.
- [23] Z. Chen, and et al., "Fast integer-pel and fractional-pel motion estimation for H.264/AVC," *J. Visual Commun. and Image Represent.*, vol. 17, no. 2, pp. 264-290, April, 2006.
- [24] Z. Zhou, J. Xin, and M. T. Sun, "Fast motion estimation and Inter-mode decision for H.264/MPEG-4 AVC encoding," *J. Visual Commun. and Image Represent.*, vol. 17, no. 2, pp. 243-263, April, 2006.
- [25] T. C. Chen and et al., "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, June 2006.

- [26] C. Y. Chen, and et al., "Analysis and Architecture Design of Variable Block Size Motion Estimation for H.264/AVC", *IEEE Trans. on Circuits Syst. I, Reg. Papers*, vol. 53, no. 3, pp. 578-593, March 2006.
- [27] Joint Video Team Reference Software JM9.0, ITU-T.
- [28] T. C. Chen, Y. W. Huang, and L. G. Chen, "Fully utilized and reusable architecture for fractional motion estimation of H.264/AVC" in *Proc. IEEE Int. Conf. on Acoust., Speech, and Signal Process.*, vol. 4, pp. 9-12, May 2004.
- [29] L. Yang, and et al., "Prediction-based Directional Fractional Pixel Motion Estimation for H.264 Video Coding", in *Proc. IEEE Int. Conf. on Acoust., Speech, and Signal Process.*, vol. 2, pp.901-904, May, 2005.
- [30] J. F. Chang, and J. J. Leou, "A Quadratic Prediction Based Fractional-Pixel Motion Estimation Algorithm for H.264," in *Proc. IEEE Int. Symp. on Multimedia*, pp. 491-498, Dec. 2005.
- [31] H. Chao, and J. Lu, "A High Accurate Predictor Based Fractional Pixel Search for H.264," in *Proc. IEEE Int. Conf. on Image Process.*, pp.2365-2368, Sep. 2006.
- [32] L. Shen, and et al., "An adaptive and fast fractional pixel search algorithm in H.264," *Signal Process.*, vol. 87, no. 11, pp. 2629-2639, Nov., 2007.
- [33] Y. J. Wang, C. C. Cheng, and T. S. Chang, "A Fast Fractional Pel Motion Estimation Algorithm for H.264/AVC", in *Proc. IEEE Int. Symp. Circuits Syst.*, pp. 3974-3977, May 2006.
- [34] C. L. Su, and et al., "Low Complexity High Quality Fractional Motion Estimation Algorithm and Architecture Design for H.264/AVC," in *Proc IEEE Asia Pacific Conf. on Circuits and Syst.*, pp. 578-581, Dec. 2006.
- [35] C. C. Lin, Y. K. Lin, and T. S. Chang, "PMRME: A Parallel Multi-Resolution Motion Estimation Algorithm and Architecture for HDTV Sized H.264 Video

- Coding," *Proc. IEEE Int. Conf. on Acoust., Speech, and Signal Process.*, vol. 2 , pp. 385-388, April 2007.
- [36] C. Y. Kao, H. C. Kuo, and Y. L. Lin, "High Performance Fractional Motion Estimation and Mode Decision for H.264/AVC," in *Proc. IEEE Int. Conf. on Multimedia and Expo*, pp. 1241-1244, July 2006.
- [37] J. W. Suh, and J. Jeong, "Fast Sub-pixel Motion Estimation Techniques Having Lower Computation Complexity," *IEEE Trans. on Consum. Electron.*, vol. 50, pp. 968-973, Aug. 2004.
- [38] C. Yang, S. Goto, T. Ikenaga, "High performance VLSI architecture of fractional motion estimation in H.264 for HDTV," in *Proc. IEEE Int. Symp. Circuits Syst.*, pp.2605-2608, May 2006.
- [39] J. C. Tuan, T. S. Chang, and C. W. Jen, "On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 1, pp. 61–72, Jan. 2002.
- [40] T. Y. Kuo, Y. K. Lin, and T. S. Chang, "SIFME: A Single Iteration Fractional-Pel Motion Estimation Algorithm and Architecture for HDTV Sized H.264 Video Coding," *Proc. IEEE Int. Conf. on Acoust., Speech, and Signal Process.*, vol. 1, pp. 1185- 1188, April 2007.
- [41] T. Wedi, and et al., "Intra-only H.264/AVC profiles for professional applications," *JVT-U120*, Oct., 2006.
- [42] C. W. Ku, and et al., "A high-definition H.264/AVC intra-frame codec IP for digital video and still camera applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 8, pp. 917- 928, Aug. 2006.
- [43] Y. W. Huang, and et al., "Analysis, fast algorithm, and VLSI architecture design for H.264/AVC intra frame coder," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 3, pp. 378-401, Mar. 2005.

- [44] K. Suh, S. Park, and H. Cho, "An efficient hardware architecture of intra prediction and TQ/IQIT module for H.264 encoder," *ETRI Journal*, vol. 27, no. 5, pp. 511-524, Oct. 2005.
- [45] F. Fu, X. Lin, and L. Xu, "Fast intra prediction algorithm in H.264/AVC," in *Proc. IEEE Int. Conf. on Signal Process.*, vol. 2, pp. 1191-1194, Aug. 2004.
- [46] B. Meng, and et al., "Efficient intra-prediction algorithm in H.264," in *Proc. IEEE Int. Conf. on Image Process.*, vol. 3, pp. 837-840, Sep. 2003.
- [47] C. L. Yang, L. M. Po, and W. H. Lam, "A fast H.264 intra prediction algorithm using macroblock properties," in *Proc. IEEE Int. Conf. on Image Process.*, vol. 1, pp. 461-464, Oct. 2004.
- [48] F. Pan, and et.al , "Fast intra mode decision algorithm for H.264/AVC video coding," in *Proc. IEEE Int. Conf. on Image Process.*, vol. 2, pp. 781-784, Oct. 2004.
- [49] C. Kim, H. H. Shih, and C. C. J. Kuo, "Feature-based intra-prediction mode decision for H.264," in *Proc. IEEE Int. Conf. on Image Process.*, vol. 2, pp. 769-772, Oct. 2004.
- [50] J. F. Wang, and et al., "A novel fast algorithm for intra mode decision in H.264/AVC encoders," in *Proc. IEEE Int. Symp. Circuits Syst.*, pp. 3498-3501, May 2006.
- [51] C. C. Cheng, and T. S. Chang, "Fast three step intra prediction algorithm for 4x4 blocks in H.264," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol.2, pp. 1509-1512, May 2005.
- [52] D. W. Li, and et al., "A 61MHz 72K Gates 1280X720 30FPS H.264 Intra Encoder," *Proc. IEEE Int. Conf. on Acoust., Speech, and Signal Process.*, vol.2, pp. 801-804, April 2007.

- [53] S. C. Hsia, S. H. Wang, and Y. C. Chou, "A configurable IP for mode decision of H.264/AVC encoder," in *Proc. NASA/ESA Conf. on Adaptive Hardware and Systems*, pp. 146-152, Aug. 2007.
- [54] S. Li, and et al., "A VLSI architecture design of an edge based fast intra prediction mode decision algorithm for H.264/AVC," in *Proc. Great Lakes Symp. on VLSI*, pp.20-24, Mar. 2007.
- [55] C. H. Tsai, Y. W. Huang, and L.G. Chen, "Algorithm and architecture optimization for full-mode encoding of H.264/AVC intra prediction," in *Proc. Midwest Symp. Circuits Syst.*, vol.1, pp.47-50, Aug. 2005.
- [56] H.264/MPEG-4 AVC reference software JM8.6.
- [57] H. Malvar, and et al., "Low-complexity transform and quantization with 16-bit arithmetic for H.26L," in *Proc. IEEE Int. Conf. on Image Process.*, vol. 2, pp.489-492, Sep. 2002.
- [58] H. Y. Lin, and et al., "Combined 2-D transform and quantization architecture for H.264 video coders," in *Proc. IEEE Int. Symp. Circuits Syst.*, pp. 1802-1805, May. 2006.
- [59] K. H. Chen, and et al., "A high-performance low power direct 2-D transform coding IP design for MPEG-4 AVC/H.264 with a switching power suppression technique," in *Proc. Int. Symp. VLSI Design, Automation & Test*, pp. 291-294, Apr. 2005.
- [60] K. H. Chen, J. I. Guo, and J. S. Wang, "High-performance direct 2-D transform coding IP design for MPEG-4 AVC/H.264," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no.4, pp. 472-483, May. 2005.
- [61] T. C. Wang, and et al., "Parallel 4x4 2D transform and inverse transform architecture for MPEG-4 AVC/H.264," in *Proc. IEEE Int. Symp. Circuits Syst.*, , pp. 800-803, May. 2003.

- [62] M. C. Tsai and T. S. Chang, "High Performance Context Adaptive Variable Length Coding Encoder for MPEG-4 AVC/H.264 Video Coding," *Proc IEEE Asia Pacific Conf. on Circuits and Syst.*, 2006, pp. 586-589.
- [63] Y. K. Lai, C. C. Chou, and Y. C. Chung, "A simple and cost effective video encoder with memory-reducing CAVLC," in *Proc. IEEE Int. Symp. Circuits Syst.*, pp. 432–435, May 2005.
- [64] T. C. Chen, and et al., "Architecture Design of Context-Based Adaptive Variable-Length Coding for H.264/AVC", *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 53, no. 9, pp. 832-836, September 2006.
- [65] C. D. Chien, and et al., "A high performance CAVLC encoder design for MPEG-4 AVC/H.264 video coding applications," in *Proc. IEEE Int. Symp. Circuits Syst.*, pp. 3839-3842, May 2006.
- [66] D. Kim, and et al., "Implementation of High Performance CAVLC for H.264/AVC Video Codec", in *Proc. Int. Workshop on System-on-Chip for Real-Time Applications*, pp. 20-23, December 2006.
- [67] Joint Video Team of ITU-T and ISO/IEC: "Draft Text of H.264/AVC Fidelity Range Extensions Amendment", *JVT-L047*, Sep. 2004.
- [68] S. Gordon, D. Marpe, and T. Wiegand, "Simplified Use of 8x8 Transforms", *JVT-K028*, March 2004.
- [69] Z. Liu, and et al., "A 1.41W H.264/AVC Real-Time Encoder SOC for HDTV1080P," in *Proc. Symp. on VLSI Circuits*, June 2007, pp.12-13.
- [70] http://www.ambarella.com/news/press_releases/pr_09102007.htm
- [71] http://www.fujitsu.com/us/services/edevices/microelectronics/h264/index_p3.html
- [72] Y. K. Lin, Y. Z. Liao, and T. S. Chang, "An Area-Efficient Design for Integer

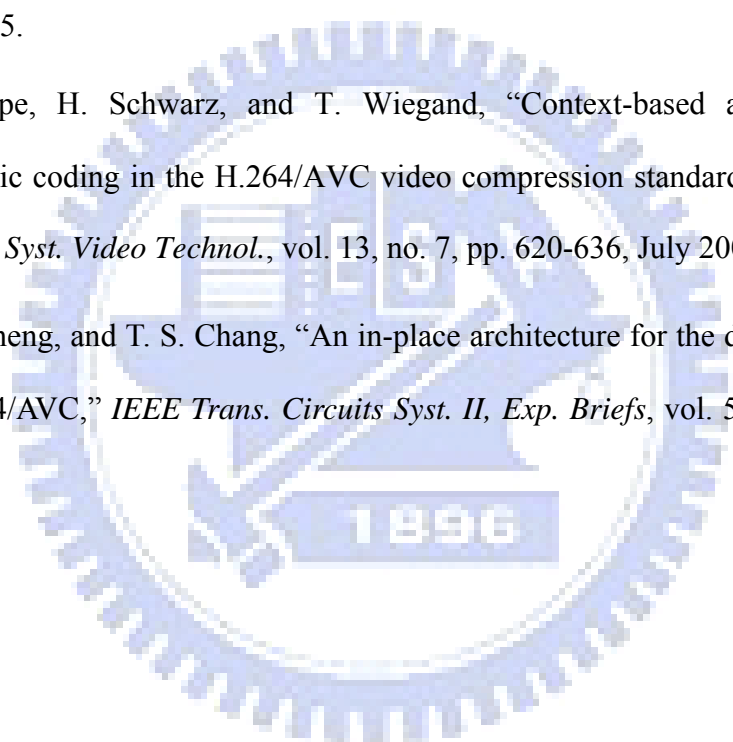
Transform in H.264/AVC,” in *Proc. VLSI Design/CAD Symp.*, pp. 517-520, Aug. 2006.

[73] J. L. Chen, Y. K. Lin, and T. S. Chang, "A Low Cost Context Adaptive Arithmetic Coder for H.264/MPEG-4 AVC Video Coding," *Proc. IEEE Int. Conf. on Acoust., Speech, and Signal Process.*, vol.2 pp. 105-108, April 2007.

[74] V. H. S. Ha, W. S. Shim, and J. W. Kim, “Real-time MPEG-4 AVC/H.264 CABAC entropy coder,” in *Proc. IEEE Conf. on Consum. Electron.*, pp.255-256, Jan. 2005.

[75] D. Marpe, H. Schwarz, and T. Wiegand, “Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 620-636, July 2003.

[76] C. C. Cheng, and T. S. Chang, “An in-place architecture for the deblocking filter in H.264/AVC,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 53, pp. 530-534, 2006.





作者簡歷

| | | | | |
|-----|-----|------|--------------------------------|---|
| 姓名 | 林佑昆 | 英文姓名 | Yu-Kun Lin |  |
| 性別 | 男 | 生日 | 1979/5/22 | |
| 出生地 | 高雄市 | 電子信箱 | yklin@twins.ee. nctu.edu.tw | |

學歷

| 學位 | 學校 | 系所 | 在學期間 |
|----|--------|----------|---------------|
| 學士 | 國立台灣大學 | 電機工程學系 | 1996/9~2000/7 |
| 碩士 | 國立台灣大學 | 電機工程學研究所 | 2000/9~2002/7 |
| 博士 | 國立交通大學 | 電子研究所 | 2002/9~2008/7 |

經歷

| | |
|--|---------------|
| 交通大學電子所 VLSI Signal Processing Lab. 系統管理員 | 2002/9~2005/6 |
| 晶片系統設計中心(CIC)約聘人員 | 2004/9~2005/6 |
| 矽智產設計課程助教 | 2004/9~2005/1 |
| 計算機結構課程助教 | 2005/2~2005/6 |
| 電子系統層級設計課程助教 | 2006/2~2006/6 |

專長

1. Digital IC design flow (from RTL to gdsII)
2. Silicon IP design methodology
3. Video codec (MPEG 1/2/4, H.264, Scalable Video Coding)
4. Digital image processing
5. VLSI signal processing
6. Computer architecture
7. Electric system level design
8. ARM-based system platform design
9. Biomedical electronics

榮譽

| | |
|--------|--|
| 學業成績優良 | <ol style="list-style-type: none"> 1. 大一上 書卷獎 全系第一名 2. 碩士班一年級 全組第一名 3. 碩士班二年級 A 類助學金(全組前 10%) 4. 91 年度斐陶斐榮譽會員 5. 交大博士班獎學金兩年(博士班前五名入學) |
| 研究成果獲獎 | <ol style="list-style-type: none"> 1. 92 學年度教育部 IP 設計競賽優勝 2. ASP-DAC 2005 Design Contest Winner 3. 45th ISSCC/DAC Student Design Contest Winner 4. 獲頂尖會議 DAC 2008 邀稿 5. 指導大學部專題生獲 Tensilica 2005 Xtensa 可配置式處理器設計大賽 第一名 6. 2008 年交大電子所博士論文獎 優等獎 |
| 社團領導 | <ol style="list-style-type: none"> 1. 87 年度台大優良服務性社團負責人 - 華南慈善獎學金 |

作者著作目錄

International Journal Papers:

[1] **Yu-Kun Lin**, Chia-Chun Lin, Tzu-Yun Kuo, and Tian-Sheuan Chang, "A Hardware Efficient H.264/AVC Motion Estimation Design for High Definition Video," to be published, *IEEE Transaction on Circuit and System I: Regular Papers*.

[2] **Yu-Kun Lin**, Chun-Wei Ku, De-Wei Li, and Tian-Sheuan Chang, "A 140MHz 94K GATES HD1080P 30 FRAMES/SEC INTRA-ONLY PROFILE H.264 ENCODER," to be published, *IEEE Transaction on Circuit and System for Video Technology*.

International Conference Papers:

[1] **Yu-Kun Lin**, De-Wei Li, Chia-Chun Lin, Tzu-Yun Kuo, Sian-Jin Wu, Wei-Cheng Tai, Wei-Cheng Chang, and Tian-Sheuan Chang, "A 242mW, 10mm² 1080p H.264/AVC High Profile Encoder Chip," to be published, *Design Automation Conference (DAC)*, June 2008. (Invited Paper)

[2] **Yu-Kun Lin**, De-Wei Li, Chia-Chun Lin, Tzu-Yun Kuo, Sian-Jin Wu, Wei-Cheng Tai, Wei-Cheng Chang, and Tian-Sheuan Chang, "A 242mW, 10mm² 1080p H.264/AVC High Profile Encoder Chip," *International Solid-State Circuits Conference (ISSCC)*, pp. 314-315, Feb. 2008.

[3] Chia-Chun Lin, **Yu-Kun Lin**, and Tian-Sheuan Chang, "Hardware Efficient Skip Mode Detection for H.264/AVC," *International Conference on Consumer Electronics (ICCE)*, Jan., 2008.

[4] Tzu-Yun Kuo, **Yu-Kun Lin**, and Tian-Sheuan Chang, "SIFME: A Single Iteration

Fractional-Pel Motion Estimation Algorithm and Architecture for HDTV Sized H.264 Video Coding," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 1, pp. 1185- 1188, April 2007,.

[5] Jian-Long Chen, **Yu-Kun Lin**, and Tian-Sheuan Chang, "A Low Cost Context Adaptive Arithmetic Coder for H.264/MPEG-4 AVC Video Coding," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol.2 pp. 105-108, April 2007.

[6] Chia-Chun Lin, **Yu-Kun Lin**, and Tian-Sheuan Chang, "PMRME: A Parallel Multi-Resolution Motion Estimation Algorithm and Architecture for HDTV Sized H.264 Video Coding," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 2, pp. 385-388, April 2007.

[7] De-Wei Li, Chun-Wei Ku, Chao-Chung Cheng, **Yu-Kun Lin**, and Tian-Sheuan Chang, "A 61MHz 72K Gates 1280X720 30FPS H.264 Intra Encoder," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol.2, pp. 801-804, April 2007.

[8] Chia-Chun Lin, **Yu-Kun Lin**, and Tian-Sheuan Chang, "A Fast Algorithm and Its Architecture for Motion Estimation in MPEG-4 AVC/H.264 Video Coding," *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*,pp. 1248-1251, Dec. 2006.

[9] Tzu-Yun Kuo, **Yu-Kun Lin**, and Tian-Sheuan Chang, "A Memory Bandwidth Optimized Interpolator for Motion Compensation in the H.264 Video Decoding," *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pp. 1244-1247, Dec. 2006.

[10] Jia-Bin Huang, **Yu-Kun Lin**, and Tian-Sheuan Chang, "A Display Order Oriented Scalable Video Decoder," *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pp. 1976-1979, Dec. 2006.

[11] **Yu-Kun Lin** and Tian-Sheuan Chang, "Analysis and architectures of MCTF for scalable video coding," *Picture Coding Symposium (PCS)*, May 2006.

[12] **Yu-Kun Lin** and Tian-Sheuan Chang, "Fast block type decision algorithm for intra prediction in H.264 FReXt," in *IEEE International Conference on Image Processing (ICIP)*, pp. 585-588, Sep. 2005.

[13] Hao-Yun Chin, Chao-Chung Cheng, **Yu-Kun Lin**, and Tian-Sheuan Chang, "A Bandwidth Efficient Subsampling-based Block Matching Architecture for Motion Estimation," *Asia and South Pacific Design Automation Conference (ASP-DAC)*, vol. 2, pp. D/7 - D/8, Jan. 2005.

Domestic Conference Papers:

[1] **Yu-Kun Lin**, Ying-Ze Liao, and Tian-Sheuan Chang, "AN AREA-EFFICIENT DESIGN FOR INTEGER TRANSFORM IN H.264/AVC FReXt," *VLSI Design/CAD Symposium*, pp. 517-520, Aug. 2006.

[2] Chia-Chun Lin, **Yu-Kun Lin**, and Tian-Sheuan Chang, "Hardware Oriented Algorithms for Motion Estimation in MPEG-4 AVC/H.264 Video Coding," *VLSI Design/CAD Symposium*, , pp. 505-508, Aug. 2006.