# 國立交通大學

## 電子工程系

## 碩士論文

# 建構在 ARM 平台的 H.264/MPEG-4 AVC 解碼器以及去方塊濾波加速器

# ARM-based Platform Design for H.264/MPEG-4 AVC Decoder and Accelerator for Deblocking Filter

研 究 生：張世騫

指導教授：蔣迪豪　教授

中 華 民 國 九 十 四 年 七 月

# 建構在 ARM 平台的 H.264/MPEG-4 AVC 解碼器以及去方塊濾波加速器

# ARM-based Platform Design for H.264/MPEG-4 AVC Decoder and Accelerator for Deblocking Filter


研 究 生：張世騫　　Student：Shih-Chien Chang

指導教授：蔣迪豪　　Advisor：Tihao Chiang


國 立 交 通 大 學

電 子 工 程 系

碩 士 論 文

A Thesis

Submitted to Department of Electronics Engineering

College of Electrical Engineering and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Electronics Engineering

July 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年七月

# 建構在 ARM 平台的 H.264/MPEG-4 AVC 解碼器以及去方塊濾波加速器

學生：張世騫　　指導教授：蔣迪豪

國立交通大學電子學系（研究所）碩士班

## 摘　　要

本論文使用最佳化的平台式設計方法去建構一個 H264/MPEG-4 AVC 解碼器。考量其高效能、低成本及廣泛的應用範圍，我們使用 ARM 微處理器作為 CPU 核心。同時，我們使用高效能控制匯排流架構（AMBA）去提升系統傳輸效能和彈性。為了提升解碼器的速度，我們同時對軟體及硬體做最佳化。同時，我們提出一個巨方塊平行處理的架構(macroblock-level pipelining) 使得軟體和硬體能夠同步處理而提升效能。在我們的硬體設計裡，我們實現三個加速器去滿足三個計算需求最強的模組: 去方塊濾波器(deblocking filter), 動作補償(motion compensation) 和轉置 DCT 運算(inverse transform)。其中,在去方塊濾波器的設計裡，我們提出適應性傳輸方法(adaptive transfer scheme)和匯排流同步傳輸的架構(bus-interleaved architecture)。考量去方塊濾波器需要大量的傳輸頻寬,我們將傳輸分成 8 種模式以適應性的方法減少傳輸資料量而使頻寬有效被利用。另外，為了減少去方塊濾波處理的時間，我們使用匯排流同步傳輸資料的架構使資料傳輸和濾波處理能平行處理。和前人去方塊濾波硬體設計比較，我們最高有 7 倍的效能改善。就整體解碼效能改善而言，我們的設計比起 H.264 參考軟體 JM6.0 有 9 到 16 倍的效能提升。整體而言，我們的平台系統設計可以快速的整合到單晶片系統(system-on-chip)的設計中。而且，我們提出的硬體架構設計也可滿足低成本與即時播放(real-time)的應用。

# ARM-based Platform Design for H.264/MPEG-4 AVC Decoder and Accelerator for Deblocking Filter

Student：Shih-Chien Chang Advisor：Tihao Chiang

Department of Electronics Engineering
National Chiao Tung University

## ABSTRACT

In this thesis, we present a baseline H264/MPEG-4 AVC decoder based on an optimized platform-based design methodology. In our platform, we employ the ARM microprocessor as the CPU core due to its high performance, low cost, and wide application. Besides, the Advanced Microcontroller Bus Architecture (AMBA) is integrated into our system as the on-chip bus due to its high performance and flexibility. To improve our system, we jointly optimize the software and hardware in the decoder. Also, we propose a macroblock-level pipelining architecture to achieve the synchronization of the software and the dedicated hardware co-processors. In our hardware design, three dedicated accelerators of deblocking filter, motion compensation and inverse transform, which are the most computationally intensive modules, are realized. Specifically, in the architecture design of deblocking filter, we proposed an adaptive transfer scheme and a platform-based bus-interleaved architecture. As considering the high bandwidth usage of bus for deblocking filter, we classify the filtering mode into 8 types and use an adaptive transmission scheme to avoid redundant data transfers so as to efficiently use the bus bandwidth. Moreover, to reduce the processing latency, we use a bus-interleaved architecture for conducting data transfer and filtering operation in parallel. As compared to the state-of-the-art designs of deblocking filter, our scheme offers up to 7x performance improvement. To compare the overall decoding performance, our experiments show that the throughput of H.264 reference software of JM6.0 decoder can be improved by 9 to 16 times. Finally, our proposed platform system can be easily applied in the system-on-chip design. Also, our proposed hardware architectures are suitable for low-cost and real-time applications.

# 誌　　謝

首先我要感謝我的指導教授蔣迪豪老師，老師豐富的學養與多元化的實務經驗，使我在專業領域上獲益良多。老師平日的關心和不吝惜指導的熱忱，我更是點滴在心頭。此外，老師對於「年輕不要留白」的人生觀大大地啟發了我對人生的看法，無形中也教育我積極為事與樂觀進取的工作態度與思考方法。也期許未來我能培養出和老師一樣擁有強大精神力卻不失風雅幽默的人格特質。

感謝一路帶領我學習的兩位學長:彭文孝學長與王士豪學長。儘管我時常無法達到要求，學長們總是不厭其煩指導我正確學習方法並且絕對不會草率妥協。對於學長們每一字一句的指導，我真的發自內心的感謝因為學長們其實都還忙碌於自己的工作卻不吝惜花費時間精力在我身上。除了指導我實事求是的做事方法，學長們「好一定還能更好」的學習精神，更教育我無止境學習的研究態度並擴展了我的眼光到全世界。

感謝我的實驗室好夥伴林承毅、楊思浩和全體實驗室的成員。除了謝謝你們平日的幫忙外，也祝福你們不管是現在或未來畢業後都能實現自己的理想並享受自己創造的人生旅程。

# Content

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Overview of Thesis

### 1.1.1 H.264/MPEG-4 AVC Standard

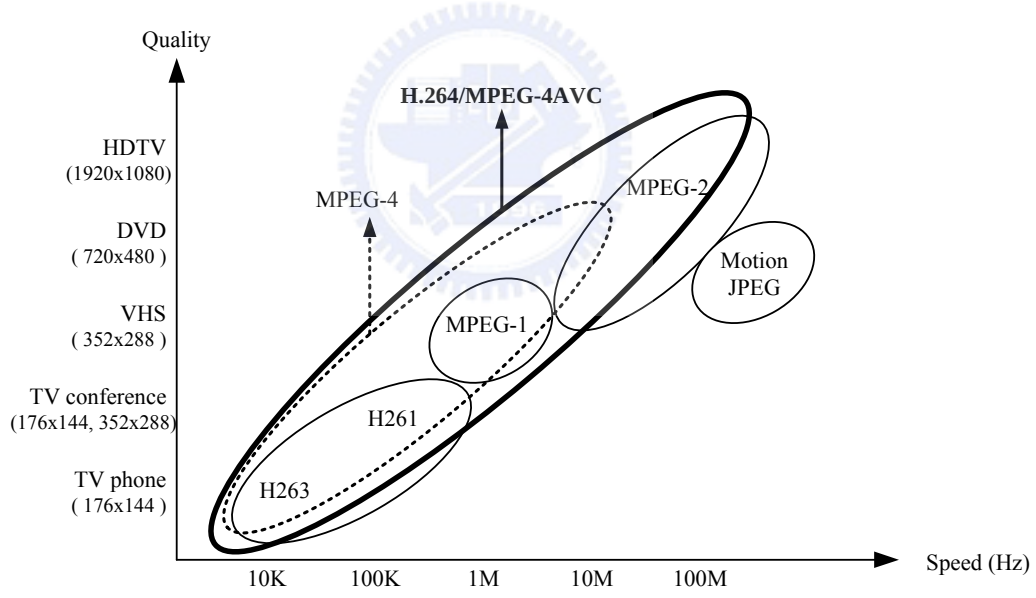



Fig. 1. Application range for well-known video standards.

For low bit-rate and real-time communication, the International Telecommunication Union (ITU-T) developed a series of standards like H.261 [1] and H.263 [2]. For high quality video application under limited bandwidth, the Motion Picture Expert Group (MPEG) of International Standard Organization (ISO) announced the standards of MPEG-1 [3], MPEG-2 [4], and MPEG-4 [5]. Fig. 1 shows the application range for

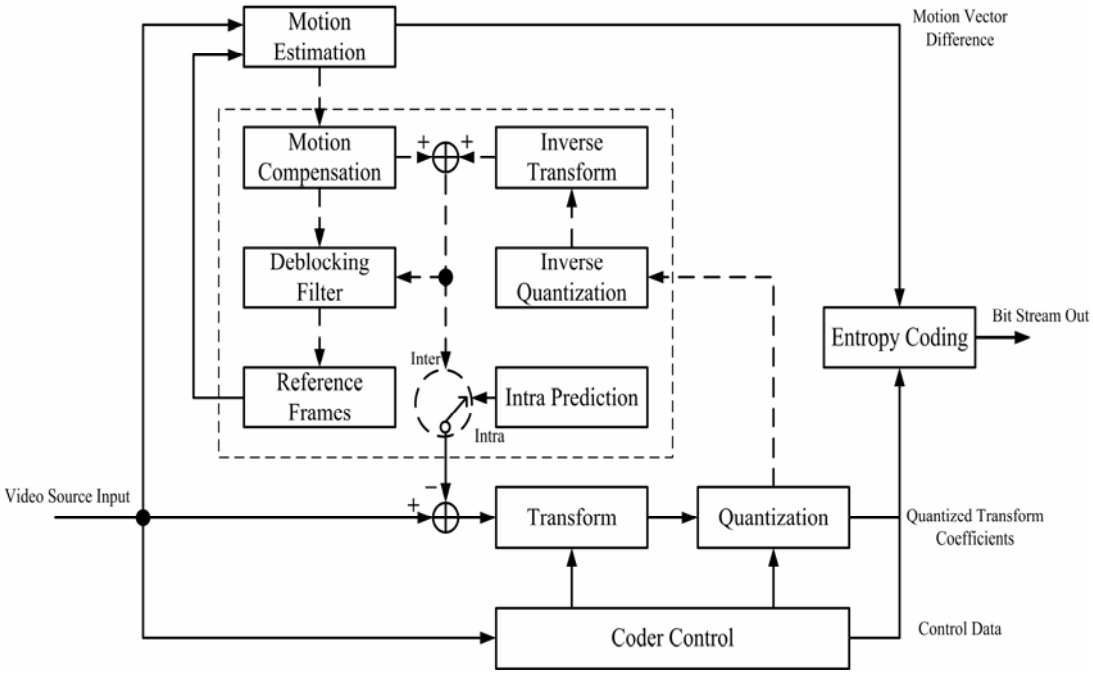



Fig. 2. Encoding architecture for H.264/MPEG-4 AVC.

the well-known video standards. To support 720x480 of frame size under high bandwidth, MPEG-2 has been widely employed in DVD and digital TV broadcast. On the other hand, H.263 is suitable for TV conference and TV phones due to its high compression ratio under low bit-rates. In 1998, MPEG-4 is announced with higher compression ability, visual quality and computational complexity. In 2001, the Joint Video Team (JVT) formed by ITU-T Video Coding Experts Group (VCEG) and ISO/IEC MPEG announced the new video coding standard, H.264/MPEG-4 Advanced Video Coding (AVC) [6]. H.264/MPEG-4 AVC has the advantages of H.263 and MPEG-4 and trades off between the coding gain and implementation cost. Under equal video quality, H.264/MPEG-4 AVC provides double compression ratio as compared to H.263 and 1.5 times compression ratio as compared to MPEG-4. Besides, H.264/MPEG-4 AVC has been proven to have much better visual quality as compared to MPEG-1, -2, -4, and H.263/+/++.

With great coding efficiency and visual quality, H.264/MPEG-4 AVC can be widely applied to many digital video applications. For example, it can be employed in low bit-rates wireless communication, high resolution HDTV, digital video

broadcasting (DVB), ADSL video phone and conference, high resolution DVD, high quality digital camera application, real-time streaming on internet, 3G applications, satellite broadcasting and so on. Besides, H.264/MPEG-4 AVC has become the official standard for the two high resolution DVD formats, HD-DVD and Blu-ray. Also, the Digital Video Broadcasters (DVB) and 3rd Generation Partnership Project (3GPP) have permitted H.264/MPEG-4 AVC as the latest official video standard. Until now, innumerable broadcasting businesses, cable providers and consumer electronic companies have employed H.264/MPEG-4 AVC as the video coding standard for developing their products.

Fig. 2 shows the encoding architecture for H.264/MPEG-4 AVC. Similar to previous standards, the prediction distortion from the difference between intra/inter prediction and reference frame is compacted by discrete cosine transform (DCT). Then, the entropy coding encodes the DCT coefficients and output the results as bit streams. However, to be more advanced, H.264/MPEG-4 AVC provides more characteristic features. Table 1 shows the comparisons for the features in MPEG-2, MPEG-4 ASP and H.264/MPEG-4 AVC. These characteristic features in H.264/MPEG-4 AVC are described as follows:

1. **Variable block size for motion estimation.** Unlike previous standards that utilize fixed size of the 16x16 macroblock, H.264/MPEG-4 AVC employs variable block size that ranges from 16x16 macroblock down to a 4x4 block for motion estimation.
2. **Multiple reference frames.** To increase coding efficiency and prediction accuracy, H.264/MPEG-4 AVC supports multiple reference frames for inter perdition. Even B-frame can be referenced. Besides, the reference order of reference frame is variable instead of depending on the display order in previous standards.
3. **Quarter pixel resolution.** The pixel resolution in many previous standards is half resolution. Quarter pixel resolution is first employed in MPEG-4 part2. In H.264/MPEG-4 AVC, the complexity of interpolation process for quarter

Table 1. Comparisons of MPEG-2, MPEG-4 ASP and H.264/MPEG-4 AVC.

| Features | MPEG-2 | MPEG-4 ASP | H.264/ MPEG-4 AVC |
|---|---|---|---|
| I, P,B frames | Yes | Yes | Yes |
| Multiple reference frames | | | Yes |
| Variable Block size | | | Yes |
| Quarter pixel resolution | | Yes | Yes |
| Weighted Prediction | | | Yes |
| Switching pictures | | | Yes |
| Slice-based motion prediction | | | Yes |
| Interlace | Yes | Yes | Yes |
| MB AFF | | | Yes |
| GMC | | Yes | |
| Integer DCT | | | Yes |
| Huffman coding | Yes | Yes | Yes |
| Arithmetic coding | | | Yes |
| Rate distortion optimization | | | Yes |
| In loop Deblocking filter | | | Yes |
| Bit rate comparison | 100% | 61% | 36% |

pixel resolution is significantly reduced.

4. **Enhanced intra prediction.** H.264/MPEG-4 AVC employs spatial intra prediction. As compared to previous standards, it increases the prediction accuracy in the details of high-motion picture.
5. **Integer 4x4 DCT**. H.264/MPEG-4 AVC employs integer 4x4 DCT instead of floating point 8x8 transform in previous standards.
6. **In-loop deblocking filter.** H.264/MPEG-4 AVC adopts deblocking filter to reduce blocking artifact. The deblocking filter is applied both on the encoder and decoder. For the encoder, the deblocking filter is performed in the compensation loop to improve the quality of reference frame so as to increase the accuracy of inter prediction.
7. **Short word length in calculation**. To save implementation cost and power consumption, H.264/MPEG-4 AVC utilizes 16 bit in calculation instead of 32 bit operation in previous standards.
8. **Enhanced error resilience and network friendliness.** H.264/MPEG-4 AVC can reduce the error rate resulted from the packet loss or channel damage. Hence, it is easier to be applied on network packet control and internet steaming service.
9. **Context-based entropy coding**. H.264/MPEG-4 AVC utilizes context-based variable length coding or context-based binary arithmetic coding for the entropy coding.

### 1.1.2 Platform-based Design for H.264/MPEG-4 AVC Decoder

H264/MPEG-4 AVC has been proven to have much better visual quality and compression ability as compared to the existing standards. However, the high complexity in H264/MPEG-4 AVC becomes the bottleneck for the low-cost and real-time applications. To improve the system performance and reduce the cost, we have to develop more system or architecture design methodologies for H264/MPEG-4 AVC.

In this thesis, we present a baseline H.264/MPEG-4 AVC decoder based on an optimized platform-based design methodology. Some characteristic features in our system design show as follows.

1. **ARM-based platform:** The ARM processor [7-8] is one of the most popular 32-bit microprocessor and widely employed in mobile phones, portable devices and multimedia digital consumer applications. Hence, to quickly integrate our proposed design into system-on-chip system and consider the IP reusability and flexibility of on-chip bus [9], we construct our system on an ARM-based platform.
2. **Software/Hardware co-operation:** In our system, we implement software /hardware partition and jointly optimize the software and hardware design of the decoder. To increase overall decoding throughput, we synchronizes the software procedures and dedicated hardware co-processors.
3. **Macroblock pipeline architecture:** To achieve synchronization so as to enhance throughput, we propose a macroblock-level pipelining [10-11]. In the pipeline schedule, the entropy decoding, motion compensation, inverse transform and deblocking filter perform the decoding process in a macroblock by macroblock manner.

### 1.1.3 Deblocking Accelerator for H.264/MPEG-4 AVC

To conduct hardware and software partition, we profile the AVC decoder and decide to realize 3 dedicated accelerators to speed up deblocking filter, motion compensation and inverse transform respectively. Specifically, we propose a platform-based deblocking filter [12-13] for H.264/MPEG-4 AVC. The deblocking accelerator represents several features as follows.

1. **Adaptive transfer scheme:** To efficiently use the bus bandwidth and reduce the power consumption, we classify the filtering modes into various types. According to the filtering type distribution, we propose an adaptive transfer

scheme to avoid redundant data transfer. Hence, we can significantly reduce bus workload and power consumption. Besides, this scheme also improves our system performance due to less transfer time.

2. **Bus-interleaved architecture:** With bus-interleaved architecture, we perform the filtering operation and the data transfer in parallel. Hence, processing latency can be reduced. Besides, this bus-interleaved architecture also has the advantages of low cost and low memory access frequency.
3. **Non-buffered memory architecture:** We propose non-buffered memory architecture for inverse transform and deblocking process. In an interleaved process manner, the inverse transform results can be propagated to deblocking module immediately without to be buffered. As compared to traditional shared memory architecture, our non-buffered architecture is more competitive that reduces significant cost in memory buffer and memory access.

## 1.2 Contribution and Organization

In this thesis, we present a macroblock-level pipelining H264/MPEG-4 AVC decoder based on an optimized platform-based design methodology. The synchronization of the software and the dedicated hardware co-processors increases throughput. Specifically, to speed up deblocking filter, we proposed a deblocking accelerator with bus-interleaved architecture. For more details, the remainder of this thesis is organized as follows:

- Chapter 2 introduces the algorithm of H.264/MPEG-4 AVC decoder.
- Chapter 3 describes our proposed platform-based system design for H.264 /MPEG-4 AVC decoder.
  - We use platform design methodology to increase system flexibility and reusability.
  - We analyze the computational complexity for each functional module and perform software and hardware partition.

- We conduct software and hardware co-operation for parallel processing.
- We propose macroblock-level pipelining architecture to improve system throughput.
- We propose non-buffered memory architecture to significantly reduce memory cost and achieve the same performance as compared to traditional shared memory architecture [17].

- Chapter 4 illustrates our proposed hardware accelerators for H.264/MPEG-4 AVC decoder. For the design of deblocking filter, our contributions include the following:
  - We propose adaptive transfer scheme to reduce 25%-94% bus bandwidth requirement as compared to [14-20]. Hence, the costly frame-length buffer used for reducing bus workload in [17],[20] can be removed.
  - We propose bus-interleaved architecture to parallel process data transfer and filtering operation. Not only reducing processing latency, the bus-interleaved architecture can avoid the usage of costly dual-ported local memory in [14],[16],[18]-[20].
  - We propose an overlapped scheme for the calculation of boundary strength. Hence, the calculation of boundary strength and deblocking filter can be performed in parallel so as to reduce latency.
  - Our design offer up to 7.1x improvement on processing latency and uses simpler memory configuration as compared to [14-20].

- Chapter 5 presents the experiment results. We compare 4 types of architecture and evaluate system performance on ARM966-based platform.
  - As compared to H.264 reference software JM6.0 decoder, we have 9-16 times improvement to achieve a decoding average rate of 7.3 fps and up to 10.4 fps for QCIF video sequences.

- Lastly, Chapter 6 concludes this work.

# Chapter 2

# H.264/MPEG-4 AVC Decoder

## 2.1 Introduction

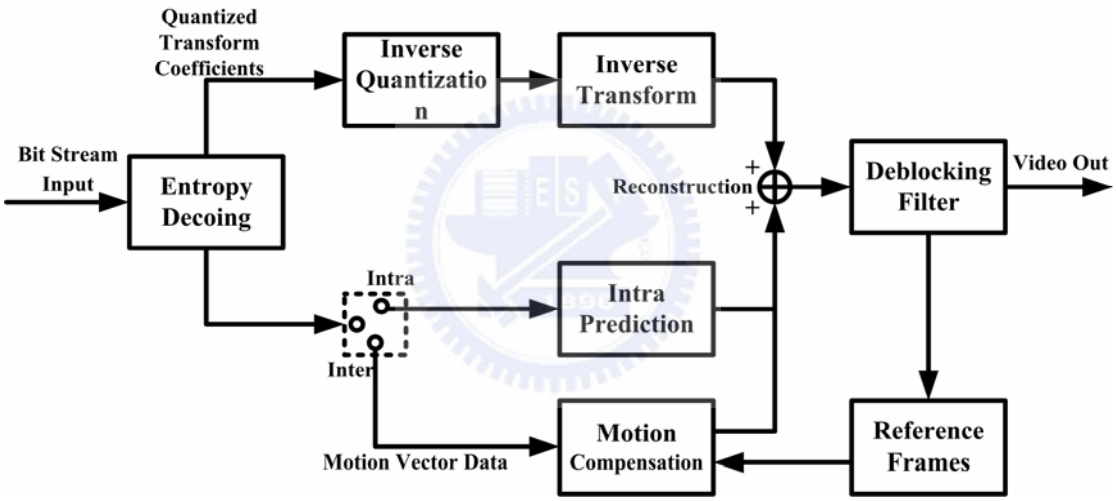



Fig. 3. Decoding architecture for H.264/MPEG-4 AVC.

Fig. 3 shows the decoding flow for H.264/MPEG-4 AVC decoder where the decoding tasks are partitioned into four main parts as follows.

1. Entropy coding.
2. Motion compensation (MC) or intra prediction.
3. Inverse quantization and inverse 4x4 discrete cosine transform (IQ-IDCT).
4. In loop deblocking filtering.

The decoding is performed by first parsing the compressed bit-stream by entropy coding. After the parsing, the quantized prediction residues and macroblock side information including macroblock type, the prediction mode, and the motion vector difference are extracted. The extracted macroblock type determines the prediction type. The intra prediction values are derived based on the neighboring pixels for an intra macroblock and the inter prediction values are generated from motion compensated pixels for an inter macroblock. The addition of the prediction and the decoded residuals produces the reconstructed frame. After the reconstruction, the in loop deblocking filter is applied to reduce blocking artifacts and the deblocking results are put into the frame buffer as reference frames.

## 2.2 Context-Based Adaptive Variable Length Coding

In H.264/MPEG-4 AVC, there are two types of entropy coding: Context-based Adaptive Binary Arithmetic Coding (CABAC) [21] and Context-based Adaptive Variable-Length Coding (CAVLD) [22]. Since our design is based on the Baseline Profile of H.264/MPEG-4 AVC, we utilize CAVLD as the entropy coding. Some features of CAVLD are described as follows:

1. The entropy encoding order for a 4x4 block is based on zigzag scan as shown in Fig. 4.
2. In a context-based adaptive manner, the number of non-zeros is encoded by using a look-up table depending on the number of non-zeros in the adjacent blocks.
3. In an adaptive manner, the value of non-zero coefficients is encoded by using VLC look-up tables.
4. By taking advantage of many zero results produced by transform, the strings of zeros before the last non-zeros are compacted by using run-level coding.

Fig. 5 shows the flowchart of CAVLC encoding. The right part in the figure goes on

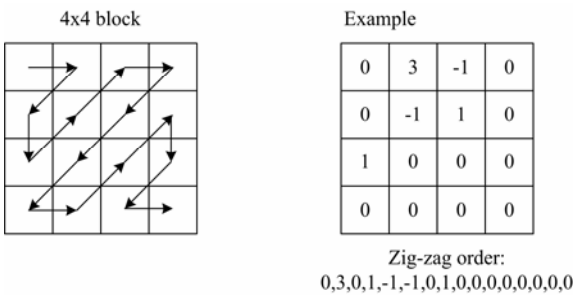



Fig. 4. Zigzag scan order for entropy encoding.

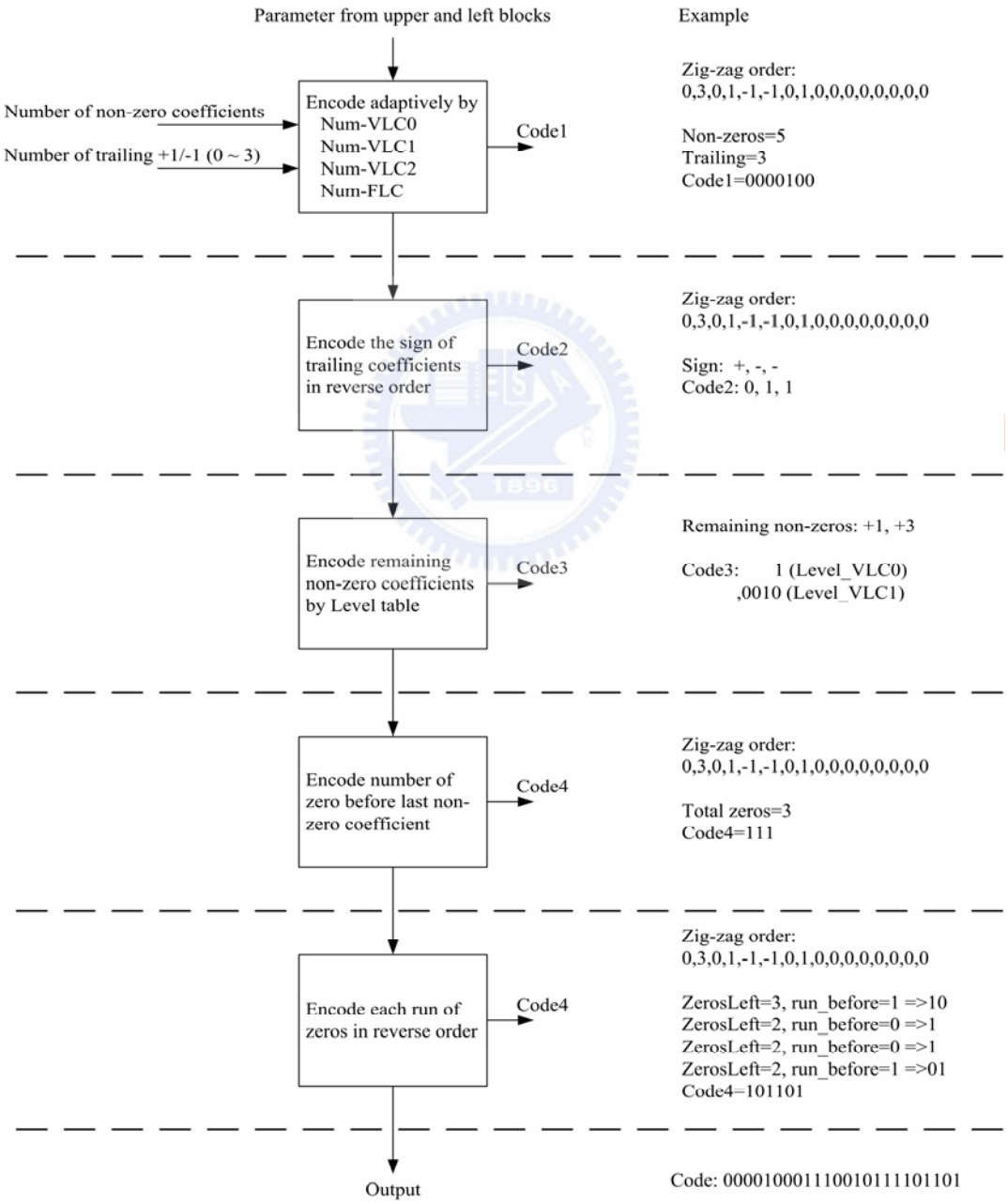



Fig. 5. Flow chart of CAVLC encoding.

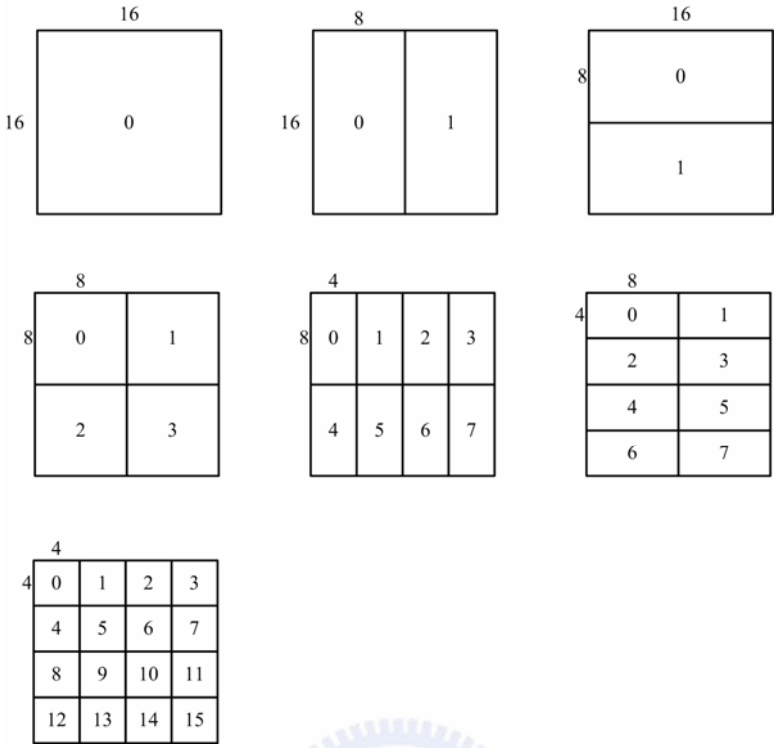



Fig 6. Variable partition sizes for inter prediction

an example. Number of non-zeros, number of +1/-1, signs of +1/-1, values of non-zeros, string of zeros and each run of zero before last non-zero coefficient are encoded in order. Also, in the decoder, we can perform CAVLD decoding in a reverse fashion.

## 2.3 Motion Compensation

In H.264/MPEG-4 AVC, inter prediction has 7 types of partition size for each 16x16 macroblock. As shown in Fig.6, the basic size for a motion vector can be 16x16, 16x8, 8x16, 8x8, 8x4, 4x8 or 4x4. In an adaptive manner, we can choose larger partition size for motion vector when a macroblock contains fewer details. Hence, the coding complexity and blocking artifacts can be reduced.

Besides, H.264/MPEG-4 AVC utilizes quarter pixel resolution for motion vector. To achieve quarter pixel resolution, we have to produce the sub-pixel samples first at half pixel positions. After all half pixel samples are filtered by a 6-tap FIR filter, the

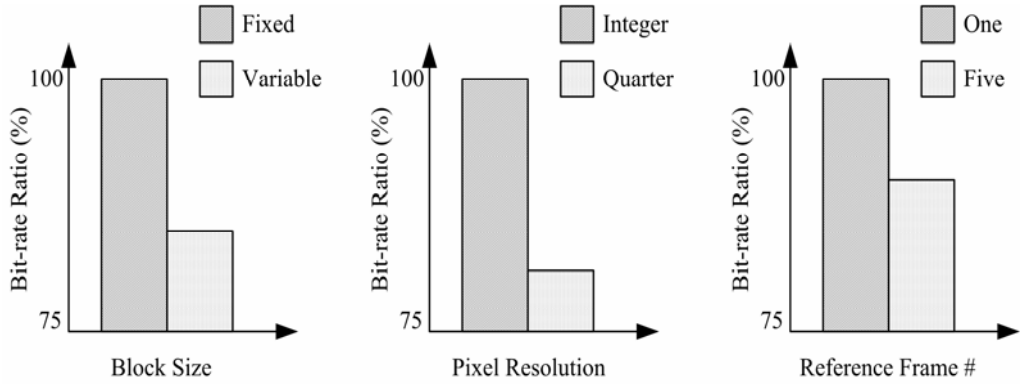



Fig.7. Bit rate comparison for the catachrestic features of motion estimation

quarter pixel samples are produced by using bilinear interpolation between the adjacent half pixel samples or integer-pixel ones. Besides, another important feature for motion prediction is the utilization of multiple reference frames. Thus, H.264/MPEG-4 AVC can provide better visual quality and more efficient encoding. Fig. 7 shows the comparisons of bit-rates for the catachrestic features in motion estimation.

Moreover, H.264/MPEG-4 AVC uses motion vector prediction scheme to reduce bit-rates. Hence, only the motion vector difference (MVD), which is the difference between the exact vector and the predicted vector, is transmitted. Then, the decoder has to calculate motion vector by adding MVD to the motion vector prediction. Next, we can compensate the prediction frame from multiple reference frames under quarter pixel resolution in the decoder.

## 2.4 Intra Prediction

H.264/MPEG-4 AVC utilizes intra prediction when the current sample is not highly correlated with other reference frames, such as I picture. To take advantage of the correlation between the neighboring samples within the same frame, the current sample can be predicted depending on the neighboring sample. By using neighboring

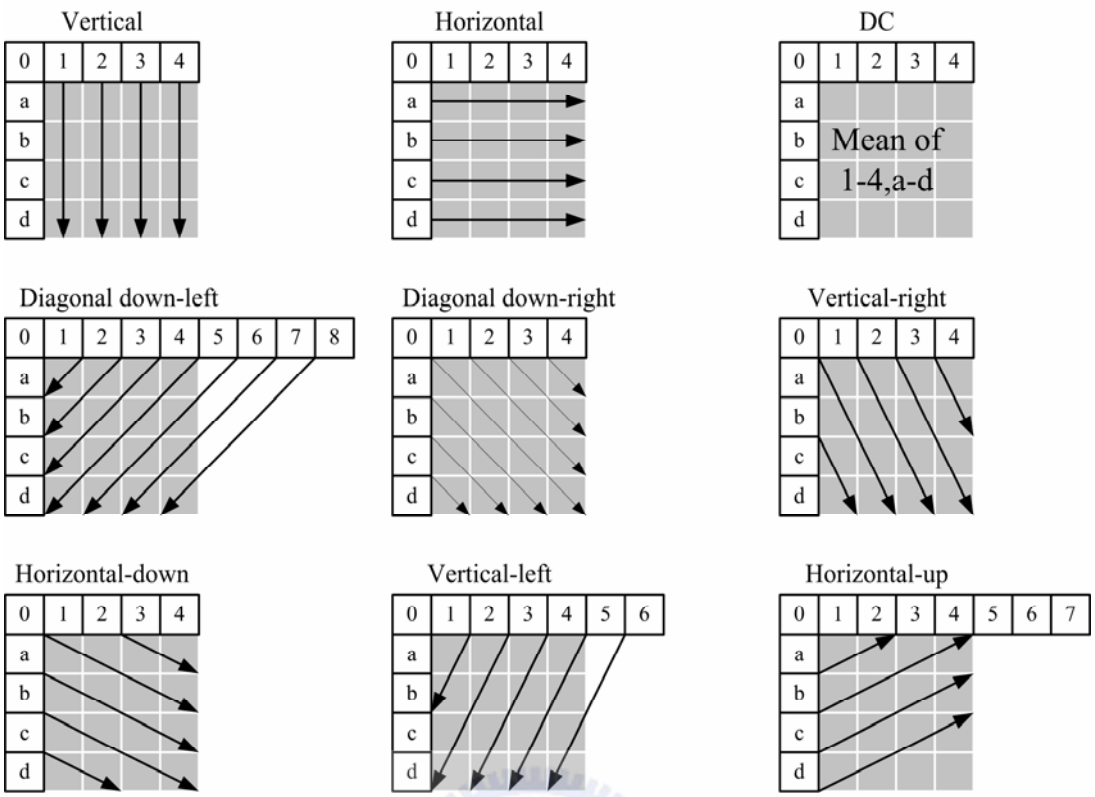



Fig.8. Nine 4x4 luma prediction modes.

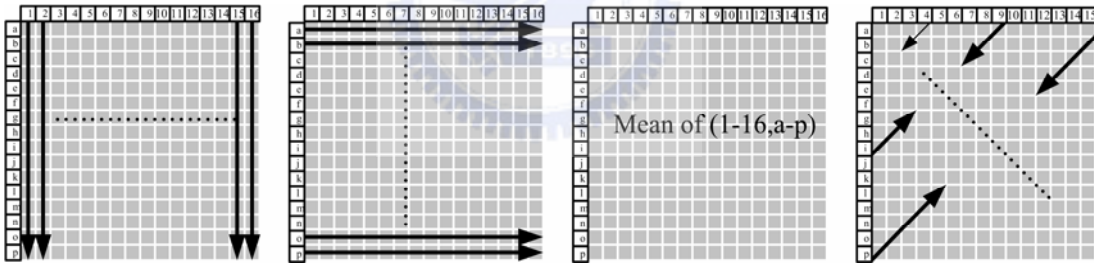



Fig.9. Four 16x16 luma prediction modes.

sample for prediction, H.264/MPEG-4 AVC provides 9 types of intra prediction mode for 4x4 luminance block as shown in Fig.8. Besides, another 4 types of intra prediction mode for 16x16 luminance macroblock as shown in Fig.9 are applied when the current sample contain fewer details. Also, the chrominance goes the same fashion for each 8x8 chrominance components.

## 2.5 Inverse Quantization

To be more accurate in trading-off between bit-rates and quality, H.264/MPEG-4 AVC defines 52 levels for quantization. There are 52 quantization parameter (QP) related to each quantization level. To conduct inverse quantization, we have to obtain the quantization step size first. As shown in Fig. 10, the quantization step size becomes double for each increment of 6 in QP. Based on the quantization step size, the inverse quantization coefficient is obtained by the multiplicity of quantized coefficient and quantization step size.

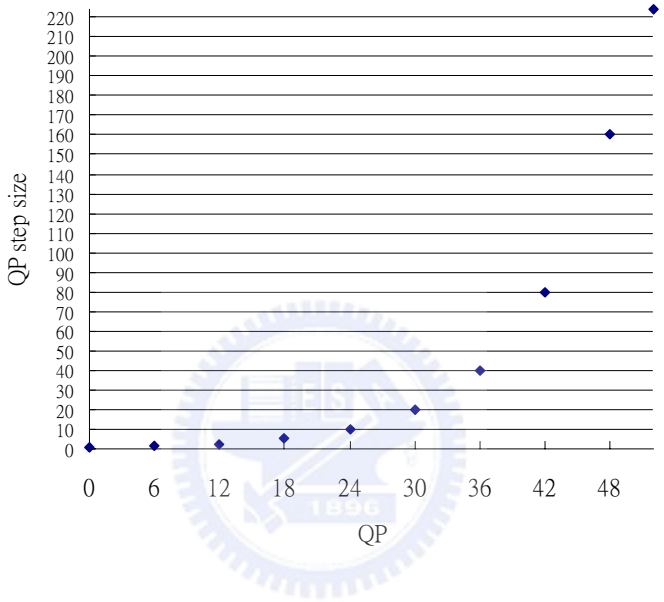



Fig. 10. Quantization step sizes and related QP.

## 2.6 Inverse Discrete Cosine Transform

H.264/MPEG-4 AVC utilizes the 4x4 integer Discrete Cosine Transform (DCT) [23] to transform prediction distortion so as to remove spatial correlation inside it. Unlike 8x8 floating point transform in previous standards such as MPEG-1, -2, -4, and H.263, integer transform can avoid the mismatch caused by floating rounding when inverse transform is performed. Besides, H.264/MPEG-4 AVC utilizes smaller size of a 4x4 block for transform because the correlation among prediction residuals is significantly reduced in H.264/MPEG-4 AVC.

To transform each 4x4 block, Fig.11 (a) shows the Two-dimensional 4x4 integer

$$\text{(a)}\quad \mathbf{Y}_{4x4} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} \mathbf{X}_{4x4} \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \otimes \mathbf{E}_{4x4}$$

$$\text{(b)}\quad \mathbf{X}_{4x4} = \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix} \begin{bmatrix} \mathbf{Y}_{4x4} \otimes \mathbf{E}_{4x4}/2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix}$$

Fig.11. Two-dimensional 4x4 (a) transform and (b)inverse transform.

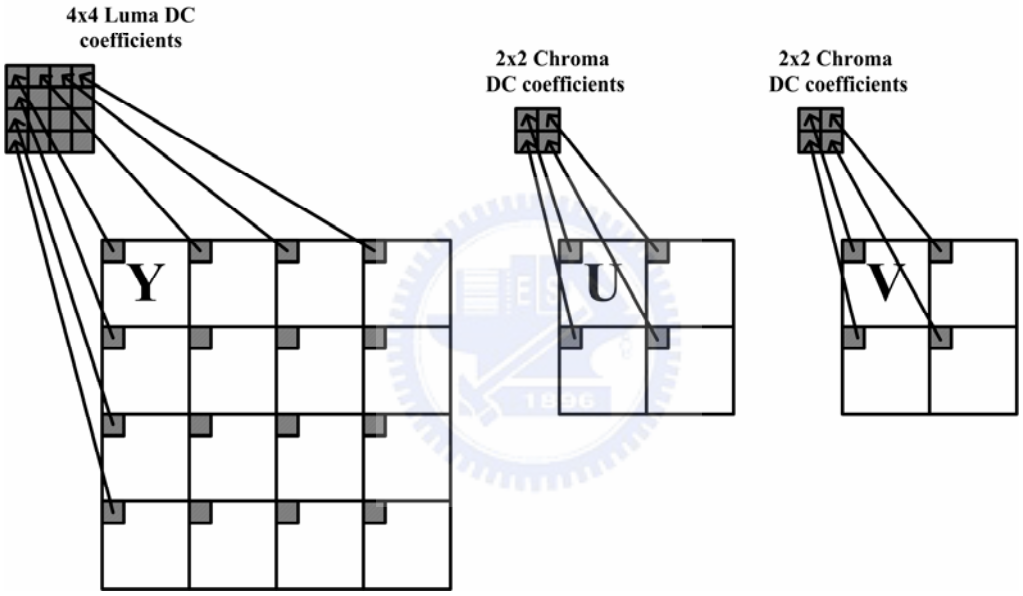



Fig. 12. DC coefficients within a macroblock.

transform in H.264/MPEG-4 AVC and Fig.11 (b) show the inverse transform. By using the post-scaling and pre-scaling the scaling factors (the 4x4 matrix E in Fig 11), the DCT and inverse DCT operation can be simplified so that only addition, subtraction and shift operation are required. Hence, it is very suitable for low-cost and high-speed hardware implementation.

If the current macroblock is intra-coded, the DC coefficients of each block as shown in Fig.12 contain much energy. Hence, these DC coefficients require to be transformed again to reduce the correlation among them so as to increase compression

$$(a)\quad W_{4x4} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} DC_{\_Luma\_4X4} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

$$(b)\quad W'_{2x2} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} DC_{\_Chroma\_2X2} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Fig.13. Two-dimensional Hadamard transform for (a) luma and (b) chroma DC coefficients.

performance. H.264/MPEG-4 AVC utilizes Hadamard transform to transform the DC coefficients. Fig.13 shows the Two-dimensional 4x4 Hadamard transform for luma DC coefficients and 2x2 Hadamard transform for chroma ones. Therefore, the decoder has to perform inverse Hadamard transform of DC coefficients first then each block can update its DC coefficient to perform the 4x4 integer discrete cosine transform.

## 2.7 Adaptive In-loop Deblocking Filter

### 2.7.1 Video filtering in previous standards

H.264/MPEG-4 AVC employs 4x4 DCT/IDCT for transform. However, it introduces noticeable blocking artifact especially at low bit-rates. The blocking artifact results from three sources: (1) the nature discontinuity of transform digital signals, (2) the distortion of quantization which enhances the blocking effect when quantization parameter is large, and (3) the propagation of blocking artifact from reference frames when conducting motion compensation.

To eliminate blocking artifact, the simplest way is to utilize a FIR low-pass filter to smooth the block boundary. However, low-pass filtering causes the blurring effect that decreases the visual quality. In H.263, overlapped-block motion compensation

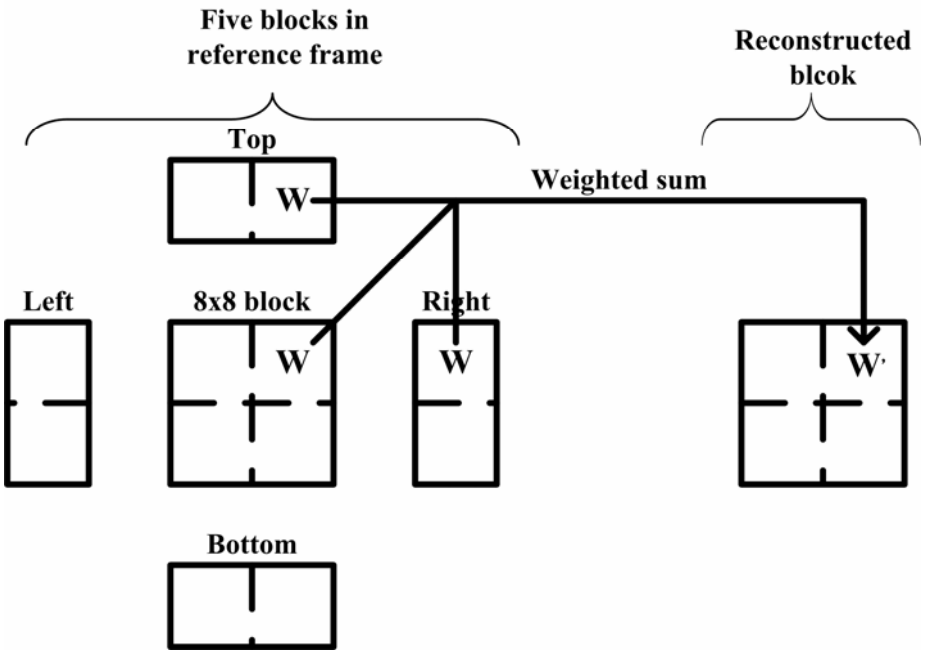



Fig. 14. Overlapped-block motion compensation (OBMC) in 263.

(OBMC) [24] is employed to reduce the blocking artifact. The OBMC is not only applied in the decoder to improve the visual quality of display video but also in the encoder to increase the accuracy of motion estimation. Fig. 14 shows the reconstruction operation in OBMC mode of H.263. Each 8x8 block is reconstructed by a combination of the upper, bottom, left right and current block in the reference frame. For a 4x4 block within the 8x8 block, each reconstructed pixel is the weighted sum of 3 prediction values depending on the motion vectors from 3 adjacent blocks in reference frame. For example as shown in Fig. 14, all pixels of the 4x4 block W' are constructed by the accumulation of every block W in the reference frame. By the weighting operation, the blocking artifact can be reduced.

Different to H.263, MPEG-4 employs an adaptive deblocking filter [25]. Because the deblocking filter is post-processing, it is only applied on the decoder to improve the quality of the output video sequence and reference frames used by motion compensation. The filter operations are performed along the 8x8 block edges. Based on the sample values, one of two filter modes, smooth mode and default mode, is judged. Then, different taps of filtering is applied depending on the quantization

parameter. As compared to OBMC in H.263, the adaptive filter has better ability to avoid the filtering on the image region where human vision is less susceptible to blocking artifact. Hence, the computational complexity can be reduced.

By taking the advantages of OBMC in H.263 and the adaptive filter in MPEG-4, H.264/MPEG-4 AVC utilizes an in-loop adaptive deblocking filter to reduce blocking artifact. Both encoder and decoder apply the deblocking filter to increase the accuracy of motion estimation or compensation. As compared to the deblocking filter in MPEG-4, the deblocking filter in H.264/MPEG-4 AVC is more complex with more control parameter, filter modes, and different types of FIR filter. The following sections describe the operation of deblocking filter in H.264/MPEG-4 AVC.

### 2.7.2 Deblocking Process

The in-loop deblocking filter in H.264/MPEG-4 AVC is designed to reduce the blocking artifacts. As compared to the decoder without applying deblocking filtering, the bit rate can be saved 5%-10% when deblocking filter is applied under the same performance. The filter operation is applied to each edge of a 4x4 block. Fig. 15 shows the edge filtering order within a 16x16 luminance macroblock. As shown, the vertical edges are filtered first and then the horizontal ones. In addition, for filtering an edge of a 4x4 block, consecutive 8 pixels from the same row (or column) of two adjacent 4x4 blocks are required. For example in Fig. 15, the pixels (A0-A3, B0-B3) are accessed for the vertical (or horizontal) filtering of a 4x4 block. Particularly, each sample pixel of (A0-A3, B0-B3) is filtered adaptively by different filter taps. To decide the filter tap for each pixel, the following factors are considered:

1. Boundary strength.
2. Thresholds of $\alpha$ and $\beta$.
3. The content of sample pixels.

### 2.7.3 Boundary Strength

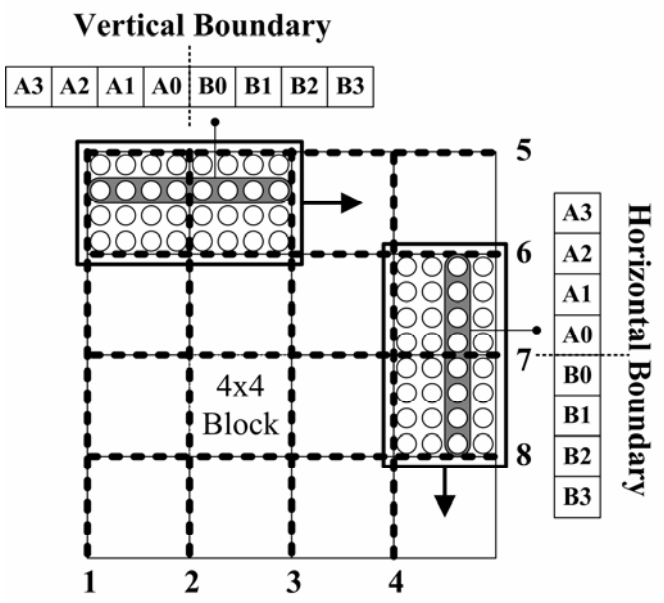



Fig. 15 Sequential order for filtering the edges of 4x4 blocks in a luminance macroblock.

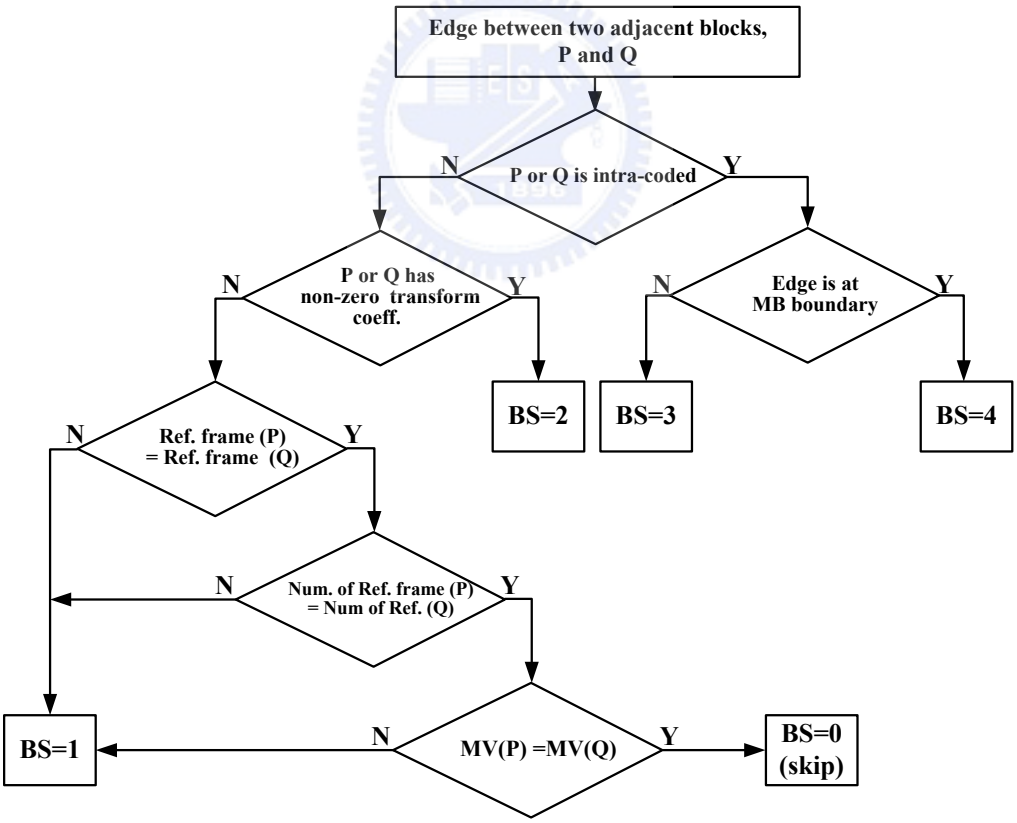



Fig.16. Decision flow of boundary strength (bS) where P and Q denote two adjacent 4x4 blocks.

The boundary strength (bS) level is mainly used to decide the necessity of filtering and filter type. In H.264, the bS has 5 levels. The actual level is determined by the MB type, edge position, reference frame type, and motion vectors of two adjacent blocks. Fig. 16 shows the decision of bS level. As shown, the strongest bS level, i.e., bS=4, is identified when two adjacent blocks are intra coded and locate at the MB boundary. In this case, obvious blocking artifact could be noticed. As a result, higher bS level invokes stronger low pass filtering. On the other hand, when the bS is at the weakest level, i.e., bS=0, there is no filtering.

### 2.7.4 One-dimension Filtering Decision

Fig.17 elaborates the detail about how these factors are used to decide the filter tap for each pixel of (A0-A3, B0-B3). In addition to the bS level, the parameters (α, β) are used to preserve the real edge. In Equation (1), the necessity of filtering is also controlled by the parameters (α, β). Specifically, α and β are assigned with higher values to increase the possibility of filtering as higher quantization parameters cause more noticeable blocking artifact. In contrast, smaller α and β are used for lower quantization parameters.

$$bS!=0 \ AND \quad |A0-B0|<\alpha \ AND \quad |A1-A0|<\beta \ AND \quad |B1-B0|<\beta \qquad \text{Equation (1)}$$

Hence, the first step is to use Equation (1) for deciding whether the filtering is required or not. Then, according to the bS level, thresholds (α, β) and the absolute differences of adjacent reconstructed pixels, different filters are applied to different pixels. Specifically, in Fig. 17, not all the input pixels (A0-A3, B0-B3) will be updated with the filtered results. For example, if bS is not of strongest level, only A0, B0, A1, B1 are updated. For those pixels without update, the pixel values are unchanged. The process is continued by sliding the filtering window one block to the right (or to the bottom) at a time as in Fig. 15. Note that the updated (B0-B3) could be

used for the filtering of next adjacent block when the filtering window slides one block to the right (or to the bottom).

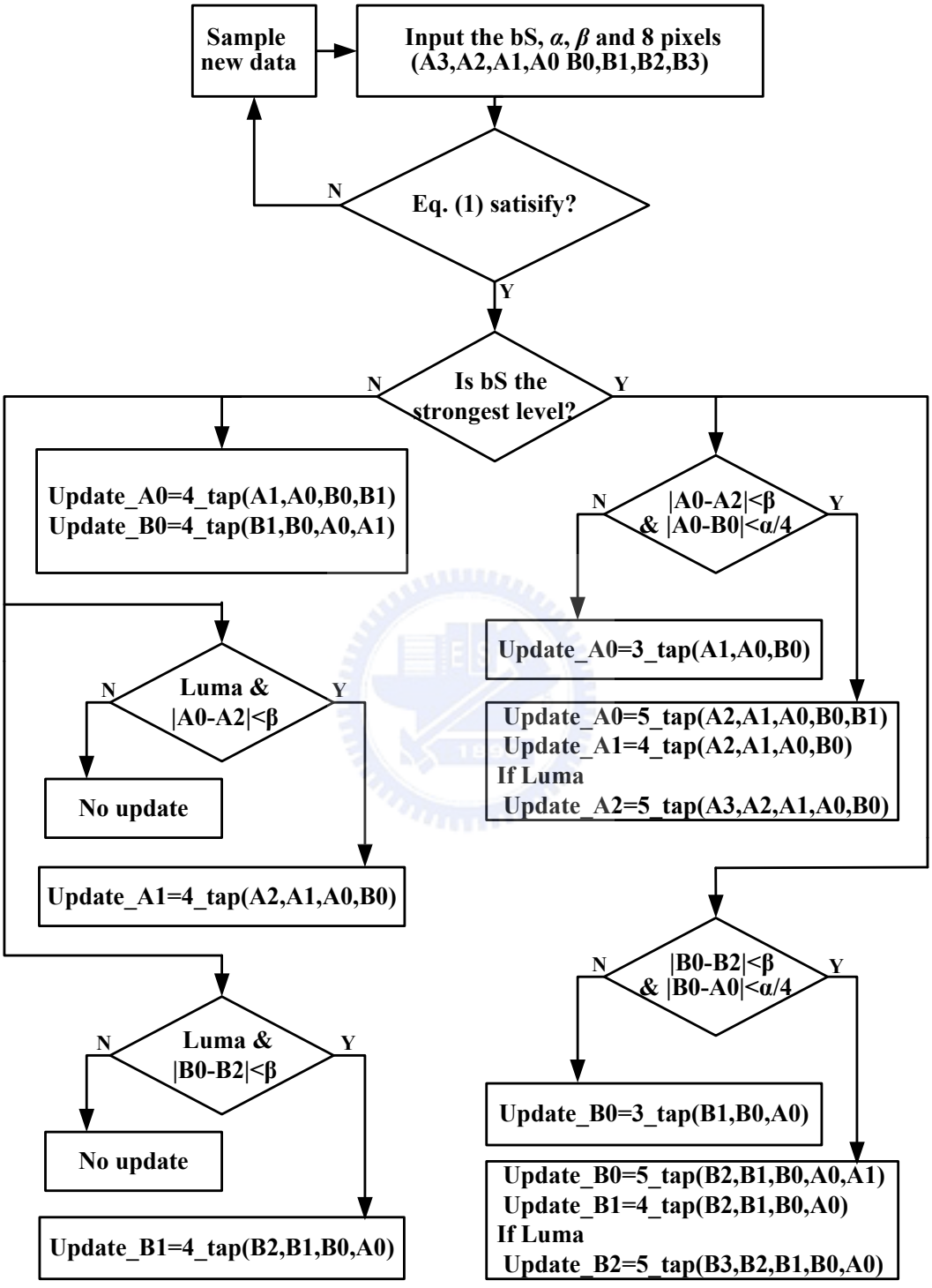



Fig. 17. Decision flow of filter tap selection.

# Chapter 3

# Platform-based System Design for H.264/MPEG-4 AVC Decoder

## 3.1 ARM Microprocessor Introduction

ARM (Advanced RISC Machines) [7] Ltd. is an IP cooperation founded in 1990 by Hermann Hauser. It leads the industrial providing of the 32-bit embedded RISC microprocessor and is the most widely-used 32-bit microprocessor family in the world. The characteristic features of ARM processors are high-performance, low-cost, low power consumption. It is especially suitable for the appication in mobile phones and about 70% of all modern mobile phones are embedded with the ARM processor core. In fact, ARM microprocessor can be integrated into all portable wireless communications, hand-held computing, automotive systems, mass storage device, and multimedia digital consumer applications, such as MP3 engines, and personal digital assistants (PDAs).

Table 2. Comparisons among different ARM process families.

| Process Family | Pipeline Stages | Memory Organization | Clock Rate (MHz) |
|---|---|---|---|
| ARM6 | 3 | Unified | 25 |
| ARM7 | 3 | Unified | 66 |
| ARM8 | 5 | Unified | 72 |
| ARM9 | 5 | Harvard | 200 |
| StrongARM | 5 | Harvard | 233 |
| ARM10 | 6 | Harvard | 400 |
| ARM11 | 8 | Harvard | 533 |

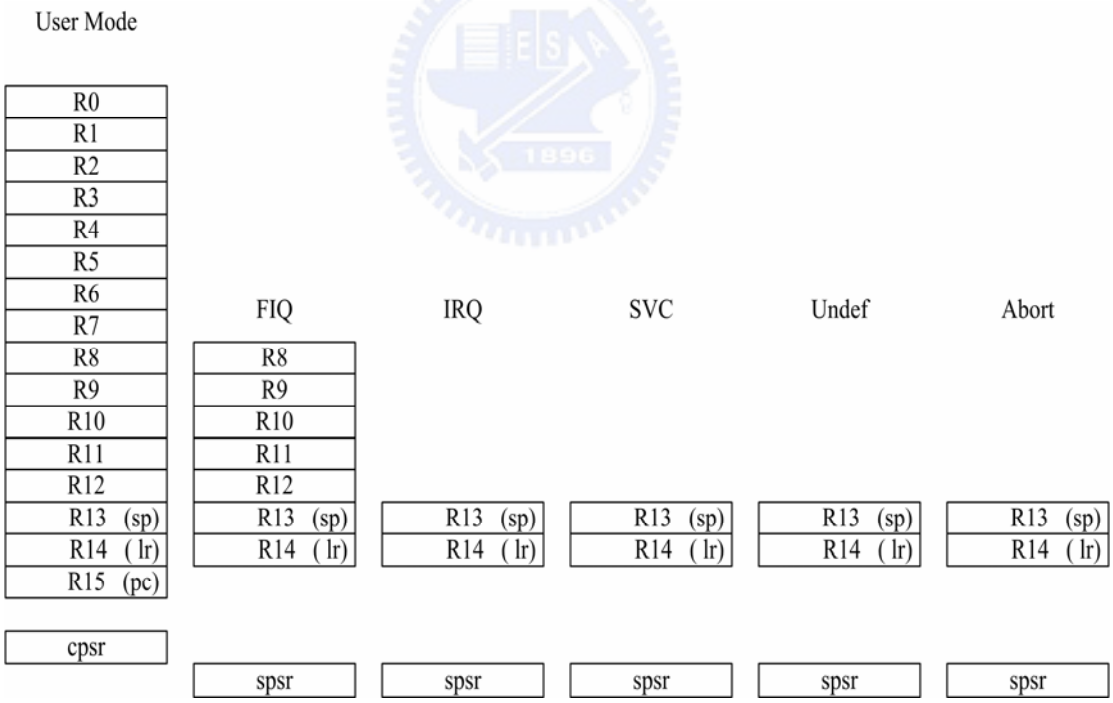



Fig. 18. 37 sets of 32-bit registers in ARM processor.

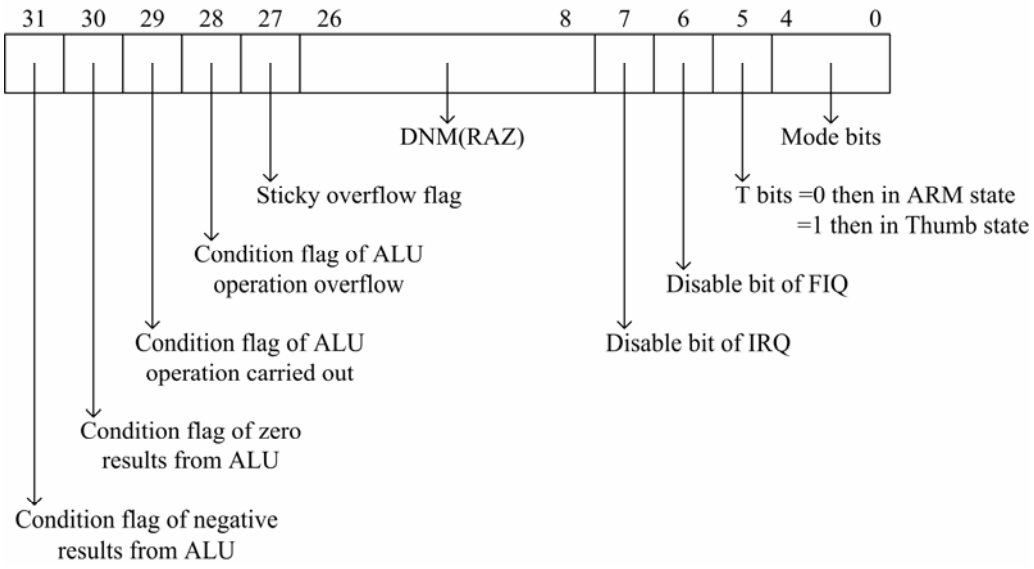



Fig. 19. 32-bit program status registers.

ARM microprocessor is based on 32/16-bit RISC architecture. There are two types of instruction set. One is the 32-bit ARM instruction set that can be applied when considering performance. The other is the 16-bit Thumb instruction set that can be applied when considering increasing the code density. Besides, the ARM has 7 operating mode (User, FIQ, IRQ, Supervisor, Abort, Undefined, and System mode) and 37 sets of 32-bit registers (31 of them are general purpose registers and 6 of them are program status registers) as shown in Fig. 18. Also, Fig. 19 shows the task for each bit of program status registers.

In this thesis, we utilize the ARM966E-S [8] as our embedded microprocessor. The ARM966E-S that belongs to the ARM9 family supports 5 stages for pipeline configuration and up to 200 MHz clock rate as shown in Table 2. Fig.20 shows the 5-stage pipeline data path. Besides, in the architecture of ARM966E-S as shown in Fig. 21, it contains the interface of tightly coupled memory (TCM), interface of optional ETM9, interface of advanced high-performance bus, and a coprocessor interface for connection of acceleration hardware.

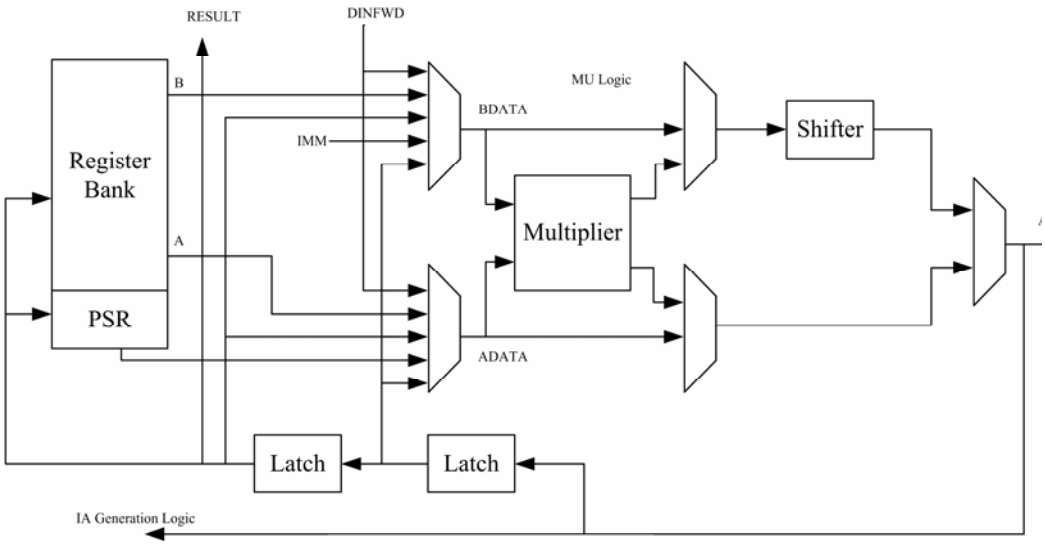



Fig. 20. Five-stage pipeline data path.

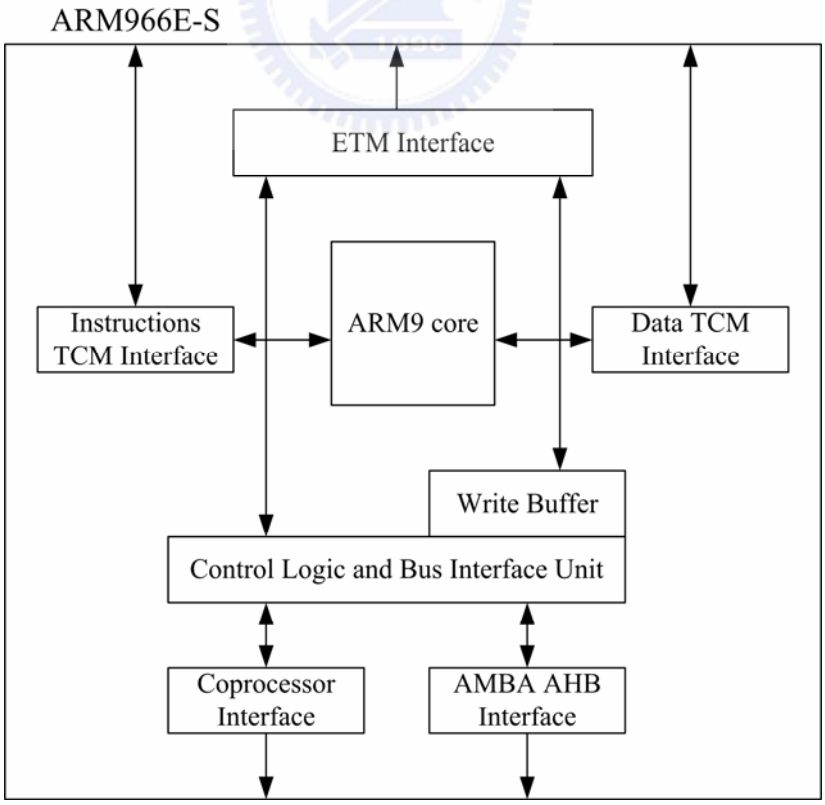



Fig. 21. Architecture of ARM966E-S.

# 3.2. Advanced Microcontroller Bus Architecture (AMBA) Overview

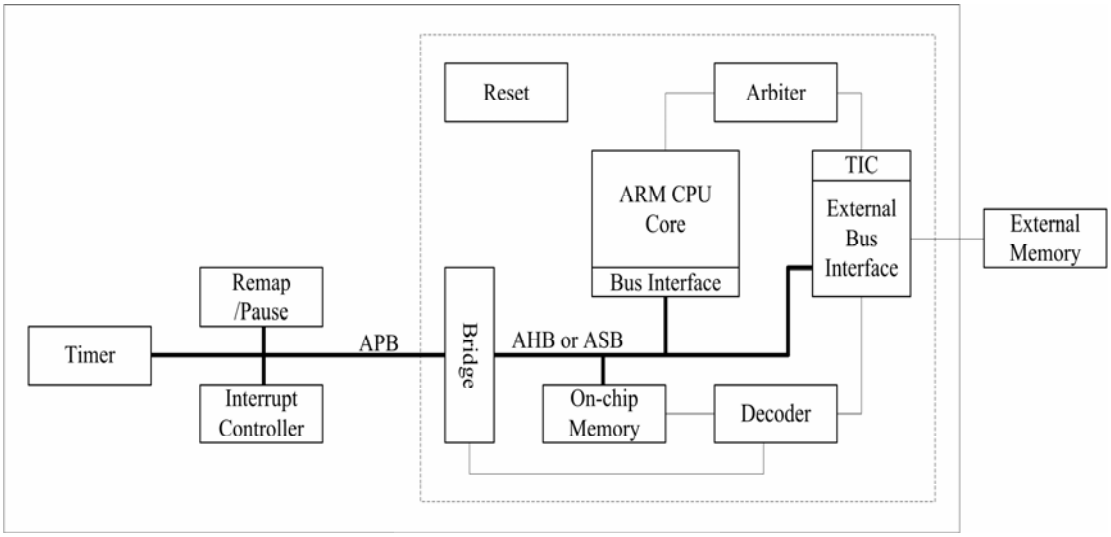



Fig. 22. AMBA architecture.

The open standard of AMBA (Advanced Microcontroller Bus Architecture) [9] provides a solution for the flexibility and reusability under system-on-chip (SOC) integration. Fig. 22 shows the architecture. With multi-layer architecture, AMBA utilizes on–chip bus connecting with embedded processors to conduct the on-chip memory or peripherals. To optimize the utilization of bandwidth and frequency of on-chip bus, AMBA defined three types of bus:

1. The Advanced High-performance Bus (AHB)
2. The Advanced System Bus (ASB)
3. The Advanced Peripheral Bus (APB)

The AHB and ASB support multiple masters and burst transfers to conduct a pipelined operation with high performance. Moreover, the AHB supports more transfer techniques, such as split transactions and wide data bus configurations. On the other hand, with simple interface, the APB is suitable for many peripherals to

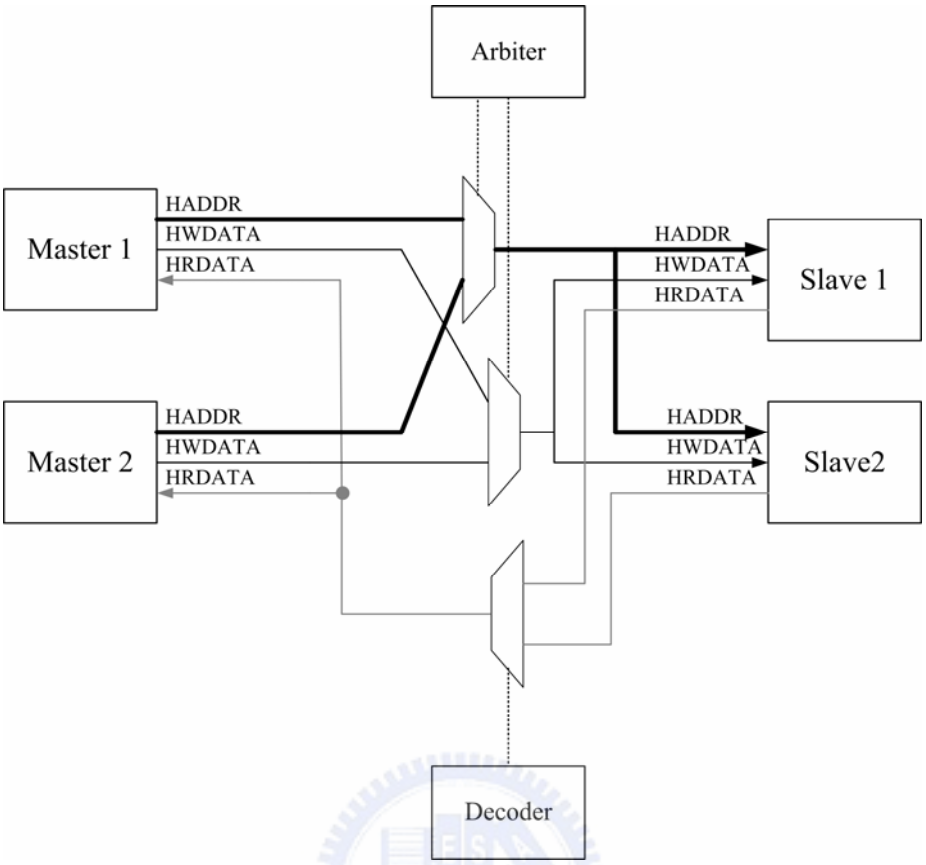



Fig 23. AHB components and multi layer interconnection.

conduct bus protocol from high performance to low power and low bandwidth bus.

## 3.3. Advanced High-performance (AHB) Bus Introduction

Multi-layer AHB is a new standard of AMBA 2.0 [9]. With single-clock edge operation, the AHB reduces the delay of multiple masters system and efficiently use the bandwidth. The AHB bus bandwidth is allowed to be 8, 16, 32, 64, 128, 256, 512, and 1024 bits. Considering the 32-bit configuration of general purpose processors and high performance of AHB, in this thesis, we apply the 32-bit AHB as our system on-chip bus.

To describe the protocols and architecture of the AHB, Fig 23 shows the AHB components and multi-layer interconnection. A typical AMBA AHB system includes four components: master, slave, arbiter and decoder. Each component is described as following:

1. AHB master: The bus master has the ability to perform read or write operation on the bus. The AMBA supports multiple AHB masters on the system. Typical AHB masters include the CPU processors, the DMA (direct memory access) controller, the DSP (digital signal processors) and so on. However, only one bus master is permitted to use the AHB bus at any one time.
2. AHB slave: The bus slave waits the reading or writing demand from masters and response the transfer condition. Typical AHB slaves include the internal memory, the external memory interface, the APB Bridge and so on.
3. AHB arbiter: Because only one bus master is permitted to use the AHB bus, an AHB arbiter is constructed on the bus to judge the access priority of active masters.
4. AHB decoder –The AHB decoder decodes the bus address and selects one of the signals from the slave modules depending on the decoded results.

Basically, the transfer signals on AHB bus include clock, arbitration, address, control signal, write data, read data, and response signal. Table 3 shows the detailed AHB signals and Fig 24 shows these signals applying on the interfaces of AHB master, slave, arbiter and decoder. Besides, to increase the system performance, the AHB utilizes some important transfer techniques:

1. Pipeline operation: An AHB transfer consists two phases. One is the address phase that conducts the transfers of address and control signals. The other is the data phase that conducts the transfers of read, write and response signals. With

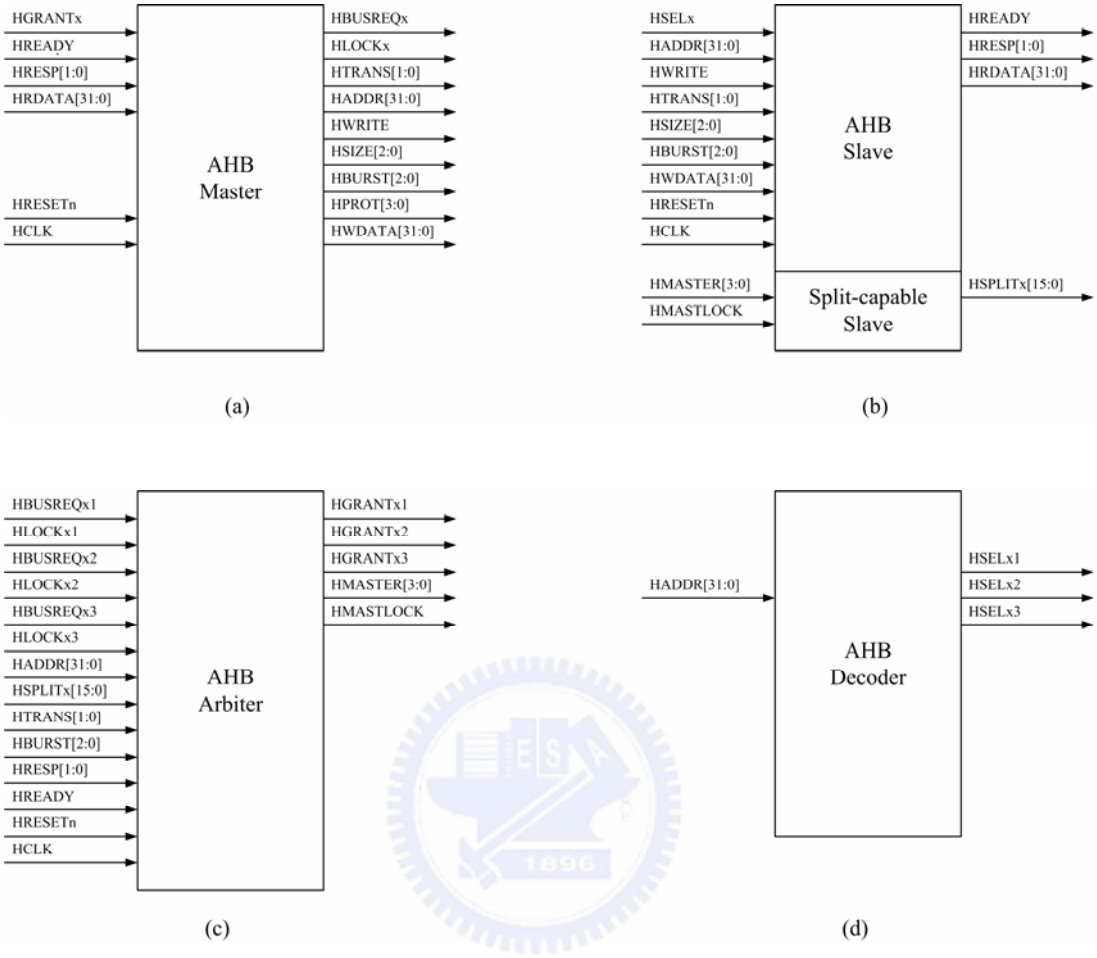



Fig. 24. The interfaces of AHB (a) master, (b) slave, (c) arbiter and (d) decoder.

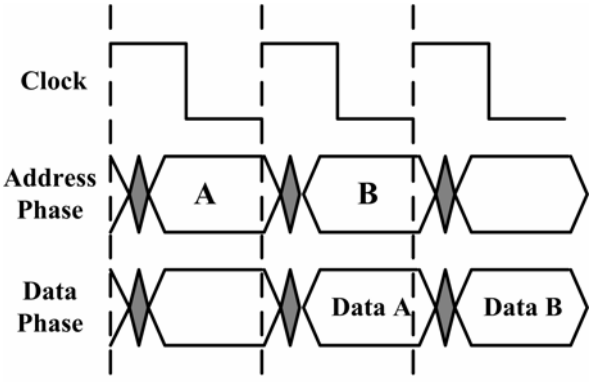



Fig. 25. AHB pipeline transaction.

Table 3. The detailed AHB signals

| Name | Source | Signal Type |
|---|---|---|
| HADDR[31:0] | Master | Address |
| HTRANS[1:0] | Master | Response |
| HWRITE | Master | Control |
| HSIZE[2:0] | Master | Control |
| HBURST[2:0] | Master | Control |
| HPROT[3:0] | Master | Control |
| HWDATA[31:0] | Master | Write |
| HSELx | Decoder | Control |
| HRDATA[31:0] | Slave | Read |
| HREADY | Slave | Response |
| HRESP[1:0] | Slave | Response |
| HBUSREQx | Master | Arbitration |
| HLOCKx | Master | Arbitration |
| HGRANTx | Arbiter | Arbitration |
| HMASTER[3:0] | Arbiter | Arbitration |
| HMASTLOCK | Arbiter | Arbitration |
| HSPLITx[15:0] | Slave | Arbitration |
| HCLK | Clock Source | Clock |
| HRESETn | Reset | Reset |

pipelined transaction, the data phase of previous transfer can be overlapped with the address phase of current transfer to increase throughput. For example as shown in Fig. 25, the address phase B is overlapped with the data phase A.

2. Burst transfer: Table 4 shows the 8 burst types depending on the HBURST signal. It supports 1, 4, 8, 16 beat and undefined length transfer. Besides, the incrementing burst and wrapping burst are supported. The address of the incrementing burst is just an increment of previous address. However, the address of the wrapping burst will wrap to start address when the boundary is reached. For example, the a start address is 0x40 and WRAP8 is performed, then the address range is 4x8=32=0x20 and the boundary is 0x40+0x20=0x60. Hence, if the address in current single time is 0x5c, the address in next single time will wrap to 0x40.
3. Retry transfer: When the slave is unable to supply data immediately, it can return RETRY response. The RETRY response does not change the master access priority in arbiter. Hence, only the master owning a higher priority can access the bus first. For example, a slave S1 returns RETRY response when a master M1 is reading data from S1. Then, a master M2 owning lower priority still can not access bus if the master M1 has the highest priority in the arbiter.
4. Split transfer: When the slave is unable to supply data immediately, it can also return SPLIT response. The SPLIT response can adjust the master access priority in arbiter. Hence, even the master owning a lower priority can access the bus. For example, a slave S1 returns SPLIT response when a master M1 is reading data from S1. Then, a master M2 owning lower priority can access the bus until the access of slave S1 is available and the master M1 also keep the highest priority in the arbiter.

Table 4. Eight burst types depending on the HBURST signal.

| HBURST | Size | Type |
|---|---|---|
| 000 | Single | Single transfer |
| 001 | Undefined length | Incrementing |
| 010 | 4 beat | Wrapping |
| 011 | 4 beat | Incrementing |
| 100 | 8 beat | Wrapping |
| 101 | 8 beat | Incrementing |
| 110 | 16 beat | Wrapping |
| 111 | 16 beat | Incrementing |

## 3.4 Emulation Platform of Our System

We adopt a platform based design methodology to construct an optimized AVC decoder with a novel scheduling to achieve macroblock-level pipelining [10-11]. The platform-based design methodology has been widely adopted to solve complicated system-level designs of a multimedia system on a single chip. The platform-based design could be defined in following two ways [26-27]. First is reuse of architecture of hardware and software blocks. Second is the construction of a system with stable microprocessor, memory hierarchy, interconnecting bus, and peripherals. The construction has the abilities to cover rapid extension, feasible customization for a wide range of applications, and a short time-to-market. In addition, the platform-based design can improve the yield in circuit design. In short, the platform-based design [26]

has the multiple advantages including efficient system-level design methodology, short time-to-market, reusability of software and hardware IP blocks, feasible chip integration, etc. Thus, the platform-based design methodology is adopted for this project.

Some systems [28-30] are developed on a circuit design level and some systems [31-32] are built on existing hardware platforms. In [28], a videophone system compliant with H.263 is developed where the system is partitioned into hardware modules and software modules based on complexity analysis. Several microcontrollers are used to manage data and functional flow. The dedicated hardware is used to improve computationally intensive parts. In addition, the modules work with macroblock-level pipelining. In [29] the MPEG-2 encoding processes is partitioned into 3 layers including processing control, video processing, and data buffering. A RISC processor controls and pipelines the 3-layer functional modules in macroblock-level. In [30], a multi-core SoC architecture is proposed for MPEG-4 streaming video. Based on the application profile characteristics, the task scheduling of the decoding processes is optimized by a macroblock engine. In addition, the SoC architecture has a global controller using a RISC processor and a computation accelerator with a DSP. On a multi-core platform, all processor cores are communicated via ARM Master Bus Architecture (AMBA). In summary, the platform-based design methodology has been widely adopted to construct the codec of H.263 and MPEG-2/4 standards as discussed [28-32]. Since the complexity of the H.264/MPEG-4 AVC codec is much higher than that of the aforementioned coding standards, it is challenging to build a H.264/MPEG-4 AVC decoder on existing platforms.

Table 5. H.264/MPEG-4 AVC baseline profile and level 1.

| | |
|---|---|
| Max macroblock processing rate | 1485 |
| Max picture size | 176x144 |
| Max bit rate | 64kbit/s |
| Max # of reference frames | 5 |
| Max horizontal and vertical MV range (full pels) | -16~16 |

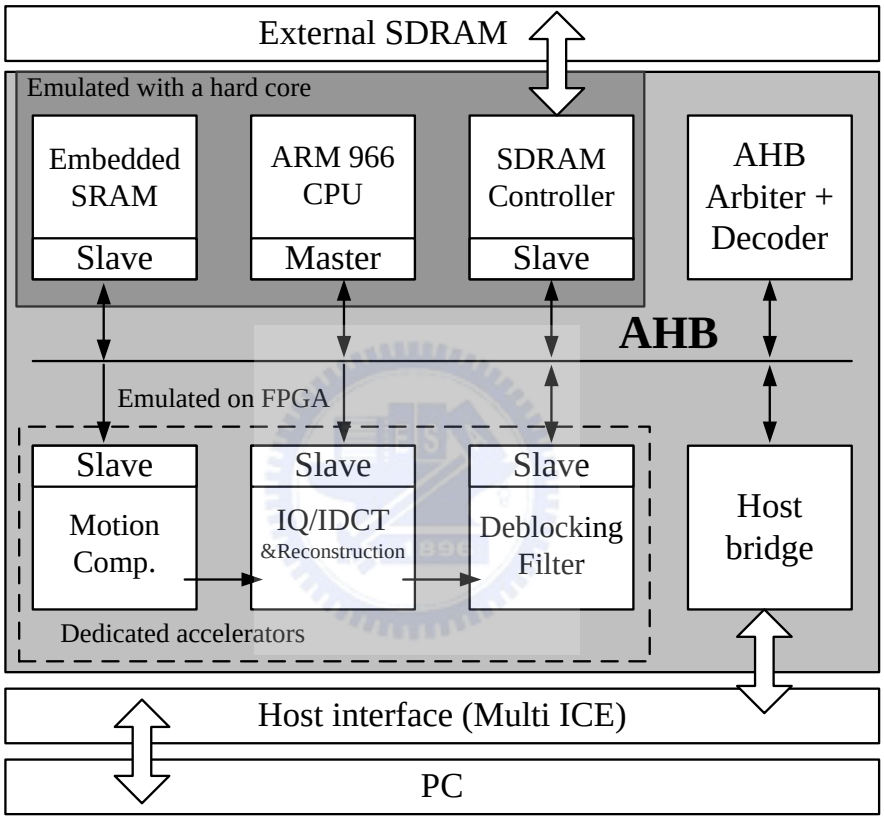



Fig. 26. Proposed ARM-based H.264/MPEG-4 AVC decoder architecture.

Fig. 26 shows our emulation configuration to test our new architecture for the H.264/MPEG-4 AVC decoder. The proposed architecture is compliant with the H.264/MPEG-4 AVC baseline profile of level 1. Table 5 summarizes its main parameters. The architecture is emulated on an ARM development board. In addition, the ARM platform provides a general purpose ARM966E-S CPU core for data flow control and a logic module for multiple dedicated accelerators. The ARM also provides an industry standardized 32-bit AHB for high-speed computation and emulations.

The ARM966 CPU acts as a master on the AHB bus and controls the synchronization among all functional blocks. All the remaining functional blocks that respond to the requests from CPU are slaves. Specifically, the dedicated accelerators are used to speed up computation or reduce memory access. The firmware of the accelerators and the software for the decoding modules are stored in the embedded SRAM. In addition to the embedded memory, our decoder also requires external memory for frame buffering. The external memory is accessed via an external memory interface.

## 3.5 Proposed Macroblock Pipeline Architecture for H.264/MPEG-4 AVC Decoder

.Table 6. Key operations for AVC decoding modules [10].

| Modules | MC | IQ-IDCT | CAVLD | Deblocking Filter |
|---|---|---|---|---|
| Operation | Mul., Add, Shift, MemA | Add, Shift | Branch, MemA | Add, Shift, MemA |

Mul.: multiplication; MemA: memory access

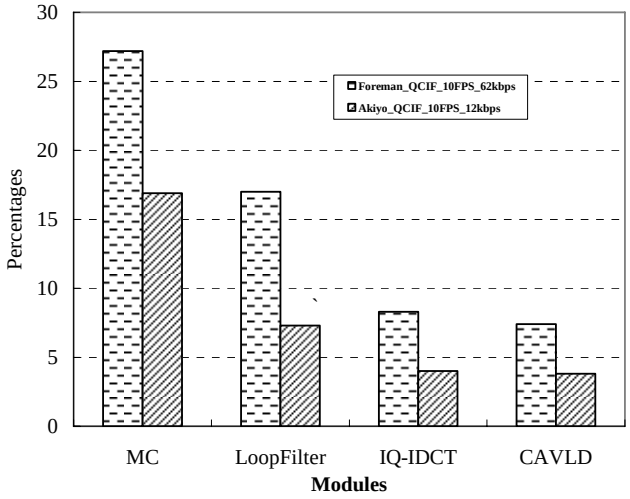



Fig.27. Decoding profiling for H.264/MPEG-4 AVC on ARM 966 CPU [10].

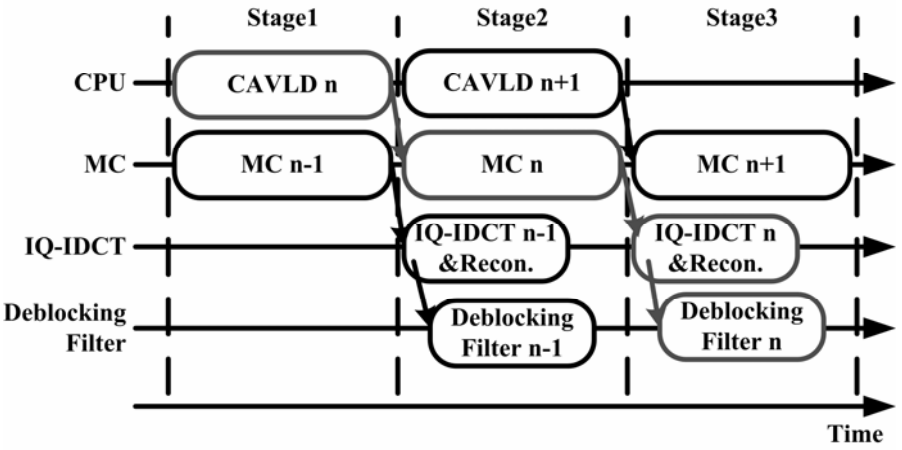



Fig. 28. Scheduling approach for macroblock-level pipelining

In the ARM platform based design, a task can be done with either the software executing on CPU or the dedicated hardware running in parallel with CPU. Thus, to optimize the overall performance through parallel processing, it's challenging to partition the tasks to separately match CPU capability and the dedicated accelerators.

Computational characteristic is a good criterion for task partition [28-30]. The modules with regular and computational intensive tasks are perfect for hardware implementation and the modules with lots of branches are more suitable for software realization. Table 6 illustrates the kernel operations of each AVC decoding module. Most modules except the IQ-IDCT require a great amount of memory access (MemA). In addition, the MC and deblocking filter require intensive arithmetic operations for interpolation and filtering, respectively. The CAVLD uses lots of branching instructions for context-adaptive table switching. Thus, to optimize the performance, the MC, deblocking filter, and IQ-IDCT are implemented in hardware and the CAVLD is realized in software. .

Fig. 27 shows the decoding profiling in relative execution time for different modules. From the Amdahl's law and the observations of Fig. 27, we reduce the computational loads by adding 3 dedicated accelerators for the deblocking filter, MC

and IQ-IDCT, respectively. The remaining coding modules are implemented and optimized in software.

To save the buffers for intermediate data and maximize the throughput, module synchronization is required. In Fig. 28, we design a scheduling approach for macroblock-level pipelining based on data dependency and working load distribution. The subscript ***n*** denotes the macroblock index. Basically, a macroblock is decoded through the three stages including (1) CAVLD, (2) MC, (3) IQ-IDCT and (4) deblocking filtering. With a macroblock pipeline manner, CPU conducts the software for CAVLD and three hardware accelerators conduct functional operation for MC, IQ-IDCT and deblocking filter respectively. Hence, all functional modules can be processed in parallel. Specifically, the scheduling for IQ-IDCT is overlapped with deblocking filter. Hence, we do not have to buffer the IQ-IDCT and reconstruction results and can pass them to deblocking filter immediately. We describe the detail in section 3.6. Fig. 29 shows the flow chart to synchronize CPU with the three accelerators at macroblock-level. At the beginning of each stage,

1. CPU proceeds to decode the CAVLD of (***n***+1)-th macroblock header.
2. CPU sends the data to the IQ-IDCT accelerators for IQ-IDCT of the (***n-1)***-th macroblock. At the same time, the IQ-IDCT accelerator outputs IQIDCT results and performs reconstruction described in Chapter 4.
3. In an adaptive manner described in Chapter 4, CPU sends the data of neighboring macroblocks and IQ-IDCT passes the reconstruction results to the deblocking filter accelerators. At the same time, the deblocking accelerator performs the filtering of the (***n-1)***-th macroblock.
4. CPU receives the filtered data and writes the data to reference memory.

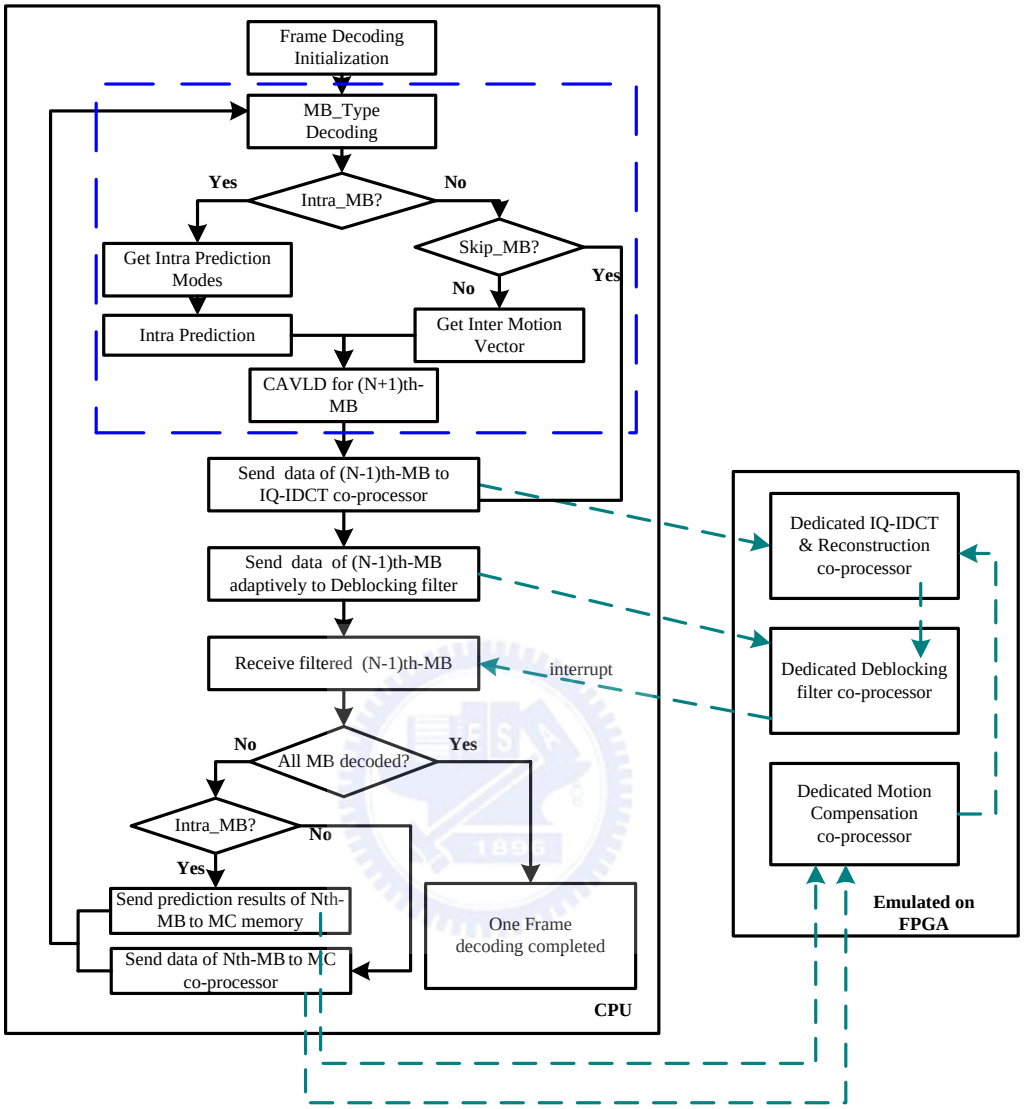



Fig. 29. Flowchart to synchronize CPU with the three accelerators at macroblock (MB)-level.

5. If the ***n***-th macroblock is inter-coded, CPU sends the MC data to the MC accelerator for motion compensation of the ***n***-th macroblock. If the macroblock is intra-coded, CPU sends the results of intra prediction to the MC local memory for the reconstruction of the ***n***-th macroblock.

## 3.6 Non-buffered Memory Architecture for IQ-IDCT and Deblocking filter

Memory architecture usually has great impact on the cost, performance and power consumption. To improve the pipeline latency and memory cost, we propose the non-buffered memory architecture for IQ-IDCT and deblocking filter. Traditionally, to process the macroblock pipelining of IQ-IDCT and deblocking filter, we can use non-shared memory architecture or shared memory architecture [33] described as follows. Fig. 30 shows the non-shared architecture. The IQ-IDCT and deblocking filter have their own single-ported local memory SRAM_1 and SRAM_2 respectively to store macroblock-sized data. To start deblocking the current macroblock, the IQ-IDCT reconstructed results stored in SRAM_1 have to load to SRAM_2 first. During the time of data copy from SRAM_1 to SRAM_2, IQ-IDCT and deblocking filter can not access SRAM_1 and SRAM_2 to perform calculation because of lack of memory bandwidth. Hence, the time of data copy causes the delay time in pipeline schedule and reduces the performance.

Besides, to enhance the throughput, some prior deblocking works propose the architecture of shared memory [17] or dual-ported memory [14],[16],[19]. Fig. 31 represents the shared architecture for IQ-IDCT and deblocking filter modules. By alternatively using the SRAM_1 and SRAM_A, we can load new macroblock data to SRAM_A prepared for next IQ-IDCT calculation while we are operating IQ-IDCT calculation for current macroblock in SRAM_1. The same method can be applied to deblocking filter by alternatively using the SRAM_2 and SRAM_B. Hence, the latency of data copy can be removed. However, some problems of shared architecture have to be considered:

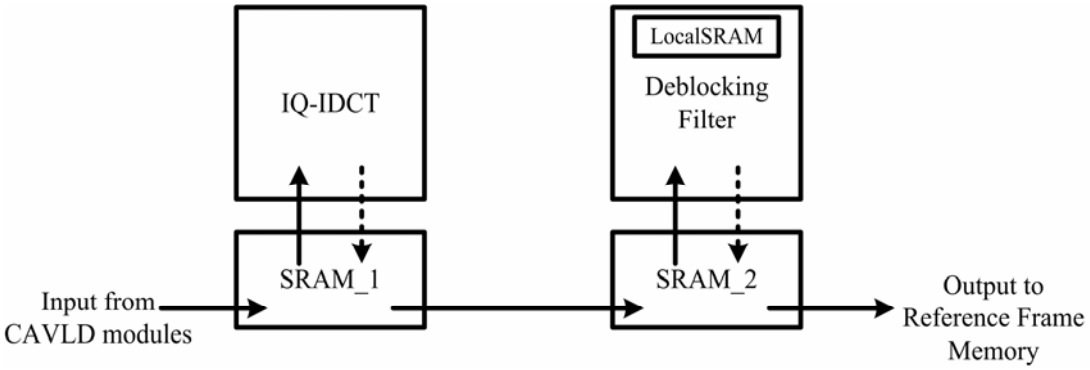



Fig. 30. Architecture of non-shared memory design

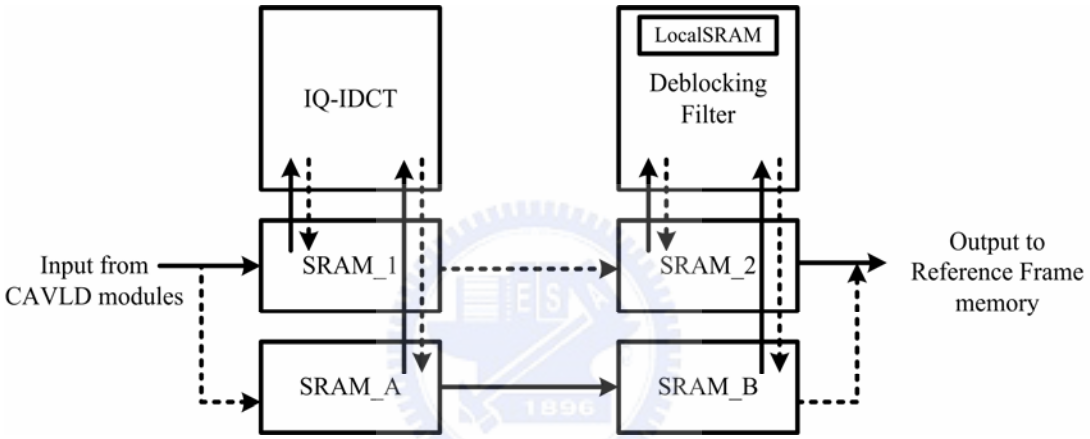



Fig. 31. Architecture of shared memory design.

1. Shared architecture is not suitable for hardware and software co-operation. Take our system design as example, we realize the CAVLD by software. Hence, the shared architecture can not be applied to the transaction of CAVLD and IQ-IDCT because the CPU can not load CAVLD results to IQ-IDCT and calculate CAVLD for next macroblock in parallel. Once the shared architecture can not be applied to the transaction of CAVLD and IQ-IDCT, it can not be applied to the transaction of IQ-IDCT and deblocking filter either.
2. Shared architecture requires double size of memory buffer that in general increase the overall cost significantly. If the IQ-IDCT and deblocking filter

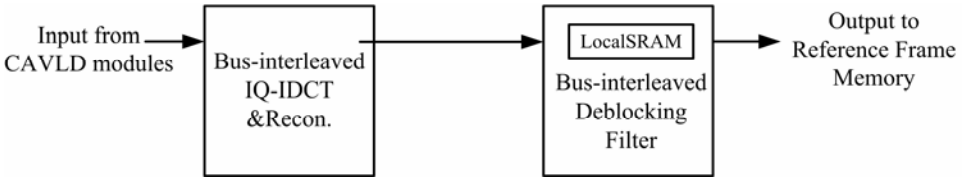



Fig. 32. Architecture of non-buffered design.

modules are integrated in the compensation loop of the encoder, the designer may not prefer to increase the cost because the critical pipeline latency is made by motion estimation.

3. The frequent memory access in shared architecture costs more power consumption than the one in non-shared architecture.

Therefore, to improve pipeline throughput, we propose a non-buffered architecture for IQ-IDCT and deblocking filter modules. Fig. 32 shows the non-buffered architecture for IQ-IDCT and deblocking filter. It mainly includes a bus-interleaved deblocking filter accelerator and a bus interleaved IQ-IDCT accelerator described in Chapter 4. With the bus-interleaved scheme, the data transfer and functional operation can be performed in parallel. Hence, our IQ-IDCT accelerator can pass the reconstruction results to deblocking filter while it performs transforming for current macroblock. Also, deblocking accelerator can perform deblocking operation while it is receiving the reconstruction results. Hence, the macroblock-size memory for buffering reconstruction results can be removed.

As compared to the non-shared and shared memory architecture, Fig. 33 shows the pipeline schedule comparison. Basically, the proposed non-buffered architecture has some advantages:

1. It has lower memory cost. Our non-buffered architecture can achieve the same performance as shared memory architecture but reduce significant memory cost. By utilizing the non-buffered architecture, the intermediate memories

(SRAM_1, SRAM_2, SRAM_A and mode.SRAM_B shown in Fig. 30 and Fig. 31) can be removed.

2. It has better latency than non-shared memory architecture. We can overlap the IQ-IDCT and reconstruction with the deblocking process. Also, the bus-interleaved design of IQ-IDCT and deblocking filter overlap the time of data transfer and the time of hardware calculation. Therefore, the pipeline latency can be reduced.
3. It cost lower power consumption. Non-buffered architecture has less memory access frequency due to the less intermediate memories are used. Hence, the power consumption of memory access is lower.

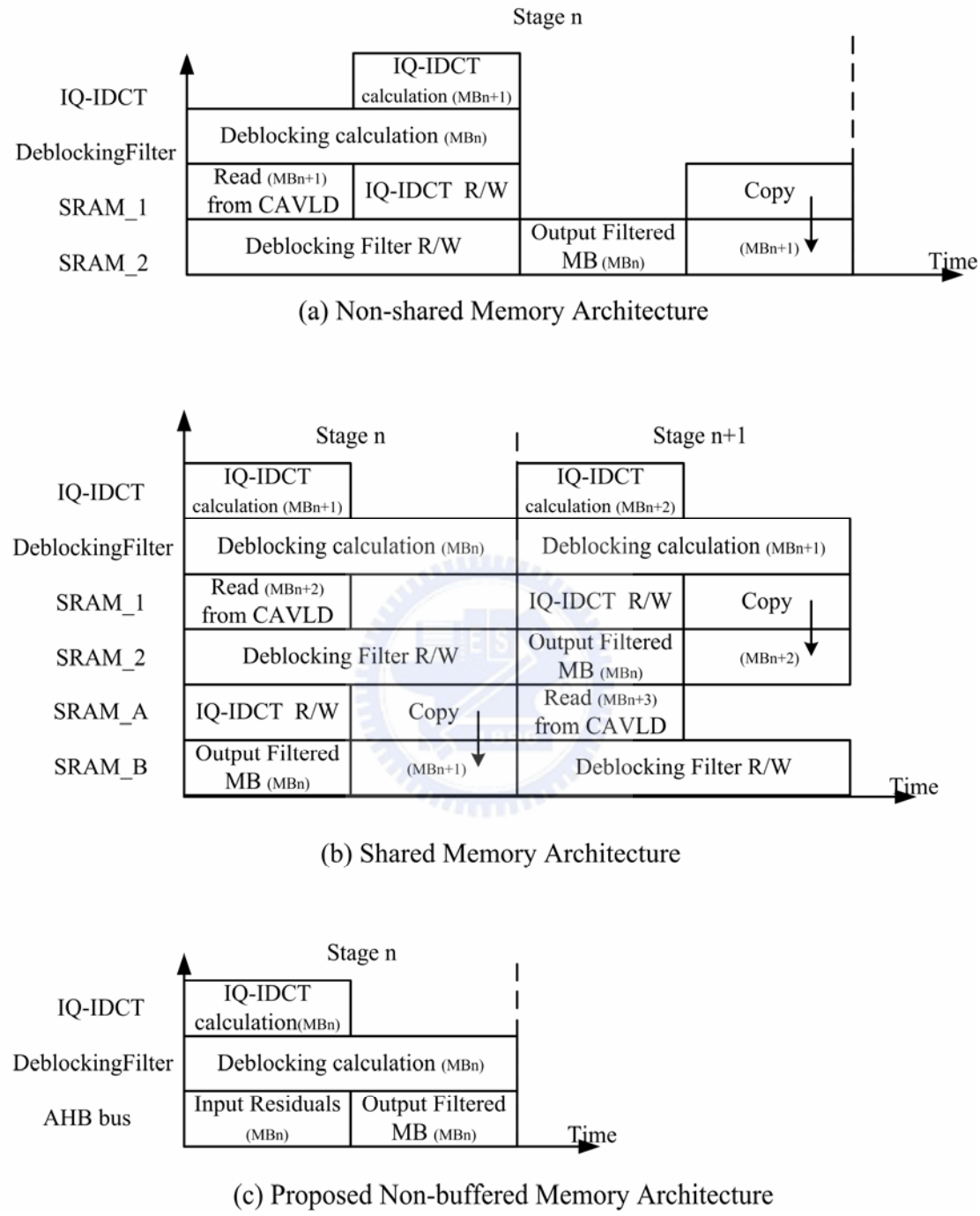



Fig. 33. Pipeline schedule comparison

# Chapter 4

# Hardware Accelerators for H.264/MPEG-4 AVC Decoder

## 4.1. Introduction

We have learned the macroblock pipelining architecture for H.264/MPEG-4 AVC decoder in Chapter 3. In the pipeline schedule, four functional modules of CAVLD, MC, IQ-IDCT and deblocking filter perform the decoding process in a macroblock by macroblock manner. Based on software/hardware partition methodology, we realize three hardware accelerators of MC, IQ-IDCT and deblocking filter and implement CAVLD and other modules by software. In this chapter, we describe the architecture designs for these accelerators.

## 4.2. Bus-interleaved Deblocking Filter

### 4.2.1 Overview of State-of-art Works

Among various coding tools in H.264, the in-loop deblocking filtering has significant impact on the visual quality improvement. To reduce the blocking artifact, the in-loop deblocking filter adaptively conducts the filtering along the boundaries of each 4x4 block according to the boundary strength (bS), the quantization parameter (Qp) and the content of the block. The blocking artifact is removed. However, the improvement is at the cost of intensive computation and memory access.

For real-time applications, the deblocking filtering becomes one of the performance bottlenecks. In [12]-[20], dedicated hardware was developed for acceleration. Table 7 shows the main features of prior works for H.264/MPEG-4 AVC deblocking filter. Specifically, the architecture of [18] is for frame-based filtering. The deblocking filtering is invoked after the reconstruction of the entire frame. Apparently, frame-based filtering requires a frame buffer and longer system latency. To reduce the buffer size and latency, macroblock-based (MB-based) filtering architectures were proposed in [12]-[17],[19],[20]. The filtering can be started upon the reconstruction of a macroblock. To achieve high throughput, in [14],[16],[18]-[20], dual-ported SRAM is used to simultaneously conduct the reading and writing during the filtering. Also, [17],[20] utilize frame-length memory to buffer data of neighboring macroblock. However, the high throughput is at the cost of complex and costly memory architecture. In addition, for filtering a macroblock, [14],[16],[19] need to first buffer the entire macroblock. The hardware is idled for waiting the data. Moreover, the data movement of [14]-[20] is not mode aware which means that the data transmission overhead is not minimized. Hence, in this thesis, we propose a parallel processing architecture and a more efficient data transmission scheme to improve the performance.

### 4.2.2 Proposed Adaptive Transfer Scheme

In this thesis, our deblocking filtering is designed to operate at macroblock level. The entire frame is filtered in a macroblock-by- macroblock manner and the macroblocks within a frame are processed in a raster scanning order. The filtering can be started upon the reconstruction of a macroblock. For filtering a macroblock, we need to first retrieve the reconstructed data from the embedded memory (or certain module) and transfer the data to the dedicated accelerator via a bus. As more and more dedicated accelerators are deployed, the limited and shared bus bandwidth could become the performance bottleneck. To reduce the demand of bus bandwidth, we propose an adaptive macroblock transmission scheme.

Table 7: Main features of prior works for H.264/MPEG-4 AVC deblocking filter

| | [14], [16] | [15] | [17] | [18] | [19] | [20] |
|---|---|---|---|---|---|---|
| Average macroblock Latency (cycles) | 614 | 386 | 250 | >600 | 510 | 286 |
| SRAM Memory Architecture | 2x Dual-port | 1x Single-port | 3x Single-port | 1x Dual-port | 2x Dual-port 1x Single-port | 1x Dual-port 1x Single-port |
| Local Memory Size (bits) | 96x32 + 64x32 | 80x32 | 96x32x2 + 2xFrame Widthx32 | Frame size | Dual: 88x32 +72x32 Single: 32x32 | Dual: 64x32 Single: 2xFrame Widthx32 |
| Processing Throughput (1280x720, 100Mhz) | 45.2fps | 71.9fps | 111.1fps | N/A | 54.5fps | 97.1fps |
| Gate Count (UMC 0.18um) | 20.6K | 9.2K | 19.6K | N/A | N/A | 14.5K |

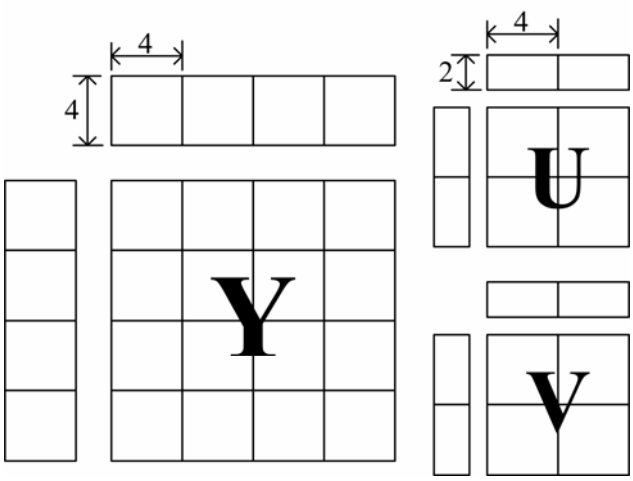



Fig. 34. macroblock data and its adjacent blocks used for macroblock-based deblocking filtering.

*A. MB Mode Classification*

Fig. 34 depicts the data required for filtering a macroblock. As shown, in addition to the current macroblock, the adjacent 4x4 blocks at the right and the left boundaries are also needed. In [14]-[20], fixed macroblock data are transferred to the accelerator. However, we find that not all the 4x4 blocks within a macroblock are to be filtered. Thus, we can more efficiently use the bus bandwidth by minimizing the redundant data transfers. To do so, we define 8 macroblock filtering modes according to the filtering necessity of the left macroblock boundary, the top macroblock boundary and the current macroblock. Table 8 and Table 9 summarize the corresponding data size transfer macroblock data of each mode. For example, mode 5 denotes the case in which only the left and the top macroblock boundaries are required for filtering. As a result, for the luminance part, we simply need the adjacent 4 blocks in the left macroblock, the adjacent 4 blocks in the top macroblock and the adjacent 7 blocks in the current macroblock. By the same token, one can derive the data size for the chrominance part. Totally, the data transfer size required for filtering a mode 5 macroblock is 116 words which include 60 words for the luminance component and 56 words for the chrominance part. Following the same principle, one can derive the data size for the other modes. By distinguishing different filtering modes, we can

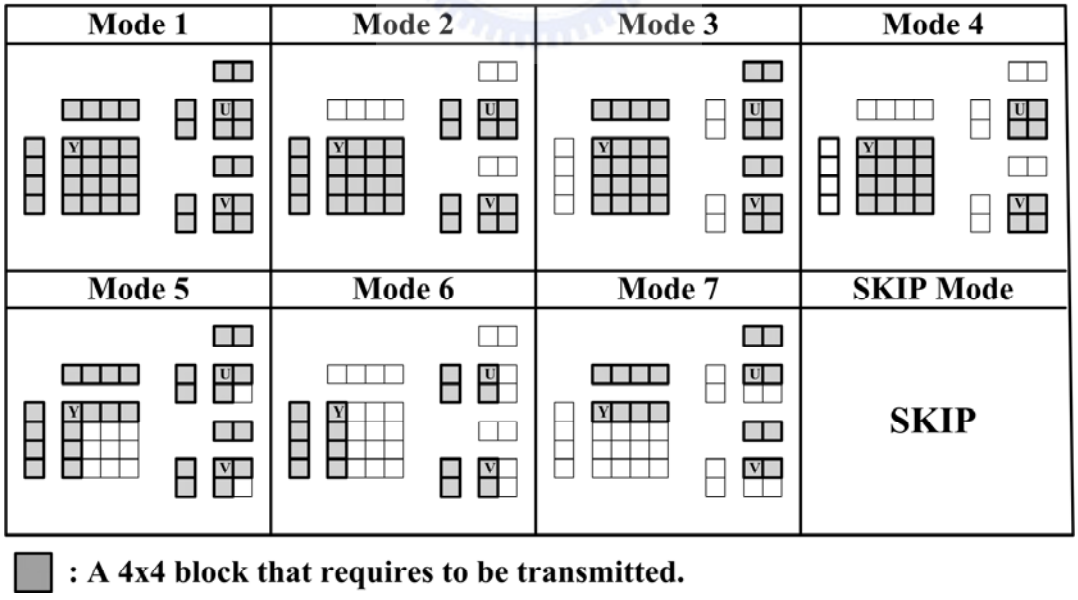
Table 8. Filtering mode for a macroblock

| Mode | Left* | Top* | Current macroblock | Data Size** |
|---|---|---|---|---|
| 1 | Y | Y | Y | 160 |
| 2 | N | Y | Y | 128 |
| 3 | Y | N | Y | 128 |
| 4 | N | N | Y | 96 |
| 5 | Y | Y | N | 116 |
| 6 | N | Y | N | 64 |
| 7 | Y | N | N | 64 |
| Skip | N | N | N | 0 |

*: The macroblock boundary required for filtering.

**: Data transfer size in words.

Table 9. Transfer macroblock data for the 8 filtering modes



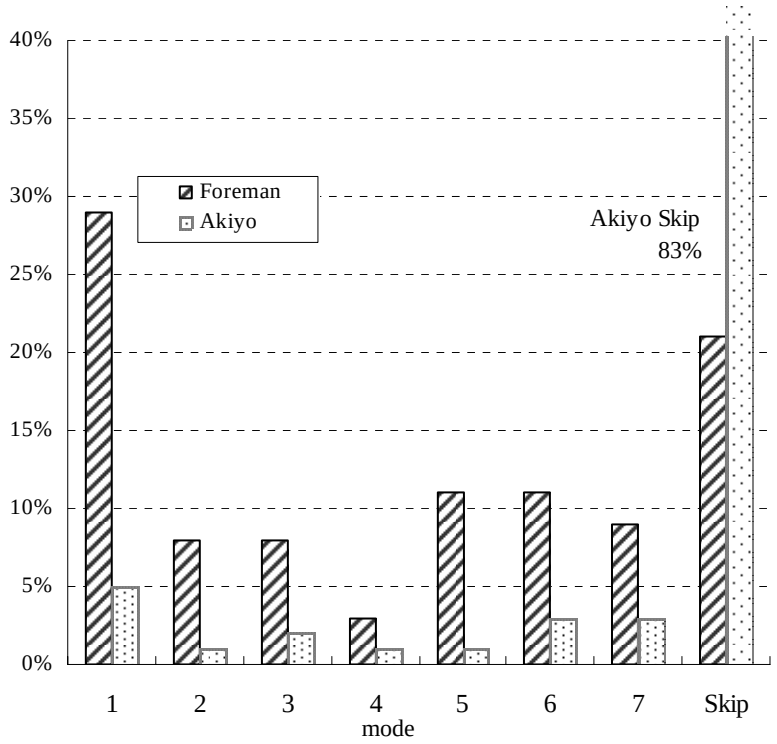



Fig. 35. macroblock filtering mode distribution in Akiyo and Foreman sequences that are coded at QCIF@15fps 64Kbps with JM6.0.

minimize the redundant data transfers.

*B. Macroblock Filtering Mode Distribution*

Fig. 35 shows the mode distribution of Akiyo and Foreman sequences based on JM6.0. Without mode classification, [14]-[16],[18],[19] treat all macroblocks as mode 1, i.e., all the input samples shown in Fig. 34 are to be transferred. Particularly, [17],[20] treats all macroblocks as mode 4 because they previously buffer the left macroblock and one row of top macroblocks. From Fig. 35, we learn that different modes have different weightings. Specifically, mode 1 is actually less than 30% and mode 4 is less than 5%. In the extreme case of Akiyo, most macroblocks use skip mode which does not require any input samples. Thus, [14]-[20] incur many redundant data transfers. With the filtering mode classification, we can more efficiently use the bus bandwidth. According to our mode analysis, in Akiyo sequence, we can save 94% of the data transfers used in [18], 89% of those in [14]-[16],[19] and 85% of those in [17],[20]. Similarly, in Foreman sequence, our design can save 70% of the data transfers in [18],

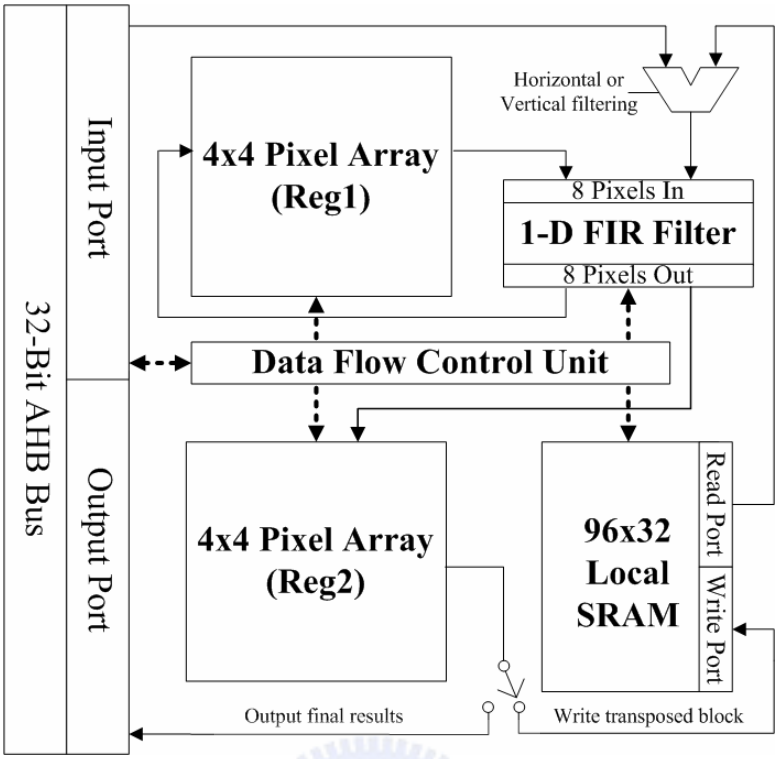



Fig. 36. Proposed bus-interleaved architecture.

41% of those in [14]-[16],[19], and 25% of those in [17],[20]. As compared to [14]-[19], significant data transfer reduction is achieved.

In addition, as compared to [17],[20], our design shows benefit with much less memory usage. The detail comparison will be shown in Section 4.2.4.

## 4.2.3. Proposed Bus-interleaved Architecture

To reduce the processing latency, we propose a bus-interleaved architecture in [12],[13]. Specifically, we perform the filtering and the data transfer in parallel. Different from the prior designs, [14],[16]-[19], we can start the filtering while the data is being streamed in and out. The processing latency is reduced due to the parallelism.

*A. Proposed Bus-interleaved Architecture*

Fig. 36 shows our proposed architecture. It mainly includes four components:

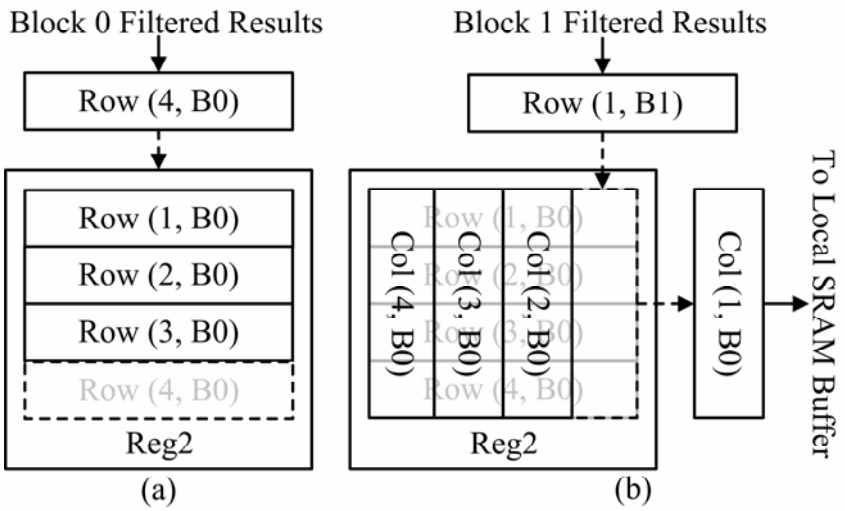



Fig. 37. Operation of the transposed memory (Reg2).

**1. One-dimensional Adaptive FIR Filter**

The one-dimensional FIR filter adaptively performs the horizontal/vertical filtering in a row-by-row manner. For each row, it takes 8 input samples from two adjacent 4x4 blocks to conduct filtering. Accordingly, it produces 4 filtered results and 4 intermediate results for the filtering of next block.

**2. Single-ported SRAM**

A single-ported SRAM is used as local memory for buffering the horizontally filtered and transposed macroblock. Specifically, for the luminance component, it stores all the 4x4 blocks in the current macroblock (i.e., 64x32 bits) and the adjacent 4x4 blocks in the top and the left macroblocks (,i.e., 32x32 bits). The total size of the SRAM is 96x32 bits. In our design, the filtering of chrominance and luminance components shares the same memory.

**3. 4x4 Pixel Arrays (Reg1 and Reg2)**

In Fig. 36, Reg1 buffers the intermediate results produced by the FIR filter. On the other hand, Reg2 acts as a transposed memory. Particularly, Reg2 performs the transposition by storing the data in either Horizontal-In-Vertical-Out or Vertical-In-Horizontal-Out fashion. Fig. 37 shows an example of the transposition

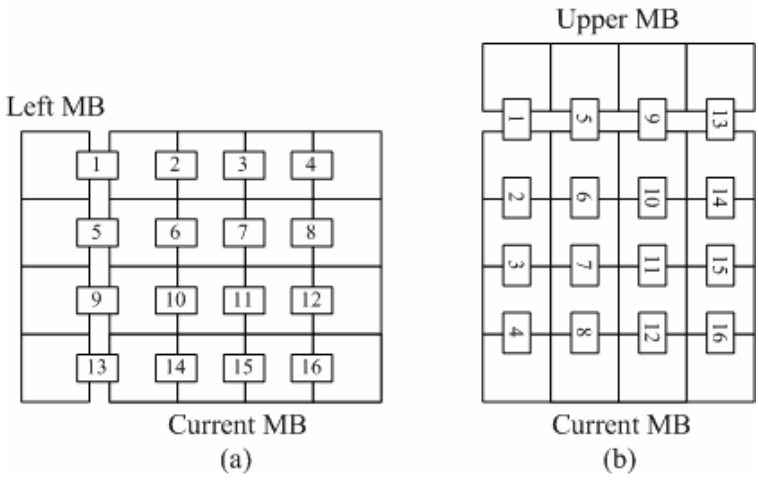



Fig. 38. Sequential edge processing order of (a) horizontal filtering and (b) vertical filtering in a luminance macroblock.

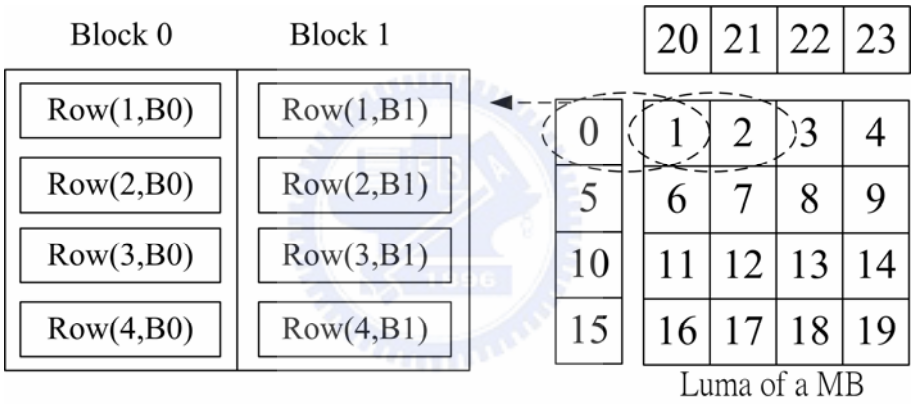



Fig. 39. The 4x4 block input order to the dedicated hardware.

where Row(n, Bm) represents the n-th row of m-th block and Col(n, Bm) denotes the n-th column of m-th block. Specifically, Fig. 37 (a) depicts the case as the horizontally filtered Block 0 is being written to Reg2 in a row-by-row manner. After Block 0 is completely buffered in Reg2, Fig. 37 (b) illustrates that the transposition is done by writing Block 0 to the SRAM in a column-by-column manner. Particularly, after Col(1, B0) is stored in the SRAM, we filled out the left space in Reg2 with Row(1, B1), i.e., the first row of next horizontally filtered block. Such replacement is continued until horizontally filtered Block 1 is

completely buffered in Reg2. Since Block 1 is stored column-by-column in Reg2, we transpose Block 1 by outputting the data row-by-row to the SRAM. Such cyclical rotation between Horizontal-In-Vertical-Out and Vertical-In-Horizontal-Out is conducted throughout the entire deblocking process. Traditional designs [14],[16],[18] require stalls for block transposition. However, our seamless design requires no stalls.

**4. Data Flow Control Unit**

The data flow control unit consists of a finite state machine which controls synchronization among 1-D FIR filter, 4x4 pixel arrays and local SRAM buffer. Moreover, it responses to the deblocking filtering request from the AHB bus.

*B. Operation of Bus-interleaved Architecture*

To describe the operation of our bus-interleaved architecture, we use the filtering of a mode 1 macroblock as an example. Fig. 38 shows the processing order of horizontal and vertical filtering for a mode 1 macroblock and Fig. 39 depicts the sequential block input order to the dedicated accelerator. In Fig. 40, we show the status of our bus-interleaved architecture during the horizontal filtering. Here, we assume Reg1 has buffered the non-filtered samples of Block 0. To perform the horizontal filtering for the edge between Block 0 and Block 1 in Fig. 39, the FIR filter takes Row(1,B1) from the bus and Row(1,B0) from the 1$^{st}$ row of Reg1 for computation. After the filtering, we overwrite the 1$^{st}$ row of Reg1 with the intermediate results, Row(1,B1)_intermediate, and save the horizontally filtered results, Row(1,B0)_h, in the 1$^{st}$ row of Reg2. The other rows are processed in the same way. When the horizontally filtered Block 0 is completely stored in the Reg2, we transpose the block by writing it to the SRAM in a column-by-column fashion. While the SRAM is being written, the FIR filter performs the horizontal filtering for the edge between Block 1 and Block 2 by receiving Row(n,B2) from bus and retrieving Row(n,B1)_intermediate from Reg1. Such process is continued until the horizontal filtering of a macroblock is done. After the horizontal filtering, we read the

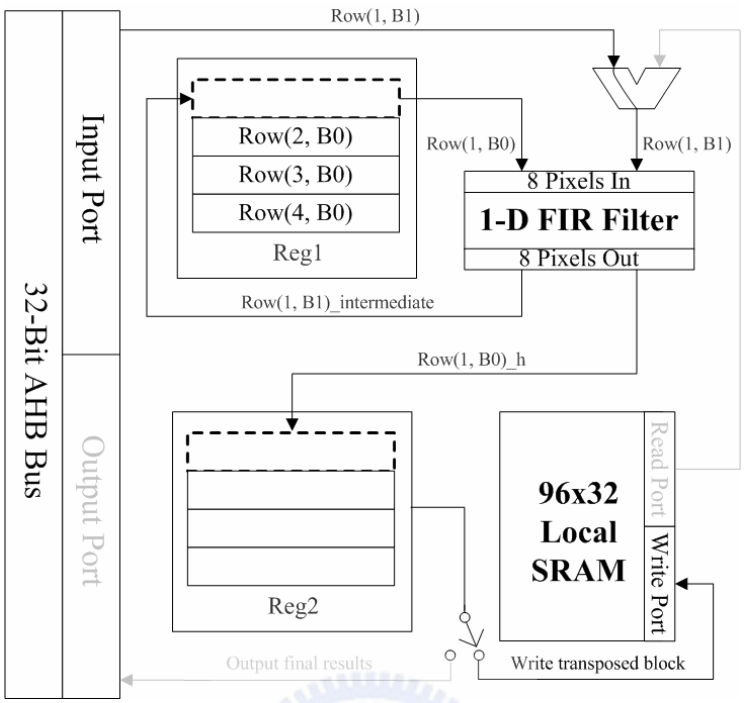



Fig. 40. Data flow of horizontal filtering in the bus-interleaved architecture.

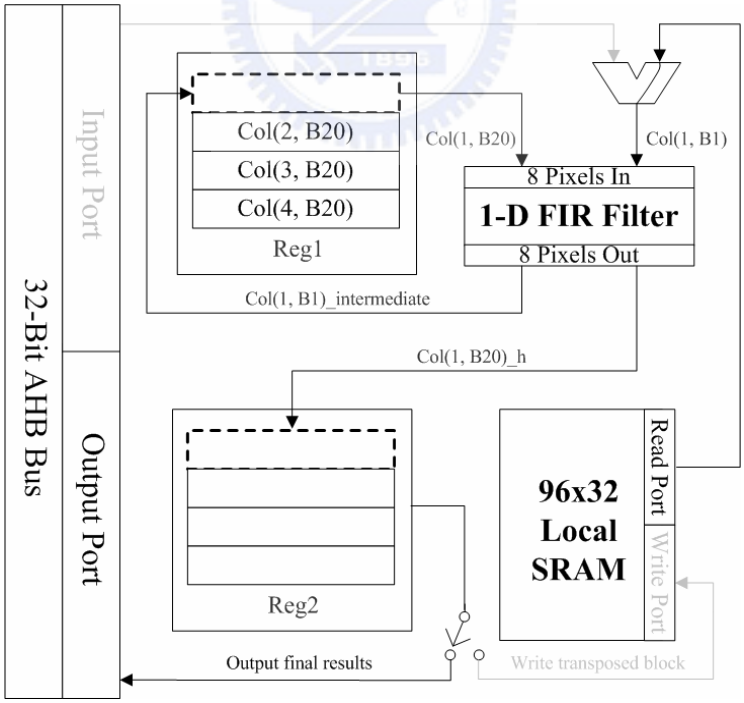
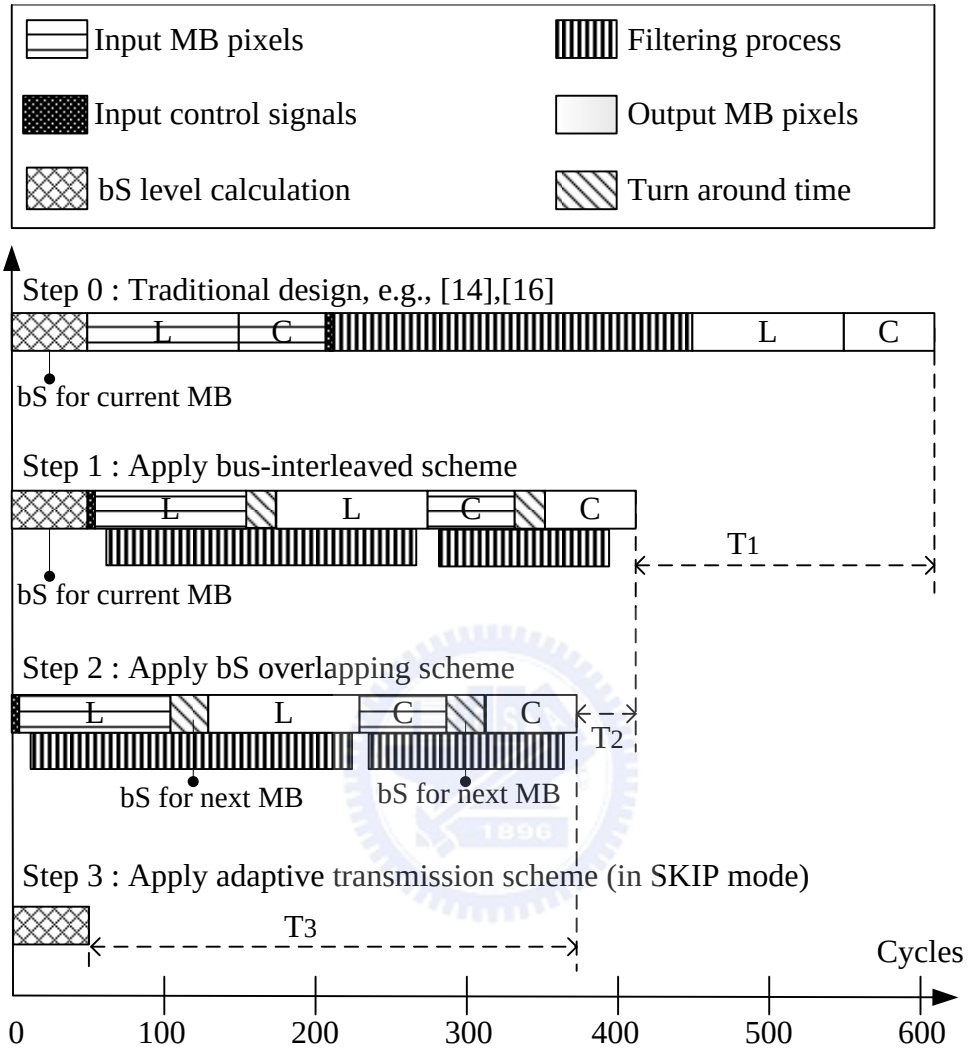



Fig. 41. Data flow of vertical filtering in the bus-interleaved architecture.




T1: Reduced latency from bus-Interleaved scheme
T2: Reduced latency from overlapping of bS calculation
T3: Reduced latency from adaptive transmission scheme
L: Luminance component
C: Chrominance component

Fig. 42. Analysis of processing latency reduction

horizontally filtered macroblock from the SRAM and perform the vertical filtering in the same manner. Specifically, during the vertical filtering, the input data of FIR filter is now configured to be from the SRAM. In addition, the filtered and transposed data is written to the CPU instead of local SRAM. Fig. 41 shows the configuration of our

Table 10. Macroblock latency for each transfer mode

| Transmission Mode | Latency Per macroblock (cycles) |
|---|---|
| 1 | 374 |
| 2 | 310 |
| 3 | 310 |
| 4 | 246 |
| 5 | 286 |
| 6 | 182 |
| 7 | 182 |
| SKIP | 50 |

bus-interleaved architecture during the vertical filtering. Note that the luminance and the chrominance components are sequentially processed in the same manner.

*C. Overlapping of bS Level Calculation*

In our design, the bS level calculation is done by hardware. Particularly, to reduce the macroblock processing latency, the bS level for current macroblock is calculated in the previous macroblock cycle so that the data dependency between the bS level and the filtering can be removed. Moreover, we overlap the computations of filtering and bS level calculation. Note that there is a turn-around time between the last input data and the first filtered output result. During the turn-around time, the bus is idled. Thus, we use this turn-around time to transmit the data required for bS level calculation and conduct the actual computation.

*D. Processing Latency Analysis*

Fig. 42 illustrates how our proposed schemes can reduce the processing latency for filtering a macroblock. We show the improvement of each proposed scheme step by step. For comparison, Step 0 shows the processing latency of traditional design, e.g., [14], which does not deploy bus-interleave architecture and macroblock adaptive transmission scheme. As shown in Step 1, our bus-interleaved architecture offers 1.5x performance improvement over the design of [14] due to the parallelism of data

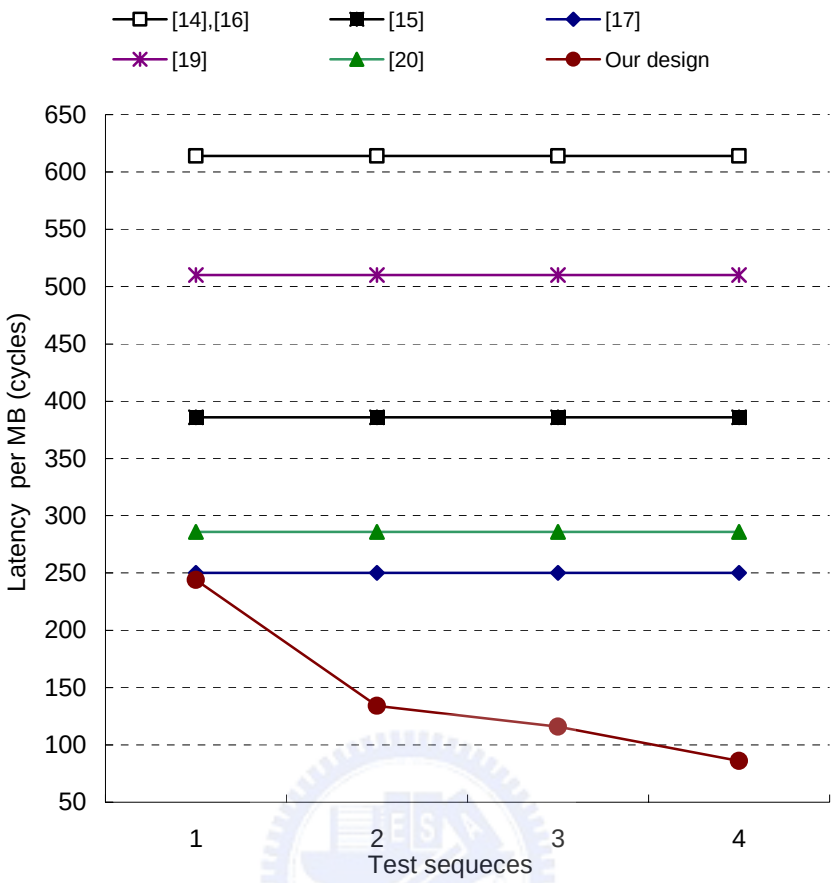



Fig. 43. Comparison of average latency per macroblock (including 50 cycles for bS

transfer and filtering. Moreover, with the overlapping of bS level calculation, Step 2 shows that the processing latency can be further reduced. Furthermore, Step 3 shows that our adaptive transmission scheme can reduce the processing latency to be merely 50 cycles when the skip mode is detected. In the skip mode, there is no need to conduct data transfer. By the adaptive transmission scheme, our design can detect skip mode and avoid the redundant data transfers. However, without mode aware, traditional design [14]-[20] incur many redundant data transfers even in the skip mode. Table 10 lists the cycle counts for the other macroblock modes. According to our mode analysis in Fig. 35, our design averagely requires 86 to 244 cycles for filtering a macroblock. As compared to [14]-[20], Fig. 43 shows that our design has up to 7.1x performance improvement. Significant latency improvement is achieved.

### 4.2.4 Comparisons of Simulation Results

In this Section, we show the comparisons of different hardware designs. Moreover, we analyze the memory access frequency in different approaches. Lastly, we use an ARM based H.264 decoder as an example to demonstrate the system performance of our design

*A. Comparison of Hardware Implementation*

Table 11 compares our accelerator with the state-of-the-art designs [14]-[20]. As shown, for filtering a macroblock, our design averagely requires less cycle counts. Specifically, as compared to [14],[16],[18]-[20], we provide 1.2x to 7.1x performance improvement with simpler and smaller single-ported memory. In addition, we have up to 4.5x performance improvement as compared to [15],[17]. While clocking at 100MHz, our design can support 2560x1280@30Hz processing throughput. Additionally, our bus bandwidth requirement
is down to 6%-30% of [18], 11%-59% of [14]-[16],[19] and 15%-75% of [17],[20].

*B. Comparison of Memory Access Frequency*

Table 12 further compares the local SRAM access frequency of different approaches. For filtering a macroblock, [14], [16]-[19] require read and write operation to previously buffer the input macroblock. In addition, [17], [20] also need to buffer one row of top macroblocks and the left macroblock. During the horizontal and vertical filtering, [14],[16],[18],[19] require more frequent read and write operation. Particularly, for [17], they use additional 4x4 pixel arrays to buffer the horizontally filtered and transposed results instead of using local memory. As compared to the prior works, our design simply needs one write operation for horizontal filtering and one read operation for vertical filtering. There is no need to previously buffer the input macroblock. Significant memory access reduction is achieved. Less frequent memory access and simpler memory architecture bring the advantages of lower power consumption and lower cost.

Table 11. Comparisons of state-of-the-art deblocking filter designs.

| | **[14], [16]** | **[15]** | **[17]** | **[18]** |
|---|---|---|---|---|
| Average macroblock Latency (cycles) | 614 | 386 | 250 | >600 |
| SRAM Memory Architecture | 2x Dual-port | 1x Single-port | 3x Single-port | 1x Dual-port |
| Local Memory Size (bits) | 96x32 + 64x32 | 80x32 | 96x32x2 + Frame Widthx2x32 | Frame size |
| Number of 4x4 Pixel Arrays | 2 | 2 | 4 | 4 |
| Number of 1-D Filter | 1 | 1 | 1 | 1 |
| Bandwidth Requirement (Normalized with respect to [18]) | 50% | 50% | 40% | 100% |
| Processing Throughput (1280x720, 100Mhz) | 45.2fps | 71.9fps | 111.1fps | N/A |
| Gate Count (UMC 0.18um) | 20.6K | 9.2K | 19.6K | N/A |

Table 11(continue). Comparisons of state-of-the-art deblocking filter designs.

| **(Continue)** | **[19]** | **[20]** | **Our design** |
|---|---|---|---|
| Average macroblock Latency (cycles) | 510 | 286 | 86 – 244 |
| SRAM Memory Architecture | 2xDual-port 1xSingle-port | 1xDual-port 1x Single-port | 1xSingle-port |
| Local Memory Size (bits) | Dual: 88x32 + 72x32 Single: 32x32 | Dual: 64x32 Single: Frame Widthx2x32 | 96x32 |
| Number of 4x4 Pixel Arrays | 11 (3 pixel array +8 FIFO) | 6 (2 pixel array +4 FIFO) | 2 |
| Number of 1-D Filter | 2 | 1 | 1 |
| Bandwidth Requirement (Normalized with respect to [18]) | 50% | 40% | 6% – 30% |
| Processing Throughput (1280x720, 100Mhz) | 54.5fps | 97.1fps | 113.8 –322.9fps |
| Gate Count (UMC 0.18um) | N/A | 14.5K | 11.8K |

Table 12. Comparisons of local memory access frequency

| | [14],[16], [18], [19] | [17] | [20] | Our design,[15] |
|---|---|---|---|---|
| Current macroblock Buffering | Read/Write | Read/Write | None | None |
| Left and Top macroblocks Buffering | None | Read/Write | Read/Write | None |
| Horizontal Filtering | Read/Write | None | Write | Write |
| Vertical Filtering | Read/Write | None | Read | Read |

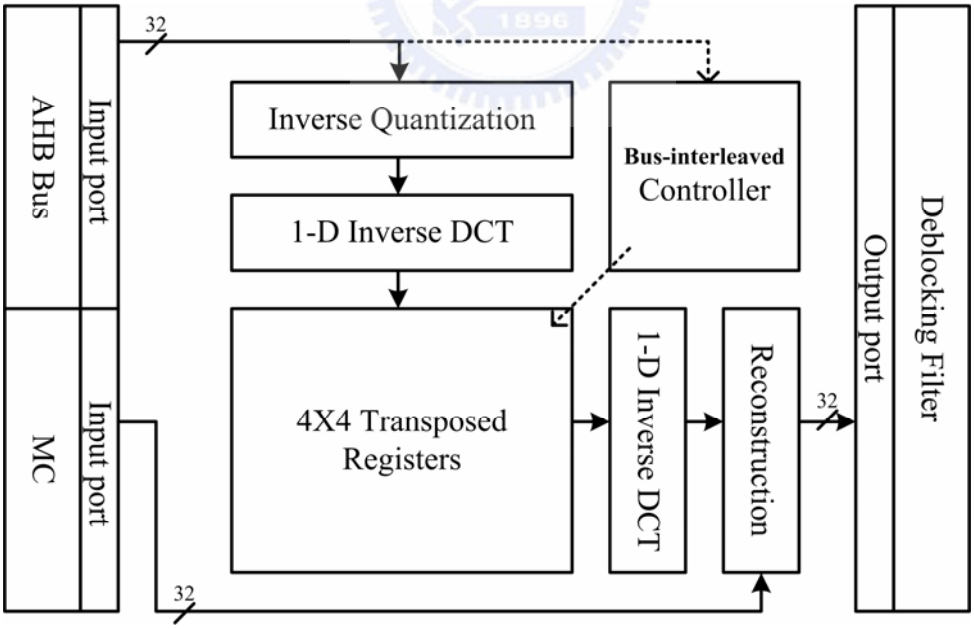



Fig. 44. Architecture of bus-interleaved IQ-IDCT

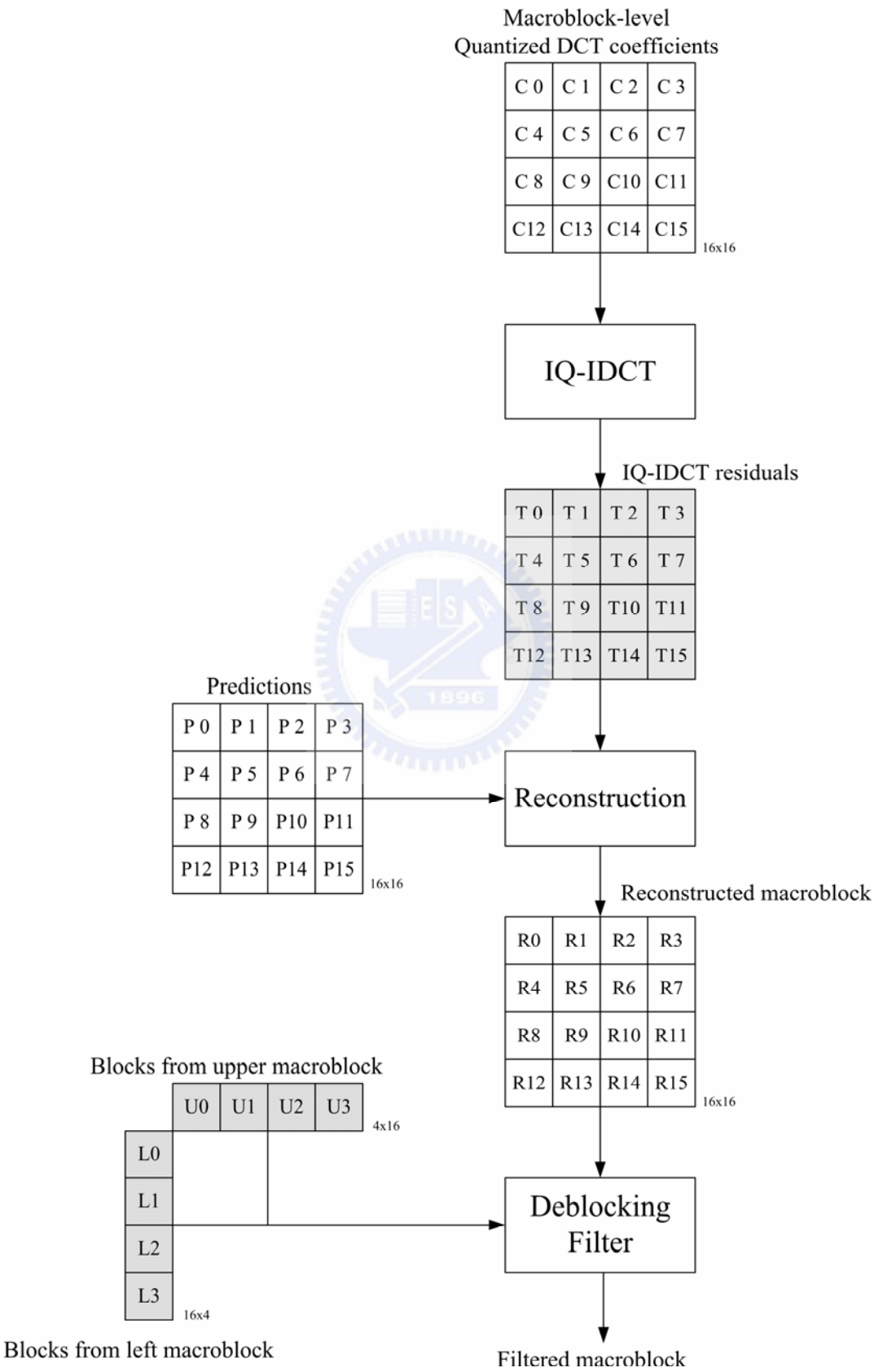
# 4.3. Accelerator for IQ-IDCT

## 4.3.1. Bus-interleaved IQ-IDCT

Prior work [34] proposed a parallel architecture for DCT in H.264/MPEG-4 AVC. This parallel architecture uses two sets of 1-domension DCT module to parallel process 2-dimension DCT and achieve 100% hardware utilization. To realize the IQ-IDCT with bus-interleaved architecture, we take advantage of the parallel architecture. Fig 44 shows the bus-interleaved architecture of IQ-IDCT. It mainly includes a 4x4 transposed buffer and two sets of one-dimension inverse DCT, an embedded AHB bus, a bus-interleaved controller and an inverse quantization module. To describe the processing flow of IQIDCT, the quantized DTC data are inputted from bus row by row at first. At the same time, we can perform inverse quantization and transform by using the inverse quantization module and the first 1-dimension inverse transform. The results are stored on 4x4 transposed registers in 4 cycles. Next, the current block on transposed buffer can perform 1-dimension inverse DCT column by column by using the second inverse transform module to complete 2-dimension inverse transform. To maintain the 100% utilization of transform modules, the first 1-D inverse transform module can process the inverse transform of the next block while the second inverse transform module is processing the inverse transform of the current block. Besides, the bus-interleaved controller controls the reading and writing of transposed registers based on the bus address. Hence, the data transfer and transform can be performed in parallel without mismatches. In our simulation, our IQ-IDCTdesign takes 104 cycles to transform one macroblock. The area used is 6680 gates.

## 4.3.2. Interleaved process for IQ-IDCT and deblocking filter

To describe the interleaved process for IQ-IDCT and deblocking filter, Fig. 45


Macroblock-level
Quantized DCT coefficients
C 0 C 1 C 2 C 3
C 4 C 5 C 6 C 7
C 8 C 9 C10 C11
C12 C13 C14 C15
16x16
IQ-IDCT
IQ-IDCT residuals
T 0 T 1 T 2 T 3
T 4 T 5 T 6 T 7
T 8 T 9 T10 T11
T12 T13 T14 T15
Predictions
P 0 P 1 P 2 P 3
P 4 P 5 P 6 P 7
P 8 P 9 P10 P11
P12 P13 P14 P15
16x16
Reconstruction
Reconstructed macroblock
R0 R1 R2 R3
R4 R5 R6 R7
R8 R9 R10 R11
R12 R13 R14 R15
16x16
Blocks from upper macroblock
U0 U1 U2 U3
4x16
L0
L1
L2
L3
16x4
Deblocking Filter
Blocks from left macroblock
Filtered macroblock

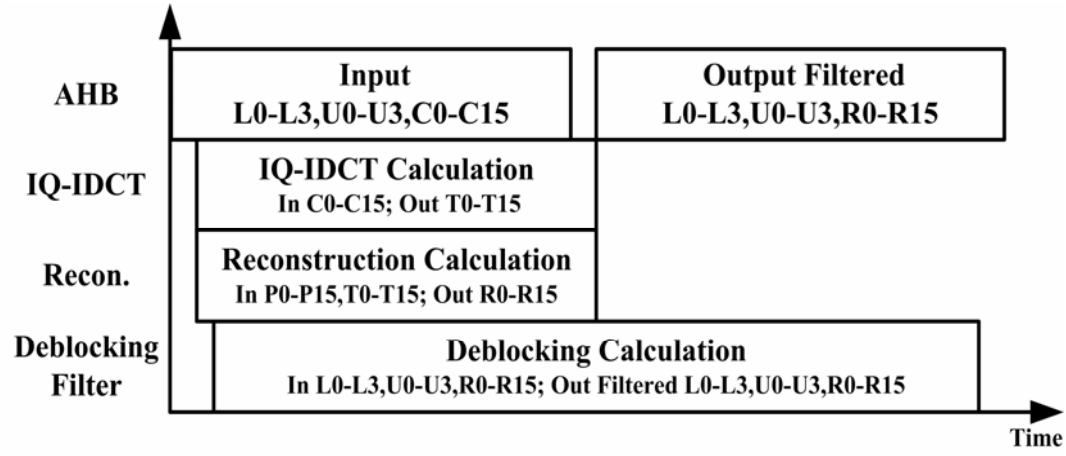



Fig. 46. Interleaved process for IQ-IDCT, reconstruction and deblocking filter.

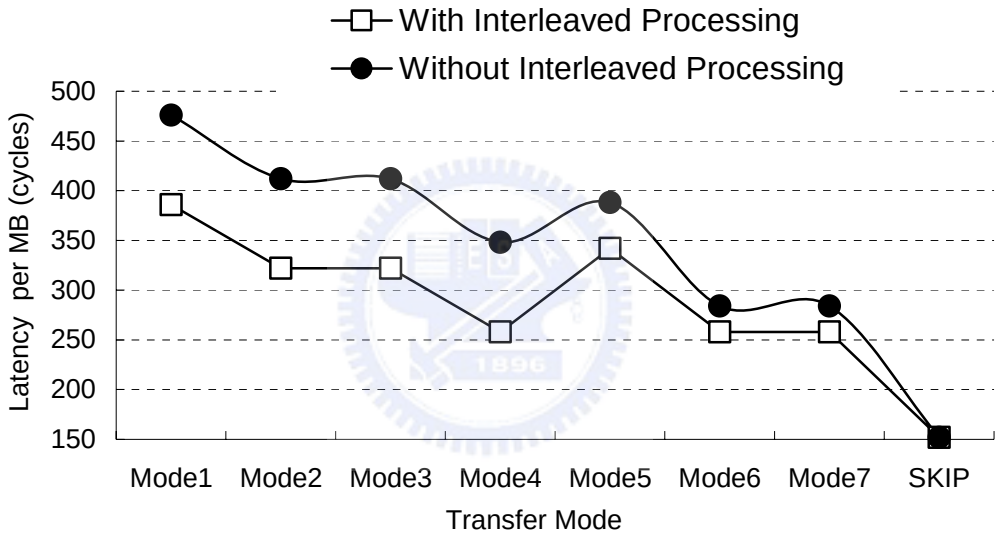



Fig. 47. Latency of inverse transforming and deblocking a macroblock.

represents the decoding flow for a macroblock. The macroblock-sized quantized DCT coefficients (C0-C15) are transformed inversely into IQ-IDCT residuals (T0- T15). Next, reconstruction macroblock (R0-R15) is produced after predictions (P0-P15) have added with the residuals. Finally, the reconstruction macroblock and the data from upper and left macroblock are passed for deblocking filtering. By using bus-interleaved IQ-IDCT and deblocking filter, the data transmission, IQ-IDCT calculation, reconstruction and deblocking filtering can be performed in parallel. Fig.

46 shows the interleaved process for IQ-IDCT, reconstruction and deblocking. By adaptive transfer scheme for deblocking filter described in section 4.2, the processing latency is variable depending on the transfer mode. Fig 47 shows the processing latency for inverse transforming and deblocking a macroblock. With interleaved process, the processing latency for IQ-IDCT can overlap with processing latency for deblocking filter so as to increase system performance.

## 4.4. Accelerator for Motion Compensation

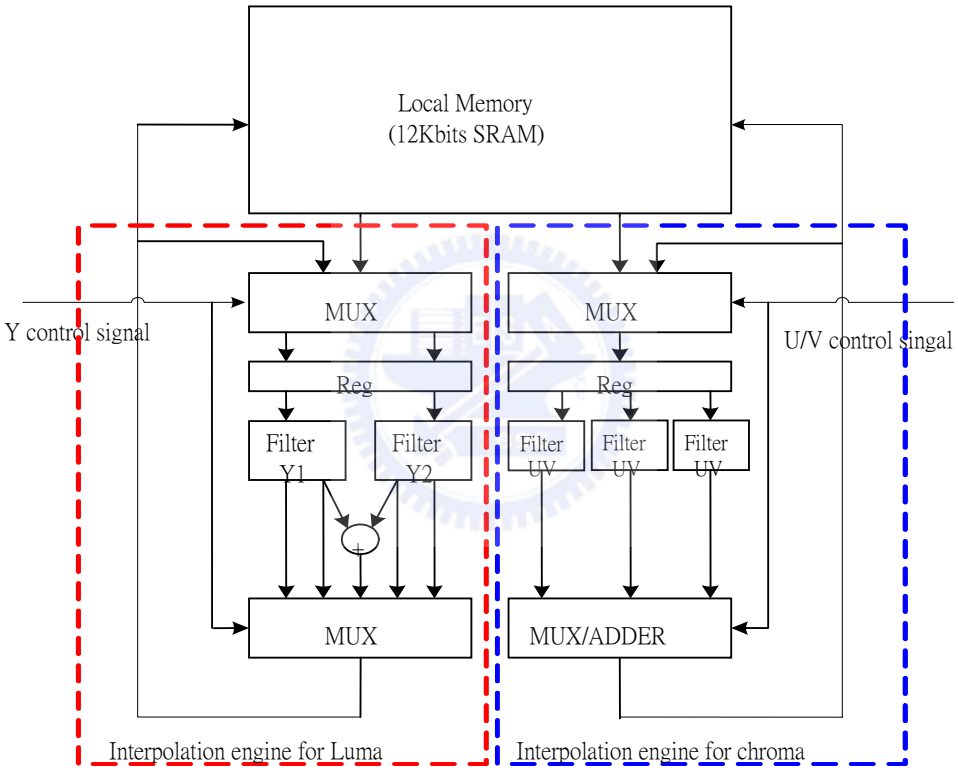



Fig. 48. Interpolation architecture for quarter-pel motion compensation [10].

To speed up interpolation, we design a dedicated co-processor for motion compensation [10] as show in in Fig. 48. For a macroblock interpolation, the motion compensation uses a local memory to store 1500 integer pixels and two interpolation engines for parallel processing of the luminance and chrominance components. Each engine consists of multiple multipliers and accumulators. Particularly, the multiplier is

implemented in a hardwired manner to maximize performance. To get one interpolated macroblock, the intermediate results (output) of row and column filtering in each engine may be fed back to conduct the filtering of another columns and rows.

The AVC specification uses variable block size motion compensation. The minimum granularity for motion compensation is a 4x4 block. Therefore, our interpolation engine is designed for a 4x4 block. The interpolation of each macroblock takes 16 iterations. In the worst case, our design takes 1280 cycles to interpolate one macroblock. As operating at 10 MHz, the throughput for each macroblock is 7812 macroblock/sec. The area used is 11172 gates.

# Chapter 5

# Experiment Results

## 5.1 Experiment Environment and Tools

In our experiment, we design an ARM-based platform for the H.264/MPEG-4 AVC decoder. Basically, our decoding system is constructed with the configuration in Fig. 49. Specifically, our ARM integrator baseboard [35] (as shown in Fig.50) employs JTAG (Joint Task Action Group) interface to connect with an ARM MultiICE (as shown in Fig.51). The MultiICE connects to a host computer to conduct the communication between computer and ARM board. Our ARM board mainly includes two parts, core module and logic module. In the core module, there are ARM966 CPU, embedded SRAM (1 MBytes), and external memory interface. On the other hand, the dedicated accelerators are implemented on the logic module which is a FPGA (Filed-programmable Gate Array). Moreover, ARM board employs the AHB bus interfaces to communicate the core module and logic module. Besides, the clock rates for ARM CPU, FPGA and AHB bus list in Table 13.

To facilitate the verification, we utilize ARM developer suite v1.2 [36] to develop our system. It mainly includes two development software tools, CodeWarrior and AXD, which window interfaces are shown in Fig.52 and Fig.53 respectively. In the software design flow of Fig.55, our source codes are coded as assembly (for firmware design) and C/C++ language. At first, we employ CodeWarrior to compile these source codes and produce ARM-based link object files. Then, the CodeWarrior link

Table 13. The processing rate of CPU, FPGA and AHB bus.

| Modules | Processing Rate |
|---|---|
| **ARM966E-S** | 130MHz |
| **FPGA** | 10MHz |
| **AHB** | 33MHz |

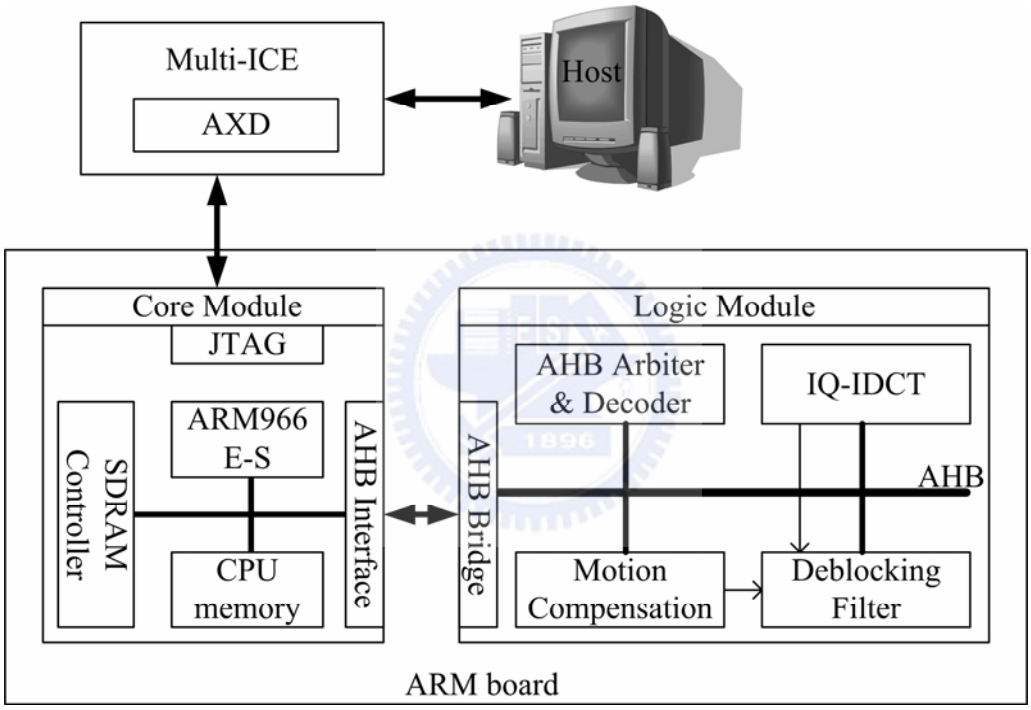



Fig. 49. Our peoposed decoding system

these object files to generate executable AXD code for ARM966E-S processor. Next, the development host use AXD to interfacing the JTAG port on ARM board through the ARM multiICE cable. Hence, we can run the executable file at the ARM966E-S processor and debug it using AXD environment.

On the other hand, we develop our hardware designs based on the design follow as shown in Fig.56. We code our hardware designs as Verilog RTL language. After debugging and simulation using the tools of Cadence Verilog Simulator, Debussy,

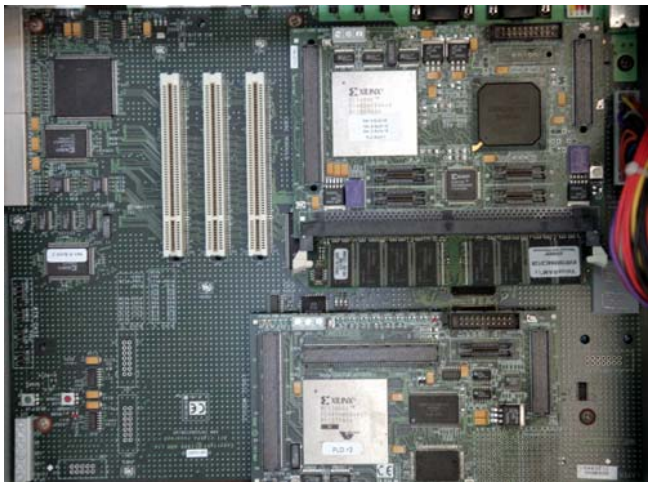


Fig. 50. ARM integrator baseboard.


Fig. 51. ARM MultiICE.

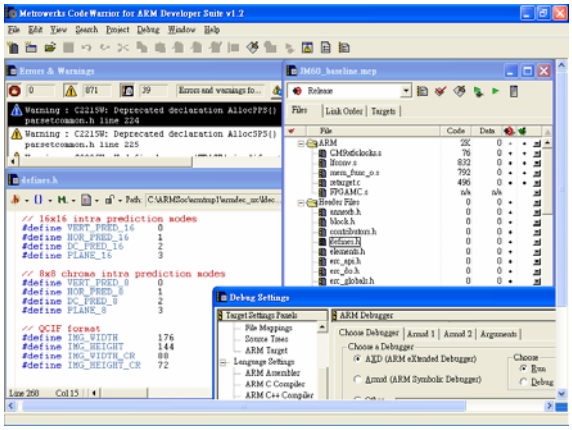



Fig. 52. Window interface to CodeWarrior [36].

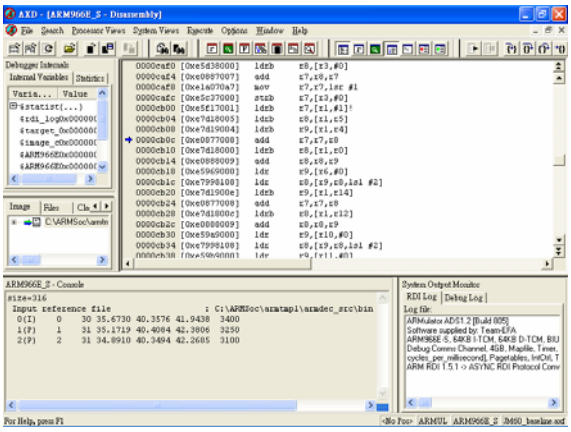


Fig. 53. Window interface to AXD [36].

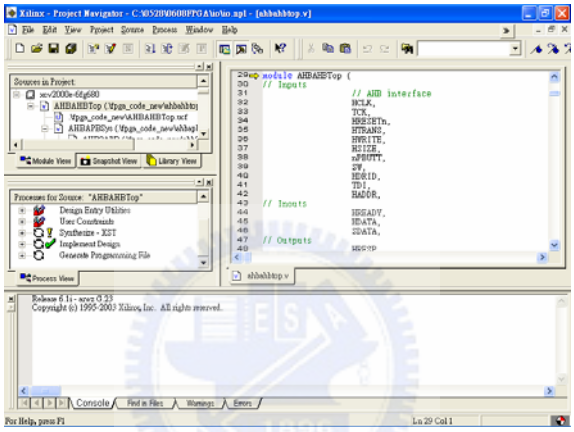


Fig. 54. Window interface to Xlinx Project Navigator [37].

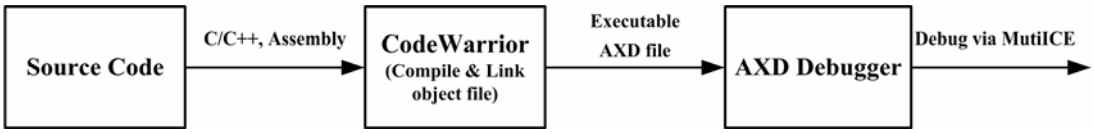



Fig. 55. Software design flow.

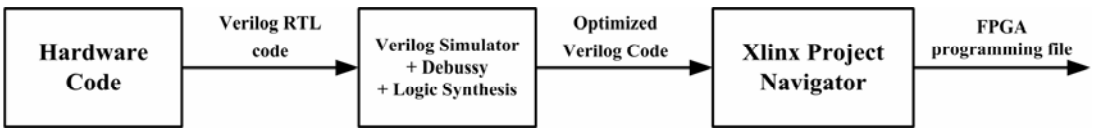



Fig. 56. Hardware design flow.

and Logic Synthesis, we employ the tool of Xlinx Project Navigator [37], which window interface show in Fig.54, to perform FPGA synthesis and P&R. The produced programming file is burned on the FPGA module on ARM board. Next, to verify software and hardware simultaneously, we can probe the hardware signal from FPGA by using the logic analyzer and debug the software and firmware by using AXD.

## 5.2 System Performance Comparison Using H.264/MPEG-4 AVC Decoder

In this section, we use an ARM based H.264/MPEG-4 AVC decoder as an example to demonstrate the system performance of our design. In Table 14, we classify 4 types of architecture based on whether the software is optimized or whether the MC, IQ-IDCT, or deblocking filter accelerator is integrated. To compare the system performance of the 4 types of architecture, Fig. 57 illustrates the decoding throughput for the 4 architectures in frames per second (fps) on ARM966 CPU. Table 15 shows the experiment parameters of test sequences. We describe these architectures as follows:

1. **The case of JM6.0 Decoder [38]:** When only the reference software of H.264 JM 6.0 decoder is executed on ARM-based platform, the decoding speed is about 0.3 to 1.2 fps for the video sequences in QCIF resolution.
2. **The Architecture A:** On the other hand, we optimize the software without integrating any hardware accelerator in Architecture A. The decoding speed is about 2.3 to 8.1 fps and 5 fps on the average.
3. **The Architecture B:** We replace the software MC and IQ-IDCT with 2 accelerators of MC and IQ-IDCT to our system in Architecture B. As compared to only software optimization of Architecture A, the throughput of Case B is increased by 30%~60% and 6.6 fps on the average.
4. **The Architecture C:** The accelerator of bus-interleave deblocking filter is embedded to our system in Architecture C. As compared to the software deblocking filter in Arcitecture B, our proposed deblocking accelerator can

Table 14. Four ahcitectures for performance evolution.

| | JM6.0 Decoder | Architecture A | Architecture B | Architecture C |
|---|---|---|---|---|
| MC | SW | Optimized SW | HW | HW |
| IQ-IDCT | SW | Optimized SW | HW | HW |
| Deblocking Filter | SW | Optimized SW | Optimized SW | HW |
| CAVLD & Others | SW | Optimized SW | Optimized SW | Optimized SW |

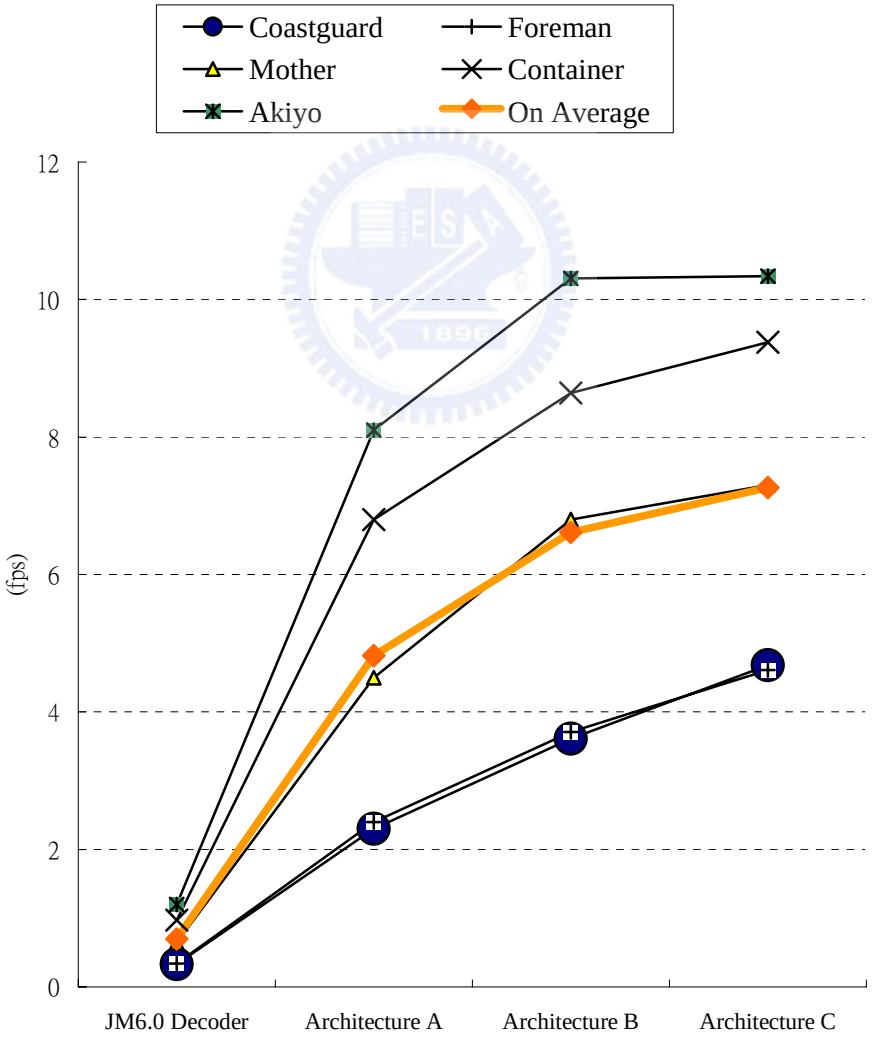



Fig. 57. System performance comparison using h.264/MPEG-4 decoder.

Table15. Experiment parameters of test sequences.

| | | |
|---|---|---|
| Frame Size | | QCIF |
| Frame Rate | | 15fps |
| Qp | | I(28)P(31) |
| Group of Picture | | 1I + 149P |
| Reference Frame Number | | 5 |
| Bit-rate (kbits/s) | Coastguard | 13.15 |
| | Foreman | 8.67 |
| | Mother | 58.69 |
| | Container | 14.43 |
| | Akiyo | 42.47 |

contribute up to 30% throughput improvement. Hence, the throughput ranges from 4.7 to 10.4 fps and 7.26 fps on the average.

In conclude, our overall throughput is enhanced from 0.3/1.2 fps in JM6.0 to 4.7/10.4 fps in Architecture C. Thus, our experiments show that the throughput of the H.264/MPEG-4 AVC reference decoder can be improved by 8.6 to 15.6 times. Besides, based on Fig. 57, some features can be observed as follows. First, the decoding throughput is sequence dependent. Our decoder performs better for slow motion sequences for some reasons: (1) slow motion sequences have more zero DCT blocks, (2) slow motion sequences have high probability of using integer pixel resolution MC, and (3) most of macroblocks in slow motion sequences require no deblocking filtering. Hence, the decoding rate is increased due to less computation for the MC, IDCT-IQ and deblocking filter modules with slow motion sequences. However, less computation for hardware accelerators also results in lower percentage of hardware operation. By Amdahl’s law, the improvement ratio using accelerators for decoding slow motion sequences is smaller than for decoding fast motion ones. In addition, especially for decoding slow motion sequences, the adaptive transfer scheme described in chpater4 is important because most data transfers in slow motion

sequences are not required. Hence, the redundant data transfer burdens the bandwidth and performance.

# Chapter 6

# Conclusion and Future Work

In this thesis, we propose a macroblock-level pipelining architecture for a H.264/MPEG-4 AVC decoder based on both platform-based design methodology and application specific circuit design methodology. With platform design methodology, the software procedures and hardware modules retain a high degree of reusability. Hence, it shortens the design cycles so that we can quickly integrate our design into industrial application. With application specific circuit design methodology, we conduct task partitioning and scheduling in the macroblock-level to enhance the overall decoding throughput. The software parts control the branching data flow and the hardware accelerators speed up the regular and computationally intensive modules.

In the hardware acceleration designs, we present a platform based bus-interleaved architecture for deblocking filter in H.264/MPEG-4 AVC. We have shown that performing the data transfer and filtering operation in parallel can significantly reduce the processing latency. Moreover, classifying macroblock filtering mode can avoid redundant data transfer so as to efficiently use bus bandwidth. Moreover, we utilize bus-interleaved IQ-IDCT and deblocking filter to perform data transfer, inverse transforming, reconstruction and deblocking filtering in parallel. As compared to traditional shared memory architecture, we have shown that we can remove intermediate buffer and achieve the same performance.

Based on the dedicated accelerators and macroblock-level pipelining, our proposed decoder achieves significant improvement in speed using both software and

hardware co-design. For the industrial applications, our proposed design is suitable for low cost and high performance multimedia applications. Also, it can be quickly embedded into the ARM based system-on-chip design.

In the future works, we dedicate to our project in two aspects:

1. *Non-buffered architecture design for H.264/MPEG-4 AVC decoder*

   In this thesis, we have shown the low-cost and high-performance of bus-interleaved designs. By taking advantage of the interleaved processing method, we can implement bus-interleaved MC and intra prediction. Our goal is to parallel process the data transfer and functional computation for all dedicated accelerators. Hence, the interleaved processing reduces the processing latency. Besides, we pass the intermediate data to next accelerator to avoid the usage of intermediate buffer for macroblock-level pipelining. Thus, our proposed decoder represents the non-buffered memory architecture among all the bus-interleaved accelerators to achieve low cost and high performance.

2. *Processor-based chip implementation*

   Our system is processor-based that contains an ARM processor to conduct software operation and hardware control behavior. As compared to VLSI ASIC circuit, it is more challengeable to implement a processor-based chip. Our goal is to realize system-on-chip implementation. Several expected features of our chip list as follows.

   - Processor-based configuration.
   - Low cost.
   - Low power consumption.
   - High processing ability.
   - Flexibility.
   - IP reusability.

# Bibliography


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

[2] *Video Coding for Low Bit Rate Communication*, ITU-T Rec. H.263, 1998.

[3] *Information technology -- Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s -- Part 2: Video*, ISO/IEC 11172-2, 1993.

[4] *Information Technology - Generic Coding of Moving Pictures and Associated Audio Information: Video*, ISO/IEC 13818-2 and ITU-T Rec. H.262, 1996.

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

[6] T. Wiegand, G. J. Sullivan, G. Bjontegaard, A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 560-576, July. 2003.

[7] *http://www.arm.com*, ARM Ltd

[8] *ARM968E-S Technical Reference Manual*, ARM Ltd., 2004.

[9] *AMBA™ Specification Rev 2.0*, ARM Ltd, 1999.

[10] S. H. Wang, W. H. Peng, Y. He, G. Y. Lin, C. Y. Lin, S. C. Chang, C. N. Wang, and T. Chiang, "A platform-based MPEG-4 advanced video coding (AVC) decoder with block level pipelining", *IEEE Pacific Rim Conf. on Multimedi*a, vol. 1, pp. 51-55, Dec. 2003.

[11] S. H. Wang, W. H. Peng, Y. He, G. Y. Lin, C. Y. Lin, S. C. Chang, C. N.

Wang, and T. Chiang, “A Software-Hardware Co-Implementation of MPEG-4 Advanced Video Coding (AVC) Decoder with Block Level Pipelining, ” *VLSI Signal Processing*, vol. 41, no. 1, pp. 93-110, Aug. 2005.

[12] S. C. Chang, W. H. Peng, S. H. Wang and T. Chiang, “A platform-based de-blocking filter design with bus-interleaved architecture for H.264”, *IEEE Int’l Conf. on Consumer Electronics*, pp. 293-294, Las Vegas, Jan. 2005.

[13] S. C. Chang, W. H. Peng, S. H. Wang and T. Chiang, “A Platform Based Bus-interleaved Architecture for Deblocking Filter in H.264/MPEG-4 AVC”, *IEEE Trans. on Consumer Electronics*, vol. 51, pp. 249-255, Feb. 2005.

[14] T. C. Chen, Y. W. Huang, C. H. T, T. W. Chen, and L. G. Chen, "A 1.3 TOPS H.264/AVC single-chip encoder for HDTV applications", *IEEE Int'l Solid-State Circuits Conf.*, San Francisco, USA, Feb. 2005

[15] C. C. Cheng and T. S. Chang, "An hardware efficient deblocking filter for H.264/AVC", *IEEE Int'l Conf. on Consumer Electronics*, pp. 235-236, Las Vegas, Jan. 2005.

[16] Y. W. Huang, T. W. Chen, B. Y. Hsieh, T. C. Wang, T. H. Chang, and L. -G. Chen, "Architecture design for deblocking filter in H.264/JVT/AVC", *IEEE Int'l Conf. on Multimedia and Expo.* vol. 1, pp. 693-696, July 2003.

[17] T. M. Liu, W. P. Lee, T. A. Lin, and C. Y. Lee, "A memory-efficient deblocking filter for H.264/AVC video coding", *IEEE Int'l Symposium on Circuits and Systems*, pp. 2140 – 2143, May 2005.

[18] M. Sima, Y. Zhou, and W. Zhang, "An efficient architecture for adaptive deblocking filter of H.264/AVC", *IEEE Trans. on Consumer Electronics*, vol. 50, no. 1, pp. 292-296, Feb. 2004.

[19] V. Venkatraman, S. Krishnan, and N. Ling, "Architecture for de-blocking filter

in H.264", *Picture Coding Symposium*, San Francisco, USA, Dec. 2004

[20] G. Q. Zheng and L. Yu, "An efficient architecture design for deblocking loop filter", *Picture Coding Symposium*, San Francisco, USA, Dec. 2004

[21] D. Marpe, H. Schwarz, and T.Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 620–644, July 2003.

[22] Bergeron, C., Lamy-Bergot, C., "Soft-input decoding of variable-length codes applied to the H.264 standard," *Multimedia Signal Processing*, pp. 87 – 90, Oct. 2004.

[23] H. S. Malvar, A. Hallapuro, M. Karczewicz, and Louis Kerosfsky, "Low complexity transform and quantization in H.264/AVC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 598–603, July 2003.

[24] M. T. Orchard and G. J. Sullivan, "Overlapped Block Motion Compensation: An Estimation-Theoretic Approach," *IEEE Trans. Image Processing*, pp. 693-699, Sept. 1994

[25] S. D. Kim, J. Yi, and J. B. Ra, "A Deblocking Filter with Two Separate Modes in Block-Based Video Coding." *IEEE Trans. Circuits Syst. Video Technol*., pp. 156-161, Feb.1999.

[26] G. Martin, "A design chain for embedded systems," *IEEE Computer Magazine*, vol. 35, pp. 100-103, March, 2002.

[27] T. Givargis and F. Vahid, "Platune: a tuning framework for system-on-a-chip platforms, " *IEEE Trans. on Computer-Aided-Design of Integrated Circuits and Systems*, vol. 21, pp. 1317 – 1327, Nov. 2002.

[28] M. Harrand, et al., "A single chip CIF 30-Hz, H.261, H.263, and H.263+ video encoder\decoder with embedded display controller," *IEEE Journal of Solid State Circuits*, vol. 34, pp.1627-1633, Nov. 1998.

[29] M. Ikeda, et al, "SuperEnc: MPEG-2 video encoder chip," *IEEE Micro magazine*, vol.19, pp. 56-65, July 1999.

[30] M. Berekovic, et al, "Muiticore system-on-chip architecture for MPEG-4 streaming video," *IEEE Trans. On Circuit and System for Video Technology*, vol. 12, no. 8, pp. 688-699, Aug. 2002.

[31] K. Ramkishor and V. Gunashree, "Real time implementation of MPEG-4 video decoder on ARM7TDMI," Proc. *IEEE International Symposium on Intelligent Multimedia, Video, and Speech Processing*, pp. 522-526, May 2001.

[32] M. Zhou and R. Talluri, "DSP-based real time video decoding," *Proc. IEEE International Conference on Consumer Electronics*, pp. 296-297, June 1999.

[33] Lukowicz, P, “Design of an efficient shared memory architecture using hybrid opto-electronic VLSI circuits and space invariant optical buses”, *Massively Parallel Processing Using Optical Interconnections*, pp. 231-238, Oct. 1996.

[34] T.C. Wang, Y.W. Huang, H.C. Fang, and L.G. Chen, “Parallel 4/spl times/4 2D transform and inverse transform architecture for MPEG-4 AVC/H.264”, *Circits and Systems (ISCAS'03)*, vol. 2, pp. 800-803, Thailand, Taiwan, April, 2003

[35] *Integrator™/LM-XCV600E+ and Integrator™/LM-EP20K600E+ User Guide*, ARM Ltd, 2001

[36] *ARM Developer Suite Getting Started*, ARM Ltd, 2001.

[37] *Xilinx-Platform FPGA Virtex-II datasheet*, Xilinx.Ltd, 2000.

[38] *Joint Mode Reference Software version 6.0*, ARM Ltd, 2001.

# 張世騫(Shih-Chien Chang)

## Contact Details:

**E-mail address :** Shihchien.ee92g@nctu.edu.tw

## Key Skills:

Hardware Design Skill:

- Verilog and VHDL coding and debugging
- Synthesis optimization
- Physical layout skill

System Architecture Skill:

- System integration on ARM-based platform
- Firmware and Embedded C programming
- AMBA on-chip bus optimization

Knowledge:

- Video compressor standards knowledge, e.g. H.264/MPEG4 AVC
- Digital communications and signal processing

## Education:

| | |
|---|---|
| 2003-2005 | **Master Degree in Electronics Engineering, National Chiao Tung University, Taiwan.** |
| 2000-2003 | **Bachelor Degree in Electronics Engineering, National Chiao Tung University, Taiwan.** |
| 1999-2000 | Bachelor study in Industrial Engineering & Management, National Chiao Tung University, Taiwan. |
| 2002 | English Language School, Australia Sydney University |

## Publications

A. Journal Paper

1. Shih-Chien Chang, Wen-Hsiao Peng, Shih-Hao Wang and Tihao Chiang, "A Platform based Bus-interleaved Architecture for Deblocking Filter in H.264/MPEG-4 AVC", *IEEE Trans. on Consumer Electronics*, 2005.

2. Shih-Hao Wang, Wen-Hsiao Peng, Yuwen He, Guan-Yi Lin, Chen-Yi Lin, Shih-Chien Chang, C.-N. Wang, and Tihao Chiang, "A Software-Hardware Co-Implementation of MPEG-4 Advanced Video Coding (AVC) Decoder with Block Level Pipelining, " *Journal of VLSI Signal Processing Systems*, 2004.

B. International Conference Papers

1. Shih-Chien Chang, Wen-Hsiao Peng, Shih-Hao Wang and Tihao Chiang, “A Platform based Deblocking Filter Design with Bus Interleaved Architecture for H.264”, *IEEE Int’l Conf. on Consumer Electronics*, Las Vegas, Jan. 2005.

2. Shih-Hao Wang, Wen-Hsiao Peng, Yuwen He, Guan-Yi Lin, Chen-Yi Lin, Shih-Chien Chang, Chung-Neng Wang, and Tihao Chiang, “A Platform-Based MPEG-4 Advanced Video Coding (AVC) Decoder with Block Level Pipelining”, *Proc. IEEE ICICS-PCM*, Singapore, Nov. 2003

D. Thesis

1. Shih-Chien Chang and Tihao Chiang, " ARM-based Platform Design for H.264/MPEG-4 AVC Decoder and Accelerator for Deblocking Filter", Master Thesis, Dept. of Electronics Engineering, NCTU, R.O.C., 2005.

## Research Project

- H.264 video encoder/decoder implementation

## Research Interests

- Digital IC design
- Video encoder/decoder implementation and optimization
- Communication network optimization
- System-on-chip architecture improvement
- Biomedical signal processing and neural network