# 國立交通大學

## 電子工程學系　電子研究所碩士班

## 碩　士　論　文

# 應用於 H.264/AVC 視訊標準之去區塊濾波器的設計

# A Deblocking Filter for H.264/AVC Applications


學生 ： 李文平

指導教授 ： 李鎮宜 教授


中華民國九十六年七月

# 應用於 H.264/AVC 視訊標準之去區塊濾波器的設計

# A Deblocking Filter for H.264/AVC Applications


研 究 生：李文平　　Student：Wen-Ping Lee

指導教授：李鎮宜　　Advisor：Chen-Yi Lee

國 立 交 通 大 學

電子工程學系 電子研究所 碩士班


碩 士 論 文

A Thesis

Submitted to Department of Electronics Engineering & Institute of Electronics

College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Electronics Engineering

July 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年七月

# 應用於 H.264/AVC 視訊標準之去區塊濾波器的設計

學生：李文平　　指導教授：李鎮宜 教授

**國立交通大學**

**電子工程學系 電子研究所碩士班**

## 摘要

H.264/AVC 是最新一代的視訊壓縮標準。藉由使用許多有效的方法，相較於之前的視訊標準，H.264 提供了更高的壓縮效能，在相同的壓縮比率下提供更好的影像品質。其中一個方法就是採用了去區塊濾波器，使得壓縮效率得以提升並有效率地移除區塊雜訊進而提升畫面品質。在本論文中，我們實作了一個資料重覆使用的高速去區塊濾波器，期望能達到高解析度的數位電視、無線通訊上的影像解碼需求。

針對能支援多種視訊編碼標準的解碼器，我們成功地整合了應用在不同視訊標準的去區塊濾波器，比起各別去用硬體實現所花費的硬體成本總合，我們所提出來的雙模去區塊濾波器能省下約 30%的硬體成本。

另外，針對即時影像通訊的應用，我們採用錯誤補償的方式以降低影像品質在傳輸過程中遇到錯誤所造成的影響。在本論文中，我們提出了以去區塊濾波器為基礎的錯誤補償演算法，並實作了一個具錯誤補償功能的去區塊濾波器。相較於一般的錯誤補償演算法，我們所提出來演算法藉由共用硬體的方法省下了約 30%的硬體成本。

最後，我們將本論文所提的具錯誤補償功能之去區塊濾器整合到 H.264 解碼器

上。從模擬上可以得知，我們所提出的去區塊濾器在硬體成本及功率消耗上只佔了整個解碼器的 13%跟 16%。

# A Deblocking Filter for H.264/AVC Applications

Student : Wen-Ping Lee Advisor : Dr. Chen-Yi Lee

Department of Electronics Engineering

Institute of Electronics

National Chiao Tung University

## ABSTRACT

H.264/AVC is the newest video coding standard. With many useful tools, H.264 provides better coding efficiency, which means that it provides better image quality at the same coding rate as compared to the previous video coding standard. One of the useful tools is deblocking filter. The coding efficiency is improved and blocking artifacts are removed by the deblocking filter specified in H.264/AVC. In this thesis, we implemented a high throughput deblocking filter with data reuse to achieve the demands on digital TV and wireless communication with high resolution.

For the multi-standard decoder, we successfully integrate the deblocking filter in H.264/AVC and post-loop filter. The proposal can save about 30% hardware cost as compared to the total hardware cost of implementations separately.

Besides, we use error concealment methods to decrease the influence of video quality caused by the bit-error in video transmission for the applications of real-time video transmission. In the thesis, we proposed an algorithm of error concealment based on the deblocking filter in H.264/AVC and implemented error-concealed deblocking filter. The

proposal saves about 30% hardware cost as compared to the total hardware cost of deblocking filter and error concealment.

Final, we integrated the ECDF into our H.264 decoder and redesigned for the real-time mobile applications. From the simulations, the ECDF contributes about 13% and 16% of hardware cost and power consumption over H.264/AVC decoder respectively.

# 誌　　謝

從大四開始做專題的那年開始，以及身為碩士生這兩年研究日子，這段時間要感謝的人很多，因為有你們的指導、陪伴，才能有今日的我。

首先要感謝我的老師李鎮宜教授。雖然老師一直以來都忙碌於行政，但是在一星期一次的會議中，老師與演講者的對談裡無時無刻都能讓人清楚地感覺到老師在研究上的專業；不管是研究方向上或是表達意見上，中肯而溫和的意見中總是有許多有用的知識。能跟隨老師做研究真的是不可多得的好事。

再來要感謝 si2 所有的成員。學長們豐富的經驗傳承及循序漸進的誘導、同窗間真誠的相互關照及甘霖般地吐槽，因為這些讓我能在繁重的課業中以最少的負擔持續不斷地做研究。也因為有大家的貢獻，源源不絕的精神糧食也提供了許多休息管道讓我能在閒暇時回復精力再做衝刺。同時感謝我那在另一片天空下奮鬥的朋友們。如同逃避般暫時性切斷與現實的聯繫並專心地挑戰暴風雪般的難關，在這樣的過程中，讓我忘卻所有研究上或課業上的煩躁與不安，重新再出發。

最後感謝我的家人。即使很少回家，卻也從不擔心我的求學情況；回到家中面對的全是好好休息與支持的話語；這樣的信賴以及支持讓我能深信自己的想法專心研究，為了自己也為了別人，自始至終。

專題生那年以及研究生這兩年，生活雖然忙碌卻也充實，即使煩悶卻也歡笑。因為你們，所有出現在我研究生生活的人，謝謝。

# Contents

# List of Figures

# List of Tables

# Chapter 1
# Introduction

## 1.1 Video Coding Standards

The video coding standards play an important role in all aspects of entertainment in the world. Common video coding standards includes MPEG-1/2/4, H.261/2/3/4. Generally speaking, the data of uncompressed video may be larger than the capacity of content storage and transmitted with a lot of time by digital broadcasting due to the limitation of bandwidth. With the video compression techniques described in these video coding standards, the video data can be stored or transmitted more efficiently.

H.261 is the earliest DCT-based video coding standard and is the basis hybrid video coding for other common video coding standards today. H.261 is designed for ISDN (Integrated Services Digital Network), the internet system in early day, the applications of videophone and videoconferencing. And the targeted bit rate of H.261 is p x 64 (p = 1, …, 30). MPEG-1/2 are designed and focused on VHS (Video Home System) and widely used for Video CD/DVD format. H.263 is designed and focused on the quality at very low bit rates (<64Kbps) and has overtaken H.261 to dominate videoconferencing codec. Object-based coding is specified in the MPEG-4 which is based on the H.263.

H.264/AVC is developed by MPEG (Moving Picture Experts Group) and VCEG (Video Coding Experts Group) that promises to outperform the earlier standards, MPEG-1/2/4 and H.263, providing better visual quality at the same bit rate or lower bit rate with the same visual quality.

With the video coding standards, each frame in video is divided into macroblocks

(MBs) and coded with intra/inter prediction each by each. The residual data is encoded with discrete cosine transform (DCT) to discard high-frequency data and quantization after prediction. Further, the quantized coefficients are coded using variable-length coding techniques, such as CAVLC/CABAC specified in H.264/AVC to form the video bit streams. On the other hand, the decoder receives the coded video bit streams to reconstruct the coded video with reverse operations in encoder.

## *1.2 H.264/AVC Standard Overview*

H.264 is the newest high compression digital video codec standard written by the ITU-T Video Coding Experts Group (VCEG) together with the ISO/IEC Moving Picture Experts Group (MPEG). Thanks to its new features, H.264 achieves higher compression gain than previous standards, such as MPEG-2 or H.263, as shown in Figure 1.1.

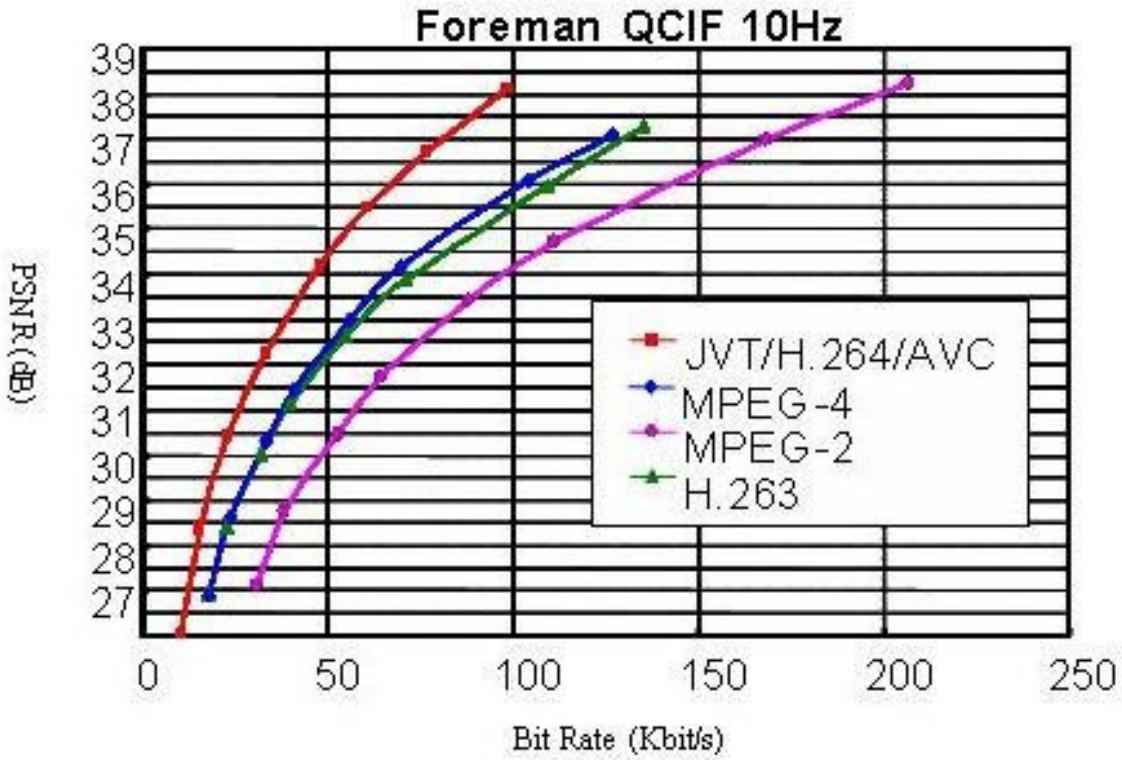



Figure 1.1: PSNR virus bit rate for different standards.

It uses many useful but complex techniques and algorithms to achieve this goal. From

the Table 1.1, we can see the different techniques used in MPEG-2 and H.264. The first difference of technique is transform block size. While MPEG-2 use 8x8 block size floating point DCT, H.264 use 4x4 block size integer DCT. For the motion vector, H.264 use higher resolution method and variable block size, 1/4 resolution for luma and 1/8 for chroma on motion vector and several block size from 4x4 to 16x16 for motion estimation. Besides, several modes are used for the prediction in spatial domain. In the entropy coding, H.264 used complex method, CAVLC and CABAC, to improve the compression efficiency. Finally, H.264 adopts the in-loop filter, deblocking filter, to eliminate the blocking effect caused by the DCT or motion estimation and improve the visual quality.

Table 1.1: Difference between techniques used in MPEG-2 and H.264/AVC.

| | MPEG-2 | H.264/AVC |
|---|---|---|
| Transform | 8x8 DCT | 4x4 DCT-like<br>2 levels transform |
| Motion compensation | ½ pel MC | ¼ pel MC |
| | 16x16 and 16x8 MC | 16x16 to 4x4 MC |
| | | Direct mode for B pictures |
| | 1 ref frame for P, 2 for B | Multiple reference pictures<br>B pictures can be used as reference<br>P can refer to pictures of the future |
| | | Weighted prediction (fading) |
| Spatial prediction | 1 mode (DCT domain) | Many intra modes (pixel domain)<br>Specific prediction for chroma |
| Entropy coding | VLC | Context Adaptive VLC<br>Context Adaptive Binary Arithm. Coding |
| Post-filtering | | Deblocking filter in coding loop |
| Quantization | Weighted matrices | Extended and finer<br>Distinct for luma and chroma |

### 1.2.1 Profiles and Levels

Like the previous standards, H.264/AVC contains several profiles for different applications as shown in Table 1.2. Each profile uses different set of coding tools for different applications. Generally speaking, the compression efficiency can be increased as well as the complexity with more coding tools. For example, Baseline profile which consists of the simplest set of compression tools, such as basic I, P slices, Context Adaptive Variable Length Coding and basic chroma format (4:2:0), results the least processing time for video decoder. It is suitable for low delay applications such as portable device or video conferencing. The main profile uses most compression tools, B slices, Context Adaptive Binary Arithmetic Coding, interlaced coding and so on, to improve the compression efficiency. The extended profile is also used for wireless mobile devices, but it is used primarily for the streaming media applications in PCs. The main difference between Baseline and Main profiles is the slice type used in coding. Moreover, the high profiles targets at high-definition TV applications such as HD-DVD and Blue-ray Disc. It uses higher bits per pixel and higher chroma format to obtain the high visual quality in high-definition TV applications.

Table 1.2: Profiles and coding tools in H.264/AVC.

| Coding tools \ Profile | Baseline | Main | Extended | High | High10 | High 4:2:2 | High 4:4:4 |
|---|---|---|---|---|---|---|---|
| Slice Type | I, P | I, P, B | I, P, B, SI, SB | I, P, B | I, P, B | I, P, B | I, P, B |
| Data Partition | NO | YES | NO | NO | NO | NO | NO |
| MBAFF | NO | YES | YES | YES | YES | YES | YES |
| Weighted Prediction | NO | YES | YES | YES | YES | YES | YES |
| Entropy Doding | CAVLC | CAVLC / CABAC | CAVLC | CAVLC / CABAC | CAVLC / CABAC | CAVLC / CABAC | CAVLC / CABAC |
| Chroma Format | 4:2:0 | 4:2:0 | 4:2:0 | 4:0:0<br>4:2:0 | 4:0:0<br>4:2:0 | 4:0:0 4:2:0 4:2:2 | 4:0:0 4:2:0<br>4:2:2 4:4:4 |
| 8x8 Transform | NO | NO | NO | YES | YES | YES | YES |
| Sample Depth | 8 bits | 8 bits | 8 bits | 8 bits | 8 ~ 10 bits | 8 ~ 12 bits | 8 ~ 12 bits |
| Flexible Macroblock Ordering | YES | NO | YES | NO | NO | NO | NO |
| Abritrary Slice Ordering | YES | NO | YES | NO | NO | NO | NO |

Much more than MPEG-2 levels can be found in the H.264 standard. We can see the

different limitations for different levels in the Table 1.3. From level 1 to level 5.1, maximum frame size ranges from 99 to 36,864 macroblocks, maximum video bit rate ranges from 64k to 240,000k bits/s for the 4:2:0 chroma format and so on. Other limitations, such as motion vector, are not mentioned here but can be found in the standard. These limitations restrict the resolution of video to support different applications.

Table 1.3: All levels defined in H.264/AVC.

| Level number | Max macroblock precessing rate (MB/s) | Max frame size (MBs) | Max decoded picture buffer size (1024 bytes for 4:2:0) | Max video bit rate for baseline, main and extended profile | Max video bit rate for High profile | Max video bit rate for High 10 profile | Max video bit rate for High 4:2:2 and High 4:4:4 profile |
|---|---|---|---|---|---|---|---|
| 1 | 1485 | 99 | 148.5 | 64kb/s | 80kb/s | 192kb/s | 256kb/s |
| 1.b | 1485 | 99 | 148.5 | 128kb/s | 160kb/s | 384kb/s | 512kb/s |
| 1.1 | 3000 | 396 | 337.5 | 192kb/s | 240kb/s | 576kb/s | 768kb/s |
| 1.2 | 6000 | 396 | 891 | 384kb/s | 480kb/s | 1152kb/s | 1536kb/s |
| 1.3 | 11880 | 396 | 891 | 768kb/s | 960kb/s | 2304kb/s | 3072kb/s |
| 2 | 11880 | 396 | 891 | 2000kb/s | 2500kb/s | 6000kb/s | 8000kb/s |
| 2.1 | 19800 | 792 | 1782 | 4000kb/s | 5000kb/s | 12000kb/s | 16000kb/s |
| 2.2 | 20250 | 1620 | 3037.5 | 4000kb/s | 5000kb/s | 12000kb/s | 16000kb/s |
| 3 | 40500 | 1620 | 3037.5 | 10000kb/s | 12500kb/s | 30000kb/s | 40000kb/s |
| 3.1 | 108000 | 3600 | 6750 | 14000kb/s | 17500kb/s | 42000kb/s | 56000kb/s |
| 3.2 | 216000 | 5120 | 7580 | 20000kb/s | 25000kb/s | 60000kb/s | 80000kb/s |
| 4 | 245760 | 8192 | 12288 | 20000kb/s | 25000kb/s | 60000kb/s | 80000kb/s |
| 4.1 | 245760 | 8192 | 12288 | 50000kb/s | 62500kb/s | 150000kb/s | 200000kb/s |
| 4.2/Lo | 491520 | 8192 | 12288 | 50000kb/s | 62500kb/s | 150000kb/s | 200000kb/s |
| 4.2/Hi | 522240 | 8704 | 13056 | 50000kb/s | 62500kb/s | 150000kb/s | 200000kb/s |
| 5 | 589824 | 22080 | 41400 | 135000kb/s | 168750kb/s | 405000kb/s | 540000kb/s |
| 5.1 | 983040 | 36864 | 69120 | 240000kb/s | 300000kb/s | 720000kb/s | 960000kb/s |

Based on the illustration of different profiles and levels, each application has the best candidate in terms of profile@level. For example, High@L4 is suitable for the high-definition TV applications since High profile used more coding tools and level 4 can support 1080HD of maximum video resolution. On the other hand, BL@L1 may be suitable

for the potable mobile device due to the lower complexity and lower power consumption.

### 1.2.2 Encoder/Decoder Block Diagram

The basic encoding process of H.264/AVC is the same as previous standards with hybrid DCT/MC coding infrastructure but more complex in detail. Figure 1.2 shows the block diagram of H.264/AVC encoder. The same as the MPEG-2 encoder, an embedded decoder exists inside the encoder that calculates the results of the motion compensation or intra prediction at the decoder side. With the embedded decoder in encoder side, the encoder can foresee the decoded result and calculate the residual values without mismatch to the decoder. One of the major improvements for H.264 is intra prediction as we can see in Table1.1. H.264 uses not only one prediction mode in spatial domain to improve the compression. Hence, there is an intra mode decision unit in the H.264/AVC encoder side as shown in Figure 1.2. Besides, inter prediction (motion compensation) has many coding tools, such as variable block sizes, multiple reference frames or short/long term prediction, to reduce the redundancy in the temporal domain. Further, the residual values are processed with DCT, quantization and entropy coding to reduce the coding redundancy. To reduce the blocking effect caused by the powerful block-based compression methods and eliminate the error propagation between frames, H.264 uses the in-loop filter, deblocking filter, to improve the visual quality. Finally, the bit stream of the H.264/AVC format is produced and transmitted or stored.

The decoder is simpler than the encoder because it lacks the decision parts like motion estimator and the intra mode decision parts. Instead, the decoder has syntax parser to decode syntax elements from the bit stream correctly as shown in Figure 1.3. The syntax parser decodes the syntax elements to decide which mode is used in motion compensation or intra prediction. The residual part of bit stream is processed with the inverse quantization and inverse DCT and transferred into residual values. With the predicted values and residual

values, the video can be constructed. Finally, the deblocking filter is invoked to eliminate the blocking effects resulted from prediction and DCT to improve the visual quality.

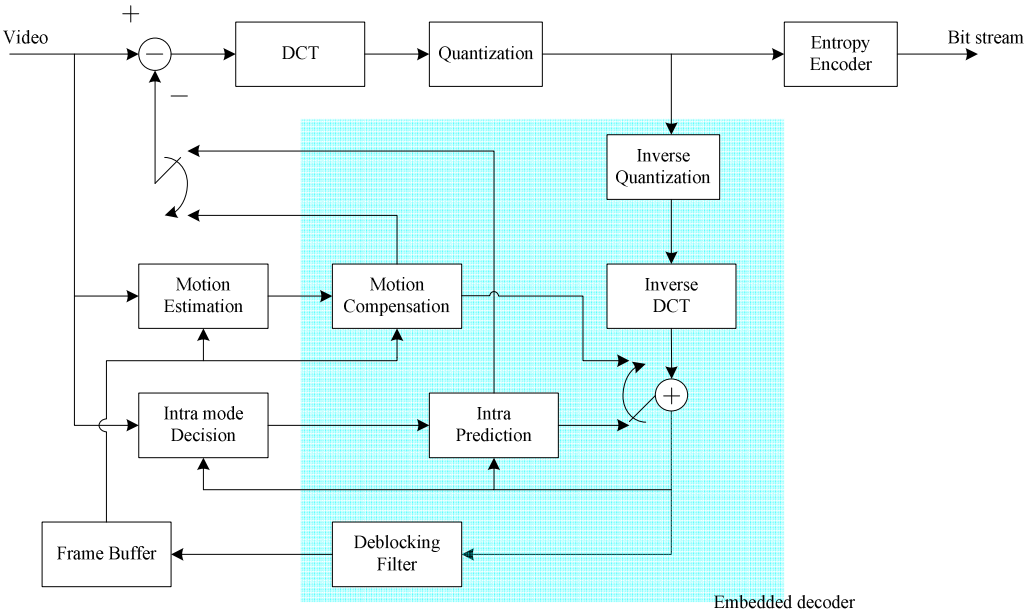



Figure 1.2: A simple block diagram of H.264/AVC video encoder.

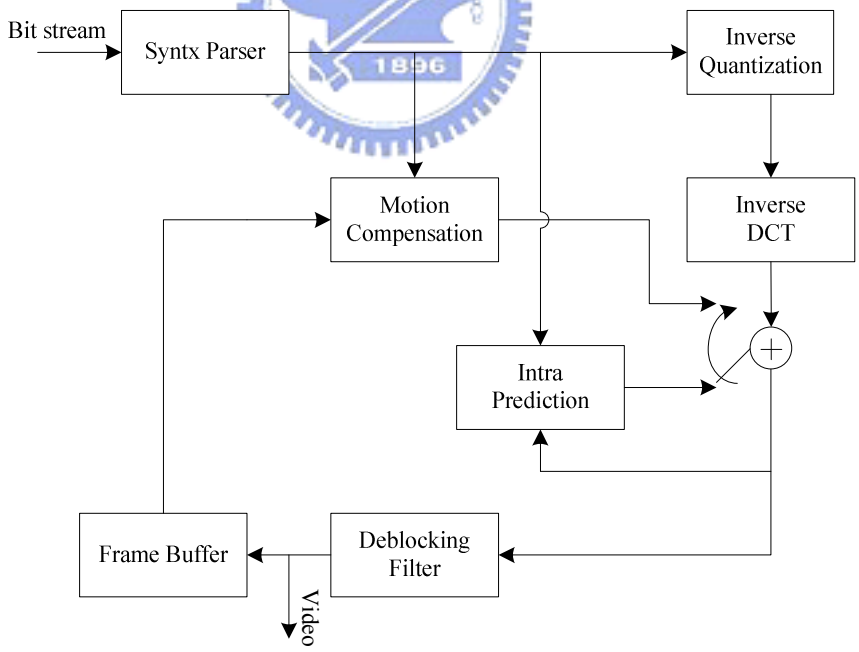



Figure 1.3: A simple block diagram of H.264/AVC video decoder.

### 1.2.3 Deblocking Filter

As we mentioned in previous sections, the major purpose of deblocking filter is to eliminate the blocking effects and error propagations between frames and improve the visual quality.


Figure 1.4: The visual quality with/without deblocking filter.

From left figure in Figure 1.4, we can see the blocking effects obviously. As we discussed in previous section, the blocking effect is caused by the powerful block-based compression tools, such as motion estimation, intra prediction, DCT and so on. Each block uses different motion vectors, intra modes or coefficients, the smooth regions may become discontinued. Hence, the blocking effects appear like the left figure in Figure 1.4. With the deblocking filter, the frame with a lot of blocking effects can be smoothed and become the right figure in Figure 1.4. Thus, the visual quality is improved and the blocking effects do not propagate to subsequent frames due to the in-loop filter, deblocking filter.

Actually, the blocking effects exist in the previous standards. Some standards has post-filter in optional to reduce the influence of blocking effects. But this influence of blocking effects becomes serious in H.264/AVC due to the compression tools. The more powerful compression tools standard use, more serious the problem is. From Table 1.1, we

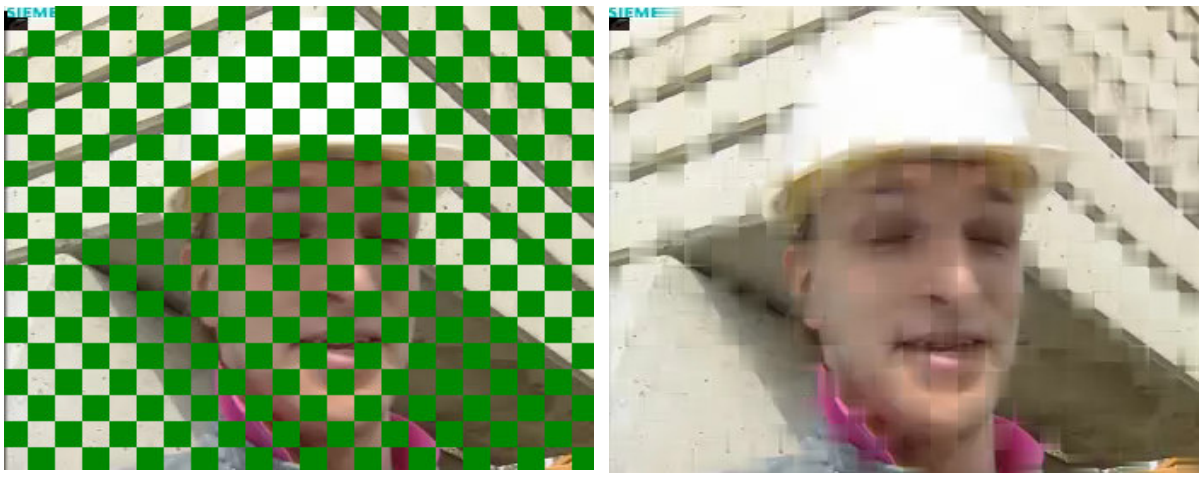
can see a lot of compression tools which may impact blocking effects, such as variable block size for motion vectors in motion compensation, intra modes in pixel domain for intra prediction and smaller block size for DCT. Hence, the in-loop filter, deblocking filter, is necessary for the H.264/AVC to reduce the side effect of the powerful compression tools in H.264/AVC and is standardized by H.264/AVC.

For the purpose of improving the visual quality, deblocking filter has to smooth all block edges in one frame. The complexity is mainly based on the high adaptivity of the filter, which requires conditional processing on the block edge and sample level. Another reason for the high complexity is the small block size employed for residual coding in H.264/AVC. Almost every sample in a picture must be loaded from memory, either to be modified or determined if neighboring samples will be modified with 1-D filter. Hence, the computational time and number of memory access of deblocking filter are larger than that of loop filter in H.263 or other post filters, which operate on the 8x8 block edges. Generally, the deblocking filter contributes about one-third of the computational complexity of the decoder [1], and it is the system bottleneck in terms of processing cycles. The throughput of deblocking is the most design issues for high-definition TV applications and power consumption is the main challenge for the applications of portable mobile devices.

## *1.3 Error Concealment*

Recently consumer electronics have increased demand for efficient and reliable video communications. Video consumer devices, such as digital television (DTV), mobile video and video telephony, have been developed rapidly. For transmission over bandwidth-limited networks, almost all video consumer electronics use compression technologies to reduce redundancies in video sequences. But compressed video streams are vulnerable to transmission errors, such as bit errors and packet errors. This can degrade the visual quality of the decoded sequence drastically because of use of motion-compensated interframe

coding and VLC coding as shown in Figure 1.5(a).



(a) (b)

Figure 1.5: The picture corrupted by the errors (a). The picture concealed by EC (b).

To protect video sequence against transmission errors, there are several techniques [2]. These techniques can be partitioned into three categories; Forward error concealment includes methods to add redundancy at the source end to enhance error resilience of the video stream. Error concealment by post-processing at the decoder side to recover the damaged region based on property of video or image as shown in Figure 1.5(b). The last category is interactive error concealment between decoder and encoder. For the approach of error resilient, some error control algorithms have been adopted in H.264 [3], like flexible macroblock ordering and multiple reference frames. But they are mainly focused on source encoder and bring more complexity and delay at the same time. The forward error control coding techniques reduces the channel capacity and also has error recovery ability limitations. Automatic Retransmission reQuest (ARQ) increases the delay and is not suitable for the applications such as conversational and multimedia streaming services with constrains on real-time delay and jitter. For the low delay applications, an indispensable method is to perform error concealment as post-processing in the decoder side.

## *1.4 Thesis Organization*

This thesis is organized as follows. At first, the overview of the deblocking filter and review of error concealment are described in Chapter 2. Chapter 3 gives the details of the architecture design of deblocking filter for high-definition applications. Then we propose an in-loop/post-loop filter for multi-standard decoder based on the high-throughput design. Further, a new error concealment algorithm, error- concealed deblocking filter, which is based on the algorithm of deblocking filter standardized by H.264/AVC, is proposed for mobile applications in Chapter 4. Finally, the implementation details, summary and conclusion are presented in Chapter 5 and Chapter 6, respectively.

# Chapter 2
# Algorithms for Deblocking Filter and Error Concealment

## 2.1 Deblocking Filter for H.264/AVC

As we discussed in the previous chapter, the deblocking filter is applied to all 4x4 block edges of one frame due to the use the 4x4 DCT and possible 4x4-block size motion compensation. Based on the macroblock-based coding, the macroblocks in a picture are processed each by each in a raster scan. And for each macroblock, the deblocking filter process has two components, luma and chroma, separately. As shown in Figure 2.1, the luma edges are shown with solid lines and chroma edges are shown with dashed lines.

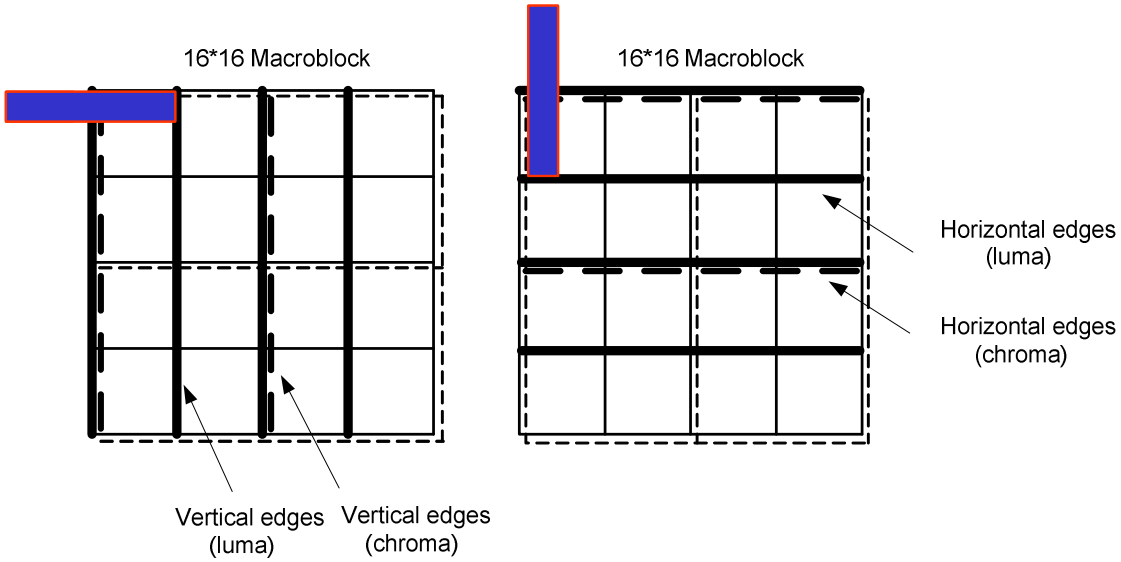



Figure 2.1: The edges applied to deblocking filter in one MB.

For each macroblock, vertical edges are filtered first, from left to right, followed by the horizontal edges from top to bottom. Deblocking filter is performed on four 16-sample

edges for luma and two 8-sample edges for each chroma as shown in Figure 2.1. Hence, the total number of samples applied by deblocking filter equals 192. The computational time of deblocking filter becomes longer for the applications with high resolution video.

### 2.1.1 Adaptivity of Deblocking Filter in H.264/AVC

As we mentioned in previous chapter, the complexity of deblocking filter is mainly based on the high adaptivity on each block edge and sample level. The deblocking filter in H.264/AVC is adaptive in several levels; on the block-edge level, boundary strength (BS) is determined according to the location of edge, intra/inter mode, frame/field mode, absolute difference of motion vectors, transform coefficient levels and so on. Then, the deblocking filter uses different 1-D filter to smooth the block edge adaptively according to BS. On the sample level, correlation in pixel domain and quantizer-dependent thresholds can turn off the deblocking filter for each sample. Last, on the slice level, encoder-selectable offsets can control the property of deblocking filter by reducing the amount of filtering or increasing the amount of filtering.

#### 2.1.1.1 Block-edge Level Adaptivity

As shown in Figure 2.2, for each edge between two 4x4 luma block, the BS is assigned by an integer from 0 to 4 according to the decoding information, such as macroblock edge, intra/inter mode, frame/field mode, absolute difference of motion vectors, transform coefficient levels and so on. As the BS is assigned to 4, the deblocking filter allows strongest filtering to eliminate the blocking effect due to Mach band effect in the boundary of intra-coded MB. The value of BS equals 0 means that the block edge between adjacent two blocks has no filtering. For BS equals 1, 2 or 3, the normal filter mode is applied. While the value of BS determine the maximum modification on sample value caused by filtering. The detail of correlation between BS and filtering is discussed in the section.

Actually, the decision of BS is more complex for the case of B slice type or

macroblock adaptive frame filed coding. For the case of B slice type, the motion vector comparison of B slice type consists of forward motion vectors and backward motion vectors. Besides, the decision of BS is also influenced by the of reference frame. For the case of macroblock adaptive frame/field coding (MBAFF), the frame/field mode of MB also has the influence on BS decision. The threshold of motion vectors also modified from 4 to 2 when the MB is coded as filed mode and the edge belongs to horizontal edge.

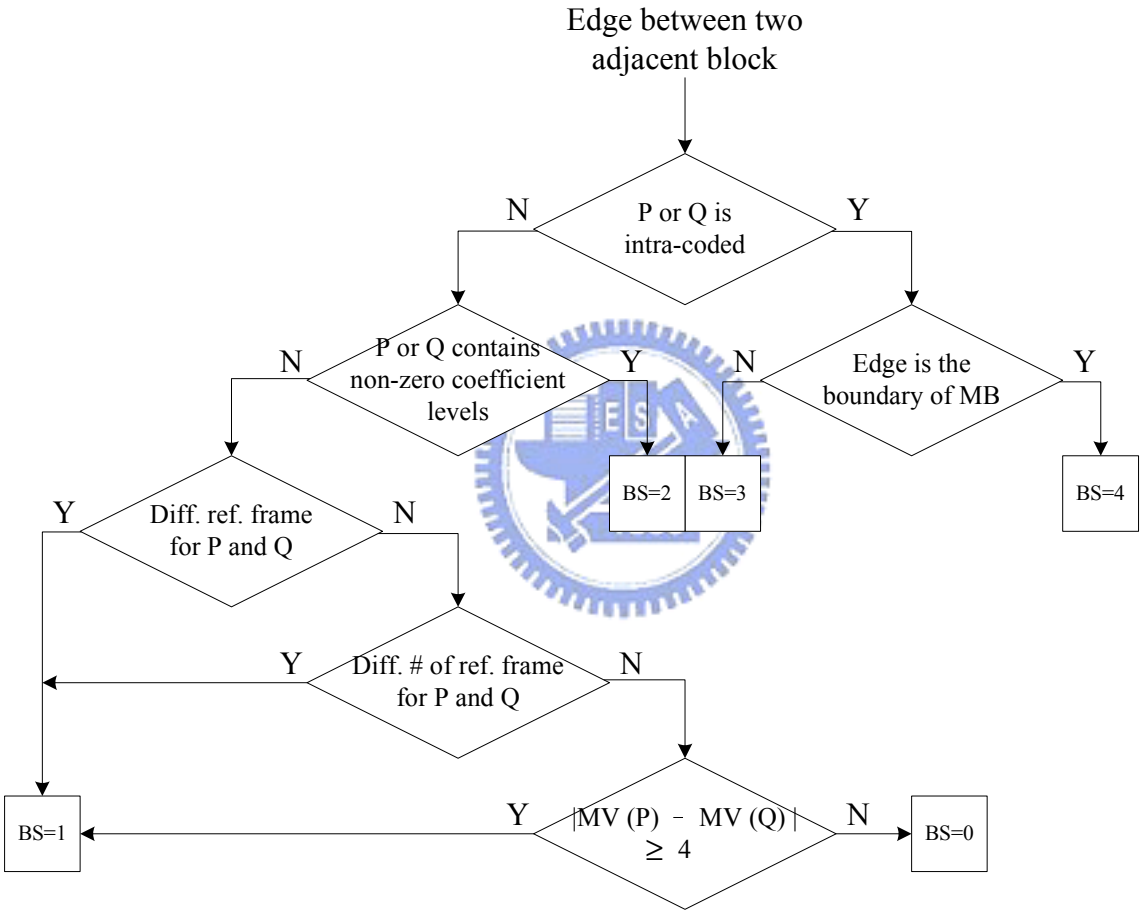



Figure 2.2: The simple vision decision flow of boundary strength without MBAFF.

### 2.1.1.2 Sample Level Adaptivity

Generally, the blocking effects do not exist everywhere in a frame. To preserve the sharpness in image, the ability of distinguishing the artificial edges from actual edges is necessary. Hence, the deblocking filter in H.264/AVC has the adaptivity on sample level using the image property in pixel domain and quantizer-dependent thresholds, α and β.

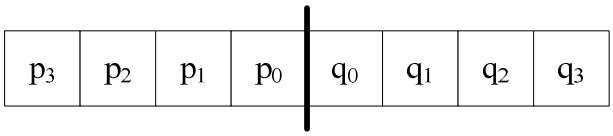



Figure 2.3: Samples across a 4x4 block boundary.

As shown in Figure 2.3, for the adaptivity on sample level, the neighboring pixels on each sample edge is loaded and analyzed with the quantizer-dependent thresholds. The sample edge is filtered only when the following conditions are true.

$$BS != 0$$
$$Abs(p_0 - q_0) < \alpha$$
$$Abs(p_1 - p_0) < \beta$$
$$Abs(q_1 - q_0) < \beta$$

The quantizer-dependent thresholds, α and β, are table-derived according to index values that dependent on the average quantization parameter (QP) over the edge and encoder selected thresholds, FilterOffsetA and FilterOffsetB, as shown in Eq. (2.1) and (2.2).

$$indexA = Clip3(0, 51, qP_{av} + FilterOffsetA) \quad (2.1)$$

$$indexB = Clip3(0, 51, qP_{av} + FilterOffsetB) \quad (2.2)$$

The clip3 operation limits the value of index from 0 to 51. With the index values, the thresholds are determined from the following table standardized by H.264/AVC.

Table 2.1: Look-up table for thresholds according to index values.

| | IndexA (for α) or inidexB (for B) | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| α | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 12 | 13 |
| β | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 4 | 4 | 4 |

Generally, the blocking effect is serious when the QP is larger, since the coding errors increases with QP. Thus, the thresholds also increase with QP (index valued is proportional to QP) to eliminate the blocking effect caused by block-based compression tools as shown in Table 2.1.

Table 2.1 (concluded): Look-up table for thresholds according to index values.

| | IndexA (for α) or inidexB (for B) | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 |
| α | 15 | 17 | 20 | 22 | 25 | 28 | 32 | 36 | 40 | 45 | 50 | 56 | 63 | 71 | 80 | 90 | 101 | 113 | 127 | 144 | 162 | 182 | 203 | 226 | 255 | 255 |
| β | 6 | 6 | 7 | 7 | 8 | 8 | 9 | 9 | 10 | 10 | 11 | 11 | 12 | 12 | 13 | 13 | 14 | 14 | 15 | 15 | 16 | 16 | 17 | 17 | 18 | 18 |

### 2.1.1.3 Slice Level Adaptivity

On the slice level, the encoder can decide the values of FilterOffsetA and FilterOffsetB to adjust the values of thresholds (see the Eqs. (2.1) and (2.2)), α and β, for different video contents. These offset values are transmitted in slice header to control the property of deblocking filter. As compared to the zero offset, the amount of filtering increases with the positive offset values or decreases with negative offset values. For the video with large amount of sharpness in detail, the encoder may select the negative offset values to preserve the sharpness in image, since there are fewer blocking effects for high-resolution video content. On the other hand, for the lower-resolution video, the encoder may choose the positive offset values due to the obvious blocking effect caused by coding errors to improve the subjective quality.

## 2.1.2 Filtering Process of Deblocking Filter in H.264/AVC

Two filtering modes are defined and selected according to BS that we discussed in previous section in H.264/AVC. The strongest mode is used when BS is assigned to 4; normal mode is used otherwise (BS = 1, 2 or 3).

### 2.1.2.1 Normal Mode of Deblocking Filter with BS Less than 4

Up to 2 pixel values at each side of edge can be filtered for luma and 1 pixel value for chroma as shown in Figure 2.4.

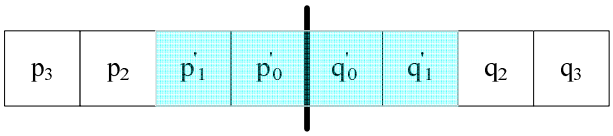



Figure 2.4: Filtered sample in normal filter mode of H.264/AVC.

The values of $p'_0$ and $q'_0$ are derived with two steps. First, the value of $\Delta$ is calculated as shown in Eq. (2.3). After that, the two neighboring pixels ($p_0$ and $q_0$) plus or subtract the value of $\Delta$ to eliminate the existing blocking effect as shown in Eq. (2.4) and Eq. (2.5).

$$\Delta = Clip3\left(-t_C, t_C, \left(\left(\left(q_0 - p_0\right) \ll 2\right) + \left(p_1 - q_1\right) + 4\right) \gg 3\right)\right) \tag{2.3}$$

$$p'_0 = Clip1\left(p_0 + \Delta\right) \tag{2.4}$$

$$q'_0 = Clip1\left(q_0 + \Delta\right) \tag{2.5}$$

The threshold of $t_C$ is derived as follows.

$$t_C = \begin{cases} t_{C0} + \left(\left(a_p < \beta\right)?1:0\right) + \left(\left(a_q < \beta\right)?1:0\right), & luma\ sample \\ t_{C0} + 1 & , otherwise \end{cases} \tag{2.6}$$

The threshold $t_{C0}$ is specified in Table 2.3 according to the values of index A and BS. And the two thresholds, $a_p$ and $a_q$, are derived as shown in Eq. (2.7) and Eq. (2.8).

$$a_p = Abs\left(p_2 - p_0\right) \tag{2.7}$$

$$a_q = Abs\left(q_2 - q_0\right) \tag{2.8}$$

Table 2.2: Value of filter clipping variable $t_{C0}$ as a function of indexA and BS.

| | indexA | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| BS = 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| BS = 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| BS = 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Table 2.2 (concluded): Value of filter clipping variable $t_{C0}$ as a function of indexA and BS.

| | indexA | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 |
| BS=1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 4 | 4 | 4 | 5 | 6 | 6 | 7 | 8 | 9 | 10 | 11 | 13 |
| BS=2 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 4 | 4 | 5 | 5 | 6 | 7 | 8 | 8 | 10 | 11 | 12 | 13 | 15 | 17 |
| BS=3 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 4 | 4 | 4 | 5 | 6 | 6 | 7 | 8 | 9 | 10 | 11 | 13 | 14 | 16 | 18 | 20 | 23 | 25 |

For the normal filter mode, deblocking filter has spatial adaptivity here according to the pixel values and threshold β. The filtering of $p_1$ and $q_1$ are applied when the value of $a_p$ or $a_q$ is less than the threshold β in the case of luma sample separately and the equations used for $p_1$ and $q_1$ are almost the same as the equations used for the pixels, $p_0$ and $q_0$, as shown in Eq. (2.9) and Eq. (2.10).

$$p'_1 = p_1 + Clip3\left(-t_{C0}, t_{C0}, \left(p_2 + \left((p_0 + q_0 + 1) \gg 1\right) - (p_1 \ll 1)\right) \gg 1\right) \tag{2.9}$$

$$q'_1 = q_1 + Clip3\left(-t_{C0}, t_{C0}, \left(q_2 + \left((p_0 + q_0 + 1) \gg 1\right) - (q_1 \ll 1)\right) \gg 1\right) \tag{2.10}$$

As we discussed in previous section, the normal filter mode of deblocking filter smoothes the artificial edges with a modification value (Δ) when BS is less than 4. And from the Table 2.2, we can see the maximum modification value is quite small when QP is less than 40. Besides, only up to 4 pixels are filtered per block edge in sample level.

### 2.1.2.2 Strong Mode of Deblocking Filter with BS Equals 4

The filtering process uses strong filter mode in H.264/AVC when BS equals 4. From Figure 2.5, at most 6 pixels are modified per edge in case of luma sample or 2 pixels are modified in case of chroma sample.

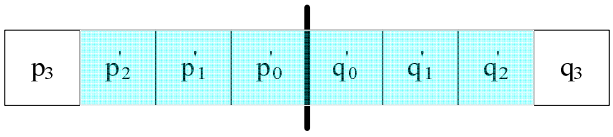



Figure 2.5: Filtered samples in strong filter mode in H.264/AVC.

The methods used for filtering processing with strong filter mode in H.264/AVC are similar to weighting averaging method. The values of filtered pixels equal the average of

neighboring pixels multiply specified weightings as shown in Eqs. (2.11) to (2.16).

$$p'_0 = \begin{cases} (p_2 + 2\times p_1 + 2\times p_0 + 2\times q_0 + q_1 + 4) \gg 3, & Eq.\ (2.17)\ is\ true\ in\ luma \\ (2\times p_1 + p_0 + q_1 + 2) \gg 2, & otherwise \end{cases} \quad (2.11)$$

$$q'_0 = \begin{cases} (q_2 + 2\times q_1 + 2\times q_0 + 2\times p_0 + p_1 + 4) \gg 3, & Eq.\ (2.18)\ is\ true\ in\ luma \\ (2\times q_1 + q_0 + p_1 + 2) \gg 2, & otherwise \end{cases} \quad (2.12)$$

$$p'_1 = \begin{cases} (p_2 + p_1 + p_0 + q_0 + 2) \gg 2, & Eq.\ (2.17)\ is\ true\ in\ luma \\ p_1, & otherwise \end{cases} \quad (2.13)$$

$$q'_1 = \begin{cases} (q_2 + q_1 + q_0 + p_0 + 2) \gg 2, & Eq.\ (2.18)\ is\ true\ in\ luma \\ q_1, & otherwise \end{cases} \quad (2.14)$$

$$p'_2 = \begin{cases} (2\times p_3 + 3\times p_2 + p_1 + p_0 + q_0 + 4) \gg 3, & Eq.\ (2.17)\ is\ true\ in\ luma \\ p_2, & otherwise \end{cases} \quad (2.15)$$

$$q'_2 = \begin{cases} (2\times q_3 + 3\times q_2 + q_1 + q_0 + p_0 + 4) \gg 3, & Eq.\ (2.18)\ is\ true\ in\ luma \\ q_2, & otherwise \end{cases} \quad (2.16)$$

Adaptivity is applied in spatial domain with strong filter mode here. For the two pixels, $p_0$ and $q_0$, which are closest to the edge, there are two weighting sets used according to the pixels characters and thresholds, α and β. Further, the other four pixels ($p_1$, $p_2$, $q_1$ and $q_2$) are filtered only when the Eq. (2.17) or Eq. (2.18) is true separately.

$$a_p < \beta \,\&\&\, Abs(p_0 - q_0) < ((\alpha \gg 2) + 2) \quad (2.17)$$

$$a_q < \beta \,\&\&\, Abs(p_0 - q_0) < ((\alpha \gg 2) + 2) \quad (2.18)$$

In summary, we can see adaptivity in variable levels and different domains. The adaptivity in slice level, 4x4 block size edge level and sample level determine if the edge is artificial edge or actual edge and which filter mode is used according to BS. There are different equations used to filter the pixels across the edge even the filter mode is determined. That is the major reason for high complexity of deblocking filter in H.264/AVC and it contributes about one-third of the computational complexity of the decoder.

## 2.2 *Error Concealment*

For a compressed video which uses variable length coding and some predictive coding, an error of one bit not only changes values of pixels, but also makes a lot of trouble for following decoding due to unexpected value of syntax elements. Therefore, a great degradation of video quality will be resulted. This may lose a macro block, a slice, even a total frame. The higher compression of a video standard has, the more serious problem is. In order to solve this problem, lots of error concealment algorithms have been proposed either in the spatial domain or temporal domain. The basic concept of error concealment is to predict the lost MB by the spatially neighboring MBs due to the observation of high spatially correlation in small areas either MB's motion vectors or pixels. By doing so, the lost MBs will be reconstructed and the quality of video will be improved.

The main different features between H.264/AVC and the previous coding standards are the motion estimation scheme and the multiple reference frame mode as shown in Figure 2.6. Unlike the previous coding standards, the motion estimation has seven different block division sizes.

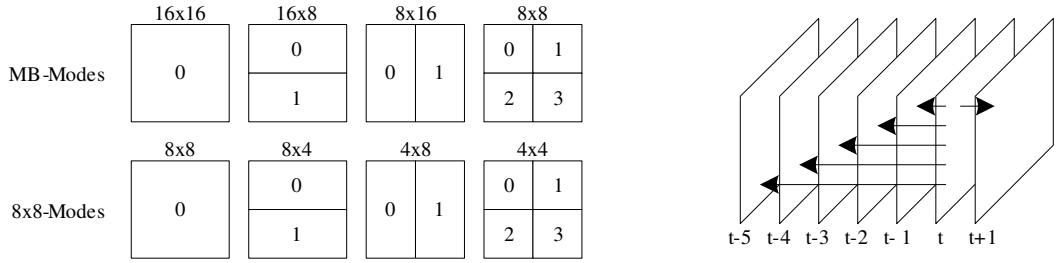



Figure 2.6: Variable block size for motion vectors and multiple reference frames.

Due to the variable block size of motion estimation and multi-direction intra prediction, the algorithms of temporal and spatial error concealment used for the previous standards is not suitable for the H.264/AVC any more. Therefore, some modifications of error concealment algorithms have been proposed recently. The review of error concealment for video is presented in the following section.

In this section, some algorithms are shortly described. And comparison of complexity and performance between these algorithms will be described. Based on the continuity in one frame, the lost pixels will be reconstructed by the spatially neighbor pixels with some weighting. And it's the same as motion vectors. According to the statistical observation that motion vectors of spatially neighboring areas are highly correlated, the motion vector of a lost MB can be easily predicted from a spatial neighbor MB's motion vectors.

### 2.2.1 Spatial Error Concealment Algorithms

The error concealment primarily uses the high correlation of neighboring pixels in spatial domain to reconstruct the corrupted pixels. The algorithms widely used in spatial error concealment (SEC) are bilinear interpolation (BI) and directional interpolation (DI). Generally, BI is suitable for texture due to the property of smooth, and DI has better performance if there are real edges exist in the corrupted MB. Hence, an algorithm which combines the DI and BI with mode decision has been presented. It conceals MB with BI to avoid the creating false edges or with DI to preserve the real edges according to the directional entropy of neighboring edges.

#### 2.2.1.1 Bilinear interpolation

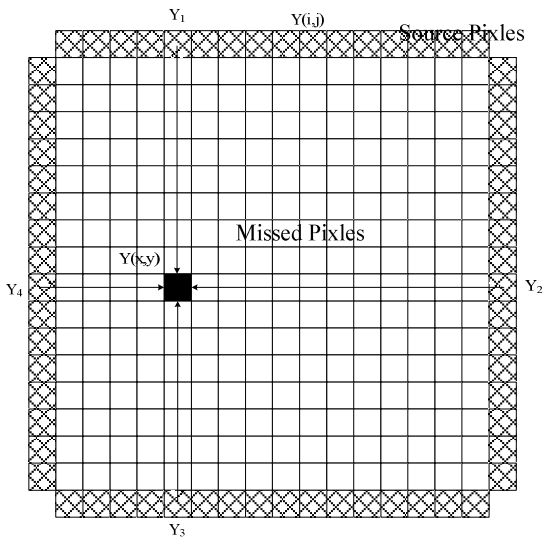



Figure 2.7: Bilinear interpolation

As shown in Figure 2.7, BI replaces each missed pixel with the weighting average of nearest pixels $(Y_1, Y_2, Y_3, Y_4)$ in the neighboring MBs with four directions. The weights they used here are inversely proportional to distance of the missing and raw pixels, $d_{(i,j)\to(x,y)}$ as shown in Eq. (2.19). And only "Correctly received" neighboring MBs are used for concealment if at least two such MBs are available. Otherwise, neighboring "Concealed" MBs are also used in the averaging operation.

$$Y(x,y) = \frac{\sum_{(i,j)\in N} Y(i,j) \times \left[15 - d_{(i,j)\to(x,y)}\right]}{\sum_{(i,j)\in N} d_{(i,j)\to(x,y)}}, N = \{Y_1, Y_2, Y_3, Y_4\}, (x,y) \in \textit{lost intra MB} \quad (2.19)$$

### 2.2.1.2 Directional Interpolation

DI consists of edge detection, edge direction ranking and 1-D interpolation [4]. In the first step, the edge detection uses Sobel mask to determine the gradient and edge direction in the neighborhood of a corrupted MB. With the information, edge direction ranking would determine the possible edge through the corrupted MB. Finally, the 1-D interpolation along the specified direction is applied to conceal the lost MB with source pixels and weights that is inversely proportional to the distance of missing and raw pixel as shown in Figure 2.8.

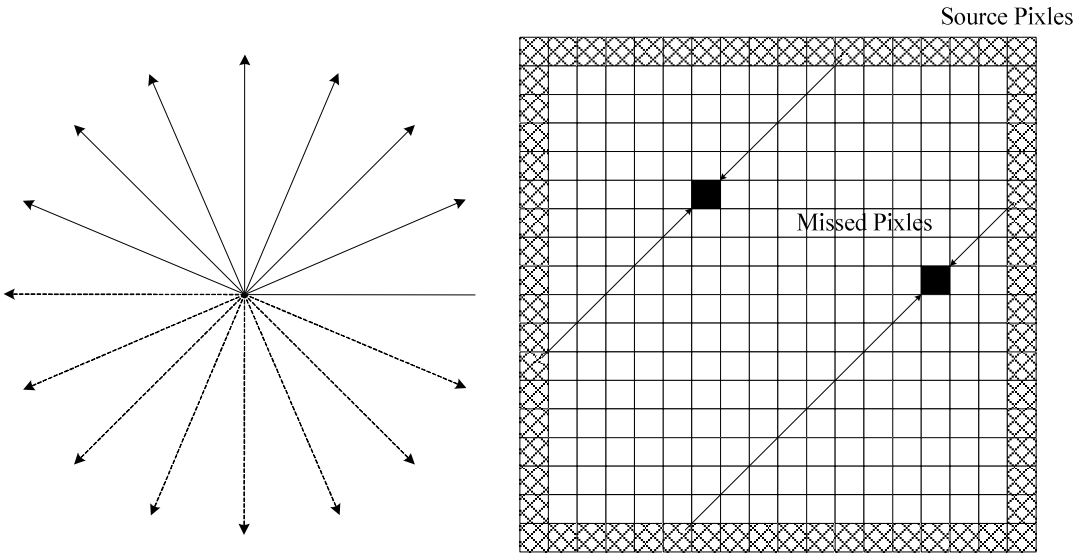



Figure 2.8: Direction interpolation.

### 2.2.1.3 Interpolation with Mode Decision

In order to preserve the real edges exist in corrupted MB but avoid the false edge creating, the hybrid interpolation method is presented [5]. A complex algorithm combines the DI and BI with mode decision to reconstruct the corrupted MB more correctly.

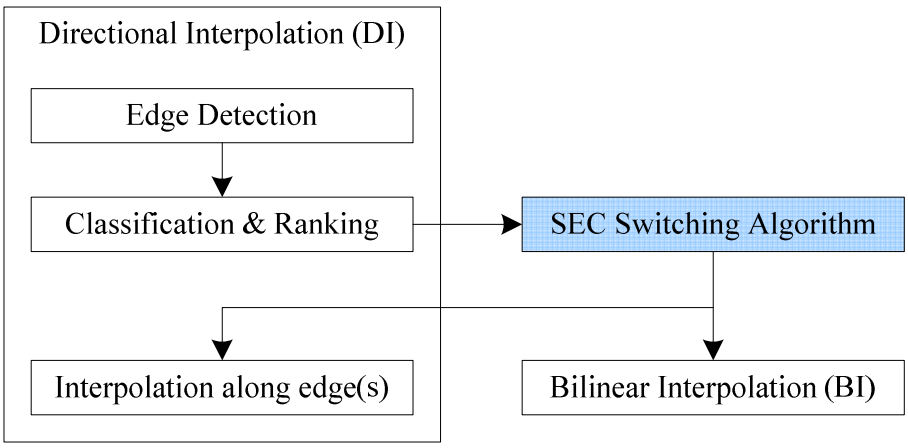



Figure 2.9: Block diagram of the proposed SEC approach. [5]

The SEC switching algorithm used the entropy of the edge direction data in the neighboring MBs which are provided by the edge detection component of the DI to switch the algorithm used between DI and BI. Generally, there is no specific edge direction when the directional activity is large since edges of different directions interact with each other.

## 2.2.2 Temporal Error Concealment Algorithms

For the corrupted MB with inter coding, the efficiency of error concealment is usually better than the intra-coded MB due to the reference frame. Generally, the main purpose of temporal error concealment (TEC) is to estimate the proper motion vector (MV) for the corrupted MB to reconstruct with pixels in reference frame. The estimated MV may be zero, the MVs of an adjacent correctly received MB, average or median of all such available adjacent MVs. Here we make a review of temporal error concealment algorithms widely used for previous video standards.

### 2.2.2.1 Temporal Replacement

Temporal replacement is the lowest complex in temporal error concealment algorithms. The process of temporal replacement is quite simple. Each lost MB is just the copy of the spatially corresponding MB in the previous frame. It means that the lost MB's motion vector is zero.

### 2.2.2.2 Boundary Matching Algorithm

Instead of using the zero, average or median over all neighboring MB's MVs, the Boundary Matching algorithm uses the MV which belongs to one of neighbor MB's MV to make the block boundary smooth as shown in Figure 2.10 [6]. The winning prediction MV is the one which minimizes the side match distortion $d_{sm}$. As shown in Eq. (2.20), $d_{sm}$ is the sum of absolute Y pixel value differences of the IN-block and OUT-block pixels.

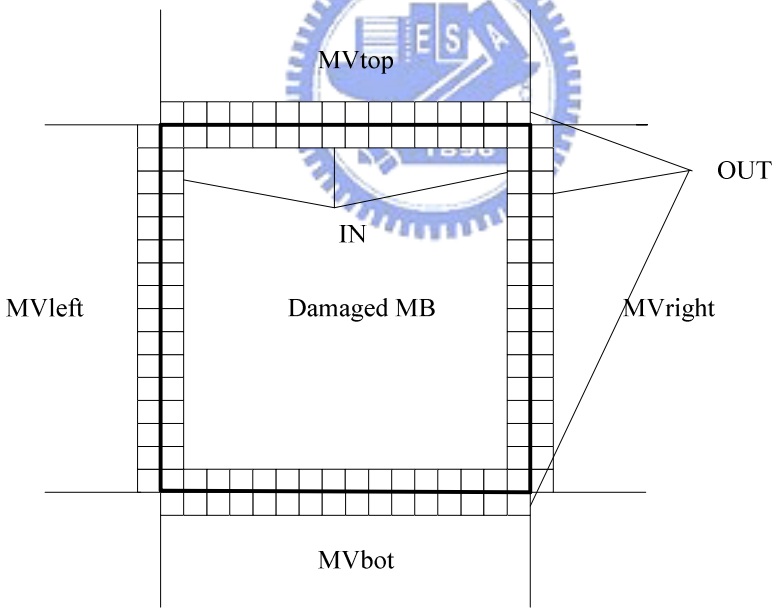



Figure 2.10: Boundary matching algorithm.

$$\min\arg_{dir\in\{top,bot,left,right\}} = \left\langle d_{sm} = \frac{1}{N}\sum_{j=1}^{N}\left|\hat{Y}(mv^{dir})_j^{IN} - Y_j^{OUT}\right| \right\rangle \tag{2.20}$$

Generally, the algorithm of boundary Matching can be extended to variable block size due to the variable block size used for motion vectors in H.264/AVC.

### 2.2.2.3 Motion Vectors Recovery Based on Polynomial Model

The MVs have some correlation in small area as the same as the correlation pixels have. To describe the change tendency of the MVs within a small area, the polynomial model is used to result in an approximate function. Hence, we can estimate the lost MVs in corrupted MB with the polynomial function which is used to describe the correlation of MVs in the small area as shown in Eq. (2.21)

$$P_m(x) = a_0 + a_1 x + \cdots + a_m x^m \tag{2.21}$$

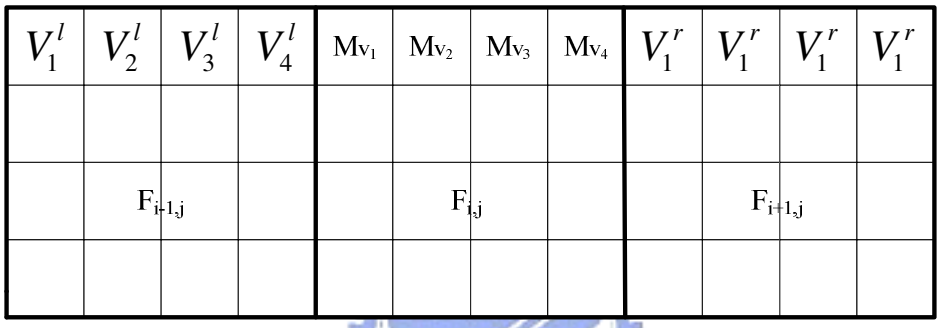



Figure 2.11: Motion vectors across the neighboring and corrupted MB in one-dimensional.

With the MVs in one-dimensional and corresponding coordinates listed in Table 2.3, we can calculate the unknown coefficients $(a_0, a_1, \ldots, a_m)$ to establish the Eq. (2.21) to describe the correlation of those correct MVs ($V_n^l, n = 1,2,3,4$ or $V_n^r, n = 1,2,3,4$) as close as possible. Hence, we can estimate the lost MVs ($Mv_n, n = 1,2,3,4$) with approximate correlation function more correctly.

Table 2.3: Corresponding coordinates of motion vectors.

| $y_i$ | $V_1^l$ | $V_2^l$ | $V_3^l$ | $V_4^l$ | $Mv_1$ | $Mv_2$ | $Mv_3$ | $Mv_4$ | $V_1^r$ | $V_2^r$ | $V_3^r$ | $V_4^r$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_i$ | -22 | -18 | -14 | -10 | -6 | -2 | 2 | 6 | 10 | 14 | 18 | 22 |

Besides, there are a lot of TEC algorithms presented due to the coding tools in H.264/AVC. The multi-frame TEC approaches have also been reported that used more than one past/future frame to conceal with the cost of increased complexity and delay.

### 2.2.3 Summary

So far, many researches focus on EC for the corrupted video bit-stream over an error-prone or wireless channel. Most of them have assumption that erroneous or incomplete slices are not decoded but discarded before decoding to simplify the simulation of proposed algorithm. The EC can be divided into two groups: spatial and temporal error concealment (SEC and TEC). The SEC exploits the neighboring correct pixels to recover the corrupted MB with the property of high correlation between the closing pixels. This kind of EC is used for corrupted MBs in I frames or I MBs in P frames. Unlike the SEC, TEC is explored to estimate the lost motion vectors (MV) in corrupted MBs and usually have better concealing efficiency than SEC.

Generally, the error scheme for video standards uses SEC and TEC adaptively according to the MB type. However, TEC is not always adequate for concealing errors in video sequences. This is especially true for video sequences with irregular motion, abrupt scene changes, and intra-coded image frames. To the viewer, poor spatial concealment of error regions leads to the error propagation in the subsequent frames.

Hence, compared to temporal EC, spatial EC is of great importance and challenges. Thereafter, we focus on the spatial EC for concealing corrupted region of image frames.

# Chapter 3
# Design of High Throughput Deblocking Filter

## 3.1 Design for High Throughput and High Resolution Applications

H.264/AVC is the newest video coding standard of Joint Video Team (JVT) [7]. H.264/AVC has achieved significant rate-distortion efficiency by many useful tools. Deblocking filter placed in the prediction loop is one important tool to remove the blocking artifacts. Generally, the deblocking filter contributes about one-third of the computational complexity of the decoder [1], and it's the system bottleneck in terms of processing cycles. Compared to the loop filters in H.263 or MPEG-4/H.263 post filters [8], the deblocking filter in H.264 operates each filter process on 4x4 block structure instead of 8x8 block structure. Therefore, large amount of computation and memory access are its penalty for the real-time decoding demand.

First, we modify the processing order of filtered block boundaries without affecting the data dependency to reduce the number of memory access. In the deblocking filter of H.264/AVC, vertical edges are filtered first from left to right, and then horizontal edges are filtered from top to bottom. For each edge, 4x4-block data is read from memory four times and wrote back to memory four times with separate filtering order. To reduce the memory access and the processing cycles, we propose a hybrid filtering method to re-schedule the filter ordering and reuse the pixel value on the different directions.

Besides, we make a decision of our memory organization between prediction unit and deblocking filter in system level. Instead of LOP in [9], we exploit the block ordering which is standard-defined to decide the Column-of-Pixel (CoP) memory organization. We can reuse the neighboring data in intra or inter prediction unit and reduce the number of memory access by using CoP.

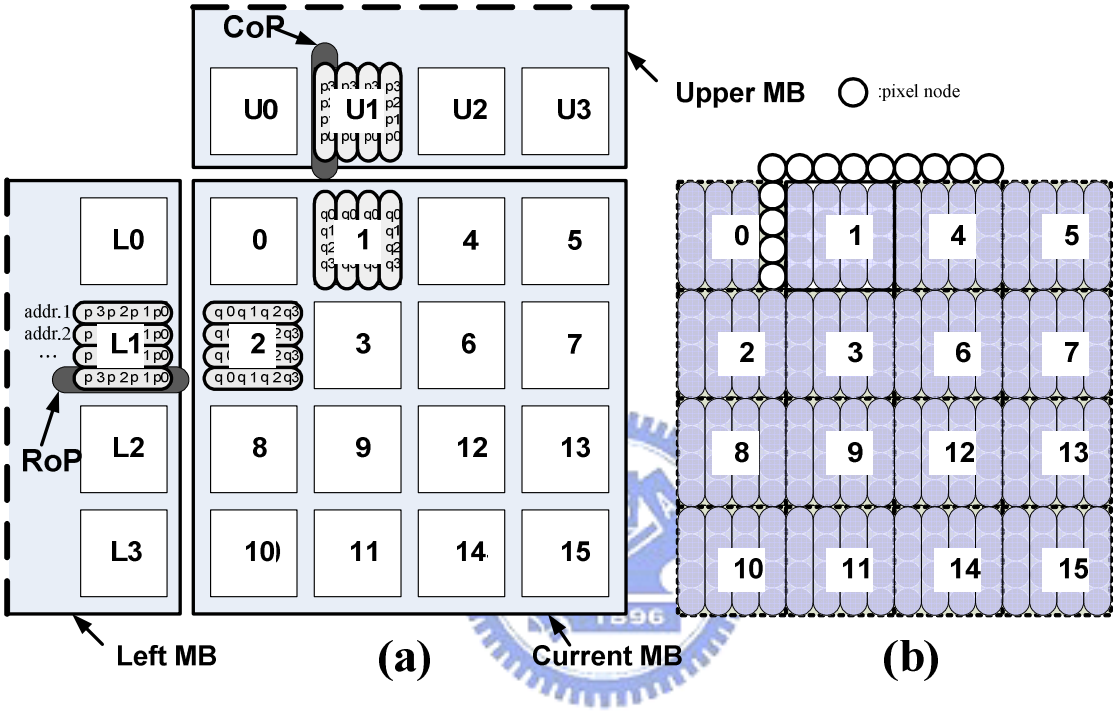



Figure 3.1: The different data arrangement in the deblocking filter (a) and prediction unit (b).

### 3.1.1 Memory Organization between Prediction and Loop-filter

Different memory organization leads to the different memory access and processing latency. The input data of deblocking filter is just the output data of prediction unit, and plus the residual data. To improve the overall processing throughput, we use two content memories between prediction and deblocking filter. Hence, the video decoder can process in MB-level. Further, we apply slice memory to store neighboring data to avoid accessing the frame buffer out of system.

#### 3.1.1.1 Memory Organization

We use one Column-of-Pixel (CoP) as the data word size in each memory address. In Figure 3.1(a), there are two policies for data arrangement. The Row of Pixel (RoP) is labeled with the case of L1 and 2 blocks, and the Column of Pixel (CoP) is in the case of U1 and 1 blocks. Each row or column of pixel contains four pixels with a total of 32-bit wide. For the deblocking filter, RoP (i.e. LOP in [9]) is a straightforward method to arrange the pixel value in the vertical edge filtering. However, it will induce extra memory access when applying to the horizontal edge filtering. By the same way, this situation is also occurred in the CoP arrangement. Different arrangements of CoP or RoP also affect the number of memory access in the intra or inter prediction unit. In Figure 3.1(b), the standard-defined 4x4 block ordering is labeled in each block. There are strong dependencies in the horizontal block order. Therefore, we choose CoP data arrangement to reuse the pixel value in the block-boundary with white-circle region. Further, we list the hardware profiling in terms of memory access in Table 3.1. The evaluated cycles with CoP or RoP data arrangement are almost the same in the deblocking filter unit. The reason is that the filtering process will be performed on not only horizontal edge but also vertical edge. However, there are improvements in the prediction unit when applying the CoP arrangement. Therefore, compared to the RoP data arrangement, we choose the CoP data arrangement in each MB to reduce the number of memory access.

Table 3.1: The analysis of average memory access per luma MB.

| # of memory access | | | |
|---|---|---|---|
| **Memory Arrangement** | Intra Prediction | Inter Prediction | De-blocking Filter |
| **CoP** | **40** | **313** | **156** |
| **RoP** | **48** | **432** | **156** |
| **Improvement (RoP-CoP)/RoP** | **17%** | **28%** | **0%** |

### 3.1.1.2 Slice and Content Memory

The content memory is used to store the unfiltered pixel value in luma or chroma block. The data word-length of memory is based on the 32-bit of CoP, and the address depth of content memory is decided by the YUV format (4:4:4, 4:2:2 or 4:2:0). For 4:2:0 format, there are 16 blocks of luma and 8 blocks of chroma should be stored. Therefore, the size of content memory is (16+8)*4 × 32 in total. Further, the data address is increased as the standard-defined block ordering of Figure 3.2(b). The grid region is stored in the slice memory and the dotted region is stored in the content memory.

The slice memory is used to store the neighboring pixel. To reduce the pin counts and avoid to access frame buffer, we keep the blocks which are not filtered completely. Further, the address depth is decided by the frame width. In Figure 3.2(a), considering the frame size with M×N, each square represents the 16x16 MB. Each MB contains the 16 points, and 4x4 pixels within each point. When the filtering process is performed from the MB index of *B* to *B+1*, the pixel data within upper and left neighbor will be updated as the arrows show. The shaded region should be kept when the filtering index is *B+1*. Therefore, the slice memory is used to keep the pixel value of upper and left neighbor and contains the size of (2N + 20) × 32 for the 4:2:0 format.

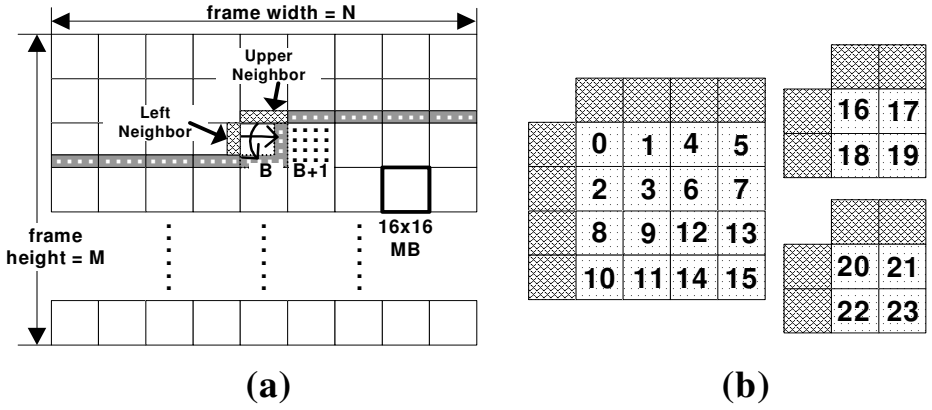



Figure 3.2: The slice memory with grid or shaded region and the content memory with black-dotted region.

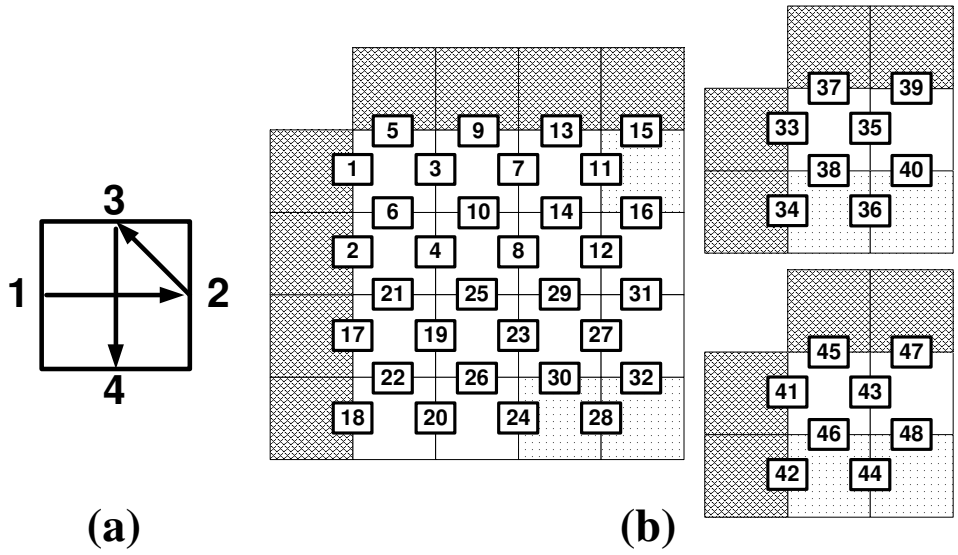



Figure 3.3: The proposed hybrid scheduling method.

### 3.1.1.3 Hybrid Scheduling

To reduce the overhead with the reloaded data when switching the filtering edge from horizontal to vertical, we propose a hybrid filter scheduling to re-schedule the standard-defined edge. The deblocking filter in H.264/AVC is performed in the vertical edge first, and then the horizontal edge. Based on the standard-defined filter ordering, we can deduce the filter order on each 4x4 block as Figure 3.3(a). In the filter ordering of one 4x4-block, left edge is filtered first and lower edge is the last one. We propose a novel filter ordering to schedule our filter process on each edge as Figure 3.3(b). Each filter order of one block obeys the rules of the left edge first and the lower edge last. Compared to the traditional scheduling [9][10], our proposed method prevents the re-access for different direction and combine the vertical and horizontal filter at the rule of standard-compliance.

We use four 4x4 pixel buffer to keep the temporary data in our hybrid scheduling process. In Figure 3.4(a), each MB has been partitioned into two main parts (i.e. *Loop Filter-MB-Upper* or *Lower*) to reduce the kept buffer size. Each part is composed of eight time-instances to process the filtering procedure in Figure 3.4(b). The grid region represents the neighboring block and the shaded region is the position of kept data buffer with the size

of four 4x4 blocks. There is no need to keep the neighboring block as the data buffer in each time instance (except for the initial state *t1* since we use the CoP data arrangement) because the neighboring block and current MB are located at different memory module. Both data of them can be accessed at the same time instance and sent to the input of edge filter.

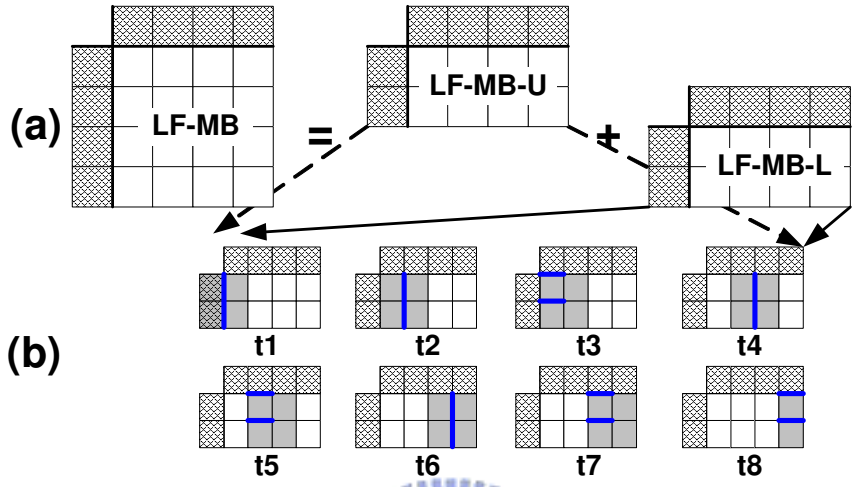



Figure 3.4: The partitioned MB and each time instance when applying the hybrid scheduling method.

We derived the filter ordering of the proposed hybrid scheduling method in Figure 3.4(b). Each bold line represents the edge to be filtered in each time instance. The filtered ordering complied with the hybrid scheduling in Figure 3.3(a) at each time instance (t1~t8). By the same way, the proposed scheduling is also performed in the 4x4-block of chroma.

The main problem of the deblocking filter in H.264/AVC is the considerable amount of memory access and processing cycles. To apply the proposed hybrid scheduling into the overall system and enhance the system throughput, we propose a high-throughput architecture design of deblocking filter.

### 3.1.1.4 Proposed Architecture of Deblocking Filter

Figure 3.5 shows the proposed design with block diagram and data flow representation. The size and organization of content and slice memory have been presented in previous section. We choose CoP memory arrangement to improve the pixel data utilization and

reduce the memory access in the prediction unit. The external frame buffer is an off-chip memory, and the size is decided by the frame size and the frame number for the long-term prediction. The shaded-arrows denote the data flow inside the deblocking filter unit, and the black-arrows denote the data flow outside. The pixel buffer is used to store the intermediate pixel value when applying the proposed hybrid scheduling. It contains the four 4x4 pixel values. Moreover, in each time instance, it locates at the position as the shaded regions of Figure 3.4(b) shows. The edge filter is a simple parallel in and parallel out process. It exploits the 3, 4 or 5-tap filter to attenuate the blocking artifacts due to the motion compensation or prediction error coding in each block boundary. More detailed algorithms are described in [1].

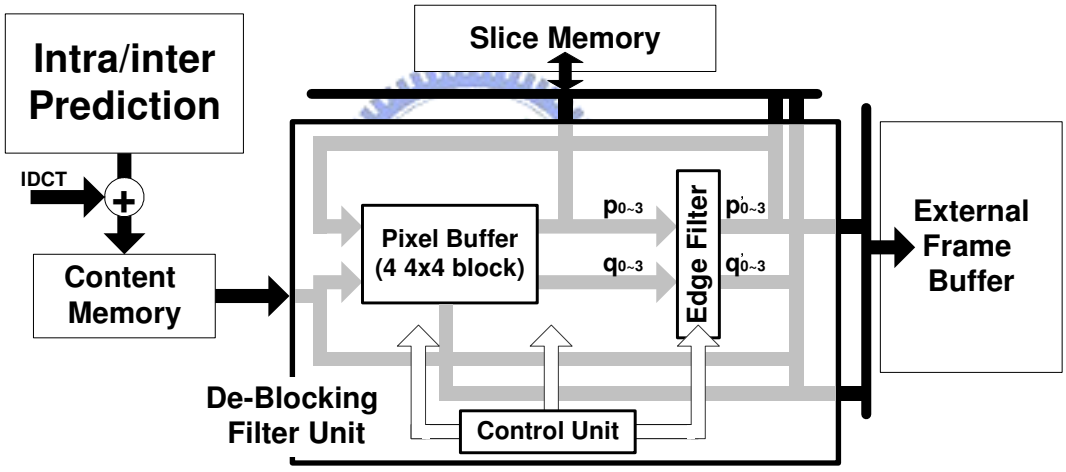



Figure 3.5: The block diagram and data flow of the proposed design.

The detailed architecture for the deblocking filter unit of Figure 3.5 has been shown in Figure 3.6. All the data signals are 32-bit wide and contain the LoP of memory organization discussed in previous section. There are four input signals {*wt_B_0*, *wt_B_1*, *wt_B_2*, *wt_B_3*} to write the buffers with 4 blocks. Further, there are three output signals {*rd_B_0*, *rd_B_1*, *rd_B_2*} to read three of them to perform the edge filter, and then write to the frame buffer, pixel buffer or slice memory. In addition, the write result of the 4 blocks is shown in Figure 3.4(b) to achieve the hybrid filtering and avoids the extra access from the filtering of different direction. By the same naming rule, each data flow represents the

writing/reading to/from the storage module including slice memory, content memory or frame buffer.

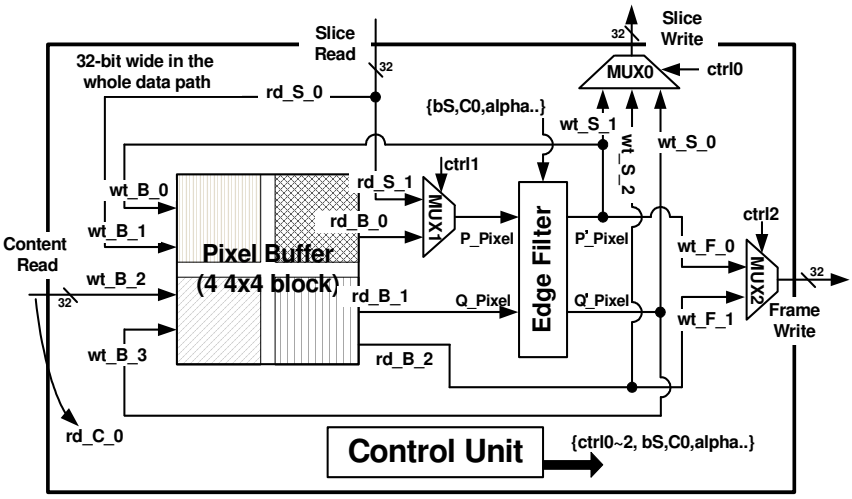



Figure 3.6: The detailed architecture for the deblocking filter.

After the behavioral illustration of pixel buffer, we use one MB with 48 edges in Figure 3.3(b) as an example to illustrate the other behavior of Figure 3.6. The behavior of Figure 3.6 can be partitioned into two main parts.

***Write Process*** is a writing mechanism through the signal {*wt_S_0~2*, *wt_F_0~1*, *wt_B_0~3*}.

***Read Process*** is a reading mechanism through the signal {*rd_S_0~1*, *rd_C_0*, *rd_B_0~2*}.

For writing to slice memory, *wt_S_0* is used to write the filtered data into the slice memory, and it will be activated only on the edge 6, 10, 14 and 16 (see Figure 3.3(b)). For the edge 6, the lower block will become the next neighboring block of LF-MB_L in Figure 3.4(b). The same condition is also applied on the edge 10, 14 and 16. Further, the *wt_S_1* will be activated on the edge 31, 32, 40 and 48. The *wt_S_2* is performed to write the dotted block data of Figure 3.3(b) into the slice memory. For the writing signal of frame buffer, *wt_F_0* is used to write filtered data into the external frame buffer. It will be activated on each filtering of horizontal edge except for the edge of activated signal *wt_S_1* and *wt_B_0*,

since *wt_F_0*, *wt_S_1* and *wt_B_0* have the same root-signal of *P'_Pixel*. For the edge 6 as an example, the upper block of edge 6 is the *P'_Pixel* of edge filter's output. This block will write to the external frame buffer since it has been filtered completely for all the edges of {1,3,5,6}. The *wt_F_1* is performed in the same way except that the input signal comes from the output of pixel buffer.

For the reading process of slice memory, *rd_S_0* is only activated on the edge of {1,2,17,18,31,33,34,39,41,42,47}. For the edge 1, the *rd_S_0* is the input of pixel buffer. We need to keep the pixel value since we apply the CoP arrangement of each data. That's why we keep the left neighboring as the pixel buffer in the *t1* of Figure 3.4(b). However, for the vertical filtering of edge {5,9,13,15,21,25,29,37,45}, it can directly feed through the edge filter by *rd_S_1*. Finally, compared to the existing approach, the content memory of our proposed design is only used for read. There is no need to store the filtered result into the content memory in one direction, and read them in another direction. By our proposed hybrid scheduling, we combine the horizontal and vertical filtering process in one filtering flow. Therefore, we need 4 blocks at most to perform the hybrid filtering.

### 3.1.2 Simulation Results

In Table 3.2, based on the proposed architecture for high throughput deblocking filter, the evaluated cycle counts are 148 cycles for Luma block and 88 cycles for chroma block. Specifically, we need 8 cycles (*LF-MB-U* + *LF-MB-L*) in the initial states. Further, there are 4x32 cycles to filter each horizontal and vertical edge in one luma MB. Finally, we need 20 cycles to write the filter result for the edge {16,30,32} and incur 3 cycles due to the data hazard in our filtering process. In sum, we need 148 (i.e. 8+4x32+12) cycles to filter horizontal and vertical edge of luma MB. By the same analysis, we need 88 (i.e. 4+4x8+8+=44 for each chroma) cycles to filter the horizontal and vertical edges in chroma block. Therefore, there are total 243 cycles with extra 7 cycles for data hazard.

Table 3.2: The cycle analysis within the deblocking filter unit.

| Cycle Counts | | [ 10 ]'s basic | [ 9 ] | Proposed |
|---|---|---|---|---|
| Vertical / Horizontal | | Seperated | Seperated | Hybrid |
| Luma | Horizontal<br>Vertical | 128<br>200 | 104<br>110 | 148 |
| Chroma | Horizontal<br>Vertical | 64<br>112 | N/A<br>N/A | 88 |
| Total | | 504 | 214 + N/A | 236+7 |

The simulation results are summarized in Table 3.3. The target technology is 0.18 $\mu$ m, and the synthesized gate count is 19.64K excluding the slice and content memory. Two single port SRAM is organized to store the content of YUV data and the neighboring data. They contain the size of 96x32 and (2N+20)x32 where N means the width of the coded frame.

Table 3.3: The comparison of architecture and processing cycles for the deblocking filter in H.264/AVC.

| Items | [ 10 ] | [ 9 ] | Proposed |
|---|---|---|---|
| Design Methodology | Shift- register based design | Line- buffer based design | Line- buffer based design |
| Kept Data Size | 2 blocks | 4 blocks | 4 blocks |
| Gate Count | 18.91K (0.25um) | N/A | 19.64K (0.18um) |
| Working frequency | 100 MHz | N/A | 100 MHz |
| Processing cycles per MB | 878 cycles/MB | 214 cycles/lumaMB + N cycles/chroma-MB | 243 cycles/MB = 148 cycles/luma MB + 88 cycles/chroma-MB + 7 data hazard |
| Memory Requirement | 2 single- port SRAM (basic architecture) | N/A | 2 single- port SRAM |

We use "foreman" as our test sequence. The evaluated cycle counts per MB are 243 cycles. Further, compared with the existing approach [9][10], our proposed architecture can save about one-half of processing cycles per MB. We use the hybrid-scheduling scheme to combine the horizontal and vertical filtering process, and slightly increase gate counts to keep the intermediate pixel value. With a pipeline methodology, Figure 3.7 shows the average processing cycles per one 16x16 MB with the decoding of first frame. Originally, the deblocking filter is a system bottleneck in terms of processing cycles. Based on the

proposed architecture, we can greatly reduce the processing cycles and improve the system throughput (i.e. 350cycle/MB = 9523MB/frame with 30fps@100MHz). Therefore, this processing capability can real-time decode 1080HD (1920x1088, i.e. 8160MB/frame) or higher with 4:2:0 format when the working frequency is 100MHz.

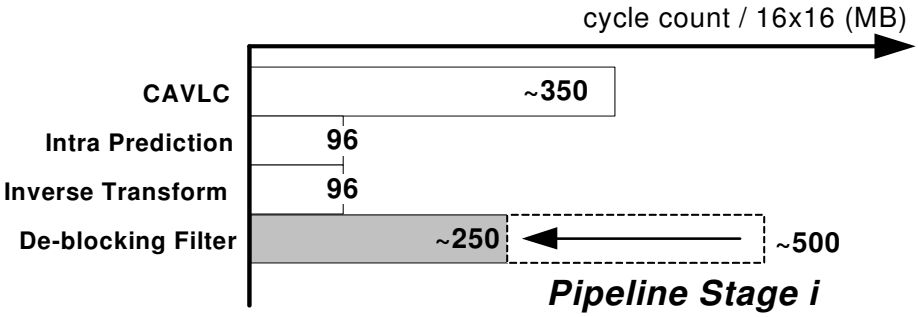



Figure 3.7: The average processing cycles in pipeline stage i.

### 3.1.3 Summary

We present a Column-of-Pixel (CoP) memory arrangement to reuse the pixel between the deblocking filter and the prediction unit. Further, we propose a hybrid scheduling to reduce the processing cycles and improve the system throughput. The main idea is that we use four pixel buffers to keep the intermediate pixel value and perform the horizontal and vertical filtering process in one hybrid scheduling flow. Moreover, the proposed design is implemented on the hardware architecture. Based on the working frequency of 100MHz, the synthesized gate counts are very small. Finally, the proposed architecture can easily achieve the real-time decoding with 1080HD@30fps in 4:2:0 format.

## *3.2 Design for Multi-standard Applications*

### 3.2.1 Motivation

Various video coding standards are in use recently. Traditional MPEG standards support the features of backward compatibility. However, H.264/AVC [7] is the newest video standard, and there is no backward compatibility of H.264/AVC to the former H.263 and MPEG-4 (Part-2) video coding standards. Therefore, the development of combined

video coding standard is a must to meet the different system requirements. Both H.264/AVC and MPEG-4 adopt the deblocking filter to eliminate the blocking artifacts. However, the H.264/AVC adopts the deblocking filter as an in-loop process and the other standards adopt it as a post-loop process. The detailed features of deblocking filter are listed in Table 3.4. To provide the unique architecture for multiple video standards, we propose a hybrid scheme to integrate the standardized in-loop filter and the informative post-loop filter. We call it as loop/post deblocking filter.

Due to the non-standardization of post-filter [11], it provides high freedom to develop a certain suitable algorithm for the integration with loop-filter. Based on the original algorithm of 4x4 loop-filters, an 8x8 post-filter has been developed. We modify the filtered ordering and the number of related pixel. Therefore, the modified post-filter can easily be integrated with the 4x4 loop-filter. Simulation results also show that the proposed loop/post filter incurs the penalty of slight PSNR loss (less than 0.05dB).

Table 3.4: Features of the deblocking filter in different standards.

| De-blocking Filter | In-loop | Post-loop | |
|---|---|---|---|
| **Standardization** | normative | informative | |
| **STANDARD** | H.264/AVC | MPEG-4 (Annex F.3) | H.263 (Annex J) |
| **Filtered boundary** | 4x4 Edge | 8x8 Edge | 8x8 Edge |
| **Filtered ordering** | Vertical edge first | Horizontal edge first | Horizontal edge first |
| **No. of related pel (max.)** | 8 (4-pel per side) | 10 (5-pel per side) | 4 (2-pel per side) |

### 3.2.2 Loop/Post Deblocking Filter

To reduce the cost overhead of deblocking filter for multiple standards, it is required to develop a hybrid algorithm and unique architecture of deblocking filter. The video standards of H.264/AVC and former MPEG adopt deblocking filter as in-loop and post-loop process respectively. However, the performance improvement is very mild when applying the loop filter as the post filter in MPEG-4. Therefore, we propose a hybrid algorithm to make a compromise between the integration cost and the performance loss. Figure 3.8 shows the

decision of our proposed loop/post filter. The proposed hybrid algorithm retains the original loop filter due to the standardization in H.264/AVC. In addition, we modify the post filter (marked as the underline in Table 3.5) to easily integrate into original loop filter design.

Table 3.5: Parameter selection of the proposed loop/post deblocking filter.

| | | Loop Filter[7] | Post Filter[11] | Proposed loop/post |
|---|---|---|---|---|
| Filtered Control | Filtered Edge | 4x4 | 8x8 | 4x4 & 8x8 |
| | Filtered Ordering | Vertical first | Horizontal first | Vertical first |
| Mode Decision | Algorithm dependency | Syntax-dependent | 10-Pixel-dependent | Syntax & modified pixel dependent: 8- pixel dependent |
| Filtering Mode | Filtered Strength (strong) | bS=4 | DC offset mode | bS=4 |
| | Filtered Strength (weak) | bS<4 | Default mode | bS <4 & modified default mode: [2- 4 4-2] |

The proposed algorithm exploits the features of loop and post filters. It can be partitioned into three main parts as identified in Table 3.5. In the filtered control, we retain the filtered edge of 4x4 and 8x8 respectively. The reason is that the basic transformation unit is located on the 4x4 sub-block and 8x8 block. Further, we modify the filtered ordering in post filter to unify into a hybrid structure.

### 3.2.2.1 Mode Decision

There are several differences between the mode decision of loop and post filter as shown in Figure 3.8. The loop filter is performed in the coding loop and controlled by the syntax parser. As we discussed in previous sections, BS is determined according to intra/inter mode, coded block pattern or motion vectors for each 4x4-block boundary. And the value of BS ranges from 0 to 4. However, the post filter is applied after the video decoder and can be considered as a post-processing unit. The variable used to decide filter mode is not the same as BS and is calculated according to neighboring pixels per sample edge. And the value of $T_2$ and $T_3$ are fixed experimented values. To merge the mode decision, we retain the mode decision features of loop and post filter. Further, we modified the mode decision of post filter into the 8-pixel related algorithm. Besides, the value of $T_2$ and $T_3$ are modified to dynamic values according to the syntax parser.

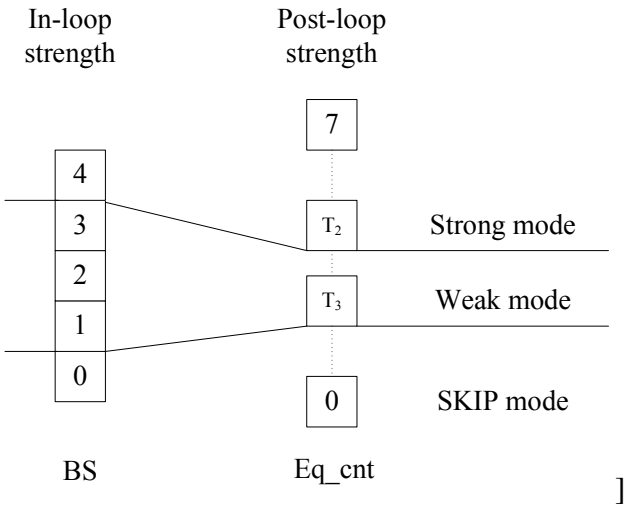



Figure 3.8: The mode decision used in in-loop filter and post-loop filter.

### 3.2.2.2 Filtering Mode

To combine the edge filtering between in-loop and post-loop filters, we modify the default mode of post filter and apply the process of "bS=4" into the DC offset mode of post filter. As shown in Figure 3.8, the filtering mode can be partitioned into strong and weak mode. Since the strong filtering mode in post filter is similar to the one in the loop filter, we apply the strong filtering mode instead of the original DC offset mode in MPEG-4 Annex F.3 [11]. Further, we modify the approximated DCT kernel (i.e. [2 -5 5 -2]) into the [2 -4 4 -2] for the delta generation in post-loop filter. Therefore, we can exploit shifter instead of constant multiplier. All the modification of post filter design can be summarized in the underline of Table 3.5. In Figure 3.9, we show the architecture of weak filtering strength for the detailed descriptions.

We implement a Pixel-in-Pixel-out edge filter (P-i-P-o Edge Filter) to integrate the loop and post filters into unified architecture. From Figure 3.9, the incurred MUX is exploited to switch different filtering functions. In the loop filter, the filtering algorithm of H.264 is implied. The modified filtering algorithm of MPEG-4 Annex F.3 [11] is also realized in the loop/post filter architecture. Therefore, the proposed loop/post filter is suitable for the implementation of multiple video standards.

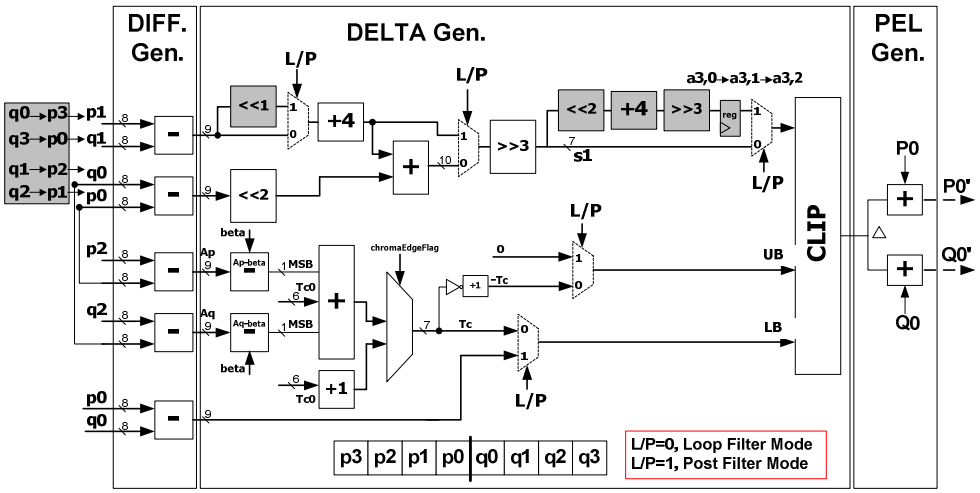



Figure 3.9: The pixel-in-pixel-out filtering process with weak strength of the proposed loop/post filter.

We implement an area efficient deblocking filter by exploiting the computational redundancy between the loop and post filter. The proposed loop/post filter with weak strength has been depicted in Figure 3.9 (only p0', q0' shown). The extra shaded regions are allocated to perform the post filter on the original loop filter design. We partition the proposed loop/post filter into three main phases. They are the phases of difference generation, delta generation and the pixel generation respectively. The difference generation phase is the pixel-difference initialization of edge filtering. After that, the delta generation phase is performed to generate the delta metric between the un-filtered and the filtered pixel. They use the CLIP operation to limit the delta value between UB (Upper Bound) and LB (Lower Bound). The phase of pixel generation adds the unfiltered data to obtain the final results.

### 3.2.3 Simulation Result

The target technology is 0.18 $\mu$m, and the synthesized gate count is 21.1K excluding embedded memory. Compare to the total hardware cost of in-loop and post-loop filter, the proposal reduce about 30% hardware cost. We modify the post filter algorithm in [11] and make a compromise between the integration cost and the performance loss. We use

"Foreman" and "Stefan" as our test sequences. In Figure 3.10, the average performance degradation of the modified post filter is less than 0.05dB as compared to the traditional post filter [11]. In addition, the processing cycles of post-loop filter are identical to that of in-loop filter because they use the same control flow.

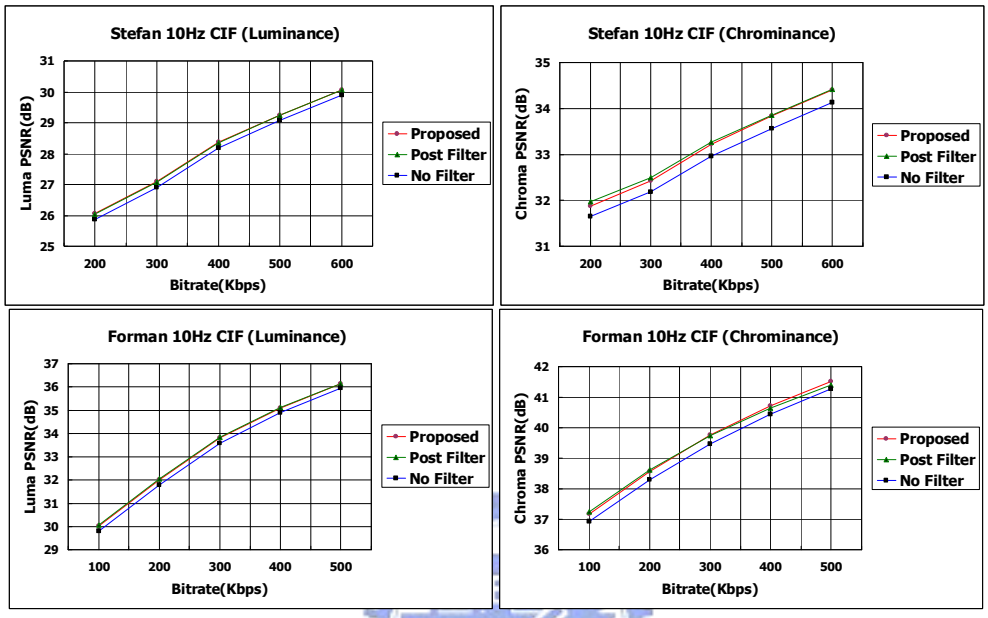



Figure 3.10: The performance comparison due to the modification of post filter.

### 3.2.4 Summary

An area-efficient and high-throughput deblocking filter has been presented to meet the different requirements for multiple video coding standards. We modify the post filter algorithm to make a compromise between hardware integration complexity and performance loss. The proposal saves about 30% hardware cost as compared to total hardware cost of in-loop and post-loop filter with penalty of slight performance degradation which is less than 0.05dB.

# Chapter 4
# Joint Architecture of Error-concealed Deblocking Filter for H.264 Decoder

## 4.1 Design for Low-cost and Real-time Application for video transmission

Of all modalities desirable for future mobile multimedia systems, high-quality motion video over a reliable transmission is the most demanding. However, the transmitted visual quality may suffer abruptly because the channel deteriorates due to fading, co-channel interference, and signal attenuations. To deal with the transmission errors over an error-prone channel, much effort has been invested to improve the error-robustness in the source decoding procedures, such as error resilient and concealment tools.

### 4.1.1 Problem Formulation

So far, many researches focus on error concealment (EC) for the corrupted video bit-stream over an error-prone or wireless channel. The EC can be divided into two groups: spatial and temporal error concealment (SEC and TEC). Generally, the error scheme for video standards uses SEC and TEC adaptively according to the MB type. However, TEC is not always adequate for concealing errors in video sequences. This is especially true for video sequences with irregular motion, abrupt scene changes, and intra-coded image frames. To the viewer, poor spatial concealment of error regions leads to the error propagation in the subsequent frames.

Hence, compared to temporal EC, spatial EC is of great importance and challenges.

Thereafter, we focus on the spatial EC for concealing corrupted region of image frames.

For the SEC, it is significant to preserve the existing edges without creating new strong ones. For the case, the algorithm of DI is given to prolong the edges entering to the corrupted MB. But for the texture area, the performance of BI is better than DI. So the algorithm which uses DI and BI with mode decision has been presented to improve the visual quality according to the edge activities [5]. However, such kind of algorithm belong to frame-based EC due to the thing that usage of pixels in four direction pixels. In order to keep the neighboring macroblocks available, the encoder has to support the FMO. Besides, the decoding system needs additional memory or local buffer to store the pixels on the boundary of neighboring MBs. These algorithms which are widely used in I frames could have an acceptable performance with FMO, but the performance would decrease a lots without FMO. Here we propose a new SEC which only needs the pixels on the top and left side stored in the slice memory used by deblocking filter [12]. The proposed SEC is combined with deblocking filter to reduce the hardware cost and reuse the memory capacity.

### 4.1.2 Error-Concealed Deblocking Filter

As we know, the deblocking filter doesn't work when the decoded MB is erroneous, and the EC will not be activated when the decoded MB is correct. Based on the design concepts, we combine the deblocking filter and EC to reduce hardware cost. Moreover we limit the pixels used to conceal corrupted MBs to the top and left neighboring 4x4-blocks. As shown in Figure 4.1, the pixels used in EC are the dotted 4x4-blocks stored in slice memory [12].

The ECDF consists of edge detection, replacement, smoothing and deblocking filter [12] which is based on our previous work. The ECDF filters all edges in a correct or corrupted MB. In Figure 4.2, the grid blocks are additional parts for error concealment capabilities and the signal, Corrupted MB, is transmitted from error detection. ECDF works

as deblocking or EC according to the signal. The pixel buffer stores the pixels from content memory or slice memory [12]. 1-D filter and boundary strength are the algorithms standardized in H.264. The pixels Q on the right (bottom) side are replaced pixels in a specified direction determined by edge detection. Therefore, the edges extended into the corrupted MB can be preserved slightly. Further, we modify the algorithm of boundary strength and filterSamplesFlag to reconstruct texture area well.

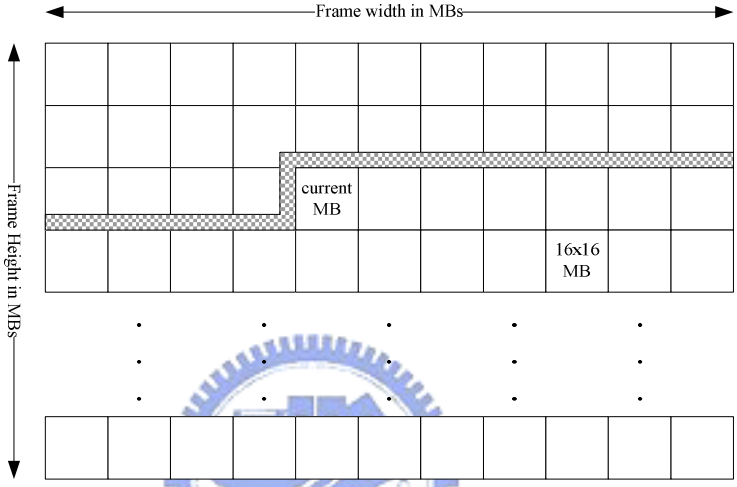



Figure 4.1: The dotted 4x4-blocks are stored in slice memory.

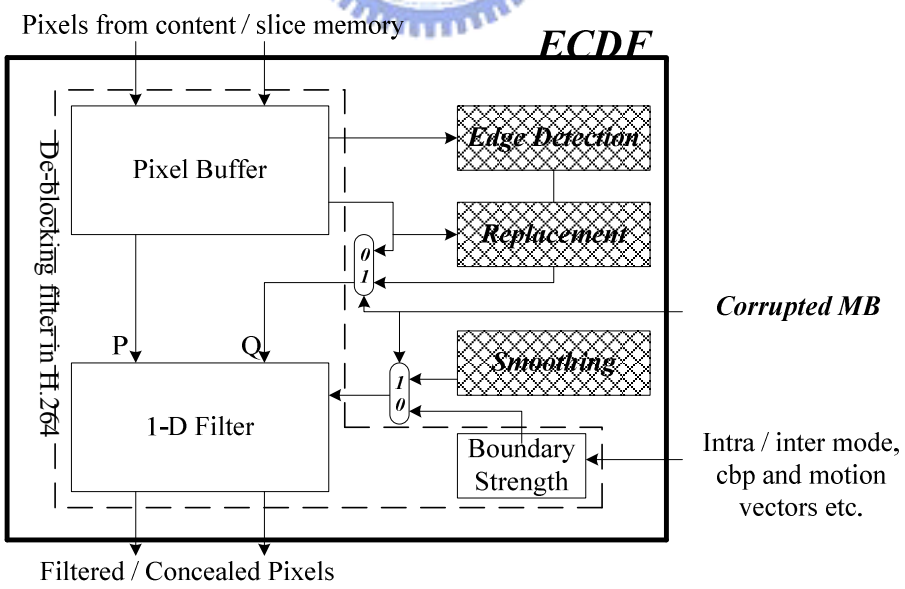



Figure 4.2: The block diagram of ECDF. Dotted regions are additional parts for the capability of error concealment.

### 4.1.2.1 Edge Detection

$$\min_{\text{dir.}\{\frac{n\times\pi}{8}\}}\left\langle \sum_{i=0}^{\left(\frac{4}{size}\right)^2}\left|P_{dir.}(i)-P_{corr.}(i)\right|\right\rangle \qquad i\in\left\{0,...,\left(\frac{4}{size}\right)^2\right\},size\in\{1,2,4\} \tag{4.4}$$

In practice, Sobel mask is used to determine the magnitude of gradient and edge slope of existing edge in the neighboring 4x4-blocks as shown in Eqs. (4.1), (4.2) and (4.3). Further, the edge direction is determined by ranking the edge slope.

$$S_x=\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}, S_y=\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix}, F=\begin{bmatrix}p_{i-1,j-1} & p_{i,j-1} & p_{i+1,j-1}\\ p_{i-1,j} & p_{i,j} & p_{i+1,j}\\ p_{i-1,j+1} & p_{i,j+1} & p_{i+1,j+1}\end{bmatrix} \tag{4.1}$$

$$G_x=(S_x)^T F, G_y=(S_y)^T F \tag{4.2}$$

$$|G|=\sqrt{G_x^2+G_y^2}, slope=\frac{G_y}{G_x} \tag{4.3}$$

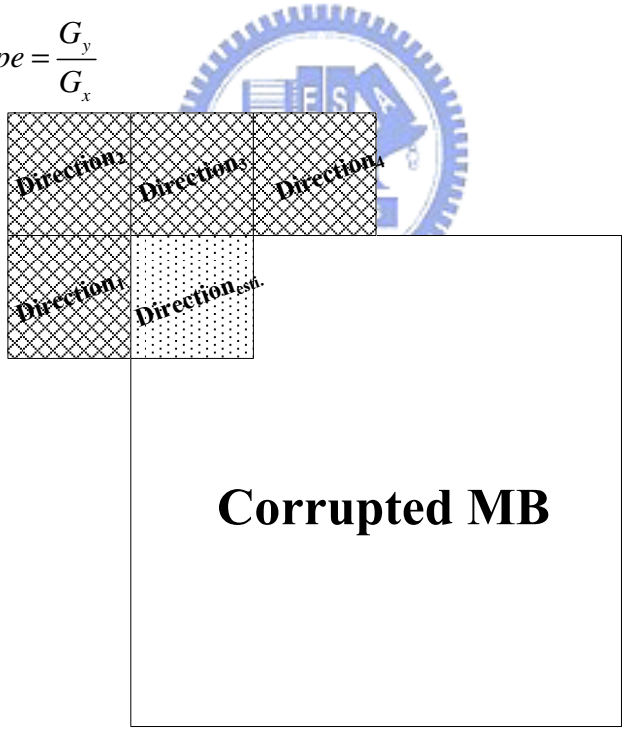



Figure 4.3: The behavior of edge detection a corrupted 4x4-block in MB.

At first, the Sobel mask calculates the proper edge directions ($Direction_1$, $Direction_2$, $Direction_3$ and $Direction_4$) and gradients in grid region (neighboring correct 4x4-blocks) as shown in Figure 4.3. Then we use edge information to predict the edge direction ($Direction_{esti.}$) in the dotted region (corrupted 4x4-block). The $Direction_{esti}$ is selected from

the four edge directions according to the value of the edge gradients in neighboring correct region and the location of corrupted block. The gradients are used to judge if there is a real edge or not and the location of corrupted block determines the priority of four edge directions. If the gradient is larger than the fixed threshold evaluated by simulations, the edge is regarded as real one. Besides, if the location of corrupted block is close to top boundary of MB, the priority of $Direction_3$ becomes highest and priority of $Direction_1$ is lowest. The process of edge detection is employed in vertical edges of 4x4-block. Therefore, we can support multiple edges extended from the top and left neighboring blocks of this corrupted 4x4-block.

### 4.1.2.2 Replacement

When the proper edge is estimated by edge detection in the corrupted block, replacement reconstructs the missed pixels in 4x4-block by duplicating instead of interpolation along the edge direction estimated in practice. By ranking the edge slope with thresholds evaluated from simulations, four replacing modes are used to recover the missed pixels as Figure 4.4 shows.

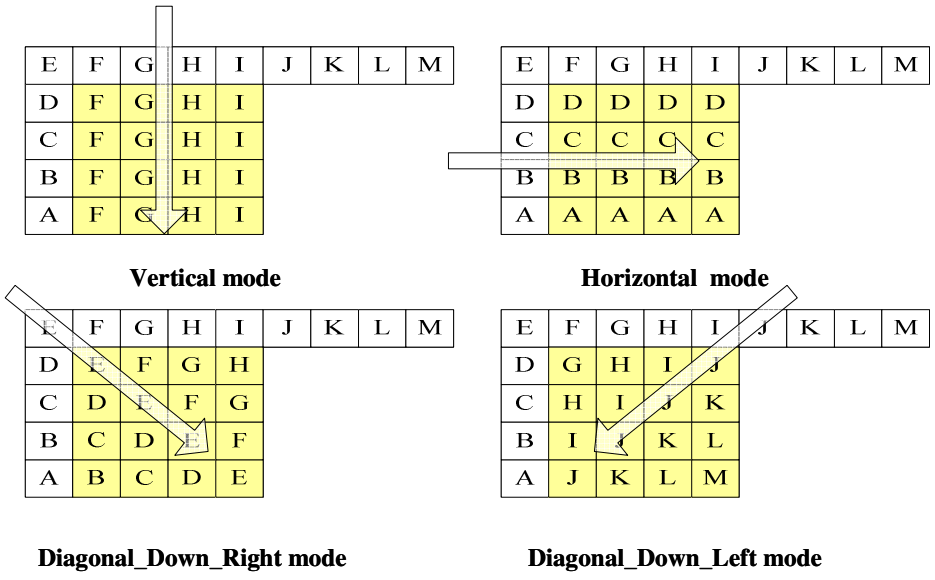


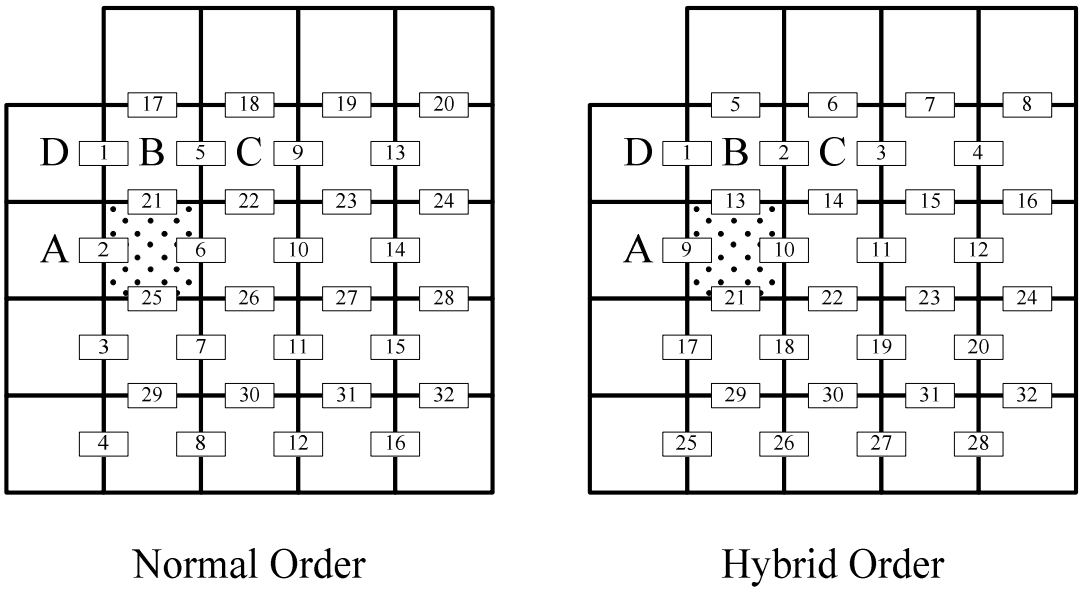

Figure 4.4: Four kinds of replacing mode.

### 4.1.2.3 Smoothing

After replacing the missing pixels on the right (bottom) side in the corrupted 4x4-block, the deblocking filter starts to smooth the edges of the corrupted block to improve the visual quality. There are two modifications in the algorithm of deblocking filter in H.264. One of the modifications is that we force the boundary strength (BS) equals 4 when the decoded MB is erroneous. With the strong filter (BS equals 4) specified in H.264, 6 pixels can be smoothed per edge. For the weak filter (BS<4), there are only 4 pixels can be filtered. The other modification is that we force the signal, filterSamplesFlag, which is specified in H.264 standard [7] as shown in Eq. (4.4), to be true when the decoded MB is corrupted.

$$filterSampleFlag = \left(BS \neq 0 \,\&\&\, |p_0 - q_0| < \alpha \,\&\&\, |p_1 - p_0| < \beta \,\&\&\, |q_1 - q_0| < \beta\right) \quad (4.4)$$

There is one problem for edge detection if the neighboring blocks of corrupted 4x4-block are concealed ones. The predicted edge in corrupted block may be not proper due to the edge information in the neighboring blocks strongly depends on replacing mode. It means that one erroneous block recovered using horizontal replacing mode has the horizontal edge information.




Normal Order Hybrid Order

Figure 4.5: The number on the boundaries of blocks is the filtering order.

Taking Figure 4.5 as an example, the neighboring blocks (B, C) are replaced ones. If we conceal each block by normal order, the edge information in blocks (B, C) strongly depend on replacing mode due to that blocks (B, C) are replaced by other neighboring pixels. On the other hand, the blocks (B, C) can be smoothed by filtering edges (2, 3, 5, 6) with hybrid order before we use the edge information in blocks B, C to predict the proper edge in corrupted block (dotted block).

#### 4.1.2.4 Reconstructing flow

As shown in Figure 4.6, the process of ECDF for capability of error concealment can be divided into two parts. For the vertical edges, replacement selects the correct neighboring pixels from pixel buffer and reconstructs the corrupted 4x4-blcok by duplicating according to edge direction estimated by edge detection. For the horizontal edges, a strong filter mode is used directly to smooth the recovered pixels.

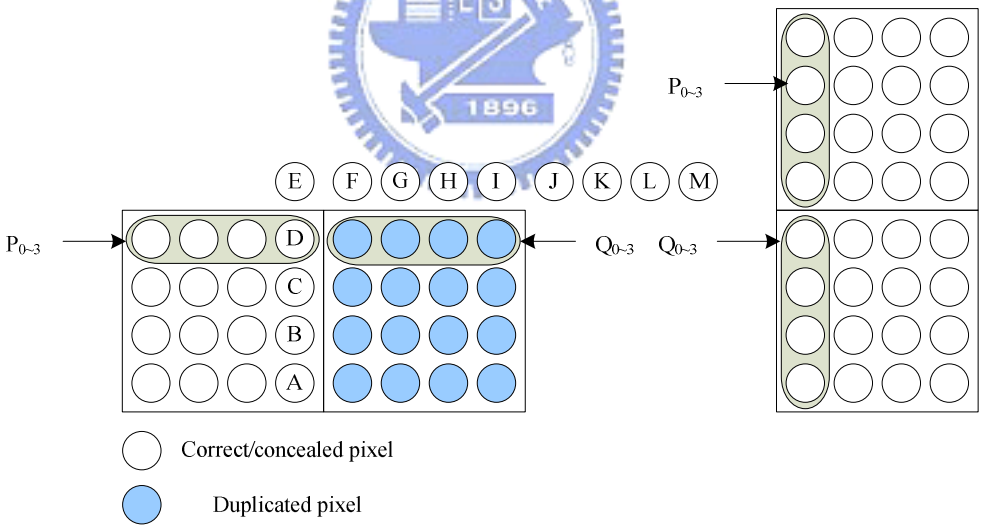



Figure 4.6: The behaviors of ECDF for vertical edges and horizontal edges.

### 4.1.3 Simulation Results

For simulations, we consider the sequence patterns which are encoded with CIF resolution using dispersed FMO and slice size of 198 MBs in JM9.8. The number of frame is equal to 30 frames. With dispersed FMO, there are different type of corrupted MB we can conceal to see the average performance in directional or texture areas. Further, we assume

that error detection is perfect when decoding process and EC is operated in MB-based processing. It means that there is no information used on the right (bottom) side of one corrupted MB in EC. From the simulation results which are shown in Table 4.1, we can see that the performance of proposal strongly depends on the sequence patterns. Since we use edge detection in our algorithm, the proposal is suitable for directional area, such as the background of "foreman", but not suitable for texture area, such as the background of "table". Generally, the performance gain of proposed algorithm ranges from 0.1dB to 0.4dB as compared to $BI_{limit}$ (BI with top and left information only). For the sequence pattern which has obvious directional areas, the performance of ECDF is significantly higher than $BI_{limit}$. As we can see in Table 4.1, the average PSNR of ECDF and $BI_{limit}$ for sequence of "foreman" are 23.83dB and 22.43dB separately. On the other hand, the performance of ECDF is slightly worst than $BI_{limit}$. As shown in Table 4.1, the average PSNR of ECDF and $BI_{limit}$ for sequence of "table" are 23.90dB and 23.99dB separately.

Table 4.1: The performance of ECDF and $BI_{limit}$.

| PSNR(dB) | table | stefan | Mobile | foreman | news |
|---|---|---|---|---|---|
| ECDF | 23.90 | 20.20 | 17.73 | 23.83 | 21.98 |
| $BI_{limit}$ | 23.99 | 20.11 | 17.5 | 22.43 | 21.60 |

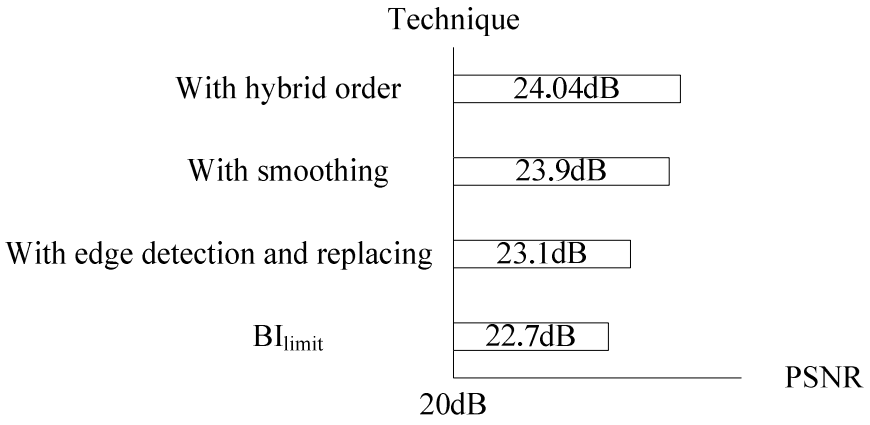
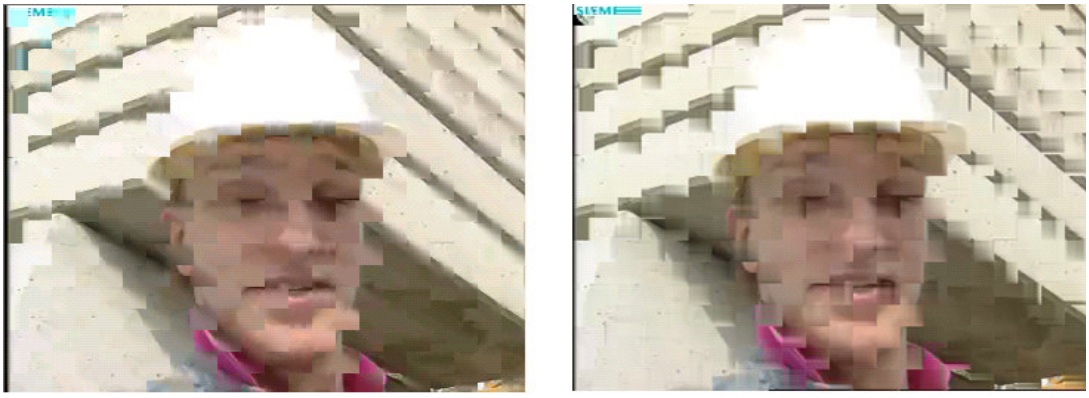



Figure 4.7: The improvement of PSNR for the sequence of "foreman".

To see the performance improvement for each technique of proposal, the Figure 4.7 is presented here. We take the 1st frame of sequence "foreman" as an example. For the basic situation of using the $BI_{limit}$, the PSNR is about 22.7dB. With the edge detection and replacement, the PSNR can be improved to 23.1 dB. By using the deblocking filter to smooth all the boundaries of blocks, the PSNR can be improved to 23.9 dB. The final PSNR is 24dB with hybrid order. Finally, the Figure 4.8(a) and 4.8(b) are presented here to show the subjective quality of frames which are concealed by ECDF and $BI_{limit}$ separately.



(a) (b)

Figure 4.8: PSNR of concealed frame with the ECDF (a) and concealed frame with $BI_{limit}$ in JM9.8 (b).

### 4.1.4 Implementation results

Table 4.2: The comparison of architecture and processing time per MB for the error concealment in H.264/AVC.

| Items | De-blocking filter + EC | ECDF |
|---|---|---|
| Gate count | 27.7K(0.13μm) | 19.5K(0.13μm) |
| Processing time per MB | 243/384 | 243/275 |
| Working frequency | 50MHz | 50MHz |
| Required pixels from external memory | 1-pixel on the right (bottom) side near the corrupted MB | none |

With the 0.13μm process, the gate count of the BI is about 12.7K including local buffer with clock period equals 20ns as shown in Table 4.2. To store all the correct neighboring

pixels around the corrupted MB, we need 512-bits (i.e. 16x4x8) registers. Hence, the total gate count for the deblocking filter and error concealment becomes about 27.7K (15K for the deblocking filter [12] and 12.7K for BI). The processing cycles per corrupted MB is 384 cycles due to the number of missed pixels are 384 per MB. The original processing cycles per correct MB of deblocking filter is 243 cycles.

On the other hand, the gate count of ECDF is 19.5K with 0.13μm process. Since the reconstructing flow is the same as filtering flow in deblocking filter, the processing cycles are 243 cycles which equals the one of deblocking filter [12] for a correct or corrupted MB. Therefore, the ECDF achieves 70% of hardware cost as compared with the direct implementation. And the power consumption can be reduced due to the reduction of hardware cost and external memory access.

### 4.1.5 Summary

The ECDF which combines the EC with deblocking filter in H.264 is proposed here. The proposal is easily implemented and integrated into the deblocking filter in H.264. The hardware cost and memory access can be reduced by sharing the memory used by deblocking filter and using the deblocking to support the capability of error concealment instead of interpolation. The proposal reduces 30% of hardware cost compared to direct implementation. Although the influence of FMO is significant to EC, not all the decoder supports FMO due to the high complexity. Hence, we focus on the simulations without FMO and try to upgrade the performance of ECDF without FMO to approach the performance of BI in JM9.8. The PSNR of ECDF is 1.4dB better than $BI_{limit}$ and comparable to BI in JM9.8 for the sequence of “foreman”.

## *4.2 Performance Analysis of ECDF*

### 4.2.1 Motivation

Although the hardware cost of ECDF can be decreased largely by hardware and

memory sharing as shown in Table 4.2. The performance is still not good enough as compared to BI algorithm used in JM9.8 with FMO. Hence, we modified the main roles in the ECDF, replacement and edge detection, to improve the performance further. Followed are some simulations to see the upper bound performance of ECDF.

### 4.2.2 Ideal Functions

As we know, the more complex algorithms we use, the better performance is. Here we focus on functions of the edge detection and replacement without considering the complexity to see the best performance it has.

#### 4.2.2.1 Edge Detection

There are challenges for the edge detection. Although the Sobel mask is useful to calculate the edge gradient and direction. But the problem is that we do not know the actual value of missed pixels in corrupted MB. So all we can do is to estimate the edge direction in erroneous 4x4-block closed to the MB edge by using the edge information in the correct neighboring correct 4x4-blocks. Then, we use these estimated edge information to estimate the edge in the deep erroneous 4x4-blocks. Here comes error propagation problem we need to overcome.

Before finding the best method, we do simulations to see the theoretical performance influenced by edge detection. Instead of estimating each edge direction in 4x4-block, we choose the best replacing direction according to the minimal absolute of sum. The absolute of sum is the difference measurement between correct block ($P_{corr.}$) and replacing one ($P_{dir.}$) as shown in Eq. (4.4). It means that we calculate the absolute of sum for each replacing block with different direction to find the best edge direction for each 4x4-block in one corrupted MB. Than we use the edge information which comes from best results to conceal all blocks in corrupted MB in raster scan order.

$$\min_{\text{dir.}\{\frac{n\times\pi}{8}\}}\left\langle \sum_{i=0}^{\left(\frac{4}{\text{size}}\right)^2} \left|P_{\text{dir.}}(i)-P_{\text{corr.}}(i)\right| \right\rangle \qquad i\in\left\{0,\ldots,\left(\frac{4}{\text{size}}\right)^2\right\},\text{size}\in\{1,2,4\} \tag{4.4}$$

From the simulation result, we can see that the performance of ECDF can be improved significantly with ideal function of edge detection as shown in Figure 4.9. And the PSNR of ideal ECDF without FMO is 2.4dB higher than implemented ECDF [13] and is even 1.1dB higher than the BI with FMO used in JM.9.8 in average.

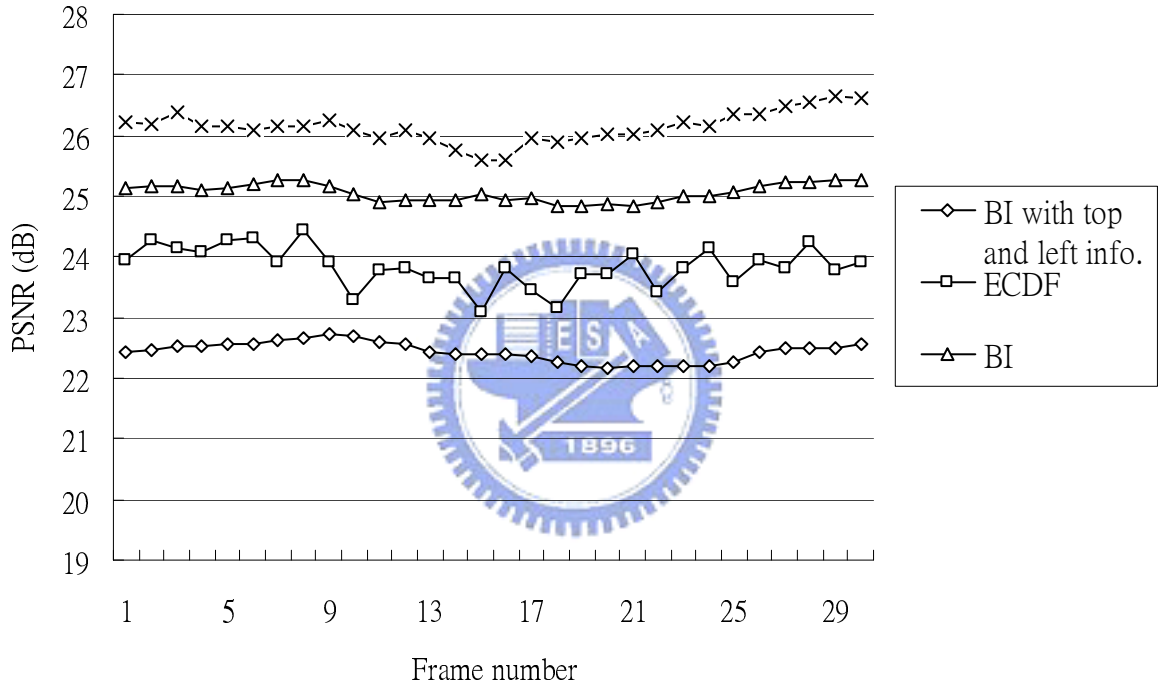



Figure 4.9: Simulation result for the ideal edge detection.

### 4.2.2.2 Replacing

In order to improve the performance of replacing, we modified the replacing with two ways. The first way is that we use additional replacing directions. There are four replacing modes used in the implemented ECDF [13]. Here we extend the replacing direction from four to eight to see the performance gain as shown in Figure 4.10.

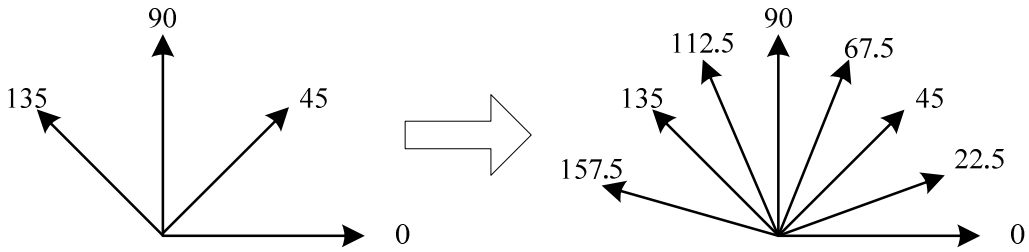



Figure 4.10: Modified replacing directions.

Second, we scale down the basic unit of edge detection to support more complicated edges. As we know that the DI algorithm has been used for EC for a long time. The basic unit of DI used for MPEG is belongs to MB level. It means that all missed pixels in MB are interpolated by using the same direction. And for the H.264, the basic unit of DI is 4x4-block due to the intra prediction described in H.264. Here we divide each 4x4-block into four 2x2-block or sixteen pixels to see the performance gain as Figure 4.11 shows. By doing so, the edge can be preserved more correctly.

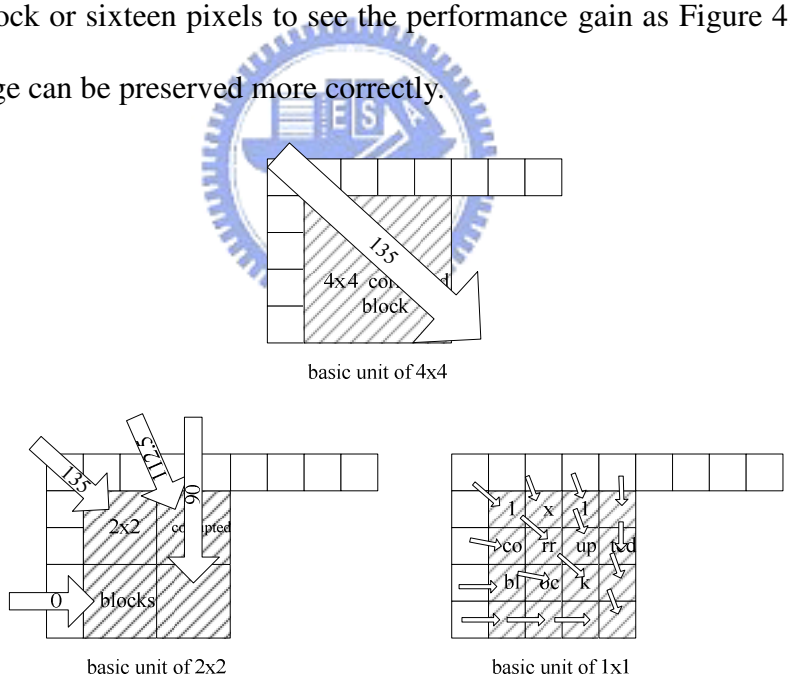



Figure 4.11: Basic unit for Edge information.

From the simulation result shown in Figure 12, we can see that about 0.2dB can be gained if we using eight directions for replacement as compared to four directions. Further, about 0.55 dB can be improved when we replace each 2x2-block with one specified direction as compared to 4x4-block. Finally, the PSNR gain can be improved to 0.88 dB if

we use each pixel as basic unit for replacing.

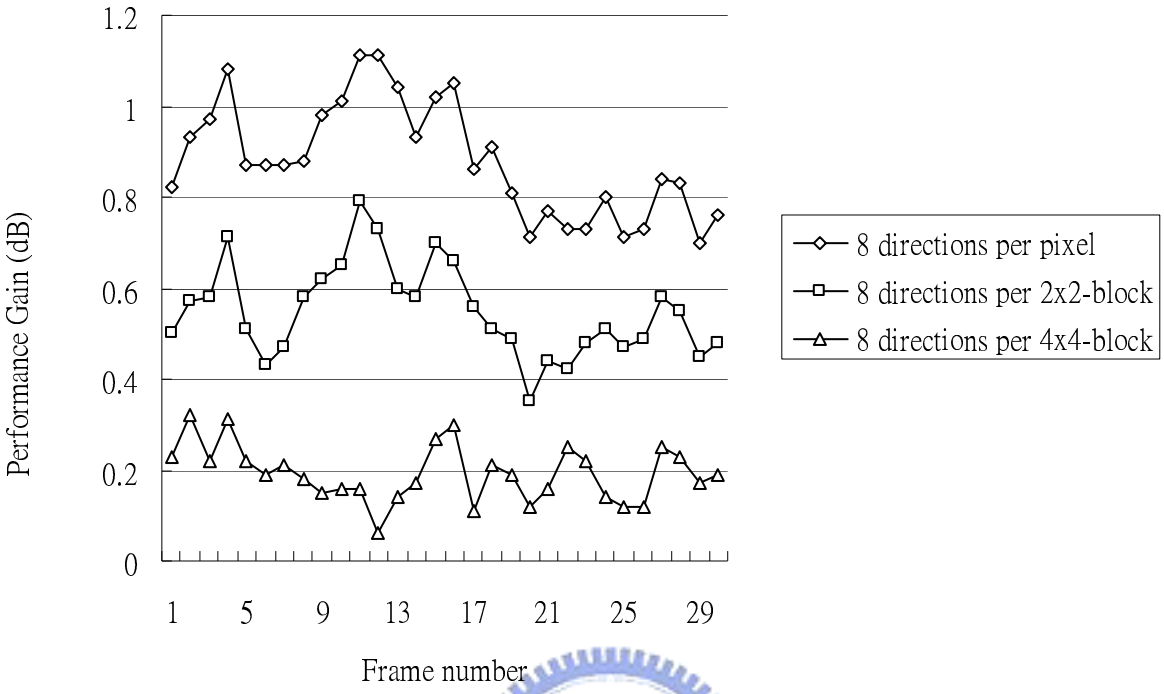



Figure 4.12: Simulation result for different basic unit.

### 4.2.3 Simulation Result

Table 4.3: Performance of Ideal ECDF.

| PSNR(dB) | table | stefan | mobile | foreman | news |
|---|---|---|---|---|---|
| Ideal ECDF | 26.58 | 22.92 | 20.44 | 27.02 | 24.86 |
| BI | 25.76 | 21.80 | 19.16 | 25.06 | 23.68 |
| ECDF | 23.90 | 20.20 | 17.73 | 23.83 | 21.98 |
| $BI_{limit}$ | 23.99 | 20.11 | 17.53 | 22.43 | 21.60 |

As shown in Table 4.3, the performance using only top and left side neighboring 4x4-blocks to conceal corrupted MBs can be better than the one of BI used in JM9.8 with

FMO. The performance gain of ideal ECDF ranges from 0.8dB to 1.96dB as compared to the one of BI in JM9.8 for difference sequence patterns.

Finally, the concealed frames with ideal ECDF and BI with correct information in four directions are presented to see the subjective performance as shown in Figure 4.13. If the video stream is corrupted due to bit error caused by video transmission, the decoded frame which is encoded into two slices with FMO may become the frame in Figure 4.13(a). The frames concealed by using the ideal ECDF and BI in JM9.8 are presented in Figure 4.13(b) and 4.13(c) separately.

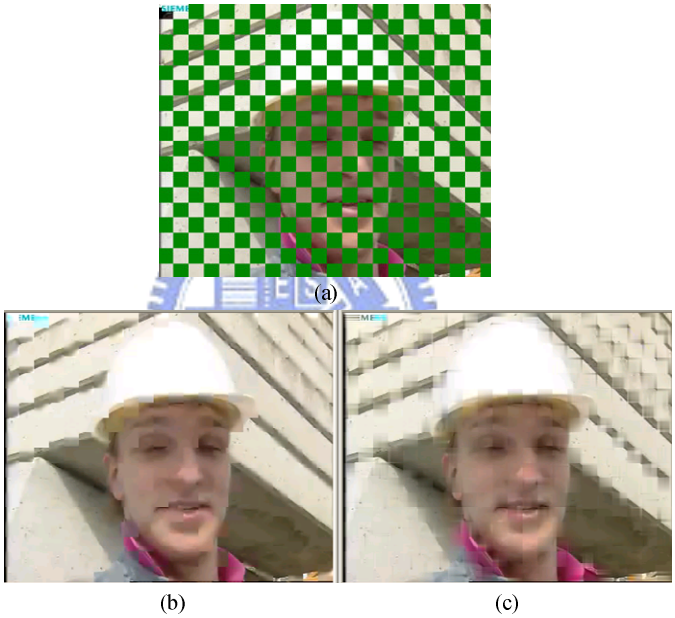



Figure 4.13: (a) Corrupted frame due to bit error. The concealed frames by BI (c) and ideal ECDF (b).

### 4.2.4 Summary

In this section, we presented the ideal function of ECDF and the upper-bound of performance. From the simulation results, the PSNR of ideal ECDF is significantly high than other algorithm. The main challenge of implementation of ideal ECDF is the edge detection in the filed of image processing.

# Chapter 5
# Chip Implementation for Mobile Applications

## 5.1 System Specification

For the real-time portable mobile applications, the low-power and low-delay are the most significant issues. We choose the baseline profile to take advantages of low coding delay since they do not use B-frames and MBAFF. And we select the level 2 as our design target for the portable devices. The details of coding tools used in baseline profile and bit rate constraint in level 2 are listed in Table 1.2 and Table 1.3.

The maximum computational capability is to support real time decoding of CIF H.264 video sequence in 30fps. Our operational frequency required for H.264 is 79.64MHz.

## 5.2 Design Flow

We use the standard cell based design flow. Figure 5.1 shows our design flow from system specification to physical-level.

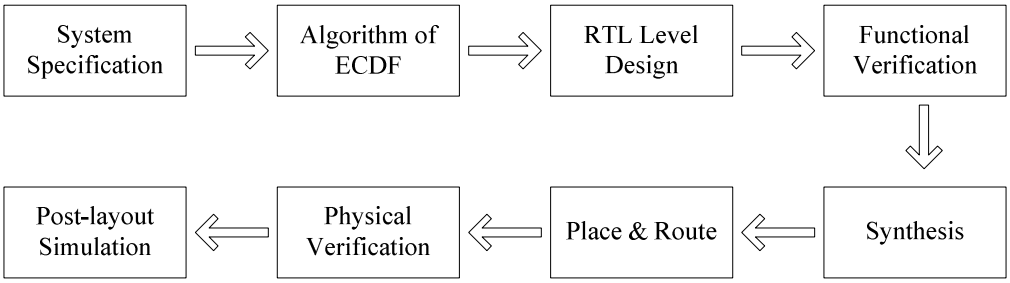



Figure 5.1 Design flow from system specification to physical-level

In the front-end design, we use standard-released reference software (Joint Model, JM) to be a high-level C-language model. The design problems are formulated and analyzed on

algorithmic level, included of the out-of-standard module, ECDF, or the post-filter for MPEG-2. After deciding the algorithms we use, we estimate the required throughput for the different applications, high-definition or low-power.

Different architectures we use cause different results, such as hardware cost, power consumption or throughput. Generally, we use Verilog RTL-level descriptions to implement a basic version design which has perfect function without considering the hardware cost. Than, we adopt some techniques in architecture level based on the basic design, such as the hybrid filtering order, memory organization and so on, to satisfy the specification we estimate.

After completing the high-throughput deblocking filter to satisfy the system specification, we integrate the deblocking filter into the whole system [14]. As for the functional verification, we use conformance patterns listed on the web site [18]. The coded block patter, QP, motion vectors, intra/inter/PCM mode, pixels in one MB or other syntax elements are dumped from the software, JM. When the inputs of the deblocking in hardware level are correct with those dumped from JM, we make functional verification by checking the outputs of hardware and software.

In the back-end design, we synthesize and route with Cadence® RTL Compiler and SoC Encounter™. The design margin, technology used, some physical effects on deep sub-micron circuits are also needed to be considered. At the end of the physical design stage, the layout verification and simulation have been made.

## *5.3 Implementation Result*

In this work, we implemented a BL@L2 of H.264 decoder with error concealment which is combined with deblocking filter standardized in H.264/AVC for the mobile handheld applications. From table 5.1, we can see the five capabilities defined by DVB-H [15] for the mobile applications. Consider the power consumption and handheld suitable

resolution, our work targets capability C.

Table 5.1: Capability defined by DVB-H for mobile applications.

| Capability | Profile @ Level | Max. frame size @ Hz | Max. bit rate (kbits/s) |
|---|---|---|---|
| A | BL @ L1 | QCIF @ 15 | 128 |
| B | BL @ L1.2 | CIF @ 15.2 | 384 |
| C | BL @ L2 | CIF @ 30 | 2000 |
| D | Main @ L3 | 625 SD @ 25 | 10000 |
| E | High @ L4 | 2Kx1K @ 30 | 20000 |

Table 5.2 Chip details

| Items | Specification |
|---|---|
| Function | H.264 Baseline@Level 2 |
| Gate counts | 279,420 (On-chip SRAM included)<br>153,170 (Excluded on-chip SRAM) |
| Technology | 0.13um 1P6M |
| Supply voltage | 3.3V/1.8V |
| Die size | 2.2x2.2mm$^2$ |
| Core size | 1.2x1.2mm$^2$ |
| Package | CQFP128 |
| Max working frequency | 50MHz |
| Core Power Consumption | 1.19mW: QCIF @ 30Hz<br>1.69mW: CIF @ 30Hz |

The detail chip specification is shown in Table 5.2. Here we assume the error detection is correct and determined out of design. This work supports the CIF resolution with maximum working frequency 50MHz for the mobile applications. The total gate count is 153.1K excluding embedded SRAM and core size is 1.2x1.2mm$^2$ in 0.13μm technology.

The gate count distribution in H.264/AVC decoder and chip view is shown in Figure 5.2 and Figure 5.3 respectively.

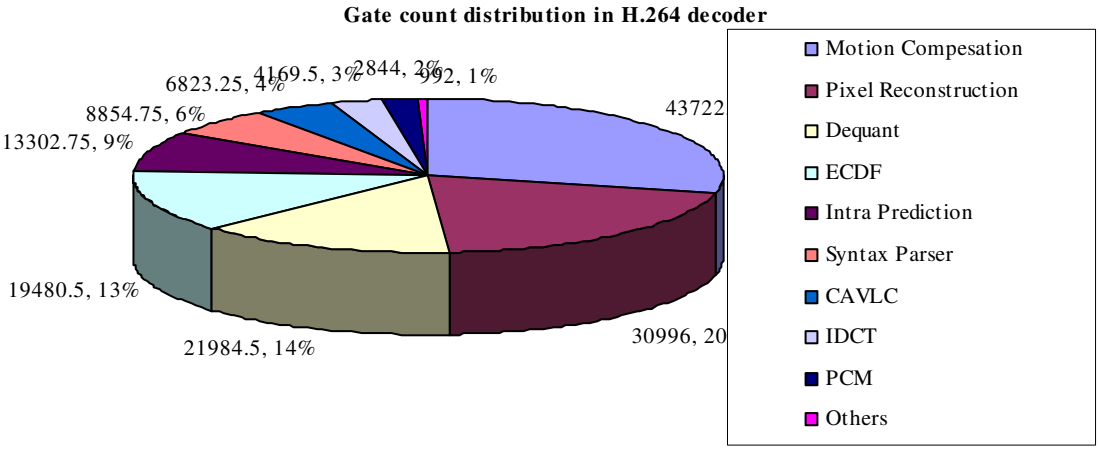



Figure 5.2: The gate count distribution of H.264/AVC decoder.

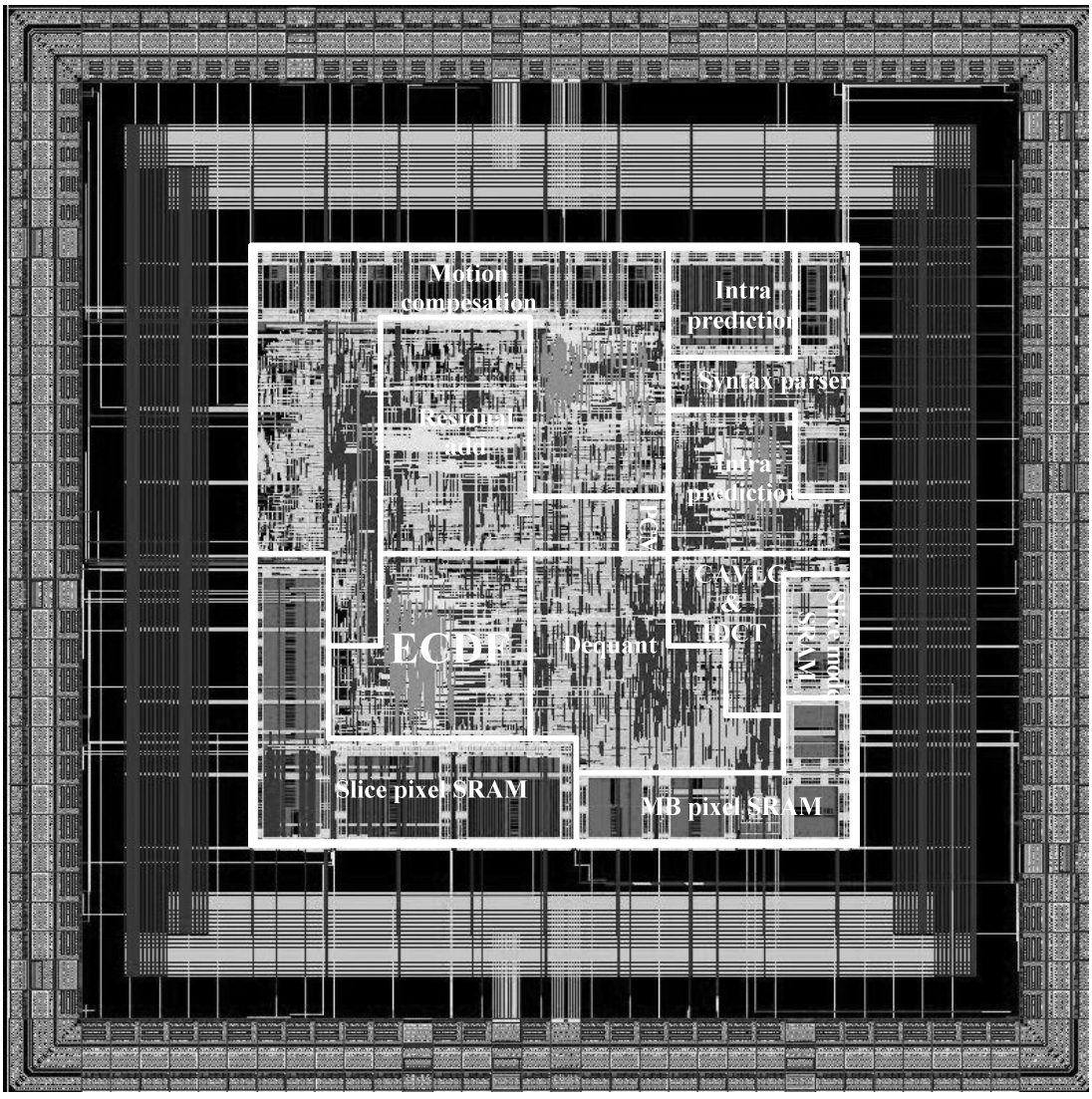



Figure 5.3: Layout of this work

## 5.4 *Measurement Results and Comparison*

The throughput per MB for different sequence is various according to the bit rate. When the bit rate is higher enough, the throughput of decoder system is dominated by throughput of CAVLC. On the other hand, the throughput of deblocking filter is the system bottleneck when decoding low bit rate sequence.

For the power consumption issue, we use the post-layout power simulation reported by Prime Power. The worst power consumptions of decoding QCIF and CIF at 30Hz are 1.81mW and 5.57mW with working frequency 6.67MHz and 25MHz respectively. For the low bit rate (<100K) sequence, the operating frequency can be reduced to 1.25MHz and 4MHz for QCIF and CIF. And the power consumptions are 0.47mW and 1.10mW respectively. The detail power reports for different sequence and working frequency are shown in Table 5.3.

Table 5.3 Power report

| Items (Core Power) | QCIF @ 30Hz | CIF @ 30Hz |
|---|---|---|
| 100K bit rate | 0.47mW@1.25MHz | 1.10mW@4MHz |
| 2000K bit rate | 1.19mW@4MHz | 1.69mW@6.67MHz |
| Worst Case | 1.85mW@6.67MHz | 5.57mW@25MHz |

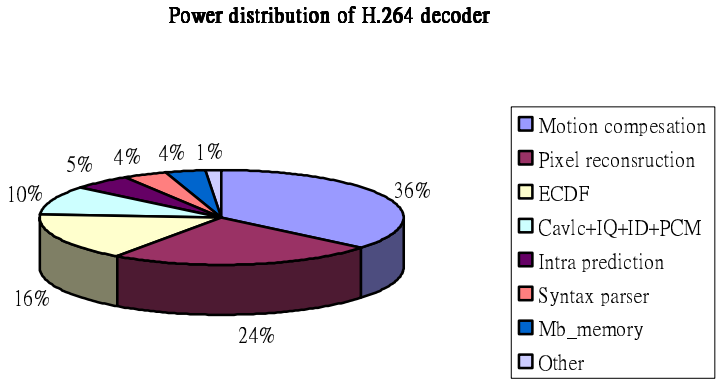



Figure 5.4: Power distribution of H.264/AVC decoder.

Further, the power distribution of H.264/AVC decoder is shown in Figure 5.4. As we can see, the power consumption of ECDF consists of 16% over the system. And embedded EC contributes 15% power consumption of ECDF.

With the embedded EC combined with deblocking filter in H.264/AVC, the post processing of EC to recover the lost MB is applied at MB-level. The ECDF reconstructs the lost MB by pixels stored in the slice memory on the top and left side without accessing the external memory. The additional hardware cost of embedded EC is 3.5K, 20% in ECDF, as shown in Figure 5.4. And the detail distribution of gate count in embedded EC is shown in Figure 5.5.

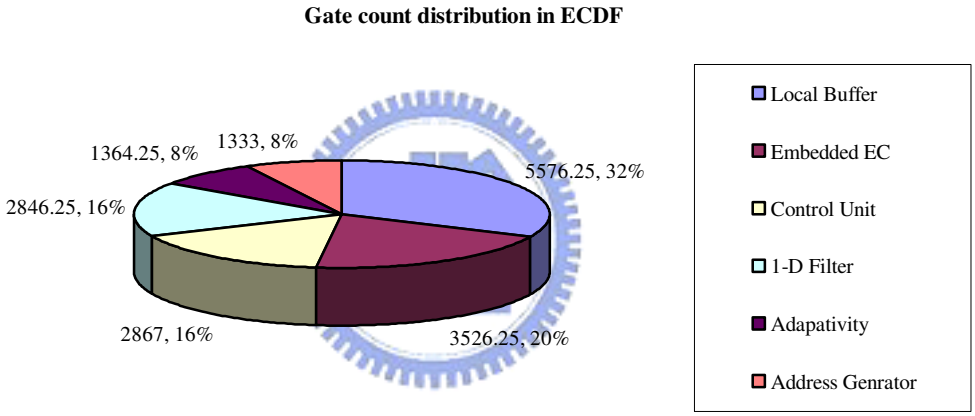



Figure 5.5: the gate count distribution in ECDF.

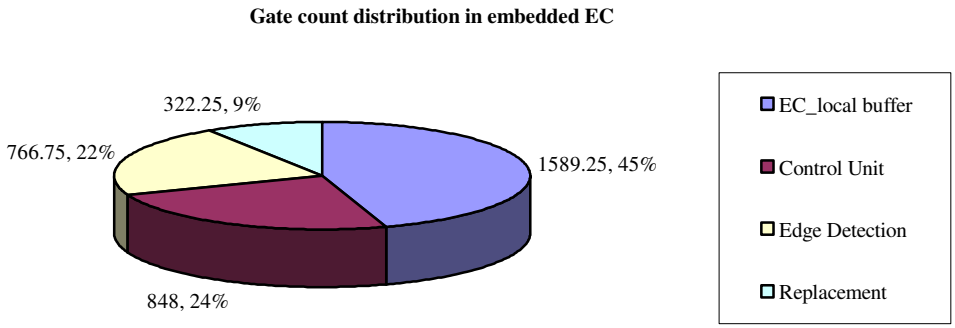



Figure 5.6: the gate count distribution in ECDF.

For the fair comparison, we select the ASIC designs of [10], [16] and [17] to see the throughput and gate count as shown in Table 5.4. Because the power consumption of memory is significant, we also estimate the number of memory access between existing different designs. Generally, the throughputs of existing designs are close to the optimal throughput of deblocking filter in H.264/AVC.

Table 5.4: A detail comparison for the deblocking filter.

| Design | [10] | [16] | [17] | proposed |
|---|---|---|---|---|
| Function | In-loop | In-loop | In-loop | In-loop/EC |
| Filtering order | Separate | Hybrid | Hybrid | Hybrid |
| # of 4x4 array | 2 | 8 | 2 | 4 |
| Memory | 1P 96x32<br>1P 64x32 | 2 2P 96x32x2<br>DP 64x32 | 1P 96x32<br>2P 32x32<br>1P (1.5xFW)x32 | 2 1P 96x32<br>1P (2xFW+20)x32 |
| Processing cycle | 878 | 446 | 214 or 246 | 243/275 |
| Process | 0.25µm | 0.25µm | 0.18µm | 0.13µm |
| Gate count | 18.91K | 24K | 20.9K | 23.6K |
| MB memory read | 320 | 160 | 96 | 96 |
| MB memory write | 224 | 64 | 0 | 0 |
| Slice read | 0 | 0 | 64 | 64+16 |
| Slice write | 0 | 0 | 52 | 52+16 |
| External read | 64 | 64 | 0 | 0 |
| External write | 160 | 160 | 108 | 108 |

# Chapter 6
# Conclusion and Future Work

## 6.1 Conclusions

Design requirements are different for different applications. In this thesis, we have implemented a high throughput deblocking filter with little memory access by data reusing for high-definition TV applications. Besides, we integrate the post-loop filter and in-loop filter for multi-standard requirement. Further, we propose and ECDF which combines the deblocking filter standardized by H.264/AVC and error concealment for the real-time portable devices over error-prone or wireless channel. Finally, we integrated the ECDF into BL@L2 H.264/AVC decoder for the mobile applications. The details of each design are presented in following section.

### 6.1.1 High-throughput Design

For the deblocking filter, we proposed the memory organization which saves the number of memory access significantly. Besides, we use parallel architecture to tack charge of the processing of BS and 1-D filter at the same time. Finally, we adopt the hybrid filtering order which still satisfies the filtering order described in standard in H.264/AVC to reuse the data, by which we can save the memory access further. By using the techniques, the processing cycle of deblocking filter can be improved to 243 cycles with single port SRAMs. The processing cycle is close to the lower-bound of throughput of deblocking filter, 192 cycles when all edges are belonged to artificial edges per MB, and gate count equals 19.64K using 0.18 $\mu$m technology. This work is suitable for low-power mobile applications due to the low memory access. Besides, it also supports the high-definition applications due

to the high throughput.

### 6.1.2 In/Post-loop filter

Further, we proposed multi-standard solution for the applications which support multiple standards, such as DVD player and DVB. Actually, the performance improvement is very mild when applying the loop filter as the post filter in MPEG-4. Hence, we proposed a hybrid algorithm for H.264/AVC and MPEG-4. The proposed hybrid algorithm retains the original loop filter due to the standardization in H.264/AVC. In addition, we modified the post filter to easily integrate into original loop filter design as shown in Table 3.5. To reduce the complexity of control unit, the filtering order of post-loop filter [11] is modified to hybrid order which is the same as the filtering order of in-loop filter. For the strength decision, we modified the fixed threshold values of post-loop filter to dynamic ones which vary according to the syntax parser, such as intra/inter mode, coded block pattern, locations of boundary or motion vectors. Finally, filter equation is modified to reduce the hardware cost: the strong mode in post-loop filter is replaced by the one applied for in-loop filter due to the similarity and the equation of weak mode used in post-loop filter is modified to avoid using multipliers. From the simulation results, the proposal saves 30% hardware cost as compared to total cost of in-loop filter and post-loop filter with performance degradation less than 0.05dB.

### 6.1.3 Error-concealed Deblocking Filter

We also proposed new spatial error concealment method, ECDF, for a real-time decoding system over an error-prone or wireless channel. There are several advantages of ECDF. The first is that ECDF can conceal I frames without the need of FMO. Second, the hardware cost of interpolation can be saved. Third, the required information for error concealment only includes the pixels in the top and left neighbors of current corrupted MB. Hence, we can conceal the corrupted MB without requiring the information in the right and

bottom side, leading to the reduction of memory space as well as bandwidth. The implementation results show that the hardware cost of ECDF can be saved about 30% compared to direct implementation. Without the right and bottom correct pixels, the proposal gains 1.4dB in PSNR compared to BI with top and left side information in JM9.8.

### 6.1.4 Chip Implementation

For the chip implementation, we integrated the ECDF into BL@L2 H.264 decoder for the real-time mobile devices. From the simulations results, the total gate count of H.264 decoder is 153K excluding embedded SRAM. The power consumption of H.264 decoder is 1.19mW and 1.69mW for QCIF@30Hz and CIF@30Hz at 4MHz and 6.67MHz respectively. The ECDF contributes 13% and 16% of hardware cost and power consumption over the decoder respectively.

## *6.2 Future Works*

### 6.2.1 Error Detection

For the error concealment in the decoder we discussed in previous chapter, we assume that the error detection works perfectly. It means that we accurately know the locations of MBs in one Frame are corrupted due to the errors over error-prone or wireless channel. Hence, we can use the error concealment to reconstruct the corrupted pixels to recover the visual quality in the correct positions, not use the error concealment in the correct MB to degrade the visual quality. The implementation of error detection is necessary for an error-concealed decoder for those real-time applications. Generally, error detection can be divided into two groups, hard detection [19] and soft detection [20].

For hard error detection, a list of error-checking conditions derived from the constraints on the H.264/AVC video bit stream syntax is checked: An invalid codeword is found for the VLC code, DCT coefficients, macroblock type or other syntax elements. Physically impossible video data are detected, such as the problem of out of frame range for

prediction. As for the soft detection, the soft information is introduced based on the multi-level de-quantization process in each demodulated symbol. It accumulates the square difference between received soft streams and decoded codeword to determine if the error occurs.

Generally, the hard error detection has the advantages of low-complexity and is easy to be implemented. But it has low performance on the error localization. It means that the hard error detection often detects the error when the several MBs are decoded after the first erroneous MB. There are many corrupted MBs determined as correct ones in this situation and degrade the visual quality significantly since we do not use error concealment on those MBs which we consider correct. On the other hand, the soft error detection has better error detection capabilities as compared to the hard error detection but is hard implemented. For a real-time decoder with ability of error concealment, a low-complexity implementation of error detection with acceptable performance on error localization is needed to reconstruct the corrupted MBs.

### 6.2.2 Temporal Error Concealment

As we discussed in Chapter 2, the major purpose of temporal error concealment is to find the lost MV in corrupted MB. Hence, we can reconstruct the corrupted MB with estimated MV and reference pixels without residual. The complexity of those algorithms which use zero, average or median of neighboring MVs as estimated MV is quite low. However, these low-complexity algorithms have worse performance as compared to BMA or overlapping BMA (OBMA) [21]. The BMA or OBMA obtain the better result with excellently high memory access and complexity to accumulate the best matching in the neighboring area. However, TEC is not always adequate for concealing errors in video sequences. This is especially true for video sequences with irregular motion, abrupt scene changes, and intra-coded image frames. Hence, the error concealment scheme is necessary

to integrate the TEC with SEC with mode decision. The mode decision determines which algorithm to be used to reconstruct the corrupted MB according the information in neighboring correct MBs, such as macroblock type, motion vectors and coded block pattern.

### 6.2.3 Multi-standard solution

DVB-H supports not only video standard of MPEG-2 and H.264/AVC, but also supports video standard of VC-1. An in-loop filter is defined in VC-1 with different algorithm with the one of H.264/AVC. They are different on filtering order. As we discussed in previous sections, the filtering order in H.264/AVC is vertical edge first. But the filtering order in VC-1 is horizontal edge first. Besides, deblocking filter is applied for each 4x4-block edges in H.264. And deblocking filter of VC-1 is applied to variable block size. Hence, the multi-standard deblocking filter for H.264/AVC and VC-1 is more complexity than the one for H.264/AVC and MPEG-2 we proposed in previous sections because the different standardized algorithm adopted by H.264/AVC and VC-1. For the mobile applications use DVB-H, hardware implemented video decoder which supports multi-standard can reduce the power consumption and complexity of coprocessor. Hence, the hardware integration for deblocking filter is a challenge for low-cost design.

# Bibliography


[1] P. List, A. Joch, J. Lainema, G. Bjøntegaard and M. Karczewicz, " Adaptive Deblocking Filter", IEEE Trans. Circuit Systems for Video Technology, vol. 13, pp. 614-619, July 2003.

[2] Y. Wang and Q. F. Zhu, " Error control and concealment for video communication: a review," Proceedings of the IEEE, vol. 86, no.5, pp 974-997, May 1998.

[3] Thomas Wiegand, Gray J. Sullivan, G. Bjøntegaard and Ajay Luthra, "Overview of the H.264/AVC video coding standard," IEEE Trans. Circuits System for Video Technology, Vol. 13, no. 7, pp. 560-576, July 2003.

[4] J.W. Suh, Y.S. Ho, "Error Concealment based on Directional Interpolation," IEEE Trans. on consumer Electronics, vol. 43, no. 3, pp. 295-302, Aug. 1997.

[5] Dimitris Agrafiotis, David R. Bull and C. Nishan Canagarajah, "Enhanced Error Concealment with Mode Selection," IEEE trans. on Circuits and Systems, vol. 16, issue 8, pp 960-973, Aug. 2006.

[6] Pei-Jun Lee, H.H. Chen and Liang-Gee Chen, "A new error concealment algorithm for H.264 video transmission," in Proc. 2004 Int. Symp. Multimedia video and Speech Processing, pp. 619~622, Oct. 2004.

[7] Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264/ISO/IEC 14496-10 AVC), Mar. 2003.

[8] ITU-T Recommendation H.263: "Video Coding for Low Bitrate Communication", March 1996.

[9] Miao Sima, Yuanhua Zhou and Wei Zhang, "an Efficient Architecture for Adaptive Deblocking Filter of H.264/AVC Video Coding" IEEE Transactions on Consumer Electronics, Vol. 50, Issue 1, pp. 292-296, Feb. 2004.

[10] Yu-Wen Huang, To-Wei Chen, Bing-Yu Hsieh , Tu-Chih Wang, Te-Hao Chang and Liang-Gee Chen, “Architecture Design for Deblocking Filter in H.264/JVT/AVC” International Conference on Multimedia and Expo(ICME’03), Vol. 1, pp. I-693-6, July 2003.

[11] ISO/IEC 14496-2:2001, “Information Technology – Generic Coding of Audio-Visual Object Part 2: Visual”, 3rd Ed. Annex F.3 – Post processing for coding noise reduction, Mar. 2003

[12] Tsu-Ming Liu, Wen-Ping Lee, Ting-An lin and Chen-Yi Lee, “A Memory-Efficient Deblocking Filter for H.264/AVC Video Coding,” IEEE International Symposium on Circuit and System (ISCAS’05), pp. 2140-2143, Kobe, Japan, May 2005.

[13] Wen-Ping Lee, Tsu-Ming Liu and Chen-Yi Lee, “A Joint Architecture of Error-Concealed Deblocking Filter for H.264/AVC Video Transmission,” IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT’07), Apr. 2007.

[14] Tsu-Ming Liu, Ting-An Lin, Sheng-Zen Wang, Wen-Ping Lee, Kang-Zheng Hou, Jiun-Yan Yang and Chen-Yi Lee, “An 865-μW H.264/AVC Video Decoder for Mobile Applications”, IEEE Asian Solid-state Circuit Conference (A-SSCC’05), pp. 301-304, HsinChu, Nov. 2005.

[15] ETSI TS 102 005, Digital Video Broadcasting (DVB); Specification for the use of Video and Audio Coding in DVB services delivered directly over IP.

[16] B. Sheng, W. Gao and D. Wu, “An Implemented Architecture of Deblocking Filter for H.264/AVC”, *IEEE International Conference on Image Processing,* Vol. 1, pp. 665-668, Oct. 2004.

[17] Shen-Yu Shih, Cheng-Ru Chang and Youn-Long Lin, “ A near optimal deblocking filter for H.264 advanced video coding.” *In Proc. IEEE ASP-DAC*, pp. 170-175, Jan. 2006.

[18] http://ftp3.itu.int/av-arch/jvt-site/draft_conformance/

[19] Yen-Lin Tung, Hsiu-Chen Shu and Jin-Jang Leou, “an Error Detection and Concealment Scheme for H.264 Video Transmission,” IEEE International Conference on Multimedia and Expo. (ICME’04), pp. 1735-1738, 2004.

[20] Tsu-Ming Liu and Chen-Yi Lee, “An Improved Soft-Input CAVLC Decoder for Mobile Communication Applications.” IEEE Asia-Pacific Conference on Circuits and Systems (APCCAS’06), pp583-586, Singapore, Dec. 2006.

[21] Donghyung Kim, Siyoung Yang and Jechang Jeong, “A New Temporal Error Concealment Method for H.264 using Adaptive Block Sizes,” IEEE International Conference on Image Processing, vol. 3, pp. 928 – 931, Sept. 2005.

# 作　者　簡　歷

姓名　　：李文平

性別　　：男

出生地　：台灣省苗栗縣

出生日期：1983.09.19

電話　　：0911-122523

學歷：　1989.9 ~ 1995.6　　苗栗縣立僑文國民小學

1995.9 ~ 1998.6　　苗栗縣立西湖國民中學

1998.9 ~ 2001.6　　苗栗市立苗栗高級中學

2001.9 ~ 2005.6　　國立交通大學 電子工程系 學士

2005.9 ~ 2007.6　　國立交通大學 電子研究所 系統組 碩士

# 得　獎　事　績

2005/05　2005 全國 IC 設計競賽設計完整獎

# 發 表 論 文

- **Wen-Ping Lee**, Tsu-Ming Liu and Chen-Yi Lee, "A Joint Architecture of Error-Concealed Deblocking Filter for H.264/AVC Video Transmission," *IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT'07)*, Apr. 2007.

- Tsu-Ming Liu, **Wen-Ping Lee**, Ting-An lin, Chen-Yi Lee, "A Memory-Efficient Deblocking Filter for H.264/AVC Video Coding," *IEEE International Symposium on Circuit and System (ISCAS'05)*, pp. 2140-2143, Kobe, Japan, May 2005.

- Tsu-Ming Liu, **Wen-Ping Lee**, Chen-Yi Lee, "An Area-Efficient and High-Throughput Deblocking Filter for Multi-standard Video Applications," *IEEE International Conference on Image Processing (ICIP'05)*, pp. III-1044 – 1047, Genoa, Italy, Sept. 2005.

- Tsu-Ming Liu, Ting-An Lin, Sheng-Zen Wang, **Wen-Ping Lee**, Kang-Zheng Hou, Jiun-Yan Yang and Chen-Yi Lee, "An 865-μW H.264/AVC Video Decoder for Mobile Applications", *IEEE Asian Solid-state Circuit Conference (A-SSCC'05)*, pp. 301-304, HsinChu, Nov. 2005.

- Tsu-Ming Liu, **Wen-Ping Lee** and Chen-Yi Lee, "An In/Post-Loop Deblocking Filter with Hybrid Filtering Schedule," *IEEE Transaction on*

*Circuit and System for Video Technology.*

- Tsu-Ming Liu, Ting-An Lin, Sheng-Zen Wang, **Wen-Ping Lee**, Kang-Cheng Hou, Jiun-Yan Yang and Chen-Yi Lee, " A 125μW, Fully Scalable MPEG-2 and H.264/AVC Video Decoder for Mobile Applications," *IEEE Journal of Solid-State Circuits, vol.42, no. 1, pp. 160-169, Jan. 2007.*