

國立交通大學

電子工程學系 電子研究所碩士班

碩 士 論 文

應用於數位電視之視訊雙標準解碼器設計與實現



**Design and Implementation of Dual Mode
Video Decoder for Digital TV Applications**

學生：林亭安

指導教授：李鎮宜 教授

中華民國九十四年六月



應用於數位電視之視訊雙標準解碼器設計與實現

Design and Implementation of Dual Mode

Video Decoder for Digital TV Applications

研究生：林亭安

Student : Ting-An Lin

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



Submitted to Institute of Electronics
College of Electrical Engineering and Computer Science
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Science
in

Electronics Engineering

June 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年六月



應用於數位電視之視訊雙標準解碼器設計與實現


學生：林亭安

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，比起 MPEG-2，H.261 及 H.263，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。在本論文中，我們實作了一個 H.264 的硬體影像解碼器。我們運用各樣的技術及架構，來提高單位時間資料流通量以及降低功率的消耗，以期達到未來不論在數位電視、無線傳輸等方面的影像解碼需求。此外，因為多標準解碼器已成為設計潮流，我們把目前最流行且運用在 DVD 的影像標準規格—MPEG-2 也納入我們的設計範圍。我們期望運用硬體共用的技巧，在不花費太多額外的硬體架構下，用現有的硬體單來實現 MPEG-2 的硬體解碼功能。

從系統設計的角度，在這篇論文我們首先提出了一個雙標準的影像解碼區塊圖，說明我們共用了那些硬體單元，及主要資料的流動路徑。我們採用了複合式 4 乘 4 區塊管線化系統架構來減少區間暫存器的使用量並加速系統的單位時間資料流通量。我們提出的有效率解碼順序也能減少在移動補償及空間預測模組之記憶體存取次數。在解碼的資料流動路徑中，剩餘像素及預測像素值的相加處發生的資料同步問題我們並

提出了一個可變長度先進先出緩充暫存器的解決方案。我們也提出了一個利用 CBP 參數來節省功率的方法。

在模組架構設計方面，我們也針對此解碼器的各模組做了介紹。在資料流分析單元，我們採用階層化的設計方式，不但架構簡單易於設計，也可以有效降低功率消耗。暫存器共用的技巧也被應用於資料流分析單元，而達到共用暫存器的目的。在空間預測模組的設計中，我們提出了三種並行的暫存器架構以幫助空間預測的運算，記憶體存取次數也可以因此減少到最低。其餘模組的設計也包含在這個論文中，許多的技巧也被應用在節省記憶體存取次數及加快單位時間資料流通率。

最後本論文利用 UMC 0.18um 1P6M 製程技術實作了這顆 H.264/MPEG-2 雙模式影像解碼晶片。根據合成與佈局繞線結果，這顆晶片的大小為 $3.9 \times 3.9 \text{mm}^2$ ，總邏輯閘數為 491K，最大操作頻率可達 83.3MHz。支援即時播放 720pHD 的 H.264 影像串流於 56MHz，720pHD 的 MPEG-2 影像串流於 35.7MHz 在每秒 30 張的規格下。




Design and Implementation of Dual Mode Video Decoder for Digital TV Applications

Student : Ting-An Lin

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. Compared with MPEG-2, H.261, and H.263, H.264 provides better coding efficiency, which means that it provides better image quality at the same coding rate. In this thesis, we implemented an H.264 video decoder. We adopted various techniques and architectures to accelerate the decoding throughput and the reduction on power consumption, to achieve the demands on future digital TV and wireless communication. Besides, because the multi-mode video decoder is a design trend, the video coding standard – MPEG-2 which has been widely used for DVD video standard is included in our design. We expect to use some hardware-sharing techniques to implement the MPEG-2 video decoder in the situation that only a few additional hardware modules are required.

From the system point of view, in this thesis we first proposed a block diagram for dual mode video decoder, to illustrate the functional blocks we used, and the data path of our work. The efficient decoding ordering we proposed can reduce the memory access times

on motion compensation and intra prediction modules. In the decoding loop, the synchronization problem occurs at the adder that adds the residual pixel values with the predicted pixel values. We proposed a variable-length FIFO architecture for the solution to this synchronization problem. We also proposed a way to save power by exploiting the system parameter “coded-block-pattern”.

In the architecture design, we give descriptions on all the important modules of this decoder. We adopt a hierarchical structure for the syntax parser design. Hierarchical structure make the parser easy design, the clock-gating power reduction technique can also be effectively applied to save power in this structure. The register sharing technique is also applied in the syntax parser unit in order to reduce the amount of register required. In intra predictor design, we proposed three kinds of buffers to reduce the design complexity on intra predictor. The memory access times can be reduced to minimum for the help of these 3 buffers. Besides, other important modules like motion compensation, de-blocking filters are also included in this thesis. Many techniques are also applied to save memory access times and to increase the throughput in these modules.

At last we implemented this dual mode H.264/MPEG-2 video decoder in UMC 0.18um 1P6M process. According to the implementation result, the size of his chip is $3.7 \times 3.7 \text{ mm}^2$, total gate count is 491K, and the maximum working frequency is 83.3MHz. This chip supports real time decoding 720pHD H.264 video sequence in 56MHz, 720pHD MPEG-2 video sequence in 35.7MHz in 30fps.

誌 謝

在這畢業的季節，想起自專題生開始的這三個年頭，真是充滿了酸甜苦辣。隨著這一本論文漸漸的完成，我的碩士班研究生涯也漸漸告一段落了。這段時間要感謝的人真是太多了。因為有你們的指導、協助及關心，才能使我有今日的成長。

首先要感謝的是我的恩師李鎮宜教授。在老師不厭其煩、耐心的教導下，我除了獲得知識、做研究的方法以外，更學習到了老師積極的人生觀。每天早上老師不間斷的晨泳精神，更是讓我欽佩不已！老師願意花這麼多時間耐心聽我們每一次的報告，並在每一次的報告中都給與了我們適當及中肯的建議，導引我們的研究到正確的方向。老師也真是一位好老師，能跟隨老師做研究也是我進研究所最幸運的事了！

不可或缺的，我要感謝我的爸爸媽媽及哥哥嫂嫂。感謝爸爸媽媽的用心栽培，才有我今日所能踏出的每一步。在上研究所後，也常常因為時間調配不好，很久才回家一次，但每次回家卻都聽到正面的鼓勵，真是謝謝爸媽的強力支持！謝謝哥哥嫂嫂，在我最後要忙不過來的時候還開車幫我搬宿舍，東西超多但全都麻煩你們幫我搬回家，真是辛苦你們了，也謝謝你們的幫忙！

接著要感謝的，就是 Si2 實驗室的大家。Si2 是以「操」出名的，能有幸待在這個認真打拼的實驗室，這兩年的收獲真是相當豐富的。感謝從專題生開始時帶著我做研究的 YY、blues、黎峰及最後人超好的 mingle 學長。這兩年間除了享受到完善的硬體資源以外，更重要的是在做研究的每一個階段，都能得到學長們豐富的經驗傳承。我希望在未來的求學或工作上，也能秉持著學長們殷勤不懈的奮鬥精神並將我所學的繼續傳承給學弟妹們。

感謝和我同一屆的勝仁、凱立、瑋哲、林宏及盧忠。感謝你們在這兩年間的互相關照，在最後一年共同於論文上奮鬥。使我在寧靜的深夜寫論文時，能聽見你們的叫聲而不致打瞌睡。也恭喜你們可以和我一起踏上人生另一階段，我也不會忘記你們

這些戰友的！

最後，也是同等重要的，在最後一年間，我一定要感謝的朋友們——佳音、思珏、筱君、聖威、新中與我的六人行、大清交及明新團契的各位、每週一 morning star 的成員們、最棒的小詩班成員們、及高級班的大家。謝謝你們帶給我不知多少發的宵夜、夜唱、夜烤、看星星、看海、吃好吃的、出遊、及每一件教會的事工。謝謝你們的每一件禮物、每一次用心準備的活動與送舊。這些數都數不清的歡樂時光也是我最後一年在離別時最捨不得的珍貴回憶。也因為有了你們，我的研究生生活變得因此充實而快樂。歡笑的每一天中有你們的影子、悲傷的每一天也更充滿了你們的身影，謝謝你們陪著我過完這豐富的一年，就讓一切的感激都獻給神吧！



Contents

CHAPTER 1 INTRODUCTION.....	1
1.1 MOTIVATION.....	1
1.2 MPEG-2 STANDARD OVERVIEW.....	2
1.2.1 Profiles and Levels.....	2
1.2.2 Picture types.....	3
1.2.3 Encoder/Decoder Block Diagram.....	4
1.2.4 Bit-stream structure.....	5
1.3 H.264/AVC STANDARD OVERVIEW.....	5
1.3.1 Profiles and Levels.....	6
1.3.2 Encoder/Decoder Block Diagram.....	7
1.3.3 Bit-stream structure.....	8
1.4 THESIS ORGANIZATION.....	9
CHAPTER 2 OVERVIEW OF MPEG-2 AND H.264/AVC DECODING FLOW.....	11
2.1 OVERVIEW OF MPEG-2 DECODING FLOW.....	11
2.1.1 Variable length decoding.....	11
2.1.2 Inverse scanning process.....	12
2.1.3 Inverse quantization.....	13
2.1.4 Inverse Discrete-Cosine-Transform (IDCT).....	15
2.1.5 Motion compensation.....	16

2.2	OVERVIEW OF H.264/AVC DECODING FLOW	17
2.2.1	<i>Entropy decoding</i>	17
2.2.2	<i>Inverse scanning process</i>	18
2.2.3	<i>Inverse quantization & inverse Hadamard transform</i>	19
2.2.4	<i>Inverse Integer Discrete Cosine Transform</i>	20
2.2.5	<i>Intra prediction</i>	20
2.2.6	<i>Motion compensation</i>	22
2.2.7	<i>De-blocking filter</i>	24
CHAPTER 3 SYSTEM DESIGN OF MPEG-2 AND H.264/AVC DECODER.....		27
3.1	MPEG-2 AND H.264/AVC COMBINED SYSTEM DECODING FLOW	27
3.2	HYBRID 4X4-BLOCK LEVEL PIPELINE WITH INSTANTANEOUS SWITCHING SCHEME FOR H.264/AVC DECODER	28
3.2.1	<i>Hybrid 4x4-Block Level Pipeline Architecture</i>	28
3.2.2	<i>Instantaneous Switching Scheme</i>	32
3.3	EFFICIENT 1X4 COLUMN-BY-COLUMN DECODING ORDERING.....	33
3.3.1	<i>Analysis on inter prediction unit</i>	34
3.3.2	<i>Analysis on intra prediction unit</i>	37
3.4	PREDICTION/RESIDUAL SYNCHRONIZATION SCHEME.....	38
3.5	POWER SAVING BY EXPLOITING CODED-BLOCK-PATTERN.....	40
3.6	NOVEL USER-DETERMINABLE LOW POWER MODE EXPLORATION.....	42
CHAPTER 4 ARCHITECTURE DESIGN OF MPEG-2/H.264/AVC DECODER		45
4.1	MPEG-2 & H.264/AVC COMBINED SYNTAX PARSERS	45
4.1.1	<i>Low-Power Hierarchical Parser Design</i>	45
4.1.2	<i>Register Sharing Parser Design</i>	48
4.2	EXP-GOLOMB DECODER FOR H.264/AVC SYNTAX PARSER	51

4.2.1	<i>Circuit design of Exp-Golomb Decoder</i>	52
4.2.2	<i>Reusability of the Exp-Golomb Decoder</i>	53
4.3	H.264/AVC INTRA PREDICTOR	54
4.3.1	<i>Low-Power Memory Fetch Upper Buffer Design</i>	54
4.3.2	<i>Reusable Left Buffer Design</i>	57
4.3.3	<i>Reusable Corner Buffer Design</i>	58
4.3.4	<i>Intra Predictor for Directional Based Modes</i>	60
4.3.5	<i>Intra Predictor for DC Mode</i>	61
4.3.6	<i>Intra Predictor for Plane Prediction</i>	61
4.4	MPEG-2 & H.264/AVC INVERSE DCT	64
4.4.1	<i>2-Stage IDCT Architecture</i>	65
4.5	MPEG-2 & H.264/AVC COMBINED MOTION COMPENSATOR	66
4.5.1	<i>Motion Compensation Engine</i>	66
4.5.2	<i>Interpolator</i>	67
4.6	MPEG-2 & H.264/AVC COMBINED DE-BLOCKING FILTER	69
4.6.1	<i>Triple-Mode Decision</i>	70
4.6.2	<i>Slice and content memory</i>	71
4.6.3	<i>Hybrid scheduling</i>	72
CHAPTER 5 CHIP IMPLEMENTATION FOR DIGITAL TV APPLICATIONS.....		77
5.1	SYSTEM SPECIFICATION	77
5.2	DESIGN FLOW	78
5.3	IMPLEMENTATION RESULT	79
5.4	MEASUREMENT RESULTS AND COMPARISON	81
CHAPTER 6 CONCLUSION AND FUTURE WORK.....		83
6.1	CONCLUSION	83

6.2 FUTURE WORK..... 84

BIBLIOGRAPHY 85



List of Figures

FIG.1.1 A SIMPLE BLOCK DIAGRAM OF MPEG-2 VIDEO ENCODER	4
FIG.1.2 A SIMPLE BLOCK DIAGRAM OF MPEG-2 VIDEO DECODER	5
FIG.1.3 HIERARCHICAL BIT-STREAM STRUCTURE OF MPEG-2 VIDEO	5
FIG.1.4 H.264 BASELINE, MAIN, AND EXTENDED PROFILE	6
FIG.1.5 A SIMPLE BLOCK DIAGRAM OF H.264/AVC VIDEO ENCODER.....	7
FIG.1.6 A SIMPLE BLOCK DIAGRAM OF H.264/AVC VIDEO DECODER.....	8
FIG. 1.7 HIERARCHICAL STRUCTURE OF H.264 VIDEO BIT-STREAM.....	9
FIG. 2.1 INVERSE SCAN PATTERN (A) <i>ALTERNATE_SCAN=0</i> (B) <i>ALTERNATE_SCAN=1</i>	13
FIG. 2.2 INVERSE QUANTIZATION PROCESS	14
FIG. 2.3 MOTION COMPENSATION PROCESS.....	17
FIG. 2.4 ZIG-ZAG SCAN.....	18
FIG. 2.5 INTRA_4x4 PREDICTION MODES	21
FIG. 2.6 INTRA_16x16 PREDICTION MODES	22
FIG. 2.7 MACROBLOCK AND SUB-MACROBLOCK PARTITIONS.....	23
FIG. 2.8 UP TO 1/4 MOTION VECTOR RESOLUTION ($MV=(+1.50, -0.75)$).....	23
FIG. 2.9 INTERPOLATION FOR PIXEL VALUES	24
FIG. 2.10 EDGE FILTERING ORDER IN A MACROBLOCK.....	25
FIG. 2.11 ADJACENT PIXELS TO HORIZONTAL AND VERTICAL BOUNDARIES	25
FIG. 3.1 MPEG-2/H.264 COMBINED DECODER DIAGRAM	28
FIG.3.2 ADDITIONAL PROCESSING CYCLES REQUIRED FOR 4x4-SUB-BLOCK-LEVEL PIPELINE	

PARALLELISM	29
FIG. 3.3 AN EXAMPLE OF THE PIPELINING SCHEDULE	33
FIG. 3.4 4x1 ROW-BY-ROW DECODING ORDERING	34
FIG. 3.5 1x4 COLUMN-BY-COLUMN DECODING ORDERING	34
FIG. 3.6 16 INITIALIZATION PROCESSES IN INTER PREDICTED MACROBLOCK UNDER 4x1 ROW-BY-ROW DECODING ORDERING	36
FIG. 3.7 8 CONTENT SWITCHES IN INTER PREDICTED MACROBLOCK UNDER 1x4 COLUMN-BY-COLUMN DECODING ORDERING	36
FIG. 3.8 REDUCTION ON MEMORY ACCESS OF THE INTRA PREDICTED MACROBLOCK.....	38
FIG. 3.9 A VARIABLE-LENGTH FIFO IS REQUIRED FOR THE SYNCHRONIZATION BETWEEN INTRA/INTER PREDICTOR AND IDCT.....	39
FIG. 3.10 OPERATION OF VARIABLE-LENGTH FIFO AS A SYNCHRONIZER.....	40
FIG. 3.11 POWER SAVING BY EXPLOITING CODED-BLOCK-PATTERN.....	41
FIG. 3.12 QP VERSUS BITRATE AND THE PERCENTAGE OF ALL ZERO COEFFICIENT BLOCKS.....	42
FIG. 4.1 HIERARCHICAL SYNTAX PARSER.....	46
FIG. 4.2 AN EXAMPLE WAVEFORM OF THE ENABLE SIGNALS IN SYNTAX PARSER.....	47
FIG. 4.3 POWER REDUCTION ON SYNTAX PARSER.....	48
(A) WITHOUT APPLYING GATED CLOCK (B) WITH APPLYING GATED CLOCK	48
FIG. 4.4 REGISTER SHARING TECHNIQUE	50
FIG. 4.5 REGISTER NUMBER REDUCTION ON SYNTAX PARSER	50
FIG. 4.6 CIRCUIT DESIGN OF EXP-GOLOMB DECODER	53
FIG. 4.7 EXP-GOLOMB DECODER SHARED FOR ALL MODULES IN THE SYNTAX PARSER.....	54
FIG. 4.8 OPERATION OF UPPER BUFFER (A).....	56
FIG. 4.9 OPERATION OF UPPER BUFFER (B).....	56
FIG. 4.10 OPERATION OF LEFT BUFFER (A).....	57
FIG. 4.11 OPERATION OF LEFT BUFFER (B)	58

FIG. 4.12 OPERATION OF CORNER BUFFER (A)	59
FIG. 4.13 OPERATION OF CORNER BUFFER (B)	59
FIG. 4.14 INTRA PREDICTOR FOR DIRECTIONAL BASED MODES	60
FIG. 4.15 INTRA PREDICTOR FOR DC MODE	61
FIG. 4.16 SLOPE CALCULATOR FOR PLANE PREDICTION.....	62
FIG. 4.17 REQUIRED CALCULATION FOR PLANE PREDICTION	63
FIG. 4.18 INTRA PREDICTOR FOR PLANE PREDICTION	64
FIG. 4.19 INTRA PREDICTOR FOR PLANE PREDICTION	64
FIG. 4.20 2-STAGE IDCT ARCHITECTURE.....	66
FIG. 4.24 (A) SLICE MEMORY WITH GRID OR SHADED REGION AND (B) CONTENT MEMORY WITH BLACK-DOTTED REGION.....	72
FIG. 4.25 HYBRID SCHEDULING METHOD.....	73
FIG. 4.26 THE PARTITIONED MB AND EACH TIME INSTANCE WHEN APPLYING THE HYBRID SCHEDULING METHOD.....	74
FIG. 4.27 THE BLOCK DIAGRAM AND DATA FLOW OF THE MPEG-2/H.264 COMBINED DE-BLOCKING FILTER	75
FIG. 5.1 DESIGN FLOW FROM SYSTEM SPECIFICATION TO PHYSICAL-LEVEL.....	78
FIG. 5.2 LAYOUT OF THIS WORK.....	80



List of Tables

TABLE 1.1: MPEG-2 LEVELS: PICTURE SIZE, FRAME-RATE AND BIT RATE CONSTRAINTS.....	3
TABLE 3.1: TRADE-OFF BETWEEN PROCESSING CYCLES AND BUFFER COST	30
TABLE 3.3 POWER DISSIPATED BY BUFFERS BETWEEN PIPELINE STAGES.....	30
TABLE 3.4 SUMMARY OF PIPELINE PARALLELISM APPLIED	32
TABLE 3.5 POWER PROFILING OF DECODING H.264 VIDEO BIT-STREAM (UNIT:MW)	44
TABLE 3.6 LOW POWER MODE EXPLORATION	44
TABLE 4.1 NUMBER OF REGISTER NEEDED FOR MPEG-2/H.264 SYNTAX PARSER	49
TABLE 4.2 AREA AND POWER REDUCTION ON REGISTER-SHARING VERSION	51
TABLE 4.3 THE ASSIGNMENT OF BIT STRINGS TO CODE VALUE	52
TABLE 4.3 AREA AND POWER REDUCTION ON REGISTER-SHARING VERSION	65
TABLE 5.1 SIMPLE PROFILE @ MAIN LEVEL OF MPEG-2 SYSTEM.....	77
TABLE 5.2 BASELINE PROFILE @ LEVEL 3.2 OF H.264 SYSTEM	77
TABLE 5.3 CHIP DETAILS.....	81
TABLE 5.4 POWER REPORT.....	81
TABLE 5.5 COMPARISONS	82



Chapter 1

Introduction

1.1 Motivation

H.264 is the new video coding standard developed by MPEG (Moving Picture Experts Group) and VCEG (Video Coding Experts Group) that promises to outperform the earlier MPEG-4 and H.263 standard, providing better compression of video images. Because of its high coding efficiency, it has great potential to be the video standard of the next generation. For content storage, like HD-DVD and Blu-Ray in the next generation (both use 450 nm Blue-Laser diode), standardized the H.264 with MPEG-2 and WMV-9 with its digital content. The video content storage for Sony PS3, which will be phased in in 2006, adopts H.264 as its video compression standard as well. For digital broadcasting, like DVB-H of handheld digital TV standardized by ETSI, combined the wireless communication standard – COFDM with the newest video coding standard – H.264, trying to migrate the digital video technology to portable. Another digital broadcasting standard for Set-Top-Box like DVB-S2 also exploits the Forward-Error-Correction (FEC) technology (LDPC used) with the H.264.

The H.264 seems so popular and with high potential to be the video coding standard of so many applications in the next generation, the demand of the decoder for H.264 is obvious. However, the penalty for its high coding efficiency is the large amount of computation. From our survey on some technical papers, it seems that the efficiency of the

platform-based approach is not that enough to achieve the required throughput of decoder. Thus we try to design a dedicated decoder which suits for the digital TV applications.

Multimode decoder is the trend, and we can see this trend everywhere. For example, we can easily find a DVD player which supports decoding VCD, MP3, or even JPG images. To design a multimode decoder is then become important for the decoder designs. Thus in our design of H.264 decoder, we try to design our decoder multimode. We choose MPEG-2 because that MPEG-2 is the video standard of DVD, has been widely used today. The hardware sharing issue, and many multimode functional block has become great issues in our decoder design.

1.2 MPEG-2 Standard Overview

MPEG-2 is mainly divided into 3 parts, the MPEG system, MPEG video, and MPEG audio. In this thesis only MPEG video is concerned.

1.2.1 Profiles and Levels

MPEG-2 video is an extension of MPEG-1 video. MPEG-1 is targeted at video with bit-rate up to about 1.5 Mbits/s. Compared with MPEG-1, MPEG-2 provides some extra coding tools that it can support bit rates of various range. Scalable coding is also included in the MPEG-2 standard. There are total 5 profiles in the MPEG-2 standard, which are Simple, Main, SNR, Spatial, and High profiles. Above these profiles, the Main profile is the most widely used profile. It supports I, P, and B pictures, uses 4:2:0 chroma sampling format, but is non-scalable. Main profile is subdivided into 4 levels, which are low, Main, High-1440, and High levels. The picture sizes, frame rate, and bit rate constraints for different levels are summarized in Table 1.1.

Table 1.1: MPEG-2 levels: Picture size, frame-rate and bit rate constraints

Level	Max. frame size	Max frame rate	Max bit rate
Low	352 x 288	30	4
Main	720 x 576	30	15
High-1440	1440 x 1152	60	60
High	1920 x 1152	60	80

1.2.2 Picture types

MPEG-2 contains 3 picture types, the I-picture, P-picture, and B-picture.

Intra picture (I-picture) is the picture that coded without reference to other pictures. It uses the reduction of spatial redundancy to achieve compression. Because that I-picture can be decoded independently without referencing to other pictures, I-pictures can be used as the access points in the bit-stream where the decoder can start to decode.

Predictive picture (P-picture) is the picture that coded by motion vectors referencing to previous I or P-pictures and residuals. It uses the reduction of temporal redundancy to achieve compression. Besides P macroblock blocks in the P-pictures, I macroblock can also exist in the P-pictures. Thus the spatial redundancy can also be reduced to achieve compression in the P-picture. P-pictures offer increased compression compared to I-pictures.

Bidirectionally-predictive picture (B-picture) is similar to P-picture. Different from P-pictures, the reference frames can be either the previous picture, next picture, or both. It offers highest degree of compression.

1.2.3 Encoder/Decoder Block Diagram

The encoding process of MPEG-2 video contains the motion estimation and residual coding. Fig. 1.1 shows the simple block diagram of the MPEG-2 encoder. An embedded decoder inside the encoder calculates the result of the motion compensation so that the residual can be calculated by subtracting the input image with the motion compensated image. A Discrete-Cosine-Transform transfers the residual values to frequency-related domain. By quantizing the transferred coefficients, a Huffman run-length coder is responsible for coding the quantized coefficients and then outputting.

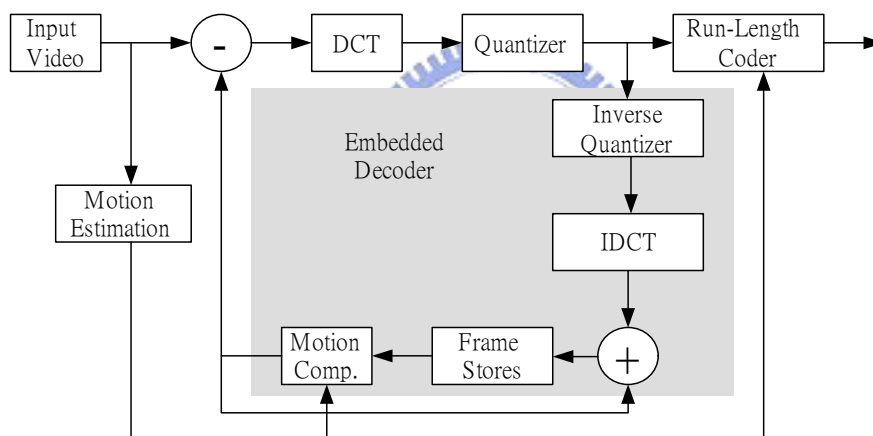


Fig.1.1 A simple block diagram of MPEG-2 video encoder

Fig. 1.2 shows the simple block diagram of an MPEG-2 video decoder. A parser with Huffman run-length decoder decodes the motion vectors and quantized residual values. After inverse quantization and inverse DCT transform, the decoder calculates the residual values. By adding the motion compensated pixel values, the decoder can recover the original picture. At the time when the decoder output the decoded image, a copy of the picture must be stored into the frame buffers for the motion compensation process on next picture. The detailed decoding process will be described in section 2.1.

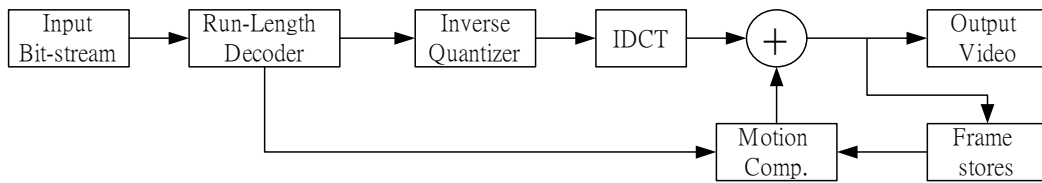


Fig.1.2 A simple block diagram of MPEG-2 video decoder

1.2.4 Bit-stream structure

Each picture is divided into several slices. Each slice is divided into several macroblocks. Each macroblock is further divided into blocks. A block is a group of 8x8 pixels, the smallest processing unit of the MPEG-2 system. Fig. 1.3 shows the hierarchical bit-stream structure of MPEG-2.

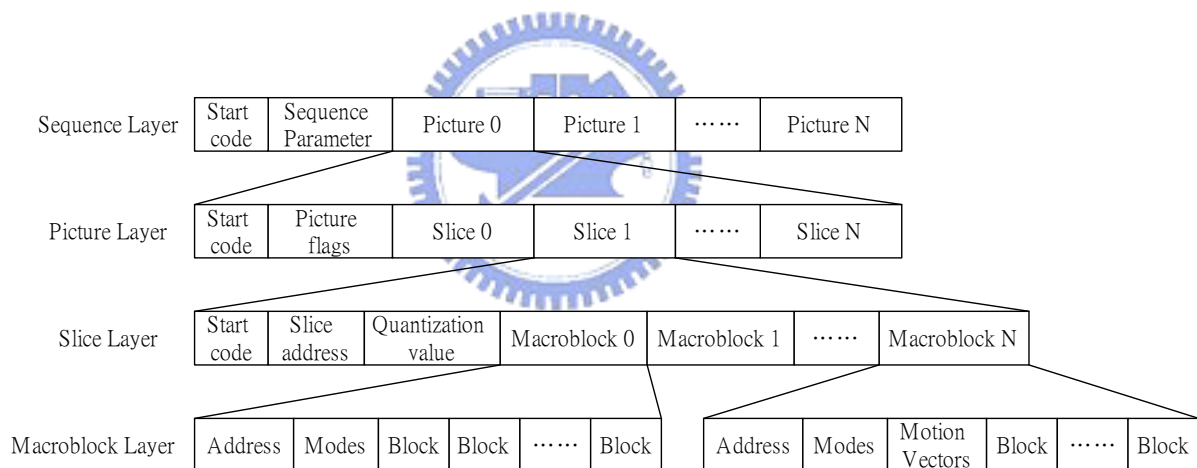


Fig.1.3 Hierarchical bit-stream structure of MPEG-2 video

1.3 H.264/AVC Standard Overview

H.264/AVC is a standard only for videos. Its extreme low data rate is achieved by several complex techniques and algorithms such as up to 1/4 resolution for luma and 1/8 for chroma on motion vector, several block size from 4x4 to 16x16, several modes in inter/intra prediction, CAVLC, or CABAC in context-adaptive entropy coding.

1.3.2 Encoder/Decoder Block Diagram

The encoding process for H.264/AVC video is more complex than the encoding process of the MPEG-2 video. Fig. 1.5 shows the simple block diagram of the H.264/AVC encoder. Same as MPEG-2 encoder, an embedded decoder exists inside the encoder that calculates the result of the motion compensation and intra prediction at the decoder side. With this embedded decoder, the encoder can foresee the decoded result and precisely calculate the residual pixel values without mismatch to the decoder. Besides inter prediction (motion compensation), intra prediction is also an important parts that tries to reduce the spatial redundancy to increase coding efficiency. Several intra prediction modes can be used for the intra predictor, and the prediction mode is decided by a mode decision block at the proceedings of the intra predictor. Not only intra prediction, the choices of the motion compensator are a lot as well. Various block sizes, multiple reference frames, short/long term prediction, and the motion vectors are all decided by motion estimation block. With these 2 strong prediction paths, the residual pixels values calculating from subtracting the input video with the prediction pixel values is closer to zero. After DCT transformation, quantization process, the entropy decoder at last reduces the coding redundancy effectively and then outputs the coded pictures.

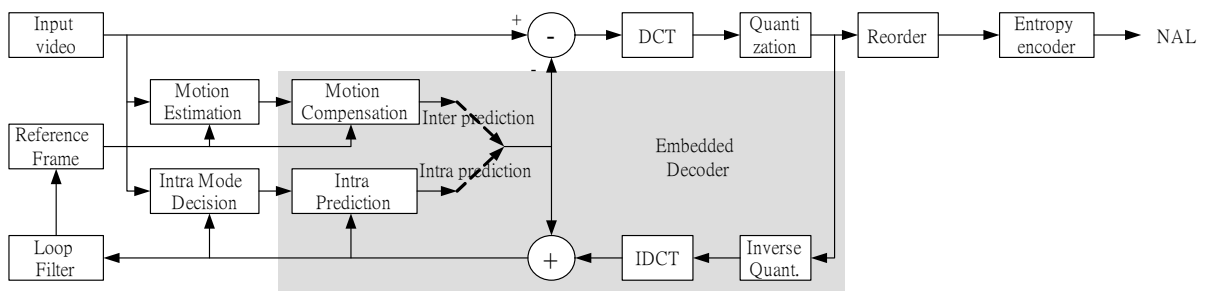


Fig.1.5 A simple block diagram of H.264/AVC video encoder

Compared with the encoder, the decoder is simpler because it lacks the decision parts like motion estimator and the intra mode decision parts. Fig. 1.6 shows a simple block diagram of the H.264/AVC video decoder. After entropy decoding the input bit-stream, the inverse quantization process and IDCT transformation transferred the bit-stream data into residual pixel values. By adding the predicted pixel values from intra predictor or motion compensator, an in-loop filter smoothed the blocking effects and then to both the output buffer and frame buffer for future reference. The details of the decoding process will be described in section 2.2.

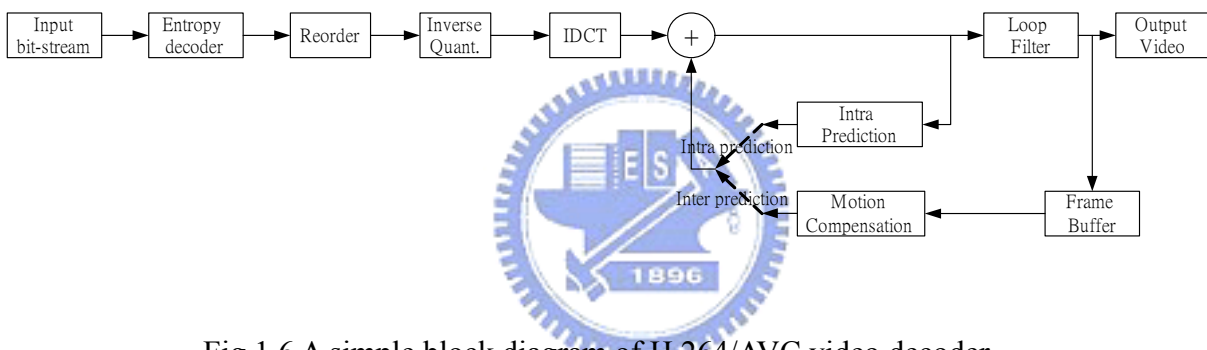


Fig.1.6 A simple block diagram of H.264/AVC video decoder

1.3.3 Bit-stream structure

Same as MPEG-2 bit-stream structure, the H.264 bit-stream is structured hierarchically, from block-level to video sequence level. Different from MPEG-2 which is the 8x8-block based system, the smallest block size in H.264/AVC system is the group of 4x4 pixels. Reference to the annex B in the H.264 standard [7], as Fig. 1.7 shows, data are all packed into NAL units. An NAL syntax element is attached in the front of each NAL unit. Each NAL unit contains an NAL unit header, which indicates the NAL unit type of the following data in this NAL unit, and the type of the RBSP (Raw Byte Sequence Payload) it contains. There're several types of RBSP. For example, the SPS (sequence parameter set), PPS

(picture parameter set), and Slice layer RBSP. Slice layer RBSP includes slice header, slice data, and sometimes slice ID or redundant picture count of the partitioned slice layer. Slice data is composed of macroblocks, each consists of prediction modes (in intra macroblock) or sub-macroblock type, motion vectors (in inter macroblock) and the 4x4 block based residual data, which contributes the size of the H.264 bit-stream the most.

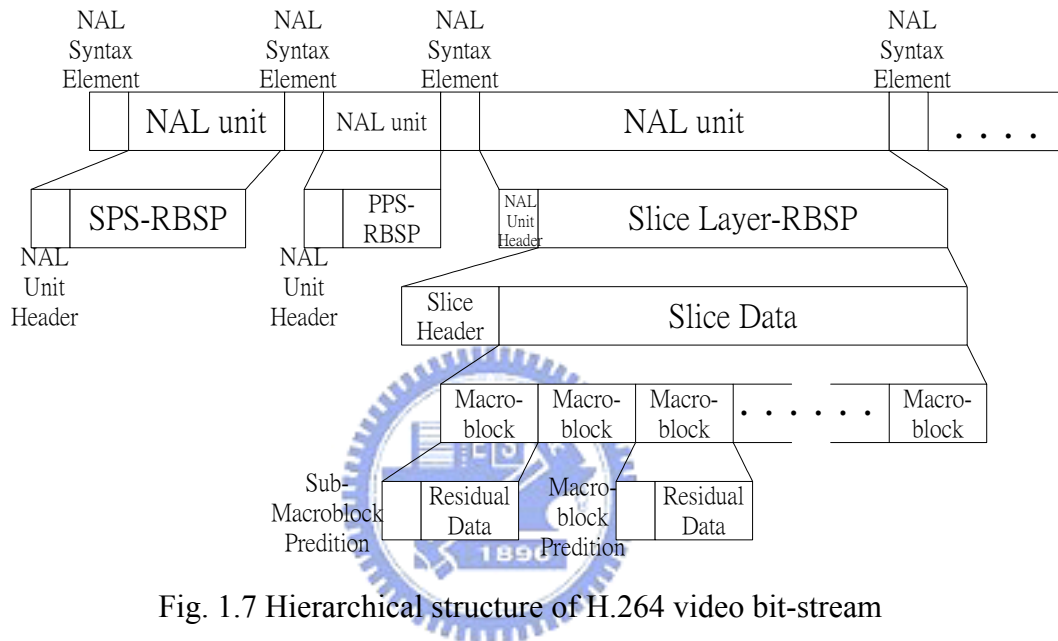


Fig. 1.7 Hierarchical structure of H.264 video bit-stream

1.4 Thesis Organization

This thesis is organized as follows. At first, the overview of the MPEG-2 and H.264/AVC decoding flow is described in Chapter 2. Chapter 3 gives the system level design consideration and some system-level schemes in this work, like pipeline architecture, decoding ordering, system synchronization, low power mode exploration, and low power design between modules. Then, details of the architecture designs of each functional block are described in Chapter 4. Finally, the implementation details, conclusion and summary are presented in Chapter 5 and Chapter 6, respectively.



Chapter 2

Overview of MPEG-2 and H.264/AVC

Decoding Flow

The overviews of MPEG-2 decoding flow and H.264/AVC decoding flow will be given in this chapter. Though there exists some differences between the decoding flow of MPEG-2 and H.264/AVC, similarities like inverse discrete cosine transform, inverse quantization, or motion compensation can still be found. In the system point of view, to make good use of every functional block suits for both systems is an important issue and good innovation of designing a multimode video decoder.



2.1 Overview of MPEG-2 Decoding Flow

The decoding process is strictly defined in the standard. With the exception of the Inverse Discrete Cosine Transform (IDCT) the decoding process is defined such that all decoders shall produce numerically identical results. As Fig. 1.2 shows, the decoding process mainly includes variable length decoding, inverse scanning process, inverse quantization, inverse DCT, and motion compensation.

2.1.1 Variable length decoding

The DC coefficients are separated from other coefficients. For DC coefficients, a predictor is used for the prediction of the DC coefficients. The predictor shall be reset to a certain value at the start of a slice, a non-intra macroblock is decoded, or a macroblock is skipped. The differential value (`dc_dct_differential`) is coded in the bit-stream. Thus the

decoder can calculate the DC coefficients (QFS[0]) by

$$QFS[0] = dc_dct_pred[cc] + dct_diff;$$

$$dc_dct_pred[cc] = QFS[0];$$

Where dc_dct_pred are the values of the 3 predictors, $Y(cc=0)$, $Cb(cc=1)$, and $Cr(cc=2)$. The dct_diff is the transformed value from $dc_dct_differential$.

For other coefficients, by table lookup of two VLC tables the values of “*run*” and “*level*” can be decoded. Then the coefficients in a macroblock can be recovered by run-length decoding process as the follows.

```
eob_not_read = 1;
while(eob_not_read){
  < decode VLC, decode Escape coded coefficient if required >
  if(< decoded VLC indicates End of block >){
    eob_not_read = 0;
    while(n < 64){
      QFS[n] = 0;
      n = n + 1;
    }
  }else{
    for(m = 0; m < run; m ++){
      QFS[n] = 0;
      n = n + 1;
    }
    QFS[n] = signed_level
    n = n + 1;
  }
}
```



2.1.2 Inverse scanning process

After total 64 coefficients are decoded by the Huffman run-length VLC decoder described above, the inverse scanning process inverse scanned these coefficients to a single 8x8 block. 2 scan patterns determined by parameter “*alternate_scan*” can be used. Fig. 2.1 shows these 2 scan patterns.

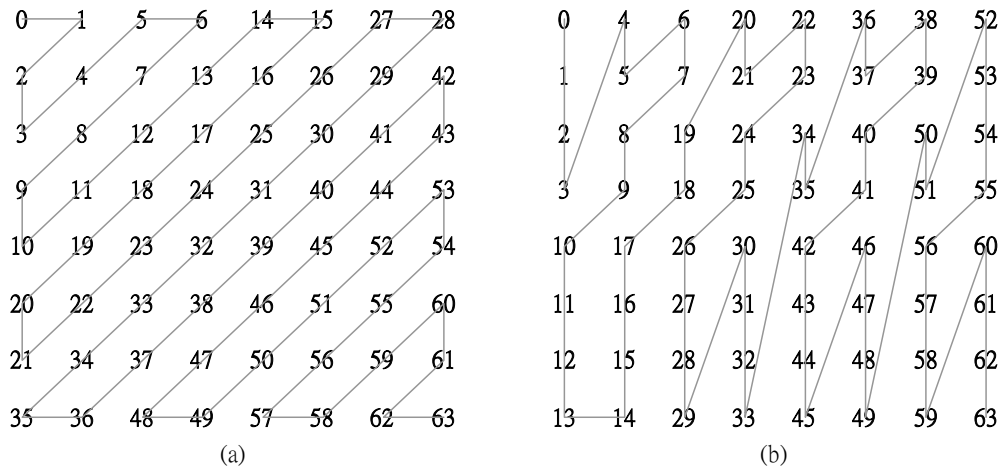


Fig. 2.1 Inverse scan pattern (a)*alternate_scan=0* (b)*alternate_scan=1*

2.1.3 Inverse quantization

As Fig. 2.2 shows, the inverse quantization process can be divided into 3 parts, the arithmetic, saturation, and mismatch control parts. In the arithmetic part, DC coefficient is separated from all the other coefficients. The parameter “*intra_dc_precision*” indicates the multiplication factor for DC coefficients, ranging from 1 to 8. For other coefficients, a weighting matrices $W[w][v][u]$ and the *Quant_scale_code* determines the multiplication factor. $W[w][v][u]$ can be either encoder-defined values or default values. The *Quant_scale_code* can be got by table lookup with the help of parameter “*quantiser_scale_code*” and “*q_scale_type*” in the bit-stream.

In the saturation part, the scaled coefficients $F''[v][u]$ are saturated to $F'[v][u]$ which lie in the range of $[-2048:+2048]$. In mismatch control, a correction is made to just one coefficient, $F[7][7]$, adding or subtracting by one.

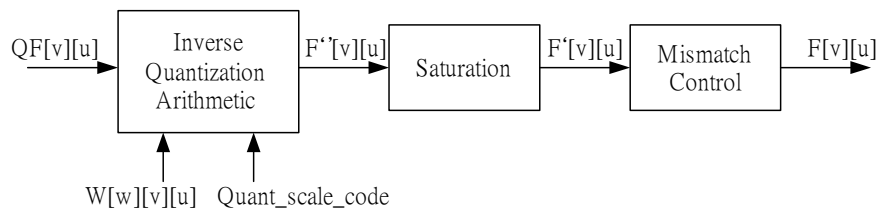


Fig. 2.2 Inverse quantization process

In summary the inverse quantization process is any process numerically equivalent to

```

// Arithmetic
for(v = 0; v < 8; v + 0+){
  for(u = 0; u < 8; u ++){
    if((u == 0) &&(v == 0) &&(macroblock_intra)){
      F''[v][u] = intra_dc_mult * QF[v][u];
    }else{
      if(macroblock_intra){
        F''[v][u] = (QqfF[v][u] * W[w][v][u] * quantisewr_scale * 2) / 32;
      }else{
        F''[v][u] = (((QF[v][u] * 2) + Sign(QF[v][u])) * W[w][v][u] * quantiser_scale) / 32;
      }
    }
  }
}

```

```

// Saturation
sum = 0;
for(v = 0; v < 8; v ++){
    for(u = 0; u < 8; u ++){
        if(F''[v][u] > 2047){
            F'[v][u] = 2047;
        }else{
            if(F''[v][u] < -2048){
                F'[v][u] = -2048;
            }else{
                F'[v][u] = F''[v][u];
            }
        }
        sum = sum + F'[v][u];
        F[v][u] = F'[v][u];
    }
}

// Mismatch Control
if((sum & 1) == 0){
    if((F[7][7] & 1) != 0){
        F[7][7] = F'[7][7] - 1;
    }else{
        F[7][7] = F'[7][7] + 1;
    }
}

```



2.1.4 Inverse Discrete-Cosine-Transform (IDCT)

The formula of Inverse Discrete-Cosine-Transform is as follows:

$$x(m, n) = \frac{2}{N} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} a(k)a(l)Y(k, l) \times \cos \frac{(2m+1)\pi k}{2N} \times \cos \frac{(2n+1)\pi l}{2N}$$

where $a(0) = \sqrt{\frac{1}{2}}$ and $a(k) = 1$ for $k \neq 0$

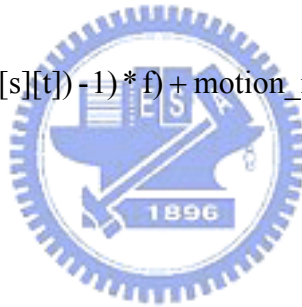
and $n, m, k, l = 0, \dots, N-1$

These transformed values shall be saturated to $[-256: +255]$.

2.1.5 Motion compensation

As Fig. 2.3 shows, the motion compensation process includes many parts. From the bit-stream, parameters like “*f_code*”, “*motion_code*”, and “*motion_residual*” can be extracted. With these parameters from bit-stream, a vector decoding module with the vector predictors (PMV[r][s][t]) decodes the motion vector vector'[r][s][t] by the following process.

```
r_size = f_code[s][t]-1
f = 1 << r_size
high = (16 * f) - 1;
low = ((-16) * f);
range = (32 * f);
if((f == 1) || (motion_code[r][s][t] == 0))
    delta = motion_code[r][s][t];
else{
    delta = ((Abs(motion_code[r][s][t]) - 1) * f) + motion_residual[r][s][t] + 1;
    if(motion_code[r][s][t] < 0)
        delta = -delta;
}
prediction = PMV[r][s][t];
vector'[r][s][t] = prediction + delta;
if(vector'[r][s][t] < low)
    vector'[r][s][t] = vector'[r][s][t] + range;
if(vector'[r][s][t] > high)
    vector'[r][s][t] = vector'[r][s][t] - range;
PMV[r][s][t] = vector'[r][s][t];
```



By scaling with a certain scaling factors, vector[r][s][t] are then sending to the address generator of frame buffers. The pixel values read from the frame buffers are then feed through a half-pel prediction filter. At the final stage, the decoded pixels is calculated by adding the combined predictions $p[y][x]$ with the residual values $f[y][x]$. Saturation process is needed to clamp the result.

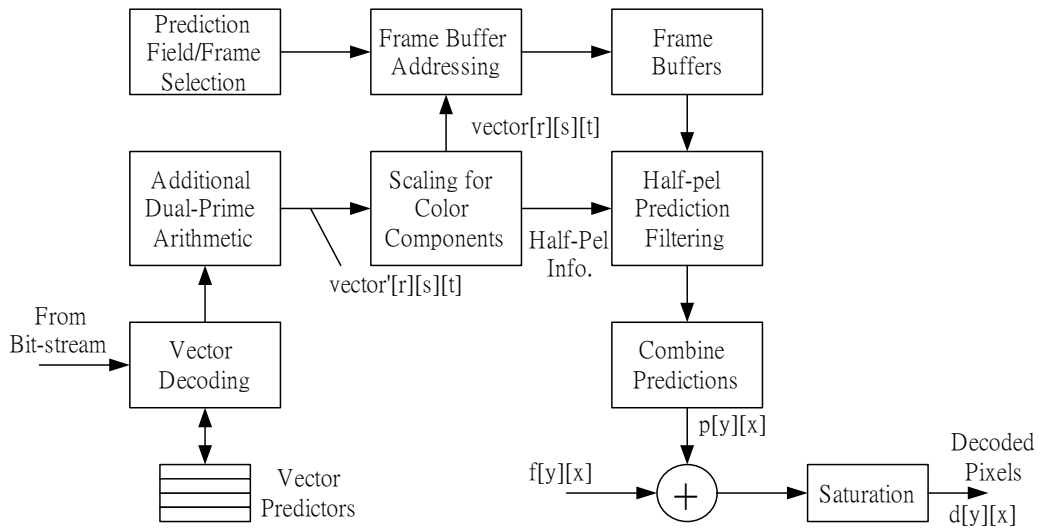


Fig. 2.3 Motion compensation process

2.2 Overview of H.264/AVC Decoding Flow

The H.264/AVC decoding flow is strictly specified in the standard such that all decoders shall produce numerically identical results. As Fig. 1.6 shows, the decoding process contains entropy decoding, reordering, inverse quantization, inverse integer discrete cosine transform, intra prediction, motion compensation, and loopfilter. In this thesis we consider the decoding process of baseline profile.

2.2.1 Entropy decoding

The H.264 bit-stream mainly contains 2 types of contents, the parameters and the coefficients. The parameters are mainly coded by Exp-Golomb code, which is a Universal Variable Length Code (UVLC). And the coefficients of residuals are coded by Context-based Adaptive Variable Length Code (CAVLC).

The entropy decoding process for Exp-Golomb code is as follows

```

leadingZeroBits = -1;
for(b = 0; !b; leadingZeroBits++)
    b = read_bits(1);

```

$$\text{code value} = 2^{\text{leadingZeroBits}} - 1 + \text{read_bits}(\text{leadingZeroBits});$$

The entropy decoding process for CAVLC is much more complicated than the decoding process of Exp-Golomb code. Many different tables are used for decoding the parameters like “*TrailingOnes*”, “*TotalCoeff*”, “*level_prefix*”, “*total_zeros*”, and “*run_before*”. For some coefficients like “*TrailingOnes*” and “*TotalCoeff*”, more than one table are used for decoding. And the so-called “Context-based Adaptive” VLC is because that the CAVLC decoder has to choose the correct table to decode a certain parameter according to the number of coefficients in neighboring block (left and upper 4x4 blocks). By decoding the intermediate parameters like “*trailing_ones_sign_flag*”, “*level_prefix*”, “*level_suffix*”, “*total_zeros*”, and “*run_before*”, the run and level of this run-level code can be calculated by the procedure defined in the standard. Then a run-level code decoder is used to recover the 16 coefficients in that 4x4 block.

2.2.2 Inverse scanning process

Input to this functional block is a list of 16 coefficients decoded by CAVLC. These 16 coefficients are then inverse scanned with a Zig-Zag scan pattern to form a 4x4 block. Fig. 2.4 shows the Zig-Zag scanning pattern.

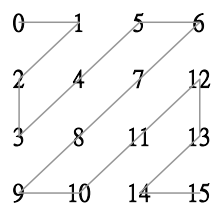


Fig. 2.4 Zig-Zag scan

2.2.3 Inverse quantization & inverse Hadamard transform

In the inverse quantization process, the operations on DC values are separated from other coefficients. Because 4x4 block is the basic unit in H.264 systems, there are total 16 luma DC coefficients in a macroblock. These 16 DC coefficients in a macroblock are first transformed through an inverse Hadamard transform matrix as the follows

$$f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

The result of inverse Hadamard transformation is then scaled by the following formula with the given QP_Y .

if QP_Y is greater than or equal to 12, the scaled result shall be derived as
 $dcY_{ij} = (f_{ij} * LevelScale(QP_Y \% 6, 0, 0)) \ll (QP_Y / 6 - 2)$
 Otherwise, the scaled result shall be derived as
 $dcY_{ij} = (f_{ij} * LevelScale(QP_Y \% 6, 0, 0) + 2^{1-QP_Y/6}) \ll (QP_Y / 6 - 2)$

Where

$$LevelScale(m, i, j) = \begin{cases} v_{m0} & \text{for } (i, j) \in \{(0,0), (0,2), (2,0), (2,2)\} \\ v_{m1} & \text{for } (i, j) \in \{(1,1), (1,3), (3,1), (3,3)\} \\ v_{m2} & \text{otherwise} \end{cases}, \quad v = \begin{bmatrix} 10 & 16 & 13 \\ 11 & 18 & 14 \\ 13 & 20 & 16 \\ 14 & 23 & 18 \\ 16 & 25 & 20 \\ 18 & 29 & 23 \end{bmatrix}$$

For the Cb and Cr in a macroblock, the 4 DC coefficients are first transformed through a 2x2 inverse transform matrix as the follows

$$f = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

After inverse transform, scaling is performed as follows

if QP_V is greater than or equal to 12, the scaled result shall be derived as

$$dcC_{ij} = (f_{ij} * LevelScale(QP_C \% 6, 0, 0)) \ll (QP_C / 6 - 1)$$

Otherwise, the scaled result shall be derived as

$$dcC_{ij} = (f_{ij} * LevelScale(QP_C \% 6, 0, 0)) \gg 1$$

For coefficients other than DC, the scaling function is

$$d_{ij} = (c_{ij} * LevelScale(qP \% 6, i, j)) \ll (qP / 6)$$

With the given qP.

2.2.4 Inverse Integer Discrete Cosine Transform

The Inverse Discrete Cosine Transform in H.264 system is much more simplified than the traditional Inverse Discrete Cosine Transform. The transform coefficients of 2-D IDCT in H.264 system are all simplified to integers. The transform matrix is as the follows.

$$c^p = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & 2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} p_{00} & p_{01} & p_{02} & p_{03} \\ p_{10} & p_{11} & p_{12} & p_{13} \\ p_{20} & p_{21} & p_{22} & p_{23} \\ p_{30} & p_{31} & p_{32} & p_{33} \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & 1 \end{bmatrix}$$

2.2.5 Intra prediction

The intra prediction process is a new prediction process that MPEG-2 system lacks. There are 2 classes of intra prediction modes, the Intra_4x4 prediction mode and Intra_16x16 prediction mode.

There are total 9 sub-modes in Intra_4x4 prediction mode. As Fig. 2.5 shows, these 9 modes are vertical, horizontal, DC, diagonal down-left, diagonal down-right, vertical-right, horizontal-down, vertical-left, and horizontal-up, respectively. In DC modes, the intra prediction process is to calculate the mean value of neighboring pixel values. Except for DC mode, all the others are directional modes. For directional modes, the intra prediction process for the prediction values can all be written as the following formula

$$\text{prediction value} = \frac{(P_0 + P_1 + P_2 + P_3) + 2}{4}$$

Where $P_0, P_1, P_2,$ and P_3 are all neighboring pixel values

The $P_0, P_1, P_2,$ and P_3 are different neighboring pixel values according to the type of mode and the position in the 4x4 block. For example, in mode 3 (diagonal down-left), the upper-left corner is predicted by $((A+2B+C)+2)/4$, which is equivalent to $((A+B+B+C)+2)/4$; and for upper-right corner of mode 5 (vertical-right), the prediction values is calculated by $((2C+2D)+2)/4$, which is equivalent to $((C+C+D+D)+2)/4$.

Note that the intra prediction process and the residual adding process are processes that must be perform iteratively. That is, for a given 4x4 block, the neighboring pixel values (upper and left) for intra prediction must be residual values added.

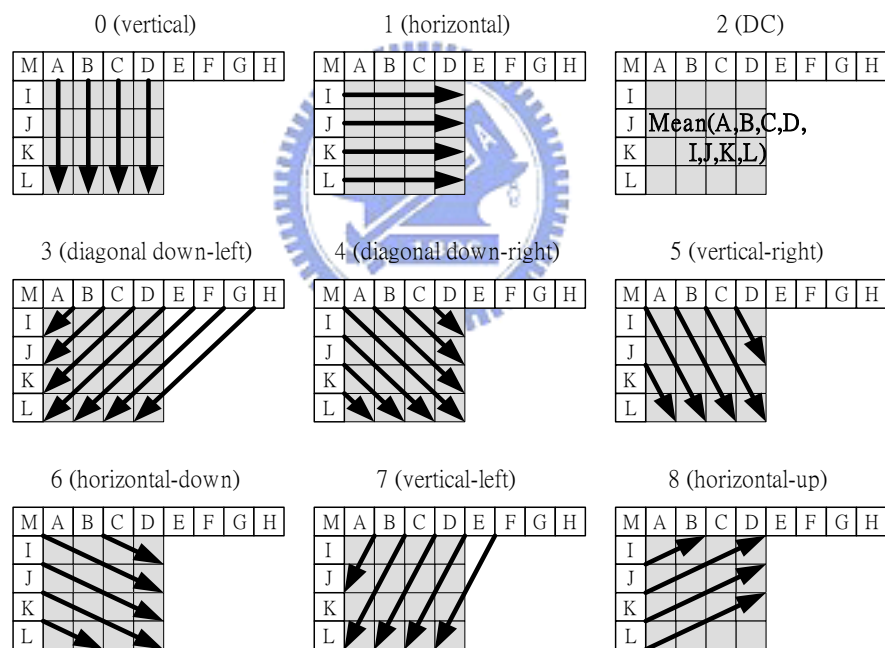


Fig. 2.5 Intra_4x4 prediction modes

In the intra_16x16 prediction mode class, there are total 4 modes – vertical, horizontal, DC, and plane modes respectively. The vertical mode and horizontal modes are easiest ones; the prediction is down by copying upper or left pixel values directly. In DC mode the mean value of all the upper and left neighboring pixel values has to be calculated and the result is

assigned to all the pixels in this macroblock. The plane prediction mode is the most complex one. The formula for luma samples is given as the follows

$$pred_L[x, y] = Clip1((a + b * (x - 7) + c * (y - 7) + 16) \gg 5)$$

Where

$$a = 16 * (p[-1,15] + p[15,-1])$$

$$b = (5 * H + 32) \gg 6$$

$$c = (5 * V + 32) \gg 6$$

$$H = \sum_{x'=0}^7 (x'+1) * (p[8+x',-1] - p[6-x',-1])$$

$$V = \sum_{y'=0}^7 (y'+1) * (p[-1,8+y'] - p[-1,6-y'])$$

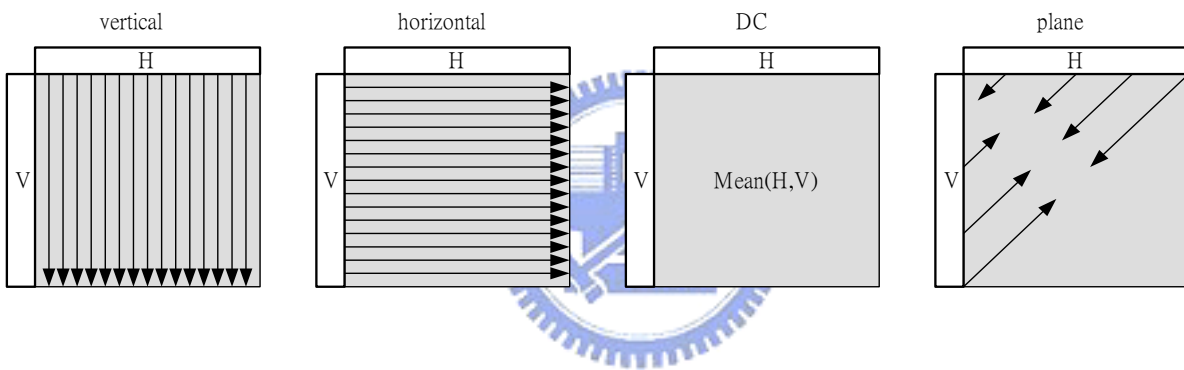


Fig. 2.6 Intra_16x16 prediction modes

For luma samples in a macroblock, both intra_4x4 prediction modes and intra_16x16 prediction modes are valid. But for chroma samples, only the 4 modes in intra_16x16 prediction class are valid and are a little different in parameters from formula for luma samples.

2.2.6 Motion compensation

In motion compensation process, each macroblock can be split into 4 types of partitions, 16x16, 8x16, 16x8, and 8x8. If the macroblock is split into 8x8 partitions, each

8x8 partition (Sub-Macroblock) can be further split into 4 types of partitions, 8x8, 4x8, 8x4, and 4x4. This hierarchical macroblock partition gives flexibilities on motion compensation process.

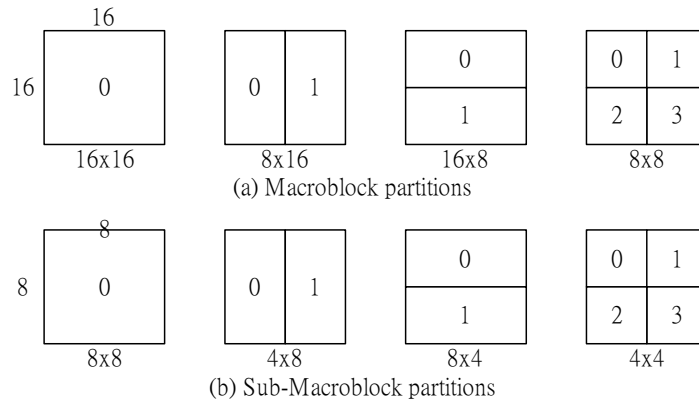


Fig. 2.7 Macroblock and Sub-Macroblock partitions

The precision of motion vectors is up to $1/4$. Fig. 2.8 shows an example of motion vector equals to $(+1.50, -0.75)$.

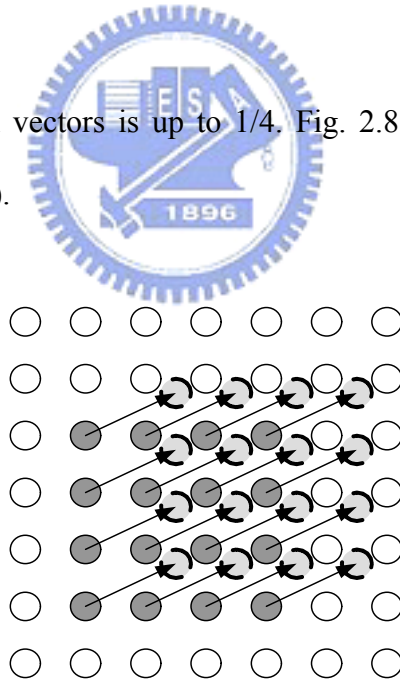


Fig. 2.8 Up to $1/4$ motion vector resolution ($mv=(+1.50, -0.75)$)

The motion compensation process requires interpolation process for inter-pixel values. As Fig. 2.9 shows, for interpolating pixels with the precision of motion vector up to $1/2$, a 6-tap interpolator is used for the interpolation. For example, pixel “b” is calculated by

$$b = \text{round}((E - 5F + 20G + 20H - 5I + J)/32)$$

For interpolating pixels with the precision of motion vector up to 1/4, a 2-tap interpolator is used for the interpolation. For example, pixel “n” is calculated by

$$n = \text{round}((c + f)/2)$$

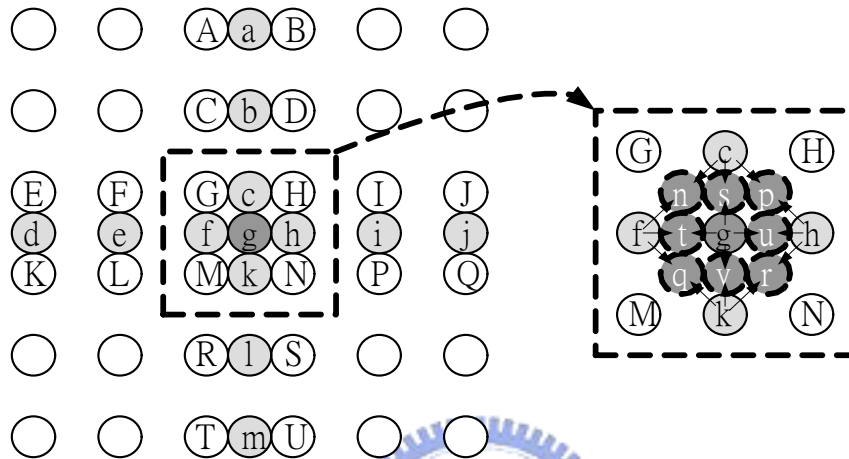


Fig. 2.9 Interpolation for pixel values

The motion vector MV is calculated by adding the MVD (motion vector difference) with the MVP (motion vector prediction). The MVD is decoded from the bit-stream. MVP is calculated from the motion vectors of neighboring blocks.

2.2.7 De-blocking filter

Same as MPEG-2, H.264/AVC system is block-based video coding system. Though we can perform discrete cosine transform to take advantage of the spatial correlation property and exploit motion compensated prediction to improve the compression ratio on the block-based systems, the disadvantage of the block-based system lies on the discontinuity on each block boundaries which is also known as blocking effects because of the quantization loss that annoying the continuity on block boundaries. Moreover, the blocking-effect propagated from frame to frame due to the motion compensation. Thus, a de-blocking filter is demanded and is included in the H.264 standard as an in-loop filter.

As Fig. 2.10 shows, the edge filtering order defined in the standard is a, b, c, d, e, f, g, then h. For a given 4x4 blocks, as long as the filter ordering to this 4x4 block is left, right, upper, and down, is standard compliant.

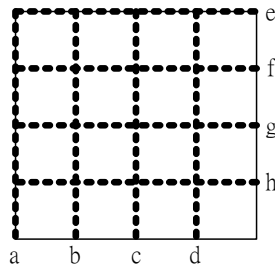


Fig. 2.10 Edge filtering order in a macroblock

The filtering process to a certain boundary is through an interpolator. Each filtering operation can at most changes 3 pixel values either in both sides of the boundary. The choice of filtering outcome depends on the boundary strength and on the gradient of image samples across the boundary. The boundary strength bS is in the range of 0 to 4, from no filtering to strongest filtering according to the quantiser, coding modes of neighboring blocks and the gradient of image samples.

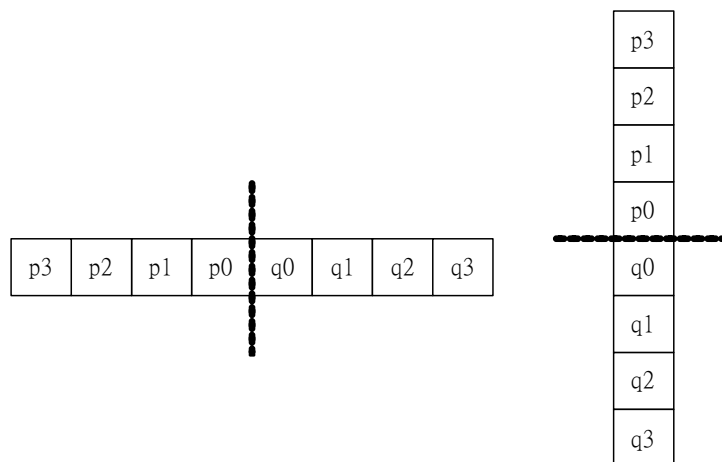


Fig. 2.11 Adjacent pixels to horizontal and vertical boundaries



Chapter 3

System Design Of MPEG-2 and H.264/AVC Decoder

In this chapter, we show some design techniques like pipeline scheme, synchronization problem and solution, decoding ordering, and power saving techniques from the system point of view.

3.1 MPEG-2 and H.264/AVC Combined System Decoding

Flow

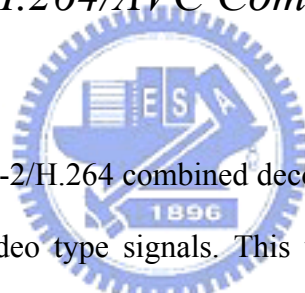


Fig. 3.1 shows our MPEG-2/H.264 combined decoder diagram. Input to this decoder is the video bit-stream and a video type signals. This video type signal acknowledges the decoder the type of the video bit-stream is feeding.

For H.264 video bit-stream, an H.264 syntax parser is firstly syntax analyzed the bit-stream, stored the system parameter into system-wide shared registers, send the bit-stream to the following residual path (CAVLC, 4x4-scaling, 4x4 IDCT) or prediction path (H.264 Intra predictor, H.264 Motion Compensator), summed them together with the help of synchronizer, a loopfilter process the summed pixel value and then output it both to frame buffer or to the display.

For MPEG-2 video bit-stream, same as the H.264 decoding flow, first, a MPEG-2 syntax parser analyzed the bit-stream, stored the system parameter into system-wide shared registers, send the bit-stream to the following MPEG-2 VLC decoder, 8x8 inverse quantizer, 8x8 IDCT, MPEG-2 Motion Compensator, and an optional MPEG-2 post filter is at the end

of the decoding flow.

For the hardware sharing issues, we share the registers in syntax parsers, design a CAVLC/VLC combined decoder for entropy decoding, a H.264/MPEG-2 combined motion compensator, a synchronizer for both system, content memory and frame buffer for both systems, and the de-blocking filter for both system which functions as an in-loop filter for H.264 system and a post-processing filter for MPEG-2 system.

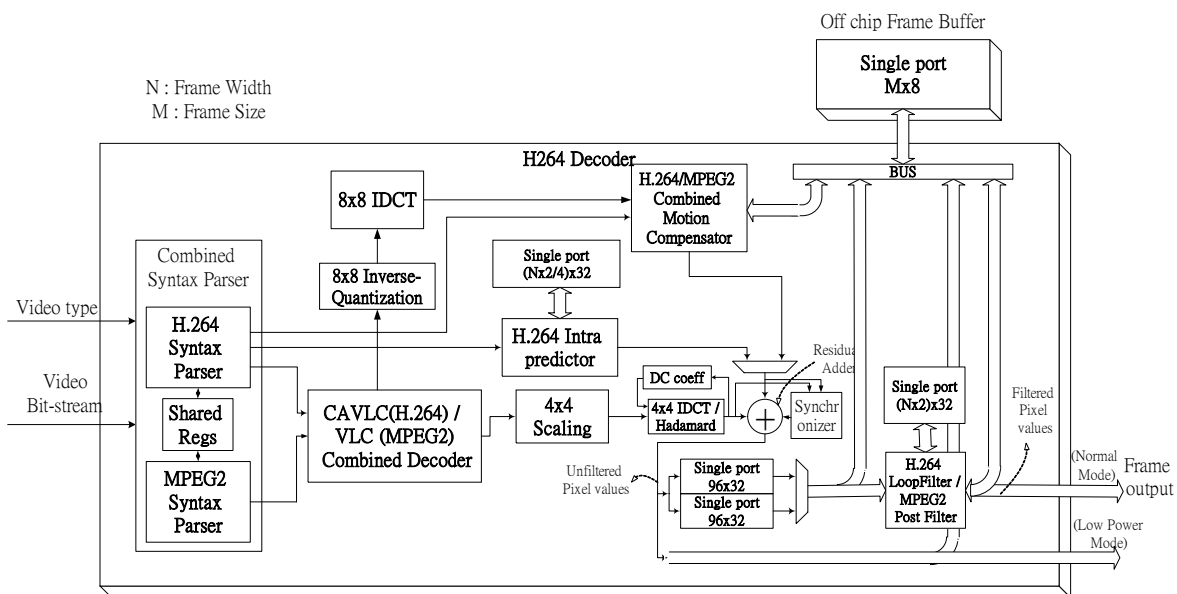


Fig. 3.1 MPEG-2/H.264 Combined Decoder Diagram

3.2 Hybrid 4x4-Block Level Pipeline with Instantaneous Switching Scheme for H.264/AVC Decoder

3.2.1 Hybrid 4x4-Block Level Pipeline Architecture

The 4x4 block is the smallest group of pixels that the H.264/AVC standard adopts. We can see from the standard that a 4x4 Inverse-Discrete-Cosine-Transform (IDCT), a 4x4-block based inverse scanning process, and a 4x4 inverse quantization matrix for

rescaling, are required in decoding H.264/AVC video sequence. Moreover, the smallest intra prediction unit is 4x4 sized block, and so do the motion compensation process. Thus in our H.264/AVC decoder design, compared with conventional macroblock-level pipelining architecture [1] [6], our 4x4-block level pipelining architecture are more suitable for the 4x4-block based H.264/AVC system.

Compared with macroblock-level (16x16) and block-level (8x8) pipeline parallelism, a trade-off exists between processing cycles and buffer cost. For the processing cycles issue, refers to Fig.3.2, we can see that the 4x4-sub-block-level pipeline parallelism requires more additional processing cycle than cycles needed of macroblock-level pipeline parallelism. Although this penalty has to be paid by the 4x4-sub-block-level pipeline parallelism, the cost saved of the buffer storage required is worthy.

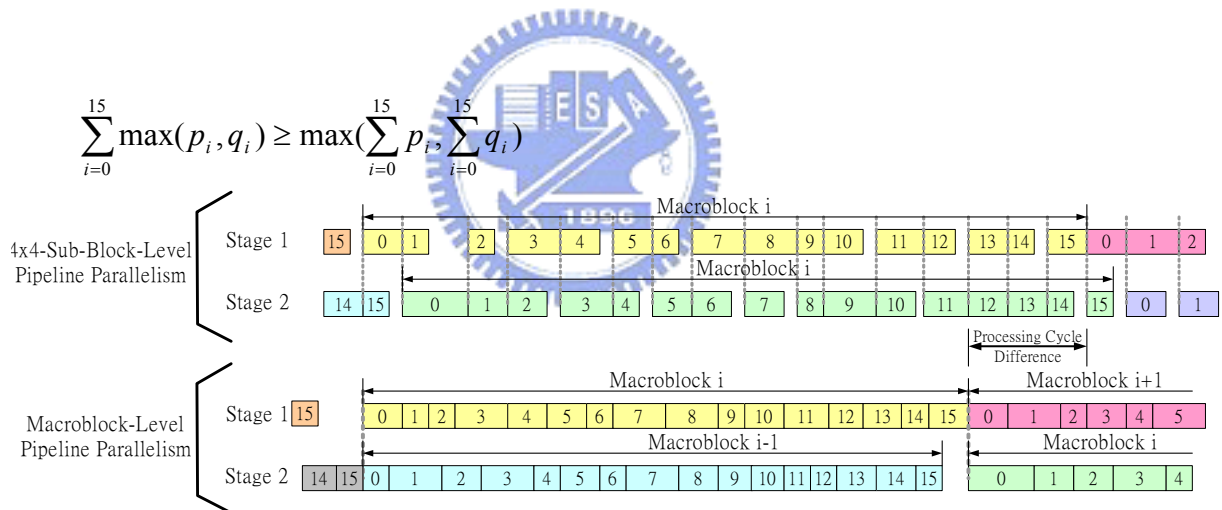


Fig.3.2 Additional processing cycles required for 4x4-sub-block-level pipeline parallelism

Compared with macroblock-level (16x16) and block-level (8x8) pipeline parallelism, because the processing unit of data in each stage is quite smaller (4x4) in 4x4-sub-block-level pipeline parallelism that the only 4x4-sub-block-level sized buffer storage is enough. We can see from Table 3.1, three different parallelisms show the trade-off between buffer cost and processing cycles. For 4x4-sub-block-level pipeline parallelism,

although 1.26 times processing cycles required compared with macroblock-level pipeline parallelism, 15/16 buffer storage can be saved.

Table3.1: Trade-off between processing cycles and buffer cost

Parallelism	Unit of Data	Buffer Cost	Processing Cycles
Macroblock-Level	16x16	X16	M cycles/MB
Block-Level	8x8	X4	1.19*M cycles/MB
Sub-Block-Level	4x4	X1	1.26*M cycles/MB

Moreover, besides the saving in storage cost, the large amount of power induced by these buffers which are active all the time could be greatly reduced as well. As Table 3.3 shows, the 4x4-sub-block-level sized storage buffers in CAVLC & IDCT consume 1.453mW and 0.864mW under clock frequency 100MHz, which contribute 2.86% of total power (81.072mW) when summed together. But if the macroblock-level sized buffers are used instead, the power of these storage buffers would be 23.251mW and 13.824mW, which is 15 times greater than the case of 4x4-sub-block-level pipeline parallelism.

Table 3.3 Power dissipated by buffers between pipeline stages

Parallelism	Storage buffer in CAVLC		Storage buffer in IDCT	
	Num. of regs	Power	Num. of regs	Power
Macroblock-Level	16x16x8 (bits)	23.251 mW	16x16x18 (bits)	13.824 mW
Block-Level	8x8x8 (bits)	5.813 mW	8x8x18 (bits)	3.456 mW
Sub-Block-Level	4x4x8 (bits)	1.453 mW	4x4x18 (bits)	0.864 mW

Although we save the cost of storage buffer and the associated power reduction by adopting 4x4-sub-block-level pipeline parallelism, this 4x4-sub-block-level pipeline parallelism can't be applied on some other modules which also exist in the decoding flow like motion compensator and loopfilter because of their macroblock-level-characteristic.

Motion compensator must support inter prediction process for several block sizes, from 4x4, 4x8, 8x4, 8x8, 16x8, 8x16, to 16x16. It is hard to divide the inter prediction process for block sized modes other than 4x4-block-sized mode into several 4x4-sub-block-sized inter prediction processes. So we choose to maintain traditional macroblock-level pipeline parallelism on motion compensation stage.

For in-loop filtering operation, i.e. loopfilter, it is also hard to be divided into several identical 4x4-sub-block filtering process because the neighboring 4x4-sub-blocks it has to fetch is irregular according to inverse scanning sequence. In contrary, the filtering process is almost identical in macroblock level. Thus we also choose macroblock-level pipeline parallelism for loopfilter.

In our overall pipeline design, we combine the 4x4-sub-block-level pipeline parallelism with macroblock-level pipeline parallelism to a hybrid pipeline scheme that suits best for each module. The pipeline parallelism applied for decoding modules is summarized in Table 3.4.

Table 3.4 Summary of pipeline parallelism applied

Module	Pipeline parallelism
Intra predictor	4x4-sub-block-level
CAVLC	4x4-sub-block-level
De-quantizer	4x4-sub-block-level
IDCT	4x4-sub-block-level
Motion compensator	Macroblock-level
Loopfilter	Macroblock-level

3.2.2 Instantaneous Switching Scheme

We also applied an instantaneous switching scheme in our 4x4-sub-block-level pipeline design, that is, we switch our pipeline stage as soon as possible. As long as all pipelined modules complete their work, we switch the pipeline into next stage instantaneously. Because of this instantaneous switching scheme we applied, any pipelined module with especially long processing cycles would be the bottleneck of the whole decoding system. The pipeline stage must be switched only if all the pipelined modules complete their work. So all the other pipelined modules must be idle and wait for the pipelined module with especially long processing cycles if exists, bubbles induced in this kind of situation would be a lot that degrades overall system throughput much. Thus, we try to balance the cycle count required for each modules, so that the idle time of these pipelined modules like CAVLC, De-quantization, IDCT, and etc could be minimized that this instantaneous switching scheme can be a great help of maximizing our system throughput.

Fig. 3.3 shows an example pipelining schedule of hybrid 4x4-sub-block-level pipeline parallelism with instantaneous switching scheme.

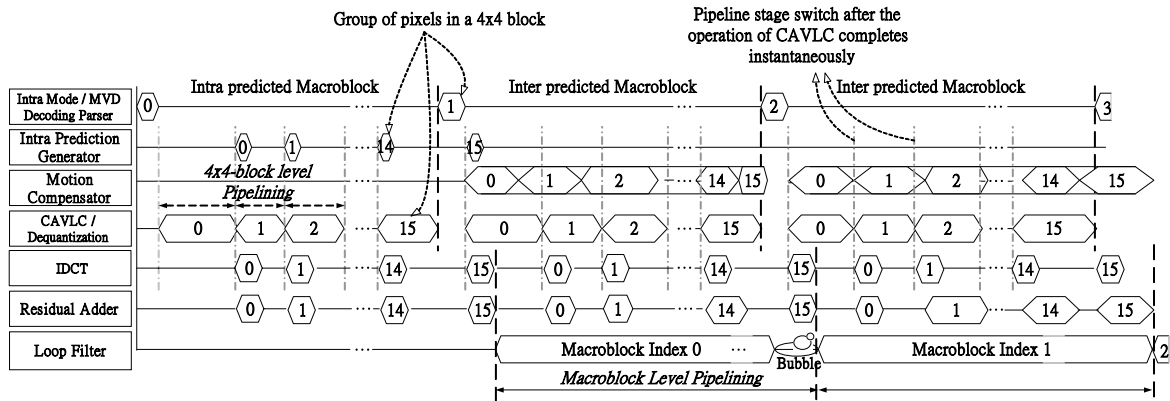


Fig. 3.3 An example of the pipelining schedule

3.3 Efficient 1x4 Column-By-Column Decoding Ordering

Based on our proposed 4x4-sub-block-level pipeline parallelism, we choose 4 pixels per cycle as our overall system throughput. The throughput of 4 pixels per cycle is also very suitable for the efficient IDCT design, inverse quantizer design, and inter/intra predictor design. Limited by the 4x4-sub-block inverse scanning sequence (also the decoding sequence) defined by H.264/AVC standard, we have two choices on the decoding ordering that are both standard compliant, the 4x1 row-by-row decoding ordering and the 1x4 column-by-column decoding ordering, as Fig. 3.4 and Fig. 3.5 shows respectively. After the analysis for inter and intra predictor on these 2 types of decoding order given in the following, we will see that the 1x4 column-by-column decoding ordering is better than 4x1 row-by-row decoding ordering both in fewer memory access times and fewer decoding cycles.

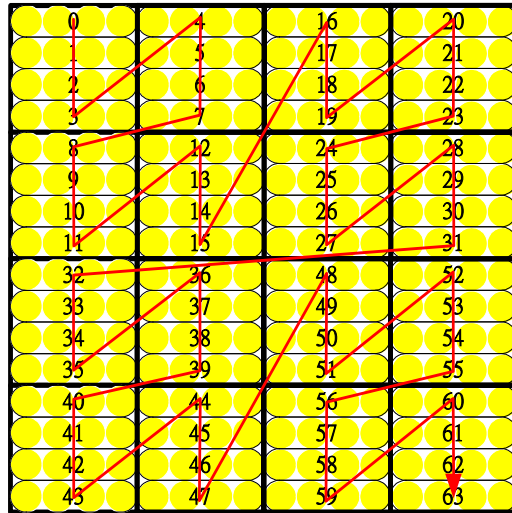


Fig. 3.4 4x1 row-by-row decoding ordering

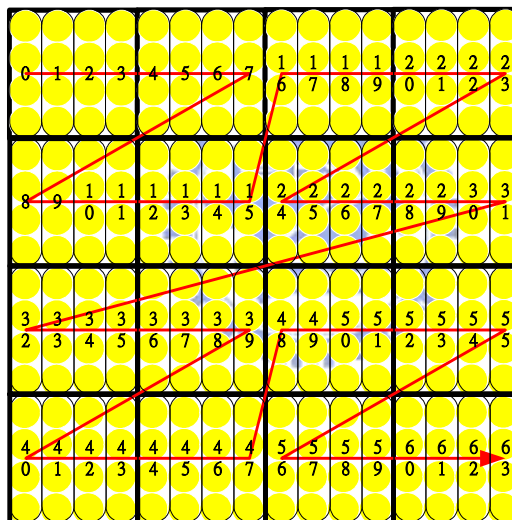


Fig. 3.5 1x4 column-by-column decoding ordering

Now we give an analysis for both inter and intra prediction units on these 2 decoding ordering.

3.3.1 Analysis on inter prediction unit

In our inter predictor design also known as motion compensator, an initialization stage is required before any contiguous output of motion compensated pixel values. The

initialization period requires 18 memory access times for loading related neighboring $9 \times 6 = 54$ pixel values from the reference frame for the 2-D interpolation (6-tap interpolation then 2-tap interpolation) of the target pixel values. 18 cycles (9 pixels per 3 cycles) also required for this operation in the initialization stage. After the initialization stage is finished for the 1st group of 4 motion compensated pixel values, the loaded pixel values in the initialization stage can be reused and only 9 new pixels are needed to be loaded for computing the following contiguous output. This computing process requires only 3 memory access times and 3 cycles for the following contiguous outputs of a group of 4 motion compensated pixel values.

For decoding an inter predicted macroblock under 4x1 row-by-row decoding ordering and 1x4 column-by-column decoding ordering, we can found that as Fig. 3.6 shows, for the 4x1 row-by-row decoding ordering, there exists 16 discontinuities (3rd, 7th, 11th, 15th, 19th, 23rd, 27th, 31st, 35th, 39th, 43rd, 47th, 51st, 55th, 59th and 63rd outputs) in decoding ordering. Each discontinuity output of a group of 4 pixel values requires an initialization process. 3 contiguous outputs are then followed by each discontinuous output. Thus for 4x1 row-by-row decoding ordering, total memory access times and total decoding cycles are

$$16 \times 18 \text{ (discontinuous output)} + 16 \times 3 \times 3 \text{ (contiguous output)} = 432 \text{ memory access} \quad (3.1)$$

$$16 \times 18 \text{ (discontinuous output)} + 16 \times 3 \times 3 \text{ (contiguous output)} = 432 \text{ cycles} \quad (3.2)$$

As Fig. 3.7 shows, for 1x4-column-by-column decoding ordering, only 8 discontinuities (7th, 15th, 23rd, 31st, 39th, 47th, 55th and 63rd outputs) exist in decoding ordering, which leads to 8 initialization process for these 8 outputs. 7 contiguous outputs are then followed by each discontinuous output. Thus for 1x4-column-by-column decoding ordering, total memory access times and total decoding cycles are

$$8 \times 18 \text{ (discontinuous output)} + 8 \times 7 \times 3 \text{ (contiguous output)} = 312 \text{ memory access} \quad (3.3)$$

$$8 \times 18 \text{ (discontinuous output)} + 8 \times 7 \times 3 \text{ (contiguous output)} = 312 \text{ cycles} \quad (3.4)$$

In summary, for an inter predicted macroblock, the memory access times and decoding cycles saved by adopting 1x4-column-by-column decoding ordering instead of 4x1-row-by-row decoding ordering are both 28%.

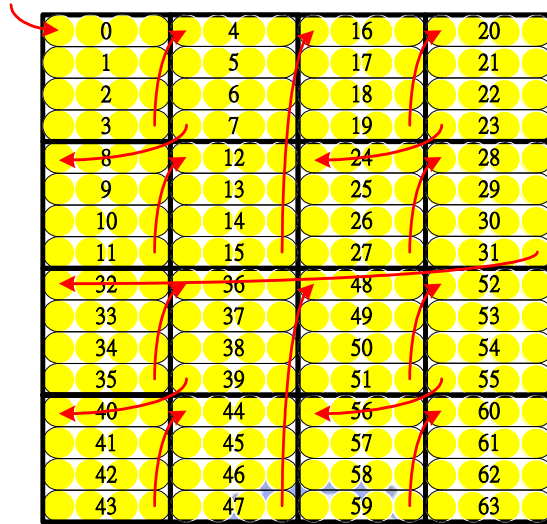


Fig. 3.6 16 initialization processes in inter predicted macroblock under 4x1 row-by-row decoding ordering

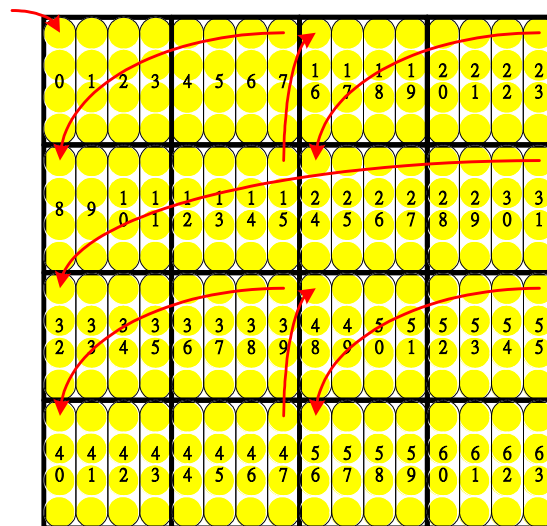


Fig. 3.7 8 content switches in inter predicted macroblock under 1x4 column-by-column decoding ordering

3.3.2 Analysis on intra prediction unit

Based on the H.264/AVC standard, for an Intra4x4 predicted macroblock, the neighboring pixels including upper 8 pixels, left 4 pixels plus a corner pixel (total 13 pixels) must be loaded before the intra prediction process. Because we follow this rule in our intra predictor design, accessing from memory for these 13 pixels are required before each intra4x4 prediction process no matter which prediction mode is for this 4x4-sub-block. We found that if we choose the 1x4 column-by-column decoding ordering as Fig. 3.5 show, a group of 4 pixels of every 4th output is just the left 4 neighboring pixels that originally required to be fetched from neighbor for intra prediction on next 4x4-block. For example, as Fig. 3.8 shows, the group of 4 pixels in the 3rd output is just the left 4-neighboring pixels to be fetched for the following 4x4-sub-block. In this way, this group of 4 pixels can be forwarded directly from previous output instead of fetching from memory that reduces the memory access times. Same situation also occurs at the 11th, 19th, 27th, 35th, 43rd, 51st, 59th outputs too. However, for 4x1-row-by-row decoding ordering, this property can not be found to reduce the memory access times. In summary, the memory access times can be reduced from $3 \times 16 = 48$ times to $3 \times 8 + 2 \times 8 = 40$ times (17% saved) by adopting 1x4-column-by-column decoding ordering instead of 4x1-row-by-row decoding ordering.

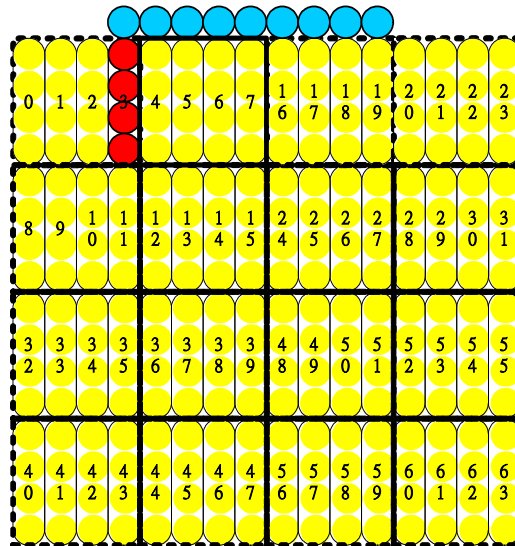


Fig. 3.8 Reduction on memory access of the intra predicted macroblock

3.4 Prediction/Residual Synchronization Scheme

In both H.264/AVC and MPEG2 decoder designs, there exist 2 decoding paths, say, inter/intra prediction path (prediction path) and residual recovery path (residual path). The prediction path predicted the pixel values from the motion vector by motion compensator or by intra prediction mode by intra predictor. The residual path decodes the residual pixel values first by entropy decoding the coded data by CAVLC/CABAC (H.264/AVC) or table based VLC (MPEG2). A de-quantization process is then performed on the decoded value. Finally, an inverse discrete-cosine-transform (IDCT) transfers the scaled values into residual values and output them at the end of the residual path. The decoder has to add the predicted pixel values from prediction path with the residual pixel values from residual path to reconstruct the original picture before an in-loop filter (H.264/AVC) or a post-filter (MPEG2).

The synchronization problem exists in this adder that adds the pixel values come from 2 different decoding paths. Because the output timing of these 2 paths is different, we can not guarantee the output timing of the pixel values come from 2 paths is simultaneous in a

certain cycle. And we can not expect which output comes earlier. Thus a synchronizer is required. As Fig. 3.9 shows, we developed a Variable-Length FIFO as a synchronizer for the synchronization of prediction path and residual path to solve this problem.

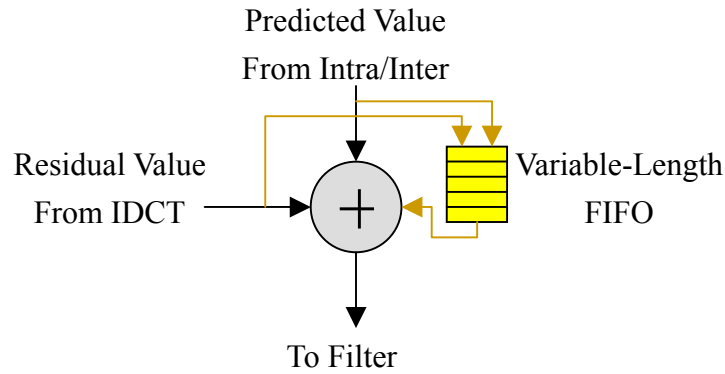


Fig. 3.9 A variable-length FIFO is required for the synchronization between intra/inter predictor and IDCT

The operation of this variable-length FIFO (VL-FIFO) solution is as Fig. 3.10 shows. The signals “sample_valid” and “IDCT_operation” indicates the output valid timing. Because these 2 signals are not identical, the VL-FIFO stores the output pixel values from either path as long as it comes alone without the company of the output from another path. The output of VL-FIFO which had been stored previously waits until the associated values come from another path. In this way, the residual adder that adds the values from prediction/residual path can correctly add them together at a certain cycle with the help of this VL-FIFO as the synchronizer.

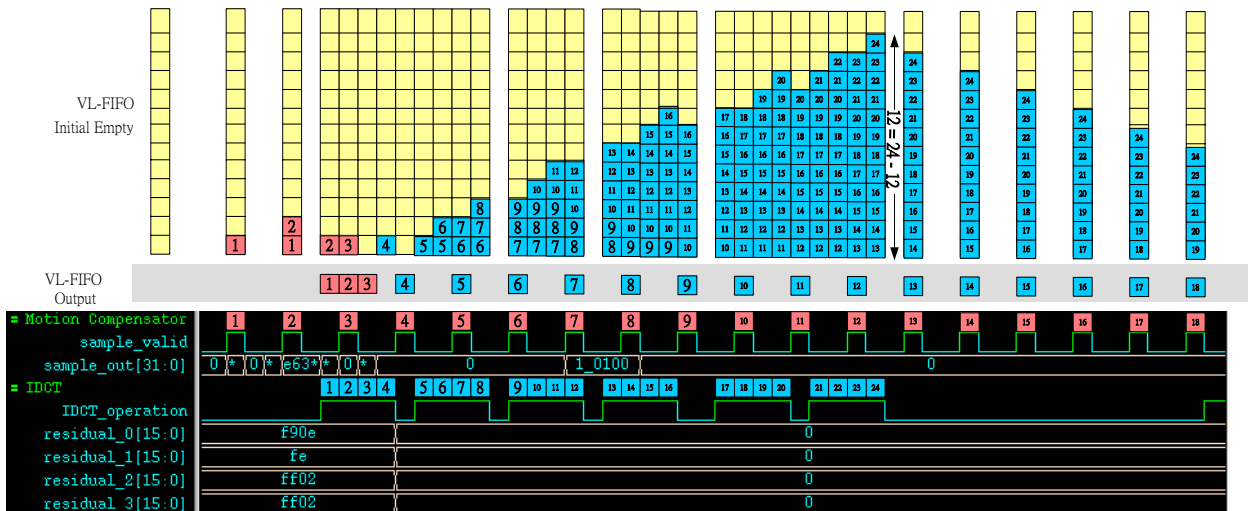


Fig. 3.10 Operation of variable-length FIFO as a synchronizer

3.5 Power saving by exploiting Coded-Block-Pattern

H.264/AVC and MPEG-2 both support videos in various data rate. High definition with high data rate targets at some high quality video applications like digital home entertainment devices, on the other hand, low definition but low data rate targets at applications like video transmission in hand held devices.

In this chapter, a power saving technique will be introduced for some low data-rate applications, especially for video sequence of high QP (Quantisation Parameter). The main idea of this power saving technique is that we can save power by shutting down the inverse quantization and inverse DCT operation at the blocks with all zero coefficients and passing these 2 modules directly because the output through these 2 modules are all zeros expectedly.

Residual coefficients which quantized to zeros are more as the QP is increased. Thus in high QP video sequence, we can find that there exists many blocks with all-zero-coefficients. And the parameter “Coded-Block-Pattern” notifies the decoder the incoming all-zero-coefficients blocks in advance. Thus by observing the decoded

“Coded-Block-Pattern”, we can foresee the blocks with all-zero-coefficients, shutting down the inverse quantizer and IDCT, and then passing the results of all zeros directly to the output of the IDCT. Fig. 3.11 shows the block diagram for this power saving technique.

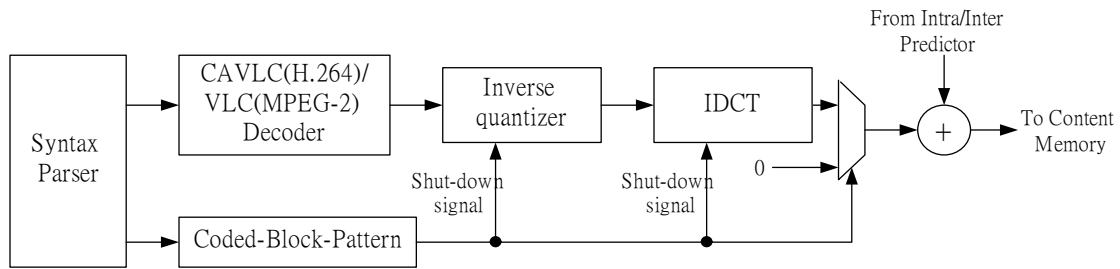


Fig. 3.11 Power saving by exploiting Coded-Block-Pattern

Fig. 3.12 shows the simulation results of QP versus bitrate (for QCIF foreman sequence) and QP versus the percentage of all zero coefficient blocks. We can see that as the QP increased, the bit rate decreased because of the quantization loss increased, which leads to the increasing of the percentage of all-zero-coefficient blocks. We can see that this power saving technique saves power dissipated from inverse quantization and IDCT from 30% to almost 100% at QP from 20 to 50. This savings in power is huge especially in high QP sequence.

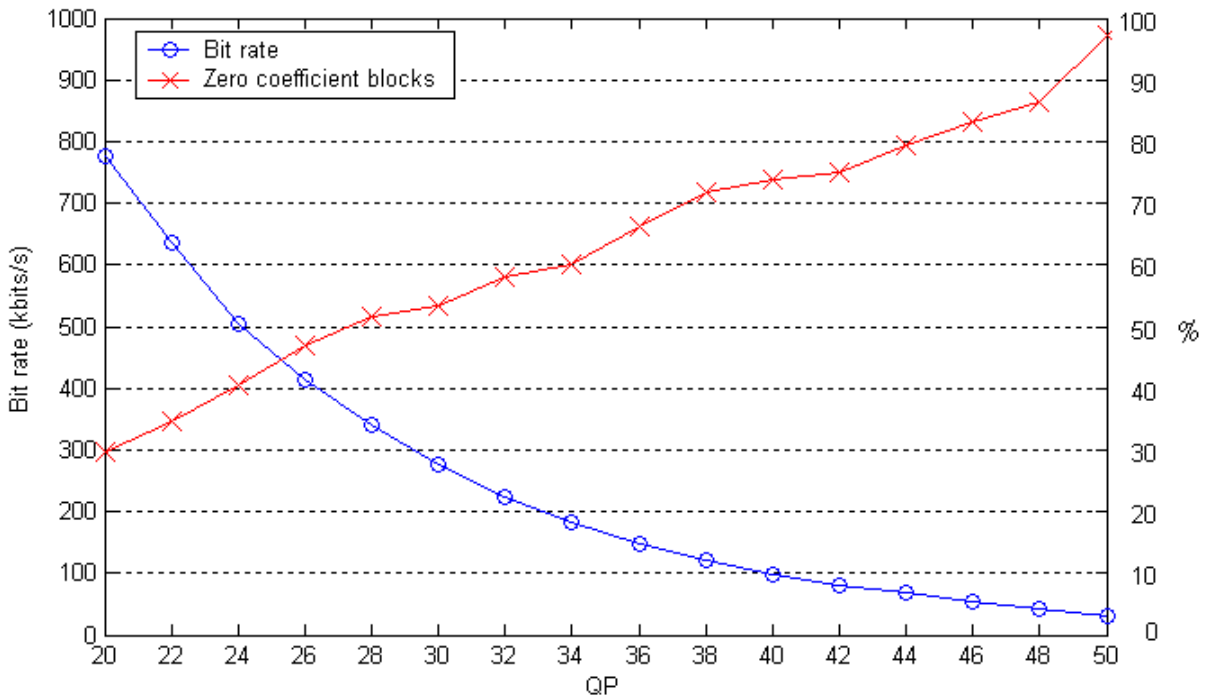


Fig. 3.12 QP versus Bitrate and the percentage of all zero coefficient blocks

3.6 *Novel User-Determinable Low Power Mode Exploration*



Because H.264 is getting more and more popular in future video applications, and is potential in future hand-held devices like PDA, mobile phone, and etc. Thus, to design a low-power decoder becomes an important issue. To reduce the power consumption, besides the low-power architecture design which will be introduced in the following chapter, a novel user-determinable low power mode is introduced here.

Table 3.5 shows the power profiling of our decoder, reported by PrimePower in decoding H.264 video sequence at 100MHz. From this report, we can see that the loopfilter consumes highest power among other modules both in decoding I-frame and P-frame. The power consumed by loopfilter mainly contributed by 4 single port SRAMs in it. Because the main purpose of this loopfilter is to smooth the decoded picture only, we might be able to shut down the loopfilter in order to save much power as long as the unsmoothed picture is

acceptable.

Imagine that one day you're watching a TV program through hand-held device on a train. You find that the battery almost ran out. At this time if power-saving solution with the acceptable performance degradation trade-off is provided, it would be a very nice choice to you.

Fortunately, the content memory which serves as to isolate the loopfilter from other decoding modules, is useless and can be shut-down too when we shut down the loopfilter. The power consumes by content memory can also be eliminated in this low-power mode. However, loopfilter operation is indispensable in the standard. Thus to make our decoder standard-compliant, we leave this performance-power trade-off choice to user by providing a user-determinable low-power mode, which leads to this novel work.



Table 3.5 Power profiling of decoding H.264 video bit-stream (unit:mW)

Module	I-frame	P-frame
LoopFilter	31.092	27.130
Content Memory	12.678	10.154
Motion compensator	10.414	17.152
VL-FIFO	9.784	10.866
Intra Predictor	7.090	5.496
CAVLC	2.604	2.754
Syntax Parser	1.926	1.280
IDCT	1.748	0.428
Inverse Quantizer	1.112	0.260
Total Core Power	81.072	78.112

Table 3.6 shows the simulation result on power consumption and savings before and after the low power mode is enabled. We can see that about 40% power can be saved under low-power mode both in decoding I-frame or P-frame.

Table 3.6 Low Power Mode Exploration

	Normal Mode	Low Power Mode	Power Saved
I-Frame	81.072 mW	49.582 mW	38.84 %
P-Frame	78.112 mW	46.504 mW	40.46 %

Chapter 4

Architecture Design Of MPEG-2/H.264/AVC Decoder

In this chapter the architecture and some low-power techniques used in each module of the MPEG-2/H.264/AVC decoder will be described in details. Syntax parsers, intra predictor, inverse quantizer, inverse DCT, motion compensator, prediction/residual synchronizer, in-loop/post filter are included.

4.1 MPEG-2 & H.264/AVC Combined Syntax Parsers

4.1.1 Low-Power Hierarchical Parser Design

The syntax structure is defined in the H.264 standard. Combined with the NAL header format defined in the Annex B in the standard, as Fig. 1.7 shows, the H.264 video bit-stream is organized hierarchically. Thus the syntax parser is designed in hierarchical structure in order to make a good use of this property.

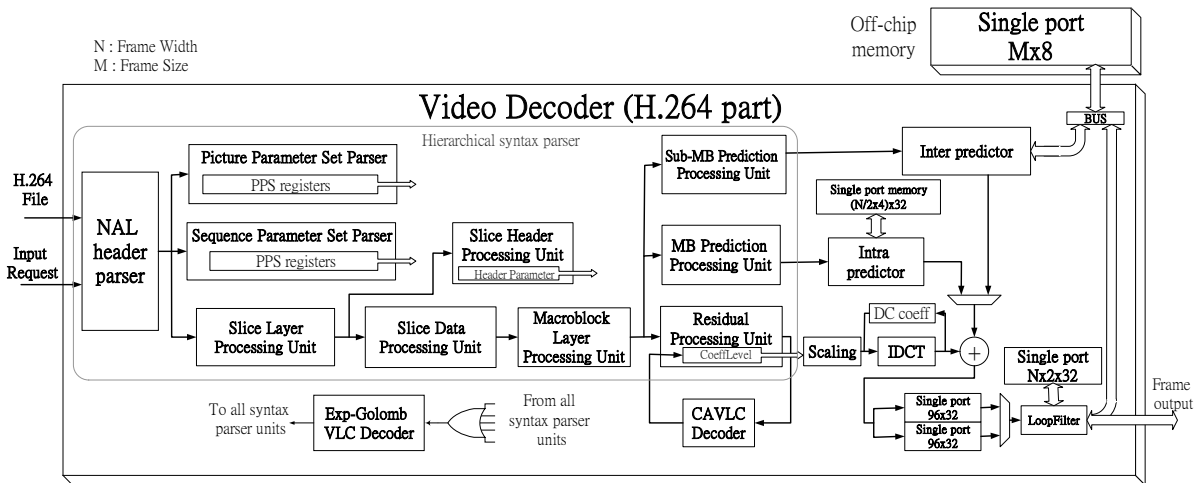


Fig. 4.1 Hierarchical syntax parser

The syntax parser is designed in hierarchical structure in order to match the hierarchical structure of the H.264 bit-stream that standard defines. As Fig. 4.1 shows, an NAL header parser, which detects the NAL syntax element and identifies the NAL unit type from the NAL unit header is at the first part of the decoder. Depending on the NAL unit type it detects, the NAL header parser sends the enable signal to the following 3 units, SPS processing unit, PPS processing units, and slice layer processing unit. Each of which is responsible for decoding the data that belongs to it. For the slice layer RBSP, instead of directly decoding data, slice layer processing unit sends the enable signal to either slice header processing unit or slice data processing unit by the aid of a simple acknowledge signal from slice header processing unit that always operates in the precedence of the slice data processing unit. Slice header processing unit, like SPS and PPS unit, are responsible for decoding the parameters which will be used by the other functional blocks in decoding times. We use global-wide registers to store these parameters in these units, so that many other modules can access easily. Hierarchical structure is also exploited on the slice data unit which acknowledges the macroblock processing unit, then the MB prediction unit,

sub-MB prediction unit, or the residual processing unit. The residual processing unit wakes up CAVLC decoder to decode the entropy encoded residual data when needed.

Many parameters are needed during decoding each macroblock. There are hundred kinds of parameters in the PPS, SPS, slice header, macroblock header, prediction mode, prediction weight table, etc. Thus these parameters are needed to be stored either in the memory or register files when parsing. And because these parameters are read frequently into the prediction unit, de-quantization unit, de-blocking filter and many other functional blocks. Using registers to store these parameters is better than memory approach. Because the number of parameters is a lot, it is important to save the power consumption on these registers.

To resolve the power issue on the huge amount of registers, we put these system-wide registers together inside their belonging syntax parser unit. In that way, we can reduce the power dissipation of these registers by disabling their belonging syntax parser units. Fig. 4.2 shows an example waveform of the enable signals of each syntax-parsing unit. With the aid of the enable signals of these syntax parser units to be their sleep controls, we can apply gated-clock technique on these syntax parser units easily and to be able to disable all the syntax parser at their idle time.

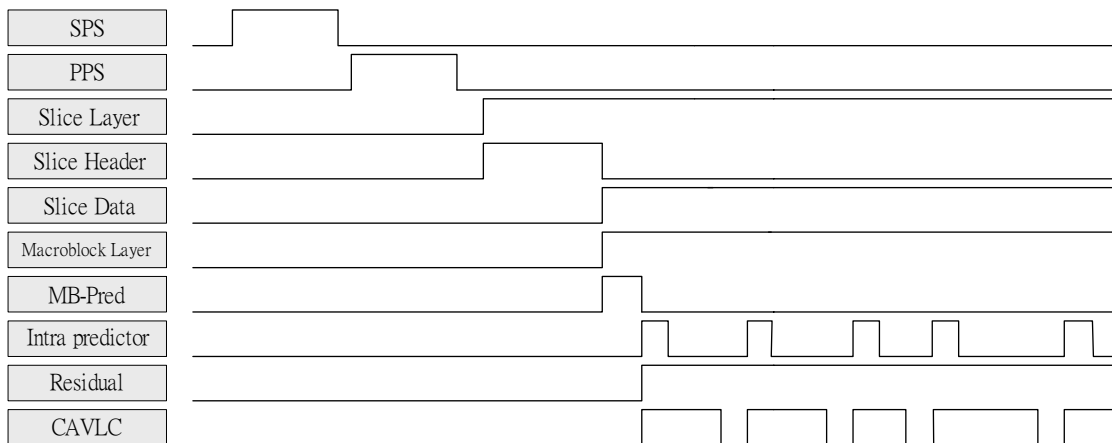


Fig. 4.2 An Example waveform of the enable signals in syntax parser

Simulation results show that applying the gated clock technique on our hierarchical syntax parser, we can greatly save 86% power consumption on these registers.

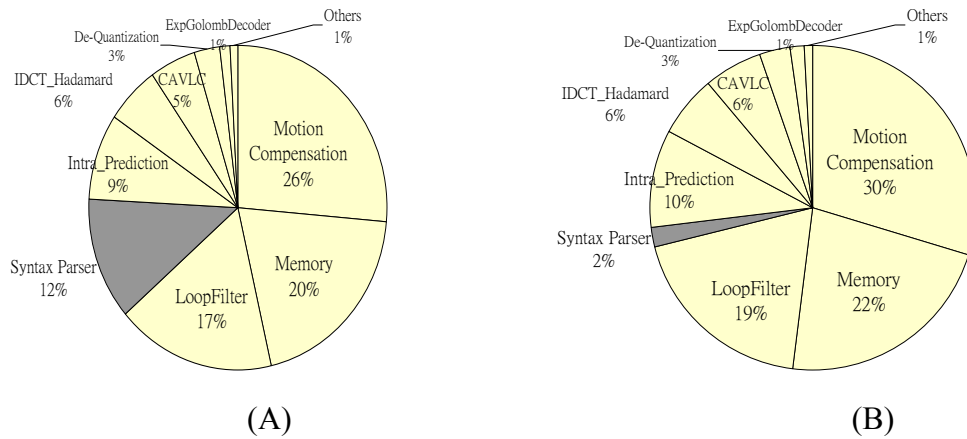


Fig. 4.3 Power reduction on syntax parser

(A) Without applying gated clock (B) With applying gated clock

4.1.2 Register Sharing Parser Design

Registers for parameter storage and control circuits are two main components in syntax parser design. And because of the various and large amount of system-wide parameters that syntax parser are responsible for the storing, registers in syntax parser almost dominate the area and also the cost of the syntax parser.

In our MPEG-2/H.264 dual decoder design, originally we have to design two syntax parsers which suits for H.264 syntax system and MPEG-2. For the high throughput issue, we custom-built these two syntax parser for the highest efficiency instead of traditional embedded software with the RISC solution. Because control circuits is complex but not area-expensing, except for the inflexibility, the main cost of our dual parser solution lies on the two set of registers for parameter storage of 2 different video coding system.

Fortunately, for the dual MPEG-2/H.264 decoder design, we can see that whether the

decoder is functioned for MPEG-2 system or H.264 system, there is always a set of syntax parser in idle situation. Which in terms that there is always a set of registers that would never be used. Thus we can combine these 2 set of registers into one set of registers which shared for these two video systems. That way, we might be able to reduce the number of registers used in syntax parser, and that leads to the reduction on area and power because registers consumes lots of power.

Table 4.1 Number of register needed for MPEG-2/H.264 syntax parser

MPEG-2		H.264	
Module	Register needed	Module	Register needed
Macroblock	29 regs	Macroblock	11 regs
Motion_vectors	25 regs	Slice_data	17 regs
Picture_coding_ext	49 regs	Slice_header	247 regs
Picture_header	36 regs	PictureParameterSet	74 regs
Slice	8 regs	SequenceParameterSet	127 regs
Sequence_header	1105 regs		
Total	1252 regs	Total	476 regs

As Table 4.1 shows, we can see that syntax parser in both these two system requires lots of register for parameter storage. And because MPEG-2 system is 8x8 block based system, 4 times bigger than the 4x4 block based H.264 system, the registers needed is also the more than the H.264 system. The sequence_header module in MPEG-2 syntax parser contains 2 set of 8x8 intra and non-intra coefficients of quantization matrix. 512 registers needed for each coefficient matrix, which dominates the total number of register needed in syntax parser.

Because these registers for quantization matrix coefficient storage are large and very regular, we make these registers for the sharing registers with H.264 syntax parser. The register sharing requires an additional multiplexer, as Fig. 4.4 shows. The select signal to this multiplexer is the MPEG-2/H.264 mode input of the decoder.

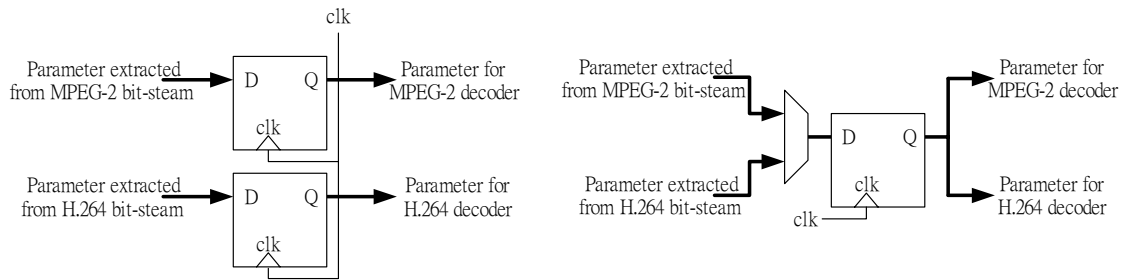


Fig. 4.4 Register sharing technique

In this work, these registers in MPEG-2 system are shared with the 3 main modules (Slice_header, PictureParameterSet, SequenceParameterSet) in H.264 system; total 448 registers are shared and can be saved as result.

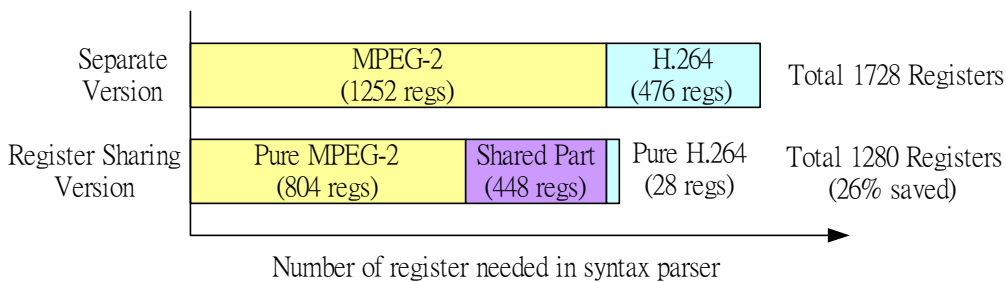


Fig. 4.5 Register number reduction on syntax parser

Because of the additional multiplexer required for the register sharing technique, the total number of combinational circuit increases slightly. But for the reduction on number of registers, the total area saved and the power saved is about 20.5% and 22.6%. Table 4.2 summarizes the result of register-sharing syntax parser compared with original register

separate version.

Table 4.2 Area and Power reduction on register-sharing version

	Separate Version	Shared Version	Percentage Saved
Combinational Area	32,307	39,993	-23.7%
Non-Combinational Area	126,066	85,984	31,8%
Total Area	158,373	125,977	20.5%
Power Dissipation	14.807 mW	11.463 mW	22.6%

4.2 *Exp-Golomb Decoder for H.264/AVC syntax parser*

Exp-Golomb code is an UVLC (Universal Variable Length Code) that is used extensively for coding of the parameter in syntax element of H.264 bit-stream.

According to the standard, the processing of the UVLC decoder shall be equivalent to the following:

```
leadingZeroBits = -1;
```

```
for(b=0;! b; leadingZeroBits++)
```

```
    b = read_bits(1);
```

```
    code value =  $2^{\text{leadingZeroBits}} - 1 + \text{read\_bits}(\text{leadingZeroBits})$ ;
```

Table 4.3 shows the assignment of bit strings to the code value.

Table 4.3 The assignment of bit strings to code value

Bit string form	Range of code value
1	0
0 1 x_0	1-2
0 0 1 $x_1 x_0$	3-6
0 0 0 1 $x_2 x_1 x_0$	7-14
0 0 0 0 1 $x_3 x_2 x_1 x_0$	15-30
0 0 0 0 0 1 $x_4 x_3 x_2 x_1 x_0$	31-62
...	...

Bit string	Code value
1	0
010	1
011	2
00100	3
00101	4
00110	5
00111	6
0001000	7
...	...



4.2.1 Circuit design of Exp-Golomb Decoder

According to the property that the number of leading zeros corresponding to the length of the Exp-Golomb code, we use an up-down counter for controlling the processing cycles of an incoming Exp-Golomb code. With the help of this up-down counter, we can also easily calculate the power of 2 of this up-down counter for the implementation of the power part of the code value. With an accumulator in the $ue(v)$ calculator and some control logics, the code value of $ue(v)$ can be calculated. Two output converters at the output stage converts the $ue(v)$ (unsigned value) to the $se(v)$ (signed value) and $me(v)$ (mapped value for coded_block_pattern), and also $te(v)$ (truncated value), for different requirement of coding the system parameters. Fig. 4.6 shows the circuit design of Exp-Golomb decoder.

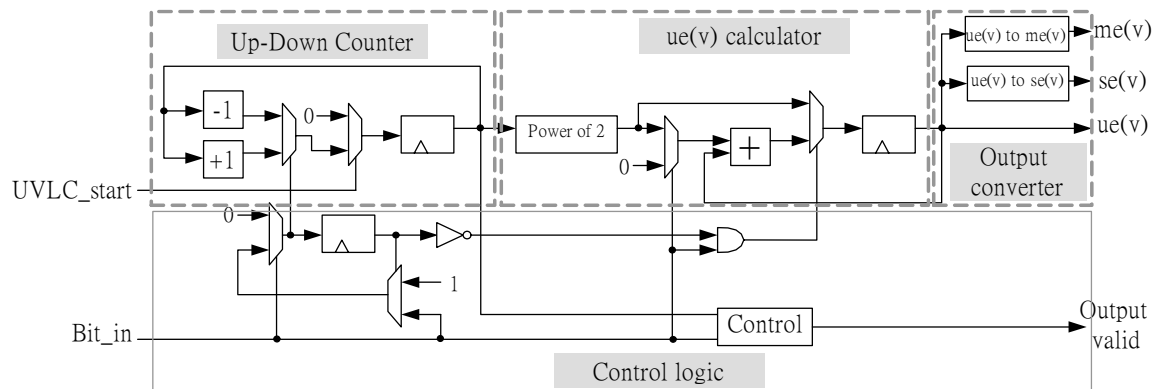


Fig. 4.6 Circuit Design of Exp-Golomb Decoder

4.2.2 Reusability of the Exp-Golomb Decoder

From the standard, we can see that the Exp-Golomb code exists in many syntax parsing modules in $ue(v)$, $se(v)$, $me(v)$, or $te(v)$ forms. Such as SPS (Sequence Parameter Set), PPS (Picture Parameter Set), Slice Data Partition, Slice Header, Reference Picture List Reordering, Prediction Weight Table, Decoding Reference Picture Marking, Slice Data, Macroblock Layer, Macroblock Prediction, and Sub-Macroblock Prediction. Thus we carefully design the interface of the Exp-Golomb Decoder such that a single Exp-Golomb decoder can be shared with all these syntax parsing modules.

The hand-shaking signals to the syntax parsing modules are signal “UVLC_start” and “Output valid”. As fig. 4.7 shows, an “OR”-gate OR-ed all the request signals from syntax parsing modules to active this Exp-Golomb Decoder. And an “Output valid” signals with the result “ $me(v)$ ”, “ $se(v)$ ”, or “ $ue(v)$ ” connect to these syntax parsing modules for the acknowledgement of the decoded value by the decoder. Because there is only one module sending the request signal to the Exp-Golomb at a certain time according to the content of the bit-stream is currently parsing, no conflict will occur and we can thus share a single Exp-Golomb Decoder for all the syntax parsing modules.



Fig. 4.7 Exp-Golomb Decoder shared for all modules in the syntax parser

4.3 H.264/AVC Intra Predictor

Intra prediction doesn't exist in MPEG-2 system, but it is the key operation in every I-frame of H.264/AVC video stream. In H.264/AVC video system, intra prediction can be divided into 2 kinds, the intra4x4 prediction and intra16x16 prediction. There are total 9 modes in intra4x4 prediction (vertical, horizontal, DC, down-left, down-right, vertical-right, horizontal-down, vertical-left, and horizontal-up), and 4 modes in intra16x16 prediction (vertical, horizontal, DC, and plane). These 13 modes in total of the intra prediction make the result of prediction process very accurate to the original image at various types of image.

Whether the mode is chosen, the neighboring pixels (upper and left) must be loaded before the prediction process because the neighboring pixel values exist in the prediction formula in all these 13 modes. Thus to load the neighboring pixels become the first operation in intra prediction process. In our work, we use 3 types of buffers (upper buffer, left buffer, and corner buffer) to store and reuse the neighboring pixels either from/to memory or from previous prediction results. The following gives the description on the operations of these 3 types of buffers.

4.3.1 Low-Power Memory Fetch Upper Buffer Design

To load the upper neighboring pixels is not that direct compared with left neighboring

pixels because the macroblock decoding sequence is row-by-row. Because the upper neighboring pixels are calculated long before and is dependent on picture width, to store the upper pixels in buffers of registers is inefficient and high in cost when decoding high-definition pictures. Thus an embedded slice memory is used to store the upper pixels that calculated long before. The size of this slice memory is N , the number of macroblock in picture width multiplies by 4, and the bandwidth of this slice memory is 32 bits.

We spend 4 cycles loading upper neighboring pixels from memory to our upper buffer at the initial stage of the intra prediction on macroblock. We make the prediction sequence same as the 4x4 block inverse raster scanning sequence defined in the standard. This 4x4 block based prediction sequence is very convenient in implementation of intra4x4 modes and is also practicable in implementation of intra16x16 modes in the way that we split the intra16x16 prediction process into 16 identical intra4x4 prediction processes.

Fig. 4.8 and Fig. 4.9 show the operation of the upper buffer. After fetching upper pixels from slice memory, these upper buffers are updated with the completion of prediction process on every 4x4 block in group of 4 pixels. In this way, only 4 sets of 4-pixel-sized buffers are needed for the upper pixel fetching of every 4x4 blocks. At the end, after all 4x4 blocks are predicted, 4 cycles are needed to store the bottom pixel values to slice memory from these upper buffers. This storing operation in memory also replaces the memory content of the same address in fetching stage.

With the help of these upper buffers, the number of memory access can be greatly reduces from at least 16 times to only 4 times which is the minimum number of memory access of reading operation in all prediction modes. And because memory fetching is very time consuming and power consuming, these upper buffers not only improve the throughput but also saves lots of power in prediction process.

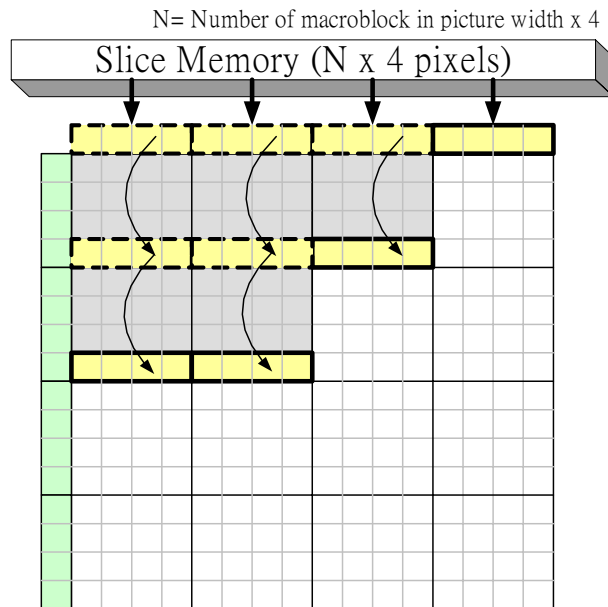


Fig. 4.8 Operation of upper buffer (A)

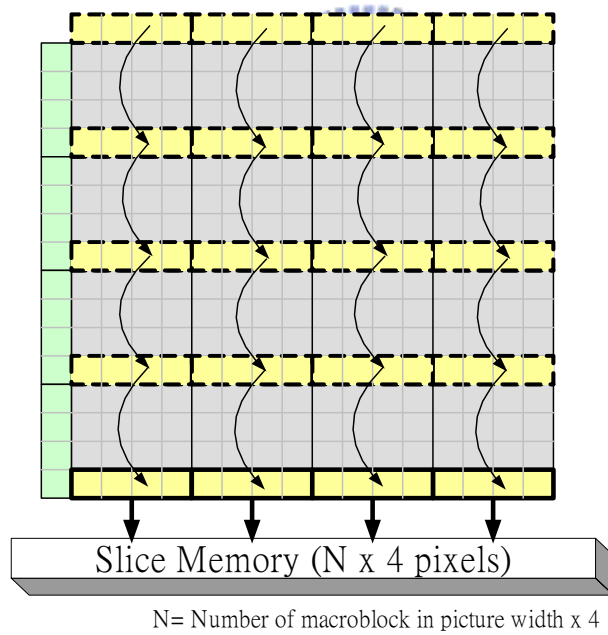


Fig. 4.9 Operation of upper buffer (B)

4.3.2 Reusable Left Buffer Design

Because macroblock are decoded row-by-row, fetching the left neighboring pixels is very easy with the help of only a set of left buffers.

Fig. 4.10 and Fig. 4.11 show the operation of the left buffer. Same as the upper buffer but without fetching operation from memory, values in these left buffers are updated with the completion of the prediction process on every 4x4 block. In this way, the content stored in left buffers is always the left neighboring pixels of the 4x4 block that will be predicted in the nearest future. Moreover, only 4 sets of buffers in group of 4 pixels are needed for all the fetching operations of left neighboring pixels in a macroblock, which is very efficient and the cost is minimum. At the end, when the prediction process on all 4x4 blocks are completed, the content of these left buffers are just all updated to the left neighboring pixels of the following macroblock to be processed. No additional fetching operations are required between macroblocks.

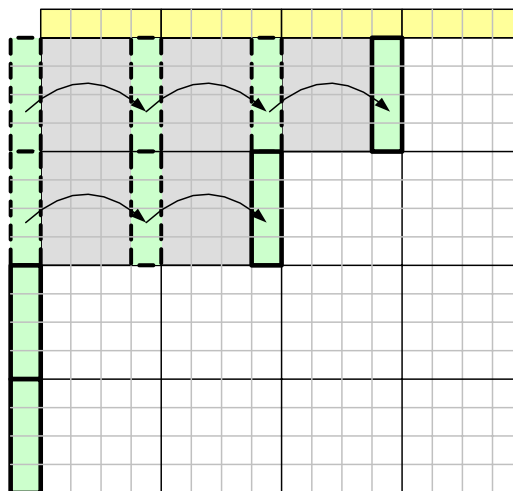


Fig. 4.10 Operation of left buffer (A)

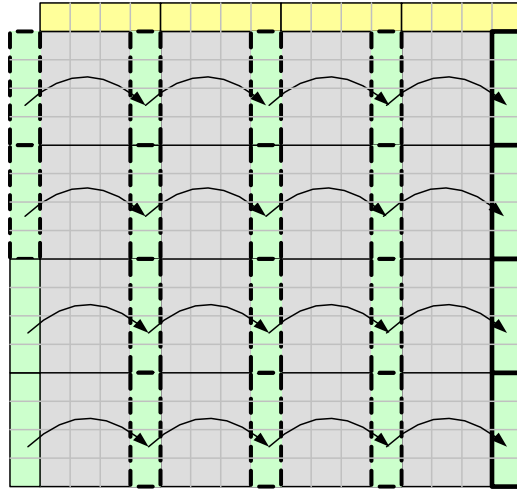


Fig. 4.11 Operation of left buffer (B)

4.3.3 Reusable Corner Buffer Design

Corner (upper-left) pixel is also needed for the prediction process in most of prediction modes. Because the upper buffers and the left buffers can not cover the corner pixel, an additional set of corner buffers are required for the intra prediction process.

Corner buffers consist of eight 1-pixel-sized buffers. Fig. 4.12 and Fig. 4.13 show the operation of these corner buffers. Initially the buffers A1, A2, A3, and C2 are loaded with the upper buffer from memory. C1, B1, B2, and B3 are all calculated in the previous macroblock can be used directly.

Same as the upper buffers and left buffers, the corner buffers are updated one by one with the completion of the prediction process on every 4x4 block. As Fig. 4.12 shows, the corner pixel values in the corner buffers are all updated to the new corner pixel values at the nearest down-right 4x4 blocks. At the end, when the prediction process of all the 4x4 blocks are completed, as Fig. 4.13 shows, we switch the set A1, A2, A3, and C2 with the set B1, B2, B3, and C1. Because at the end of prediction process buffers A1, A2, A3, and C2 are just updated to the left corner of the following macroblock, thus we update buffers B3, B2, B1, and C1 to the upper corner of the following macroblock. Switching these two sets of corner

buffers between macroblocks smoothly accomplished the work of feeding corner values to every intra prediction process on 4x4 blocks.

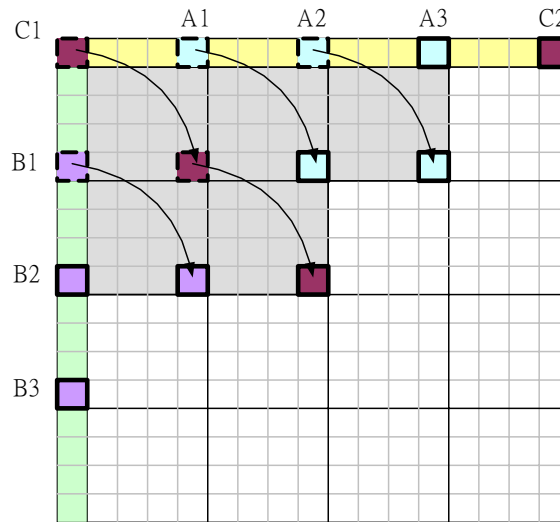


Fig. 4.12 Operation of corner buffer (A)

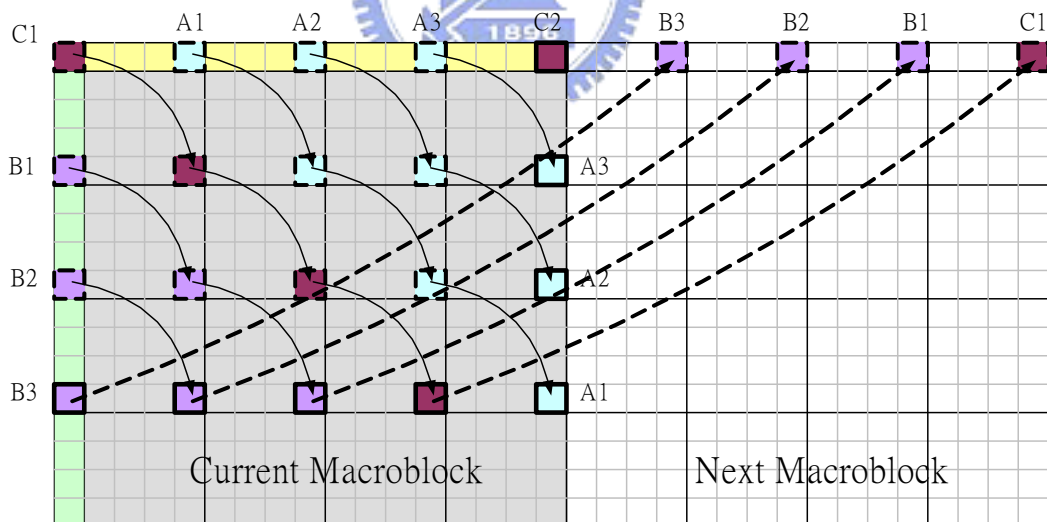


Fig. 4.13 Operation of corner buffer (B)

4.3.4 Intra Predictor for Directional Based Modes

With the help of the upper, left, and corner buffer introduced above, the complexity of the predictor for directional based modes reduces a lot. In directional based intra prediction modes, like vertical, horizontal, down-left, down-right, vertical-right, horizontal-down, vertical-left, and horizontal-up both in intra4x4 or intra16x16, the prediction formula can always be written as the following form:

$$pred[x, y] = ((A + B + C + D) + 2) \gg 2$$

Thus in our predictor for directional based modes design, as Fig. 4.14 shows, we simply calculate the prediction output by first selecting the corresponding neighboring pixels A, B, C, and D from upper, left, or corner buffers, and then do the adding and rounding.

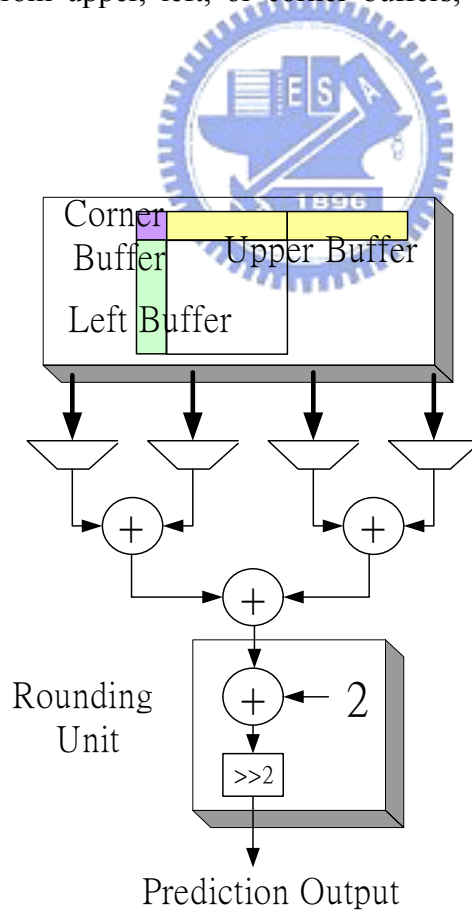


Fig. 4.14 Intra predictor for directional based modes

4.3.5 Intra Predictor for DC Mode

In DC mode, the predictor has to calculate the average pixel value of the upper and left neighboring pixels. Thus we can simply design an accumulator to implement this operation. And we can increase the throughput by adding some additional adders.

This intra predictor for DC mode can also be shared with the slope calculator for plane prediction, will be described in the following section.

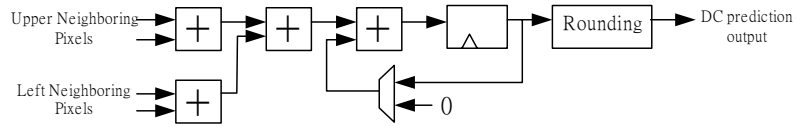


Fig. 4.15 Intra predictor for DC mode

4.3.6 Intra Predictor for Plane Prediction

Plane prediction is the most complex part in intra prediction. The followings are the formula of intra plane prediction.

For luma samples,

$$pred_l[x, y] = Clip1((a + b * (x - 7) + c * (y - 7) + 16) \gg 5)$$

Where

$$a = 16 * (p[-1,15] + p[15,-1])$$

$$b = (5 * H + 32) \gg 6$$

$$c = (5 * V + 32) \gg 6$$

$$H = \sum_{x'=0}^7 (x'+1) * (p[8+x',-1] - p[6-x',-1])$$

$$V = \sum_{y'=0}^7 (y'+1) * (p[-1,8+y'] - p[-1,6-y'])$$

And for chroma samples,

$$pred_c[x, y] = Clip1((a + b * (x - 3) + c * (y - 3) + 16) \gg 5)$$

Where

$$a = 16 * (p[-1,7] + p[7,-1])$$

$$b = (17 * H + 16) \gg 5$$

$$c = (17 * V + 16) \gg 5$$

$$H = \sum_{x'=0}^3 (x'+1) * (p[4+x',-1] - p[2-x',-1])$$

$$V = \sum_{y'=0}^3 (y'+1) * (p[-1,4+y'] - p[-1,2-y'])$$

Because the prediction values of plane prediction varying smoothly both along the x-axis and y-axis in a macroblock (that's why it named plane prediction), we can see from the formula that the value b and c just like the slope of x and y axis in this virtual plane. We calculate these slope values first as Fig. 4.16 shows, and for the pipeline issue which will be introduced later, 2 sets of registers - slope_b and slope_c (for Y & Cr), slope_d and slope_e (for Cb) are used to store the calculated slopes in plane prediction. And also, the accumulator part in the slope calculator can be shared with DC predictor which described in the previous section.

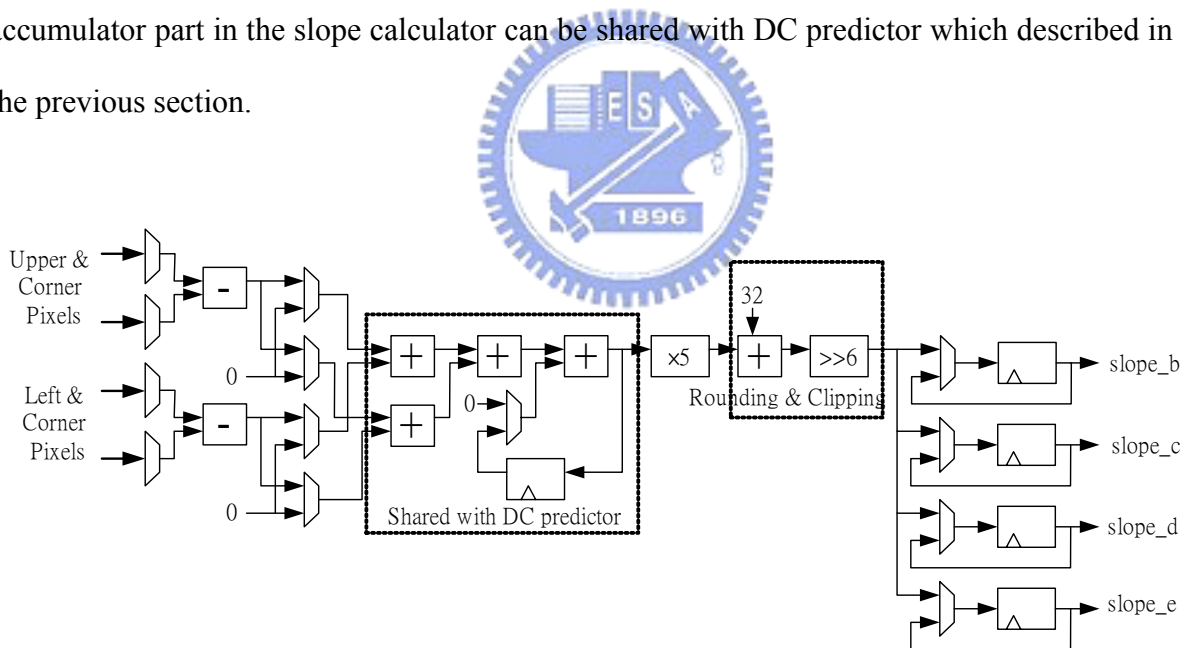


Fig. 4.16 Slope calculator for plane prediction

From [10], for the final prediction output, we can first calculate an intermediate value, a1 where

$$a1 = a + b * (-7) + c * (-7) + 16$$

Then $pred_L[x,y]$ can easily be calculate from the a1

$$\begin{aligned} pred_L[0,0] &= Clip(a1 \gg 5), \\ pred_L[1,0] &= Clip((a1 + b) \gg 5), \\ pred_L[2,0] &= Clip((a1 + 2b) \gg 5), \dots \text{and} \\ pred_L[0,1] &= Clip((a1 + c) \gg 5), \\ pred_L[0,2] &= Clip((a1 + 2c) \gg 5), \dots \end{aligned}$$

After calculating the slope b and c, Fig. 4.17 shows the remaining calculation required for plane prediction of luma samples. And once the a' is calculated, those 3 pixels below it can be calculated by $a'+c$, $a'+2c$, and $a'+3c$.

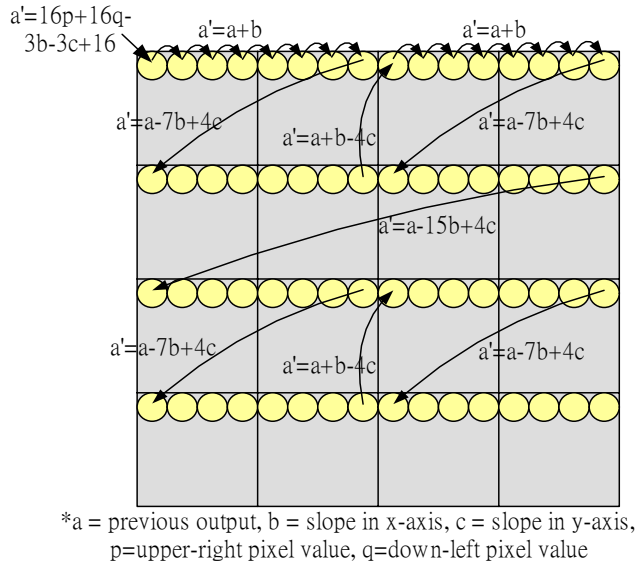


Fig. 4.17 Required calculation for plane prediction

For hardware sharing and low cost issue, we can rewrite these required calculations as the following form,

$$\begin{aligned} a' &= 16p + 16q - 3b - 3c + 16 \rightarrow a' = 4 * (4p + 4q - b - c + 4) + (b + c) \\ a' &= a + b - 4c \rightarrow a' = a + 4(-c) + b \\ a' &= a - 7b + 4c \rightarrow a' = a + 4 * (c - 2b) + b \\ a' &= a - 15b + 4c \rightarrow a' = a + 4 * (c - 2b - 2b) + b \end{aligned}$$

In this way, no multiplication is required for the plane prediction. Combined with plane

prediction on Cb and Cr, Fig. 4.18 shows the intra predictor for plane prediction.

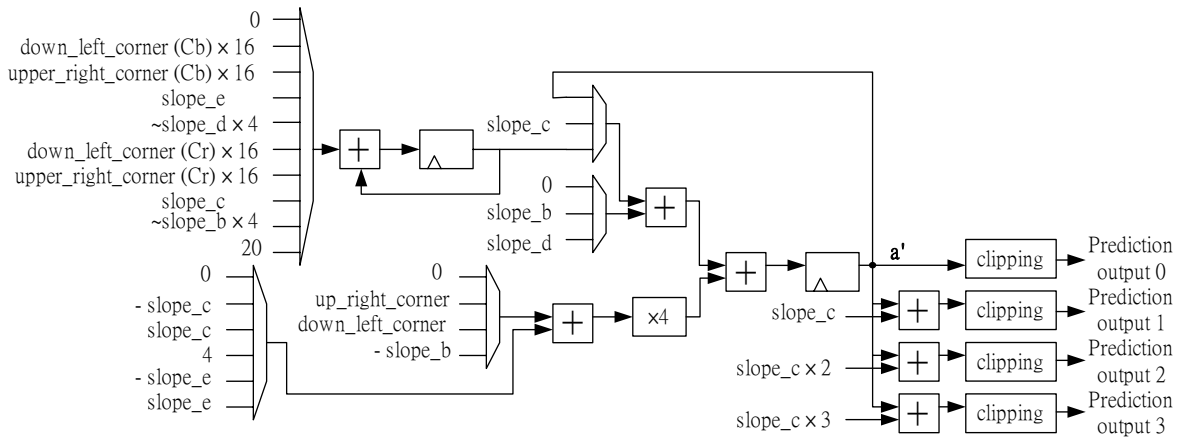


Fig. 4.18 Intra predictor for plane prediction

Fig. 4.19 shows the pipeline scheme for the situation that plane prediction applies on both luma and chroma samples. With the additional slope storage (slope_d and slope_e) for slope calculator, the total processing cycles can be reduced.

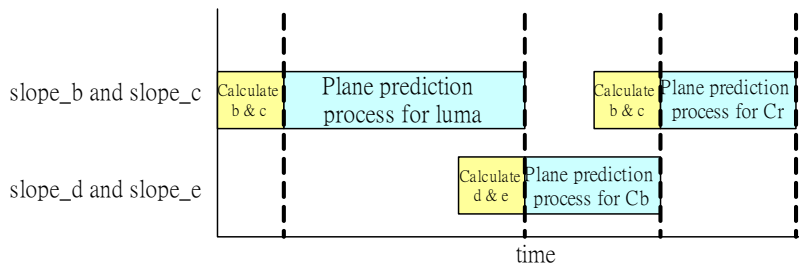


Fig. 4.19 Intra predictor for plane prediction

4.4 MPEG-2 & H.264/AVC Inverse DCT

Inverse DCT (Discrete-Cosine-Transform) is a key component in decoder design. Inverse DCT operation is very complex and time consuming in MPEG-2 system. The complexity raises proportional to the square of block size. Thus for the 8x8 block based

MPEG-2 system, IDCT is often the bottleneck of the system throughput. In H.264/AVC system, though Inverse DCT is simplified to integer transform, we still need to design a high throughput IDCT which suits for digital TV application.

Processing IDCT operation requires many multiplication and addition. From [6][17], we can separate a 2-D IDCT into two 1-D IDCTs, by which the computational complexity can be greatly reduced. In the 1-D IDCT design, for the trade-off between throughput and cost, we have choices to design a serial-in-serial-out version or a parallel-in-parallel-out version. From Table 4.3, we can see the result and the comparison for these 2 implementations. We here choose the parallel-in-parallel-out version

Table 4.3 Area and Power reduction on register-sharing version

Architecture	Serial-in-Serial-out	Parallel-in-Parallel-out
Gate count	22 K	46 K
Throughput	102 cycles/8x8block	32 cycles/8x8block
Clock rate for HDTV application*	155 MHz	47 MHz

*HDTV application: Assuming no any other latency, real time playing sequence of
1920x1088@30fps

4.4.1 2-Stage IDCT Architecture

The following shows the formula of a 2-D IDCT.

$$x(m, n) = \frac{2}{N} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} a(k)a(l)Y(k, l) \times \cos \frac{(2m+1)\pi k}{2N} \times \cos \frac{(2n+1)\pi l}{2N}$$

where $a(0) = \sqrt{\frac{1}{2}}$ and $a(k) = 1$ for $k \neq 0$

and $n, m, k, l = 0, \dots, N-1$

Which equals to the following equations

$$Z(k, n) = \sum_{l=0}^{N-1} \sqrt{\frac{2}{N}} a(l) Y(k, l) \times \cos \frac{(2n+1)\pi l}{2N}$$

$$x(m, n) = \sum_{k=0}^{N-1} \sqrt{\frac{2}{N}} a(l) Z(n, k) \times \cos \frac{(2m+1)\pi k}{2N}$$

Thus we can split a 2-D IDCT into two 1-D IDCTs with a transpose matrix as Fig. 4.20 shows.

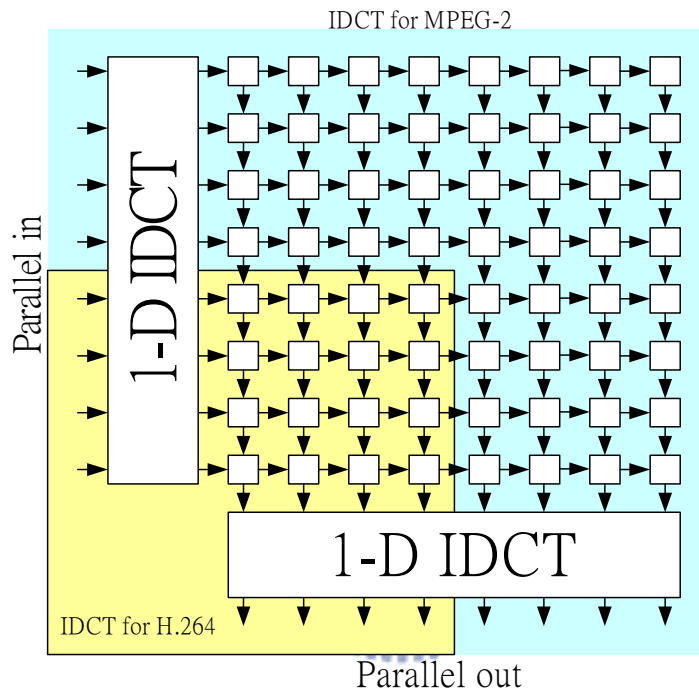


Fig. 4.20 2-stage IDCT architecture

4.5 MPEG-2 & H.264/AVC Combined Motion

Compensator

Processing motion compensation is the most complex operation in H.264/AVC system. Various block size from 4x4 to 16x16 plus up to 1/4 resolution on motion vectors increase the complexity on motion compensator design.

4.5.1 Motion Compensation Engine

Fig. 4.21 shows the motion compensation engine. It is composed of motion vector predictor, interpolator and reconstruction. Firstly, motion vector predictor generates MVP value according to motion vectors of neighboring blocks, which is stored in shift registers and mv buffers. The interpolator then fetches the proper samples from external reference frame memory according to the address calculated from address generator. These interpolated data add to residual data which is calculated from entropy decoder, rescaling and IDCT. Finally, the reconstructed data restores to external frame memory after de-blocking filter. Two frame memories are exploited to keep current frame and previous frame reciprocally.

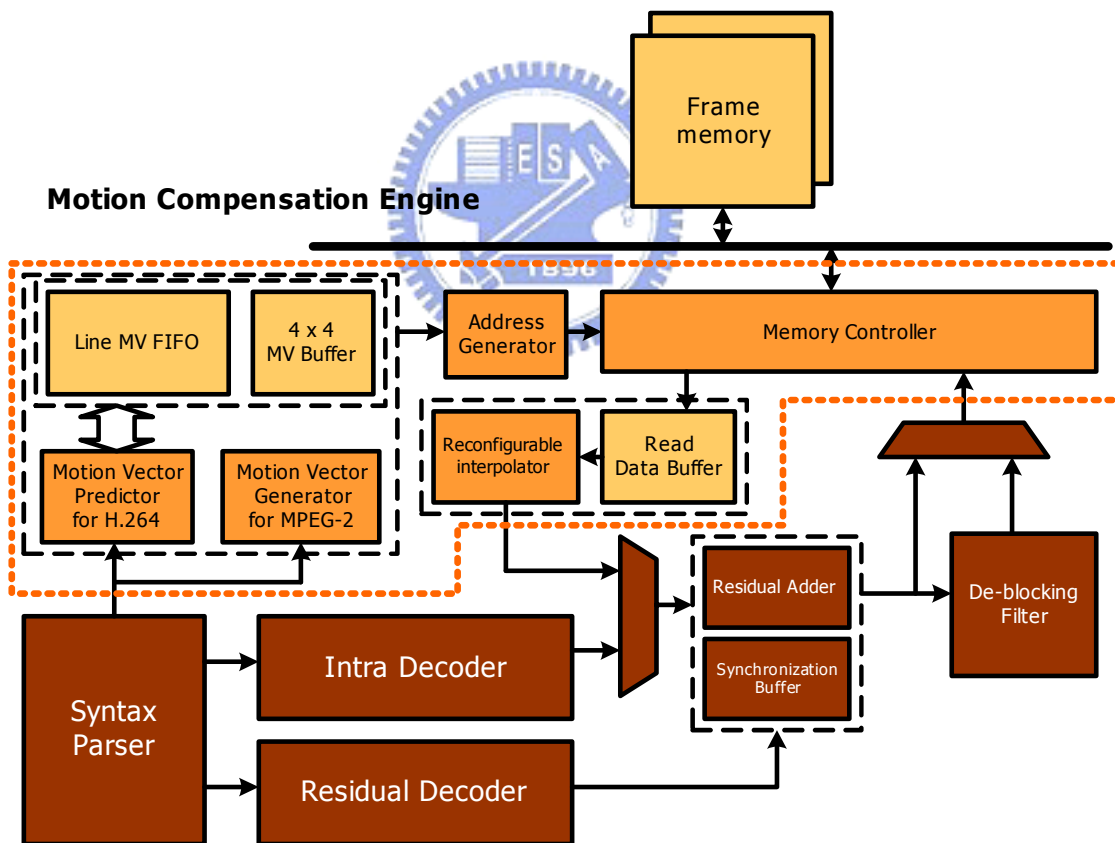


Fig. 4.21 Motion compensation engine

4.5.2 Interpolator

Of all the components in motion compensator, the interpolation unit is always the most time-consuming both in H.264/AVC and MPEG-2 systems. Some interpolation schemes or architectures applied in recent standards have been proposed [1][9][15][16]. They can be classified into three structures: 1-D based [1] [15], 2-D based [9] and separate 1-D based [9] [16] approaches. 1-D based approach has lowest cost; however, it is not efficient for memory access and data reuse, especially in high motion resolution for H.264/AVC. 2-D based approach may cause longest latency and highest cost in multiple high coefficients interpolation. Among three approaches, separated 1-D approach can achieve the most efficient data reuse. Therefore, we developed a novel interpolator based on separate 1-D approach to improve the data reuse and greatly reduce the memory access.

The concept of the interpolator is to separate a 2-D FIR interpolator into vertical and horizontal 1-D FIR interpolators. Considering interpolation in H.264/AVC system, the half sample is interpolated by applying 6-tap FIR filter (1, -5, 20, -5, 1) and quarter sample is performed by using bilinear filter. And for MPEG-2 system, the resolution on motion vector is only up to 1/2, this down-compatible interpolator can be shared for both systems.

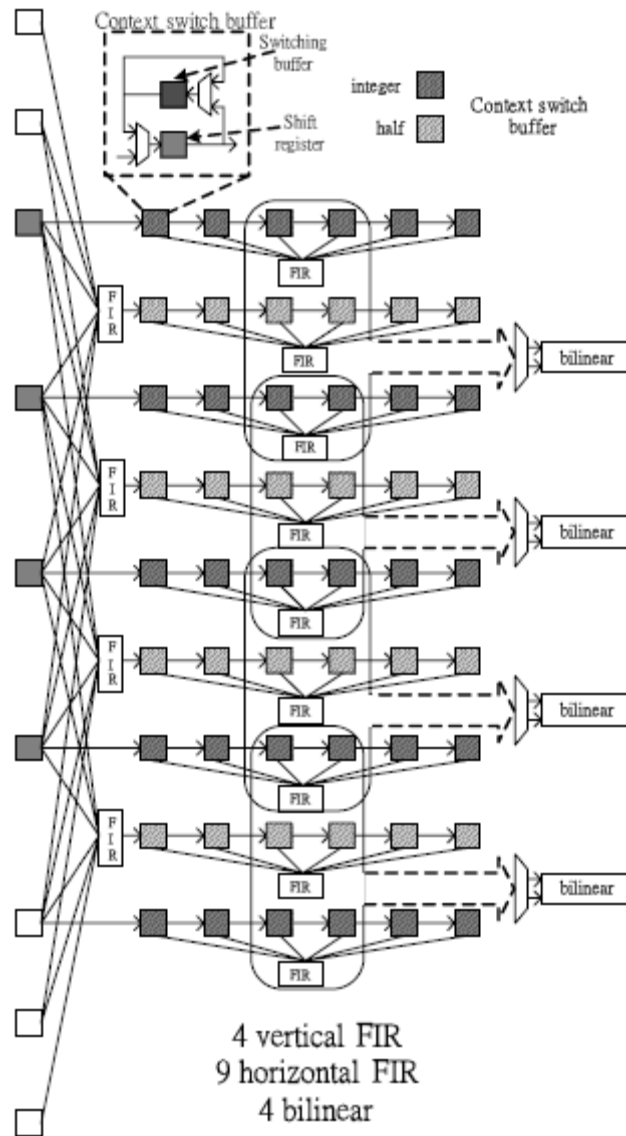


Fig. 4.22 Motion compensation engine

4.6 MPEG-2 & H.264/AVC Combined De-blocking Filter

De-blocking filter can be used as a post-processing operation in MPEG-1/2/4 standard. The MPEG-2 standard doesn't mention about the de-blocking filter at all. But it is mentioned in MPEG-4 standard with informative process. That is, algorithms and design of de-blocking filter in MPEG-2 standard is free to the designer without any constraints that the designers have freedom choosing the most suitable architecture and algorithms of the

de-blocking filter. However, in H.264/AVC video coding system, because of the importance of the de-blocking filter has risen, the de-blocking filter becomes an in-loop filter and is definitely included in the H.264/AVC standard. Different from MPEG-2 system, to design a standard compliant de-blocking filter is necessary in H.264/AVC system.

In our work, we combine the post-processing filter for MPEG-2 system with the in-loop de-blocking filter in H.264/AVC system into a single de-blocking filter. We maintain the filtered edge of 4x4 in H.264/AVC system and 8x8 in MPEG-2 system. For the 8x8 based de-blocking filter design, we adopted and do a little modification on the post-processing de-blocking filter defined in MPEG-4 Annex F.3 as our post-filter for MPEG-2 system.

4.6.1 Triple-Mode Decision

In the edge filtering design, there are three modes introduced. According to the strength and mode decision, we develop triple Pixel-in-Pixel-out (i.e. P-i-P-o) edge filter to realize the in/post-loop de-blocking filter. We modify the default mode of post-loop filtering process and apply the algorithm of strong and weak mode in the in-loop filter. The default mode is of prime concern while the DC offset mode tends to be less occurred. Further, the DC offset mode is broadly similar to that in the in-loop filter, in the sense that we can apply the filter process of “bS=4” instead of that in MPEG-4 Annex F.3 with a little performance loss. Further, we modify the approximated DCT kernel (i.e. [2 -5 5 -2]) into [2 -4 4 -2]. That is, we make use of shifter instead of constant multiplier. Based on the above discussion, we deduce that three data flows will be generated in our triple P-i-P-o filter algorithm. They are strong, weak and skip filtering process. Particularly, the strong mode filtering in in-loop and post-loop has been merged into single structure. The data flows of the proposed hybrid filter algorithm are depicted on Fig. 4.22.

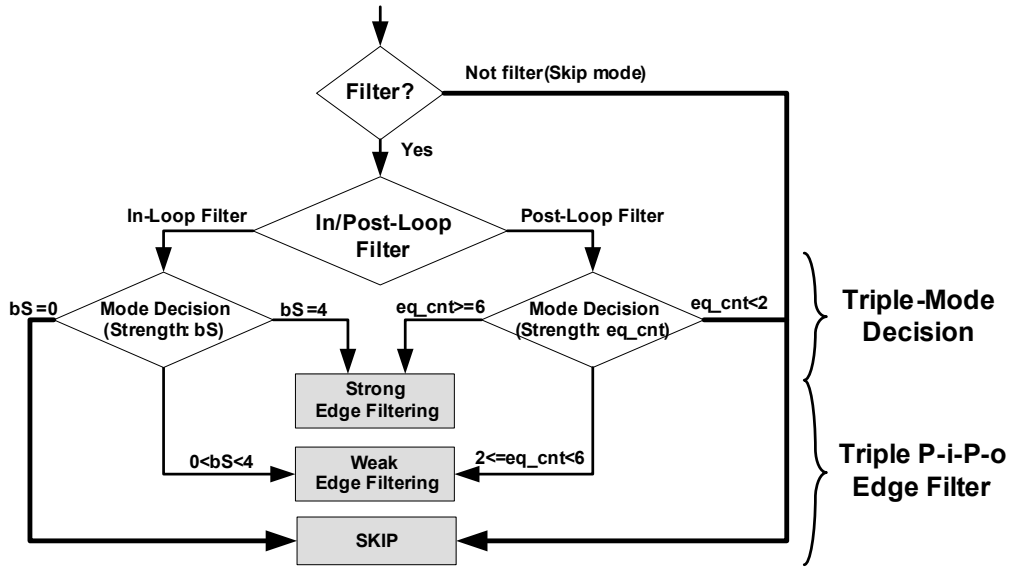


Fig. 4.23 The data flow of the in/post-loop filter algorithm

4.6.2 Slice and content memory

To facilitate the data access with each block pixel or neighboring pixel, we use two slice memory (single-port SRAMs) and content memory to keep the neighboring pixel and block-content pixel value. The fetching and restoring pixel value is very frequently since de-blocking filter in H.264/AVC is performed on each 4x4 block level. To reduce the pin counts and speed up the filtering process, internal SRAM module is essential to meet the real-time decoding demand.

The slice memory is used to store the neighboring pixel. It is required to keep them until they have been filtered completely. Further, the address depth is decided by the frame width in the slice memory. In Fig. 4.24(a), considering the frame size with $M \times N$, 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 about $2N \times 32$

for the 4:2:0 format.

The content memory is used to store the unfiltered pixel value in luma or chroma block. The data word-length of memory is 32-bit, 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 $(16+8)*4 \times 32$ in total. Further, the data address is increased as the standard-defined block ordering of Fig. 4.24(b). The grid region is stored in the slice memory and the dotted region is stored in the content memory

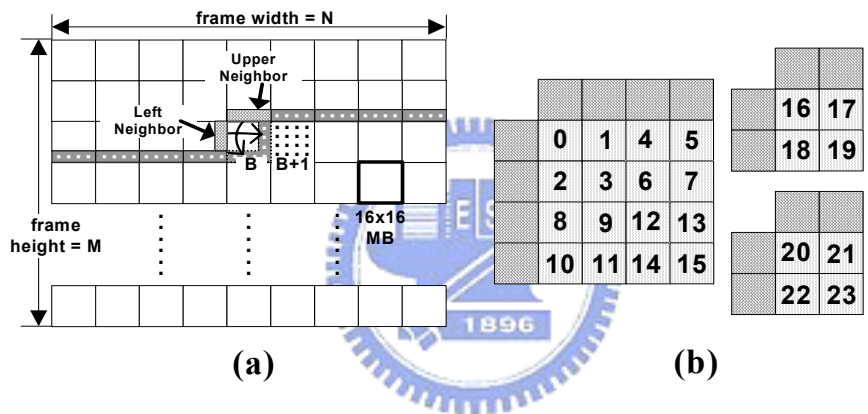


Fig. 4.24 (a) Slice memory with grid or shaded region and
(b) Content memory with black-dotted region

4.6.3 Hybrid scheduling

To reduce the overhead with the reloaded data when switching the filtering edge from horizontal to vertical, we adopt a hybrid filter scheduling to re-schedule the standard-defined edge. The de-blocking filter in H.264/AVC system 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 4×4 block as Fig. 4.25(a). In the filter ordering of one 4×4 block, left edge is filtered first and lower edge is the last one. We

develop a novel filter ordering to schedule our filter process on each edge as Fig. 4.25(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 [13][14], our method prevents the re-access for different direction and combine the vertical and horizontal filter at the rule of standard-compliance.

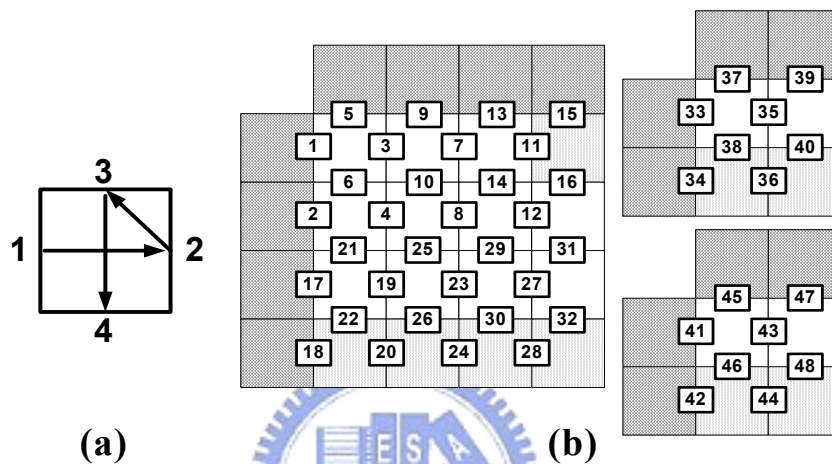


Fig. 4.25 Hybrid scheduling method

We use four 4x4 pixel buffer to keep the temporary data in our hybrid scheduling process. In Fig. 4.26 (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 Fig. 4.26 (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 certain time instance (except for the initial state $t/1$) 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.

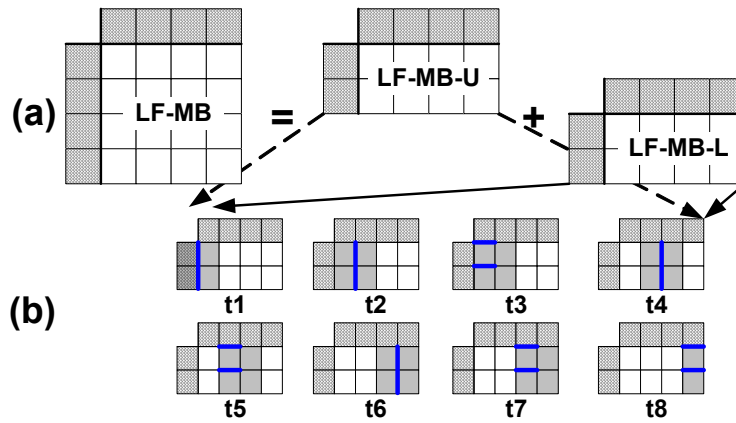


Fig. 4.26 The partitioned MB and each time instance when applying the hybrid scheduling method

We derived the filter ordering of the proposed hybrid scheduling method in Fig. 4.26(b). Each bold line represents the edge to be filtered in each time instance. The filtered ordering complied with the hybrid scheduling in Fig. 4.25(a) at each time instance t1 ~ t8. By the same way, the proposed scheduling is also performed in the block of chroma 4x4 block.

The main problem of in/post-loop de-blocking filter 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 use a high-throughput architecture of de-blocking filter. Fig. 4.27 shows the proposed design with block diagram and data flow representation. The external frame buffer is an off-chip memory and the size is decided by the frame size. The shaded-arrows denote the data flow inside the de-blocking 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 four 4x4 pixel values. Moreover, in each time instance, it locates at the position as the shaded regions of Fig. 4.26(b) shows.

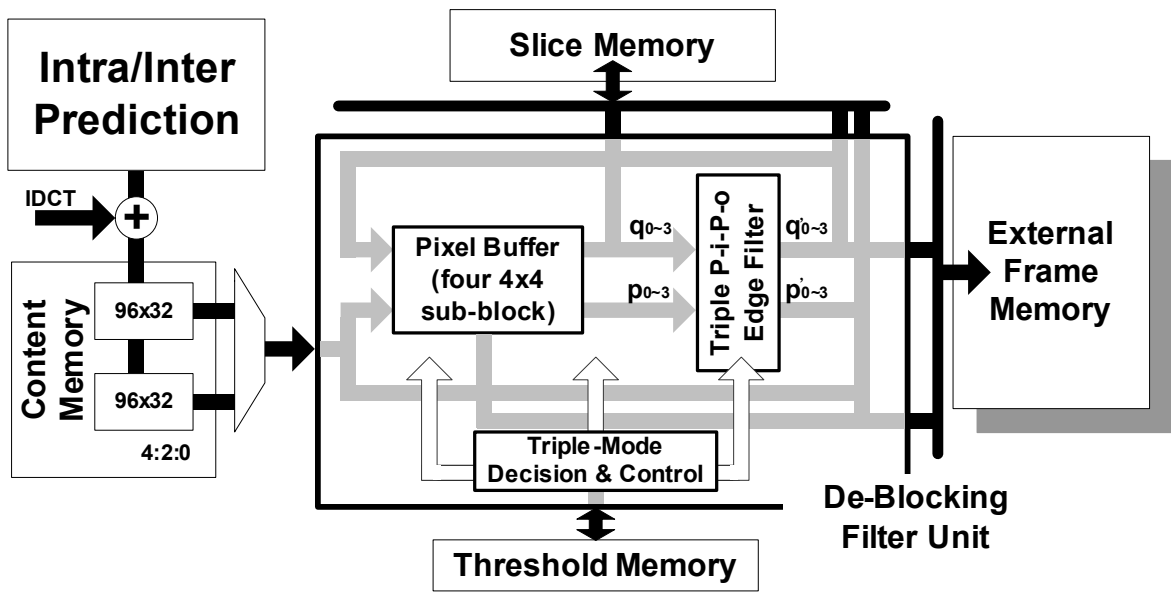


Fig. 4.27 The block diagram and data flow of the MPEG-2/H.264 combined de-blocking filter





Chapter 5

Chip Implementation for Digital TV Applications

5.1 System Specification

In our MPEG-2/H.264 dual mode decoder design, the specification of the MPEG-2 part is MPEG-2 simple profile at main level (SP@ML), table 5.1 shows the details of this profile. In the H.264/AVC part, our specification is H.264/AVC baseline profile at level 3.2, table 5.2 shows the details of this profile.

Table 5.1 Simple profile @ Main level of MPEG-2 system

No. of layers	Layer id	Scalable mode	Maximum sample density (H/V/F)	Maximum sample rate	Maximum total bit rate /1000000	Maximum total VBV buffer	Profile and level indication
1	0	Base	720/576/30	10,368,000	15	1,835,008	SP@ML

Table 5.2 Baseline profile @ level 3.2 of H.264 system

Level	Max macroblock processing rate MaxMBPS (MB/s)	Max frame size MaxFS (MBs)	Max decoded picture buffer size MaxDPB (1024 bytes)	Max video bit rate MaxBR (1000 bits/s or 1200 bits/s)	Max CPB size MaxCPB (1000 bits or 1200 bits)	Vertical MV component range MaxVmvR (luma frame samples)	Min compression ration MinCR	Max number of motion vectors per two consecutive MBs MaxMvsPer2MB
3.2	216,000	5,120	7,680.0	20,000	20,000	[-512,+511.75]	4	16

The maximum computational capability is to support real time decoding of 1080i (1920x1088) MPEG-2 video sequence and SXGA (1280x1024) H.264 video sequence in

30fps. Our operational frequency required for MPEG-2 is 80.92MHz, and for H.264 is 79.64MHz.

5.2 Design Flow

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

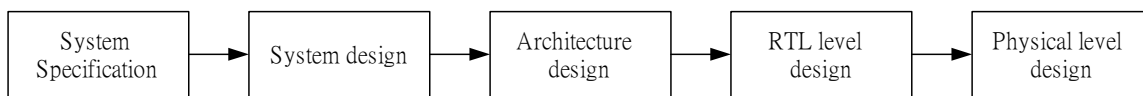


Fig. 5.1 Design flow from system specification to physical-level

In system design stage, first we estimated the required throughput for the specification, applied the 4x4-sub-block level pipeline scheme and modified it to hybrid scheme for the reason that macroblock-level pipelining scheme is suitable for some modules. We carefully estimate the efficiency of different decoding ordering for all the modules because it would be an important interface between modules. We choose 1x4-column-by-column decoding ordering for the implementation at last. Because we aimed at multi-mode decoder design, the hardware sharing issue shall be considered as well in this first stage. The overall block diagram and data flow is designed in this stage.

In architecture design stage, we divide the work mainly to 4 people, one for motion compensation, one for entropy decoding, one for de-blocking filter, and one for the system design and other modules (me). We have to consider the hardware sharing issue for both systems in designing each module. The throughput required is the aim of designing each module. Under the constraint of the throughput requirement, we focus on the architecture design and to make each module low-complexity and low-power. Some low-complexity architecture and low-power techniques are derived in this stage.

The RTL-design is along with the architecture design. The work for RTL-design is mainly to translate the architecture of each module to RTL description. To make the synthesis result identical to the architecture of our design is the goal of the RTL-design. Of course that some coding techniques for the synthesizer are considered in this stage. To write the RTL-code synthesizable and easy understanding is also important.

In physical design stage the CAD tools are important. To make a good use of these tools and to do the remaining job to the best is the key point to our final result. The design margin, technology used, some nano effects on deep sub-micron circuits are also needed to be considered. At the end of the physical design stage, our work is taped-out for the prototyping and final verification.

5.3 *Implementation Result*

In our work, we implemented an MPEG-2/H.264 dual mode decoder. Fig. 5.2 shows the layout of this work. The total gate count is about 491K, chip size is 3.9x3.9mm² in 0.18um technology. Maximum working frequency is 83.3MHz, support decoding 720p H.264 video sequence under 56MHz, decoding 720p MPEG-2 video sequence under 35.7MHz in 30fps. Power consumptions are 44.35mW and 30.15mW, respectively.

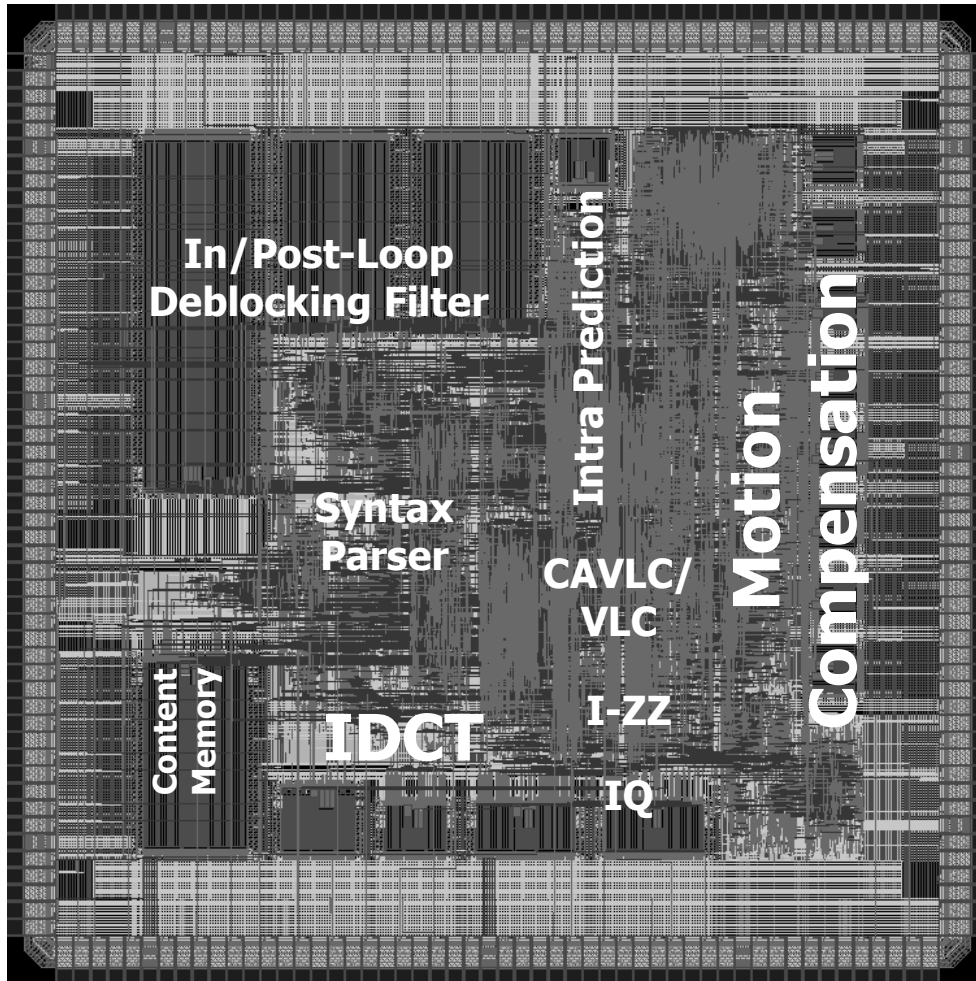


Fig. 5.2 Layout of this work

Table 5.3 Chip details

Items	Specification
Function	H.264 Baseline@Level 3.2 MPEG-2 SP@ML
Gate counts	491,260 (On-chip SRAM included)
Technology	0.18um 1P6M
Supply voltage	3.3V/1.8V
Die size	3.9x3.9mm ²
Package	208CQFP
Max working frequency	83.3MHz
Core Power Consumption (720p@30fps)	44.35mW@56MHz (H.264) 30.15mW@35.7MHz (MPEG-2)

5.4 Measurement Results and Comparison

Table 5.4 shows the power report of our work. The power consumption of decoding CIF, NTSC, and 720pHD MPEG-2 video sequences are 3.12mW, 11.15mW, and 30.15mW; the power consumption of decoding CIF, NTSC, and 720pHD H.264 video sequences are 4.51mW, 16.39mW, and 44.35mW, respectively.

Table 5.4 Power report

Items (Core Power)	MPEG-2's power analysis	H.264's power analysis
720pHD (1280x720)	30.15mW@35.7MHz	44.35mW@56MHz
NTSC (648x486)	11.15mW@13.2MHz	16.39mW@20.7MHz
CIF (352x288)	3.12mW@3.7MHz	4.51mW@5.7MHz

Table 5.5 shows the comparisons to the State-Of-the-Art. It's hard to find a pure ASIC decoder but RISC included or ARM-based works. Thus it's hard to have a fair comparisons. However, we can still see that our work is a good solution to dual mode H.264/MPEG-2 decoder.

Table 5.5 Comparisons

	Proposed [1]-[4] ISCAS'05 VLSI-TSA'05	C&S [11] ISCAS'04	Conexant [18] ISCE'04	NTU [19] ISCAS'05
Specification	1280x720@30fps	1920x1088@30fps	2048x1024@30fps	2048x1024@30fps
Operating Frequency	56MHz	130MHz (local bus:170MHz)	200MHz	120MHz
Technology	180nm (1.8V)	130nm (1.2V)	130nm (1.2V)	180nm (1.8V)
Profile	H.264 baseline MPEG-2 SP@ML	H.264 baseline MPEG-4 SP H.261,H.263,JPEG	H.264 main	H.264 baseline
Implementation	ASIC	ARM-based	ARM-based	ASIC+RISC
Gate Count	491K	910K	300K	217K
Internal Memory	24K bytes	N/A	74K bytes	10K bytes
Power	44.35mW	554mW	160mW	N/A
Normalized power*	100.92mW	2422.88mW	691.89mW	N/A

*Normalized to 180nm(1.8v), 2048x1024@30fps

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this work, we implemented a dual mode H.264/MPEG-2 video decoder. We adopt many design techniques both on system-point-of-view and architectures.

From the system point of view, first we proposed the hybrid 4x4-sub-block pipelining scheme, by which we can save 93.75% intermediate buffers compared with macroblock-level pipelining scheme at the penalty of slightly throughput degradation. The instantaneous switching scheme reduces the latency to minimum during pipeline stages. Second, we proposed the efficient 1x4-column-by-column decoding ordering, by which the 28% memory access times and 28% processing cycles in motion compensation process can be saved. 17% memory access times can be saved as well in intra predictor. Third, we proposed a variable length FIFO architecture for the synchronization problems in adding pixels from residual/prediction paths. Forth, the exploration on coded-block-pattern technique saves power in inverse quantizer and IDCT modules from 30% to 86% under qP ranging from 20 to 48.

In architecture design, first we proposed a hierarchical syntax parser. The hierarchical syntax parser is easy to design and is very suitable for the bit-stream in hierarchical structure. With the hierarchical enable signals in these parsers, the power savings by clock-gating technique can be up to 86% in these parsers. Second, the register sharing technique is applied on syntax parsers for both systems. This technique reduces the amount of registers required for both system and 26% registers can be saved. Third, we implement

the Exp-Golomb decoder for parsing the H.264 bit-stream. The dedicated interface of this decoder enables this decoder to be shared for all parsers. Forth, 3 types of reusable buffers in intra predictor are proposed. By the aids of these reusable buffers (upper, left, and corner), implementation of the directional modes becomes very easy and the memory access times can be reduced also.

In our final chip implementation, the total gate-count is about 491K, maximum working frequency is 83.3MHz, supports real time decoding 720pHD H.264 sequence@56MHz and 720pHD MPEG-2 sequence @35.7MHz in 30fps. The power consumption for these 2 systems is 44.35mW (720p H.264 sequence) and 30.15mW (720p MPEG-2 sequence), respectively.

6.2 *Future Work*

In our future work, first, we will try to integrate and combine more functional blocks for both systems like IDCT, and inverse quantizer. For IDCT, we will take efforts on splitting the 8x8 IDCT formula into 2-stage 4x4 and 2x2 IDCT so that the 4x4 IDCT module for H.264 can be shared for the 8x8 IDCT operation of MPEG-2. Then, we will try to add the CABAC with other functional blocks to our current work to support H.264 main profile. We will also try to find the critical path in our work such that we can speed up our decoder to work under more than 120MHz to support real time decoding 1080i H.264 video sequence in 30fps.

Bibliography

- [1] Ting-An Lin, Sheng-Zen Wang, Tsu-Ming Liu and Chen-Yi Lee, "An H.264/AVC Decoder with 4x4-block level pipeline", ISCAS 2005
- [2] Ting-An Lin, Tsu-Ming Liu and Chen-Yi Lee, "A Low-Power H.264/AVC Decoder", VLSI-TSA 2005
- [3] Sheng-Zen Wang, Ting-An Lin, Tsu-Ming Liu and Chen-Yi Lee, " A New Motion Compensation Design For H.264/AVC Decoder", ISCAS 2005
- [4] Tsu-Ming Liu, Wen-Ping Lee, Ting-An Lin and Chen-Yi Lee, "A Memory-Efficient Deblocking Filter For H.264/AVC Video Coding", ISCAS 2005
- [5] Shih-Hao Wang, Wen-Hsiao Peng et al., "A Platform-Based MPEG-4 Advanced Video Coding (AVC) Decoder with Block Level Pipelining", Information, Communications and Signal Processing, ICICS-PCM December 2003
- [6] Tung-Chien Chen, Yu-Wen Huang, and Liang-Gee Chen, "Analysis and design of macroblock pipelining for H.264/AVC VLSI architecture", ISCAS 2004
- [7] Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification" ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC, May 2003
- [8] Iain E. G. Richardson, "H.264 and MPEG-4 video compression", John Willey & Sons, autumn 2003, ISBN 0-470-84837-5
- [9] Ville Lappalainen, Antti Hallapuro, and Timo D. Hamalainen, "Complexity of Optimized H.26L Video Decoder Implementation", Circuits and Systems for Video Technonlogy, IEEE Transactions, July 2003
- [10] Yu-Wen Huang, Bing-Yu Hsieh, Tung-Chien Chen, and Liang-Gee Chen, "Hardware architecture design for H.264/AVC intra frame coder", ISCAS 2004

- [11] Hae-Yong Kang, Kyung-Ah Jeong, Jung-Yang Bae, Young-Su Lee, Seung-Ho Lee, "MPEG4 AVC/H.264 decoder with scalable bus architecture and dual memory controller", ISCAS 2004
- [12] K. Suhring, Ed., JM 8.2 reference software (online), 2004. Available at <ftp://ftp.imtc.org/jvt-experts/>
- [13] 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.
- [14] 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.
- [15] He-Wei Feng, Zhi-Gang Mao, Jin-Xiang Wang, Dao-Fu Wang, "Design and implementation of motion compensation for MPEG-4 AS profile streaming video decoding,". 5th International Conference on ASIC, Oct. 2003. Proceeding.
- [16] Tung-Chien Chen, Yu-Wen Huang, and Liang-Gee Chen, "Fully utilized and reusable architecture for fractional motion estimation of H.264/AVC," IEEE International Conference on Acoustics, Speech, and Signal Processing, May 2004.
- [17] Shu-Tzu Lin, Chen-Yi Lee, "Analysis and design of a high-throughput two dimension inverse scan discrete cosine transform processor", Master Thesis, Department of Electronics Engineering, National Chiao Tung University, Taiwan, June 2000
- [18] Y. Hu, A. Simpson, K. McAdoo, and J. Cush, "A high definition H.264/AVC hardware video decoder core for multimedia SoC's," Proc. ISCE 2004.
- [19] T. W. Chen, Y. W. Huang, T. C. Chen, Y. H. Chen, C. Y. Tsai, and L. G. Chen, "Architecture Design of H.264/AVC Decoder with Hybrid Task Pipelining for High Definition Videos," Proc. ISCAS 2005.

作者簡歷

姓名：林亭安

出生地：台灣省台北市

出生日期：1980. 11. 24

學歷：1987. 9 ~ 1993. 6 台北市立大安國民小學

1993. 9 ~ 1996. 6 台北市立和平國民中學

1996. 9 ~ 1999. 6 台北市立松山高級中學

1999. 9 ~ 2003. 6 國立交通大學 電子工程系 學士

2003. 9 ~ 2005. 6 國立交通大學 電子研究所 系統組 碩士

得獎事績

2000/06 書卷獎

2001/01 書卷獎

2003/05 2003 全國 IC 設計競賽設計完整獎

2003/06 電子實驗專題競賽榮獲殷之同獎學金

2004/05 2004 全國 IC 設計競賽設計完整獎

2004/06 書卷獎

2004/10 2004 全國系統晶片設計比賽-光電通訊類 SIP 組特優

2005/05 2005 全國 IC 設計競賽優等獎

發 表 論 文

- Ting-An Lin, Sheng-Zen Wang, Tsu-Ming Liu and Chen-Yi Lee, "**An H.264/AVC Decoder with 4x4-block level pipeline**", ISCAS 2005
- Ting-An Lin, Tsu-Ming Liu and Chen-Yi Lee, "**A Low-Power H.264/AVC Decoder**", VLSI-TSA 2005
- Sheng-Zen Wang, Ting-An Lin, Tsu-Ming Liu and Chen-Yi Lee, "**A New Motion Compensation Design for H.264/AVC Decoder**", ISCAS 2005
- Tsu-Ming Liu, Wen-Ping Lee, Ting-An Lin and Chen-Yi Lee, "**A Memory-Efficient Deblocking Filter for H.264/AVC Video Coding**", ISCAS 2005
- Ting-An Lin, and Chen-Yi Lee, "**Predictive Equalizer Design for DVB-T system**", ISCAS 2005

