Video Decoder for H.264/AVC Main Profile
Power Efficient Hardware Design

YIM, Ka Yee

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Master of Philosophy
in
Electronic Engineering

The Chinese University of Hong Kong
August 2011

Abstract of thesis entitled:
Video Decoder for H.264/AVC Main Profile
Power Efficient Hardware Design
Submitted by YIM, Ka Yee
for the degree of Master of Philosophy in Electronic Engineering
at The Chinese University of Hong Kong in August 2011.

The purpose of this thesis is to implement a Main Profile H.264/AVC Decoder, which supports Full HD (1920x1088p) resolution, by reusing a Baseline Profile Decoder, with a lower QCIF (176x144) resolution. The resolution performance has therefore been scaled up from a frame size of 176x144 to 1920x1088.

Since H.264/AVC has a higher coding efficiency, this technology is adopted in a wide range of applications, ranging from video streaming and storage to digital TV broadcast. Nowadays, display hardware commonly support high resolution like Full HD. High capacity storage makes storing video in Full HD format a common practice. Improved network speed has greatly boosted up user demand for higher quality video. In fact, H.264/AVC high coding efficiency gains from more complex computation compared to older codec, like H.263. Dedicate H.264/AVC decoding accelerators are usually added to systems to improve overall efficiency. Therefore, this thesis aims to design a Main Profile H.264/AVC Decoder that supports Full HD resolution.

For faster turnaround time and maintaining power efficiency, a Baseline Profile Decoder implementation was reused. The reused design features low power consumption, but it can only support a lower resolution.

This thesis proposes to design a CABAC decoder integrated on a Baseline Profile to form a standalone H.264/AVC decoder, so that it can support CAVLC/CABAC entropy coding while the resolution has been sized up to 1920x1088.

As the frame width has increased from 176 to 1920, line buffers storing upper line pel data and top neighbouring info are 10.9 times of the original size. In H.246/AVC, either one entropy coding mode, CAVLC or CABAC, would be used to decode a slice data, that is, CAVLC/CABAC would not be used at the same time. Also both modes of operations require line buffer resources. Line buffer for CAVLC and CABAC sharing scheme has therefore been proposed. This scheme saves 600 Byte memories, which is equivalent to 2.5% of the total local memory.

The reused design contains a macroblock-based processing architecture. The average cycle required for one macroblock in the new design is around the same as in the reused design. To satisfy the 82-times throughput requirement in Full HD resolution, the working frequency has to be increased to 82 times of its original. As a result, the timing path for one cycle must be short enough to meet the setup time constraint. Since the reused design working frequency is as low as 1.5MHz, those critical paths that do not exist in the reused design now appear in the new design. The main contribution of this thesis is to fix those setup time violations.

The proposed CABAC decoder adopts a three-stage pipeline architecture. For low hardware cost and power efficiency, a context model buffering scheme with two 28-bit buffers is proposed to allow read advance write actions into the context model memory. One buffer is waiting to write back to memory while another buffer is used to store new context model groups from the memory. This

thesis also proposes stall cycle reduction methods to further reduce 8.5% to 55.2% stall cycles due to context model switching.

The critical path of the proposed three-state pipeline architecture starts from the context model buffer, goes through the bin decoding and bin matching, and ends at the next context model selection. LUT method bin matching has a disadvantage of long timing path. An FSM method bin matching schemes is proposed to resolve the critical path issue and to satisfy the setup time requirement.

The proposed design aims for 90nm technology. At a working frequency of 143MHz, the proposed standalone CABAC can decode 72Mbin/s on average. The integrated H.264/AVC decoder can decode 1920 x1088 progressive frames at 36.2 frames per second.

摘要

本論文的目的是通過重用一個支持較低解像度（QCIF 176X144）的 Baseline Profile H.264/AVC 解碼器，以實現一個支持全高清（FullHD 1920x1088p）解像度 Main Profile H.264/AVC 解碼器。

由於 H.264/AVC 具有更高的編碼效率，這項技術已廣泛被採用，例如視頻串流、存儲、數字電視廣播等。現今，顯示器通常支持高分解像度，再加上高容量存儲硬件及網絡速度大大提高，高質量的視頻的需求不斷提升。事實上，H.264/AVC 編碼效率高，源於更複雜的運算。爲了以提高系統的整體效率，其中一方案是增添獨立的 H.264/AVC 解碼加速器。因此，本論文因此設計一個支持全高清解像度的 Main Profile H.264/AVC 解碼器。

爲了加快的開發周期及保證能源效率，本文重用了一個 Baseline Profile 解碼器。那設計具有已驗證之低功耗的特性，但是只支持較低的解像度。

本論文提出設計一個 CABAC 解碼器然後再集成到重用的 Baseline Profile 解碼器，這樣就能形成一個獨立的 H.264/AVC 解碼器。它將可以支持 CAVLC / CABAC 熵編碼，且支持到解像度最大爲 1920x1088。

H.246/AVC 只會選一個熵編碼模式，CAVLC 或 CABAC。也就是說，CAVLC/ CABAC 不會被用於在同一時間。本文提出 CAVLC 和 CABAC 共享緩衝區，如此就能節省 600Bytes 的記憶，這相當於 2.5%的總本地內存。

本文提出用於 CABAC 內的模型緩衝方法。當一個緩衝區等待剛使用的模型寫回內存時，另一個緩衝區就會用來接收從內存讀出的模型。本論文還提出減少停頓週期的方法，以進一步減少 8.5%至 55.2%的停頓週期。

爲求縮短一些關鍵路徑，本文另提出 CABAC 裡使內爲 FSM 式作配對 。

本文保存重用設計的基礎架構，如此，一個 macroblock 平均需要的時間大致和原來設計一樣。本來重用的工作頻率低至 1.5MHz，在全高清解像度下，吞吐量爲了原來的 82 倍，工作頻率快因此增加至 82 倍。結果，一個的週期時間大大縮短了。一些之前沒有違反 setup time 限制的路徑，現在違反 setup time 限制了。在本論文其中的貢獻是解決這些違反 setup time 限制路徑。

擬議的設計目的是爲 90nm 標準 CMOS 工藝下實現。設計工作頻率爲 143MHz，本文提出的 CABAC 解碼獨立運行時可達 72Mbin/s。集成的 H.264/AVC 解碼器最高支持解像度爲 1920 x1088 並每秒平均輸出 36.2 幀。

Acknowledgements

I would like to show my gratitude to my supervisor Professor CHOY Chiu Sing who has fully supported my research.

Thanks also go to Professor Alex, LEUNG Ka Nang.

I am pleased to thank ASIC laboratory technician, Mr. YEUNG Wing Yee, who has helped me a lot in networking and CAD tools matters.

I specially thank my mother and my family.

YIM, Ka Yee

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 : INTRODUCTION

1.1. Motivation

H.264/AVC was released in 2003[1]. In recent years, this codec is widely adopted, such as for video conferencing, broadcast, storage, streaming because of its high coding efficiency. Meanwhile, display hardware that support high resolution like Full HD are more common. High capacity storage makes storing video in Full HD format a general practice. Improved internet speed has greatly boosted up user demand for videos of higher quality and resolution.

In fact, H.264/AVC high coding efficiency gains from more complex computation compared to older codec, like H.263. Dedicated H.264/AVC decoding accelerators are usually added to high performance systems to improve efficiency. With these in mind, this thesis aims to design a Main Profile H.264/AVC Decoder supporting Full HD (1920x1088) resolution.

Power efficiency is important to every design in recent decades. There is always a trade-off between power and performance. Full HD codec capability requires working at higher clock frequency, and needs more gate count in design. Thus it consumes more power. Power efficient schemes in all aspects, including front-end and back-end designs, will be needed.

For faster turn-around in the scaling up development, this project re-uses an existing silicon-proven power efficient design from Xu [2], which was targeted for low resolution (QCIF 176x144) baseline profile, as a starting point. In the scaling up development, in general, data range and buffer size are increased. In this project,

the data range and buffer size are reviewed, and necessary changes are made to support Full HD resolution.

Within H.264 codec, there are two entropy coding mode options, either CAVLC or CABAC. CAVLC is less complex than CABAC but has lower coding efficiency. So CAVLC is usually used in low resolution video while CABAC is used in high resolution video. Since the reused design only support CAVLC mode, this project has to design a CABAC decoder and integrate it to the reused decoder to match the reality of high resolution application.

In summary, by reusing the existing design, this project aims to design a CABAC and integrate it to the baseline decoder. It targets a resolution of Full HD at a rate 30 frames per second.

1.2. Overview

The thesis will highlight the background information of H.264 in later sections in this chapter. Information closely related to this project will be emphasized. Chapter 2 provides the design of a CABAC and a CABAC standalone performance analysis. The reused design is reviewed in Chapter 3. It also gives the integration details and performance analysis. Chapter 4 is the conclusion.

1.3. H.264 Overview

H.264/AVC were published in 2003 [3]. It is a macroblock-based codec algorithm. Any frame size will be divided into macroblocks for processing. One macroblock will be decoded after going through the process described in section

1.31. For higher interoperability of video streams, profile and level are defined by the standard. The values of profile and level imply the decoder requirement on supporting features, for example, throughput performance. The details of profile and level are reviewed in section 1.3.2.

1.3.1. Macroblock-based Decoding Data Flow

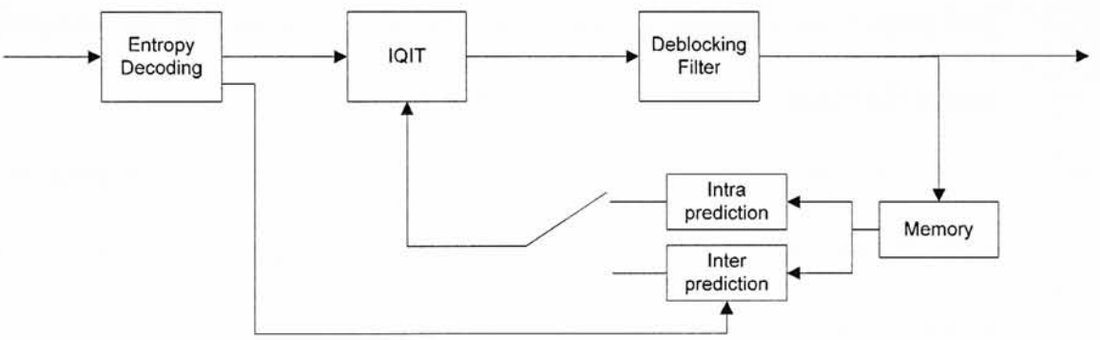

Figure 1-1: Decoder block diagram

A video sequence is a composite of frames. Each frame consists of luma (Y) and chrominance (UV) components to form a coloured frame. Since human eyes are less sensitive to UV components. UV components are sub-sampled in old codec standard to reduce the bandwidth of the chrominance. For example, in YUV420, the Y:U:V ratio is 4:1:1; in the same physical space, 4 Y samples will be captured while only one U and one V samples are captured. For a frame size of 1920x1088, the Y component becomes an array of 1920x1088, but U or V component is only an array of 960 x544.

The H.264 decoding is a macroblock-based process. Depending on the YUV sampling ratio, the macroblock size of Y and UV can be different. For YUV420, a macroblock of Y is 16x16; a UV macroblock size is 8x8. YUV components

are further divided into 24 4x4 blocks: 16 blocks of Y, 4 of U and 4 of V. The decoding sequence is to decode 16 Y blocks first, then the 4 U blocks, and lastly the 4 V blocks. All YUV 4x4 blocks share a common decoding logic. The variants are specified by corresponding syntax elements.

Figure 1-1 shows a decoder block of H.264. Entropy decoding includes universal variable length coding (UVLC), context-adaptive variable length coding (CAVLC), and context adaptive binary arithmetic coding (CABAC). The UVLC is used for decoding sequence, picture information, and intra- and inter-prediction syntax elements. The CAVLC is only used to transform coefficient syntax elements. The CABAC is used for intra-prediction, inter-prediction and to transform coefficients syntax elements. Either CAVLC or CABAC coding method will be used. The mode of selection is signalled from the sequence information. In the CAVLC mode, UVLC is used for intra- and inter-prediction of information.

After entropy decoding, all essential information will be ready for decoding of one macroblock. Intra- or Inter-prediction information is used for macroblock prediction. In intra-prediction, intra-prediction mode provides information that neighbouring pel data will be used. In inter-prediction, reference frame id and location of data are provided. Decoder will load the data from the frame buffer for prediction.

The prediction errors are recovered from transform coefficients. Sum of inverse transform and prediction will be the macroblock result.

In-loop deblocking-filter is used to remove block artifacts. The usage of deblocking-filter is signalled in the syntax element and edge detection on the

macroblock results. Final modified macroblock results will be stored in the frame buffer. The stored data would be used in intra-prediction by other macroblocks in the same frames or inter-prediction in other frames.

1.3.2. Differences of Main Profile and Baseline Profile

Features in only Baseline Profile	Common Tools in Baseline and Main Profile	Features only in Main Profile
FMO	CAVLC	CABAC
Red. Pictures	I & P slices	Weighted inter Prediction
ASO	Different Block Size	Field Coding
	1/4 Pel MC	MB-AFF
	Multiple Ref Frames	
	In-Loop Deblocking filter	
	Intra Prediction	

Table 1-1: H.264/AVC profiles and coding tools

H.264 has been developed for a wide range of applications, bit rates, resolution, qualities and services. Different applications have different functional requirements. To maximize the interoperability while limiting the complexity, profile and level together specify the decoder requirement. Profile defines the coding tools supported. Level defines the syntax values ranges or implicitly the bit-rate.

Table 1-1 shows the subset coding tools for the Baseline Profile and the Main Profile. The reused design supports CAVLC, I& P slices, different block size, 1/4 pel MC, single reference frame, in-loop deblocking filter and intra prediction. All these features are basic capabilities of a H.246 baseline decoder. The proposed design supports CABAC in addition to the original features.

Other features like FMO, Red. Pictures, ASO, Weighted Prediction, Field Coding and MB-AFF are not implemented in this thesis. The first three are targeted for noise reduction in network applications. The B slice and weighted inter prediction are generalized versions of P slice and non-weighted inter prediction. Field coding and MB-AFF are similar to frame coding. The difference is in the macroblock data representation. An estimation of the performance impact by these features can be based on the baseline profile, and it is easier than by the CABAC.

In this thesis, only CABAC is added to the baseline profile decoder to get the performance outlook. Detail of the CABAC proposed design is described in Chapter 2. The proposed integration design is described in Chapter 3.

1.3.3. CAVLC and CABAC

Both entropy coding, CAVLC and CABAC, are context adaptive. Syntax element values of neighbouring macroblocks, sub-macroblocks or transform blocks will affect decision making in the decoding process. Neighbouring info has to be stored up for later use in both modes. Specified by H.264 standard, either CAVLC or CABAC mode will be used to decode a slice.

The thesis proposes to share neighbouring info memory for CABAC and CAVLC. The hardware sharing detail will be described in Chapter 3.

CHAPTER 2 : CABAC

2.1. Introduction

This chapter describes the CABAC implementation design in detail. CABAC-related designs are reviewed in section 2.2. The CABAC algorithm is reviewed in section 2.3. The thesis proposal CABAC designed is described in section 2.4. The proposed design performance analysis is reported in section 2.5.

2.2. CABAC Decoder Implementation Review

In CABAC algorithm, the decision making process depends on prior decoded results. Since it is impossible to detect the syntax element boundary from the bitstream, it is difficult to increase parallelism in implementation. To improve its throughput, previous designs [4] [5] [6] [7] tend to optimize arithmetic decoding engines (AE) to decode multi-bins per cycle for specified syntax elements. Peng [5] proposed to decode 16 bins per cycle for one coefficient level. Yu [6] proposed to cascade two regular decoding engines and cascade four bypass decoding engines to get a multi-bin decoding engine. Jian-Wen [7] proposed to decode the second bin by a method of symbol prediction, that is by decoding MPS and LPS in parallel. These designs reported the utilization rate of first AE and second AE up to 90% and 43% respectively. As second or later bin arithmetic decoding engines are difficult to achieve more than 50% utilization rate, in order to save power, this thesis proposes to use a single bin arithmetic decoding engine. Although the throughput of the proposed design is not the highest among the related designs, it is good enough to decode a resolution of 1920x1088 at a rate of 30 frames per second in real time.

Previous designs attempted to solve the long delay caused by switching context models. Yu and He [4] loaded all context models of one syntax element to a context model register from memory. It [4] required 44x7 bit registers for context model registration as it has a large group size of 44 models, and 7 bits per model. Yi and Park [8] tried using models of smaller group sizes. It required 8x7 bit registers for context model registration. This design has solved the delay caused by model switching within a group but not between two groups. Yu [6] used two sets of registers. When one set serves for the decoding bin, it writes back the last group and pre-fetches the next group model with the highest probability to another set of register. He [6] did not mention the context model registration size.

AE needs to stall whenever the inputs are not ready. To increase the throughput, one way is to reduce the idle cycles of the AE, in which, a pipelined architecture can help. A previous design [8] has proposed a two-stage architecture. When the syntax element switches, two idle cycles are caused to load new context model groups from memory. Shi [9] proposed a four-stage architecture. Along with one memory for the entire context model and one memory for part of the context models, all next possible context models will be loaded before the arithmetic decoding stage. It removes all idle cycles introduced by the syntax element during switching in the pipelined architecture. However it needs to maintain two context model memories.

This thesis proposes a three-stage pipelined architecture with two identical 4x7 bits context model registers and one context model memory. With this scheme, there will only be one idle cycle when switching context model groups. As the switching will degrade the performance, the thesis proposes an idle cycle reduction

scheme. One is by context model grouping enhancement. The other one is by preloading the next context model. Furthermore, it is to skip loading from memory if the group is found in the register. With these proposed schemes, it can reduce stall cycles due to context model switching by 8.5% to 55.2%.

2.3. CABAC Algorithm Review

CABAC decoding is an iterative process. The process decodes bin by bin until the decoded bin string matches the pre-defined pattern. This section will review the CABAC algorithm. In reverse approach, the arithmetic decoding engine will first be described, followed by LPS range calculation and context model selection.

2.3.1. Arithmetic Decoding Engine

There are two modes of arithmetic decoding engines (AE): decision bin and bypass bin decoding. For decision bin decoding, the inputs to the AE are its current RANGE, current OFFSET, LPS_RANGE and MPS value. The current OFFSET is limited by the value from 0 to its current RANGE. (RANGE – LPS_RANGE) is the decision boundary of Most Probable Symbols (MPS) and Least Probable Symbols (LPS). If the current OFFSET is less than (RANGE – LPS_RANGE), the current decoding bin will be equal to the MPS value; otherwise, the decoding bin will be equal to the LPS value. Maximum value of RANGE and OFFSET is 0x1FF, 9-bit binary number.

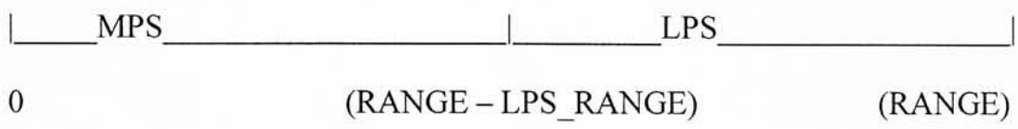

Figure 2-1: Decision bin decoding

If the current bin is MPS, the next RANGE value is set equal to the decision boundary value. The next RANGE value is set equal to the current RANGE.

If the decision boundary is less than 0x100, which is half of the maximum RANGE value, renormalization occurs. Renormalization is a process to prevent RANGE values from becoming a very small value after a number of decoding iterations. Both OFFSET and RANGE will be shifted to the left by one. Moreover, one bit from the bitstream data is shifted into OFFSET. Left shift is iterated until the next RANGE value is equal or larger than 0x100.

If the current bin is LPS, the next range value is set equal to the LPS range. The next offset value is set equal to (current offset – the decision boundary). Since the LPS range must be less than 0x100, renormalization must occur.

Figure 2-2 illustrates the decision bin decoding process for MPS bin and LPS bin.

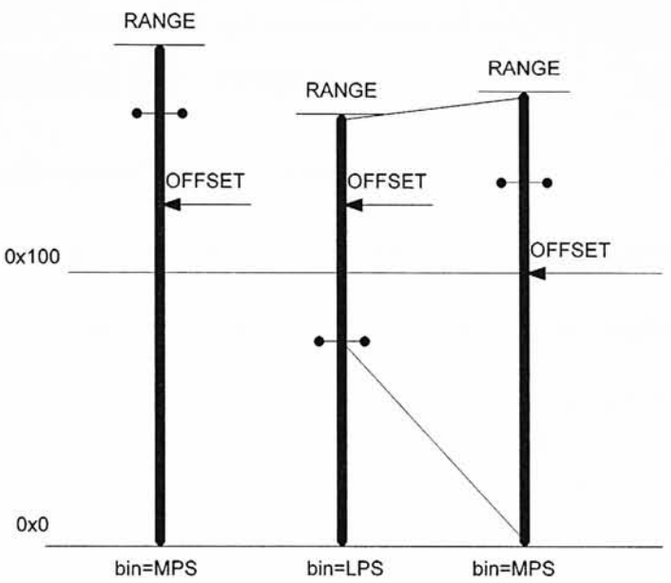

Figure 2-2: MPS bin and LPS bin

For bypass bin decoding, only the current OFFSET and the current RANGE are needed. The decision boundary is equal to half of the RANGE.

Similarly, if the current OFFSET is less than (1/2 RANGE), the current decoding bin is equal to 1; otherwise it is equal to 0.

|_____0_____|_____1_____|

0 (1/2 RANGE) (RANGE)

Figure 2-3: Bypass decoding

If the decoding bin is 0, the next OFFSET will be set equal to the current OFFSET; otherwise, the next OFFSET will be set equal to (OFFSET- ½ RANGE). The next RANGE is always set equal to ½ the current RANGE and renormalization must occur for both bin results.

2.3.2. LPS Range Generation

The LPS_RANGE is not a fixed value in CABAC algorithm. It is derived from the context model and the current RANGE. It changes when the context model changes or when the context model is updated. The LPS_RANGE is obtained by LUT method. Inputs to the LUT are its current RANGE [8:7][1] and PSTATE which is a context model variable representing probability when MPS occurs. PSTATE ranges from 0 to 63. The output is an approximate value of RANGE multiplied by the probability of LPS. Based on the decoding results, the decoded bin is the MPS. The next PSTATE is determined by TRAN_MPS_LUT; otherwise, by TRAN_LPS_LUT. If the current PSTATE is 0 and the decoded bin is LPS, the next MPS value will be set equal to the LPS value. The next PSTATE and MPS values will be used to update the context model. When the next same context model is used, the latest PSTATE and MPS will be used.

2.3.3. Context Model Selection

In CABAC, each syntax element has a set of context models to store the statistics of bin occurrence. Each model has two variables, PSTATE and MPS. PSTATE represents the probability of the MPS symbol to occur. MPS is the value of the MPS symbol. These two variables are the inputs to decision bin decoding. The context model selection process depends on the syntax element type, prior to the decoded syntax element values, to the decoded bins and to the current decoding bin index.

---

[1] RANGE[8:7] = (RANGE>>6) & 0x3

Before decoding the first bin, Context Models, OFFSET and RANGE have to be initialized. The RANGE is simply set to 0x1FE. OFFSET is shifted 9 bit data to the left from the bit stream. Context models require a LUT method to get init variables (m,n). PSTATE and MPS values are derived from (m,n) and SliceQP[2].

2.4. Proposed CABAC Decoder Implementation

Without loading all models for the decoding syntax element, this scheme loads only 4 models. So the context model registers can be reduced to two 4x7-bit register sets. When one register set is being used by the AE, the other register set data will be written back to memory or preloaded to the next context model group. It reduces the idle cycles caused by switching context model groups. To reduce the critical path, that connects through the context model registers, arithmetic decoding engines, bin matching stages, and the next context model group results, FSM bin matching method is used.

2.4.1. Pipelined Architecture

This thesis proposes a three-stage pipeline architecture. The first stage is a LOAD_MEM; the second is a CTXIDC; and the third is a DEC/MATCH. In this section, it will first describe each stage in detail followed by the example of context model group switching.

a. LOAD_MEM

---

[2] SliceQP is a syntax element.

In this stage, one context model group is loaded from memory to a context model register set (CMRS). This stage is executed when a context model group switching is needed.

b. CTXINC

In this stage, it determines which context model in a group will be used. Model selection depends on syntax element types, bin indices (binIdx), previous decoded bins, and neighbouring syntax element values.

In the proposed design, all required neighbouring information for current macroblocks is stored in an nA-nB info buffer. The buffer has two partitions, one for nA and one for nB. The nA info will be updated once the syntax element is decoded. The nB info is loaded from the nB info memory. This nB info memory consists of 67 bits x 120 and 9 bits x120 single-port on-chip SRAMs. At the end of the macroblocks, info will be used as neighbouring info written to the nB memory.

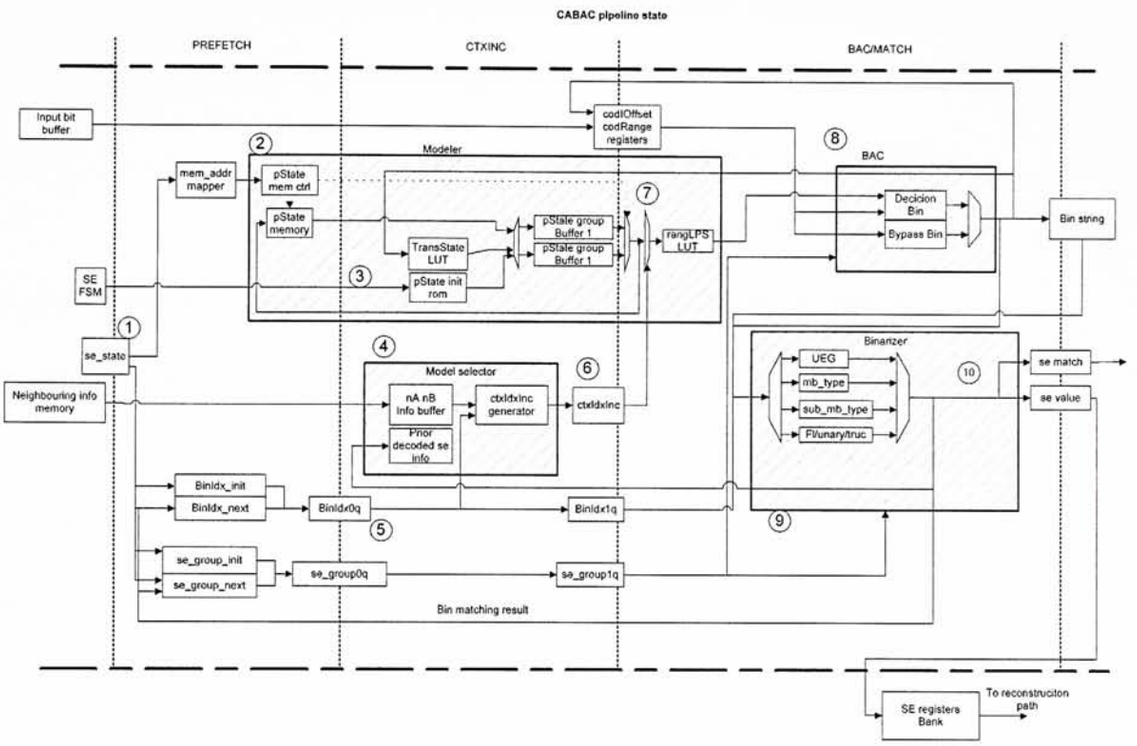

Figure 2-4 : Pipelined Architecture of proposed CABAC decoder

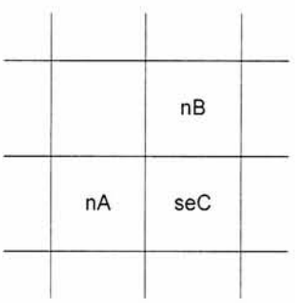

Figure 2-5 : top neighbouring (nA) and left neighbouring (nB) of current syntax element (seC)

c. DEC

This stage performs arithmetic decoding. For decision bin decoding, model variable PSTATE is used to generate an LPS_RANGE. The current bin is decoded by checking whether the OFFSET is less than (RANGE –

LPS_RANGE). Based on the decoded result, RANGE, OFFSET, PSTATE, MPS are updated.

d. MATCH

In this stage, decoded bin is matched to the syntax values. The decoding complete condition of current syntax element is detected at this stage.

2.4.2. Context Model Group Switching and Enhancements

The proposed pipelined architecture need to stall one cycle to load context model groups from memory to a CMRS. High switching rate results in more stall cycles; thus a lower throughput. Three methods have been proposed to reduce the idle cycle caused by such switching.

The first two methods aim to handle two known back and forth switching cases, prev_intra4x4_pred_mode – rem_intra4x4_pred_mode and significant_coeff_flag – last_significant_coeff_flag. For the former pair, this thesis proposes to group all models of these pairs into one single group so that no group switching will be needed. prev_intra4x4_pred_mode and rem_intra4x4_pred_mode have one context model each. Since one context model group can have four models in maximum, there is no problem to group them together.

For the later pair, discovered from the decoding sequences, significant_coeff_flag must follow the last_significant_coeff_flag. This thesis proposes to preload the context model group for the last_significant_coeff_flag in its next cycle when the group for significant_coeff_flag is loaded.

The first two methods are for special cases. The conditions are easy to detect.

For general cases, this thesis proposes a detection logic to check whether the context model group is previously loaded to the context model registers. If a context model group is found, the ready flag will be asserted and no stall cycles will occur in this case. This detection can reduce stall cycles due to back and forth switching between context model groups of any syntax elements.

With the proposed detection, it would not overwrite contents in context model register sets until it has determined the content is not useful.

Figure 2-6 shows the timing diagram of two syntax elements decoding. se0 represents the first syntax element with context model group ctx_group_0. It has three bins. se1 represents the second syntax element with context model group ctx_group_0. The figure also shows the content of two context model register sets (CMRS).

At clk1, ctx_group_0 is loaded to CMRS_0. At clk2, context model is selected. At clk3, se0_bin0 is decoded. Two cycles are stalled at the beginning of each decoding. As se0_bin0 does not match with any possible syntax element value, the context model for se0_bin1 is selected in the same cycle as DEC/MATCH. se0_bin1 goes through a similar process.

At clk5, se0_bin2 is decoded, the decoded bins finally match. The next context model group ctx_group_1 is loaded to CMRS_1.

At clk6, ctx_group_0 is written back to memory.

At clk8, se1 decoded bins match. With the detection logic, next context model group ctx_grp_0 will not be loaded from memory as it has been loaded to CMRS_0 already. No stall cycles occur for this case.

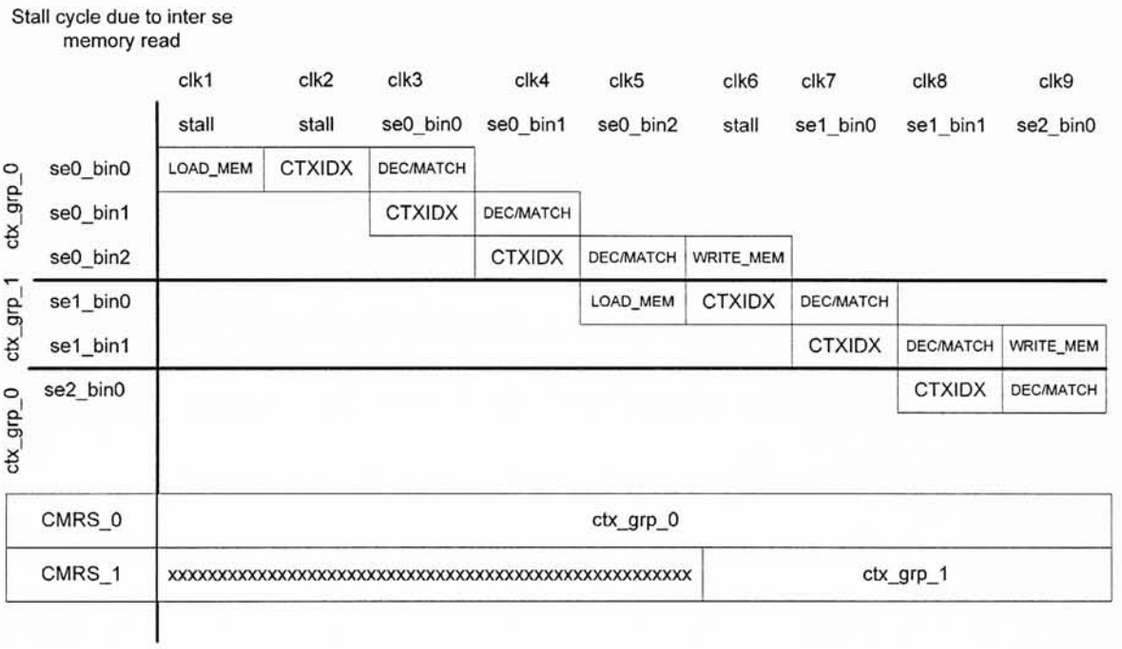

Figure 2-6 : Timing diagram of the proposed pipelined architecture

2.4.3. Context Model Grouping

Except for syntax element coeff_abs_level_minus1 that has five context models, all other elements only have a maximum of four context models for each decision bin decoding. Useful four context models should be loaded from context model memory. In H.264, context model is addressed by ctxIdx which is equal to the sum of ctxIdxOffset and ctxIdxInc. Each syntax element type has a unique ctxIdxOffset. The range of ctxIdxInc is different for different syntax elements. For example, mb_type for I slice ctxIdxInc can be from 0 to 7. prev_intra4x4_pred_mode and rem_intra4x4_pred_mode can only be 0. Except for prev_intra4x4_pred_mode, rem_intra4x4_pred_mode and mb_type_prefix and mb_type_suffix for P slice, the context model grouping is based on ctxIdxInc. For valid ctxIdxInc, four ctxIdxInc ranges, 0 to 3, 4 to 7, 8 to 11, 12 to 15, form the four different groups.

prev_intra4x4_pred_mode and rem_intra4x4_pred_mode only have one context model each. As mentioned before, they frequently switch to each other. These two context models are grouped in the same group to reduce group switching.

The same context model is addressed by mb_type_prefix (ctxIdxInc=3) and mb_type_suffix (ctxIdxInc=0) for P slice. This context model can be either grouped to mb_type_prefix or mb_type_suffix but not both.

With the proposed grouping scheme, 399 context models are grouped into 112 groups. The required context memory size is 28 bit x 112, with a total of 3136 bits.

2.4.4. Initialization of Context Models

The organization of context model init ROM is the same as the context model grouping, which is four in a group. There are four init tables, each of which specifies the values of its init variables (m, n).

Each ROM 16-bit entry stores a pair of context model variables (m, n) of 8 bits each. The first group is written to a ROM address between 0x00 and 0x03. The second group is written from address 0x04 to 0x07 and so on. The required ROM size is 16bits x 4 x 112 x 4, in a total of 28,672 bits.

When initialization starts, the context model ROM address counter resets. A pair of (m, n) is read out each time. The address counter is incremental by one. The initialization results of PSTATE and MPS will be stored in one of the context register sets first. After all context model groups, four context models, are initialized, the result will be written to the context model memory for later

use. At the same time, the next context model group initial result will be written to other unoccupied or free context model register sets.

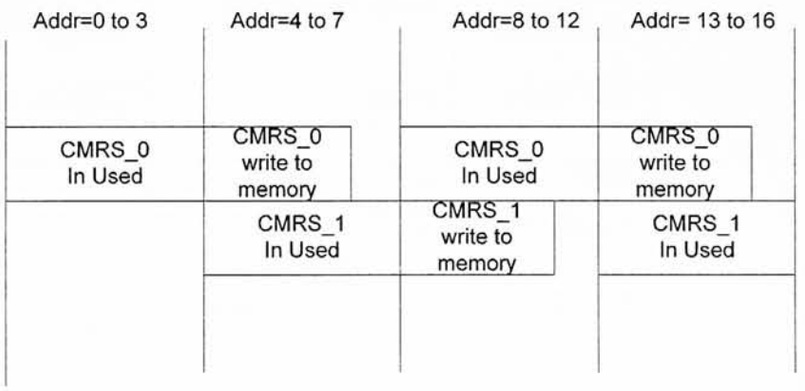

Figure 2-7 : Initialization of context models Rom address from 0 to 16

2.5. FSM Method Bin Matching

Table 2-1 shows some examples of syntax element values and their corresponding bin strings.

There are two categories: binary and non-binary. Syntax elements in Binary categories are 1-bit flags. The bin matching process is simple for this category.

In non-binary categories, syntax elements have bin strings of more than one bin.

Binarization is a process converting syntax values to bin strings. Matching process is an inverse process of binarization. There are four types of binarization: table mapping, unary, truncated unary, fixed length and concatenated unary/ k-th order Exp-golomb (UEGk). The completion of the matching process is simply signalled by binVal or bin count. For table mapping and UEGk, it depends on the

prior decoded bin. In our proposed design, separate FSMs are designed for table mapping and UEGk syntax values matching.

Category	Type of binarization	Syntax element (SE)	Example of bin string	
			SE value	Bin string ($b_0b_1b_2...b_n$)
Non-binary	Table Mapping	mb_type	I slice, mb_type =0	0
		sub_mb_type	P slice, sub_mb_type =3	010
	Unary	ref_idx_l0, ref_idx_l1	0	0
		mb_qp_delta	2	110
	Truncated Unary	intra_chroma_pred_mode	1	10
		coded_block_pattern_suffixed	2	11
	Fixed Length	rem_intra4x4_pred_mode	3	111
		coded_block_pattern_prefix	15	1111
	Concatenated unary/ k-th order Exp-Golomb	mvd_l0, mvd_l1	10	9'{1}, 001
		coeff_abs_level_minus1	23	14'{1},1110001
Binary		mb_skip_flag	1	1
		mb_field_decoding_flag	1	1
		prev_intra4x4_pred_mode_flag	1	1
		coded_block_flag	1	1
		significant_coeff_flag	0	0
		last_significant_coeff_flag	0	0
		coeff_sign_flag	0	0
		end_of_slice_flag	0	0

Table 2-1 : Example of Syntax element and type of binarization

For tabling mapping syntax element, there are two ways to implement it. First it is by using LUT method; second it is by using FSM method. In LUT method, a prior decoded bin is stored in the bin string register. The current bin and bin string register form a search bin string, if the search bin string matches with the LUT. When the decoding process is done, the syntax value is decoded. Otherwise, the current bin will be shifted to the bin string register. And the matching process will need to be redone for the next bin. This method results in a longer combinational path.

There is a critical timing path from CMRS_0/CMRS_1 register to the next context model group which passes through a match complete signal. It is necessary

to keep all combinational paths as short as possible; thus leading to the FSM method. By FSM method, the combinational path is shorter. This can help to shorten the critical path. Fig 2-8 shows the FSM of mb_type for I slice and P slice.

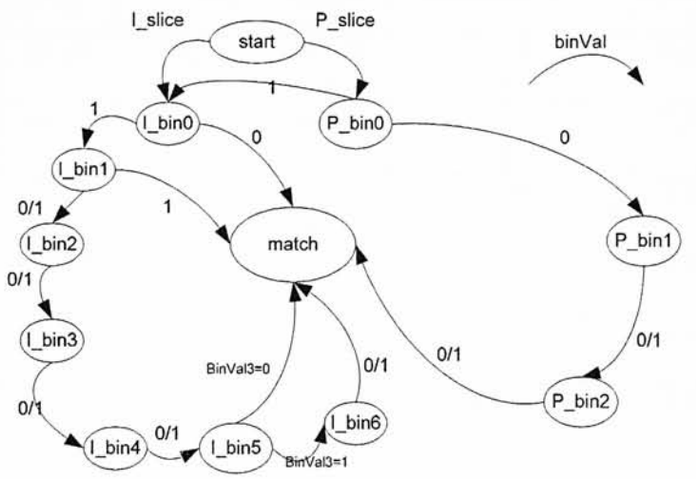

Figure 2-8 : FSM of CABAC mb_type bin matching

For example, there are 25 possible bit string patterns for mb_type (I Slice).

With FSM method, the decode completion is the value of one bin data only, the current binVal or the one prior. If the state encoded with one –hot encoding, the match flag logic is an OR gate with 5 inputs, and 5 2-input AND gates. The timing path is much smaller than then LUT method.

2.6. CABAC Experimental Results

The proposed CABAC decoder is implemented using UMC 9nm 9-metal-layer technology. The CABAC decoder is part of the standalone Main Profile decoder. The standalone decoder works at 143MHz. This CABAC decoder uses a two-port on-chip 112 x28 bit SRAM as Context Memory, and the total logic equivalent gate count for CABAC is 40,373. According to the simulation result, the

average throughput rate of the proposed CABAC decoder is 0.51bit/cycle. At working frequency of 143MHz, the proposed decoder achieves 72.42Mbit/s.

Test sequence name	Bitstream bit rate @30 frame (Mbit/s)	bit rate (I frame/ P frame) (kbit /frame)	Stall cycle saved (I frame / P frame) (kbit /frame)	Bin /Cycle	MBin/s	Average Decoding time for one intra period (ms)	Cycle /bin
Station	1.9	89.7/ 3.5	26.2%/ 8.5%	0.45	63.88	43.79	2.24
sunflower	2.7	75.0/ 6.8	28.0%/ 16.3%	0.45	63.88	43.79	2.24
rush_hour	3.1	49.5/ 12.9	22.7%/ 16.8%	0.51	73.57	71.36	1.94
blue_sky	3.9	118.4/ 11.8	44.9%/ 11.9%	0.46	66.27	68.52	2.17
pedestrian _area	4.4	61.7/ 15.4	26.8%/ 17.8%	0.51	73.27	87.2	1.95
Tractor	8.8	160.8/ 25.6	33.9% 19.1%	0.53	75.24	142.65	1.90
Riverbed	22.2	148.5/ 90.4	33.1%/ 25.%	0.57	81.64	369.8	1.75
Park_joy	25.9	247.9/ 64.6	55.2%/ 33.0%	0.57	81.64	244.4	1.75
(Average)				0.51	72.42		1.99

Table 2-2 : Performance of standalone CABAC

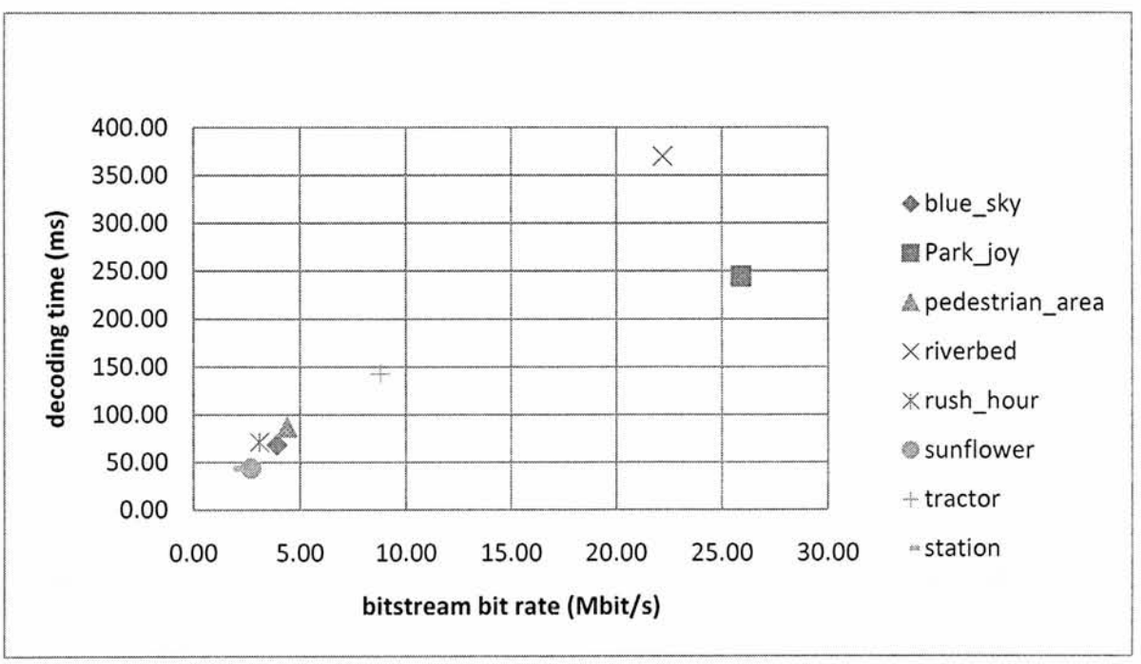

Figure 2-9 : Decoding time vs. bitstream bit rate of different bitstreams

Table 2-3 shows the simulation results of the proposed CABAC decoder. The test sequences have a frame size of 1920x1080. They are encoded with setting QP=29, Main Profile, Level 4.0, Intra Period=30 (1 I frame and 29 P frame every second), sampling rate =30fps.

I frame is a frame which only contains intra-prediction macroblocks. P frame is a frame which can contain intra- and inter-prediction macroblocks. Sampling rate represents the number of frame sampled during shooting. Intra Period represents the interval between two I frames during encoding. The number of I frame is larger if Intra Period is smaller. Since I frame bit rate is larger than that of P frame, smaller Intra Period will increase bitstream bit rate. The simulation bitstream bit rates ranges from 1.9 Mbit/s to 25.9 Mbit/s.

Figure 2-9 shows the decoding time for one Intra Period against the bitstream bit rate of different bitstreams in simulation. It shows the decoding time is longer for bitstream of higher bit rate in general. However, the decoding time of the "park_joy" case does not increase with the bit rate because the proposed stall reduction technique is working efficiently in this case.

From simulation results, 8.5 % to 55.2 % of stall cycles are saved. In the "park_joy" case, 55.2% and 33.0 % of stall cycles have been saved in I frame and P frame respectively. The result shows the proposed stall reduction technique has reduced the decoding time efficiently.

In addition, from simulation results, the proposed CABAC average decoding time for one Intra Period is within a second. It shows the proposed CABAC can decode Full HD bitstream in real time.

Design	Yu [4]	H.Yu [6]	C.Jian [7]	Yi [8]	B.Shi [9]	Proposed Design
Year	2005	2009	2009	2007	2008	2011
Technology	0.18	0.13	0.13	0.18	0.18	0.09
Gate Counts	n/a	47,081	43.6 K	81,162	28,956	40,373
Memory (Bytes)	420	434	1033	12.18K	10.81K	392 (context) + 1140 (nB info)
Maximum Frequency (MHz)	150	333	225	238	200	143
Resolution	720x480 @30fps	4000x2000 @30fps	4x1080HD @30fps	1920x1080 @25fps	HD1080i	1920x 1088 @30 frames
Data Rate (Bin/cycle)	N/A	1.08	1.32	0.254	1.27	0.5
Date Rate (cycle/Bin)	N/A	0.925	0.75	3.93	0.78	2
Throughput (Mbin/s)	N/A	360	314	57	254	72

Table 2-3 : Comparison of proposed CABAC and other design

Table 2-4 shows the comparison of the proposed CABAC and other designs.

The designs of Yi and B. Shi contain pipelined architectures. Two stall cycles due to switching context model group and single bin AE causes Yi to need 3.93 cycles to decode one bin. B. Shi does not have stall cycles. It generates 1bin/cycle for decision bin and 2bin/cycle for bypass bin. The average data rate of the B. Shi design is 1.27 bin/cycles, which is more than one bin per cycle. The statistical result of the B. Shi design performance actually is limited by statistics of the decision bin to bypass bin ratio. If more bypass bin occurs, the performance is closer to 2bin/cycle; otherwise, it is more close to 1bin/cycle. The average data rate of the proposed design is 0.5bin /cycle or 2 cycle/bin. Compare to Yi's design, the proposed design has two cycles reduction. One cycle is due to two context model register sets. The other cycle is due to the proposed stall cycle reduction scheme.

The proposed design cannot achieve more than one bin per cycle because it only has one-bin AE and some cycles are idle for the control.

Although the designs of H. Yu[6] and C. Jian [7] do not have pipelined architectures but they use two–bin AE. They achieve more than one bin per cycle in data rate. C. Jian's design has two-bin AE. It can perform one decision bin, one bypass bin, two decision bins, two bypass bins or one decision one bypass bins decoding. It is the most efficient AE among the designs. As H. Yu can work at high frequency, even though it does not have the highest data rate, it has the highest throughput at 360Mbin/s among other designs.

2.7. Summary

Long timing paths can be broken into several stages to meet the timing requirement. The throughput arithmetic decoding engine can be formulated as working frequency / no. of cycle per bin. With a fixed frequency, reducing the number of cycles per bin can increase the throughput. For a single bin arithmetic decoding engine, reducing the idle cycle in an arithmetic decoding engine is the only way to reduce the number of cycles per bin. The proposed design uses a pipelined architecture to reduce idle cycle. Two 4x7 bit context model register sets have been proposed to reduce 9 to 34 % stall cycles caused by context model memory access. The proposed CABAC design can decode Full HD video in real time.

CHAPTER 3 : INTEGRATION

3.1. Introduction

This thesis aims to develop a Main Profile Decoder supporting Full HD resolution. To shorten the development time, this thesis proposes to reuse a Baseline Decoder design for low resolution. Since the reused design does not support CABAC, the proposed CABAC needs to be integrated. To support Full HD resolution, the neighbouring info memory size in the reused design has to be increased, so that it can support the data range of the syntax element. As the proposed decoder will work at a higher frequency, some combinational paths that did not violate timing constraints in the original design now violate the timing constraints imposed by the proposed integration design.

This chapter will first review the reused design. Integration with CABAC and memory size change will be described. Later, new violation timing paths in the integrated design and proposed solutions will be described. Finally, it summarizes the proposed design performance.

3.2. Reused Baseline Decoder Review

The reused design, from Xu [2], was published in 2007. It is silicon-proven and has a very low power consumption. The design consists of two functional blocks, bitstream parser and reconstruction. The function of the bitstream parser is to decode bitstream to syntax elements values. It consisted of syntax element FSM

and entropy decoders, UVLC and CAVLC. Syntax element FSM controls the syntax element decoding sequence.

The function of reconstruction is to compute the video frame from syntax elements. It consists of an inverse transform block, an intra-prediction block, an inter-prediction block, a summation block and a deblocking filter block. A pipelined architecture in reconstruction allows data reuse, and reduces main memory access which results in improved power efficiency. Figure 3-1 is the architecture of the reused design. Two functional blocks are separated by syntax element registers.

When bitstream parser decodes one syntax element, the result is stored in the syntax element registers. As we have reviewed in Chapter 1, a macroblock is divided into 4x4 blocks. The inter- or intra-prediction will start after bitstream parser decodes all macroblock layer syntax elements until its residual and one 4x4 block prediction result is generated. IQIT will start after decoding residual syntax elements for 4x4 blocks. The output frame data is the summation of prediction and IQIT. The output is written to the output frame memory or written to the deblocking filter memory if a deblocking filtering is needed. This process iterates through all 4x4 blocks in the same macroblock. After one macroblock is decoded and reconstructed, bitstream parser will decode syntax elements for another macroblock. The reconstruction process does the same thing in every macroblock. When all macroblocks are reconstructed, the whole frame is reconstructed.

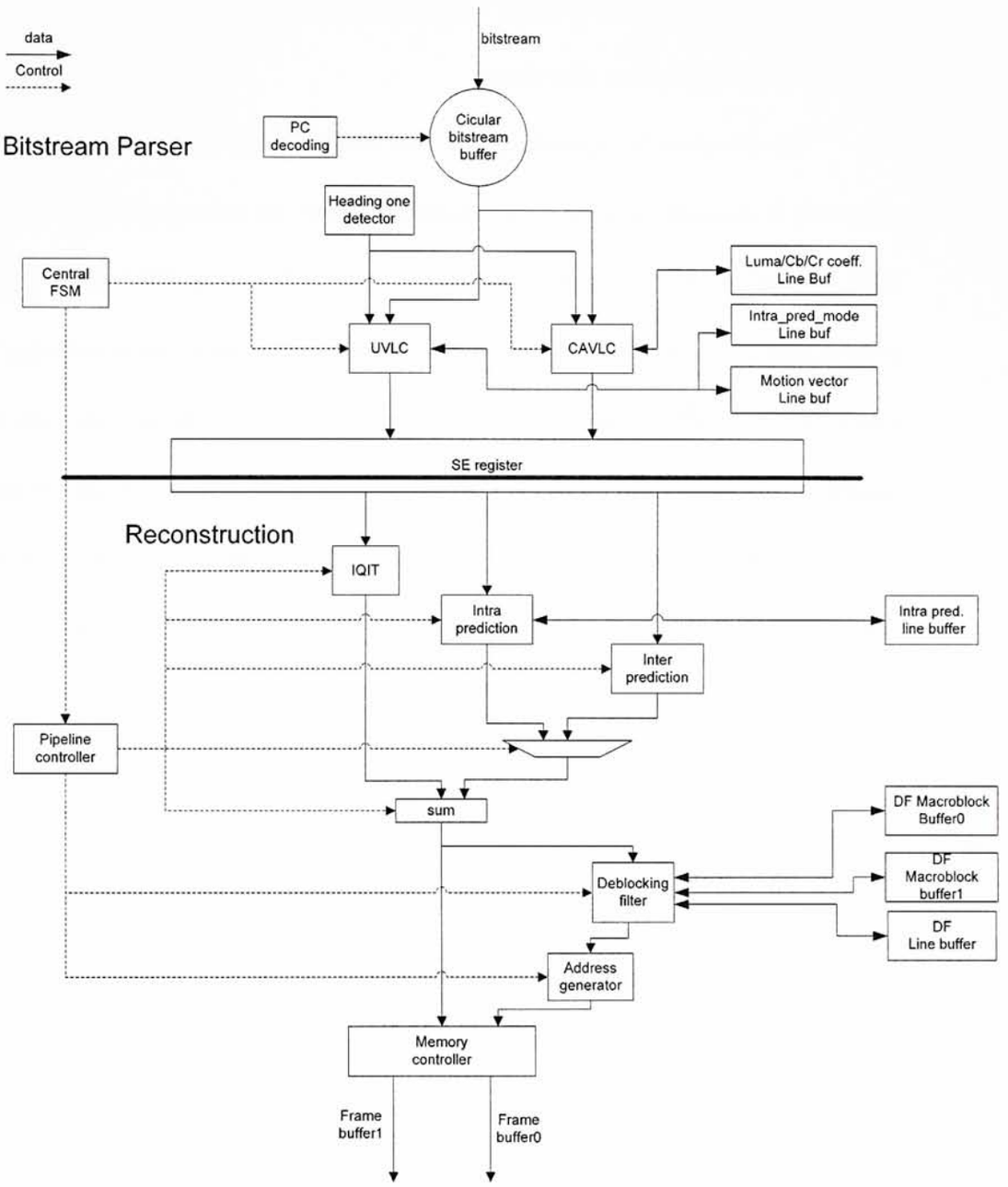

Figure 3-1 : Architecture of the reused design

Xu's design was designed for 0.18um technology. The reconstruction path can run at a maximum frequency of 200MHz. The working frequency is 1.5MHz. Its throughput is QCIFx30fps. The power consumption is 293uW@ 1.0V or 973uW@1.8V. Table 3-1 is the summary of the reused design.

Technology	UMC 0.18um CMOS IP6M	
Supply Voltage	1.8V core, 3.3 I/O	
Size	3.8x3.8mm2 core	
Design Cost	Logic Gates	169K (in NAND2)
	Memory	2.5k Byte SRAM
Operating Frequency	1.5MHz for QCIF @30fps	
Power Rating	293uW @ 1.0V 973uW @ 1.8V	
Reconstruction Path Max Freq	200MHz	
Max Throughput of Reconstruction Path	48.53 Full HD @ Max freq	

Table 3-1 : Summary of reused design, Xu design

3.3. Integration

3.3.1. CABAC

To support CABAC, the proposed CABAC is integrated into the reused design.

CABAC becomes part of Bitstream parser. The Bitstream buffer inputs to the CABAC, and the CABAC outputs to the syntax element registers. States related to CABAC decoding are added to central FSM. If the current decoding syntax element is a CABAC syntax element, CABAC will be triggered. Since either CAVLC or CABAC works at one time, Luma/Cb/Cr coefficient Line Buffer is replaced by Common Info Line Buffer. Common Info Line Buffer is shared by CAVLC and CABAC.

3.3.2. Memory Change In Size

The supporting resolution changes from QCIF (176x144) to Full HD (1920x1088). The new frame width becomes 10.9 times wider than the old one. All line buffers that store the upper pel data and upper neighbouring info have to be scaled up linearly. In addition, motion vectors, mvx and mvy, bit width increases from 8 bits to 14 bits. The motion vector memory is more than 18.9 times of the reused design.

RAM/RF	Function	Previous Size	New Size	No. of times
Intra4x4_PredMode	Intra4x4 pred mode decoding	16bitx11	16bitx120	10.9
LumaLevel_mbAddrB	CAVLC Luma coefficient level decoding	20bitx11	20bitx120	10.9
ChromaLevel_Cb_mbAddrB	CAVLC CB coefficient level decoding	10bitx11	10bitx120	10.9
ChromaLevel_Cr_mbAddrB	CAVLC CR coefficient level decoding	10bitx11	10bitx120	10.9
mvx_mbAddrB	Motion vector decoding	32bitx11	56bitx120	19.1
mvx_mbAddrC	Motion vector decoding	8bitx11	14bitx119	18.9
mvy_mbAddrB	Motion vector decoding	32 bitx10	56bitx120	19.1
mvy_mbAddrC	Motion vector decoding	8 bitx10	14bitx119	18.9
DF_mbAddrA	Deblocking filter	32 x32bit	unchanged	1
Intra_mbAddrB_RAM	Intra prediction	32bit x88	32bitsx960	10.9
DF_mbAddrB_RAM	Deblocking filter	32bitx352	32bitsx3840	10.9
rec_DF_RAM0	Deblocking filter	32bitx96	unchanged	1
rec_DF_RAM1	Deblocking filter	32bitx96	unchanged	1
mb_type_mbAddrB	SE decoding	2 bitx11	2bitx120	10.9
bs_coefficient_mbAddrB	Deblocking filter	4bitx11	4bitx120	10.9
common_info_memory_data	CAVLC/CABAC	N/A	67 bit x120	N/A
common_info_memory_type	CAVLC/CABAC	N/A	9 bit x120	N/A
Context memory	CABAC	N/A	28bitx 112	N/A

Table 3-2 : Memory in the integrated design

CABAC requires 67 bits x120 memory for neighbouring info storage. It is not in used for non-CABAC mode. Therefore, this thesis proposes to share this memory with CAVLC Luma/Cb/Cr coefficient level decoding. LumaLevel_mbAddrB, ChromaLevel_Cb_mbAddrB, ChromaLevel_Cr_mbAddrB are packed to form 60 bit data and is written to common_info_memory_data.

mb_type_mbAddrB and bs_coefficient_mbAddrB are originally 22 bits and 44 bits registers respectively but now it becomes 240 bits and 480 bits in size. The thesis proposes mb_type_mbAddrB, bs_coefficient_mbAddrB and 3 bit CABAC info data to be written onto a memory common_info_memory_type.

In summary, the total local memory is 23.5Kbyte. The sharing scheme can save 600 Bytes memory which is 2.5 % of the total memory.

3.3.3. Data Range of Motion Vector

Besides the change of buffer size, increase in supported resolution also affects the data range of motion vector. Motion vector is used during inter prediction. It has two components and they describe location displacement in x and y directions. Motion vector component is a signed number with a 2-bit fractional part and an N-bit integral part where N is derived from maximum value. For QCIF, the maximum integral part is 176 so N is equal to 8. For Full HD, the maximum integral part is 1920 so N is equal to 11. If CABAC mode is used, motion vector is decoded by CABAC; otherwise it is decoded by Exp-Golomb decoder in UVLC. The proposed CABAC can support new motion vector data range but not Exp-Golomb decoder in the reused design. Since Exp-Golomb decoder involved heading one vector, by simply increasing the data

width can solve the data range problem but timing violations will happen. This thesis proposes a line bitstream buffer to replace circular bitstream buffers, and changes one-cycle process to two-cycle process. Details of motion vector decoding and timing violation will be described in section 3.4.

3.3.4. Address Generator

Frame memory address generator is also modified to fix the time violations in synthesis. To shorten the timing path, intermediate values like yoffset, xoffset are calculated one cycle before use.

Figure 3-2 is the proposed integrated design. Blocks in white are new design. Blocks in grey are unchanged design. Blocks with dots are modified units.

3.4. Proposed Solution for Motion Vector Decoding

3.4.1. Motion Vector Decoding in the Reused Design

In non-CABAC mode, motion vector is decoded in UVLC. The codeword for motion vector is in signed Exp-Golomb format. The syntax element value is mapped to codeNum. The codeNum is the number represented by the codeword. The codeword has three parts. The first part is N leading zero. The second part is one "1". The third part is N bit binary number, binaryN, where N is equal to the floor integer of log2 (codeNum+1). The total length is 2N+1 bits. Table 3-3 is an example of codeNum, codeword, syntax element value.

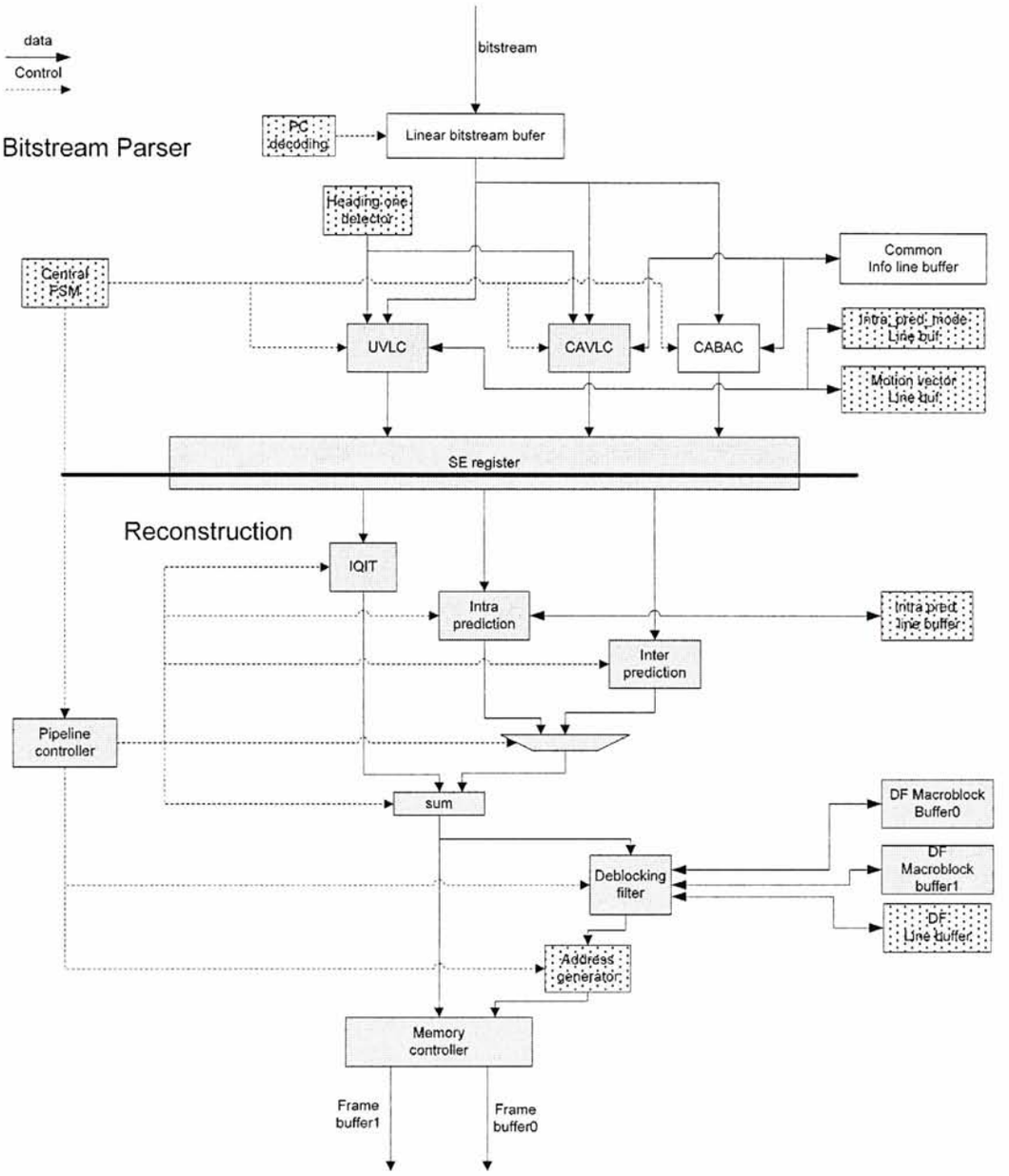

Figure 3-2 : Architecture of the proposed integrated design

codeword	codeNum	Syntax element value
1	0	0
010	1	1
011	2	-1
00100	3	2
00101	4	-2
00110	5	3
00111	6	-3
{{N-leading "0"},1,{N-bit binary}}	{1,{N-bit binary}} -1	
	k	(-1)^k+1 ceil(k/2)

Table 3-3 : codeword and codeNum of Exp-Golomb

In Full HD resolution, the largest codeNum of motion vector has 14 bits. The codeword has 13 leading zeros, one "1" and 13 bit binaries. There are 27 bits in total.

In the reused design, Exp-Golomb decoder decodes the codeword in one cycle but it only supports codeword up to 16 bits. The limitation factor is the 16 bits output of the bitstream buffer. Increasing the width of the bitstream data output can remove the limitation but it requires increasing the number of MUX in the bitstream buffer.

In the reused design, the bitstream buffer was a 128 bit circular buffer. There were 16 x 128-to-1 MUX in the output. To support codeword of 27 bits, the number of 128-to-1 MUX has to be at least 27. From the synthesis result, there are serious timing violations at the bitstream buffer data output. The violation paths passed through the bitstream buffer pointer and Exp-Golomb decoder.

To resolve the timing problem, this thesis has proposed a 48-bit line bitstream buffer to replace the circular bitstream buffer. There are two stages in the line buffer. The first stage has 16 bits shift registers. The second stage has 32 bits registers. Figure 3-3 shows the proposed line bitstream buffer. Bitstream data is read from memory and fills up the 32-bit register. The second stage register will then be filled by the first stage register. The 16 bits upper part of the bitstream data output come from the 16-bit register. The 16 bits lower part is selected from the 32-bit register. When one bit is consumed, one bit from the 32-bit register is shifted into the 16-bit register. When 32-bit register contains less than 16 valid bits, one entry from memory is read and shift into the 32-bit register. The proposed bitstream buffer has less registers and MUX. With the proposed bitstream buffer many timing violations related to bitstream buffer output and bitstream buffer pointer are resolved.

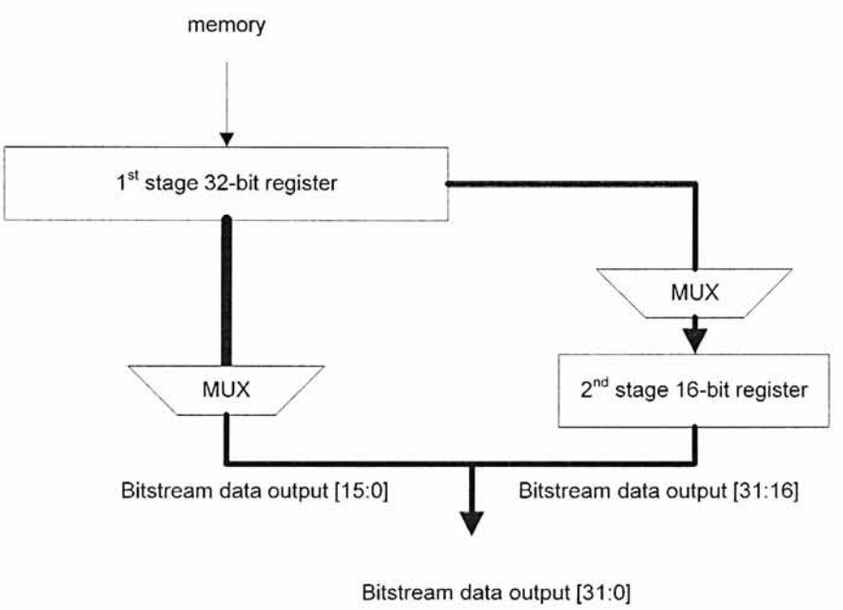

Figure 3-3 : The proposed line bitstream buffer

3.5. Synthesis Result and Performance Analysis

The proposed integration design is implemented using UMC 9nm 9-metal-layer technology. The post layout STA reports critical path being 6.571ns. The maximum frequency is 152M Hz.

The proposed integration occupies 1.33 x 1.33 mm^2 silicon area with the hardware complexity of 183K equivalent gates and 20.5 KB of local memory. Figure 3-4 shows the layout of the proposed integration.

The design is simulated with test video sequences in size of 1920x1088. At working frequency of 143MHz, the decoder average frame rates in CAVLC and CABAC modes are 38.2 fps and 36.2 fps respectively. It can achieve real-time H.264 video decoding on HD1080 video (1920x1088@30Hz). The average throughput in CAVLC and CABAC modes are respectively 464cycles/MB and 490 cycles/MB. Table 3-5 is the comparison of the proposed design and other designs. The proposed design has a gate count of 1.14 times and memory of 4.4 times to Lin's design. The proposed design can work at higher frequencies. The estimated power consumption is 234mW.

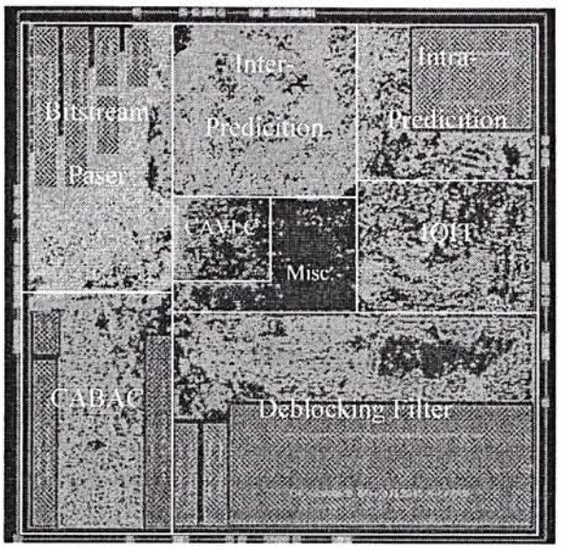

Figure 3-4 : The layout of the proposed integration

	CAVLC			CABAC		
Sequence	Bit rate (Mb/s) @30.00Hz	Average Cycle /MB	Average Frame Rate	Bit rate (Mb/s) @30.00Hz	Average Cycle /MB	Average Frame Rate
Blue_sky	4.97	435	40.3	4.45	441	39.7
Park_joy	28.75	516	34.2	27.25	578	30.8
Pedestrian_area	5.35	421	41.9	4.62	425	41.3
Riverbed	27.03	407	43.0	22.46	464	37.8
Rush_hour	4.35	434	40.7	3.85	445	39.4
Station2	2.64	483	36.5	2.29	499	35.3
Sunflower	3.41	496	35.6	3.02	515	34.1
tractor	10.25	523	33.5	9.26	559	31.3
Average		464	38.2		490	36.2

Table 3-4 : The proposed integrated design simulation result

	Proposed design	Lin [10]	Hu[11]
Specification	1920x1088 @30fps	1920x1088 @30fps	2048x1024 @30fps
Profile	Baseline/Main	Baseline/Main	Main
Gate Count	183K (in NAND2)	160K (in NAND2)	300K
Memory	20.5KB	4.5KB	74KB
Working Freq. (MHz)	143	120	200
Technology	90nm	180nm	130nm
Power consumption	234mW (estimated in synthesis)	320mW	--

Table 3-5 : Comparison of the proposed design with the other design

CHAPTER 4 : CONCLUSION

4.1. Main Contribution

The objective of this research is to develop a Main Profile decoder supporting Full HD resolution by reusing a Baseline Profile decoder. The key contributions are summarized below:

- Development of a full function CABAC decoder;
- Development of a context model buffering and stall cycle reduction scheme;
- Integration of the proposed CABAC to a Baseline Profile decoder;
- Reuse and modification of the Baseline Decoder to support CABAC and Full HD resolution;
- Development of a line bitstream buffer to resolve timing violations in the integrated design;
- Fixing of timing violations caused by memory address generator in the integrated design;
- Development of RTL to Layout design using common ASIC flow;
- Passing of RTL-to-gate equivalency test.

4.2. Reflection on the Development

Reusing design has advantages of shorter development time, verified functionalities and predictable performance. The disadvantages are the limitation in performance of the original reused design.

In this research, many timing violations occur after integration. There is a lot of work in fixing the problem. The performance degrades a little after the timing is fixed. It is important that the reused design has timing close to the target frequency. To minimize problems at later stages, the reused design should be evaluated before the design stage, for example, the maximum frequency and maximum throughput which can limit the overall performance. The maximum throughput can be found from the design document. However, the maximum frequency is only known until the synthesis stage. Therefore, trial synthesis at the target frequency should be taken place before the actual design stage. From the trial synthesis, timing violations can be identified in the very beginning. The effort to fix the violation and consequences can therefore be estimated.

Moreover, the importance of coding style is unexpectedly high. This is the first time the author goes through all procedures from RTL design to physical design. At the end of this research, the author agrees a good coding style can improve the development efficiency, especially in the debugging process in the post-RTL design stages.

After the RTL design, the RTL codes are synthesized to become gate netlist. Timing violations are reported at this stage and they have to be fixed before moving on to the next stage. Fixing timing violation is an iterated process between RTL design modification and synthesis.

Violation path is initially identified by synthesis tools. It has to be investigated whether it is a false path or not. If it is a false path, the violation can be ignored and the path is defined as the exceptional path. If it is a real violation, the source of the violation has to be located. Replacing intermediate signals with shorter

timing paths and adding flopping stages to break down the violation paths are some of the possible solutions. However, the process usually requires reviewing RTL design extensively.

Good coding style, like consistent signal naming style, is good for identifying signal properties when reviewing RTL design. Say, all registers are named with suffix "0q", all input signals are named with prefix "i_", all output signals are named with "o_" and all internal signals are named without suffix or prefix. Signals with "i_" are generated in other blocks. They are the signals from upstream blocks and could be the problem sources. Signals with "o_" are consumed by other blocks. Any change to the output signals will affect theirs consumer blocks. So if the output signals change, their consumer blocks have to be reviewed too.

4.3. Future Work

The first approved version of H.264 was released in 2003. Since it was first released, some additional features, Fidelity Range Extensions (FRExt), Scalable Video Coding (SVC) and Multiview Video coding (MVC) had been included in the standard. FRExt allows higher quality video coding and primarily for professional applications. SVC allows the construction of bitstream that contains sub-bitstream with different spatial or temporal resolution. MVC allows bitstream representing more than one view of a video. The well known application is stereoscopic 3D video coding. All the three features are built from the basic feature of H.246. The author thinks that, after small changes like increasing pel data width, modifying syntax element FSM and adding context models, the proposed design can support

FRExt feature. However, it is not that simple for SVC and MVC. The author thinks that multi decoding cores may be needed as SVC and MVC involving multi bitstream decoding.

Future research can be conducted on SVC and MVC hardware implementation.

BIBLIOGRAPHY

[1] ITU-T, "Recommendation H.264 "Advanced video coding for generic audiovisual services."" 2003.

[2] K. Xu, "Power-efficient design methodology for video decoding. PhD Dissertation, The Chinese University of Hong Kong, Aug. 2007.," 2007.

[3] *IEEE Transactions on circuits and systems for video technology,* vol. 13, 2003.

[4] W. Yu and Y. He, "A high performance CABAC decoding architecture," *Consumer Electronics, IEEE Transactions on,* vol. 51, pp. 1352-1359, 2005.

[5] Z. Peng, G. Wen, X. Don, and W. Di, "High-Performance CABAC Engine for H.264/AVC High Definition Real-Time Decoding," in *Consumer Electronics, 2007. ICCE 2007. Digest of Technical Papers. International Conference on*, 2007, pp. 1-2.

[6] H. Yu, L. Peilin, Z. Hang, Y. Zongyuan, Z. Dajiang, and S. Goto, "A 360Mbin/s CABAC decoder for H.264/AVC level 5.1 applications," in *SoC Design Conference (ISOCC), 2009 International*, 2009, pp. 71-74.

[7] C. Jian-Wen and L. Youn-Long, "A high-performance hardwired CABAC decoder for ultra-high resolution video," *Consumer Electronics, IEEE Transactions on,* vol. 55, pp. 1614-1622, 2009.

[8] Y. Yi and I. Park, "High-Speed H. 264/AVC CABAC Decoding," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY,* vol. 17, p. 490, 2007.

[9] B. Shi, W. Zheng, H. Lee, D. Li, and M. Zhang, "Pipelined Architecture Design of H. 264/AVC CABAC Real-Time Decoding," 2008, pp. 492-496.

[10] C. C. Lin, J. W. Chen, H. C. Chang, Y. C. Yang, Y. H. O. Yang, M. C. Tsai, J. I. Guo, and J. S. Wang, "A 160K Gates/4.5 KB SRAM H.264 Video Decoder for HDTV Applications," *Solid-State Circuits, IEEE Journal of,* vol. 42, pp. 170-182, 2007.

[11] A. S. Y. Hu, K. McAdoo, and J. Cush, "A high definition H.264/AVC hardware video decoder core for multimedia SoCs," *Proc. IEEE Int. Symp. Consumer Electronics,* pp. 385–389, 2004.