國立交通大學

資訊工程所

碩 士 論 文

H.264 視訊壓縮迴路濾波器之超大型積體電路設計

VLSI Architecture for the in-loop filter of H.264 Video Codec

研 究 生：彭彥璁

指導教授：蔡淳仁　教授

中 華 民 國　九 十 三　年 六 月

H.264 視訊壓縮迴路濾波器之超大型積體電路設計

VLSI Architecture for the in-loop filter of H.264 Video Codec

研 究 生：彭彥璁　　Student：Yan-Tsung Peng

指導教授：蔡淳仁　　Advisor：Chun-Jen Tsai

國 立 交 通 大 學

資 訊 工 程 系

碩 士 論 文

A Thesis

Submitted to Department of Computer Science and Information Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science and Information Engineering
June 2004
Hsinchu, Taiwan, Republic of China

中華民國九十三年六月

H.264 視訊壓縮迴路濾波器之超大型積體電路設計

學生：彭彥璁　　　　　　指導教授：蔡淳仁

國立交通大學資訊工程所碩士班

摘要

以 Block 爲單位的動態補償變換編碼之視訊編碼/解碼方法是目前最成功的視訊編碼技術。在低位元率的應用下，這種編碼/解碼的方式會產生 blocking artifacts，讓畫面品質變差。即使能夠使用去塊濾波器去減少 blocking artifacts，這種濾波器因爲在運算上十分複雜，所以在處理器較弱的嵌入式編碼/解碼平台上對效能的影響很大。此篇論文目的在設計一個有效率的超大型積體電路去塊濾波器架構。 此外，在本研究中是先分析控制邏輯、計算單元和記憶器子系統，進行分來幫助有效率的設計整個硬體架構。本論文的迴路濾波器是以符合 MPEG-4 AVC/H.264 [1]的演算法爲目標。MPEG-4 AVC/H.264 是一個新一代的視訊壓縮標準，它的視訊壓縮效率更優於 MPEG-4 Advanced Simple Profile [2] 和 H.263+。爲這個標準，本論文設計了一個二階層的管線去塊濾波器和一個可以和 AMBA[3] 匯流排架構界面溝通的 codec 加速器架構。管線的切法是以在特定目標晶片上能以 50 MHz 達到 CIF 解析度即時壓縮的目的來設計的。在本論文中提出的方法主要是在設計一個 Real-time 濾波器的硬體設計和整個匯流排系統架構，此外，我們使用一個 SoC emulation Platform, ARM INTEGRATOR [4], 來驗證整個系統的功能性和量測整體的效能。

VLSI Architecture for the in-loop filter of H.264 Video Codec

Student : Yan-Tsung Peng Advisor : Chun-Jen Tsai

Department of Computer Science and Information Engineering
Nation Chiao Tung University

ABSTRACT

Block-based hybrid motion compensated transform video codecs are the most successful class of video coding technologies. For low bit-rate applications, this type of codecs suffer from blocking artifacts that causes an unpleasant visual effect. Even though deblocking filters can be used to smooth out blocking artifacts, it is quite often being omitted from low power embedded video terminals due to the computational complexity of a post processor. This thesis studies efficient VLSI architecture for deblocking filters for video applications. Thorough analysis on the complexity of control logic, computational units, and memory subsystem are conducted. In particular, an efficient implementation for the in-loop filter of the emerging new video coding standard, namely MPEG-4 AVC/H.264 [1] with superior performance compared to MPEG-4 Advanced Simple Profile [2] and H.263+, is presented. A two-stage pipeline deblocking hardware architecture and an generic codec accelerator infrastructure with an interface to AMBA bus protocol [3] is proposed. The feature of the proposed method is focused on an efficient hardware design for In-Loop Filter and the whole bus system architecture. Furthermore, to verify the functionality and performance of the proposed hardware design, an SoC emulation platform, the ARM INTEGRATOR[4], is used for H.264 hardware/software co-development.

Acknowledgement

At first, I would like to thank my advisor, Professor Chun-Jen Tsai, for providing me a great environment to research and guide me toward a right direction when I encounter a bottleneck in my research. Secondly, I would like to thank Yueh-Yi Wang, who is a Ph.D. student at the same lab with me, to give me advice about hardware design. At last, I want to thank my family to support me and encourage always me without any complaints.

Index of Content

Index of Figure

Index of Table

1. Introduction to Video Filtering

1.1. Introduction

Block-based transform video coders suffer from an annoying visual distortion, blocking artifacts [6]. This phenomenon is characterized by luma and chroma discontinuities on block boundaries in video frames. The cause of blocking artifacts is that the inter-block correlation is lost during the quantization process of a video codec. To remove these blocking artifacts, a low pass filter can be used to smooth out the block boundaries [7]. These filters are usually referred to as the deblocking filters or smoothing filter. One of the simplest deblocking filters is a moving average filter. In a moving average filter, each pixel in the image is replaced by the average of neighboring pixels [7]. An $n \times n$ region is typically used as the averaging window. The size n of the window controls the degree of smoothing or blurring of the deblocking process. Unfortunately, a space-invariant moving averaging filter introduces blurring distortion in addition to removing the blocking artifacts. As a result, the visual quality is still not improved. To remove the blocking artifacts without blurring distortion, an adaptive deblocking process is required. Furthermore, an adaptive filter can reduce the computational complexity since degree of filtering can be reduced around image area where human vision is less susceptible to blocking artifacts [6].

In order to find an efficient deblocking filter, one must understand how the human visual system works. There are three interesting observations about the human visual systems [5]. First of all, the human visual system is more sensitive to blocking artifacts in flat regions than in complex regions. Therefore, a deblocking filter should perform strong smoothing in flat regions, while in complex regions, only a few pixels around block boundaries need to be processed. Secondly, in complex regions, it is easy to introduce the undesired blurring distortion if filtering is not done carefully. An adaptive filter must prevent the image details from being smoothed out. Extra checking must be used to determine whether it is necessary to smooth a pixel or not in a complexity region. Finally, due to motion compensation, blocking artifacts will be propagated to the next frame. Due to that, the pixels inside the flat regions must be filtered as well.

The chapter is organized as follows. Section 1.1 first presents the two main strategies of deblocking, namely post processing versus in-loop processing. The pros and cons of each strategy will be discussed. In section 1.2, an introduction to the deblocking post processor [5] recommended by the MPEG-4 Visual Standard [8] is presented. Section 1.3 describes the in-loop deblocking filter of the emerging new video coding standard, ISO MPEG-4 AVC/ITU-T H.264 [1]. Finally, a brief overview the remaining part of the thesis is given in section 1.4.

1.2. Post Processing versus In-loop Processing

1.2.1. The Post Processing Approach

Post processing is to compensate the blocking artifacts closely related to the block-based structure of quantization. A significant improvement in subjective quality can be achieved by using filters designed to remove coding artifacts, in particular blocking and ringing. The goal of post processing is to reduce coding artifacts while maintaining visually important image features.

In general, there are two coding artifact filtering approaches which are deblocking filter and deringing filter. Two different kinds of deblocking filters are described in this chapter. The ISO MPEG-4 Visual Standard post processor is presented in section 1.3 and the in-loop filter of AVC is described in section 1.4.

The aim of deringing filter is to remove ringing artifacts. Quantization with large quantizer step size can have a low-pass filtering effect, since higher-frequency AC coefficients tend to be removed during quantization. This low-pass effect can cause ringing or ripples near strong edges in the original images. This is similar to the effect of applying a low-pass filter to a signal with a sharp change in amplitude: low-frequency ringing components appear near the sharp transition position [3]. MPEG-4 Annex F describes an optional post-decoder de-ringing filter [4]. In this algorithm, a threshold is set for each reconstructed block based on the mean pixel value in the block. The pixel values within the block are compared with the threshold and 3 x 3 regions of pixels that are all either above or below the threshold are filtered using a 2-D spatial filter. This has the effect of smoothing homogeneous regions of pixels on either side of strong image edges while preserving the edges themselves: it is these regions that are likely to be affected by ringing.

1.2.2. The In-loop Filtering Approach

In-loop filtering was first adopted in the ITU-T H.261 video coding standard [11]. H.261 utilizes a spatial low-pass filter in the predictor, the “filter in the loop” or “loop filter”, which can be switched on a macroblock basis. It was later dropped from the succeeding standard, H.263 [12], because the bilinear interpolation used in H.263 for half-pel MC introduces spatial low-pass filter as a side effect. In H.264 [1], it was added again to force users to filter the decoded images for better visual quality.

An in-loop filter sits inside the video coding loop and is applied to the reconstructed reference frame both in the encoder and the decoder. There are two major advantages of this approach. First, the filtering within an encoder loop can improve the visual quality of the reconstructed referenced frame. As a result, motion prediction is more accurate since there is less quantization noise to bias the matching criteria (sum of absolute difference is usually used here). Secondly, the in-loop filter is an integrated part of the codec so it guarantees that the decompressed frames are filtered before display. This is not the case with a out-of-loop post processor, for example, the informative filters used in MPEG-4 Simple Profile, which is often omitted to reduce the complexity and cost of the decoder.

The main disadvantage of an in-loop filter is simply that it increases the complexity and is not feasible with many embedded platforms without extra hardware support. Therefore, an efficient and low-cost VLSI architecture for in-loop filters is crucial to the success of advanced video codecs such as AVC/H.264.

1.2.3. The Comparisons of In-loop and Post Processing

Post processing filters can be decoder-dependent as shown in Fig 1, or decoder-dependent, as shown in Fig 2. The decoder-dependent ones are applied after the decoder and makes use of decoded parameters to improves the performance. An example is the deblocking filter design in Annex F.3.1 of MPEG-4 Visual Specification [2]. For example, a useful decoder parameter is the quantizer step size which can be used to predict the expected level of distortion in the current processed block. For example, high distortion will occur when the quantizer step size is large. This enables the decoder to adjust the strength of the filter according to the expected distortion. A strong filter may be applied when the quantizer step size is large, reducing the relevant type of distortion. A weak filter is applied when the step size is small, preserving detail in blocks with lower distortion. Moreover, in order to minimize dependence on the decoder, the filter may be applied after decoding without any information of decoder parameters. This approach gives the maximum flexibility but the performance, however, is generally not as good as decoder-dependent filters.

The in-loop filter, which is shown in Fig 3 and Fig 4, is applied to the reconstructed frame both in the encoder and in the decoder. Applying the filter within the encoder loop can improve the quality of the reconstructed reference frame, which in turn improves the accuracy of motion-compensated prediction for the next encoded frame since the quality of the prediction reference is improved. On the other hand, the complexity of the codec is higher. This is in particular a problem with weak decoders.

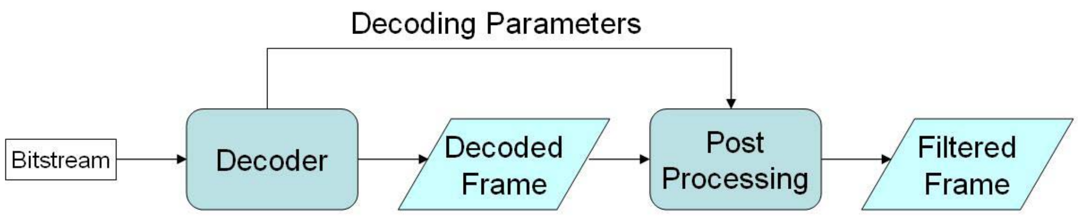


Fig 1. Decoder-dependent Filter

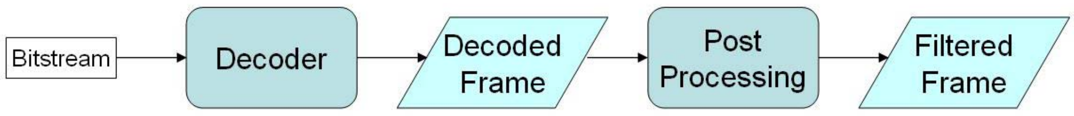


Fig 2. Decoder-independent Filter

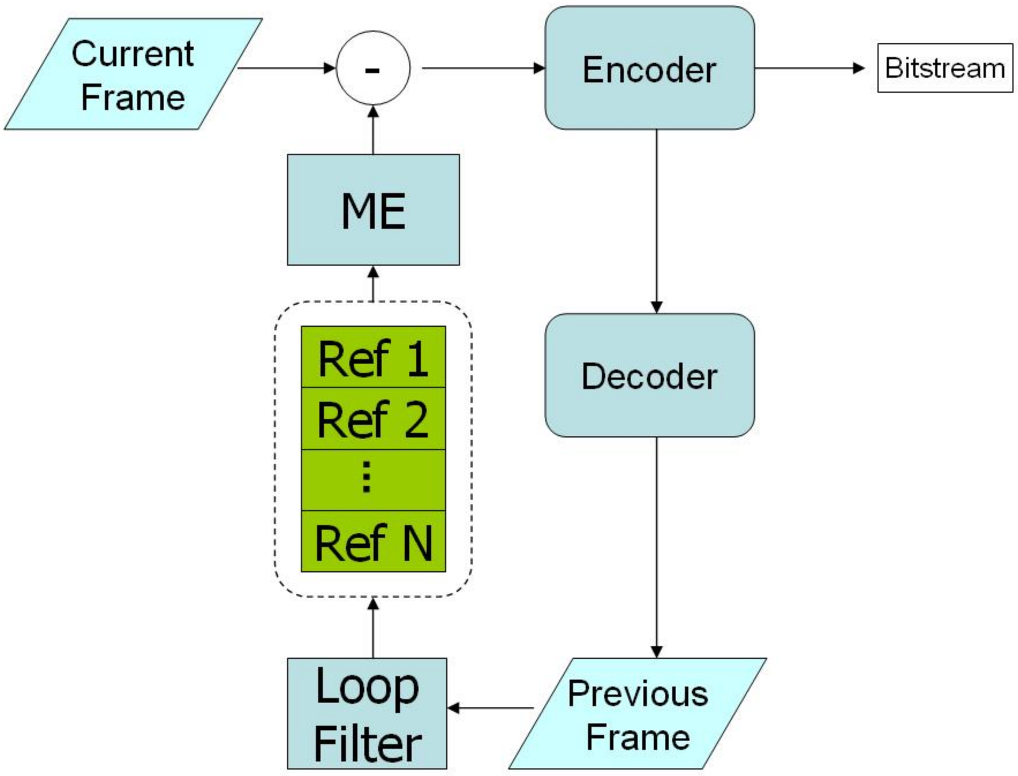


Fig 3. In-Loop Filter in Encoding Process

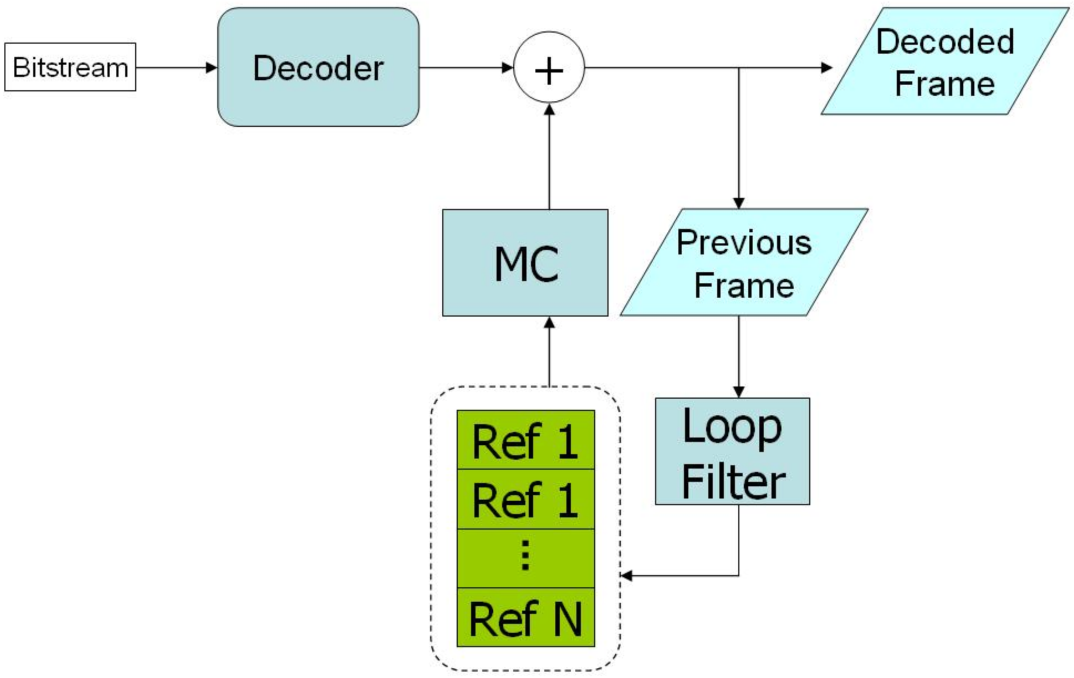


Fig 4. In-Loop Filter in Decoding Process

1.3. The ISO MPEG-4 Visual Standard Post Processor

1.3.1. Introduction to ISO/IEC MPEG-4 Visual Standard

In November 1992, a new work item proposal for very low bit-rate audio-visual (AV) coding was presented in the context of ISO/IEC JTC1/SC29/WG11, well known as MPEG (Moving Pictures Experts Group). The scope of the new work item was described as "*the development of international standards for generic audio-visual coding systems at very low bitrates (up to tens of kilobits/second)*" [5]. The main motivations for the starting of the new work item were basically the prevision that the industry would need very low bit-rate video coding algorithms in a few years. The MPEG-4 work happened in July 1994, at the Grimstad meeting, when members were faced with the need to broaden the objectives of MPEG-4 which could no more be based on a pure compression gain target coming from new coding approaches such as region-based, analysis-synthesis, fractals or any other, since very few people believed in a compression improvement sufficient to justify (alone) a new standard (beside the LBC, latter H.263, standard). There are several functionalities in MPEG-4 Visual Standard, especially for very low bit-rate applications:

- **Improved coding efficiency** – This is clearly functionality useful for very low bit-rate applications since improved coding efficiency is asked for, to supply an answer to the requests coming from, e.g. mobile network users.
- **Robustness in error-prone environments** – The access to video applications through channels with severe error conditions, such as some mobile channels, requires sufficient error robustness is added. This functionality, by providing significant quality improvements, will with no doubts stimulate very low bit-rate applications, notably in mobile environments.
- **Content-based scalability** – The ability to achieve scalability with a fine granularity in content, spatial or temporal resolution, quality and complexity, or any combination of these cases, is a fundamental concept for very low bit-rate applications since it provides the capacity to adapt the AV representation to the available resources.
- **Improved temporal random access** – The provision of random access efficient methods, within a limited time and with fine resolution, including conventional random access at very low bit-rate is the target.
- **Content-based manipulation and bitstream editing** – This functionality is more related to the syntactic organization of the information than to any specific bit-rate resources.

- **Content-based multimedia data access tools** – The possibility to content-based selectively access AV data is for sure an important capability in the context of very low bit-rate applications since it will allow to optimize the video information to transmit depending on the available resources.
- **Hybrid natural and synthetic data coding** – Although this functionality is quite bit-rate-independent, the efficient integration of natural and video data will for sure make no harm to very low bit-rate applications.

The very low bit-rate applications in MPEG-4 are quite important, however, it would suffer from annoying blocking artifacts because high quantization step sizes are applied to achieve low target bitrate. Due to that, a good post processor to remove the blocking artifacts is essential. In next paragraph, ISO MPEG-4 Visual Standard post processor will be introduced and the MPEG-4 decoding process with a deblocking filter is shown in Fig 5.

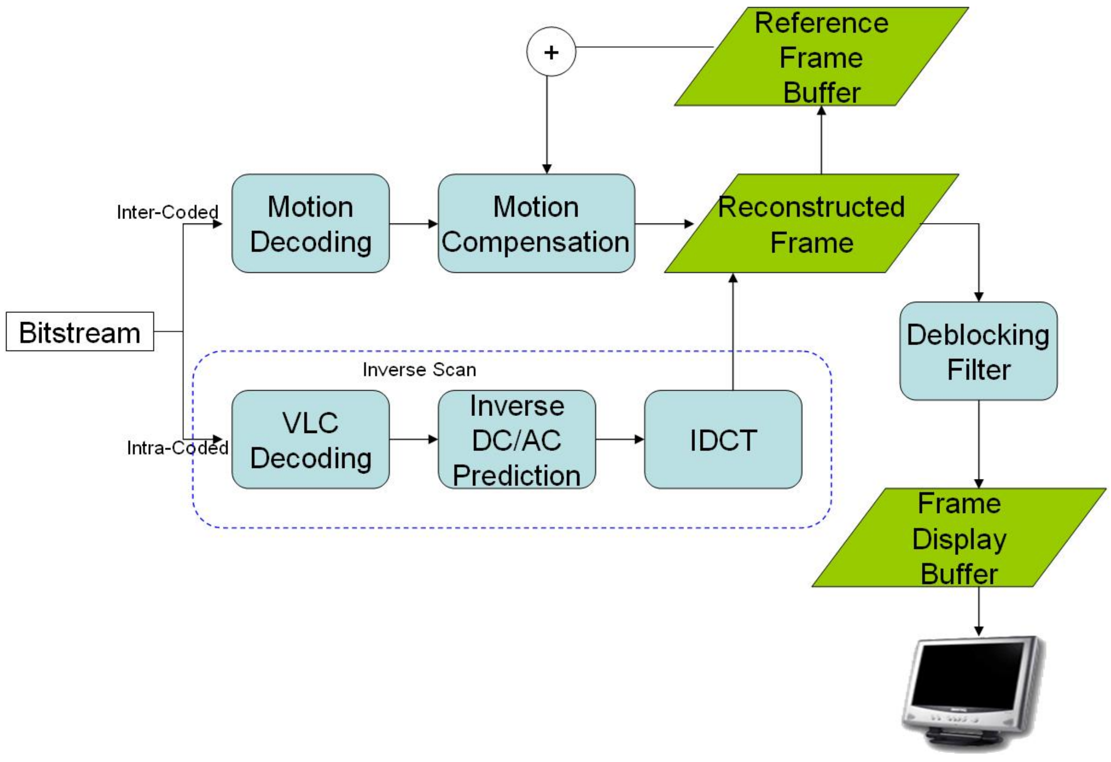


Fig 5. MPEG-4 Decoding Process with Deblocking Filter

1.3.2. ISO MPEG-4 Visual Standard Post Processing

This filtering process consists of three major functional modules, i.e. mode decision, filtering for the flat region mode, and filtering for the complex region mode.

All of this three operating modes are block-based. One block contains 8*8 pixels. This de-blocking process filters the decoded Y, Cb and Cr frames. The filter performs filter operation along 4 block boundaries, i.e. top edge, bottom edge, left edge, and right edge. Filtering left and right edges is called vertical filtering. The horizontal filtering is for the other two edges. Let us take the vertical filtering as an example to illustrate this algorithm. In one filtering step, it processes 10 pixels within two neighboring blocks. Along one block edge, the left 5 pixels are called v0, v1, v2, v3, and v4 and the right 5 pixels are called v5, v6, v7, v8, and v9. The block edge is between v4 and v5. In the following three paragraphs, the three function modules will be described.

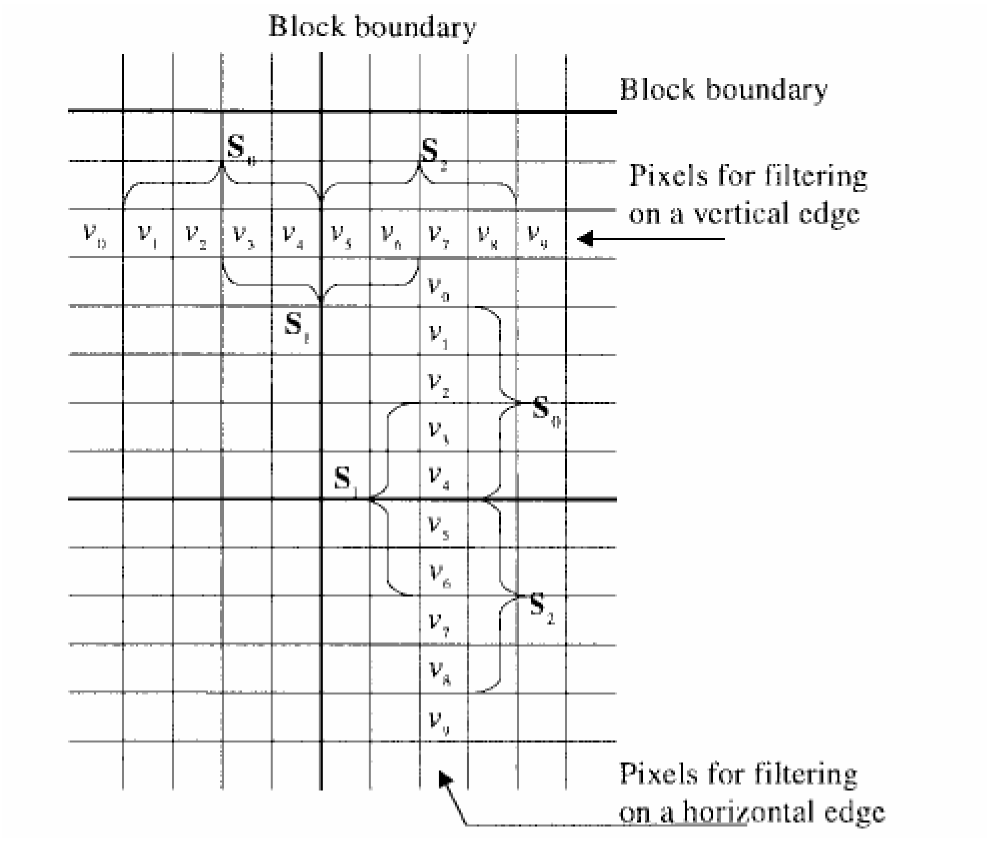


Fig 6. 8x8 Block Boundary

A. Mode Decision

The flatness of the 10 pixels is calculated, v0 to v9, as shown in Fig 6:

Flatness(v) = φ(v0-v1) + φ(v1-v2) + φ(v2-v3) + φ(v3-v4) + φ(v4-v5) + φ(v5-v6) + φ(v6-v7) + φ(v7-v8) + φ(v8-v9)

Where

φ(v) = 1, if |v| <= T1

φ(v) = 0, otherwise.

T1 represents a threshold to reflect the flatness. If the Flatness (v) is smaller than a threshold, T2, the filtering must be applied for the complex region mode. On another condition, the filtering must be applied for the smooth region mode.

B. Filtering in the Smooth Region Mode

MAX and MIN, represent the maximum value among v1 to v8 and minimum value among v1 to v8 relatively. The QP represents the specified quantization parameter of this block.

```
if ( |MAX–MIN| < 2*QP ) {
    $v'_n = \sum_{k=-4}^{4} b_k \cdot p_{n+k}, 1 \le n \le 8$
    $p_m = \begin{cases} (|v_1 - v_0| < QP)?v_0 : v_1, if \quad m < 1 \\ v_m, \quad if 1 \le m \le 8 \\ (|v_8 - v_9| < QP)?v_9 : v_8, if \quad m > 8 \end{cases}$
    $\{b_k : -4 \le k \le 4\} = \{1,1,2,2,4,2,2,1,1\} // 16$
}
Else
    No change will be done.
```

The purpose of the statement, if (|max–min| < 2*QP), is to prevent real edges in this filtering process from being smoothed out, for example: if there are two neighboring blocks, the color of the left block is white and the right block is black. In this case, the block edge is a real edge in the original image. But if the condition |max-min| < 2*QP is not checked, it will regard this edge as a blocking artifact and apply the strong smoothing filter over it.

C. Filtering in the Complex Region Mode

In this mode, the frequency components $a_{3,0}$, $a_{3,1}$, and $a_{3,2}$ are defined, which are evaluated from the simple inner product of the approximated DCT kernel [2 -5 5 -2] with the pixel vectors, i.e.,

a3,0 = ([2 -5 5 -2] • [v3 v4 v5 v6]T) // 8,
a3,1 = ([2 -5 5 -2] • [v1 v2 v3 v4]T) // 8,
a3,2 = ([2 -5 5 -2] • [v5 v6 v7 v8]T) // 8.

In this mode, it only filters two pixels next to the block edge.

```
if(|a3,1|<QP)
{
```

$v_4' = v_4 - d$,

$v_5' = v_5 + d$,

where

$d = \mathrm{CLIP}(5\cdot(a_{3,0}' - a_{3,0})//8, 0, (v_4 - v_5)/2))$

$a_{3,0}' = \mathrm{SIGN}(a_{3,0}) \cdot \mathrm{MIN}(|a_{3,0}|, |a_{3,1}|, |a_{3,2}|)$.

}

The check on the condition (|a3,1|<QP) is to preserve image details. The three functional modules mentioned above produces good subjective and objective visual quality because they apply several useful judgments to adaptively select a proper filtering process.

1.4. The In-loop Filter of MPEG-4 AVC/H.264

1.4.1. Introduction to ISO/IEC MPEG-4 AVC/ITU-T H.264

MPEG-4 and H.263 video codecs standards are based on video compression (video coding) technologies from 1995. The groups responsible for these standards, the Motion Picture Experts Group (MPEG) and the Video Coding Experts Group (VCEG) joint forces and developed a new standard, MPEG-4 AVC/H.264, that significantly outperforms MPEG-4 and H.263. The new codec, released in 2003, provides better compression of video images as well as a range of features supporting high-quality, low-bitrate streaming video. The origin of the new codec can be traced back to 1995. After finalizing the original H.263 standard for video-telephony in 1995, ITU-T VCEG started working on two further development projects: a short-term effort to add extra features to H.263 (resulting in Version 2 of the standard) and a long-term effort to develop a new standard for low bit-rate visual communications. The long-term effort led to the draft H.26L standard, offering significantly better video compression efficiency than previous ITU-T standards. In 2001, ISO/IEC MPEG issued a call for proposal (CfP) for next generation high performance video coding tools. VCEG responded to the call with H.26L and as a result the Joint Video Team (JVT), composed of experts from both MPEG and VCEG was formed. During the two-year time frame of standardization, most the H.26L components have been replaced with new design. The new standard is named MPEG-4 Advanced Video Codec (AVC) by ISO and H.264 by ITU [6]. H.264/AVC offers enhanced compression performance and provides a network-friendly video representation which addresses conversational (video-telephony) and non-conversational (storage, broadcast or streaming) applications. The H.264/AVC adopts a three-layer design. The

Video Coding Layer (VCL) accounts for the compression of the video content. The Network Abstraction Layer (NAL) defines a packet-based representation of the compressed video content. However, NAL is transport-independent. The transport dependent design is governed by the Transport Encapsulation Layer (TEL). TEL is not a normative part of the standard.

The baseline profile of MPEG-4 AVC includes:

- I and P slice types
- Chrominance format 4:2:0
- Progressive coding
- 1/4-sample motion compensation
- Tree-structured motion segmentation down to 4x4 block size
- VLC-based entropy coding
- In-loop deblocking filter
- Some enhanced error resilience features
  - Flexible macroblock ordering (maximum 8 slice groups)
  - Arbitrary slice ordering
  - Redundant slices

The emerging H.264 standard saves as much as 50% bit-rate over H.263 while maintaining the same or better visual quality. On the other hand, the computational complexity is much higher than the previous video coding standards.

1.4.2. Adaptive Deblocking Filter in H.264/AVC

There are several advantages for de-blocking filters to perform filtering inside the coding loop. First of all, a loop filter guarantees certain level of quality of any standard-compliant implementations. This is especially important in low bitrate communications systems. If a loop filter is enforced inside the codec, content providers can assume that their material would always be processed by a deblocking filter so that the quality of their material is guaranteed.

Secondly, in an extreme case, an frame buffer in decoder can be saved because in the post-filtering approach, a reference frame buffer is required to store the decoded frame and another frame buffer is necessary to store the filtered frame for display. These two buffers are merged into one with the loop filter design.

Finally, the overall visual quality of video streams with a loop filter is better than that without it because the filtered reference frames offer higher quality prediction for motion compensation. The decoded video streams with a loop filter are sharper than those with a post filter. Computation complexity can be reduced by taking into account the fact that the image area in past frames is already filtered.

Even though there are several advantages, it is inevitable that applying a loop filter in codec design requires high computational complexity. The complexity comes from the highly adaptive nature of the filter. Therefore, it is very important to design an efficient and effective algorithm for loop filter.

The algorithm of H.264/AVC In-Loop filter is described as follows. In the loop filtering process of H.264, there are three steps, determining the boundary filtering strength, getting thresholds for each block boundary, and filtering process according boundary strength. In each macroblock, the deblocking filter process applies to both luma and chroma. For each macroblock, vertical edges are filtered first, from left to right, and then horizontal edges are filtered from top to bottom. The luma deblocking filter process is performed on four 16-sample edges in each direction, as shown in Fig 7.

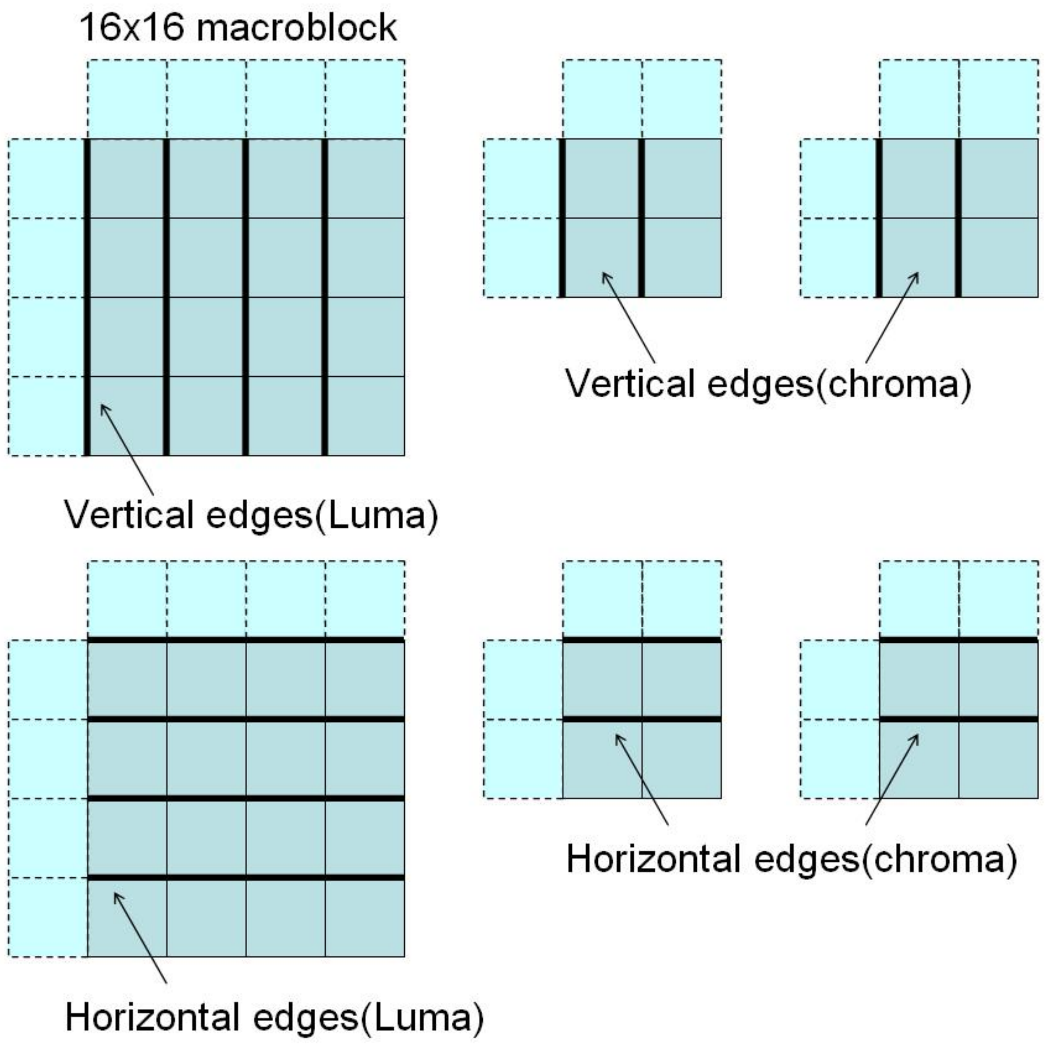


Fig 7. Filtered Boundaries in a Macroblock

For determining each boundary between neighboring 4x4 luma blocks, boundary strength is assigned as shown in Fig 8. If one of the neighboring blocks is intra-coded, boundary strength is set to 3 for strong filtering. If the previous condition is true and its block boundary is also macroblock boundary, boundary strength is set to 4 for even stronger filtering. If neither of the blocks is intra-coded and coefficient coded in one of the neighboring blocks, the medium filtering with boundary strength is applied. Boundary strength is 1 when encoding mode of two neighboring blocks are not coded and have different reference frames, the number of reference frame is not identical, or the difference of motion vectors of two blocks are greater than 4 for weak filtering. If all of previous conditions are not satisfied, boundary strength is equal to 0 for no filtering.

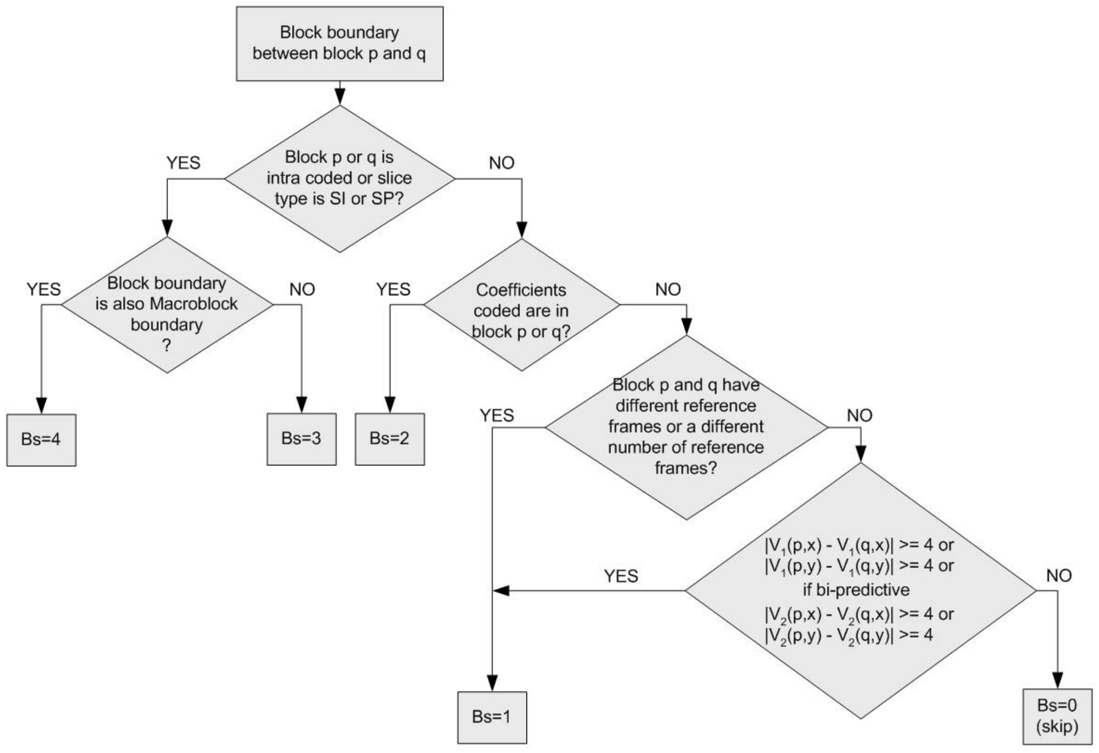

Block boundary between block p and q
Block p or q is intra coded or slice type is SI or SP?
YES
NO
Block boundary is also Macroblock boundary ?
YES
NO
Coefficients coded are in block p or q?
YES
NO
Bs=4
Bs=3
Bs=2
Block p and q have different reference frames or a different number of reference frames?
YES
NO
|V1(p,x) - V1(q,x)| >= 4 or |V1(p,y) - V1(q,y)| >= 4 or if bi-predictive |V2(p,x) - V2(q,x)| >= 4 or |V2(p,y) - V2(q,y)| >= 4
YES
NO
Bs=1
Bs=0 (skip)

Fig 8. Flowchart for Getting Boundary Filtering Strength

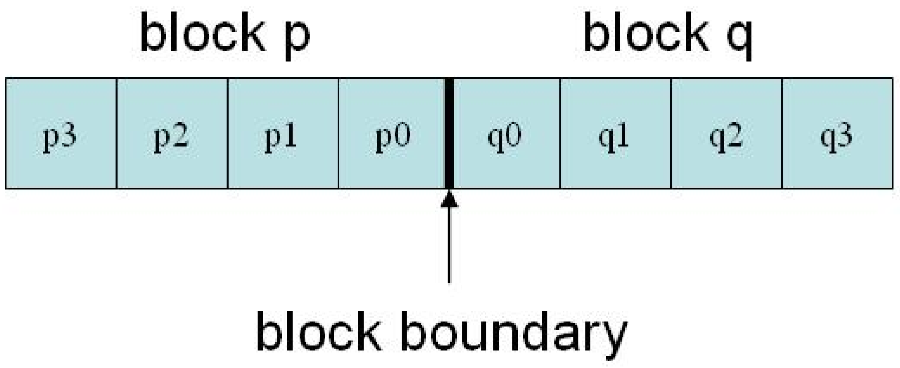


Fig 9. Convention for Describing Samples across a 4x4 Block

Samples across this edge are only filtered if the conditions, Bs $\neq$ 0, Abs($p_0 - q_0$) $< \alpha$, Abs($p_1 - p_0$) $< \beta$ and Abs($q_1 - q_0$) $< \beta$, are all true, which Bs represents boundary strength, and p3, p2, p1, p0, q3, q2, q1, q0 represent the pixels between block boundary as shown in Fig 9. Besides, α and β are quantization parameter dependent thresholds.

A strong filtering process is applied for edges with boundary strength equal to 4.

The two intermediate threshold variables, a_p and a_q, which represents Abs($p_2 - p_0$) and Abs($q_2 - q_0$) relatively, are used to determine whether the luma samples p_1 and q_1 is needed to filtered at this position of the edge. For luma samples, if the conditions , $a_p < \beta$ and Abs($p_0 - q_0$) < ((α >> 2) + 2), are all holds, the following filtering operation must be executed:

$$P_0 = (p_2 + 2p_1 + 2p_0 + 2q_0 + q_1 + 4) >> 3$$

$$P_1 = (p_2 + p_1 + p_0 + q_0 + 2) >> 2$$

$$P_2 = (2*p_3 + 3*p_2 + p_1 + p_0 + q_0 + 4) >> 3$$

P_0, P_1, and P_2 are the filtered values. For chroma samples or for luma samples which either $a_p < \beta$ or Abs($p_0 - q_0$) < ((α >> 2) + 2) does not satisfy, only p_0 shall be filtered according to:

$$P_0 = (2*p_1 + p_0 + q_1 + 2) >> 2$$

The q values are modified in a similar manner by substituting condition $a_q < \beta$ for $a_p < \beta$.

2. Previous Work

The importance of deblocking filter for visual quality improvement is introduced in the previous chapter. In addition, the two well-known algorithms, namely the informative deblocking filter in ISO MPEG-4 Visual Standard and the in-loop filter of H.264/AVC, are described briefly. Since the research topic in this thesis is about a VLSI design of deblocking filter, some prior arts are surveyed in this chapter.

For real-time multimedia applications of complex algorithms, pipelining and carefully designed memory subsystems are crucial for VLSI implementations. So far, there are few papers on VLSI implementation of deblocking filters. We discuss two most directly related papers in this chapter. The first one is “Real-time deblocking filter for MPEG-4 systems” [16] by Fang et al., The second one is “Architecture design for deblocking filter in H.264/JVT/AVC” [17] by Y.W Huang, et al.

2.1. Prior proposals on VLSI Design of Deblocking Filters

Two previously published architectures of video deblocking filters would be discussed here. Fig 10 shows the deblocking filter architecture for MPEG-4 systems [16]. Fig 11 is the in-loop filter for H.264/AVC published in.[17]. In Fig 10, the input pixels are shifted into the ISR pixel-by-pixel, which provides storage of input pixels for other modules. As the detailed design in ISR depicted in Fig 12 shows, ten 8-bit registers are used in this module to store input pixels from outside memory and it sequentially propagates from P9 to P0. There are five multiplexers at the input of P8, P3, P2, P1, and P0 for pixels replacement from input pixels to padded pixels. The MD module determines which mode should be used for the filtering operation. The DF module is the deblocking process performing the default mode which only filters two pixels. The SF module is the deblocking of smooth mode filtering eight pixels. Both DF and SF modules are two-stage pipeline architecture for reducing the critical path and increasing the frequency. The CP is the module to generate the padding pixels for filtering for the ISR module. Filtered pixels are fed in the OSR, which is shown in Fig 13, and sequentially shifted out.

For the in-loop filter design for H.264/AVC (shown in Fig 11), the solid lines denote data path, and the dotted lines denote control signals. It has to load a macroblock and adjacent block from external RAM via system bus to on-chip SRAM before filtering. To support the parallel filter with high utilization, there two SRAM modules are well-organized that classify 4x4 blocks in different columns and use a word (32 bit) to store four pixels.

There are two architectures using different kinds of SRAM, the single port SRAM and the dual port SRAM. When using the single port SRAM, it uses two blocks for the base architecture with only one read and one write ports to store the macroblock pixels. In Fig 14, it shows the careful organization of pixels in two single-ported SRAM blocks that makes it able to get one unfiltered pixel per clock cycle. The processing order of block boundaries for both directions is shown in Fig 15, which the write label with black number denotes the horizontal filtering on vertical edges and the black label denotes the vertical filtering on horizontal edges. The data path of basic architecture is shown in Fig 16 where the solid lines and dotted lines denote the horizontal filtering across vertical edges and the vertical filtering across horizontal edges relatively.

On the other hand, if dual port SRAM is used, it only needs one SRAM block without modifying the data flow to store the macroblock pixels, as shown in Fig 17. The 160x32 dual-port SRAM has two separate read and write ports that can perform two read operations at the same cycle, as well as two write operations, or one read and one write at the same cycle. The processing order of boundaries for the advanced architecture must be modified to make use of dual-port SRAM, as shown in Fig 18.

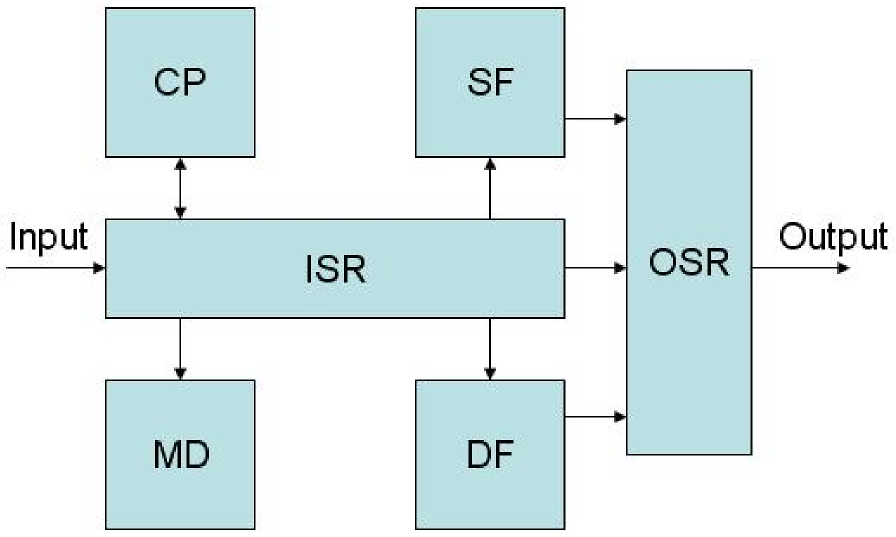


Fig 10. Block Diagram of Deblocking Filter Architecture in MPEG-4 Systems

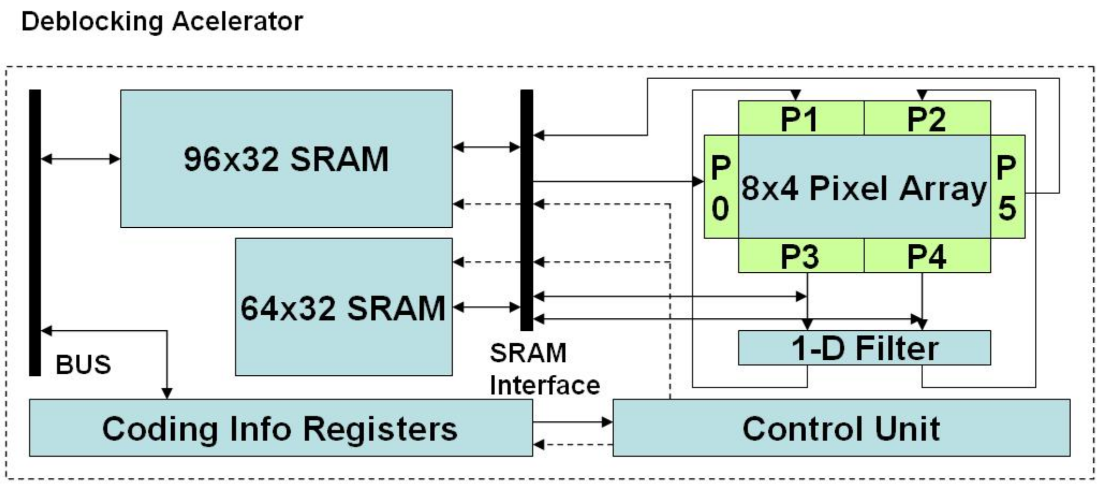


Fig 11. Architecture Design for Deblocking Filter in H.264/AVC

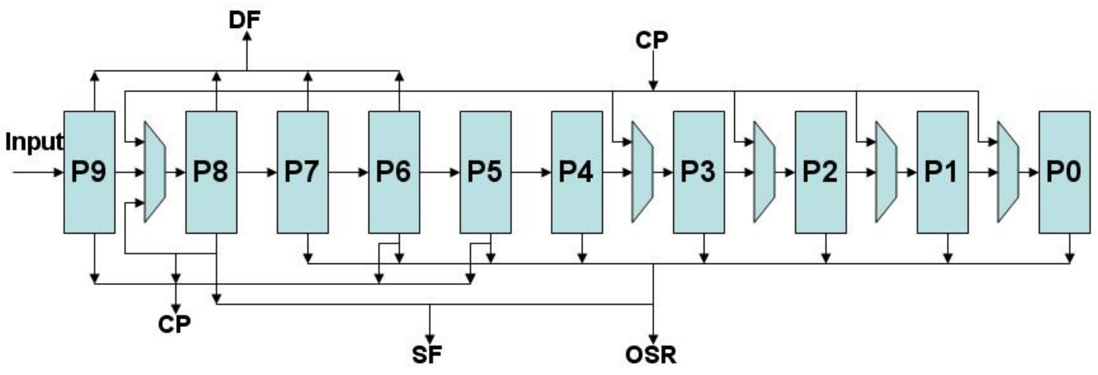


Fig 12. Interconnections between ISR Module and Other Modules

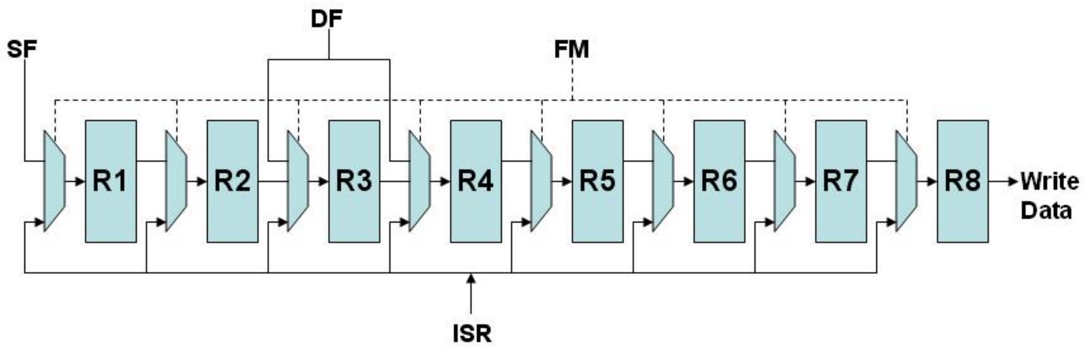


Fig 13. Interconnections between OSR Module and Other Module

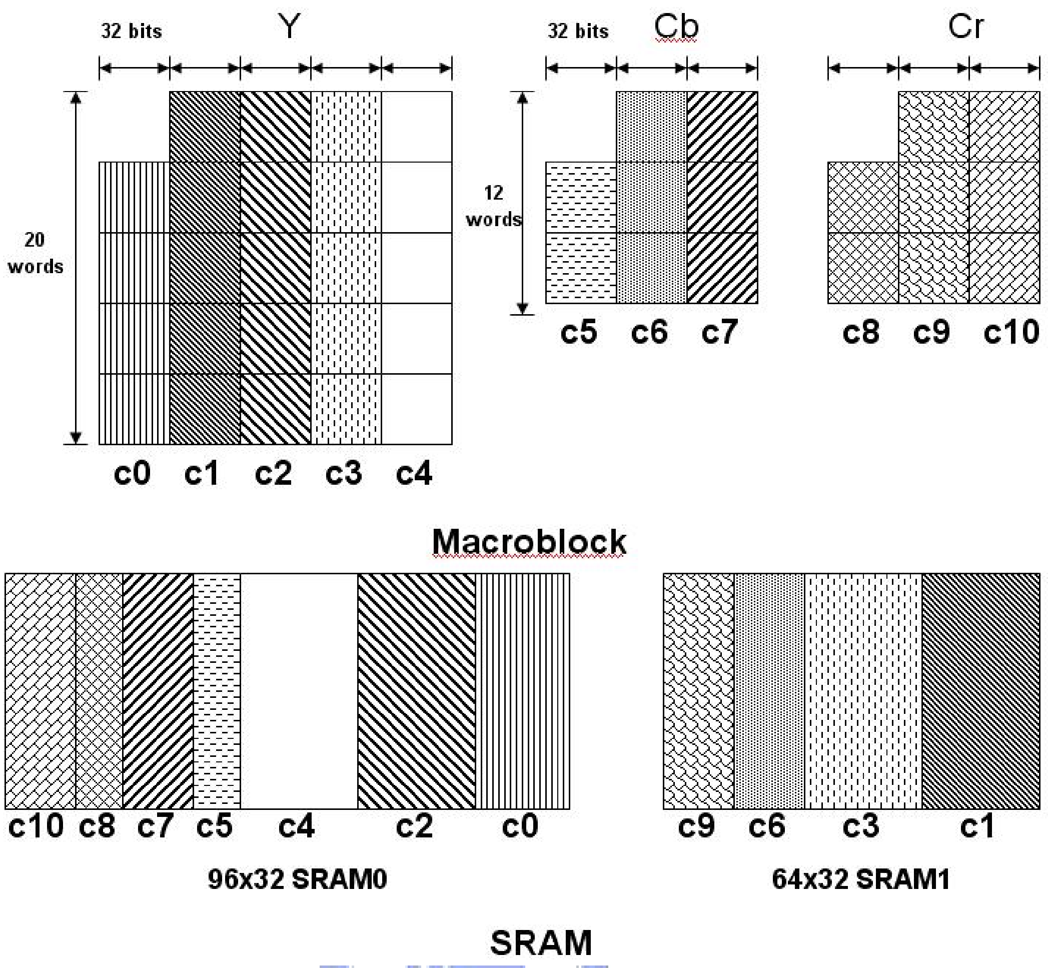


Fig 14. Organization of On-chip Single Port SRAM Modules

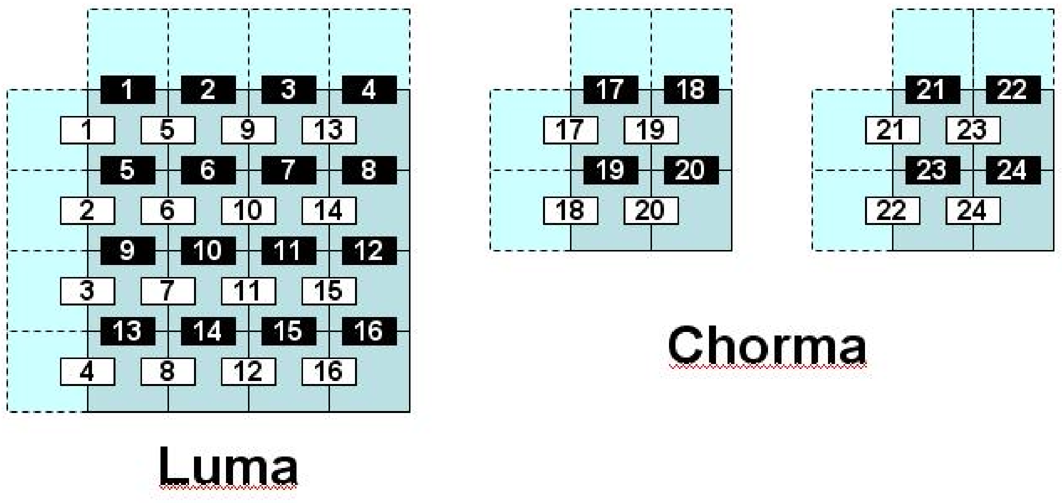


Fig 15. Processing Order on Boundaries of Basic Architecture

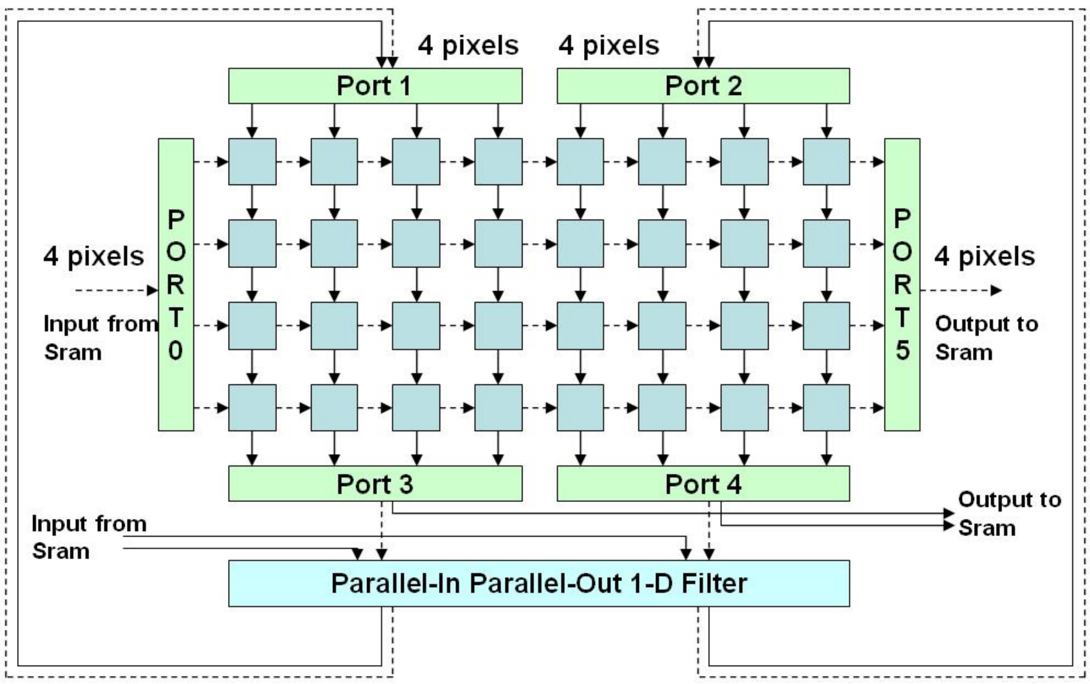


Fig 16. Data path for Basic Architecture with Two Single-port SRAM Modules

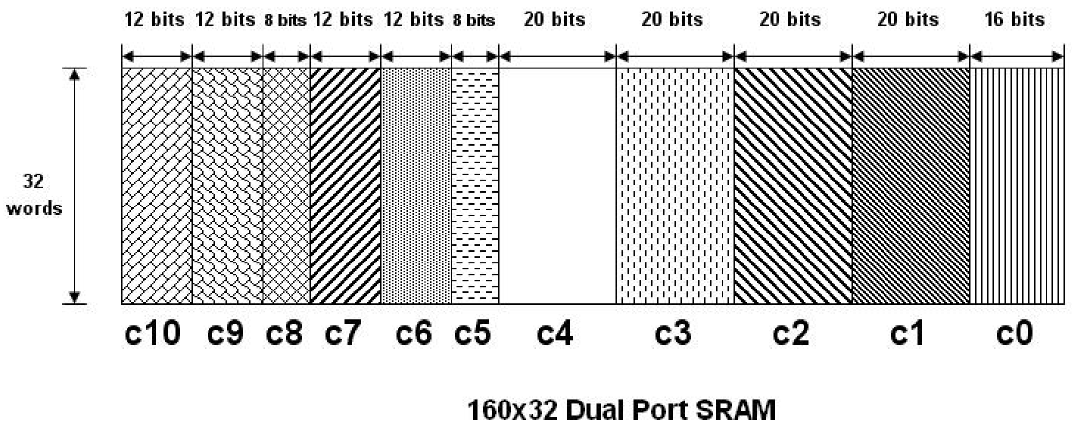


Fig 17. Organization of On-chip Dual Port SRAM

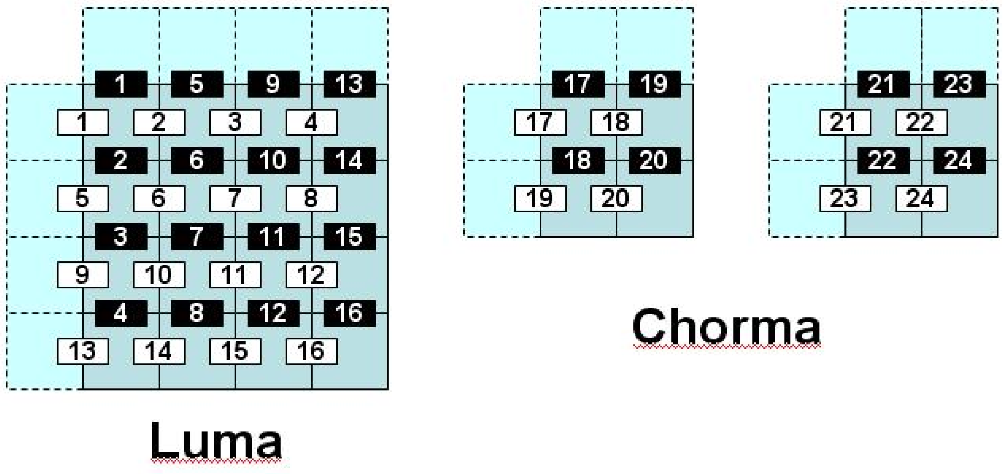


Fig 18. Processing Order of Boundaries for The Advanced Architecture

3. The Emulation Platform and the System Overview

3.1. System Design Overview

The bus system design for H.264 video accelerator hardware architecture is based on the *Advanced Microcontroller Bus Architecture* (AMBA) [3]. AMBA-based system includes a high performance system bus (AHB), on which the CPU, on-chip memory and other *Direct Memory Access* (DMA) devices reside. The overall architecture of the codec accelerator platform used in this thesis is shown in Fig 19.

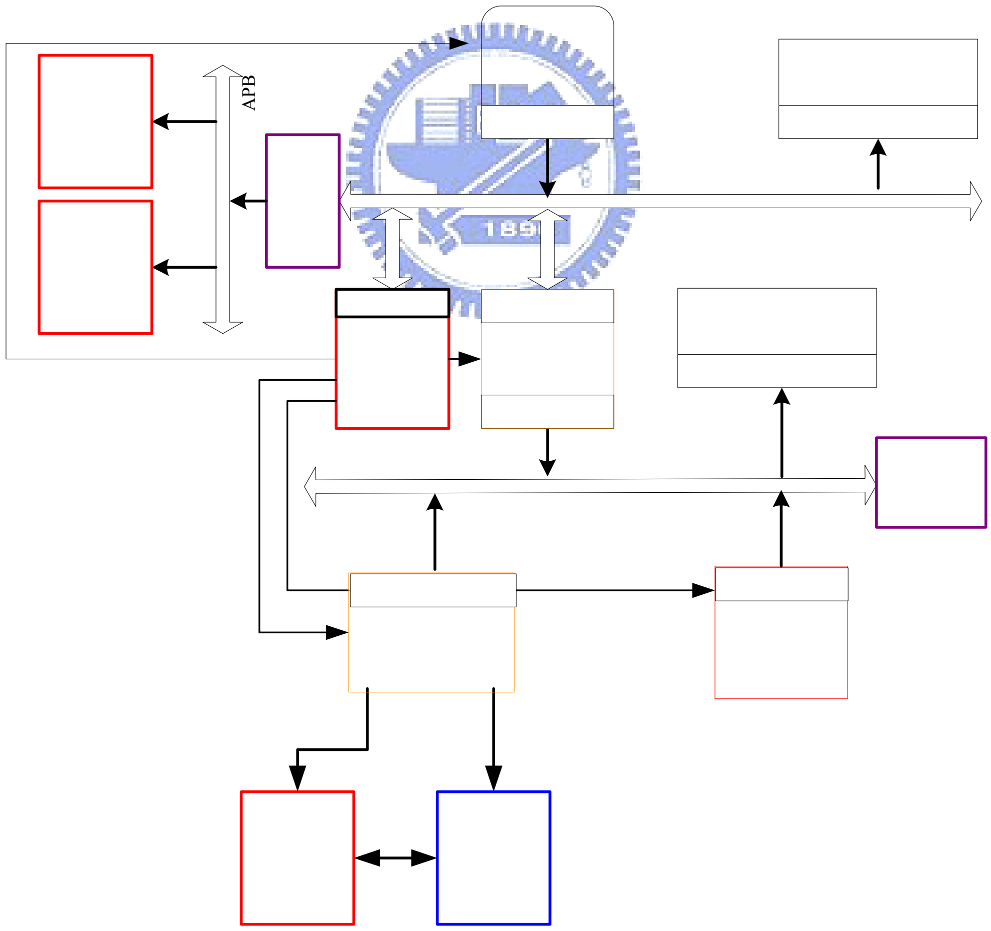


Fig 19. The Bus System Design for H.264 Video Accelerator Hardware Architecture

IRQ

Reg File

3.2. Introduction to AMBA

The *Advanced Microcontroller Bus Architecture* (AMBA) includes three distinct buses defined within the AMBA specification [3] :

- ♦ The *Advanced High-performance Bus* (AHB)
  - ■ High performance
  - ■ Pipelined operation
  - ■ Multiple bus masters
  - ■ Burst transfers
  - ■ Split transactions
- ♦ The *Advanced System Bus* (ASB)
  - ■ High performance
  - ■ Pipelined operation
  - ■ Multiple bus masters
- ♦ The *Advanced Peripheral Bus* (APB)
  - ■ Low power
  - ■ Latched address and control
  - ■ Simple interface
  - ■ Suitable for many peripherals

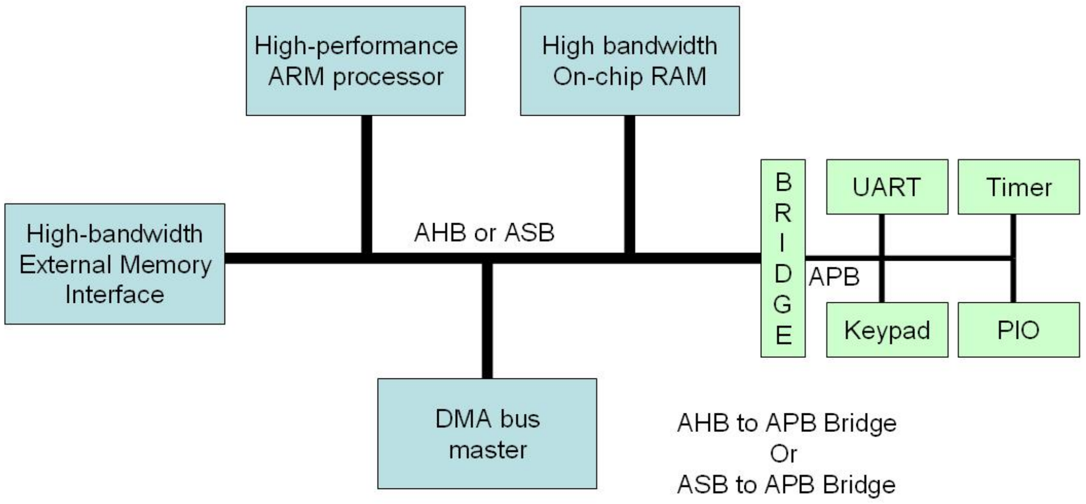

High-performance ARM processor
High bandwidth On-chip RAM
High-bandwidth External Memory Interface
AHB or ASB
BRIDGE
UART
Timer
APB
Keypad
PIO
DMA bus master
AHB to APB Bridge Or ASB to APB Bridge

Fig 20. A typical AMBA System

A typical AMBA system consists of a high-performance system backbone bus (AMBA AHB or AMBA ASB) and the APB is attached to a bridge transforming bus protocol from high performance to the low power and bandwidth bus, where most of the peripheral devices in the system are located, as shown in Fig 20. Since only AHB and APB are applied for bus system of the proposed video accelerator, in following paragraphs, AHB 3.2.1 and APB 3.2.2 protocols will be described with more details but ASB will be omitted.

3.2.1. AMBA AHB

AHB is a new generation of AMBA bus specialized for high-performance synthesizable designs. The high-performance protocol provides a multiple bus masters support and high-bandwidth operation.

A typical AMBA AHB system design contains the following components:

- AHB slave – An AHB slave responds to transfers initiated by bus masters within the system. When HSELx of the slave is active, it should respond to a bus transfer, as shown in Fig 22.
- AHB master – A bus master is able to initiate read and write operations to access data from slave by providing an address, data, and control information. Besides, there is only one bus master is allowed to use the bus at any one time, as shown in Fig 23.
- AHB arbiter – The bus arbiter ensures that only one bus master at any one time is allowed to use the bus to transfer data, as shown in Fig 24. Only one AHB arbiter is implemented on the bus.
- AHB decoder – An AHB decoder is used to decode the address of each transfer and provide a select signal for the slave that is involved in the transfer, as shown in Fig 25. The decoder in an AMBA system is used to perform a centralized address decoding function, which improves the portability of peripherals, by making them independent of the system memory map.

The AMBA AHB bus protocol is designed to be used with a central multiplexer interconnection scheme, as shown in Fig 21. All bus masters drive out the address and control signals and the arbiter determines which master can be granted to access the slaves.

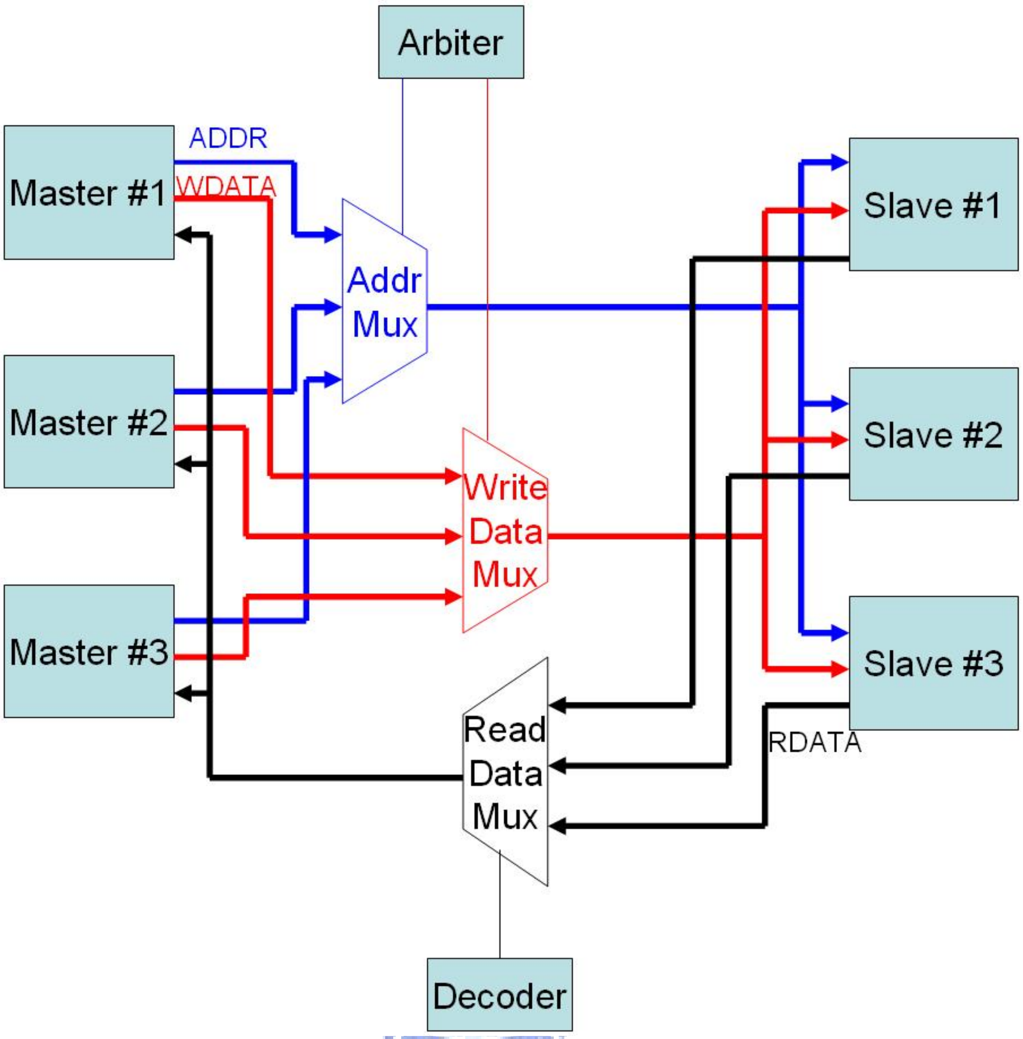


Fig 21. Multiplexer Iinterconnection

The AMBA AHB signals and brief descriptions are shown in Table 1.

Name	**Source**	**Description**
HCLK	Clock source	This clock times all bus transfers. All signal timings are related to the rising edge of **HCLK.**
HRESETn	Reset	The bus reset signal is active LOW and is used to reset the system and the bus. This is the only active LOW signal.
HADDR[31:0]	Master	The 32-bit system address bus.
HTRANS[1:0]	Master	Indicates the type of the current transfer.
HWRITE	Master	When HIGH this signal indicates a write transfer and when LOW a read transfer.
HSIZE[2:0]	Master	Indicates the size of the transfer.

HBURST[2:0]	Master	Indicates if the transfer forms part of a burst data transfer.
HPROT[3:0]	Master	The protection control signals provide additional information about a bus access and are primarily intended for use by any module that wishes to implement some level of protection.
HWDATA[31:0]	Master	The write data bus is used to transfer data from the master to the bus slaves during write operations.
HSELx	Decoder	Each AHB slave has its own slave select signal and this signal indicates that the current transfer is intended for the selected slave.
HRDATA[31:0]	Slave	The read data bus is used to transfer data from bus slaves to the bus master during read operations.
HREADY	Slave	When HIGH the **HREADY** signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer.
HRESP[1:0]	Slave	The transfer response provides additional information on the status of a transfer.
HBUSREQx	Master	A signal from bus master x to the bus arbiter which indicateds that the bus master requires the bus.
HLOCKx	Master	When HIGH this signal indicates that the master requires locked access to the bus and no other master should be granted the bus until this signal is LOW.
HGRANTx	Arbiter	A signal indicates bus master x is granted to use the bus.
HMASTER[3:0]	Arbiter	These signals from the arbiter indicate which bus master is currently performing a transfer.
HMASTLOCK	Arbiter	Indicates that the current master is performing a locked sequence of transfers.
HSPLITx[15:0]	Slave	This 16-bit bus is used by a slave to indicate to the arbiter which bus masters should be allowed to re-attempt a split transaction.

Table 1 AMBA AHB Signals and Arbitration Signals

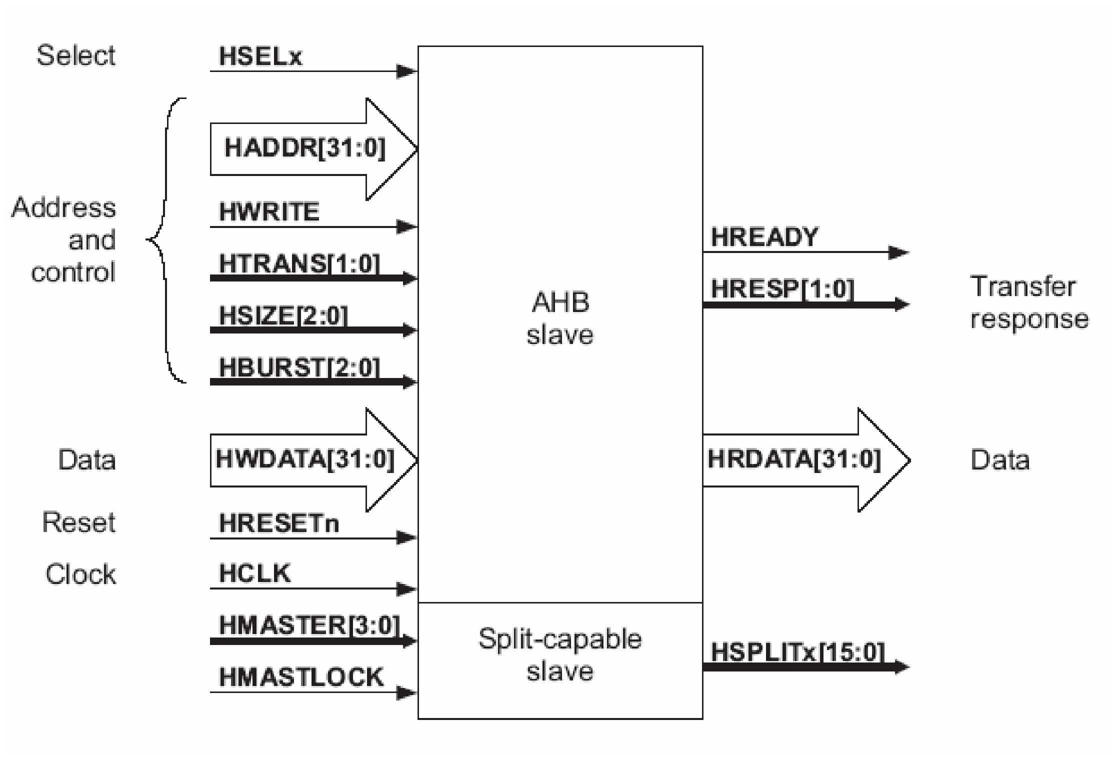


Fig 22. AHB Bus Slave Interface

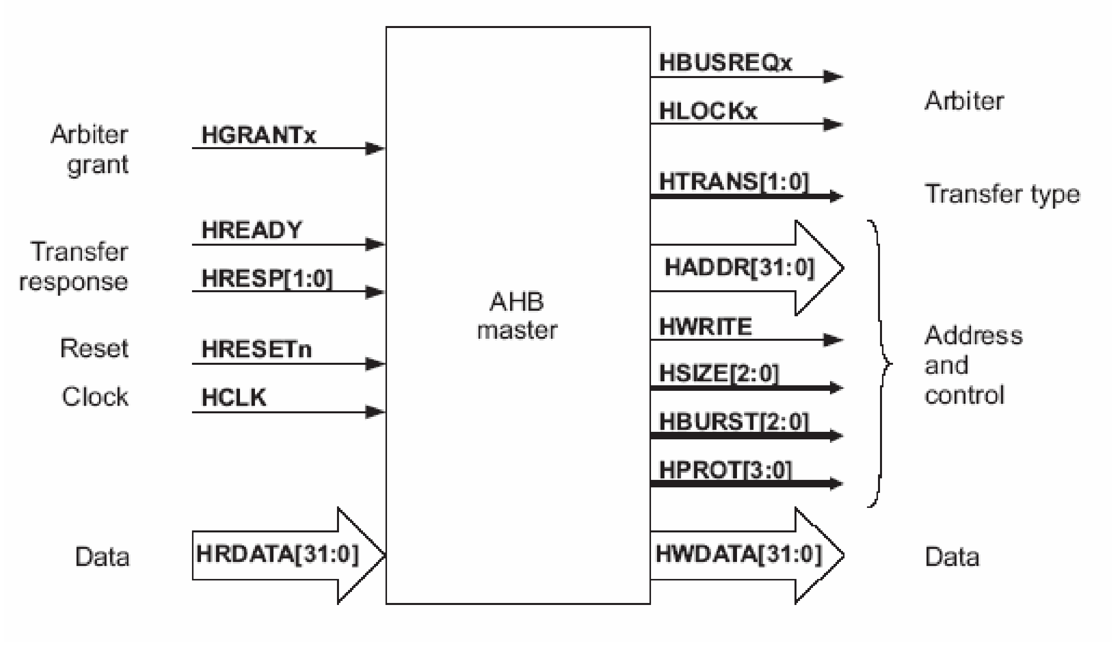


Fig 23. AHB Bus Master Interface

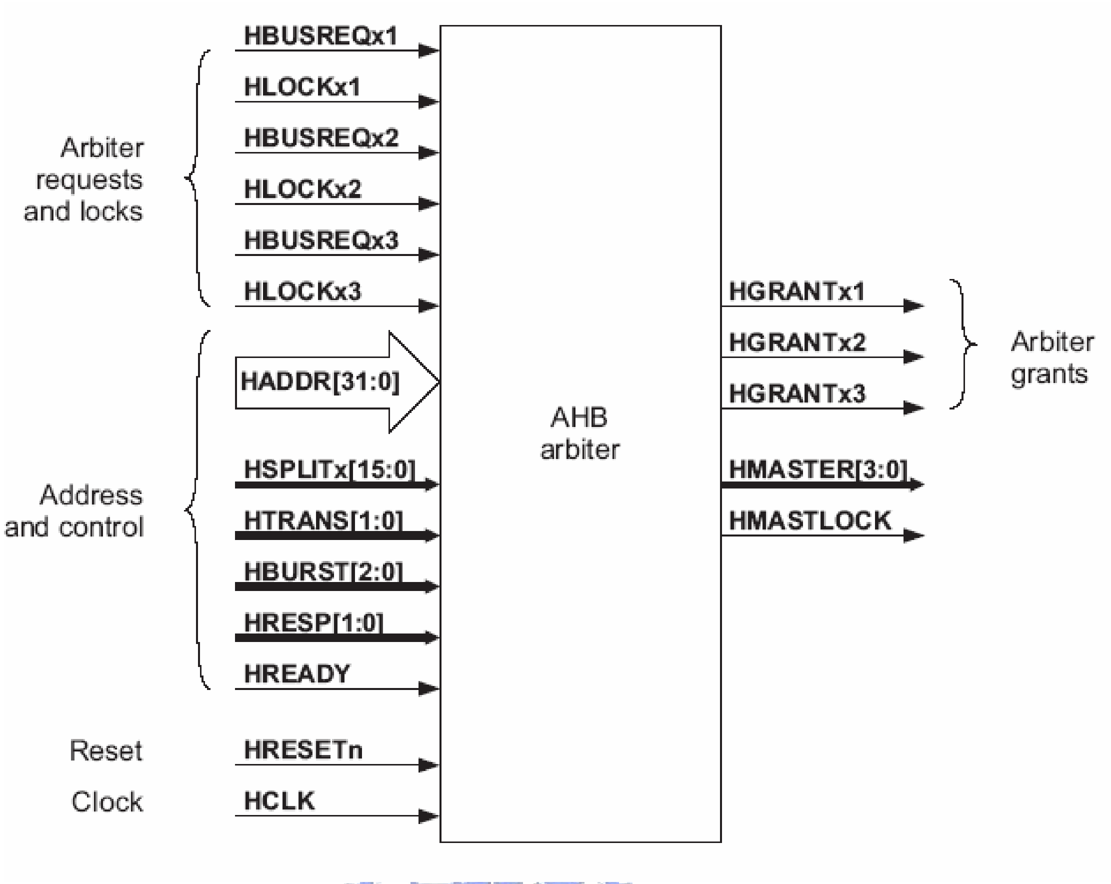


Fig 24. AHB Bus Arbiter Interface

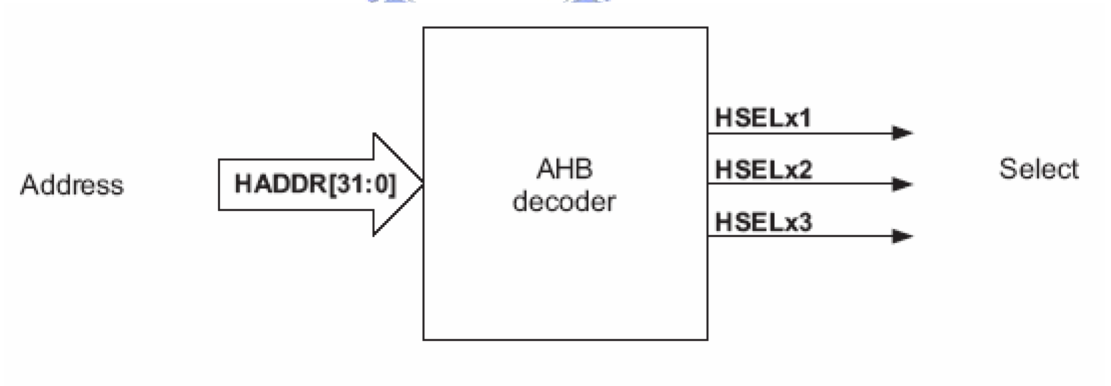


Fig 25. AHB Bus Decoder Interface

3.2.2. AMBA APB

The *Advanced Peripheral Bus* (APB) is part of the *Advanced Microcontroller Bus Architecture* (AMBA) hierarchy of buses and is optimized for minimal power consumption and reduced interface complexity. The AMBA APB should be used to interface to any peripherals which are low bandwidth and do not require the high performance of a pipelined bus interface. The data transfer on APB must follow a state diagram shown in Fig 26. The default state is IDLE for peripheral bus. When a data transfer is required the bus moves from IDLE state to SETUP state for one clock cycle and always move to ENABLE state on next rising edge of the clock. In ENABLE state, the enable signal, PENABLE, is asserted. The ENABLE state also only lasts for a single clock cycle and after this state the bus will return to the IDLE state if no further transfers are required. Alternatively, if another transfer is to follow then the bus will move directly to the SETUP state.

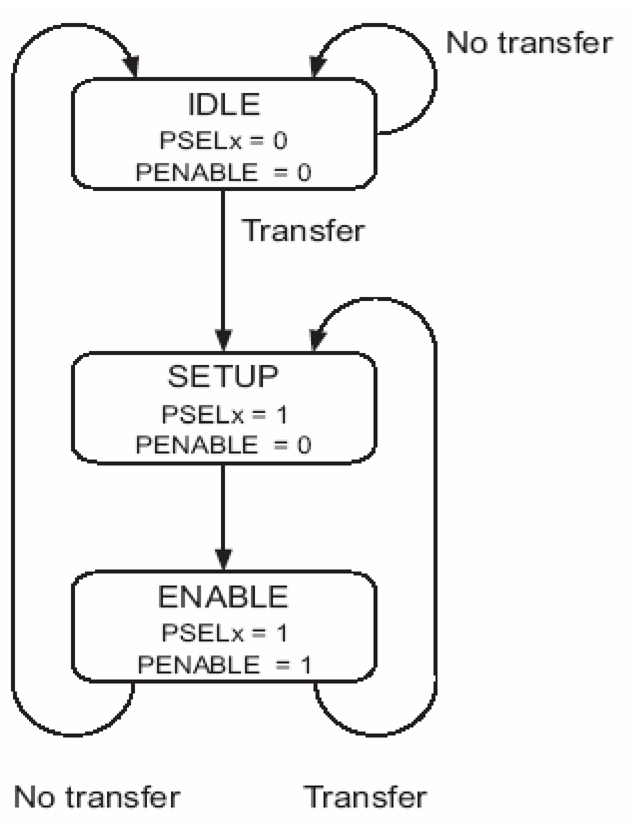


Fig 26. APB State Diagram

A typical AMBA APB system contains the following components:

- ♦ APB Bridge – The APB bridge is the only bus master on the AMBA APB and also a slave on the higher-level system bus, as shown in Fig 27. The bridge unit converts system bus transfers into APB transfers, which decodes the address and generates a peripheral select, PSELx, drives the data onto the APB for a write transfer and the APB data onto the system bus for a read transfer, and generates a timing strobe, PENABLE, for the transfer.

♦ APB Slave – For a write transfer, the data can be latched either on the rising edge of PCLK or the rising edge of PENABLE, when PSEL is HIGH. For read transfers the data can be driven on to the data bus when PWRITE is LOW and both PSELx and PENABLE are HIGH. While PADDR is used to determine which register should be read.

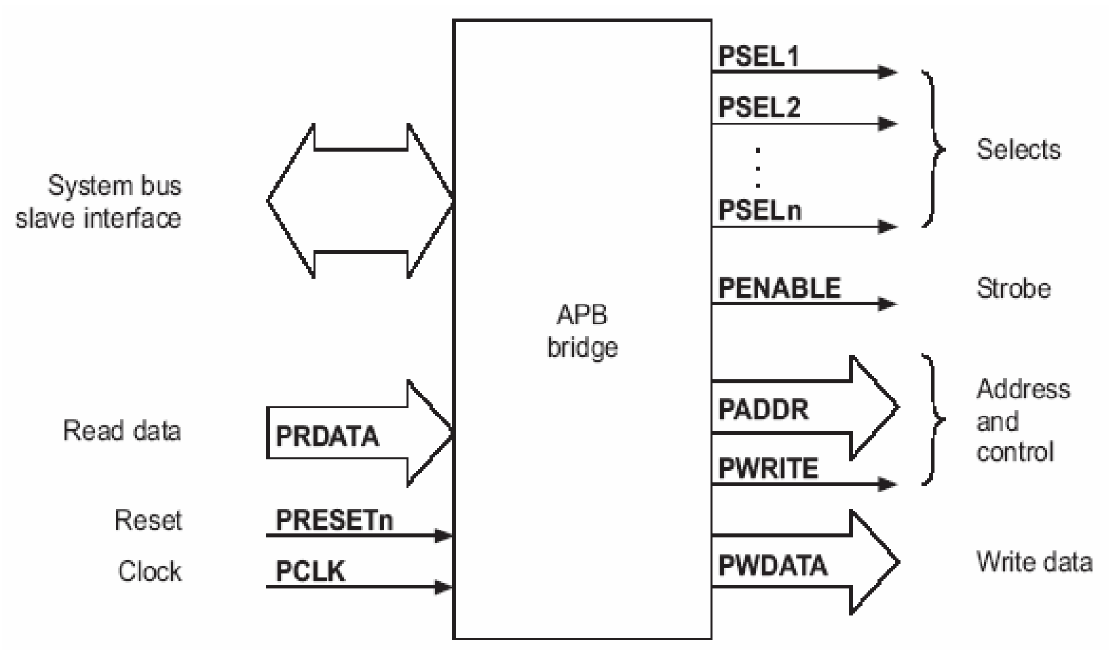


Fig 27. APB Bridge Interface Diagram.

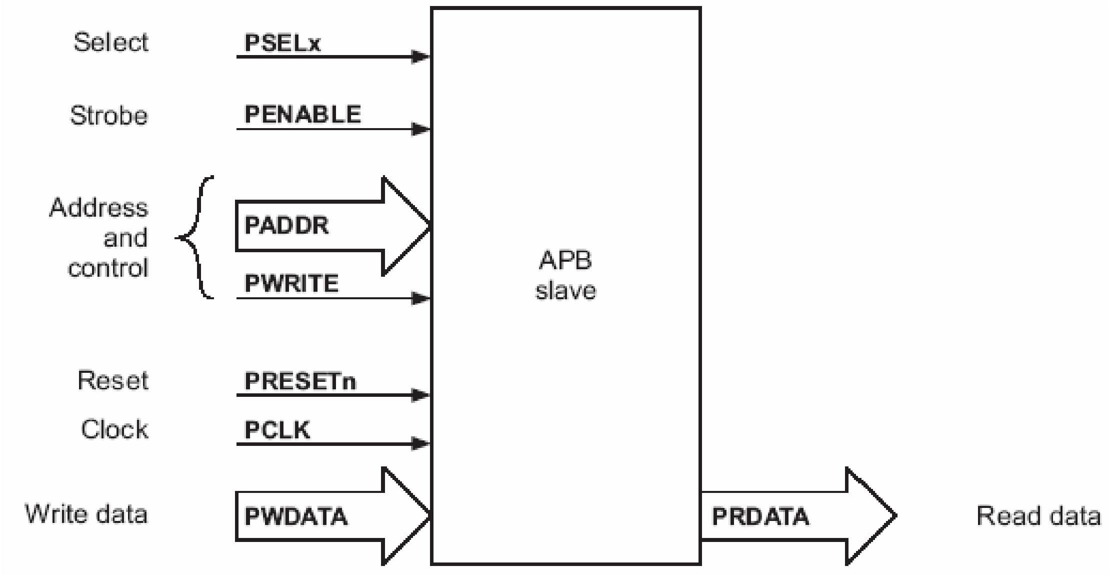


Fig 28. APB Slave Interface

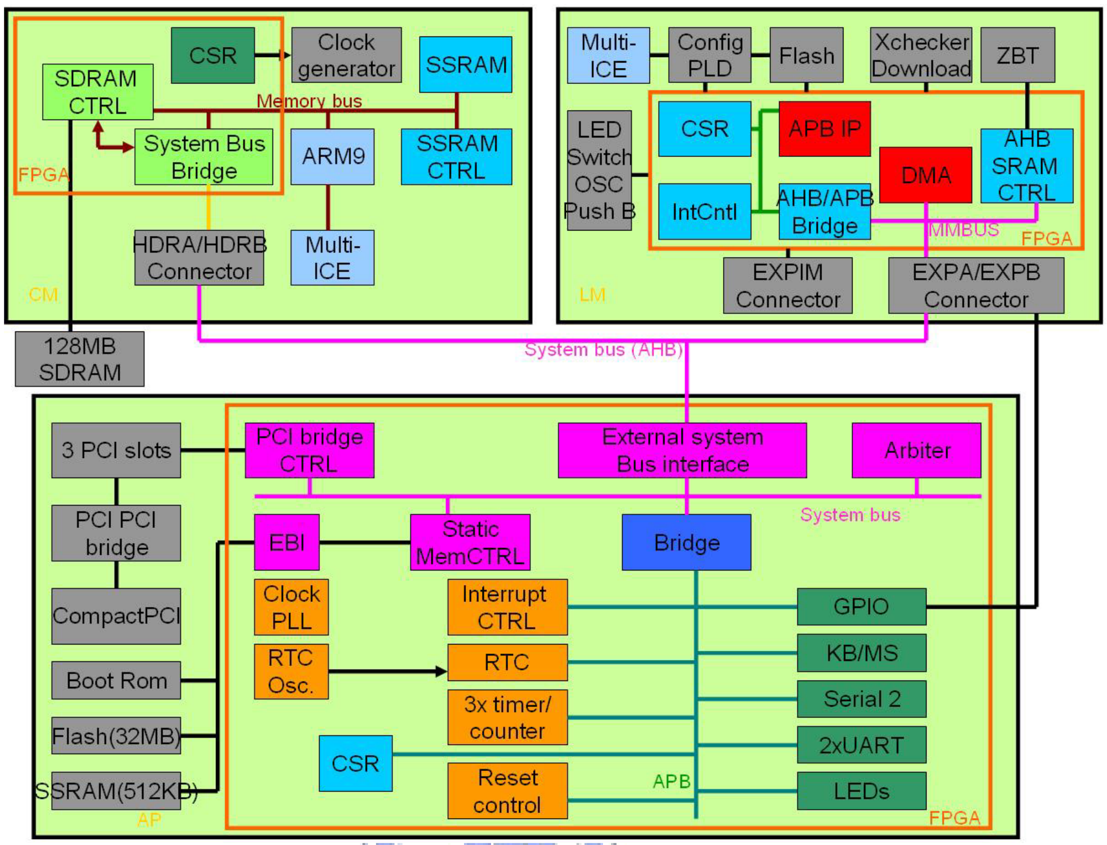


Fig 29. ARM Integrator Platform Architecture

3.3. Prototype Platform Description

The ARM Integrator is selected as the prototype platform for hardware/software co-design of the codec, as shown in Fig 29. There are three main modules in our platform, a motherboard (Integrator/AP) [13], an ARM9 core module (Integrator/CM920T) [14] and a Xilinx FPGA logic module (Integrator/LM-XCV2000E).[15].

3.4. System Description

There two masters, ARM processor and DMA controller, are on the *Advanced High-performance Bus* (AHB). These bus masters are able to initiate read and write operations by providing an address, control information, and/or data to AHB slaves. There are several slaves in the accelerator system, including a SDRAM used to store video frames and coding information, a AHB register file used to control the H.264 hardware accelerator blocks by the ARM core, and the APB register file and interrupt register file. The *Multi-Media Bus* (MMB) is a simplified AHB with 32-bit width, which is limited in transfer type, burst operation, control signal, and transfer response. Those are reduced to fit the specific design of the video accelerator, discussed in 3.6.

3.5. Software and Hardware Co-operation

3.5.1. S/H Co-design Feature

The memory map of the design in this thesis is shown in Fig 30.

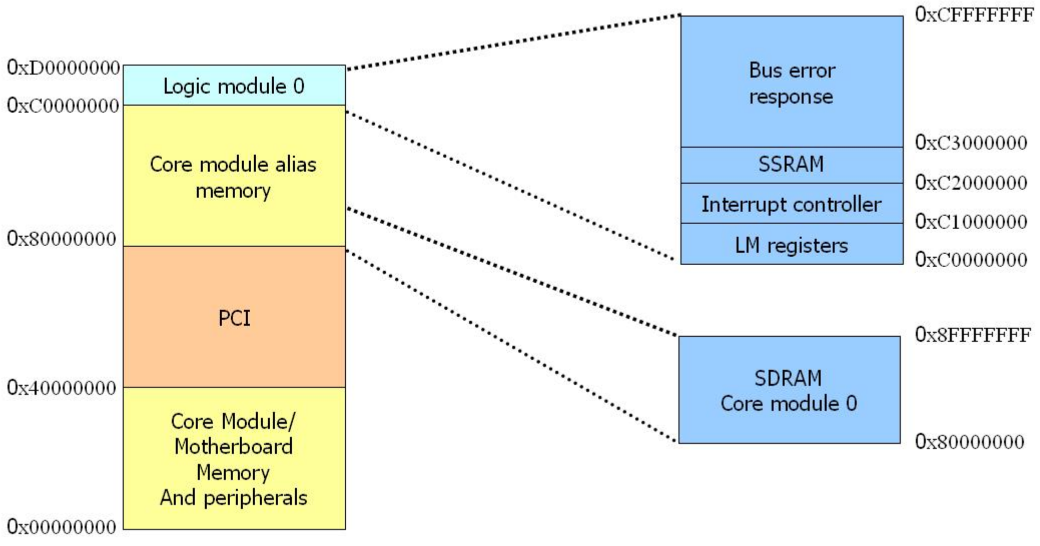


Fig 30. Memory Map

The communication method from "Core Module" (CM) to "Logic Module" (LM) is through LM registers which can be read or written by CM. On the other hand, the method from LM to CM is through interrupts asserted by LM so that CM must install an interrupt handle routine. The flowchart is shown in Fig 31.

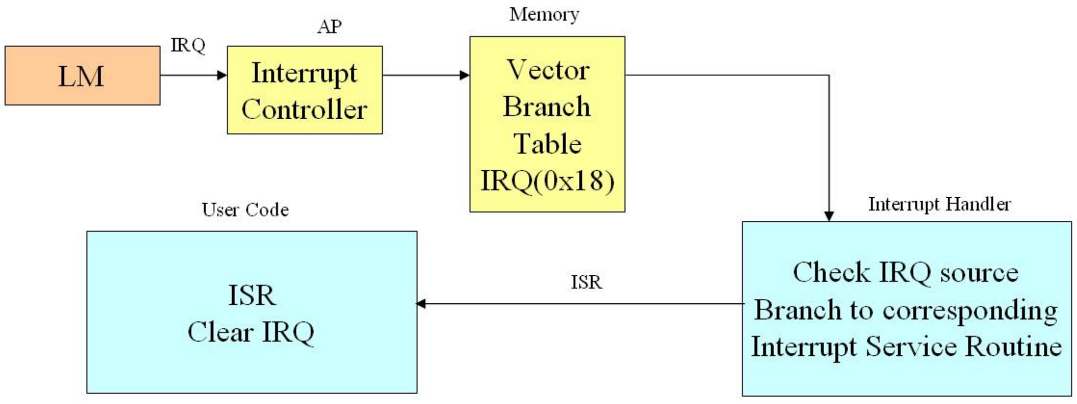


Fig 31. Flowchart for Installing Interrupt Handle Routine

3.5.2. S/H Co-operation with DMA

On MMB, there are three masters, DMA, In-Loop filter, and Motion Estimation Module. There is only one slave, ZBT SRAM on MMB. The procedure of video encoding or decoding in this system is described as follows.

- Step 1 – Video frames and coding information are stored in the SDRAM on the Integrator/CM by the processor.
- Step 2 – The processor enables the DMA by setting source address, target address, and data count in the AHB register file to move video frames and extra information from the SDRAM to the ZBT SRAM on the Integrator/LM
- Step 3 –the processor enables a video accelerator to process data in the ZBT SRAM
- Step 4 – After processing, video accelerator stores the results back to the ZBT SRAM and enables the DMA to move the results from the ZBT SRAM to the SDRAM.
- Step 5 – When the DMA finishes the data transfer, an interrupt will be triggered to notify the processor to get the results from SDRAM.

3.5.3. S/H Co-operation without DMA

If there is no DMA in the design, the processor must move the data from SDRAM to ZBT SRAM on LM by itself. Contention may raises when both the processor and the accelerators on MMB both try to access the ZBT SRAM. For the proposed system, the accelerators may stay in idle state until the processor enables them by writing to the ENABLE register in the AHB register file. Therefore shared accesses to the ZBT SRAM can be fully controlled by the processor to avoid data corruption. The detailed steps are listed as follows.

- ♦ Step 1 – Video frames and coding information are stored in the SDRAM on the Integrator/CM by the processor.
- ♦ Step 2 – The processor moves video frames and extra information from the SDRAM to ZBT SRAM on Integrator/LM.
- ♦ Step 3 – The processor enables a video accelerator to process data in the ZBT SRAM. Meanwhile, the processor will not access the ZBT SRAM until the video accelerator finishes its job and returns to the idle state.
- ♦ Step 4 – After processing, video accelerator stores the results back to the ZBT SRAM and issues an interrupt to notify the processor to retrieve the results from the ZBT SRAM to the SDRAM.

3.6. Details for MMB Design

In this thesis, a simplified bus protocol called MMB that is based on AHB is proposed as the bus of a generic codec accelerator platform. For example, MMB only provides 32-bit data transfer and INCR burst mode. The signals of MMB are shown in Table 2.

Signal Name	**Description**
MMCLK	System clock
MMRESETn	Reset signal
MMADDR	The 32-bit system address bus.
MMTRANS[1:0]	It only provides the transfer type, IDLE, SEQUENTIAL, and NONSEQUENTIAL.
MMWRITE	HIGH means a write transfer, and LOW means a read transfer.
MMWDATA[31:0]	The write data bus.
MMSELx	Each MMB slave has its own slave select signal.
MMRDATA[31:0]	The read data bus.
MMREADY	Indicates that a transfer has finished on the bus and maybe drives LOW to extend a transfer.
MMRESP[1:0]	The transfer response only provides OKAY and ERROR.
MMBUSREQx	The signal is asserted when a bus master requires the bus.

MMGRANTx	When this signal is HIGH, the master x is granted to use the bus.
MMMASTER[3:0]	These signals from the arbiter indicate which bus master is currently performing a transfer.

Table 2 MMBUS Signals and Arbitration Signals

There are seven basic components on MMB, listed as follows and the diagram is shown in Fig 21:

- ♦ MMMaster
  - ■ A bus master on MMB.
- ♦ MMArbiter
  - ■ A bus arbiter on MMB is up to a maximum of 4 bus masters, which will be discussed in 3.6.1 later.
- ♦ MMSlave
  - ■ A bus slave on MMB.
- ♦ MMMstMux
  - ■ A central multiplexer deal with the signals from all masters to slaves.
- ♦ MMMux
  - ■ A central multiplexer deal with the signals from all slaves to masters.
- ♦ MMDefaultSlave
  - ■ The default slave provides the MMREADY and the MMRESP outputs for MMMux when an address in the MMBUS memory map space is not covered by one of the peripherals present in the design.
- ♦ MMDecoder
  - ■ A decoder gives MMSELx module select outputs to the AHB system slaves and controls the read data multiplexer.

3.6.1. MMBus Arbitration

The arbitration algorithm used for MMBus is a method similar to token ring scheduling. One token is passed among four masters orderly, and the master which gets the token would be granted the right to use the bus. In initial state, the token is given to Master 1, and the master will pass the token to the next master if it does not require the bus. For example, if Master 1 requires the bus in the beginning, then it would get the token at first clock cycle and be granted. After that, Master 3 requires the bus and it will get the token after two clock cycles when Master 1 releases the bus if Master 2 does not require the bus. However, if Master 2 requires the bus before Master 1 releases the bus, it will be the next one getting the token. In this case, Master 3 has to wait until Master 2 releases the bus to pass the token to Master 3. In Fig 32, the lock means that if a master is locked, the token would not passes though it.

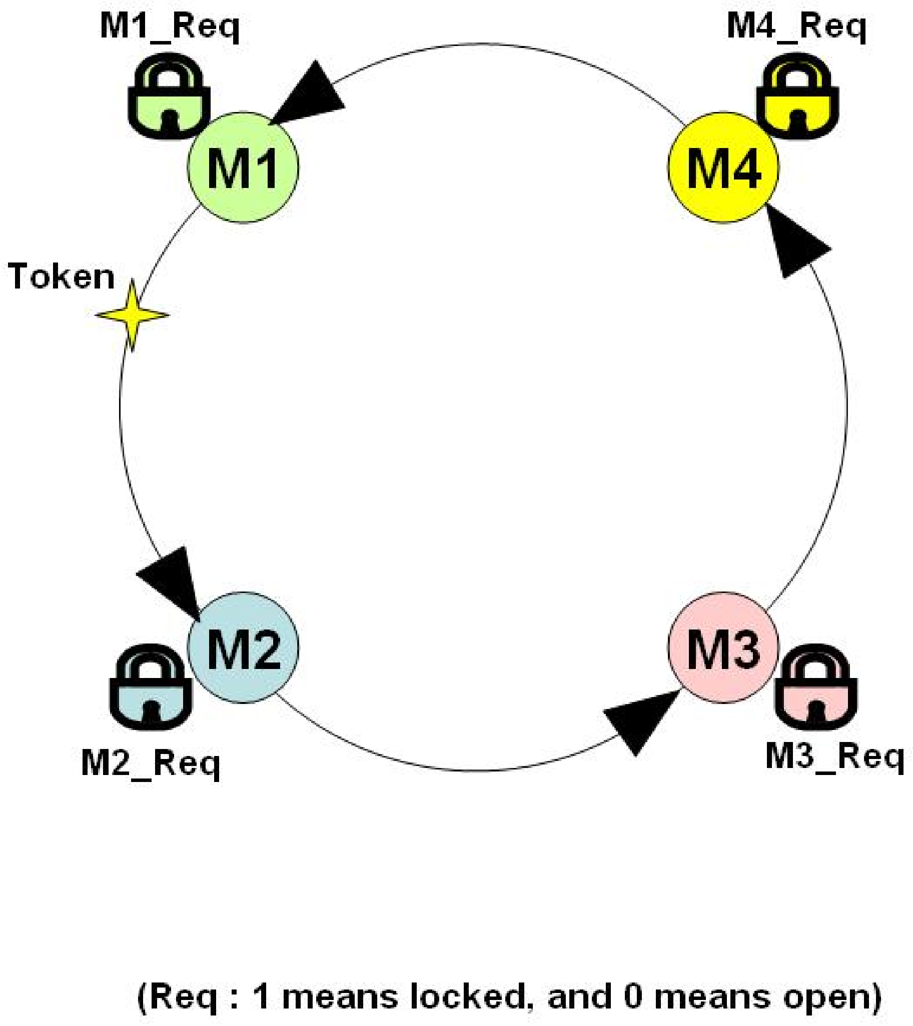


Fig 32. MMBus Arbiter with Token Ring Scheduling

4. Proposed VLSI Architecture of H.264 In-Loop Filter

4.1. Overview of the VLSI Architecture

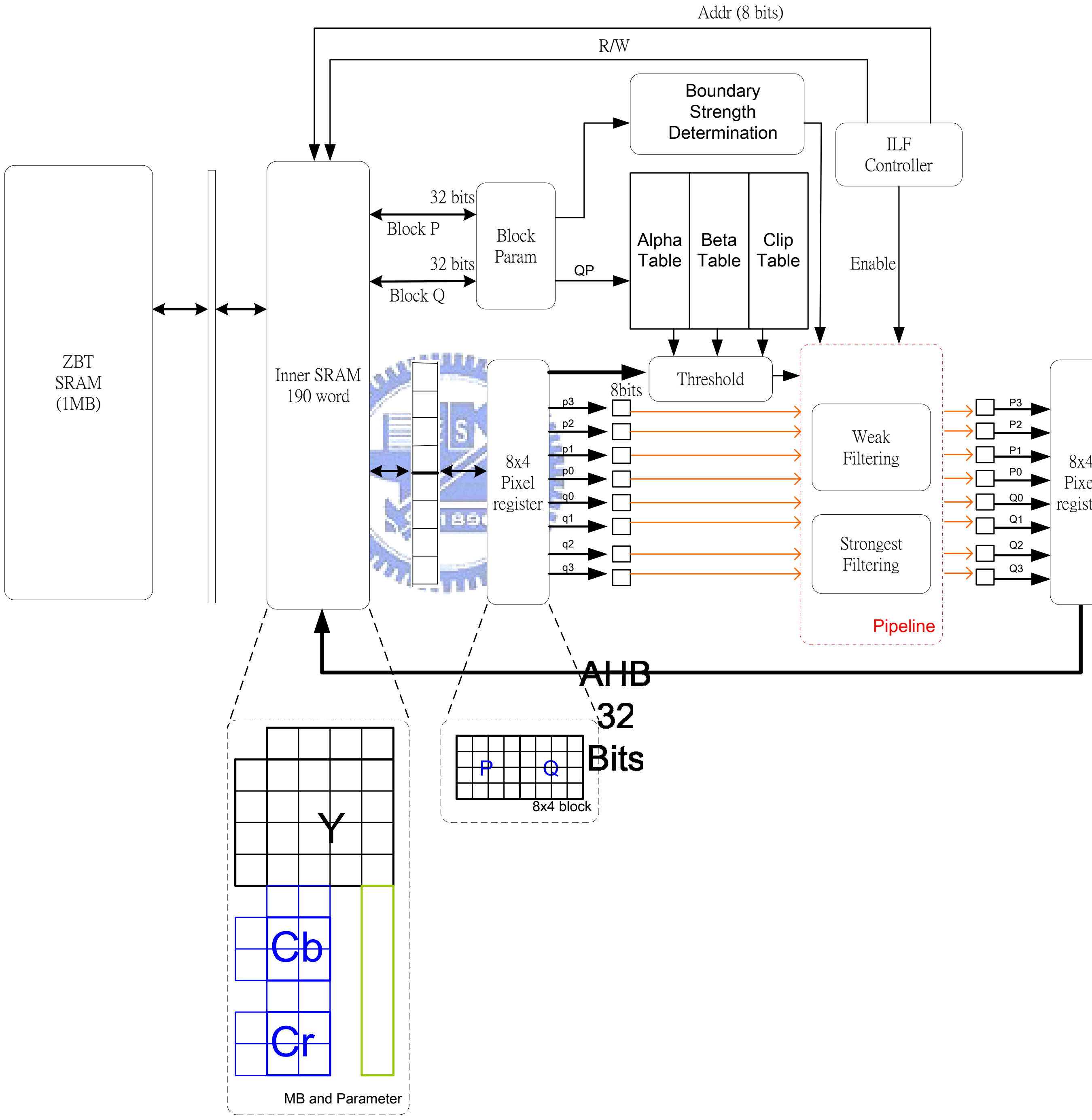


Fig 33. Hardware Design for In-Loop Filter

p3 p2 p1 p0

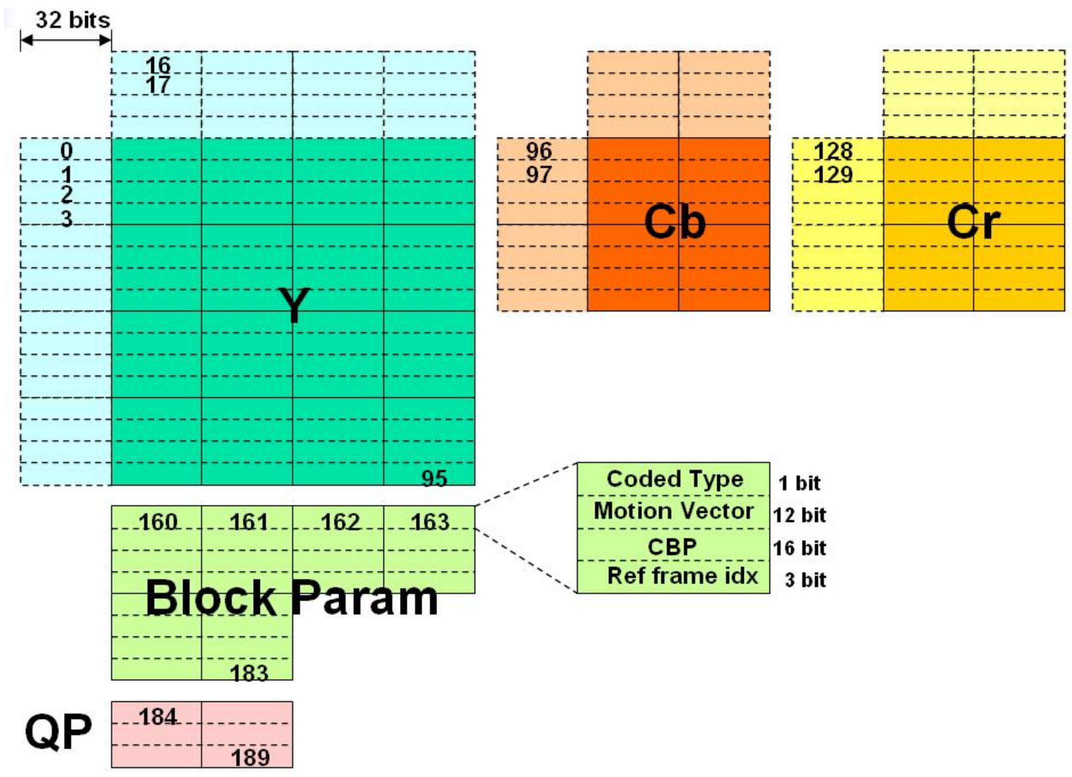


Fig 34. Organization of Dual Port Distributed SRAM Modules.

Fig 33 shows the proposed hardware architecture design of the In-loop filter for H.264/AVC. There are two RAM blocks. The first one is ZBT RAM for storing the video frames and coding information and the other one is a 764-byte dual-port distributed SRAM with 32-bit bandwidth synthesized by FPGA. The dual-port SRAM can perform two read operations per clock cycle and one write operation per clock cycle. It is used for temporarily saving one target macroblock and extra blocks at the left and top side of the target macroblock boundaries. In addition, it stores the block parameters containing coded type, motion vectors, code block pattern, and reference frame index, as shown in Fig 34. A 8x4-byte register file is designed to store unfiltered pixels in a group that can be filtered together since two 4x4 blocks have one identical boundary strength. In the deblocking filter module, 8 pixels are fed into the logic once and the filtering operations finish in two cycles. The filter adopts a two-stage pipeline design. As described in section 1.4, there are two types of filters, namely, a weak one with the boundary strength between 1 and 3 and a strong one with the strength equals 4. Zero strength represents that no filtering will be applied. The pipeline design of the "In-Loop filter" tries to evenly split the computational complexity into two stages so that the logic can run at higher frequencies with parallel computations of intermediate results.

4.2. Pipeline Design for In-Loop Filter

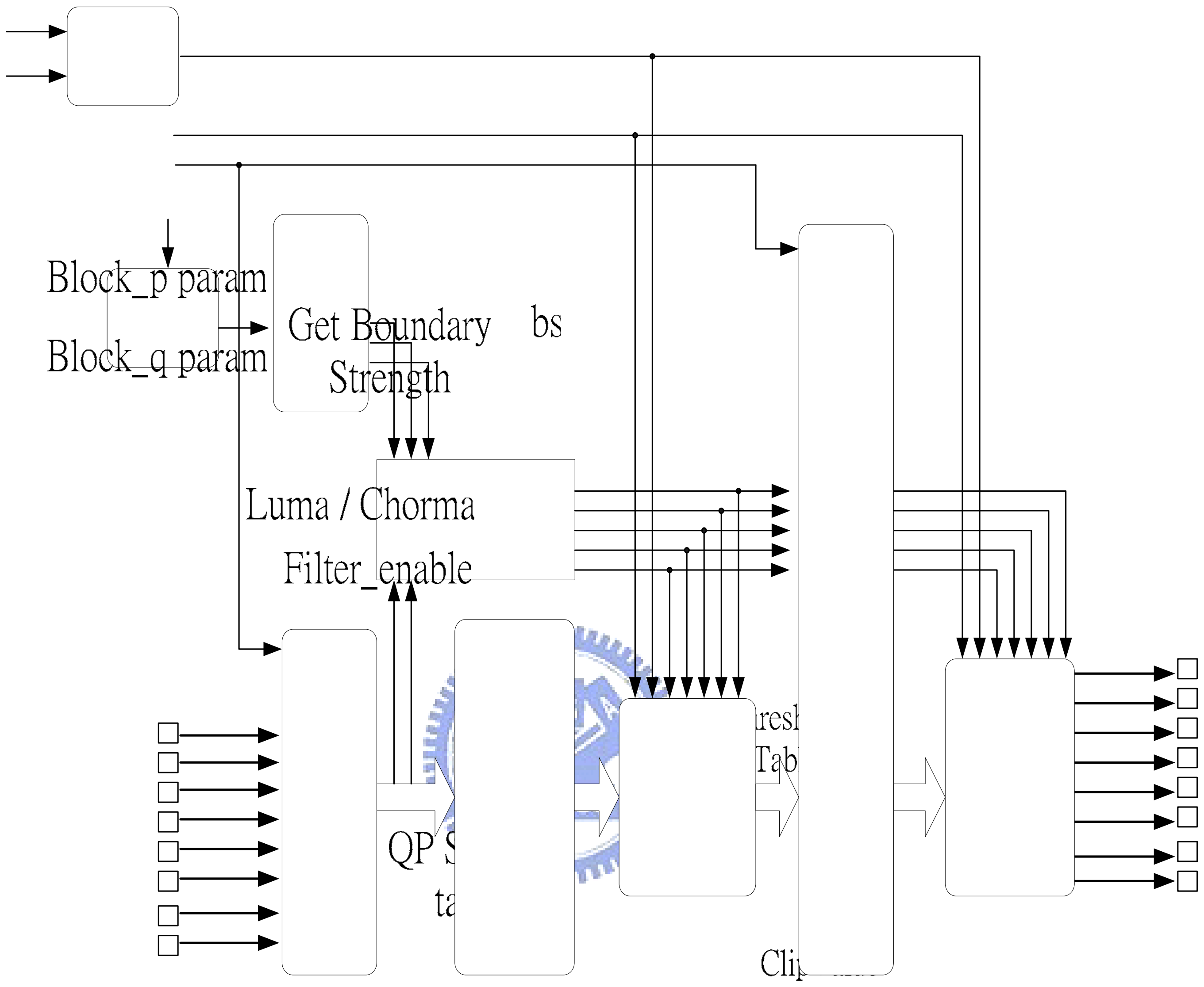
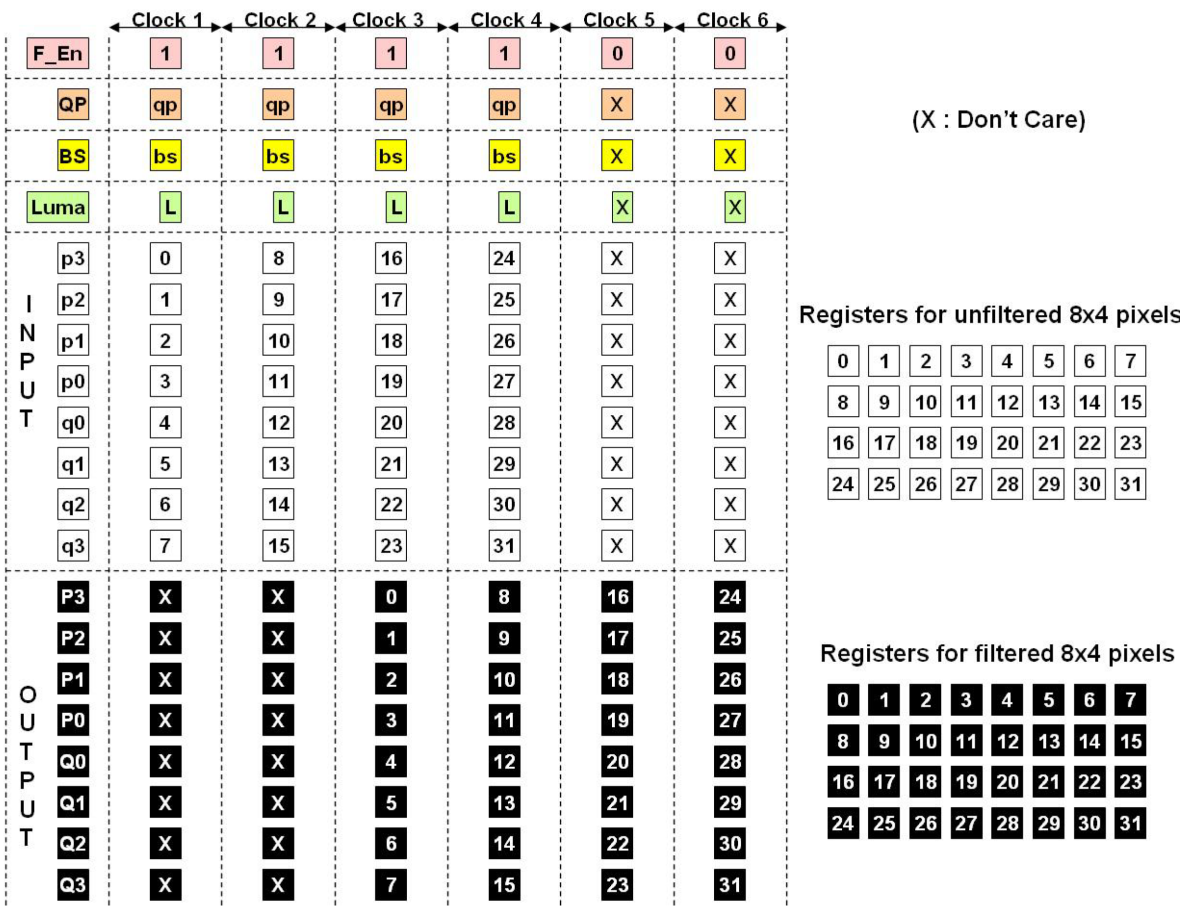


Fig 35. Two-stage Pipeline Design for In-Loop Filter

Fig 35 illustrates the pipeline design of the loop filter. The logic uses a 64-bit register file to store unfiltered pixels from SRAM input in stage one and a 209-bit register file to store computation results, control signals, and extra information in stage two. The QP scaling table and the threshold table which stores the average of quantization stepsizes of two blocks are stored in ROMs. Since the control signals including boundary strength, luma/chroma, and the signal, filter_en, are independent to pipeline stages, they are connected to the two computational components directly. Since the entire system runs 50MHz, it has to balance the time delay in each of the two stages and fit the 50MHz clock rate. In order to do so, it is necessary to find the critical path of calculations and divide it evenly to balance the computation in pipeline in order to raise the clock rate.

The following paragraph discusses the pipeline design for operations of in-loop filter process with more details.



		Clock 1	Clock 2	Clock 3	Clock 4	Clock 5	Clock 6
	F_En	1	1	1	1	0	0
	QP	qp	qp	qp	qp	X	X
	BS	bs	bs	bs	bs	X	X
	Luma	L	L	L	L	X	X
INPUT	p3	0	8	16	24	X	X
	p2	1	9	17	25	X	X
	p1	2	10	18	26	X	X
	p0	3	11	19	27	X	X
	q0	4	12	20	28	X	X
	q1	5	13	21	29	X	X
	q2	6	14	22	30	X	X
	q3	7	15	23	31	X	X
OUTPUT	P3	X	X	0	8	16	24
	P2	X	X	1	9	17	25
	P1	X	X	2	10	18	26
	P0	X	X	3	11	19	27
	Q0	X	X	4	12	20	28
	Q1	X	X	5	13	21	29
	Q2	X	X	6	14	22	30
	Q3	X	X	7	15	23	31

(X : Don't Care)

Registers for unfiltered 8x4 pixels

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

Registers for filtered 8x4 pixels

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

Fig 36. The Operating Cycles of Every Register for Filtering

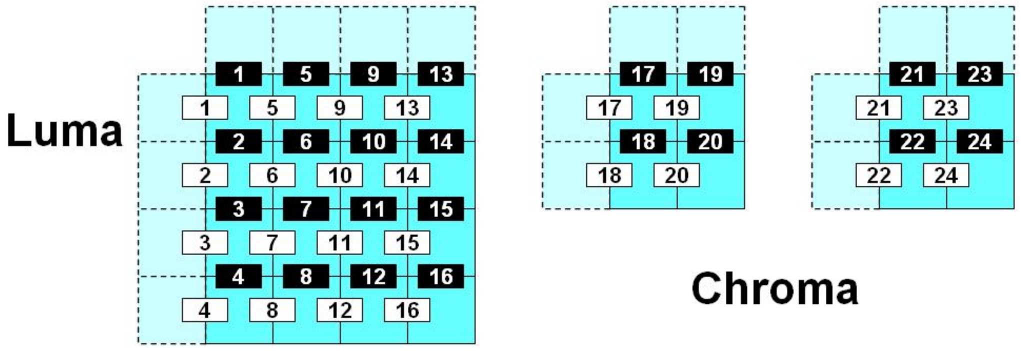


Fig 37. The Processing Order of One MB Filtering

The total operations of the in-loop filter are listed in Table 3 .

	Strong Filtering (Bs=4)	Weak Filtering (0<Bs<4)
Luminance	Q0 = (p1 + 2*p0 + 2*q0 + 2*q1 + q2 + 4) >> 3	Delta = Clip(-C, C, (((q0 – p0) << 2 + (p1 – q1) + 4) >> 3))
	Q0 = (2*q1 + q0 + p1 + 2) >> 2	
	Q1 = (p0 + q0 + q1 + q2 + 2) >> 2	P_0 = Clip(0, 255, p_0 + Delta)
	Q2 = (2*q3 + 3*q2 + q1 + q0 + p0 + 4) >> 3	
	P0 = (p2 + 2*p1 + 2*p0 + 2*q0 + q1 + 4) >> 3	P_1 = p_1 + Clip(– C0, C0, (p_2 + ((p_0 + q_0 + 1) >> 1) – (p_1 << 1)) >> 1)
	P0 = (2*p1 + p0 + q1 + 2) >> 2	
	P1 = (p2 + p1 + p0 + q0 + 2) >> 2	Q_0 = Clip(0, 255, q_0 – Delta)
	P2 = (2*p3 + 3*p2 + p1 + p0 + q0 + 4) >> 3	
		Q_1 = q_1 + Clip(– C0, C0, (q_2 + ((p_0 + q_0 + 1) >> 1) – (q_1 << 1)) >> 1)
Chrominance	P0 = ((p1 + q1 + p1 + p0 + 2)) >> 2	Delta = Clip(– C, C, (((q_0 – p_0) << 2 +
	Q0 = ((q1 + q0 + p1 + q1 + 2)) >> 2	(p_1 – q_1) + 4) >> 3))
		P_0 = Clip(0, 255, p_0 + Delta)
		Q_0 = Clip(0, 255, q_0 – Delta)

Table 3 Complete In-Loop Filter Operations

According to Table 3, we can design two pipeline stages with almost balanced computation cycles. In addition, all the multiplications are replaced by shift and add to increase speed. The design of the pipeline stages was not achieved through detail analysis of computation time of each shift/add operation; instead, it is achieved by putting roughly the same amount of operations in the computational components of each stage relatively. The filtering for each 8x4 pixels takes 6 clock cycles and the operating period of each register is shown in Fig 36, where X means “Don’t Care”. F_En is a 1-bit register that specifies if the filter module is enabled or not. And Luma is also a 1-bit register that specifies whether the input pixels is from the luma channel or the chroma channel. The processing order of one macroblock filtering with luminance and chrominance is shown in Fig 37. The design of the computational component of each stage is shown in Fig 38 and Fig 39.

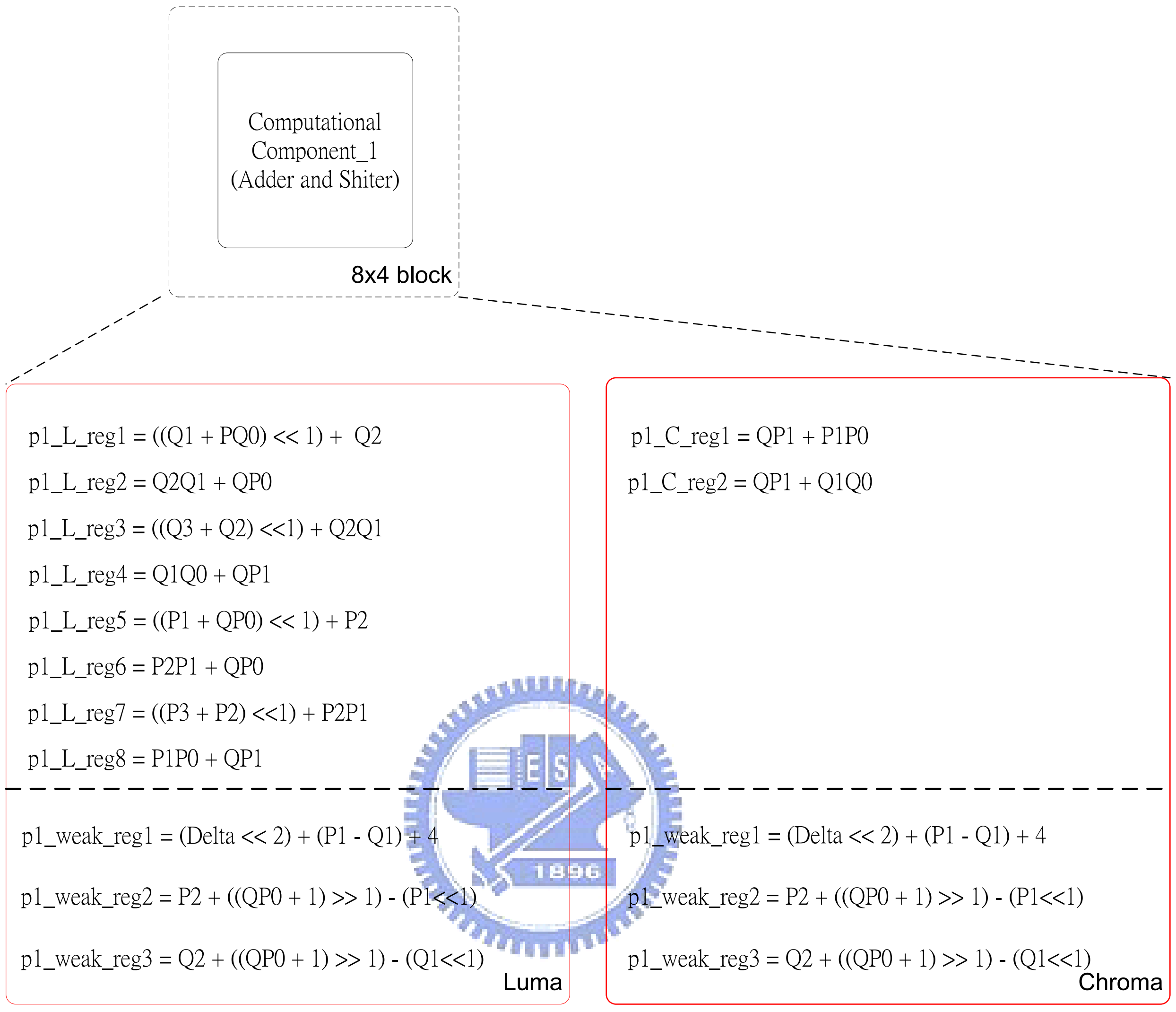

Computational Component_1 (Adder and Shiter)
8x4 block
p1_L_reg1 = ((Q1 + PQ0) << 1) + Q2
p1_L_reg2 = Q2Q1 + QP0
p1_L_reg3 = ((Q3 + Q2) <<1) + Q2Q1
p1_L_reg4 = Q1Q0 + QP1
p1_L_reg5 = ((P1 + QP0) << 1) + P2
p1_L_reg6 = P2P1 + QP0
p1_L_reg7 = ((P3 + P2) <<1) + P2P1
p1_L_reg8 = P1P0 + QP1
p1_weak_reg1 = (Delta << 2) + (P1 - Q1) + 4
p1_weak_reg2 = P2 + ((QP0 + 1) >> 1) - (P1<<1)
p1_weak_reg3 = Q2 + ((QP0 + 1) >> 1) - (Q1<<1)
Luma
p1_C_reg1 = QP1 + P1P0
p1_C_reg2 = QP1 + Q1Q0
p1_weak_reg1 = (Delta << 2) + (P1 - Q1) + 4
p1_weak_reg2 = P2 + ((QP0 + 1) >> 1) - (P1<<1)
p1_weak_reg3 = Q2 + ((QP0 + 1) >> 1) - (Q1<<1)
Chroma

Fig 38. Computational Component of Stage 1 in Pipeline Design

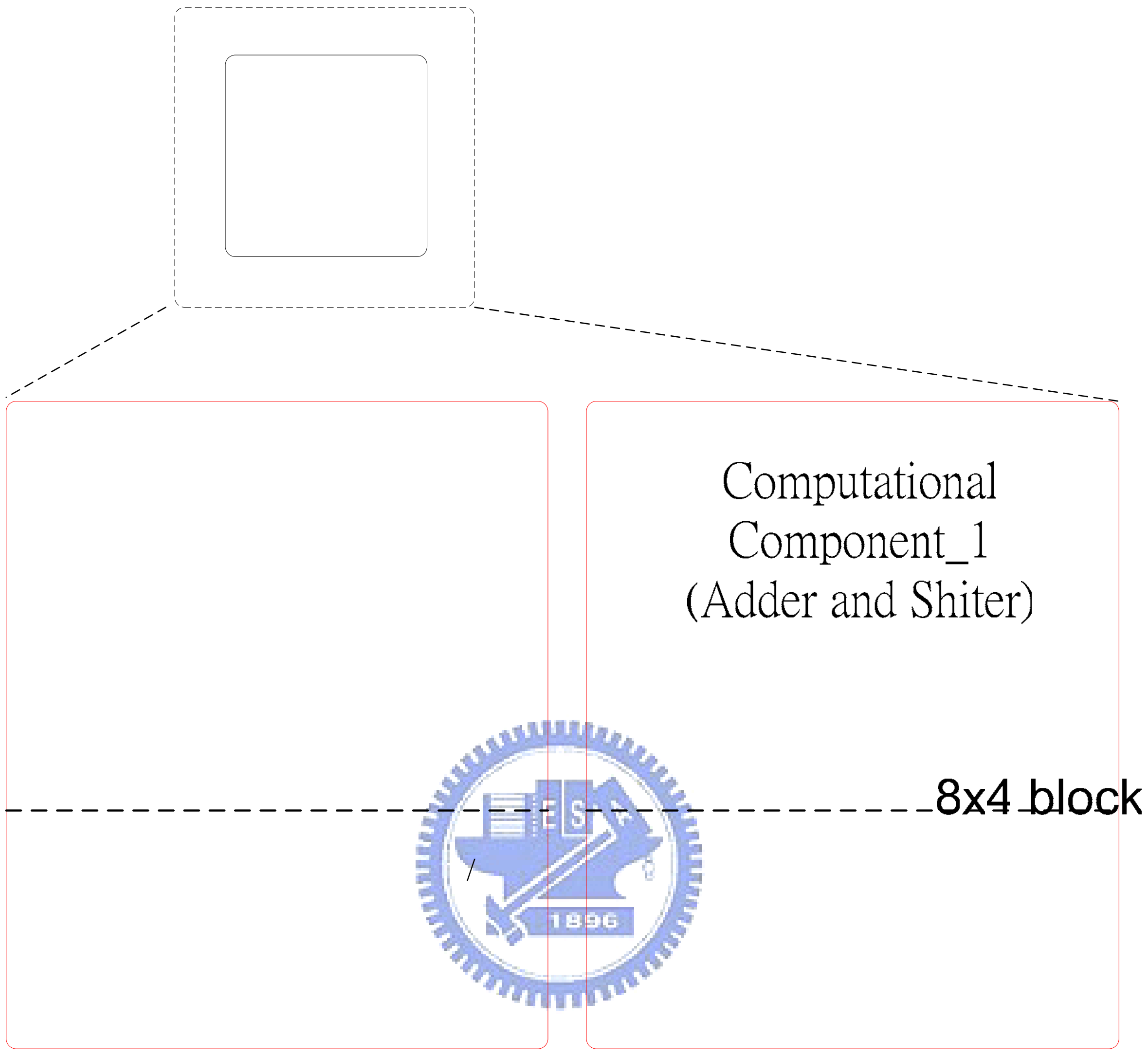


Fig 39. Computational Component of Stage 2 in Pipeline Design

$$Q0 = (P1 + p1\_L\_reg1 + 4) >> 3$$

$$Q1 = (p1\_L\_reg2 + 2) >> 2$$

$$Q2 = (p1\_L\_reg3 + QP0 + 4) >> 3$$

$$P0 = (Q1 + p1\_L\_reg5 + 4) >> 3$$

$$P1 = (p1\_L\_reg6 + 2) >> 2$$

$$P2 = (p1\_L\_reg7 + QP0 + 4) >> 3$$

$$Q0 = (p1\_L\_reg4 + 2) >> 2$$

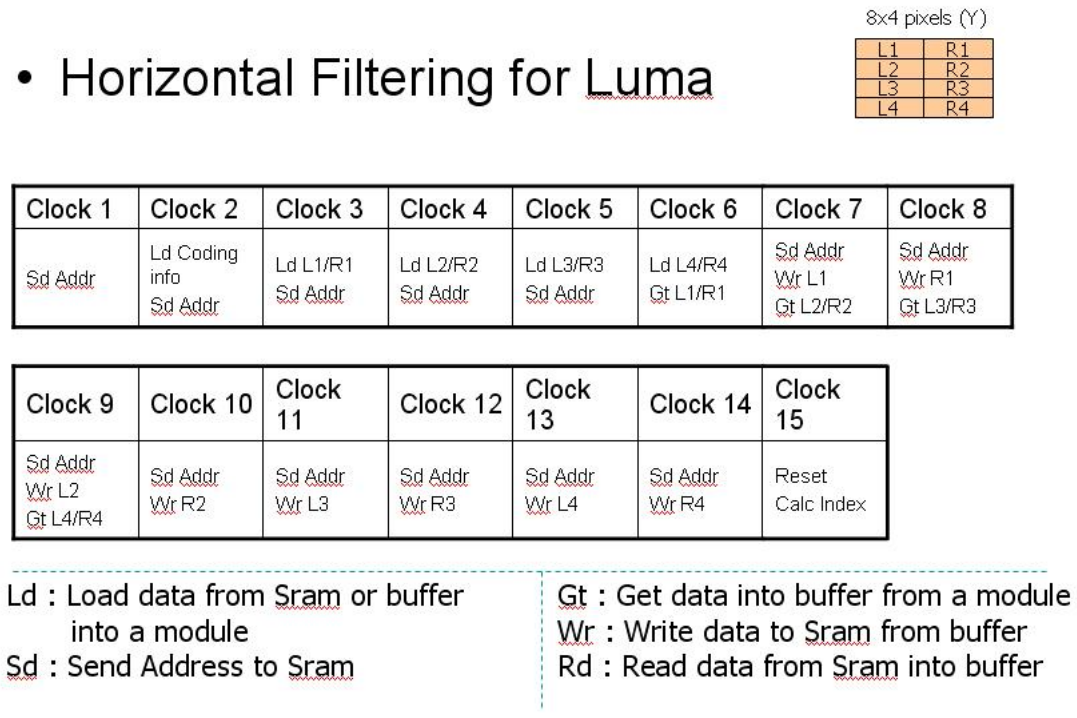
$$P0 = (p1\_L\_reg8 + 2) >> 2$$

5. Simulation Results

The System clock rate in the design is targeting at 50MHz. The whole architecture is coded in VHDL and the circuit is synthesized for an FPGA, the VirtexE XCV2000E, using Xilinx ISE. The total gate count is 120K. The filtering for every 8x4 pixels takes 6 clock cycles and one extra cycle for getting boundary strength. It spends 191*3 clock cycles loading unfiltered pixels and coding information from outside SRAM into inner SRAM. For filtering one macroblock, it takes 15 cycles to filter vertical edges of 8x4 luminance pixels and 27 cycles to filter vertical edges of

$$P0 = P0 + Clip(-C0, C0, (p1\_weak\_reg1) >> 3)$$

8x4 chrominance pixels. Otherwise, it takes 22 cycles to filter horizontal edges of 8x4 luminance pixels and 30 cycles to filter horizontal edges of 8x4 chrominance pixels. The operating cycles for every filtering process is depicted in Fig 40, Fig 41, Fig 42, and Fig 43. It also takes 190*3 clock cycles to store back the results from inner SRAM to outside SRAM. The sum of total clock cycles to filter 30 CIF (352x288 pixels) frames is 22 * 18 * 30 * (16 * 15 + 27 * 4 +22 * 16 + 30 * 4) + 3*190*2, so the time is 0.195 second for filtering when targeting clock rate is 50MHz. It is easy to achieve real-time deblocking with CIF 30fps.



• Horizontal Filtering for Luma

8x4 pixels (Y)

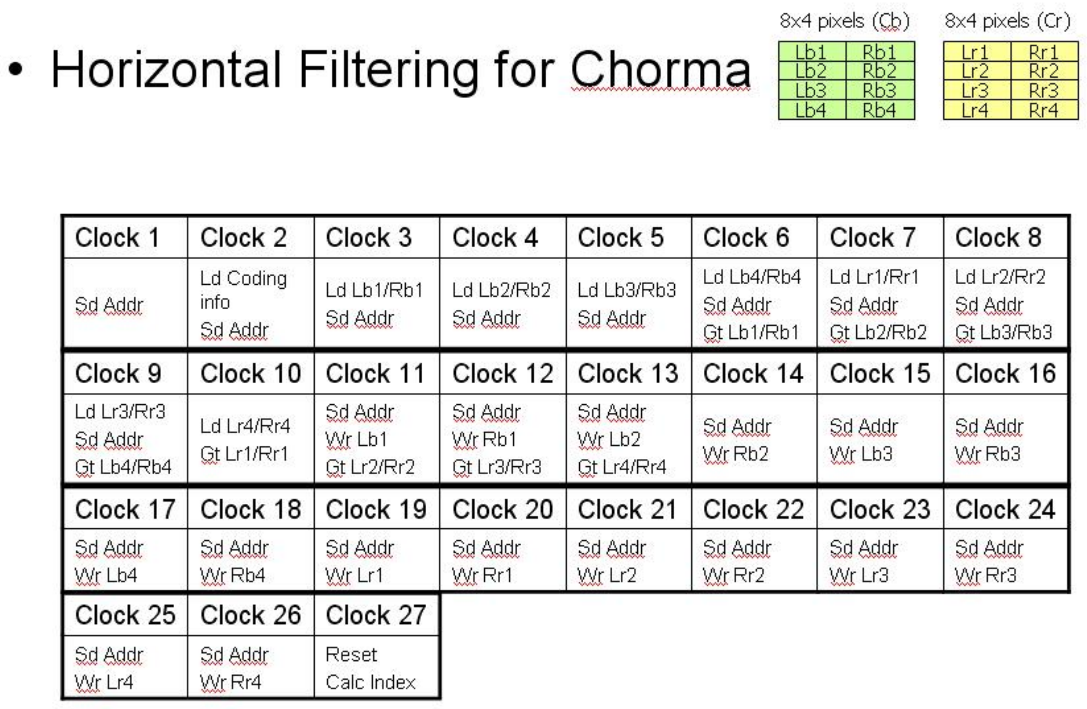
L1	R1
L2	R2
L3	R3
L4	R4

Clock 1	Clock 2	Clock 3	Clock 4	Clock 5	Clock 6	Clock 7	Clock 8
Sd Addr	Ld Coding info Sd Addr	Ld L1/R1 Sd Addr	Ld L2/R2 Sd Addr	Ld L3/R3 Sd Addr	Ld L4/R4 Gt L1/R1	Sd Addr Wr L1 Gt L2/R2	Sd Addr Wr R1 Gt L3/R3

Clock 9	Clock 10	Clock 11	Clock 12	Clock 13	Clock 14	Clock 15
Sd Addr Wr L2 Gt L4/R4	Sd Addr Wr R2	Sd Addr Wr L3	Sd Addr Wr R3	Sd Addr Wr L4	Sd Addr Wr R4	Reset Calc Index

Ld : Load data from Sram or buffer into a module	Gt : Get data into buffer from a module
Sd : Send Address to Sram	Wr : Write data to Sram from buffer
	Rd : Read data from Sram into buffer

Fig 40. Operating Cycles for Vertical Edges of 8x4 Luma Pixels



• Horizontal Filtering for Chorma

8x4 pixels (Cb)

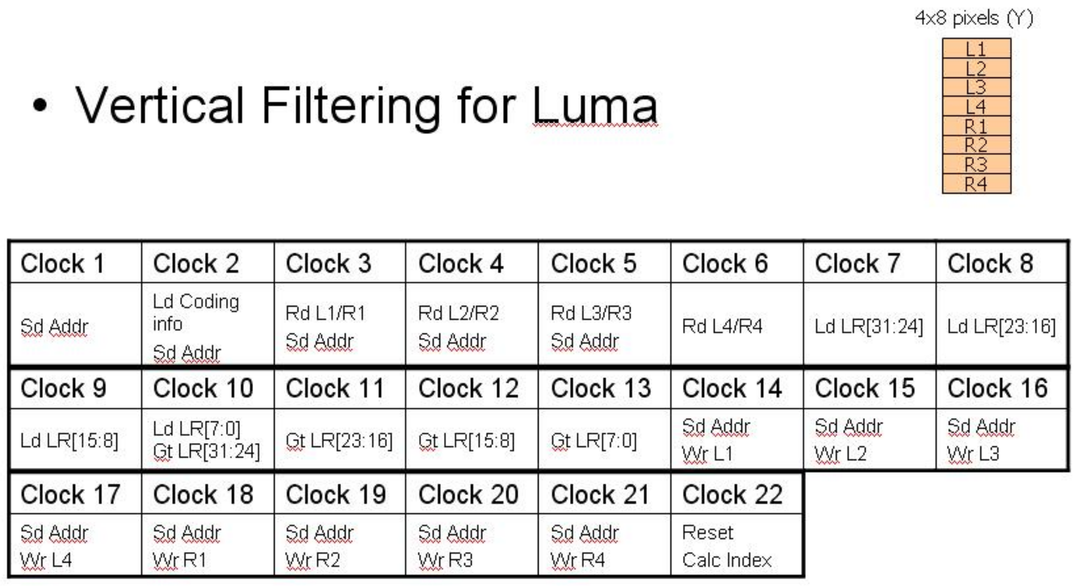
Lb1	Rb1
Lb2	Rb2
Lb3	Rb3
Lb4	Rb4

8x4 pixels (Cr)

Lr1	Rr1
Lr2	Rr2
Lr3	Rr3
Lr4	Rr4

Clock 1	Clock 2	Clock 3	Clock 4	Clock 5	Clock 6	Clock 7	Clock 8
Sd Addr	Ld Coding info Sd Addr	Ld Lb1/Rb1 Sd Addr	Ld Lb2/Rb2 Sd Addr	Ld Lb3/Rb3 Sd Addr	Ld Lb4/Rb4 Sd Addr Gt Lb1/Rb1	Ld Lr1/Rr1 Sd Addr Gt Lb2/Rb2	Ld Lr2/Rr2 Sd Addr Gt Lb3/Rb3
Clock 9	**Clock 10**	**Clock 11**	**Clock 12**	**Clock 13**	**Clock 14**	**Clock 15**	**Clock 16**
Ld Lr3/Rr3 Sd Addr Gt Lb4/Rb4	Ld Lr4/Rr4 Gt Lr1/Rr1	Sd Addr Wr Lb1 Gt Lr2/Rr2	Sd Addr Wr Rb1 Gt Lr3/Rr3	Sd Addr Wr Lb2 Gt Lr4/Rr4	Sd Addr Wr Rb2	Sd Addr Wr Lb3	Sd Addr Wr Rb3
Clock 17	**Clock 18**	**Clock 19**	**Clock 20**	**Clock 21**	**Clock 22**	**Clock 23**	**Clock 24**
Sd Addr Wr Lb4	Sd Addr Wr Rb4	Sd Addr Wr Lr1	Sd Addr Wr Rr1	Sd Addr Wr Lr2	Sd Addr Wr Rr2	Sd Addr Wr Lr3	Sd Addr Wr Rr3
Clock 25	**Clock 26**	**Clock 27**					
Sd Addr Wr Lr4	Sd Addr Wr Rr4	Reset Calc Index					

Fig 41. Operating Cycles for Vertical Edges of 8x4 Chroma Pixels



• Vertical Filtering for Luma

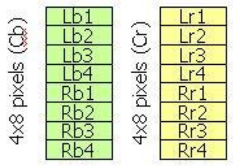
4x8 pixels (Y)

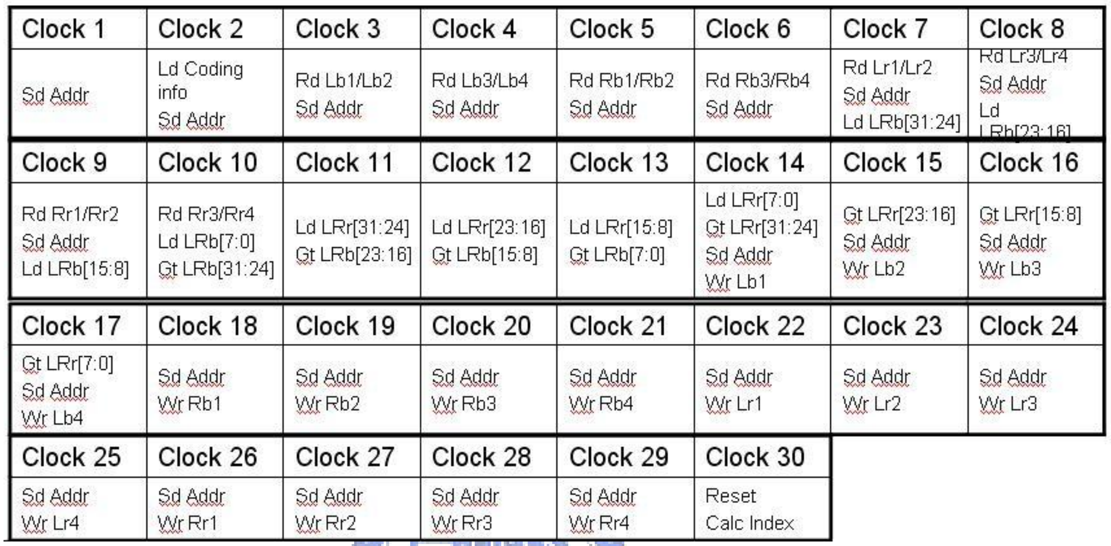
L1
L2
L3
L4
R1
R2
R3
R4

Clock 1	Clock 2	Clock 3	Clock 4	Clock 5	Clock 6	Clock 7	Clock 8
Sd Addr	Ld Coding info Sd Addr	Rd L1/R1 Sd Addr	Rd L2/R2 Sd Addr	Rd L3/R3 Sd Addr	Rd L4/R4	Ld LR[31:24]	Ld LR[23:16]
Clock 9	**Clock 10**	**Clock 11**	**Clock 12**	**Clock 13**	**Clock 14**	**Clock 15**	**Clock 16**
Ld LR[15:8]	Ld LR[7:0] Gt LR[31:24]	Gt LR[23:16]	Gt LR[15:8]	Gt LR[7:0]	Sd Addr Wr L1	Sd Addr Wr L2	Sd Addr Wr L3
Clock 17	**Clock 18**	**Clock 19**	**Clock 20**	**Clock 21**	**Clock 22**		
Sd Addr Wr L4	Sd Addr Wr R1	Sd Addr Wr R2	Sd Addr Wr R3	Sd Addr Wr R4	Reset Calc Index		

Fig 42. Operating Cycles for Horizontal Edges of 8x4 Luma Pixels

• Vertical Filtering for Chorma





Clock 1	Clock 2	Clock 3	Clock 4	Clock 5	Clock 6	Clock 7	Clock 8
Sd Addr	Ld Coding info Sd Addr	Rd Lb1/Lb2 Sd Addr	Rd Lb3/Lb4 Sd Addr	Rd Rb1/Rb2 Sd Addr	Rd Rb3/Rb4 Sd Addr	Rd Lr1/Lr2 Sd Addr Ld LRb[31:24]	Rd Lr3/Lr4 Sd Addr Ld LRb[23:16]
Clock 9	Clock 10	Clock 11	Clock 12	Clock 13	Clock 14	Clock 15	Clock 16
Rd Rr1/Rr2 Sd Addr Ld LRb[15:8]	Rd Rr3/Rr4 Ld LRb[7:0] Gt LRb[31:24]	Ld LRr[31:24] Gt LRb[23:16]	Ld LRr[23:16] Gt LRb[15:8]	Ld LRr[15:8] Gt LRb[7:0]	Ld LRr[7:0] Gt LRr[31:24] Sd Addr Wr Lb1	Gt LRr[23:16] Sd Addr Wr Lb2	Gt LRr[15:8] Sd Addr Wr Lb3
Clock 17	Clock 18	Clock 19	Clock 20	Clock 21	Clock 22	Clock 23	Clock 24
Gt LRr[7:0] Sd Addr Wr Lb4	Sd Addr Wr Rb1	Sd Addr Wr Rb2	Sd Addr Wr Rb3	Sd Addr Wr Rb4	Sd Addr Wr Lr1	Sd Addr Wr Lr2	Sd Addr Wr Lr3
Clock 25	Clock 26	Clock 27	Clock 28	Clock 29	Clock 30		
Sd Addr Wr Lr4	Sd Addr Wr Rr1	Sd Addr Wr Rr2	Sd Addr Wr Rr3	Sd Addr Wr Rr4	Reset Calc Index		

Fig 43. Operating Cycles for Horizontal Edges of 8x4 Chorma Pixels

6. Conclusion and Future Work

In this thesis, an VLSI accelerator architecture for the in-loop filter of H.264/AVC is proposed. The architecture is implemented and synthesized on an FPGA for verification. Simulation results show that the gate count is small and the efficiency of the proposed design is very good and can be used for real-time processing of CIF resolution video sequences.

The architecture of the system is designed with future extension in mind. Other functional components of H.264/AVC will be designed and added into the system. In order to allow these modules to share the bus more efficiently, a better arbiter of MMB must be implemented. In addition, when more functional units are added, a more sophisticated distributed SRAM subsystem must be used. Another key point of study is the reduction of the communication overhead between the processor core and these functional units.

7. REFERENCES

[1] T. Wiegand (ed.), JVT-F100d1, 6th Joint Video Team Meeting, Awaji Island, Japan, December 5-13, 2002.

[2] ISO/IEC 14496-2 Information technology- Coding of audio-visual objects- Part 2: Visual, 2nd edition, ISO/IEC, December 1, 2001.

[3] AMBA™ Specification (Rev 2.0) Copyright ARM Limited 1999.

[4] ARM, Integrator/AP User Guide, ARM Ltd., April 2001.

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

[6] E.G. Richardson, “VIDEO CODEC DESIGN Developing Image and Video Compression Systems.” Copyright © 2002 by John Wiley & Sons Ltd. pp. 199-200

[7] R. C Gonzalez, and R. E. Woods, “Digital Image Processing.” pp. 119-123

[8] ISO/IEC JTC 1/SC 29/WG 11 N4350, INFORMATION TECHNOLOGY –

CODING OF AUDIO-VISUAL OBJECTS – Part 2: Visual.

[9] L. Chiariglione, Doc. ISO/IEC JTC1/SC29/WG11 N271, "New work item proposal (NP) for very-low bitrates audio-visual coding", London meeting, November 1992

[10] ITU-T Rec. H.264 / ISO/IEC 11496-10, "Advanced Video Coding", Final Committee Draft, Document JVT-E022, September 2002

[11] ITU-T Recommendation H.261, "Video CODEC For Audiovisual Services At p x 64 kbits", March 1993

[12] ITU-T Recommendation H.263, "Video Coding For Low Bitrate Communication", Version 1, May 1996

[13] ARM, Integrator/AP User Guide, ARM Ltd., April 2001.

[14] ARM Integrator/CM-920T User Guide, ARM Ltd., 2001, 2002.

[15] ARM Integrator/LM-XCV600E+ User Guide, ARM Ltd., 2000-2001.

[16] H. C Fang, T. C Wang, and L. G Chen, "Real-time deblocking filter for MPEG-4 systems." *IEEE CNF Circuits and Systems*, pp.541-544, vol 1, 28-31 Oct. 2002.

[17] Y. W Huang, T. W Chen, B. Y Hsieh, T. C Wang, T. H Chang, L. G Chen, "Architecture design for deblocking filter in H.264/JVT/AVC." *Multimedia and Expo, 2003. ICME '03. Proceedings*. pp. 693 - 696, vol 1, 6-9 July 2003.