Hardware Implementation of a High Speed Deblocking Filter for the H.264 Video Codec

by

Brian Dickey

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2012

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

H.264/MPEG-4 part 10 or Advanced Video Coding (AVC) is a standard for video compression. MPEG-4 is currently one of the most widely used formats for recording, compression and distribution of high definition video. One feature of the AVC codec is the inclusion of an in-loop deblocking filter.

The goal of the deblocking filter is to remove blocking artifacts that exist at macroblock boundaries. However, due to the complexity of the deblocking algorithm, the filter can easily account for one-third of the computational complexity of a decoder.

In this thesis, a modification to the deblocking algorithm given in the AVC standard is presented. This modification allows the algorithm to finish the filtering of a macroblock to finish twenty clock cycles faster than previous single filter designs.

This thesis also presents a hardware architecture of the H.264 deblocking filter to be used in the H.264 decoder. The developed architecture allows the filtering of videos streams using 4:2:2 chroma subsampling and 10-bit pixel precision in real-time.

The filter was described in VHDL and synthesized for a Spartan-6 FPGA device. Timing analysis showed that is was capable of filtering a macroblock using 4:2:0 chroma subsampling in 124 clock cycles and 4:2:2 chroma subsampling streams in 162 clock cycles. The filter can also provide real-time deblocking of HDTV video (1920×1080) of up to 988 frames per second.

Acknowledgements

This work could not have be accomplished without guidance from Dr. Andrew Kennings. I would also like to thank Dr. Shaowen Song and SOC Technologies for giving permission to write this thesis based on the project I worked on with them.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Video compression systems are used in many commercial products ranging from consumer electronic devices such as high definition televisions to video teleconferencing systems. These applications have made hardware capable of implementing video compression an inevitable part of many commercial products. In order to improve performance and to enable the use of video compression in new real-time applications, a new international standard for video compression was developed. The new standard offers significantly better video compression efficiency than the previous video standards such as MPEG-2 and H.263. This new standard was developed with the collaboration of the International Telecommunication Union (ITU) and International Organization for Standardization (ISO). As such it is referred to by two different names, H.264 and MPEG-4.

The new standard uses a combination of encoding and decoding modules in order to achieve its high performance [1][2][3]. One such module in this new standard is the adaptive deblocking filter. Figure 1.1 the deblocking filter is applied to the decoded bitstream after an inverse quantization and inverse transform has been applied. The purpose of the deblocking filter is to improve the visual quality of decoded frames by filtering out the blocking artifacts and discontinuities in a frame that arise from the coarse quantization of macroblocks and motion prediction [4]. Once the filtering of a frame is complete, it is used as a reference frame for motion-compensated prediction of future frames. As a result, the deblocking filter also increases coding efficiency resulting in bit-rate savings [5].

The deblocking filter algorithm used in the H.264 standard is more complex than the deblocking filter algorithms used by previous standards. This complexity reduces the speed

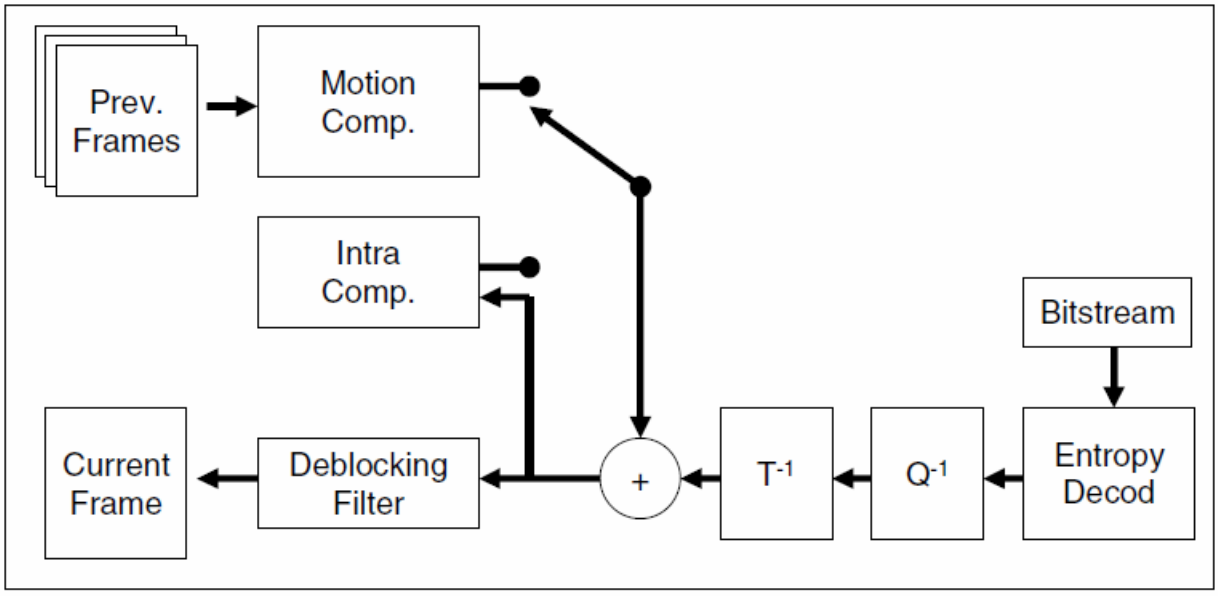

Figure 1.1: Block diagram of a H.264 decoder system [6].

at which a H.264 decoder can run. This can be traced to several facts since the deblocking algorithm is highly adaptive. The filtering algorithm must be applied to each edge of all the 4×4 luma and chroma blocks present in a macroblock. Deblocking can update three pixels in during each filtering operation. Finally, to decide whether the filter will be applied to an edge, the current and neighbouring pixels of the macroblock must be read from memory. Consequently, memory organization becomes a bottleneck. As such, the deblocking filter can account for about one third of the complexity of an H.264 encoder or decoder [5].

1.1 Research Contribution

This thesis presents the design and description of a hardware implementation a deblocking filter written in VHDL. The specific contributions of this thesis include

1. a hardware implementation of a deblocking filter capable of decoding video with frame rates of up to sixty frames per second (60 FPS),

2. a hardware implementation of a deblocking filter with the ability to filter 8- or 10-bit pixel information, and

3. a hardware implementation of a deblocking filter with the ability to filter 4:2:0 or 4:2:2 colour space.

The motivation for an optimized deblocking filter should be clear from the previous description. A specific hardware implementation of the deblocking filter offers a low cost alternative (low-cost FPGAs) to traditional embedded encoder/decoder systems.

1.2 Thesis Organization

Chapter 2 outlines the necessary background information for the reader to have a better understanding of the H.264 codec and the need for the deblocking filter. Chapter 3 presents past research on deblocking filter designs and algorithms. Chapter 4 discusses the implemented algorithm and design of the deblocking filter. The results of the implemented deblocking filter are presented in chapter 5. Finally, chapter 6 concludes the thesis with a discussion of the results.

Chapter 2

Background

Two main groups responsible for the standardization of video compression techniques are the Video Coding Experts Group (VCEG) of the International Telecommunications Union (ITU-T) and the Moving Picture Experts Group (MPEG) of the International Organization for Standardization and International Engineering Consortium (ISO/IEC). Each organization develops standards targeted towards applications ranging from handheld devices and standard-definition television (SDTV) to high-definition television (HDTV) broadcasts.

VCEG generally develops standards targeted towards low bit-rate communication, while MPEG focuses on high performance standards. As shown in figure 2.1 [7], a number of video compression standards have been developed by both MPEG and VCEG over the years.

While the standards have been mostly developed independently, MPEG and VCEG have worked together to develop video codec standards. The first joint venture between these two groups resulted in the H.262/MPEG-2 standard. This standard is widely used for television broadcast (SDTV and HDTV) and for DVD codecs.

In 2001, another joint venture occurred and was referred to as the Joint Video Team (JVT). In March of 2003, the JVT submitted a new video standard that is called by many names: H.264, ISO/IEC 14496-10, H.264/AVC, MPEG-4 and MPEG-4 Part 10. The standard will be referred to as H.264 for the remainder of this document.

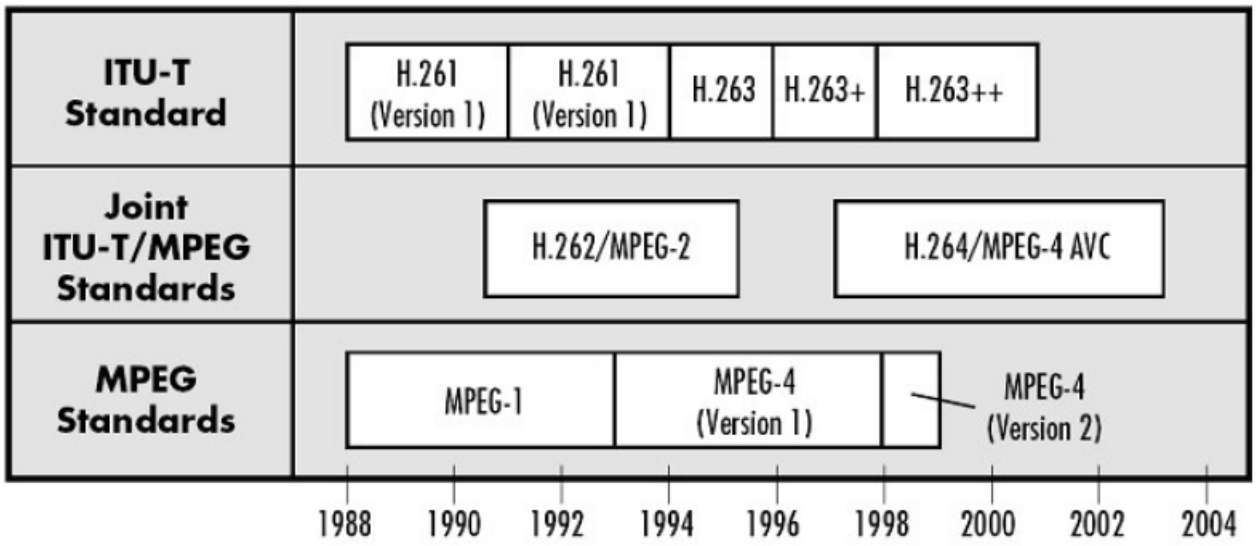

Figure 2.1: Video codec standards [7][8].

2.1 Video Information

A video signal is composed of a series of still pictures referred to as frames. The number of frames displayed per second, referred to as the frame rate, determines how smooth changes in the scene will be perceived. The human visual system can process 10 to 12 separate images per second, perceiving each image individually. The visual cortex holds onto one image for about one-fifteenth of a second. Therefore, the higher the frame rate, the smoother the moving picture is perceived.

The most common frame rates that are in use are 24, 25 and 30. Frame rates of 24 or 30 FPS are the standard for National Television Systems Committee (NTSC) television. The frame rate 25 FPS is popular since it derives from the Phase Alternating Line (PAL) television standard of 50i (or 50 interlaced fields per second). This rate is used in 50 Hz regions for direct compatibility with television field and frame rates. Many new high performance video distributors are using 60 FPS on 1080p (1920×1080 image dimensions) content to provide even more apparent smooth motion.

Video signals can be transmitted in two different formats, interlaced or progressive. When a progressive scan is performed, an image is captured, transmitted, and displayed

Figure 2.2: Interlaced video on a progressive monitor with very poor deinterlacing [9].

line-by-line from top to bottom. However, interlaced scan completes this pattern as well, but only for every second line. This is carried out from the top left corner to the bottom right corner. This process is repeated again, only this time starting at the second row, in order to fill in those gaps left behind while performing the first progressive scan on alternate rows only. Each picture is called a field and two fields create a single frame of video. A benefit of interlaced video is its ability to double the perceived frame rate introduced with the composite video signal used with analog television without consuming extra bandwidth. It also enhances motion perception to the viewer and reduces flicker by taking advantage of the persistence of vision effect.

Due to each frame of interlaced video being composed of two fields that are captured at different moments in time, interlaced video frames will exhibit motion artifacts known as "interlacing effects" or combing. Combing results from recorded objects moving fast enough to be in different positions when each individual field is captured. These artifacts may be more visible when interlaced video is displayed at a slower speed than it was captured or when still frames are presented; an effect can be seen in figure 2.2.

Progressive scan as mentioned above displays the captured image line by line instead of alternating as in interlaced. While this eliminates the motion artifacts that can arise in interlaced video, progressive scan requires additional bandwidth in order to display the image. Progressive scan offers other advantages over interlaced such as

1. intentional blurring of video to reduce interline twitter and eye strain is not required;

2. clearer and faster results for scaling to higher resolutions than its equivalent interlaced video, such as upconverting 480p to display on a 1080p HDTV; and

3. frames have no interlace artifacts which can be captured for use as still photos.

Whether interlaced or progressive scan is used for active video, the image must be digitally encoded with a colour space in order to be displayed. RGB, a common and simple color space, divides pixels into three components, red (R), green (G) and blue (B). The YCbCr color space is used in H.264 and is also composed of three components. The luma component (Y) represents brightness in an image. Cb and Cr represent how much the colour deviates from gray toward blue or red, respectively. The human visual system is less sensitive to the position and motion of color than luminance [1]. As a result, video bandwidth can be optimized by storing more luminance detail than colour detail. In H.264, YCbCr can take advantage of this by representing chroma components using fewer samples relative to luma components.

Chroma subsampling is usually designated by a string of 3 integers separated by colons (occassionally 4 integers are used). The relationship among the integers denotes the degree of vertical and horizontal subsampling. The first digit refers to the luma horizontal sampling rate. The second digit specifies the horizontal subsampling of both Cb and Cr with respect to luma. The third digit can take on two different values. If it is the same as the second digit, no vertical subsampling occurs. If it is a 0, a 2:1 vertical subsampling of both Cb and Cr occurs. So a 4:4:4 colour space would mean all YCbCr components are sampled at the same rate while 4:2:2 means that the chroma components are sampled at half the rate of the luma components. Figure 2.3 shows several different chroma subsampling formats.

2.2 Image Compression

The H.264 standard has each video frame divided into macroblocks. A macroblock is usually composed of two or more blocks of pixels. In H.264, a macroblock consists of 16×16 array of luma samples and an 8×8 array of both chroma samples. For example, a 720p video (1280×720) has each frame divided into 3600 macroblocks (80×45).

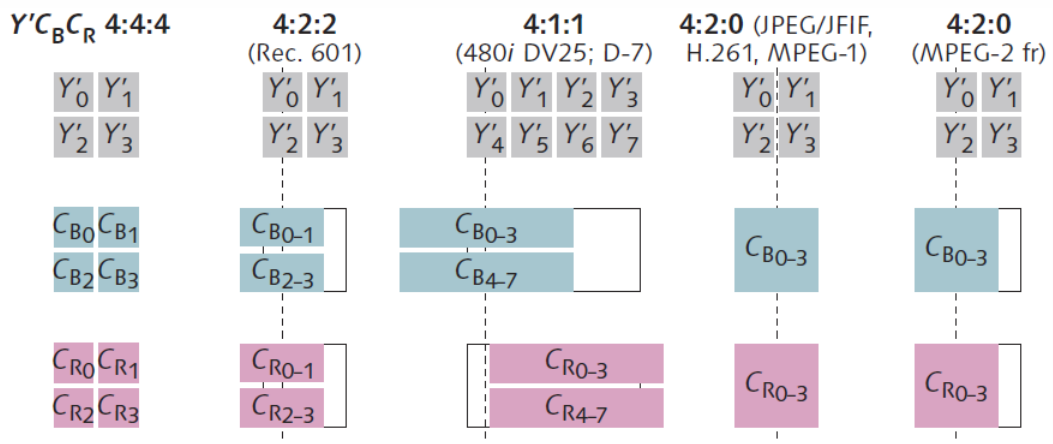

Figure 2.3: In 4:4:4, each Cb and Cr sample are incident on a luma sample. In 4:2:2, 4:1:1, and 4:2:0 used in MPEG-2, Cb and Cr are positioned horizontally coincident with a luma sample. In 4:2:0, Cb and Cr are midway between luma samples [10].

2.3 Overview of the H.264 Standard

A high-level diagram describing the encoding process of the H.264 standard as described in [8] is shown in figure 2.4.

Uncompressed video flows through either the inter- or intra-prediction blocks, transform block and then the quantization block. The data is then fed back through inverse quantization and inverse transform blocks. These blocks provide data that is used to calculate the difference between the actual incoming video and the estimated, transformed and quantized frames that would be seen by a decoder. The results of these calculations are called the residual values (residuals). The residuals are then encoded along with settings and parameters used for motion compensation, transformation and quantization to estimate the original frame. The data is then sent through entropy coding. The encoded parameters are sent in headers. The run-length decoder takes samples and header information and forms a packet of data for compressed transmission. The decoder block diagram seen in figure 1.1, follows a similar feedback path as in the encoder.

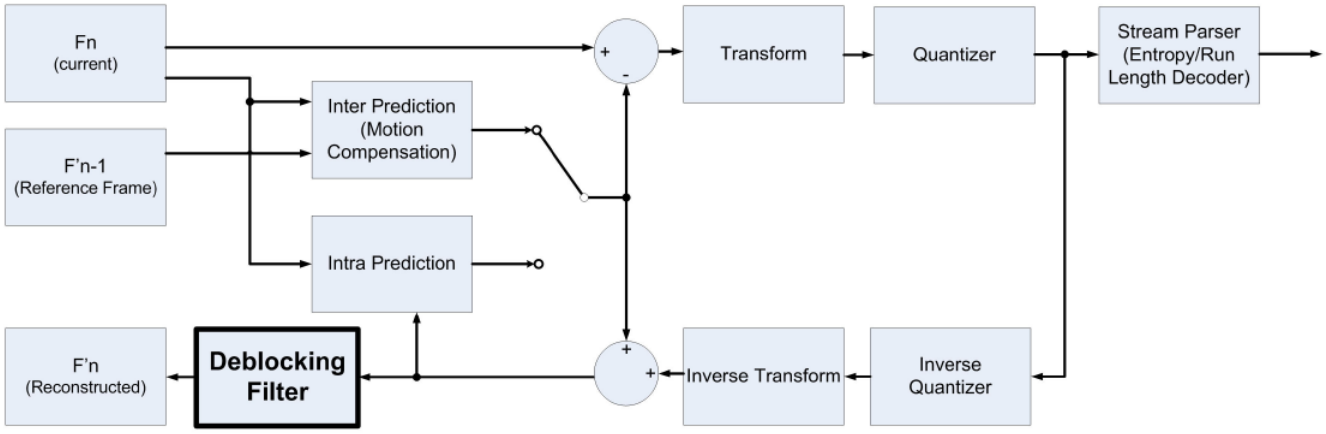

Figure 2.4: H.264 encoder [8].

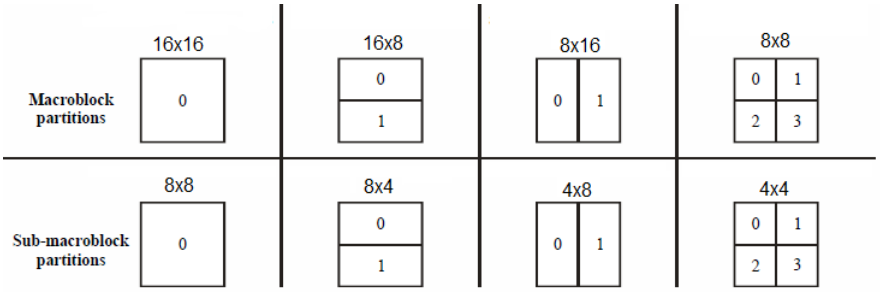

Figure 2.5: Macroblock and sub-macroblock partitions.

2.3.1 Inter Prediction

With multiple frames being displayed every second, it is highly likely that consecutive images contain similar data. Inter-prediction creates a prediction model for an estimated block using blocks from previous video frames. Motion estimation is improved in H.264 over previous standards by including support for many different block sizes, which in turn may be composed of a combination of various sub-blocks. The defined macroblock partitions in H.264 can be seen in figure 2.5.

A two-dimensional vector is used for inter prediction that provides an offset from the coordinates in the decoded picture to the coordinates in a reference picture [1]. This vector is referred to as a motion vector. It is possible that the motion vector will not be an integer

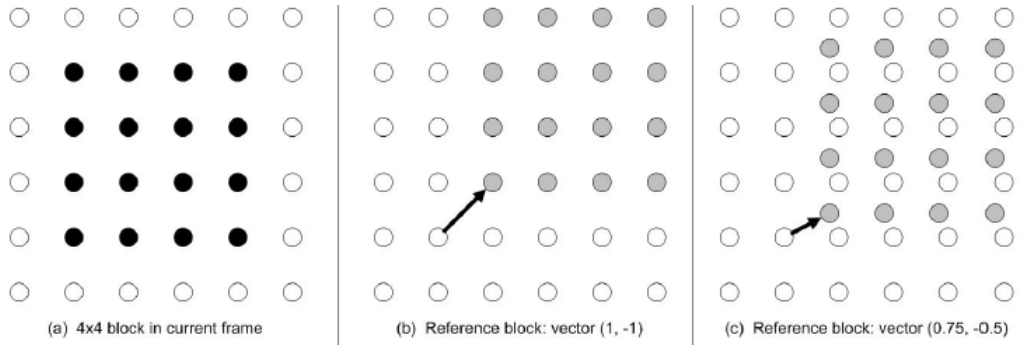

Figure 2.6: Integer and non-integer motion vectors.

number of samples and therefore interpolation must be used to calculate non-integer motion vectors. H.264 luma components can have integer, half or quarter-sample resolution and chroma components have one-eighth-sample resolution. An example of integer and sub-sample predictions can be seen in figure 2.6. Figure 2.6 (a) shows the current 4×4 block in the frame. Figure 2.6 (b) shows an integer motion vector while (c) shows a non-integer motion vector.

The prediction values for one-half sample positions are calculated by applying a 6-tap finite impulse response (FIR) filter horizontally and vertically. Quarter-sample positions are obtained by taking the average of the integer and half-sample positions. The process of estimating sub-sample accurate motion compensation is depicted in figure 2.7. Uppercase letters indicate actual pixels in the reference frame. The lowercase letters represent the half or quarter locations. Half-sample locations are generated first using the 6-tap filter. An N-tap filter is a finite impulse response filter where N refers to order of the filter.

The half sample b_1 is obtained by filtering pixels E through J (the horizontal row of pixels). Similarly, h_1 is obtained by filtering the vertical column of pixels. The samples are then rounded and then clipped to the range of 0 to 255 or 4095 (depending on the pixel bit depth), to provide the half-sample positions b and h.

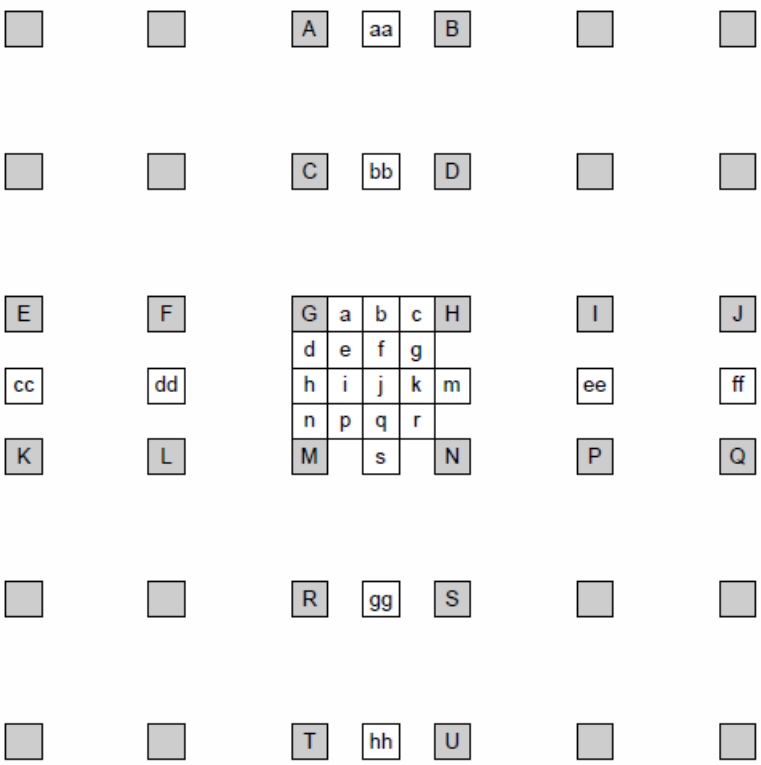

Figure 2.7: Integer and non-integer motion vectors.

$$b_1 = (E - 5F + 20G + 20H - 5I + J),$$

$$h_1 = (A - 5C + 20G + 20M - 5R + T),$$

$$b = \frac{b_1 + 16}{32} \text{and}$$

$$h = \frac{h_1 + 16}{32}.$$

The half-pixel position j is calculated similarly as h except this time, half sample positions are used. Due to j's position in the grid, it can be calculated using horizontal or vertical half samples. The equation below uses the horizontal calculation. Again j is rounded and clipped. The calculations are

$$j_1 = (cc - 5dd + 20h_1 + 20m_1 - 5ee + ff) \text{ and}$$
$$j = \frac{j_1 + 512}{1024}.$$

Quarter-samples a, c, d, n, f, i, k and q are derived by taking the average of the two nearest samples at integer and half sample positions. A sample calculation is

$$a = \frac{G + b + 1}{2}.$$

The quarter samples at e, g, p and r are calculated by taking the average of the two nearest half-samples in the diagonal direction. The chroma components are calculated in a similar manner to luma, only with a one-eighth sample resolution. A sample calculation is

$$e = \frac{b + h + 1}{2}.$$

A high correlation exists between motion vectors of a macroblock and its neighbours. H.264 takes advantage of this by predicting motion vectors based on neighbouring vectors, reducing the amount of data that needs to be sent. Based on this, only the differences between the predicted motion vector and the actual motion vector is sent in the data stream.

In previous standards, reference order and display order were one in the same. H.264 had this restriction removed allowing the encoder to choose the order of pictures for reference purposes regardless of the display ordering in order to optimize the decoding process.

There are three types of picture frames that are used for motion compensation; I-frames, P-frames and B-frames. An I-frame is an intra-coded picture. This is essentially a fully specified frame. These act as a reference for other frames to be encoded. P-frames or predicted picture frames hold changes that are different in the frame. For example, if there is movement in a scene across a stationary background, then only the movement needs encoding. B-frames are bi-directionally predictive-coded frames whose coding is based upon both past and future frames.

2.3.2 Intra Prediction

Intra prediction is another feature added to the H.264 standard and is used to reduce spatial redundancies. This operation occurs after the inverse transform block, referred to as the spatial domain. Intra prediction predicts pixel values by extrapolating neighbouring pixels from adjacent reconstructed blocks. Four different intra-coding modes are available:

- 4×4 intra prediction: Predicts each 4×4 luma block based on bordering 4×4 blocks and nine different directions.
- 8×8 intra prediction: Similar to 4×4 intra prediction except 8×8 luma blocks are used instead.
- 16×16 intra prediction: Predicts the luma portion of each macroblock in a single operation based on neighbouring macroblocks.
- I_PCM: Transform and prediction stages are bypassed, allowing for the transmission of image samples directly.

Intra 4×4 Prediction Mode

Figure 2.8 shows a 4×4 luma block requiring intra prediction. The values A-M are reference pixels that are used to encode the current 4×4 block. The pixels a-p are calculated from the reference samples A-M based on the ones of the selected modes in table 2.1.

Intra 8×8 Prediction Mode

The intra prediction for 8×8 blocks is similar to 4×4 blocks with a couple small differences. The number of vertical reference samples extends to eight and the number of horizontal samples are now sixteen. Intra 8×8 prediction uses the same prediction modes as 4×4 prediction [11].

Intra 4×4 **Prediction Mode**	**Description**
Vertical	A, B, C, D are extrapolated vertically
Horizontal	I, J, K, L are extrapolated horizontally
DC	Samples are predicted by taking the average of A-D and I-L
Diagonal down-left	Samples are interpolated at a 45° angle between lower-left and upper-right.
Diagonal down-right	Samples are extrapolated at a 45° angle between down and to the right.
Vertical-right	Samples are extrapolated at a 26.6° angle to the left of vertical.
Horizontal-down	Samples are extrapolated at a 26.6° angle below horizontal.
Vertical-left	Samples are extrapolated (or interpolated) at a 26.6° angle
Horizontal-up	Samples are interpolated at a 26.6° angle above horizontal.

Table 2.1: The nine 4×4 intra prediction modes [2].

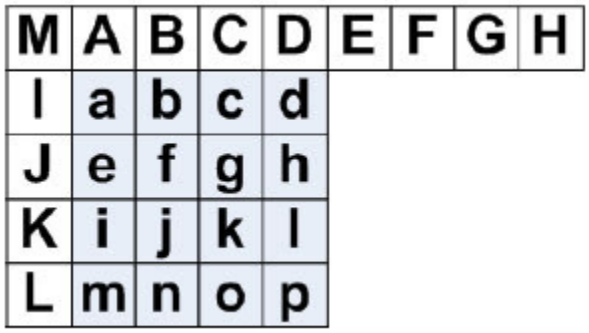

Figure 2.8: Intra 4×4 prediction mode directions [2].

Prediction Mode	Description
Vertical	Upper samples extrapolated vertically
Horizontal	Left-hand samples extrapolated horizontally
DC	Samples are predicted by taking the average of left-hand and upper reference samples
Plane	Linear plane function used on upper and left-hand samples

Table 2.2: The four 16×16 intra prediction modes [3].

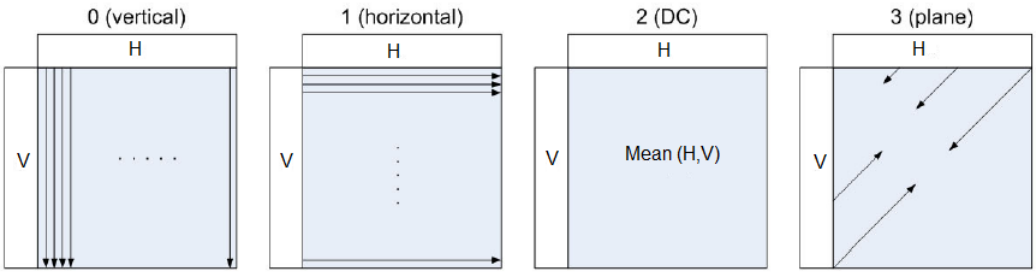

Figure 2.9: Intra 16×16 prediction mode directions [2].

Intra 16×16 Prediction Mode

The 16×16 intra prediction mode predicts all luma components of a macroblock in a single operation. This prediction is faster and is useful in smooth regions of the frame. This prediction mode only has four prediction modes shown in table 2.2. These prediction modes can be seen in figure 2.9 [8].

Chroma Prediction

Each 4×4 chroma block is predicted using reconstructed chroma samples from above and to the left. The prediction is carried out using the same four modes that are used for 16×16 luma intra prediction.

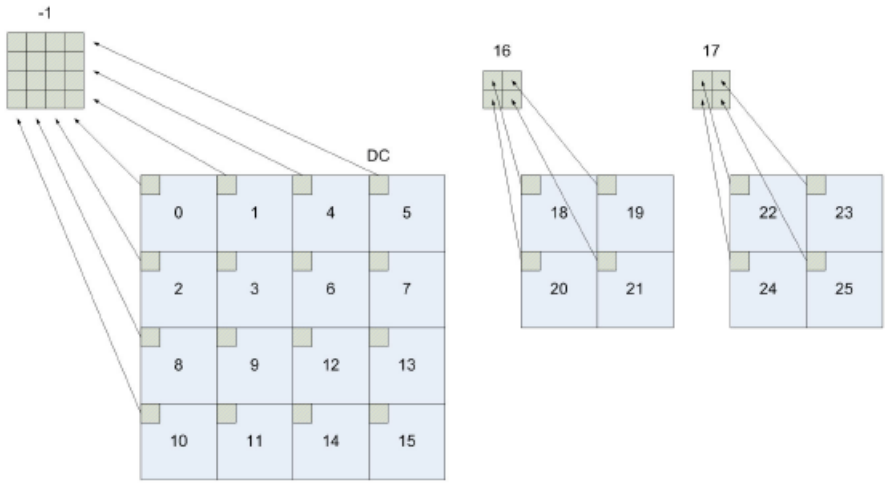

Figure 2.10: Scanning order of the residual blocks [2].

2.3.3 Transforms and Quantization

H.264 uses three transforms depending on the type of residual data that is to be coded. They are a Hadamard Transform for the 4×4 array of luma DC coefficients in intra macroblocks that were predicted in the 16×16 mode, a Hadamard transform for the 2×2 chroma DC coefficients array and a Discrete Cosine Transform (DCT) based transform for all other residual data [2].

Macroblock data is transmitted to the transform in the order shown in figure 2.10. If the macroblock is coded using 16×16 intra prediction, then the block labeled -1, the DC coefficients, is sent first. The luma residual blocks 0-15 are transmitted next followed by blocks 16 and 17. These two blocks create a 2×2 array of DC coefficients from the Cb and Cr components. Finally, chroma residual blocks 18-25 are sent.

4×4 residual transform and quantization

The transform that is used on the 4×4 residual data blocks (labelled 0-15 and 18-25 in figure 2.10) is a DCT based transform but has a few differences from a true DCT. It is integer based meaning all operations can be carried out without worrying about a loss of accuracy. Zero mismatch can be ensured between the encoder and decoder by using

integer arithmetic. The transform can be implemented entirely by using additions and shifts. Finally, a scaling multiplication is included in the transform which reduces the total number of multiplications that are needed. The transform is shown below

$$\mathbf{Y} = \mathbf{C_f X C_f^T} \otimes E_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \mathbf{X} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix}$$

where a and b are defined as

$$a = \frac{1}{2} \text{ and } b = \sqrt{\frac{2}{5}}.$$

The $\mathbf{C}_f\mathbf{X}\mathbf{C}_f^T$ portion of the transform is the core of the 2D transform. Matrix $\mathbf{E}_f^T$ is a scaling factor designed to avoid multiplications that would result in a loss of accuracy. Transformed coefficients ($\mathbf{C}_f\mathbf{X}\mathbf{C}_f^T$) that do not use the scaling matrix is denoted by $\mathbf{W}$.

Following the transform, quantization occurs. The amount of quantization that occurs is dependent on the quantization parameter (QP). Quantization of $\mathbf{Z}$ is carried out in the following manner

$$|Z_{ij}| = \frac{|W_{ij}| \times MF + f}{qbits}$$

where *qbits* and f are defined as

$$qbits = 15 + \left\lfloor \frac{QP}{6} \right\rfloor \text{ and } f = \begin{cases} \frac{2^{qbits}}{3}, & \text{Intra blocks;} \\ \frac{2^{qbits}}{6}, & \text{Inter blocks.} \end{cases}$$

The multiplication factor MF is determined from table 2.3.

4×4 Luma DC Coefficient Transform and Quantization for 16×16 Intra-mode

If 16×16 intra prediction is used, each 4×4 residual block is transformed using the above core transform ($\mathbf{C}_f\mathbf{X}\mathbf{C}_f^T$). Once this is done, each DC coefficient is transformed again this

QP	Positions (0,0),(2,0),(2,2),(0,2)	Positions (1,1),(1,3),(3,1),(3,3)	Other positions
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

Table 2.3: Multiplication factor MF [2].

time using a 4×4 Hadamard transform. The Hadamard transform is example of a Fourier transform. It performs an orthogonal, symmetric, involutional, linear operation on 2^{m} real numbers. The output $\mathbf{Z}_D$ is then quantized to produce the quantized DC coefficients. This calculation is

$$|Z_{D(ij)}| = \frac{|Y_{D(ij)}| \times MF_{(0,0)} + 2f}{qbits + 1}$$

where $\mathbf{Y}_D$ is defined as

$$\mathbf{Y}_D = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \mathbf{W}_D \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}.$$

2×2 Chroma DC Coefficient Inverse Transform and Quantization

After the Cr and Cb DC Chroma coefficients are transformed using the core transform, a 4×4 Hadamard transform is applied once again. The transformations are then quantized:

$$|Z_{D(ij)}| = \frac{|W_{QD(ij)}| \times MF_{(0,0)} + (2 \times f)}{qbits + 1}$$

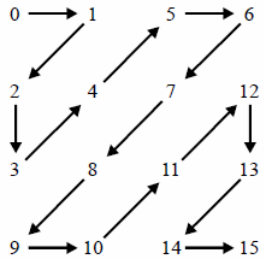

Figure 2.11: Zig zag scan pattern. Each number represents a 4×4 block [11].

where $\mathbf{W}_{QD}$ is defined as

$$\mathbf{W}_{QD} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \mathbf{W}_D \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

2.3.4 Video Stream Parsing

H.264 video streams are composed of video packets. The parser takes the scaled 4×4 coefficients from the quantizer, entropy encodes them and organizes the data into video packets for transmission. Each 4×4 block of quantized transform coefficients is mapped to a 16-element array in a zig-zag order (figure 2.11).

2.3.5 Entropy Coding

Entropy coding is used to convert a series of symbols representing elements of the video sequence into a compressed bitstream that is suitable for transmission or storage. Three different entropy coding types exist in H.264: basic, Exp-Golomb and context adaptive coding. Syntax elements above the slice layer are encoded as fixed or variable length binary codes. At or below the slice layer, elements are coded using variable length codes (VLCs) or context-adaptive arithmetic coding (CABAC) depending on the encoding mode. If the entropy coding mode is zero, residual block data is coded using context adaptive variable length coding (CAVLC) and other syntax elements are coded using Exp-Golomb schemes.

2.3.6 H.264 Profiles and Levels

The H.264 standard originally defined three primary profiles for decoders and encoders, targeting different video applications: Baseline, Main and Extended. This has since been updated to include additional profiles such as High, High 10, and High 4:2:2 [11]. Each profile defines a subset of features supported by all codecs conforming to that profile. The Baseline profile generally targets real-time conversation services such as networked video conferencing, providing minimal complexity with high robustness and flexibility. The Main profile is designed for digital storage media and standard-definition digital television broadcasting. The Extended profile is intended for use mainly in streaming video. The High profile focuses on high-definition broadcasts and disc storage, such as Blu-Ray. The High 10 profile is used in the same applications as the High profile except each pixel contains 10 bits of precision instead of 8. High 4:2:2 builds on both the High and High 10 profiles except the chroma subsampling is now 4:2:2.

As per the standard, each profile will be associated with an appropriate level. Levels define information about the video stream such as the frame rate, image resolution, maximum size of the frame, just to name a few.

2.4 Deblocking Filter

Blocking artifacts are issues that occur in block-based video coding schemes. In H.264, the main source of these artifacts is coarse quantization of coefficients from the integer transform used in the inter frame prediction error coding [5]. Blocking artifacts are also sourced from imperfect motion compensation prediction.

2.4.1 Motivation

The purpose of the deblocking filter in an H.264 decoder is to give a higher level of perceived quality by removing the blocking artifacts from the image. In previous standards, the deblocking filter was an optional feature. As seen in figure 1.1, the deblocking filter is the last step before the current frame is output.

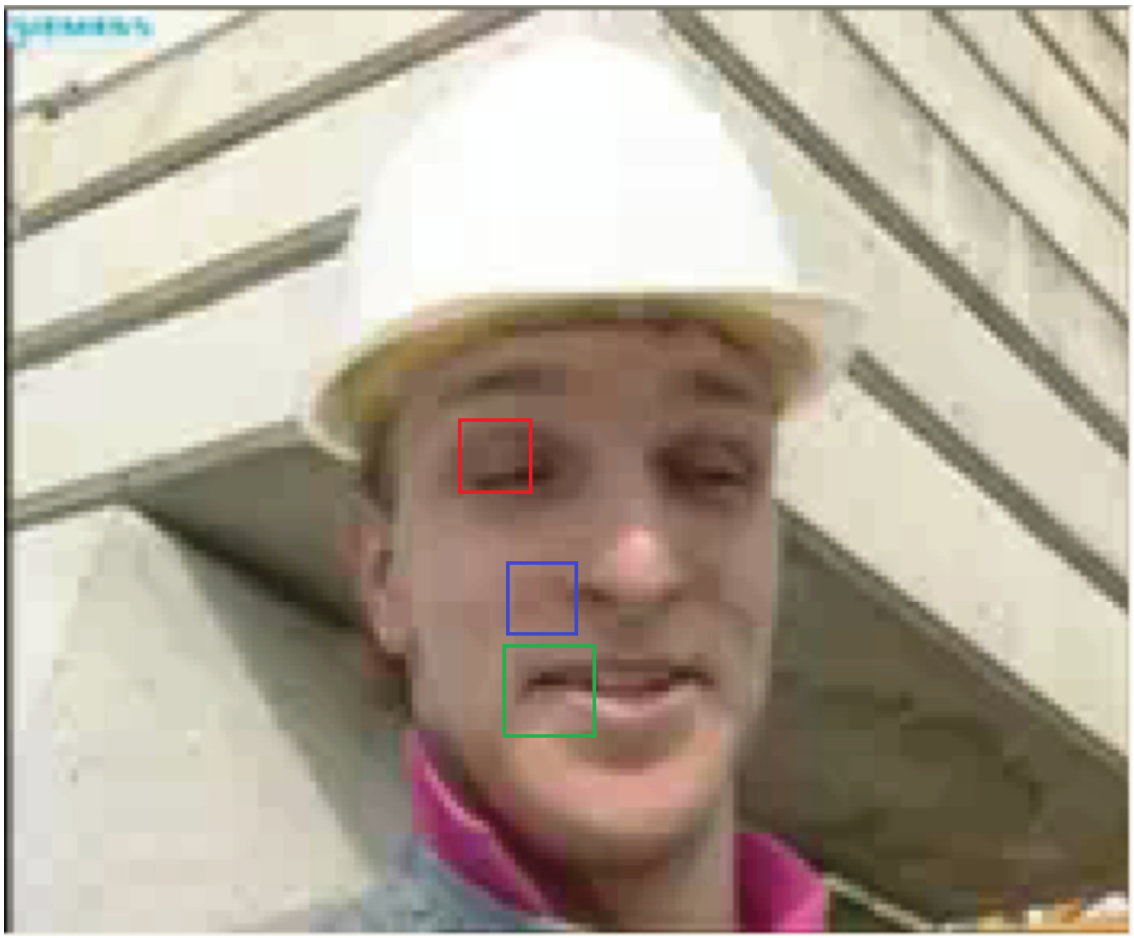

Figure 2.12: Image before deblocking process.

Figures 2.12 and 2.13 show two versions of the same image. Figure 2.12 shows an unfiltered image from the Foreman YUV test sequence. Three areas have been highlight where blocking artifacts are easily seen. This figure has a peak signal-to-ratio (PSNR) of 30.978 dB. Figure 2.13 shows a filtered version of the same image. The areas where blocking artifacts were quite apparent in figure 2.12 have again been highlighted. In this image, the blocking artifacts have been smoothed. The PSNR of this image is 31.375 dB [12]. This shows the importance of the deblocking filter.

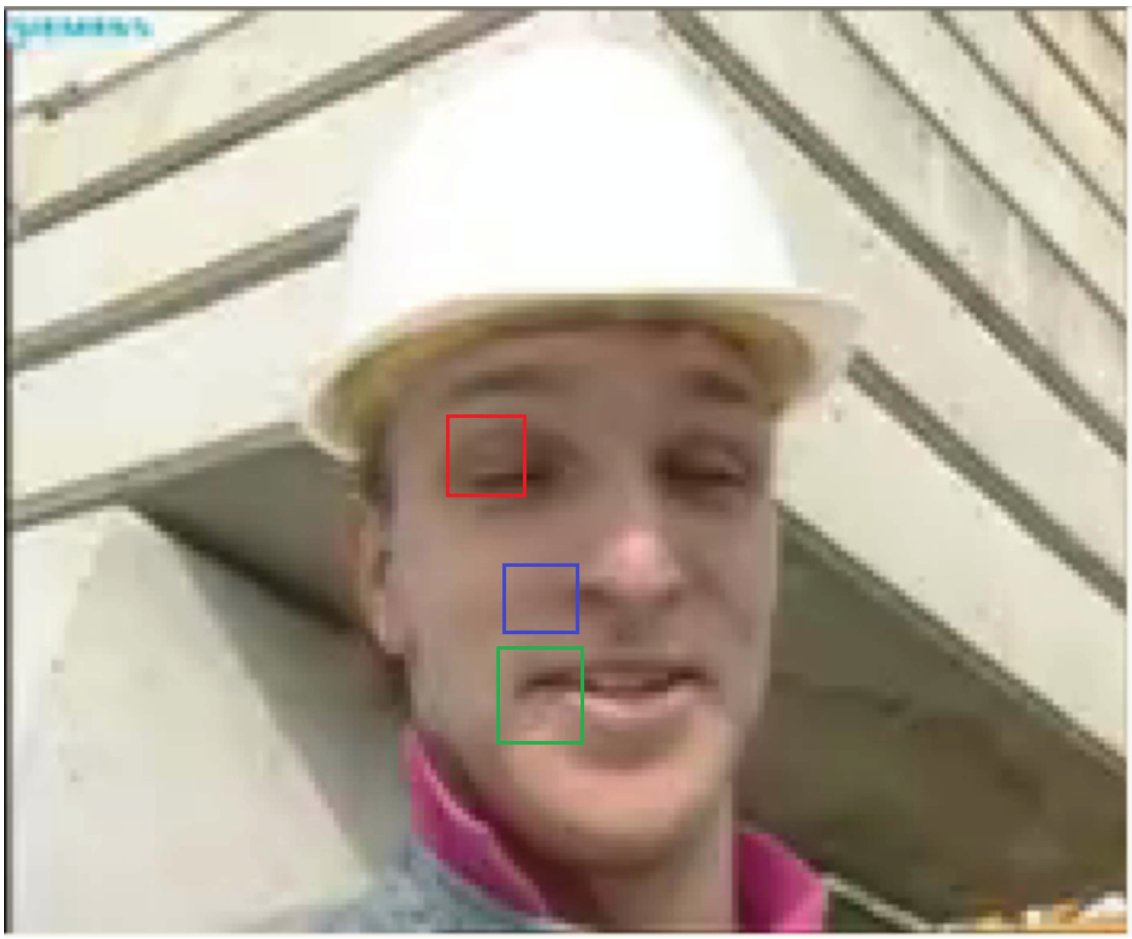

Figure 2.13: Image after deblocking process.

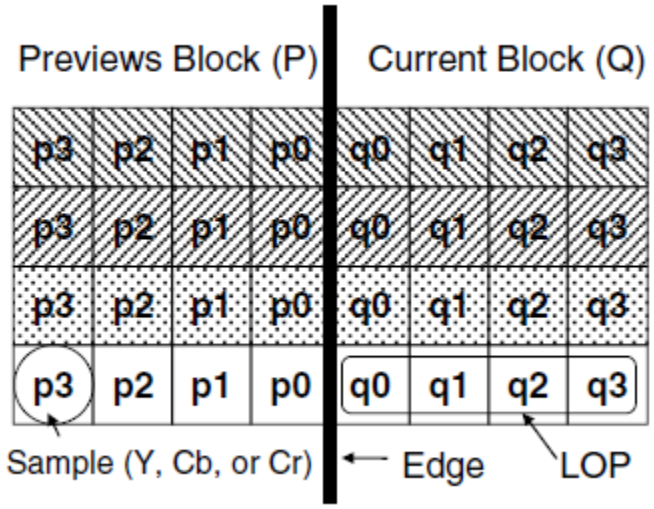

Figure 2.14: An unfiltered 4×4 boundary edge showing the pixels required for filtering.

2.4.2 Filter Operation

The deblocking filter takes in information regarding the boundary strength, certain threshold values and the pixels that are to be filtered. Each 4×4 sub-block inside a macroblock has its vertical and horizontal edges filtered. To filter each edge, eight pixels are required (see figure 2.14); four current pixels (q_0, q_1, q_2, q_3) and four reference pixels (p_0, p_1, p_2, p_3). Based on the pixel, threshold and boundary strength values, pixels $p_0 - p_2$ and $q_0 - q_2$ may be modified. Due to the way the filtering process is defined, pixels p_3 and q_3 remain unfiltered.

Pixels can be filtered as many as four times due to overlap in filtering between edges, and between vertical and horizontal filters. Chroma samples are filtered in the same manner as luma The basic filtering order, as defined for H.264, is shown in figure 2.15.

Filtering Decision

The strength of the filter is dependent on the boundary strength (bS) variable and the gradient of the image samples across the boundary. The strength is based off of the

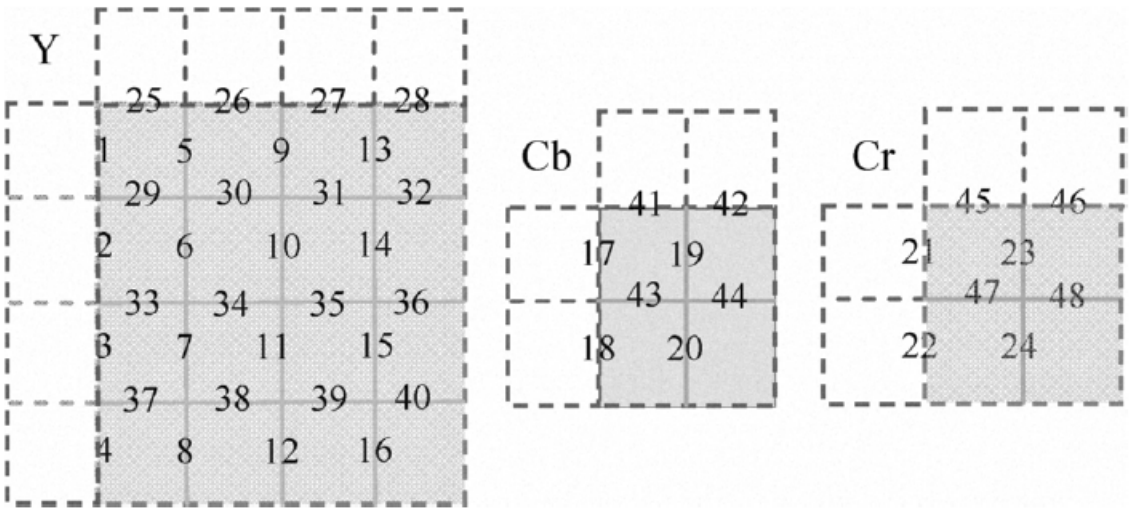

Figure 2.15: Orignal H.264 filtering order [1].

bS	Operation
0	No filtering
1-3	4-tap linear filter is applied producing p_0, q_0 and possibly p_1 and q_1 (depending on α and β)
4	3-, 4-, or 5-tap linear filter may be applied producing p_0, q_0 and possibly p_1 and q_1 (depending on α and β)

Table 2.4: Filtering strength summary

.

decision tree shown in figure 2.16. If a bS of 0 results then no filtering occurs while a bS of 4 results in the strongest filtering. Table 2.4 shows a summary of the different filtering strengths.

The values α and β are threshold values defined by the H.264 standard. They increase with the average quantization parameter (QP) value of the p and q blocks. These values assist in the decision as to when filtering should be turned on or off. A small QP value indicates that a small gradient present across a block boundary is likely due to features of the current image and should be preserved. In this case, α and β are small. When a large QP results, blocking distortion is likely present and α and β are higher so more pixel

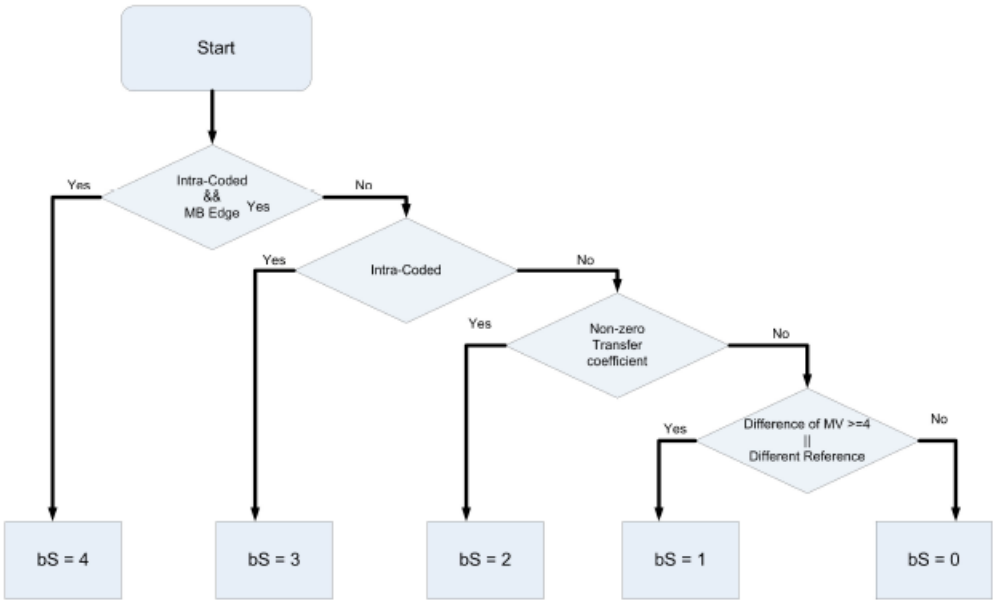

Figure 2.16: Calculation of boundary strength parameters in H.264 [8].

samples are filtered.

Edge Filtering when bS = 1, 2 or 3

When bS is equal to 1, 2 or 3 two additional threshold values are calculated, t_c and t_{co}. t_{co} is a threshold value defined by the H.264 standard. t_c is then calculated from t_{co}. t_c is calculated as follows:

$$t_c = t_{co} + x$$

where x is defined as

$$x = \begin{cases} 2 & |q_2 - q_0| < \beta \text{ and } |p_2 - p_0| < \alpha \\ 1 & \text{when others.} \end{cases}$$

Once t_c and t_{co} are calculated, the filtered samples p'_0 and q'_0 must be calculated. They filtered as follows

$$p'_0 = \text{Clip1}(p_0 + \Delta) \text{ and } q'_0 = \text{Clip1}(q_0 - \Delta).$$

Δ is defined as

$$\Delta = \text{Clip3}\left(-t_c, t_c, \frac{(2 \times (q_0 - p_0)) + (p_1 - q_1) + 4}{3}\right).$$

Clip1 and Clip3 are clipping functions that are used to specify a maximum range for the filtered samples so too much filtering does not occur on a boundary. If the change in intensity is low on either or both sides, stronger filtering is applied, resulting in a smoother final image. If sharp changes are occurring on the ends, less filtering is required, preserving image sharpness. These two functions are

$$\text{Clip1}(z) = \text{Clip3}\left(0, 2^{\text{pixel\_bit\_length}-1} - 1, z\right)$$

and

$$\text{Clip3}(a, b, c) = \begin{cases} a & if c < a \\ b & if c > b \\ c & \text{otherwise.} \end{cases}$$

Filtered samples p'_1 and q'_1 are calculated in a similar manner. In order for a 4-tap filter to be applied to samples q_1 equation 2.1 must be satisfied. Similarly, for a 4-tap filter to be used on sample p_1, equation 2.2 must be satisfied.

$$|q_2 - q_0| < \beta \tag{2.1}$$

$$|p_2 - p_0| < \beta \tag{2.2}$$

If 2.2 is satisfied and luma samples are present then p_1' is calculated according to 2.3. Otherwise, if 2.2 is not met or chroma samples are present then p_1' is calculated according to 2.4.

$$p_1' = p_1 + \text{Clip3}\left(-t_{c0}, t_{c0}, p_2 + \frac{p_0 + q_0 + 1}{2} - 2p_1\right) \tag{2.3}$$

$$p_1' = p_1 \tag{2.4}$$

Similarly, if 2.1 is satisfied and luma samples are present then q_1' is according to 2.5. Otherwise, if is not met or chroma samples are present then q_1' according to 2.6.

$$q_1' = q_1 + \text{Clip3}\left(-t_{c0}, t_{c0}, q_2 + \frac{p_0 + q_0 + 1}{2} - 2q_1\right) \tag{2.5}$$

$$q_1' = q_1 \tag{2.6}$$

The values of q_2' and p_1' are set to the incoming values of q_2 and p_2 respectively.

Edge Filtering when bS = 4

When filtering with bS of 4, two filters may be used depending on sample content. For luma pixels, a very strong 4- or 5-tap filter, which modifies the edge values and two interior samples, if the condition

$$|p_0 - q_0| < \frac{\alpha}{4} + 2 \tag{2.7}$$

is met.

If equations 2.1 and 2.7 are not met, then a 3-tap filter is used to calculate q_0' and the values of q_1 and q_2 pass through the filter. Similarly, if 2.2 and 2.7 are not met, then a 3-tap filter is used to calculate p_0' and p_1 and p_2 pass through the filter. These calculations are

$$
\begin{aligned}
p_0' &= \frac{2p_1 + p_0 + q_1 + 2}{4}, \\
p_1' &= p_1, \text{and} \\
p_2' &= p_2;
\end{aligned}
$$

and

$$
\begin{aligned}
q_0' &= \frac{2q_1 + q_0 + p_1 + 2}{4}, \\
q_1' &= q_1, \text{and} \\
q_2' &= q_2.
\end{aligned}
$$

If the conditions of equations 2.1, 2.2 and 2.7 are met, the filtered values of $q_0' - q_2'$ and $p_0' - p_2'$ are calculated as

$$
\begin{aligned}
p_0' &= \frac{p_2 + 2p_1 + 2p_0 + 2q_0 + q_1 + 4}{8}, \\
p_1' &= \frac{p_2 + p_1 + p_0 + q_0}{4}, \text{and} \\
p_2' &= \frac{2p_3 + 3p_2 + p_1 + p_0 + q_0 + 4}{8}
\end{aligned}
$$

and

$$
\begin{aligned}
q_0' &= \frac{q_2 + 2q_1 + 2q_0 + 2p_0 + p_1 + 4}{8}, \\
q_1' &= \frac{q_2 + q_1 + q_0 + p_0}{4}, \text{and} \\
q_2' &= \frac{2q_3 + 3q_2 + q_1 + q_0 + p_0 + 4}{8}.
\end{aligned}
$$

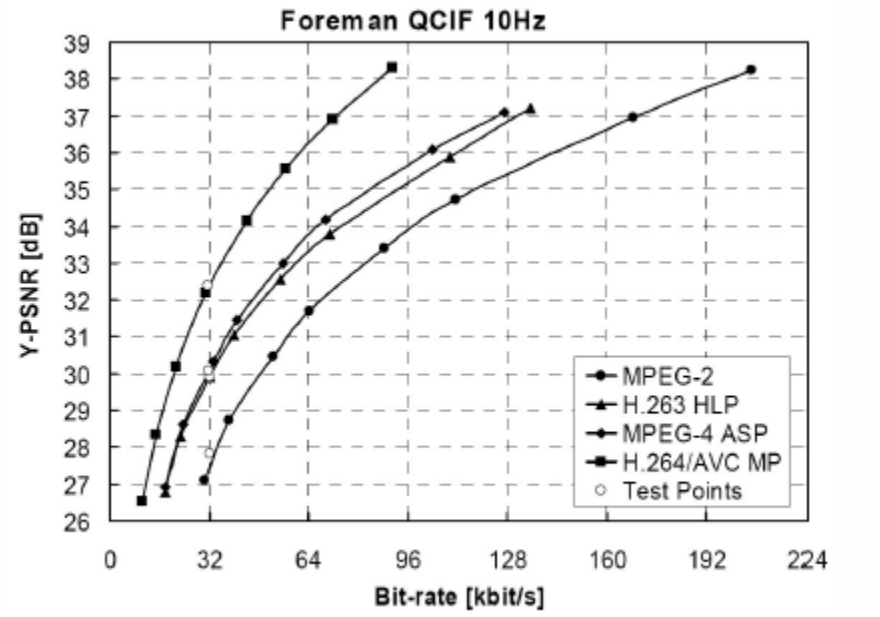

Figure 2.17: Filtering over macroblock in H.264 [13].

2.5 H.264 Performance

Wiegand et al. [13] measured the PSNR for given bit-rates on sequences encoded and decoded by a number of standards. They determined that the codec with the lowest PSNR for given bit-rates has the best performance. Figure 2.17 shows their results comparing PSNR against bit-rate for MPEG2, H.263, MPEG-4-ASP and H.264 (main profile). The stream used was the Foreman YUV test sequence. For a given bit-rate, H.264 had the highest PSNR.

When comparing entertainment workloads, they measured an average 45 % saving for H.264 over MPEG-2 for a given PSNR. These results, shown in figure 2.18, proved how important the use of the H.264 codec is for efficient compression and transmission of video streams.

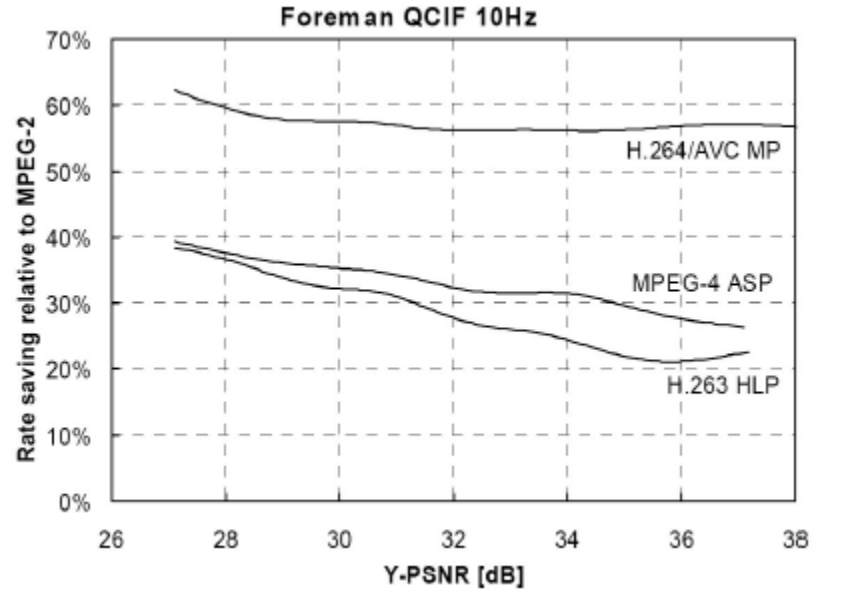

Figure 2.18: Filtering over macroblock in H.264 [13].

2.6 Summary

This chapter provided the reader with an overview of the H.264 standard. It was shown that the H.264 codec offers better video compression and quality over previous standards. However, the codec is very complex. A large portion of its complexity can be traced to the deblocking filter. Since deblocking is the final step that occurs before the video stream is output, a significant amount of research has been done in order to decrease the complexity of the deblocking filter.

Chapter 3

Previous Work

Due to its complexity, a significant amount of research has been undertaken into the implementation of deblocking filters for H.264 [5] [6] [8] [12] [14] [15] [16] [17]. The main two sources of complexity that have been studied are the high adaptivity of the filter and efficient memory management to take advantage of data dependence.

3.1 Memory Management Complexity

One source of filter complexity can be attributed to the fact that each pixel needs to be read multiple times during the filtering of a single macroblock. Filtering is carried out on the edges of each 4×4 pixel blocks with sixteen of these blocks existing. Additionally, to completely filter a macroblock, eight more 4×4 pixel blocks from the macroblocks adjacent to the left and above must be read.

The original edge filtering order as defined by the H.264 standard is shown in figure 2.15. The vertical borders of the luminance and chrominance blocks are all filtered before the horizontal borders. Since the results of the vertical filtering is used in the horizontal filtering, these intermediate results must be stored. This processing order is expensive in terms of memory use; it requires that twenty-four 4×4 blocks (sixteen luminance blocks and four blocks for each chrominance) be stored until horizontal filtering occurs. Also, this processing order does not take advantage of any data dependence.

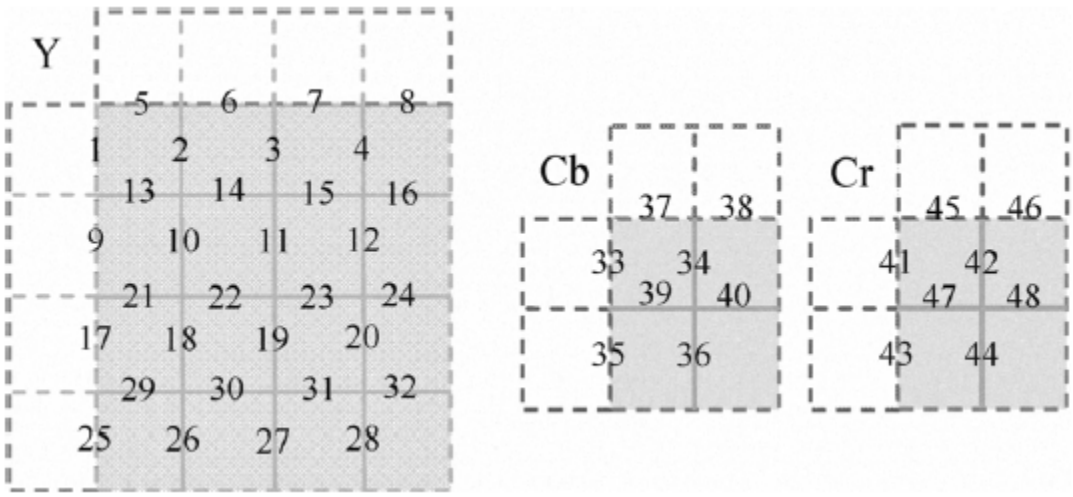

Figure 3.1: Filtering order propsed by Khurana et al. [16].

Another issue arises when filtering horizontal edges. Once a vertical edge is filtered, the data is misaligned and proper horizontal filtering cannot occur. This happens because when 4 horizontal pixels are stored in a line of memory (i.e. $p_3 - p_0$ in horizontal filtering), each pixel is in a different line in memory. This misalignment has caused many groups to research methods of removing these issues [12] [16] [17].

Khurana et al. [16] proposed a new filtering order shown in figure 3.1. The processing order is based on an alternation between the horizontal and vertical filterings of the blocks. This solution results in a decrease of the local memory size, as just one line of 4×4 blocks must to be stored in order to be used by the next filtering steps.

Sheng et al. [12] proposed a filtering order, shown in figure 3.2, that is based on the same alternation principle proposed by Khurana et al.In this case, the frequency of change between horizontal and vertical borders (and vice-versa) is higher, occurring after most of the 4×4 border filterings. As a result, there is a larger decrease in local memory used. Both of these processing orders rely on the use of transpose circuitry to resolve the misalignment issue.

The processing order proposed by Li [17] bases itself on both data reuse and concurrence principles. Shown in figure 3.3, this order significantly reduces the number of cycles required to filter a macroblock. This is achieved through the execution of horizontal and vertical

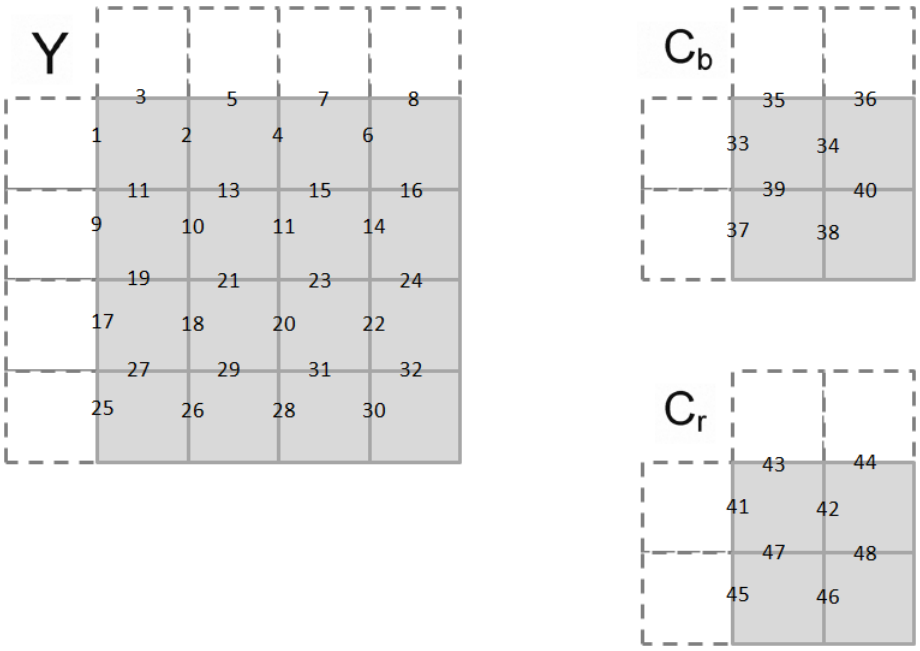

Figure 3.2: Filtering order propsed by Sheng et al. [12].

filterings at the same time. For that, two filters are needed, one for each filtering direction.

Corrêa et al. [15] took a different approach. Rather than using block level filtering, they based their processing order on sample level filterings. This allows the first line of pixels in a block to be filtered as soon as the first line of pixels in the previous block is available. This processing order is shown in figure 3.4. The processing order allows a better use of the architecture parallelism without significantly increasing the use of local memory. The repeated numbers correspond to the filterings which occur in parallel, in the same cycle, by different filter cores. While this design is very quick in terms of filtering a macroblock, it is a very large design in terms of the number of gates and registers required for implementation.

3.2 Complexity of High Adaptivity

Most research cites the high adaptivity of the filter as the greatest source of complexity [5] [14] [12] [15] [16] [17]. To handle this complexity, many strategies have been studied. Most

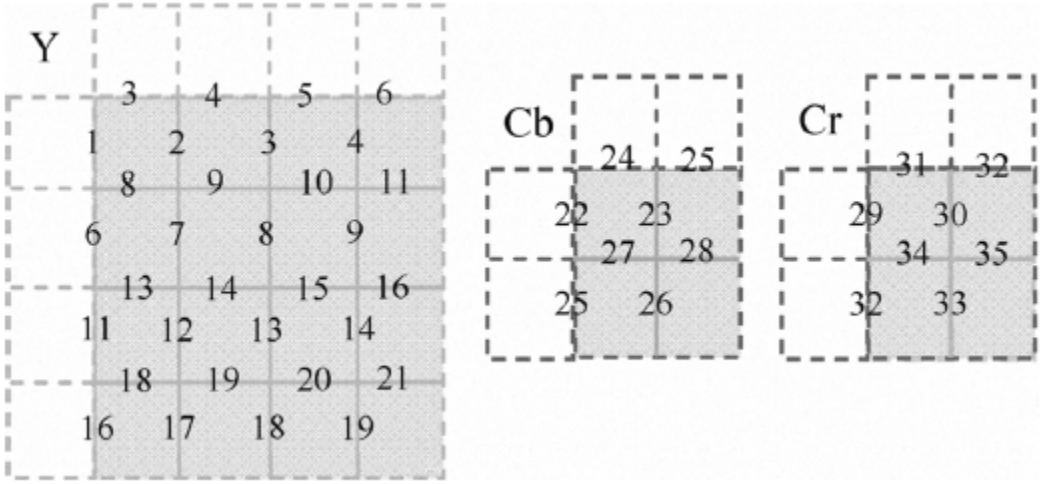

Figure 3.3: Filtering order propsed by Li et al. [17].

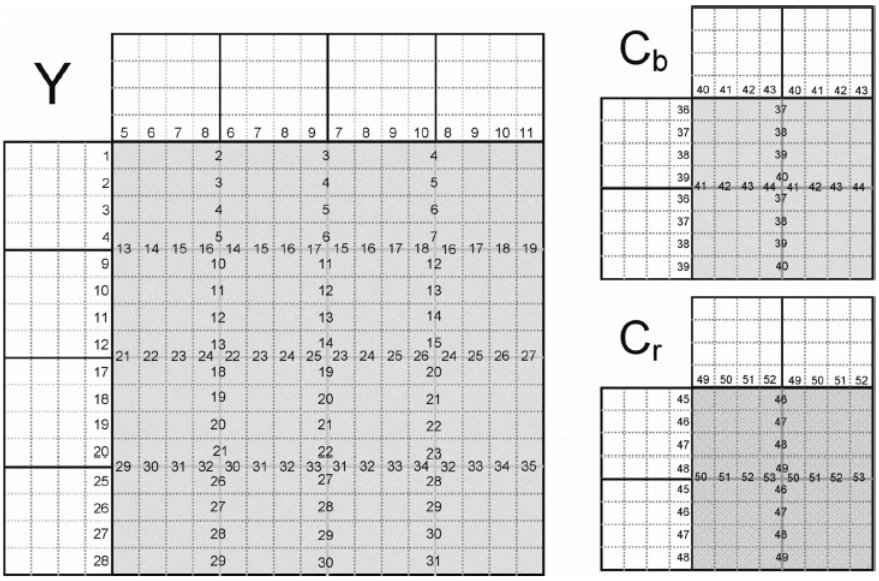

Figure 3.4: Filtering order propsed by Corrêa et al. [15].

approaches focus on pipelining the filter execution.

Khurana et al.[16] created a 4-stage pipelined design. This design is a sequential design hence no horizontal edges are filtered before the vertical edges are completed (see figure 3.1). This design only requires two transpose modules in order to account for the misalignment in the pixels. Their design is capable of filtering picture sizes of up to 2048×1024 and is only capable of filtering 30 FPS. Furthermore, while this filtering order and design increases the throughput of the deblocking filter and operating frequency (200 MHz), the number of clock cycles required to filter a single macroblock is almost 200.

As previously shown, the filtering order proposed by Sheng et al. [12] should have been a vast improvement, as only a single filter core is used. Rather, Sheng's design requires six transpose modules as opposed to the two used by Khurana. As such, 446 clock cycles are needed to filter a single macroblock, almost 2.5 times as many as Khurana. This design can only operate at 100 MHz and filter video streams as large as 1280×720. However, Sheng's design is capable of filtering streams at 62.3 FPS, an improvement over Khurana.

Ernst [8] chose to create a comparison between several designs; single-serial, single-concurrent, double-serial and double-concurrent filters. These designs are shown in figures 3.5 to 3.8. The single serial design operates with a single edge filter, filtering a single macroblock at a time. The data ordering used is advanced relative to the order defined in the standard, allowing for a greater temporal locality in the data. This design used the filtering order proposed by Sheng.

The single concurrent design expands on the first by exploring overlapped block filtering within a macroblock. This design results in a reduction in filtering time while keeping the amount of time needed for loading and storing a macroblock constant. The data ordering for this design used Li's filtering order.

The third design introduces parallel filtering at the macroblock level. Ernst noted the dependency on the macroblock to the left and above in a frame, meaning that a great deal of concurrency could be explored on the frame level. The same design from the single serial design is used, with modifications made at the frame-level. Two separate instances of the deblocking filter unit are used from the single edge filter design. A separate boundary strength calculation unit is needed for each deblocking filter. Two modified macroblock buffers are needed to support the two filters. The amount of storage for these buffers is

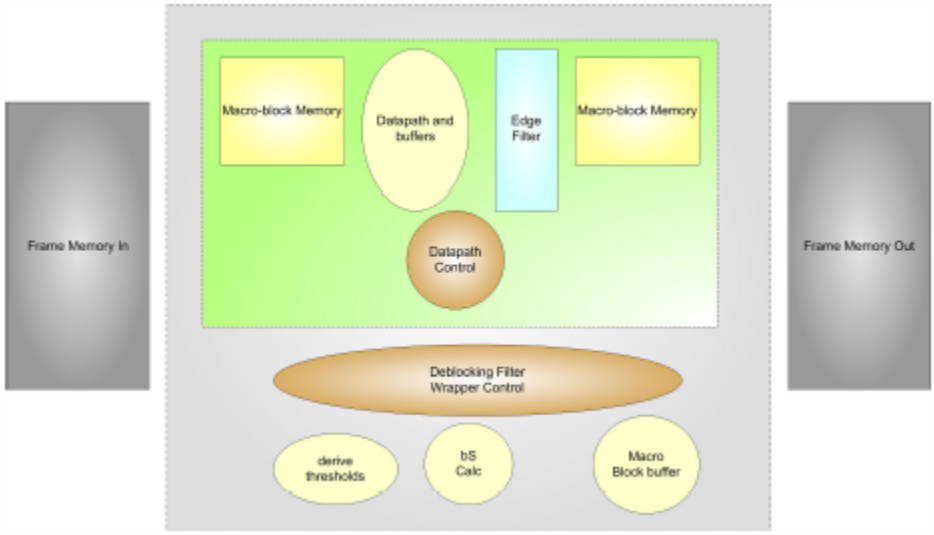

Figure 3.5: Ernst's single serial design [8].

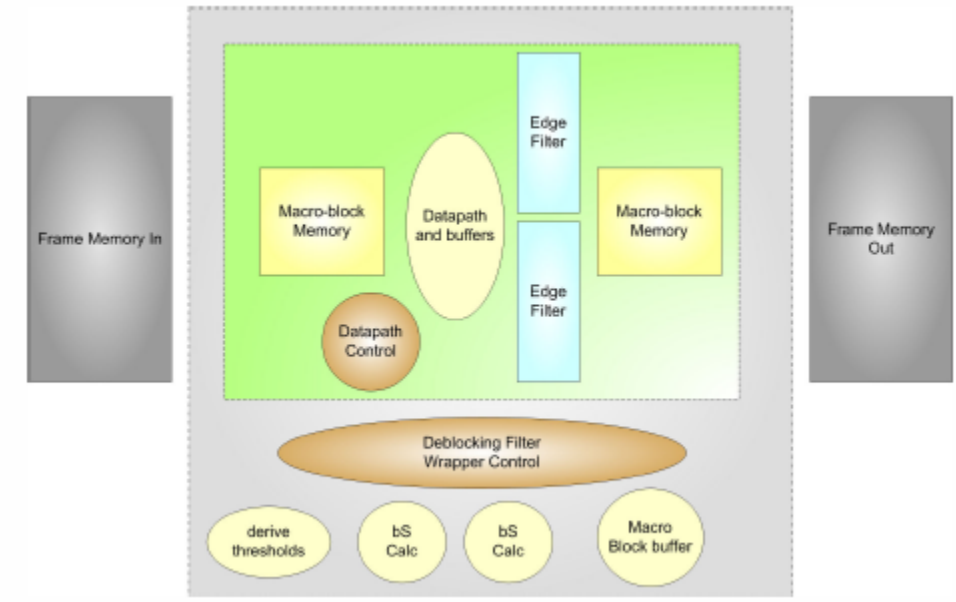

Figure 3.6: Ernst's single concurrent design [8].

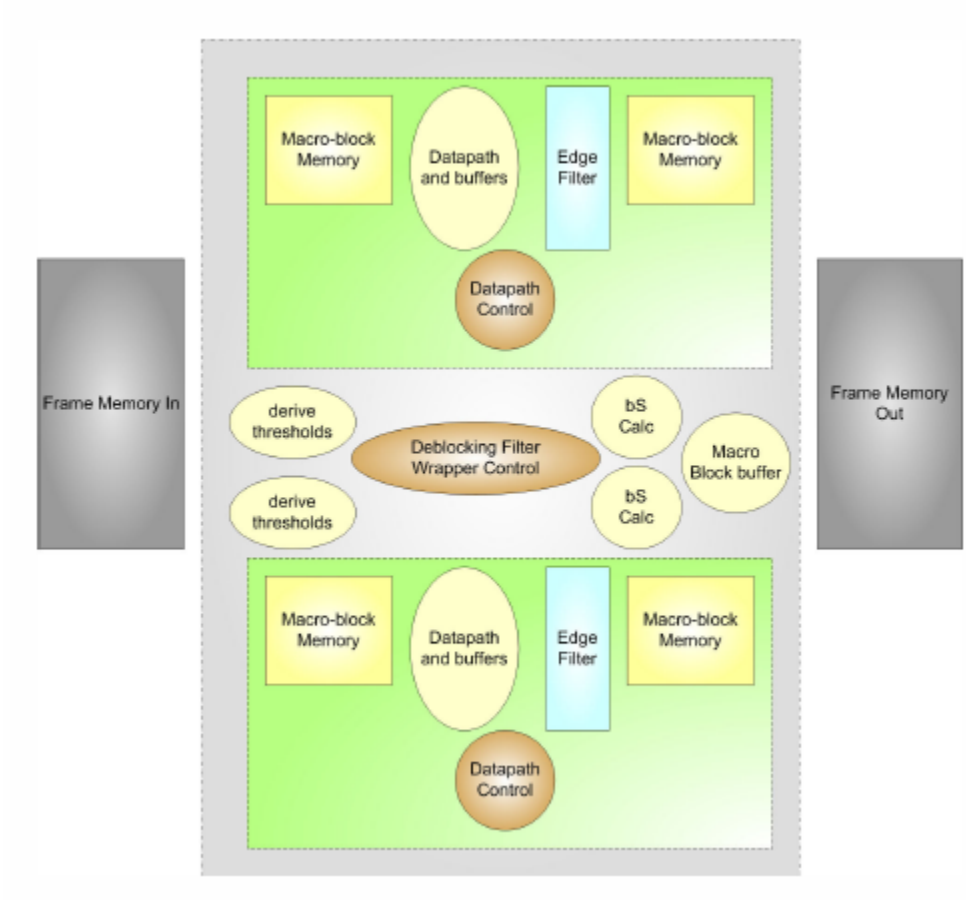

Figure 3.7: Ernst's double serial design [8].

less than that of two of the previous macroblock buffers; the buffer size does not need to be larger since an additional frame-wide row of samples is never needed. When the upper filter (that is, the filter which operates to the right of the first macroblock) completes, the bottom row of that macroblock becomes the top of the lower filter (that is, the filter which operates below the first macroblock). This allows marginal increases in macroblock buffer complexity.

The final design from Ernst is similar to the double deblocking filter with single edge filters, only with support for the deblocking filter with concurrent edge filters. As a result of this change, two additional boundary strength units are required, for a total of four. Slight modifications to the top controller are needed to support the extra boundary strength units

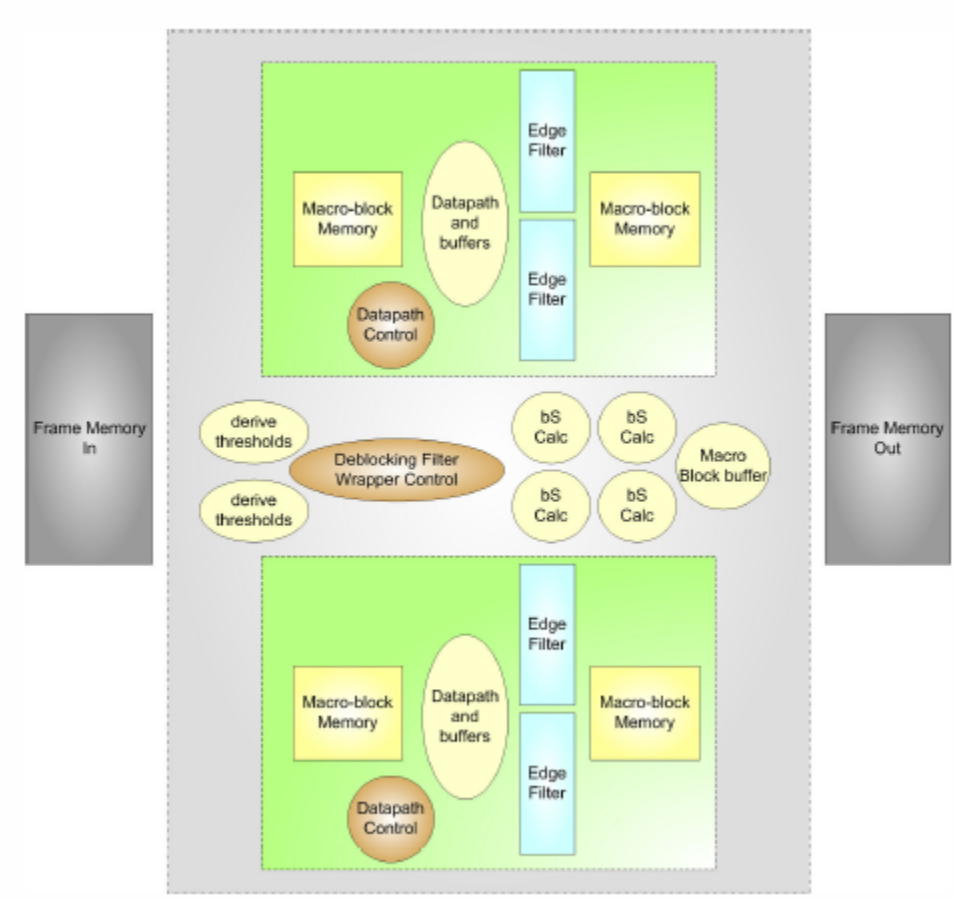

Figure 3.8: Ernst's double concurrent design [8].

and the modified interface. This last design is the most complex of the four.

The results of Ernst's designs show that the number of cycles required to filter a macroblock decreases as the design complexity increased. While the single serial design needed 200 clock cycles to filter, the double concurrent design only needs 75 clock cycles. However, the clock frequencies of the designs are very slow, ranging from 43 to 60 MHz. The double concurrent design can filter video streams at 77 FPS.

Corrêa's filter design uses only one bS calculator module, as opposed to the 4 in Ernst's double concurrent design. A significant difference in this design is that eight transpose matrices are used. Another difference in this design is that no block RAMs are used to store pixel samples, only dedicated registers. By using these registers, the number of clock

cycles to access the stored pixel information is reduced. This allows the design to filter a macroblock in 53 clock cycles and filter video streams at up to 130 FPS.

One unique technique proposed by Sima et al. [18] suggests implementing multiple pipeline paths each with different filters that are all executed. Each pipe would filter according to a single boundary strength and a select line would be calculated concurrently through the normal evaluation of bS. This select controls an output mux which selects the output from the proper filter pipeline. While this may speed filtering up, it is not a power-efficient design.

3.3 Summary

Many researchers have proposed novel designs to address the complexities of deblocking; these designs were described above. However, most have not been designed with a specific target in mind. Previous research presents designs compared against previous research which targets different profiles, use different technologies or target a specific area upon which to improve.

For this project, the deblocking filter must not only support all previous profiles present in the H.264 standard, it also must support the newer High 10 and High 4:2:2 profiles. Since this deblocking filter will be used in a commercial product, it must be small, fast and low on power consumption. The design of the deblocking filter is presented in the following chapter.

Chapter 4

Implemented Design

In the previous chapter, a number of different designs were presented. Ernst showed the a single concurrent design is the most efficient in terms of filter performance/gate count [8]. Furthermore, while slightly larger than a single serial design, a single concurrent design is much smaller than a double serial design and the design of Corrêa. With some optimizations, a single concurrent filter should easily meet the requirement of 60 FPS.

A new feature present in this deblocking filter is its ability to decode pixel information that is either 8 or 10 bits in length. Additionally, the deblocking filter can switch between 4:2:0 or 4:2:2 chroma subsampling when required. This chapter will discuss the design of the major components within the deblocking filter. A new filtering order is also used to improve the throughput of the deblocking filter.

4.1 Design Components

Figure 4.1 shows the high level block diagram of the deblocking filter. When the deblocking filter is ready, unfiltered pixel data from the current macroblock enters the filter at a rate of four pixels per clock cycle. This rate was determined to be the minimum rate at which 60 FPS would be achievable. The unfiltered pixels are stored in a block RAM, referred to as Unfiltered Pixel Ram in figure 4.1. Since the deblocking filter is targeted for use in a Spartan-6 FPGA, block RAMs are a better choice for storage instead of registers. This is because the number of available block RAMs is numerous.

As long as a single macroblock is present in the RAM, pixels enter the vertical filter core for filtering. Each filter core contains a second level of components needed for filtering, not shown in this diagram. Once the pixels are filtered, the Q' pixels are either fed back to the P input of the vertical core or a temporary buffer. The P' pixels are fed to a mux that selects which pixels enter the Q input of the horizontal filter core. The selected pixels are then transposed so the horizontal edge of the input pixels can be filtered. The data is filtered a second time, this time the horizontal edges are filtered. After the second filtering, the Q' pixels are then stored in another block RAM (shown as reference memory block) to be used later. The P' pixels are transposed once again, putting the pixels back in their original orientation, and then exit the filter.

4.2 Top Level Components

The following components are required for proper operation of the deblocking filter. Each component will be described in the following subsections.

4.2.1 Unfiltered Pixel Memory

A simple dual port block RAM is used to store the incoming unfiltered pixel data. The type of chroma subsampling (4:2:0 vs. 4:2:2) determines when the RAM is considered full and hence starts the filtering process.

4.2.2 Q' Pixel Buffer

A buffer is needed in the design to hold the Q' pixel results from the vertical filter which need to be reused for the filtering of a later edge. This buffer is designed to hold 4 rows of pixels, each containing 4 pixels. Since the edge filter operates on a single row of samples at a time, a simple shift register is used to hold the data. The interface for this shift register buffer can be seen in figure 4.2.

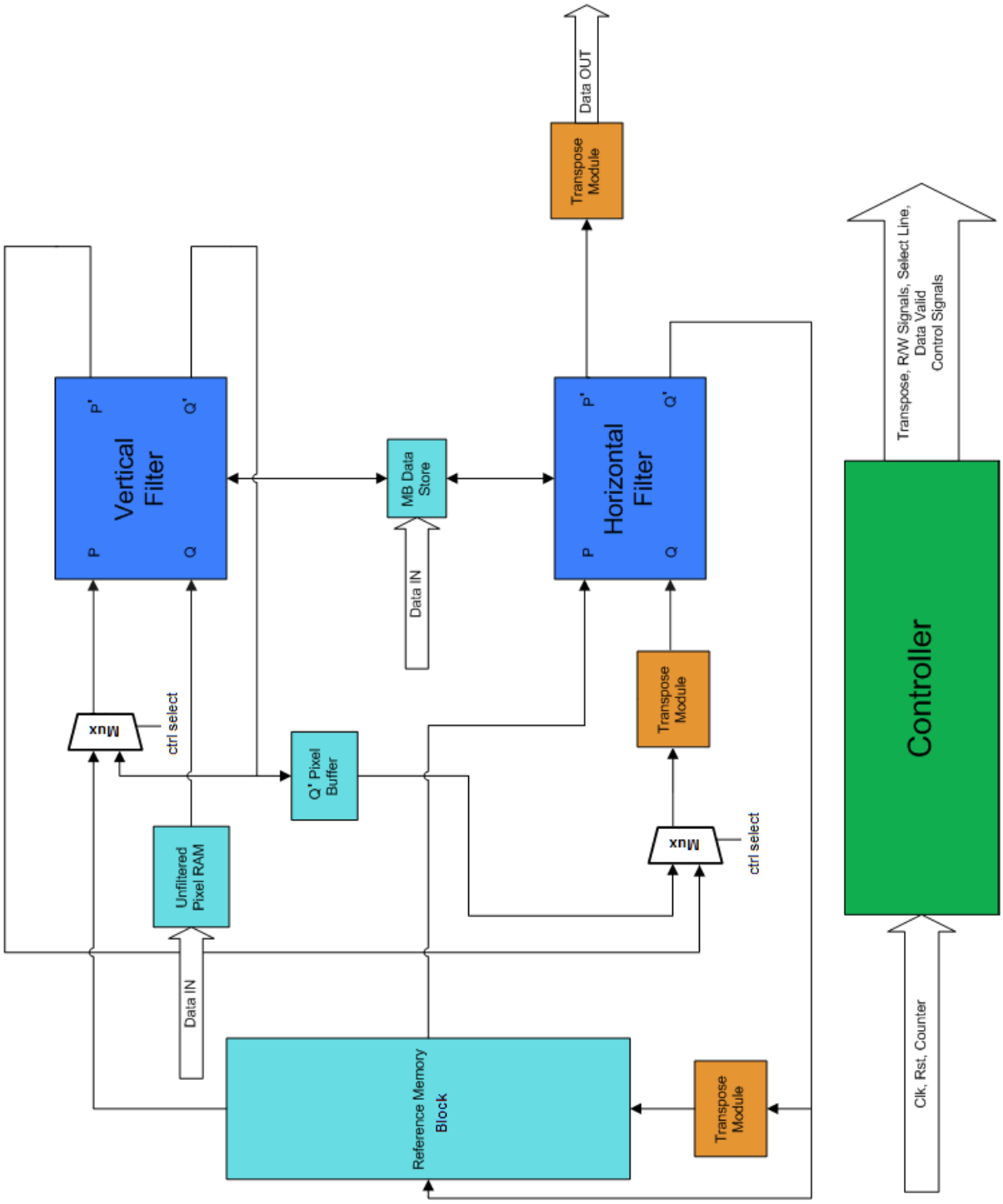

Figure 4.1: Architecture of designed deblocking filter.

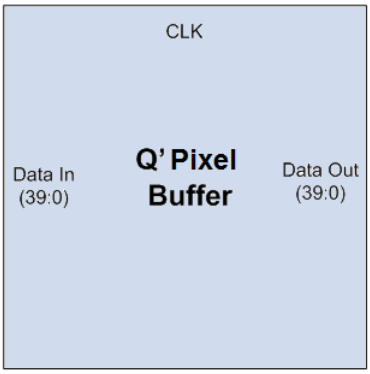

Figure 4.2: Vertical filter Q' pixel shift register

Inputs/Outputs to block	**Description**
CLK	System clock
Data In (39:0)	Four vertically filtered pixels. Each pixel is 10 bits in length.
Data Out (39:0)	Four vertically filtered pixels. Each pixel is 10 bits in length.

Table 4.1: Description of inputs and outputs to Q' pixel buffer.

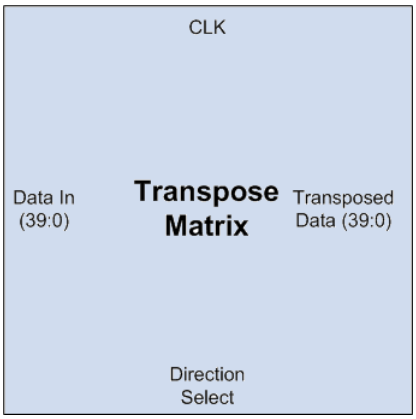

Figure 4.3: Transpose matrix module interface.

4.2.3 Transpose Module

When vertical edges are filtered, a single row of memory holds all the required edge samples. However, when horizontal filtering is performed, the sample data is stored in a single column across four rows of memory. To remedy this, a 4×4 transpose matrix is used to realign the data so horizontal filtering can be performed. A line of pixels is shifted in, and after 4 line of pixels are present in the matrix, a transposed 4×4 block is created by switching the direction select input. The interface for the matrix can be seen in figure 4.3. The design of the transpose module is shown in figure 4.4.

4.2.4 Controller

A control unit is required in order to ensure the proper filtering of a macroblock. The controller has two main functions. Its primary job is to issue requests to a pipeline that feeds the deblocking filter with new macroblocks for filtering. Its secondary job is to issue all the read and write controls and addresses for the local pixel data store and reference memories. The controller is also responsible for issuing the data valid signal, transpose direction and mux select controls.

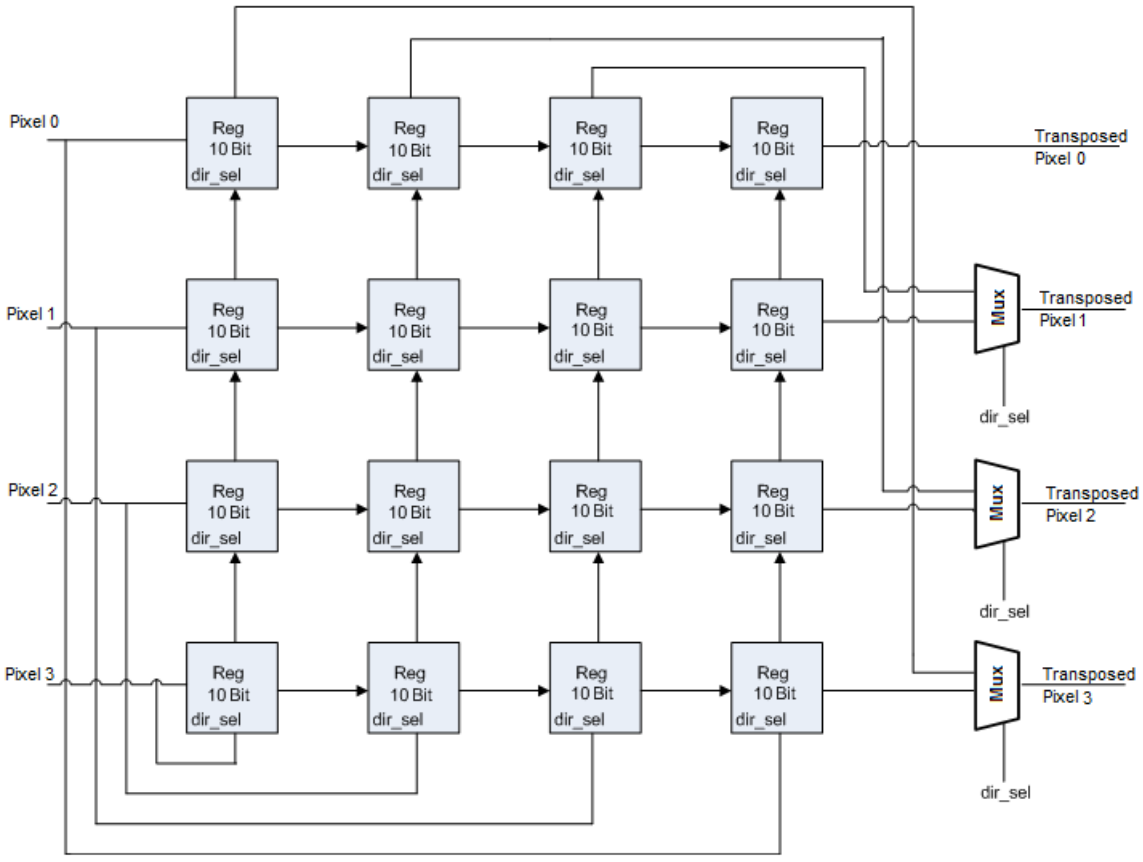

Figure 4.4: Block diagram of the matrix transpose module.

Inputs/Outputs to block	Description
CLK	System clock
Data In (39:0)	four input pixels. Each pixel is at most 10-bits in length.
Transposed Data (39:0)	Four transposed pixels. Each pixel is at most 10-bits in length.
Direction Select	Single bit input used to switch the output direction of the transpose module.

Table 4.2: Description of inputs and outputs to transpose module.

4.2.5 Reference Pixel Memory

The block labelled reference pixel memory contains two types of reference memory; storage for the vertical reference blocks and storage for the horizontal reference blocks.

Vertical Reference Pixel Memory

The blocks coloured blue in figure 4.5 are required in order to filter the current macroblock. To store these blocks, forty-eight lines of 40-bit wide memory is needed. In order to reduce the number of clock cycles needed to filter a macroblock, a dual port read write block ram is used. This allows a memory address to be overwritten with new data as soon as a line of pixels is ready from memory.

Horizontal Reference Pixel Memory

As seen in chapter 3, in order to filter a macroblock the bottom four 4×4 luma blocks and bottom two 4×4 blocks of each chroma block are needed. To store all this information, 5760 lines of 40-bit wide memory is required. This many addresses are needed since blocks 12-15, 18, 19, 22, 23, 26, 27, 30 and 31 from figure 4.5 must be stored until the next row of macroblocks are reached. The reference memory is also used as a temporary storage for

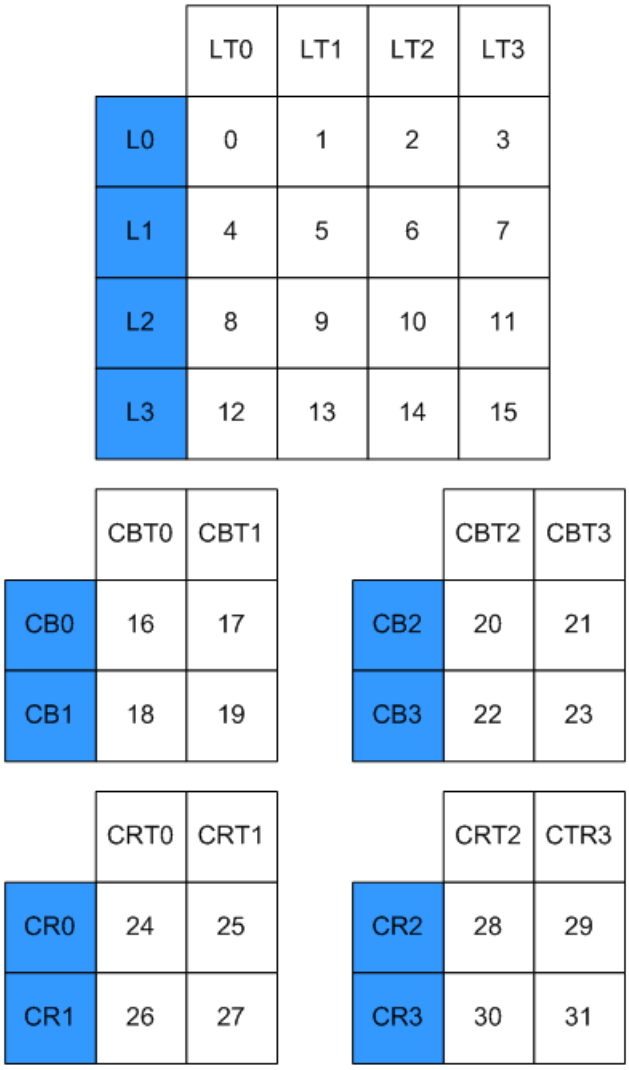

Figure 4.5: 4×4 blocks stored in vertical reference memory. Note, if 4:2:0 subsampling is chosen, blocks 24-31 are not present and CB2, CB3, CBT2 and CBT3 are renamed as CR0, CR1, CRT0 and CRT1 respectively.

the reference 4×4 blocks within the current macroblock and overwritten as needed. For example, if block 4 is being horizontally filtered, block 0 is stored in this memory and then overwritten with the filtered results of block 4. This process repeats until the bottom row of 4×4 blocks is reached. Figure 4.6 shows the memory layout.

4.2.6 Macroblock Data Store

Due to the nature of the deblocking algorithm, additional block RAMs are needed to store data from the previous macroblock (blocks B2, B7, B11 and B15) in addition to data from the previous 4×4 used. This data includes information such as the boundary strength value to be used for chroma block filtering, difference between the previous and current motion vectors, and transform levels.

4.3 Filter Core Architecture

Figure 4.7 shows a high level overview of the filter core. Eight pixels enter the filter core. From here, the pixels are sent to the calculation modules. Two calculation modules exist in the filter core: one for handling edges with boundary strengths between one and three and one for edges with boundary strength of four. A bypass path exists for when the filter is either disabled or no filtering is needed. Two additional modules are present in the core. These modules are used for the creation of the indexA, indexB, alpha, beta, t_{co} and boundary strength (bS) parameters.

For proper operation of the filtering algorithm, the pixel data and filter parameters must be pipelined. The initial deblocking filter design uses a 5-stage pipeline in each core. Figure 4.8 shows the execution flow for the first 12 clock cycles of the vertical filter. Since the boundary strength and threshold parameters each require a single clock cycle to calculate, a stall is introduced into the pipeline.

To eliminate the stall, a second pipeline design was for the filter core. This pipeline calculates the boundary strength and threshold parameters in the same clock cycle and therefore no stall is present. As a result, in every clock cycle, a new line of pixels can been fed to each filter core. However, this increases the longest path in the circuit reducing the

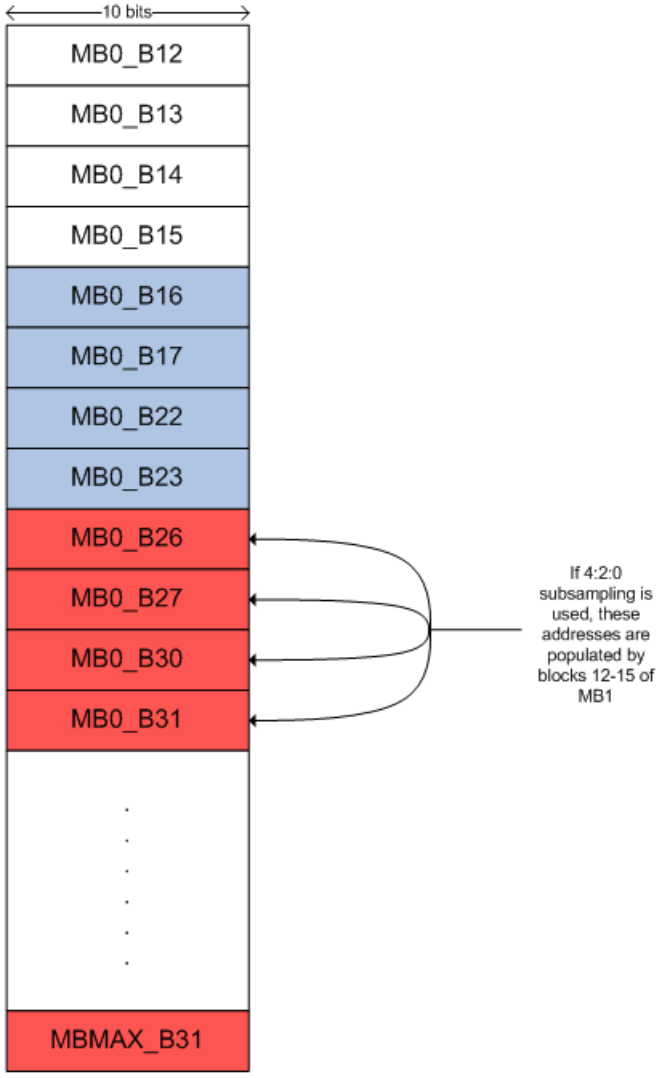

Figure 4.6: Horizontal reference memory layout.

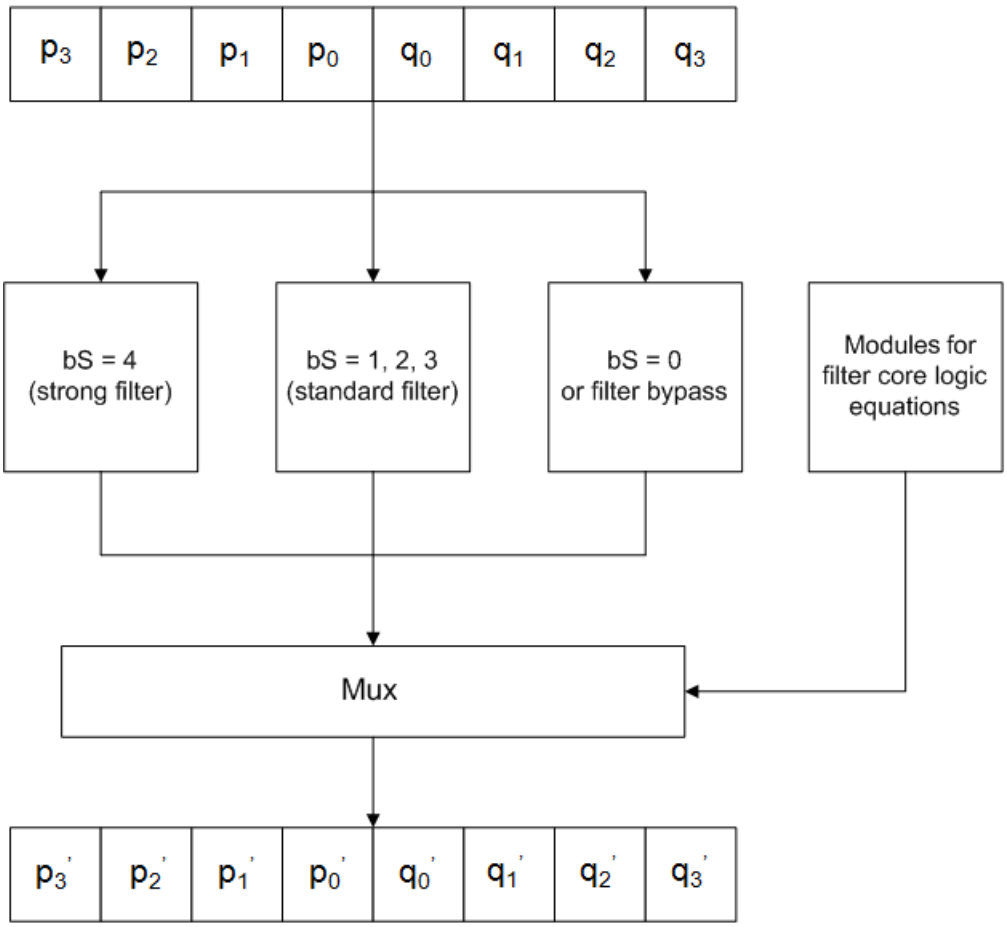

Figure 4.7: Architecture of edge filter.

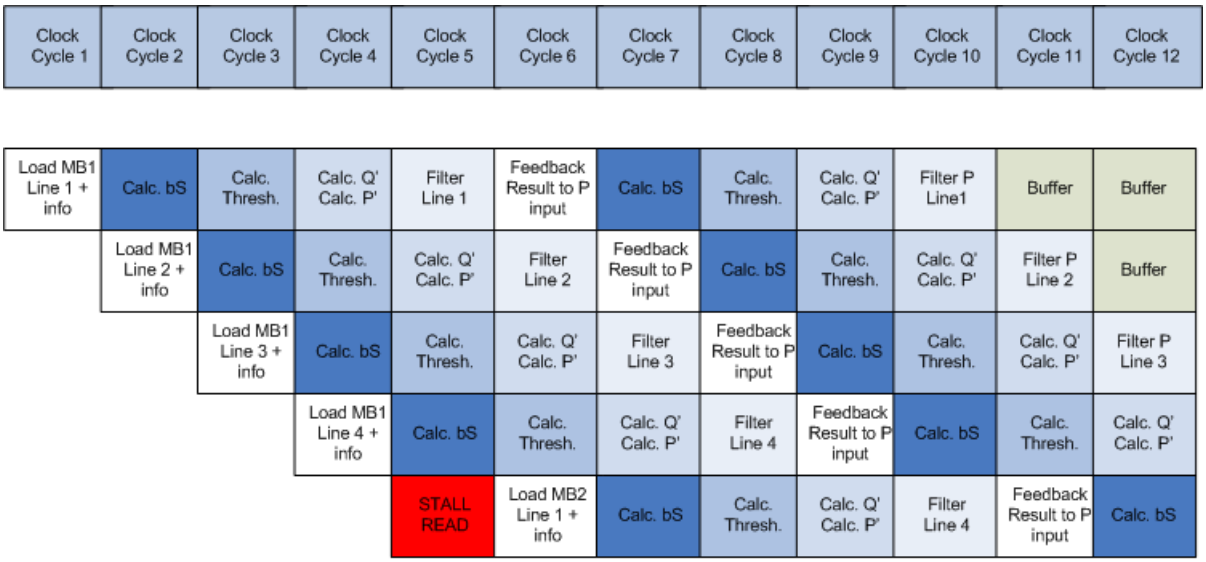

Figure 4.8: Execution flow of vertical filter (first 12 cycles).

operating frequency of the deblocking filter. Figure 4.9 shows the revised execution flow for the first twelve clock cycles of the vertical filter.

4.3.1 Threshold Parameters

The threshold parameter unit is responsible for the creation of indexA, indexB and bS. indexA and indexB are used for the creation of alpha, beta and t_{co}. The interface is shown in figure 4.10. The second threshold unit is responsible for the derivation of alpha, beta and t_{co}. These values, along with bS, are used to select the filtering strength. The interface of the second unit is shown in figure 4.11.

4.3.2 Filter Calculation Module

The filter calculation module is designed to carry out filtering on a row of eight pixel values shown in figure 4.12. The amount of filtering that takes place depends on the input bS, indexA, alpha, beta and the input pixels. The output of this logic block is 8 filtered pixels. The interface is the same for both the horizontal and vertical filtering cores.

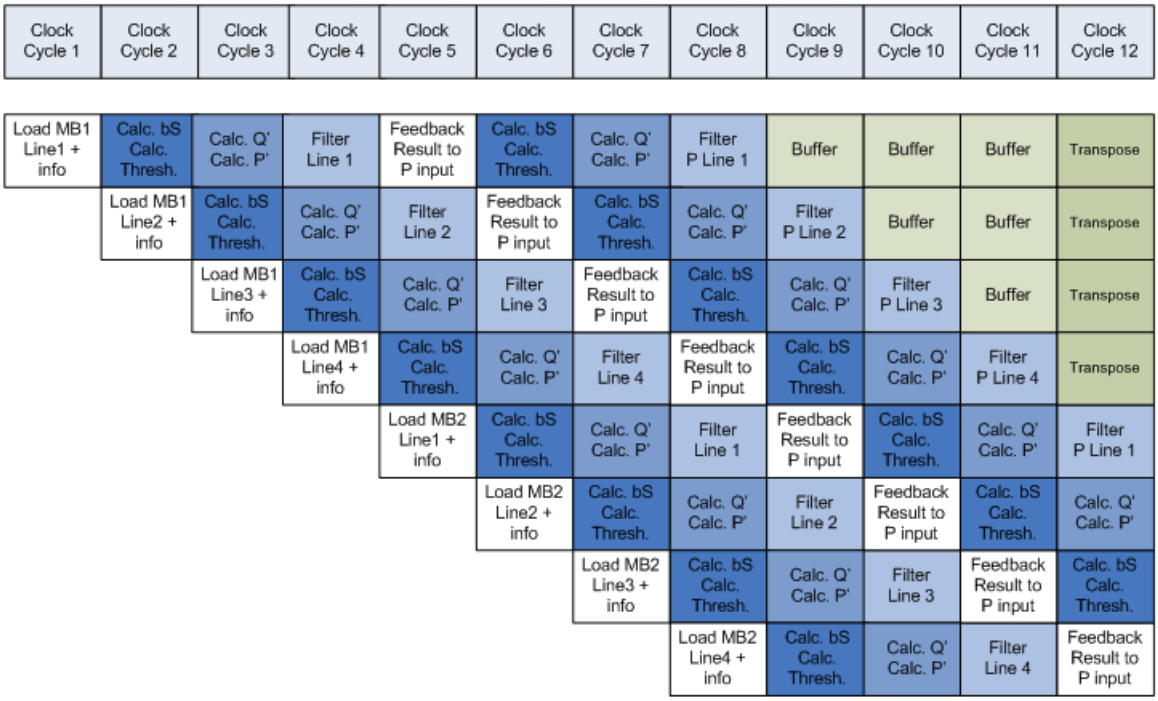

Figure 4.9: Execution flow of vertical filter of first 2 macroblocks (first 12 cycles).

Figure 4.10: Interface for bS calculator and index values.

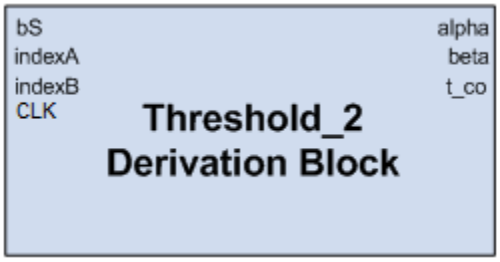

Figure 4.11: Interface for additional threshold values.

Inputs/Outputs to block	Description
CLK	System clock
chromaEdgeFlag	Bit indicating whether 4×4 block is a Luma or Chroma block
previous_intraPredictionMode	Bit indicating whether intra or inter prediction in the previous 4×4 was used
current_intraPredictionMode	Bit indicating whether intra or inter prediction in the current 4×4 was used
transform_levels	Bit indicating if the transform coefficient levels are non-zero. Used to calculate bS.
curr_slice_type	bS is dependant on the current slice type
prev_slice_type	bS is dependant on the previous slice type
qPy_values	bS is dependant on the Y quantization parameters
qPc_values	bS is dependant on the C quantization parameters
field_pic_flag	indicates if the current macroblock is field or frame coded (interlaced vs. progressive).
filterLeftMbEdgeFlag	bS changes if the currently filtered edge is at the left most edge of the active video region
filterTopMbEdgeFlag	bS changes if the currently filtered edge is at the top most edge of the active video region
slice_alpha_c0_offset_div2	indexA and indexB is dependant on the offset value present in the encoded header data
slice_beta_c0_offset_div2	indexA and indexB is dependant on the offset value present in the encoded header data
mv_dfferences	bS is dependant on how large the difference in the motion vectors are
bS	Filtering strength required for deblocking. required for deblocking.
indexA	Index used to determine alpha
indexB	Index used to determine beta

Table 4.3: Description of inputs and outputs to first Threshold Parameter module.

Inputs/Outputs to block	Description
CLK	System clock
bS	Filtering strength required for deblocking.
indexA	Index used to determine alpha
indexB	Index used to determine beta
alpha	Threshold value used in determining which tap filter is used
beta	Threshold value used in determining which tap filter is used
t_co	Limiting value for q_1' and p_1'. Used for $0 < \text{bS} < 4$

Table 4.4: Description of inputs and outputs to second Threshold Parameter module.

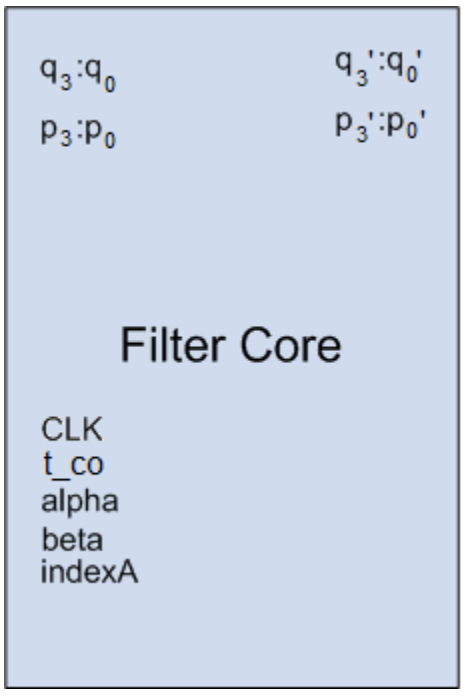

Figure 4.12: Filter core interface.

Inputs/Outputs to block	Description
CLK	System clock
indexA	Index used to determine alpha
alpha	Threshold value used in determining which tap filter is used
beta	Threshold value used in determining which tap filter is used
t_co	Limiting value for q'_1 and p'_1. Used for $0 < \text{bS} < 4$
q_3:q_0	Four input pixels from current macroblock
p_3:p_0	Four input pixels from reference macroblock
$q'_3 : q'_0$	Four filtered pixels from current macroblock
$p'_3 : p'_0$	Four filtered pixels from reference macroblock

Table 4.5: Description of inputs and outputs to filter core.

4.4 Implemented Design

As shown in figure 4.1, our design uses two edge filters to filter a single macroblock. The data ordering used is a modification on the overlapping order presented by Li [17]. This filtering order is fairly efficient, since once two vertical edges have been filtered, the horizontal edge can be filtered. While this horizontal edge is being filtered, a third vertical edge can be filtered. With Li's filtering order, all luma pixels have to be filtered before filtering a chroma pixel. The pipelines presented above allow chroma pixels to enter the vertical filter while luma pixels are still present in the pipeline. This reduces the number of clock cycles required for filtering. The modified order is shown in figure 4.13. By using this order and two edge filters, the amount of time needed to filter a macroblock is significantly reduced. Furthermore, by using this filtering order the amount of memory needed for storing filtered 4×4 blocks is also reduced.

In Ernst's single concurrent design, a total of seven transpose matrices were required to filter a macroblock. However, by taking advantage of the parallelism in the pixel ordering, less pixel buffering is needed so certain resources could be removed. As such, the design presented in this thesis only requires the use of four transpose matrices.

This chapter presented the reader with a description of the implemented design of the deblocking filter. Additionally, a description of each relevant component has been presented. As stated in the previous section, the selected implementation is the single

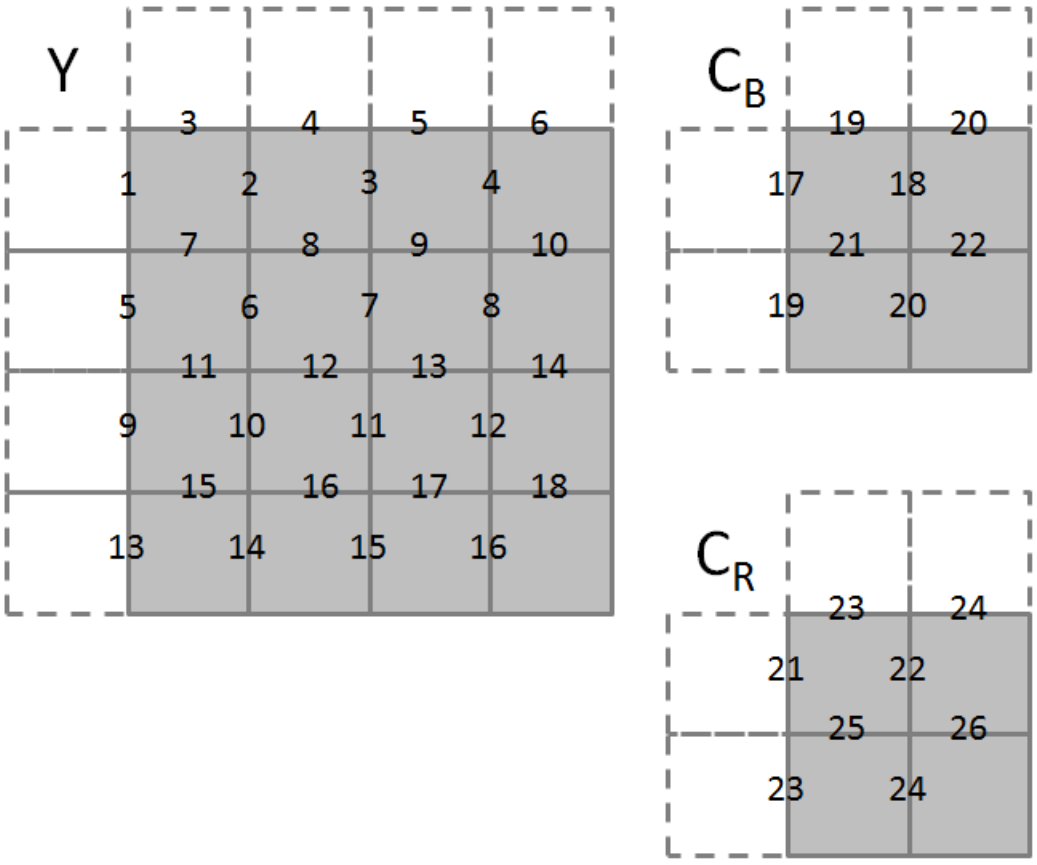

Figure 4.13: Modified filtering order.

concurrent design due to its fairly small size and having the highest performance per gate shown by Ernst [8]. The following chapter will present the synthesized results of each pipeline design, the results of testing and a comparison between these results and other deblocking filter designs.

Chapter 5

Design Results

As explained previously, the proposed deblocking filter architecture in this thesis was designed to filter both 8- and 10-bit pixel inputs as well as 4:2:0 or 4:2:2 chroma subsampling. In the previous chapter, the filter core was presented as using two different pipeline designs. In this chapter, the two different pipeline designs are evaluated to determine if they meet the project requirements of filtering video streams of up to 1080p at 60 FPS. Also, both pipelines are evaluated on the amount of resources they consume.

5.1 Test Streams

To test the filter's ability to remove blocking artifacts, a simple pipeline to feed the filter with new macroblocks was designed. The deblocking filter requests a new macroblock from the pipeline when it is ready to filter and feeds the filtered output back to the pipeline to be displayed. The filter and pipeline were placed in an existing H.264 decoder system developed by SOC Technologies. In addition to this, a input signal was tied to two dip switches on the Spartan-6 FPGA to test each filtering strength of the filter (bypass, standard filtering, strong filtering) in real time. The main rationale behind this was to test each filtering strength to ensure the filter was transposing and rebuilding each macroblock properly. Furthermore, there is no guarantee that each filtering strength would be tested in a single stream; manually selecting the filtering strength ensured proper filter operation.

Stream Number	Resolution	FPS	Chroma Subsampling	Bit Length
1	1080p	24	4:2:0	10
2	1080p	60	4:2:0	8
3	720p	60	4:2:0	8
4	1080i	30	4:2:0	8
5	720p	50	4:2:0	8
6	720p	24	4:2:2	10

Table 5.1: SOC Technologies teset streams.

Each filter was tested using test streams provided by SOC Technologies. These test streams varied in resolution from 480p to 1080p, 24 to 60 FPS, 4:2:0 and 4:2:2 chroma subsampling, pixel bit length as well as animated and live video streams. Table 5.1 shows a subset of the test streams used.

5.2 8-bit 4:2:2 Filter

The first design that was evaluated using the provided test streams was the 8-bit filter design. Each design was synthesized using Xilinx ISE 13.2 targeted for use on a Spartan-6 FPGA device.

5.2.1 5-stage Pipeline

Tables 5.2 and 5.3 below show the synthesis and performance results for the 8-bit filter using the 5-state pipeline.

The lowest frame rate will occur in 1080p video streams. Therefore, 1080p values will be used when calculating the maximum frame rate of the deblocking filter. The 1080p dimensions are divided by 16 because there are 16 pixels in a single line of a macroblock. To calculate the maximum frame rate of the deblocking filter we used

$$\text{Max Frame Rate} = \left(126 \text{ ns} \times \frac{1920}{16} \times \frac{1088}{16}\right)^{-1}$$

Synthesis Metrics	Value
Slice Registers	2675
Slice LUTs	4156
Fully Used LUT-FF Pairs	2316
BRAMs	26
Local Memory	38 kB

Table 5.2: 8-bit filter resource results for 5-stage pipeline.

Performance Metrics	Value
Cycles/MB (4:2:0)	146
Cycles/MB (4:2:2)	186
Max. Frame Rate (FPS) @ 1080p 4:2:0	839
Max. Frame Rate (FPS) @ 1080p 4:2:2	658
Minimum Period	8.567 ns
Max. Operating Freq.	116.724 MHz

Table 5.3: 8-bit filter performance results for 5-stage pipeline.

The 5-stage pipeline was capable of filtering all the test streams listed in table 5.1. In terms of performance, the 5-stage pipeline meets the requirement of a 100 MHz clock frequency and filtering 60 FPS streams. While the 5-stage pipeline meets the requirements listed in chapter 1, it finishes two clock cycles quicker than Ernst's single concurrent design.

5.2.2 Four-stage pipeline

Tables 5.4 and 5.5 below show the synthesis and performance results for the 8-bit filter using the 4-stage pipeline.

The 4-stage pipeline was able of filtering all the test streams listed in table 5.1. In terms of performance, the 4-stage pipeline meets the requirement of a 100 MHz clock frequency and filtering 60 FPS streams. This pipeline design filters a macroblock 22 clock cycles faster than Ernst's design. This shows that the 4-stage pipeline is the better design option. In terms of performance, the 4-stage pipeline is better than the 5-stage pipeline. The 4-stage pipeline operates at a higher frequency, filters a macroblock faster and can filter streams with a higher frame rate than the 5-stage pipeline.

Synthesis Metrics	Value
Slice Registers	2320
Slice LUTs	3726
Fully Used LUT-FF Pairs	2316
BRAMs	26
Local Memory	34 kB

Table 5.4: 8-bit filter resource results for 4-stage pipeline.

Performance Metrics	Value
Cycles/MB (4:2:0)	126
Cycles/MB (4:2:2)	162
Max. Frame Rate (FPS) @ 1080p 4:2:0	988
Max. Frame Rate (FPS) @ 1080p 4:2:2	756
Minimum Period	8.437 ns
Max. Operating Freq.	118.519 MHz

Table 5.5: 8-bit filter performance results for 4-stage pipeline.

Figure 5.1 shows a resource comparison between the two pipelines. This figure shows that 4-stage pipeline is the better choice for the 8-bit filter design. Looking at figure 5.1, it is easy to see that the 4-stage pipeline is an improvement over the 5-stage pipeline. There was an average resource reduction of 12.6 % when using the 4-stage pipeline. By using a 4-stage pipeline as opposed to a 5-state pipeline, a macroblock using 4:2:0 chroma subsampling can be filtered in twenty fewer clock cycles leading to a throughput speedup of 13.7 %. If 4:2:2 chroma subsampling is used, the 4-stage pipeline requires 24 fewer clock cycles, a speedup of 12.9 %. These results suggested that the 4-stage pipeline will be the better choice for the 10-bit design.

5.3 10-bit 4:2:2 Filter

The 10-bit design consumes more resources and has a slightly lower operating frequency than the 8-bit version. Again, each design was synthesized using Xilinx ISE 13.2 targeted for use on a Spartan-6 FPGA device.

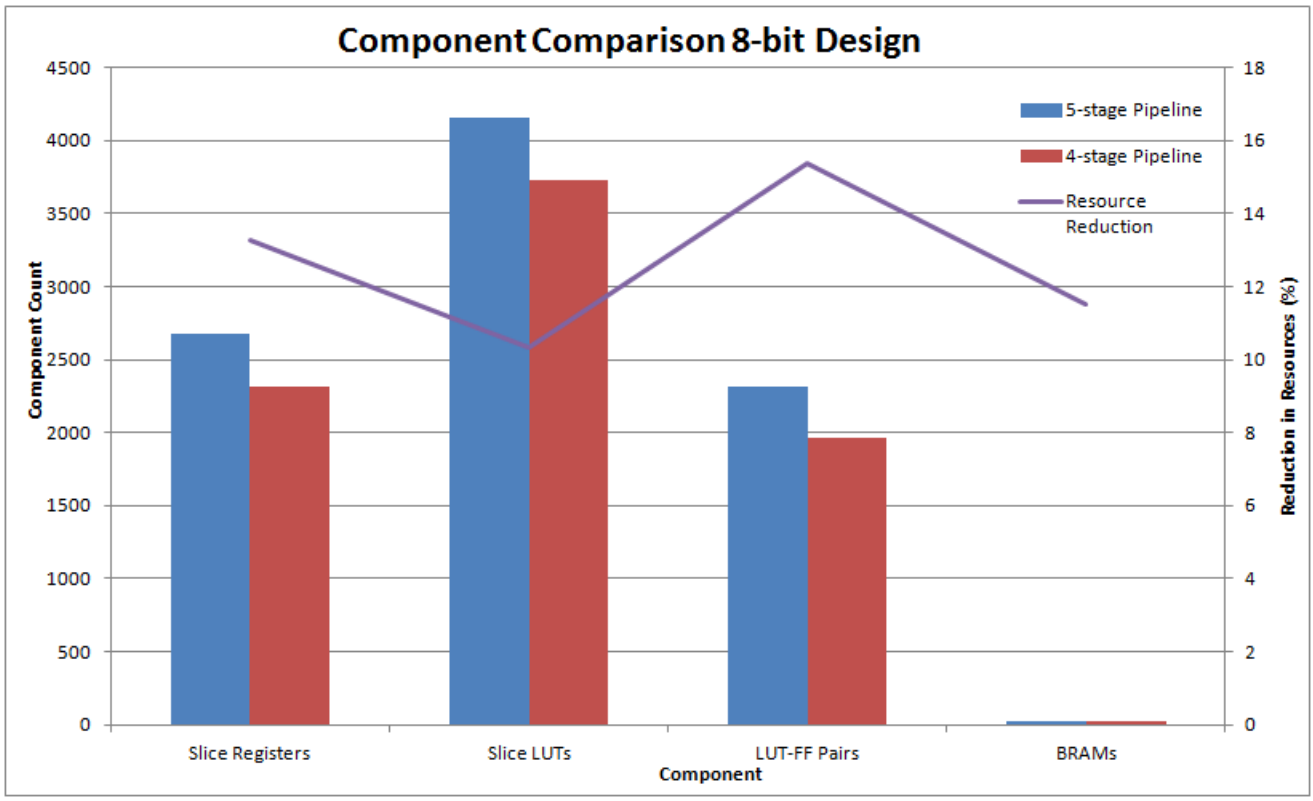

Figure 5.1: Comparison of resource usage in the 8-bit designs.

Synthesis Metrics	Value
Slice Registers	3220
Slice LUTs	4936
Fully Used LUT-FF Pairs	2785
BRAMs	32
Local Memory	46 kB

Table 5.6: 10-bit filter resource results for 4-stage pipeline.

Performance Metrics	Value
Cycles/MB (4:2:0)	146
Cycles/MB (4:2:2)	186
Max. Frame Rate (FPS) @ 1080p 4:2:0	839
Max. Frame Rate (FPS) @ 1080p 4:2:2	658
Minimum Period	8.899 ns
Max. Operating Freq.	112.372 MHz

Table 5.7: 10-bit filter performance results for 4-stage pipeline.

5.3.1 Four-stage Pipeline

Tables 5.6 and 5.9 below show the synthesis and performance results for the 10-bit filter using the 5-stage pipeline.

Again, the 5-stage pipeline was capable of filtering all the test streams listed in table 5.1. In terms of performance, the 5-stage pipeline meets the requirement of a 100 MHz clock frequency and filtering 60 FPS streams.

5.3.2 Five-stage Pipeline

The synthesized results are presented below in table 5.8. Again, the 10-bit design consumes more resources and has a slightly lower operating frequency than the 8-bit version.

As with the 8-bit filter, the 4-stage pipeline was capable of filtering all the test streams listed in table 5.1. In terms of performance, the 4-stage pipeline meets the requirement of a 100 MHz clock frequency and filtering 60 FPS streams. This pipeline design filters a

Synthesis Metrics	Value
Slice Registers	2772
Slice LUTs	4574
Fully Used LUT-FF Pairs	2353
BRAMs	28
Local Memory	42 kB

Table 5.8: 10-bit filter resource results for 4-stage pipeline.

Performance Metrics	Value
Cycles/MB (4:2:0)	126
Cycles/MB (4:2:2)	162
Max. Frame Rate (FPS) @ 1080p 4:2:0	988
Max. Frame Rate (FPS) @ 1080p 4:2:2	756
Minimum Period	8.750 ns
Max. Operating Freq.	114.284 MHz

Table 5.9: 10-bit filter performance results for 4-stage pipeline.

macroblock 16 clock cycles faster than Ernst's design. This shows that the 4-stage pipeline is the better design option. In terms of performance, the 4-stage pipeline is better than the 5-stage pipeline. The 4-stage pipeline operates at a higher frequency, filters a macroblock faster and can filter streams with a higher frame rate than the 5-stage pipeline.

Figure 5.2 shows a resource comparison between the two pipelines. This figure shows that 4-stage pipeline is the better choice for the 8-bit filter design. Looking at figure 5.2, it is easy to see that the 4-stage pipeline is an improvement over the 5-stage pipeline. There was an average of resource reduction of 12.3 % when using the 4-stage pipeline. Again, by using a 4-stage pipeline as opposed to a 5-stage pipeline, a macroblock using 4:2:0 chroma subsamplig can be filtered in twenty fewer clock cycles leading to a speedup of 13.7 %. If 4:2:2 chroma subsampling is used, the 4-stage pipeline requires 24 fewer clock cycles, a speedup of 12.9 %.

It should be noted that both deblocking filter designs were capable of decoding all the test streams with the required frame rate. Additionally, the 10-bit filter design was able to decode all 8-bit streams.

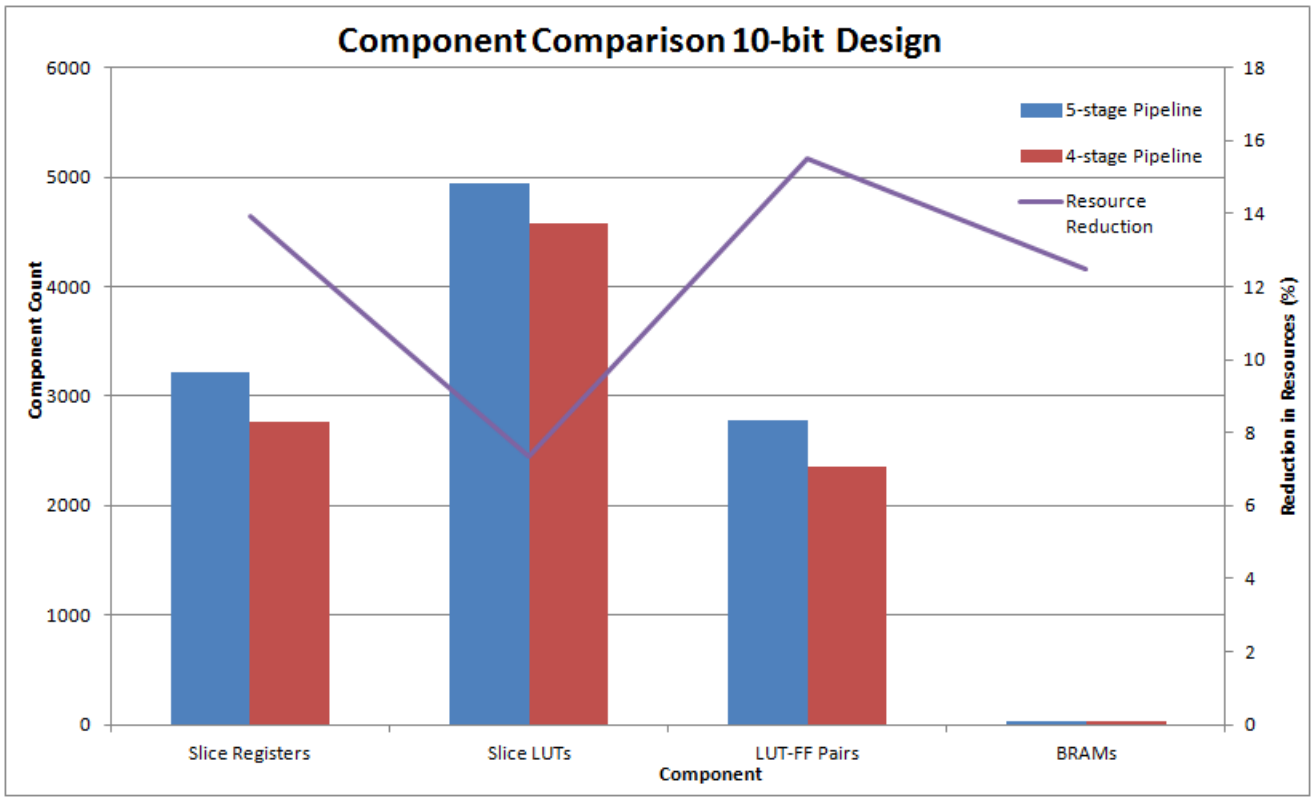

Figure 5.2: Comparison of resource usage in the 10-bit designs.

5.4 Critical Path

As shown above, the 4-stage pipeline is the better choice for the deblocking filter. It met all the requirements listed in chapter 1. Examining the timing report generated by the Xilinx ISE, the path limiting the operating frequency exists in the calculation module using a boundary strength of 4. The limiting path starts from the P pixel input and ends at an internal register named *signed_tc*.

Having the critical path present in this module is hardly surprising when one examines the equations used for filtering. This calculation module uses both 4-tap and 5-tap filters, meaning more resources must be used to ensure proper filtering. Therefore, should the operating frequency of the filter need to be increased, this path would have to be optimized.

5.5 Comparison to other work

A comparison of hardware resources used is not easy, since the technologies of the presented designs are different. It is also difficult to make an accurate comparison between the 10-bit design and the other designs since none state what pixel length they targeted. However, it is possible to compare the number of filter cores used in each design in addition to the amount of memory consumed. Only this design and the Ernst design [8] used two filtering cores. Every other solution used one core, with the exception of Corrêa who used 4 filtering cores [15].

Comparing the 8-bit design to Ernst's design, which uses the same number of filter cores, the design presented here is clearly more optimized in terms of LUTs and cycles needed to filter a macroblock (for 4:2:0 chroma subsampling). More cycles are used for 4:2:2 chroma subsampling, but this is expected since the additional chroma blocks also require filtering. It would be reasonable to assume that if Ernst was to redesign to account for 4:2:2 chroma subsampling, the number cycles needed for complete filtering would increase as well. Additionally, these designs have an operating frequency more than double that of Ernst. Significantly more BRAMs are used in these designs than by Ernst. This increase in block ram can be attributed to the changes between H.264 standards. Finally, the designs presented in this thesis were only tested with streams that have a maximum frame rate

	Tech.	Operating Freq. (MHz)	Slice LUTs or Gates	Memory Size	Cycles/MB	FPS @ HDTV
8-bit solution	Spartan-6	118.5	3726	23 BRAMs	126 (4:2:0)	988 (4:2:0)
					162 (4:2:2)	756 (4:2:2)
10-bit solution	Spartan-6	114.3	4574	30 BRAMs	126 (4:2:0)	988 (4:2:0)
					162 (4:2:2)	756 (4:2:2)
Corrêa[15]	Stratix III	270	7868	206 registers	53	130
Ersnt[8]	Virtex 5	43.08	7605	8 BRAMs	148	77
Rosa[6]	Virtex-2P	135	4331	64 BRAMs	256	116
Chen[19]	0.18μm	200	18700	384 bytes	222	30

Table 5.10: Comparison to other designs.

of 60 FPS. The calculations presented in the previous suggest that the deblocking filter will be capable of filtering streams when the next highger standard frame rate resolution is introduced.

Chapter 6

Conclusions

This thesis presented the implementation of a deblocking filter for the H.264 decoder developed by SOC Technologies. The presented architecture uses the processing order developed by Li [17]. The proposed designs are capable of decoding 4:2:0 and 4:2:2 chroma subsampling on demand. Additionally, a design has been presented capable of decoding 10-bit pixel inputs.

The architecture was designed in VHDL, synthesized and validated for an Xilinx Spartan-6 FPGA. The 8-bit and 10-bit designs reached a maximum operating frequency of 118 MHz and 114 MHz respectively. While the maximum operating frequencies are slower than most other presented designs, the SOC Technologies decoder operates at a maximum frequency of 100 MHz. As such, there was no need to optimize the design to further increase the minimum operating frequency.

A key component of the presented single concurrent design was the reduction in clock cycles by using a 4-stage pipeline as opposed to the initial 5-stage pipeline. This optimization reduced the number of clock cycles required to filter a macroblock by twenty, fourteen fewer clock cycles than Ernst's [8] single concurrent design. Furthermore, 14 % less LUTs are used in the 8-bit design than Ernst's [8] design.

The filters are capable of decoding HDTV at 1080p resolution at 60 FPS as per the requirements of the design. The presented designs filter 4:2:0 chroma subsampling in fourteen fewer clock cycles than the previous presented single concurrent design. Currently, there is no literature to which the 4:2:2 design can be compared.

Future work should investigate the use of a double concurrent design using the designed 4-stage pipeline to see if there is a reduction in the number of clock cycles needed to filter a macroblock. Additionally, it would be interesting to see how many resources are consumed by a double concurrent design.

Bibliography

[1] T. Wiegand, G. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 13, pp. 560–576, July 2003.

[2] I. Richardson, *H.264 and MPEG-4 Video Compression.* Wiley, 2003.

[3] T. Wiegand, G. Sullivan, and A. Luthra, "Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T rec.h.264 ISO/IEC 14496-10 AVC)," May 2003.

[4] ITU-T Rec. H.264/ISO/IEC 14496-10, "Advanced video coding for generic audiovisual services," Mar. 2005.

[5] P. List, A. Joch, J. Lainema, G. Bjøntegaard, and M. Karczewicz, "Adaptive deblocking filter," *IEEE Trans. on Circuits and systems for Video Technology*, vol. 13, pp. 715–727, 2003.

[6] V. Rosa, A. Susin, and S. Bampi, "A high performance H.264 Deblocking Filter," *PSIVT 2009*, pp. 955–964, 2009.

[7] "H.264/MPEG-4 AVC video compression tutorial." [Online]. Available: http://www.lsilogic.com

[8] E. Ernst, "Architecture design of a scalable adaptive deblocking filter for H.264/AVC," Master's thesis, Rochester Institute of Technology, Rochester, NY, 2007.

[9] User:Mikus, "File:interlaced video frame (car wheel).jpg." [Online]. Available: http://en.wikipedia.org/wiki/File:Interlaced_video_frame_(car_wheel).jpg

[10] C. Poynton, "Chroma subsampling notation." [Online]. Available: http://www.poynton.com/PDFs/Chroma_subsampling_notation.pdf

[11] T. S. S. O. ITU, "Series H: Audiovisual and multimedia systems infrastructure of audiovisual services," 2010.

[12] B. Sheng, W. Gao, and D. Wu, "An implemented architecture of deblocking filter for H.264/AVC," in *2004 International Conf. on Image Pfocessing (ICIP)*, 2004, pp. 665–668.

[13] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. J. Sullivan, "Rate-constrained coder control and comparison of video coding standards," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 13(7), pp. 688–703, 2003.

[14] M. Parlak and I. Hamzaoglu, "A low power implementation of H.264 adaptive deblocking filter algorithm," 2007.

[15] G. Corrêa, T. Silva, L. A. Cruz, and L. Agostini, "Design of an interlayer deblocking filter architecture for H.264/SVC based on a novel sample-level filtering order," *2009 IEEE Workshop on Signal Processing Systems*, pp. 102–108, 2009.

[16] G. Khurana, T. Kassim, T. Chua, and M. Mi, "A pipelined hardware implementation of in-loop deblocking filter in H.264/AVC," *IEEE Tran sactions on Consumer Electronics*, 2006.

[17] L. Li, S. Goto, and T. Ikenaga, "A highly parallel architecture for deblocking filter in H.264/AVC," *IEICE Trans. on Information and Systems*, 2005.

[18] M. Sima, Y. Zhou, and W. Zhang, "An efficient architecture for adaptive deblocking filter of H.264/AVC video coding," *IEEE Trans. on Consumer Electronics*, vol. 50(1), pp. 292–296, 2004.

[19] Q. Chen, W. Zheng, J. Fang, K. Luo, B. Shi, M. Zhang, and X. Zhang, "A pipelined hardware architecture of deblocking filter in H.264/AVC," in *3rd International Conf. on Communications and Networking in China,*, 2008, pp. 815–819.

[20] SOC, "SOC technologies." [Online]. Available: http://www.soctechnologies.com