UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

FELIPE VOGEL DALCIN

# A Deblocking Filter Architecture for High Efficiency Video Coding Standard (HEVC)

Final report presented in partial fulfillment of the Requirements for the degree in Computer Engineering.

Advisor: Prof. Dr. Sergio Bampi
Co-advisor: MSc. Cláudio Machado Diniz

Porto Alegre
2014

*“Docendo discimus.”*
Seneca

## AKNOWLEDGEMENTS

I would like to express my gratitude to my advisor, Prof. Dr. Sergio Bampi, who accepted me as lab assistant in 2010 to work for the research group under his supervision, bringing me the opportunity to stay in touch with a high-level academic environment.

A very special thanks goes out to my co-advisor, MSc. Cláudio Machado Diniz, whose expertise, understanding, patience and unconditionall support were vital to the accomplishment of this work, specially taking into considerations the reduced amount of time he had to review this work.

Last but not least, I would like to thank my parents and friends for their unconditional support throughout my degree. A special remark goes to my girlfriend, who kept me motivated during the last weeks of work to get this report done.

In conclusion, I recognize that this research would not have been possible without the opportunities offered by both the Universidade Federal do Rio Grande do Sul, in Brazil, and the Technische Universität Kaiserslautern, in Germany.

**ABSTRACT**

The new video compression standard High Efficiency Video Coding (HEVC) contains a variety of new encoding tools to provide higher compression rate compared to previous H.264/AVC standard. The higher compression rate achieved in HEVC results in an increase in computational complexity for video encoding and decoding. The need to encode and decode ultra-high definition video sequences under real-time constraints demands hardware architectures to accelerate execution time of certain HEVC tools, due to their high computational complexity and to the huge amount of data that needs to be processed. In this work we discuss the complexity of HEVC tools and we propose a hardware architecture for Deblocking Filter. We discuss in details the implementation of the hardware datapaths that implement the filtering conditions and the filtering operations, as well as the control unit of our architecure.

**Keywords**: Digital Video Coding, High Efficiency Video Coding Standard, Deblocking Filter, Hardware Architecture

## RESUMO

O novo padrão de compressão de video HEVC (*High Efficiency Video Coding*) contém uma variedade de ferramentas de codificação que fornecem taxas de compressão mais altas em comparação com o antigo padrão H.264/AVC. As taxas de compressão mais altas atingidas pelo HEVC resultam em um aumento da complexidade computacional para codificar e decodificar vídeo. A necessidade de codificar e decodificar vídeos de definição ultra alta (UHD) atendendo a restições de aplicações tempo-real exigem que arquiteturas de hardware sejam desenvolvidas para acelerar o tempo de execução de algumas ferramentas do HEVC, devido a suas altas complexidades computacionais e ao grande volume de dados que necessita ser processado. Neste trabalho, nós discutimos algumas das ferramentas do HEVC e as complexidades de algumas delas, a fim de propor uma arquitetura para o filtro de deblocagem (*Deblocking Filter*). Nós discutimos em detalhes a implementação dos datapaths do hardware que implementam as condições e operações de filtragem, além da unidade de controle de nossa arquitetura.



**Palavras-chave**: Codificação de Vídeo Digital, Padrão de Codificação de Vídeos de Alta Eficiência, Filtro de Deblocagem, Arquiteturas de Hardware

## LIST OF FIGURES

## LIST OF TABLES

## LIST OF ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| ASIC | Application Specific Integrated Circuit |
| AVC | Advanced Video Coding |
| Bs | Boundary Strenght |
| CABAC | Context Adaptive Binary arithmetic Arithmetic Coding |
| CB | Coding Block |
| CU | Coding Unit |
| DBF | Deblocking Filter |
| DBF | Deblocking Filter |
| DCT | Discrete Cosine Transform |
| FPGA | Field-programable gate Gate arrayArray |
| FSM | Finite State Machine |
| GB | Giga-byte |
| HD | High definition |
| HEVC | High Efficiency Video Coding |
| I/O | Input/Output |
| JCT-VC | Joint Collaborative Team on Video Coding |
| JVT | Joint Video Team |
| LCU | Largest Coding Unit |
| MC | Motion Compensation |
| ME | Motion Estimation |
| MV | Motion Vector |
| PU | Prediction Unit |
| QP | Quantization Parameter |
| Qstep | Quantization Step |
| RGB | Red, Blue and Green |
| SAD | Sum of Absolute Differences |
| SAO | Sample Adaptive Offset |
| TU | Transform Unit |
| UHD | Ultra-high definition |
| VHDL | VHSIC Hardware Description Language |
| YCbCr | Luminance, Blue Chrominance and Red Chrominance |

**CONTENTS**

# 1 INTRODUCTION

The increasing number of multimedia devices capable of recording and reproducing video has stimulated the growth and dissemination of digital video. Consumers can find a wide variety of these devices in the market, such as tablets, smartphones, camcorders, high definition televisions, among others. The increasing popularity of digital video can also be noted due to the increase in internet traffic. Projections foresee that about 700 billion minutes of video will be downloaded in 2015 (Bull, et al., 2011).

Since new coming devices are capable of capturing high resolution (HD) and ultra-high resolution (UHD) video in real time, huge amount of data is produced for video storage or transmission. Therefore, video compression plays an important role in order to reduce the required amount of video data to be stored or transmitted, whilst preserving high visual quality video.

Among the different existing video coding standards, the H.264/Advanced Video Coding (AVC) (ITU, 2003), which has been developed by the Joint Video Team (JVT), is the video coding standard currently used in the market. It has been released in 2003 and produced double compression rate if compared to the previous dominant standard in the market, the MPEG-2 (ITU, 1994).

Even though the H.264/AVC represented an important breakthrough in video encoding, the demand for ultra-high resolution video requires even higher compression rates. To address this demand, the Joint Collaborative Team on Video Coding (JCT-VC) was created in 2010 aiming to develop a new video encoding standard. High-Efficiency Video Coding (HEVC/H.265) (ITU-T, 2013) achieves approximately 50% bit-rate reduction compared with H.264/AVC (Sullivan, 2012).

Such a compression improvement results in an increase of computational complexity, since each of the video coding tools are improved compared with H.264/AVC. Therefore, dealing with more complex data structures and the addition of a much more complex decision tree has to be considered when implementing this new standard.

Hardware acceleration is often employed to address the high complexity of video coding tools when targeting ultra-high resolution video encoding in real time (Diniz, 2013) (Afonso, 2012). The objective of this work is to analyze the HEVC standard and propose a dedicated hardware architecture for a selected tool (the in-loop deblocking filter). In this work we analyzed and proved that the deblocking filter represents a computational hot spot in the software implementation of the HEVC standard, thus requiring dedicated hardware in order to meet throughput and power consumption constraints.

This work is organized as follows. Chapter 2 reviews video coding concepts and gives an overview of the new HEVC standard. Chapter 3 shows a detailed overview about deblocking filter, the focus of this work. Chapter 4 introduces the proposed hardware architecture for deblocking filter. Chapter 5 shows results a comparisons with related work. Chapter 6 concludes this work.

# 2 VIDEO CODING CONCEPTS AND THE HEVC STANDARD

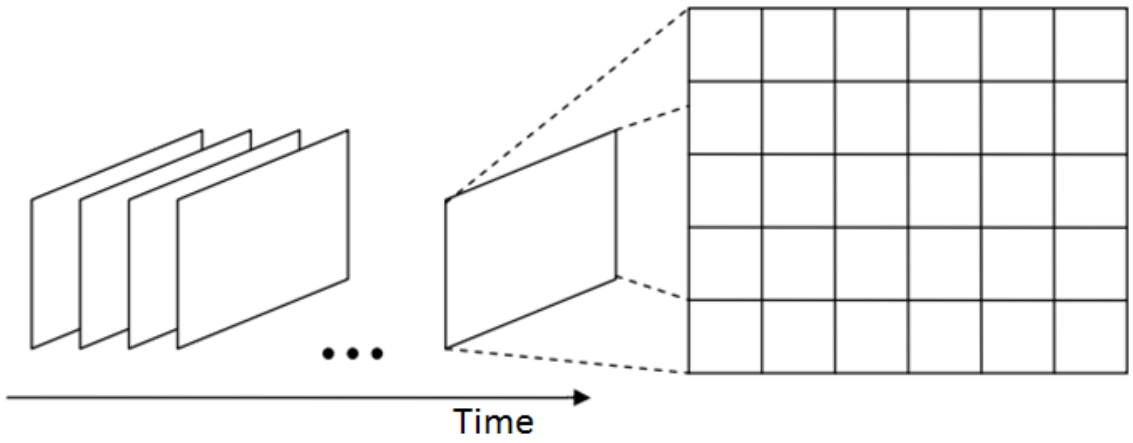
## 2.1 Video capturing and representation

Recording and transmitting digital uncompressed video require a huge amount of data storage capability and a very large bandwidth, which represents a wastage of resources and lead to increasing cost.

A digital video is a set of digital sequential pictures. Each picture is divided into blocks, constituted by pixels. The pixel is the smallest element of an image, and it is usually represented by samples in three different color components, i.e. red (R), green (G) and blue (B), when considering a RGB color space. For video compression purpose, video is often represented in YCbCr color space, in which Y is luminance component, Cb is the chrominance blue component and Cr is the chrominance red component (more details in Section 2.4).

In order to achieve the motion feeling, each picture has to be sequentially displayed at a determined picture-rate. The smoothness of video display strongly depends on the picture rate that is being used. Figure 2.1 shows the sequentially ordered pictures, as well as each block inside of a picture.

Figure 2.1 - Picture sequence of a video and the division of a picture into blocks




Source: Porto, 2008

Considering a 10-minutes raw video sequence with 1024x768 pixels resolution and 24 bits per pixel (8-bits for each component) recorded at a 30 pictures per second

rate, 19 GB would be needed for it to be stored. By using video compression techniques, such amount of data can be substantially reduced.

## 2.2 Lossless and lossy compression

Video compression is divided into 2 different categories: (i) lossless compression and (ii) lossy compression. The first technique achieves compression preserving the same video at the end of a compression/decompression process, i.e. the original video remains unmodified and can be recovered by decompressing the compressed video. On the other hand, lossy compression techniques consist on the elimination of some of the video data available in the input during the compression process. This technique eliminates part of data of video, in a way that the quality of video is satisfactory after decompression. Considering the huge amount of data that ultra-high resolution video applications require, lossy compression techniques are the most used ones in order to achieve low bit-rates.

## 2.3 Redundancies in video

Videos exhibit a lot of redundancy. In videos, data is considered redundant if it does not contribute with new information to the representation of the content (Agostini, 2007). Video encoding techniques take advantage of three forms of redundancy: (i) spatial redundancy, (ii) temporal redundancy and (iii) entropic redundancy.

Spatial redundancy is the correlation of pixels within the same picture, i.e. pixels close to each other tend to be similar. It is also called intra-picture redundancy. This redundancy is higher with the increase in video resolution. Spatial redundancy exists both in spatial domain (among pixels) and in the frequency domain (among coefficients after transform).

Temporal redundancy exists in videos due to the high picture capturing rates. The regions within a picture do not considerably change when comparing to the next picture in the video sequence. The exploitation of this kind of redundancy is the one that achieves the highest compression rates in video coding standards.

Unlike previous discussed kinds of redundancy, the entropic redundancy is not directly dependent on video content, but on the way video is represented in encoded form after prediction and transform. Symbols of the video coding representation and the probability of occurrence of each symbol are exploited by lossless compression techniques.

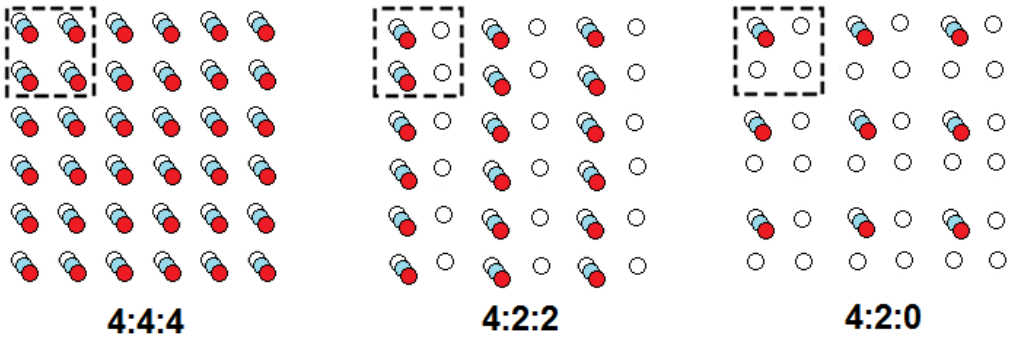
## 2.4 Color Spaces

The most common way of representing one pixel is with three different color components: red (R), green (G) and blue (B). This color space is called RGB. Each combination of intensity of these colors represents one color within the visible spectrum perceived by human eye. This mathematical system is widely used by a variety of multimedia devices. However, the high degree of data correlation between its components imposes a difficulty to video coding.

In order to provide better compression capacity by using a system with a minor degree of correlation among its components, the YCbCr system was conceived. Instead of having 3 components that represent the intensity of three main colors present in each pixel, the YCbCr system has one component to represent the luminance information (Y) and two components to represent the chrominance information: Cb for blue and Cr for red. This model suits better for representing pixel information when it comes to video compression, since it allows encoding algorithms to deal separately with most relevant information, the luminance in this case, which is more sensibly perceived by human eye.

Since the human eye needs less resolution on chrominance information (compared with luminance), video encoding process performs color sub-sampling in order to achieve data compression. This technique consists on the reduction of the sample rate of the chrominance information when compared to the luminance information. There are three basic profiles of color sub-sampling: (i) 4:4:4, (ii) 4:2:2 and (iii) 4:2:0. In the first profile, sub-sampling is not performed. At the second profile, for every 4 samples of luminance, there are 2 respective blue and red chrominance samples each. At the third profile, for every four samples of luminance information, there is only one sample of blue and one sample of red chrominance (see Figure 2.2).

Figure 2.2 - Three different downsampling profiles




Source: Afonso, 2012

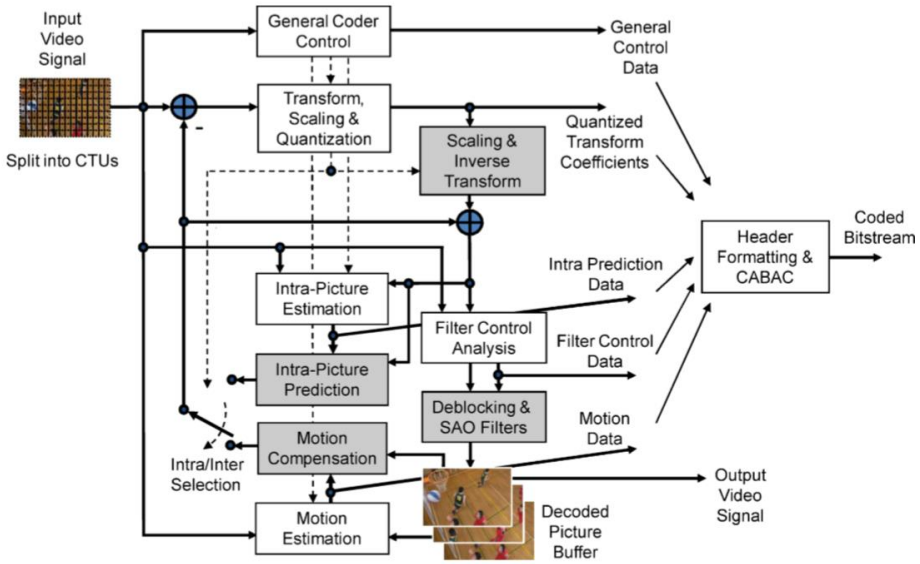
The sub-sampling is the first step in order to achieve high compression rates. It dramatically reduces the required information that has to be processed by the video encoders. The HEVC was primarily designed to be used with the third sub-sampling profile (4:2:0). The support for other profiles is going to be considered in future extensions of the standard.

Another important information is the number of bits used to represent each component of a pixel. The HEVC standard requires components to be represented by 8 bits, thus enabling $2^8$ = 256 possible intensities for each color component. Sample representation with 10 bits are also supported by the standard (ITU-T, 2013).

## 2.5 The HEVC standard

The video coding layer of the HEVC employs essentially the same hybrid approach (inter- and intra-picture prediction and two-dimensional transform coding) used in all video compression standards since H.261. Figure 2.3 depicts the block diagram of a hybrid video encoder, which creates a bit stream that conforms to the HEVC standard.

Figure 2.3 - Block diagram of an HEVC encoder.

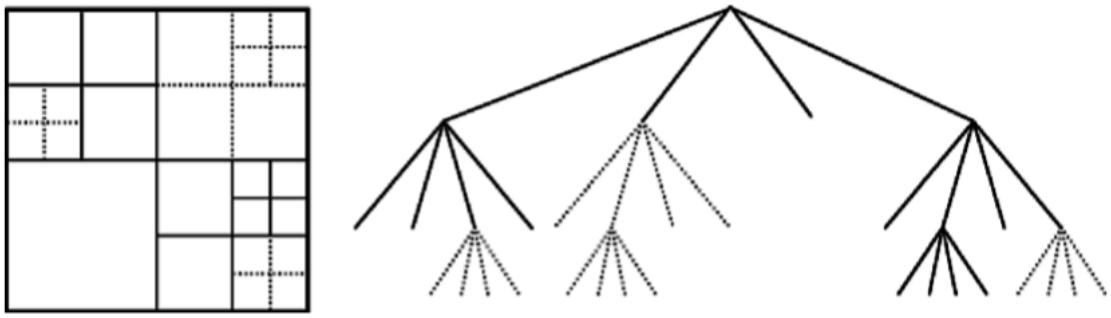



Source: Sullivan, 2012

### 2.5.1 Video coding layer

Like its predecessor, the HEVC also has his codification units based on blocks. In the H.264/AVC standard, the basic structure is a macroblock, which is composed by one luminance block with size of 16x16 pixels and two chrominance blocks (8x8 pixels size), when considered the downsampling 4:2:0 profile. The HEVC standard has generalized the concept of the block to the coding unit (CU). The CU consists of a squared area of 8, 16, 32 or 64 pixels width. The largest coding unit (LCU) is the maximum size that a CU is allowed to have. This parameter is signalized in the bitstream by the encoder. Besides that, a CU can be recursively partitioned into smaller squared-form CUs. This partitioning style generates a quad-tree data structure, called Coding Tree Unit (CTU), as depicted in Figure 2.4. Each CU is composed by one luminance Coding Block (CB), and the corresponding two blue and red CBs. Whether to use inter-frame or intra-frame prediction to encode the CU is decided at this level.

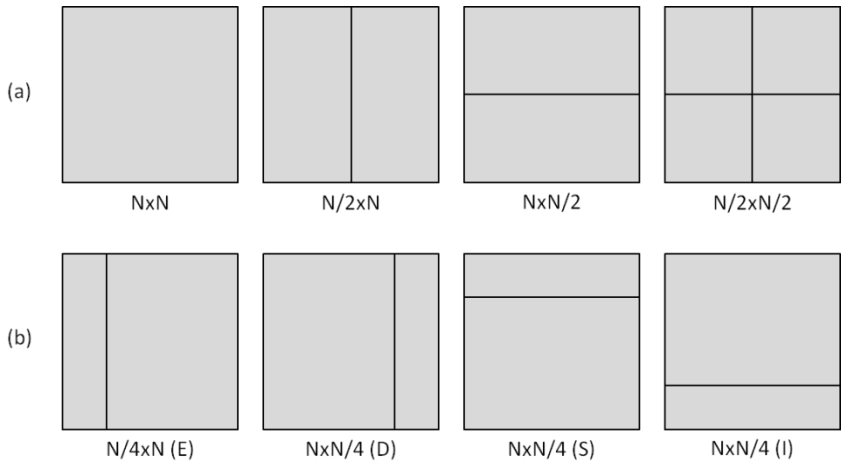
Figure 2.4 - Subdivision of CTUs into CUs and PUs and its corresponding quadtree. Solid lines indicate CU boundaries and dashed lines indicate TU boundaries.



Source: Sullivan, 2012

The basic unit of the prediction processes is called prediction unit (PU). It is obtained thorough the partitioning of a CU according to different possible formats. A PU can assume (i) symmetric formats and (ii) asymmetric formats. Symmetric formats can be square or rectangular. Squared PU formats are used both in inter-prediction and intra-prediction. Rectangular and asymmetric formats can be used only in inter-prediction. Figure 2.5 illustrates the different formats that PUs can assume.

Figure 2.5 - Modes for splitting a CU into PUs.



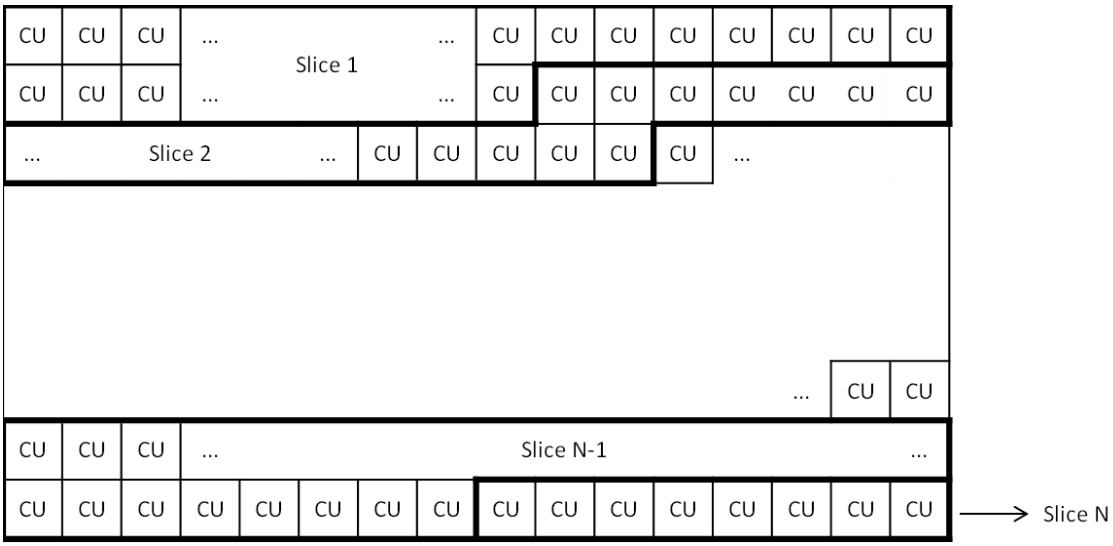

Source: Sullivan, 2012

### 2.5.2 High-level syntax

Slice is a partition of a picture already used in H.264/AVC and is also used in HEVC. Slices contain a number of CUs that are processed in a raster scan order, as illustrated in Figure 2.6. One video picture is allowed to contain one or more slices.

Slices in each picture can be encoded and decoded independently. In H.264/AVC it was previously used to enable parallel processing.

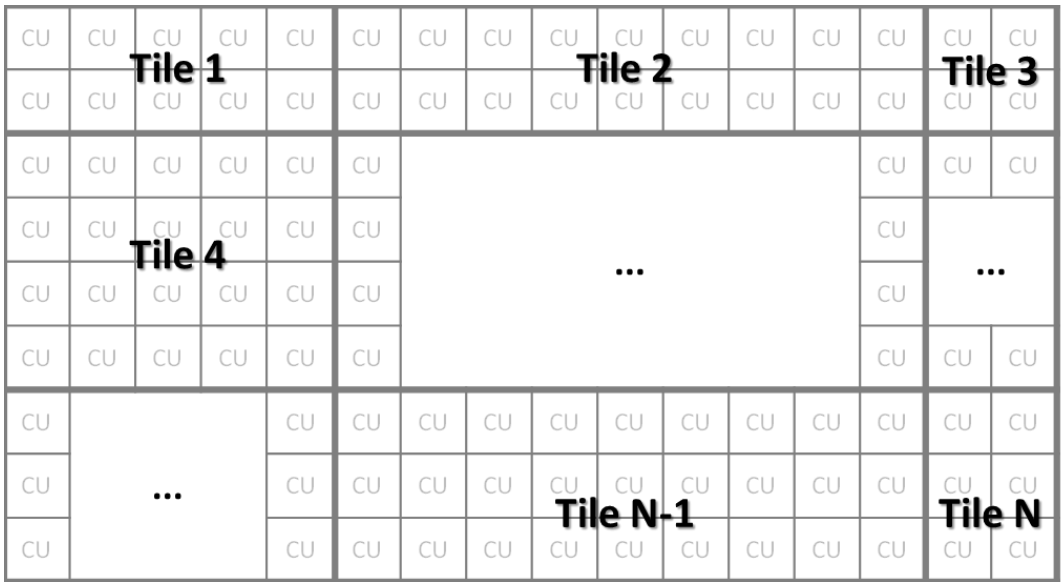
Figure 2.6 - Division of a picture into different slices




Source: Sullivan, 2012

A new data structure named tiles has been introduced in the HEVC standard to specifically target the use of parallel processing techniques. Tiles (as shown in Figure 6) are self-contained rectangular structures composed by a number of CUs that can be independently decoded.

Figure 2.7 - Division of a picture into rectangular tiles

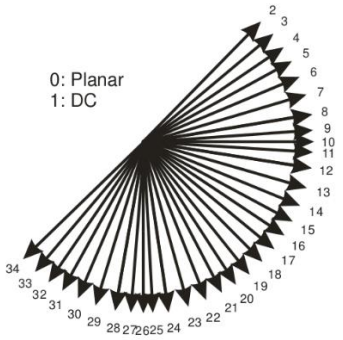



Source: Sullivan, 2012

2.5.3 Intra-picture prediction

As mentioned in section 2.3, the intra-picture prediction aims to exploit the spatial redundancy present at video data in order to achieve high compression rates. The HEVC standard introduces a directional prediction, as in H.264/AVC. It uses information of samples in the borders of previously encoded PUs in the neighborhood of the PU to be encoded. The goal is to reproduce a prediction following the directional borders found in the picture of the video. Unlike the 9 directional orientations in H.264/AVC, HEVC standard introduces 33 different directional prediction orientations, besides planar and DC modes, as depicted in Figure 2.8.

Figure 2.8 - Modes and directional orientation for intra-picture prediction.




Source: Sullivan, 2012

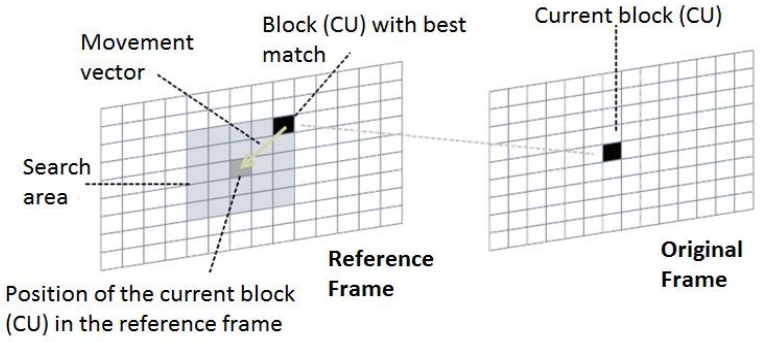
2.5.4 Inter-picture prediction

As mentioned section 2.3, the inter-picture prediction aims to explore temporal redundancy present at video data. In order to do so, it is divided in two different stages: (i) Motion Estimation (ME) and (ii) Motion Compensation (MC). The inter-picture prediction produces the most compression rates of video encoder, at the cost of huge computational complexity.

ME process is detailed in Figure 8. It predicts the current picture (to be encoded) by comparing it to previously encoded and reconstructed pictures, called reference pictures. It is performed for each PU inside a search area formed around the current PU to be encoded. The similar PU in the reference picture is used as prediction. The comparison to find the similar PU (a.k.a matching) is performed with the use of a

distortion metric, e.g. Sum of Absolute Differences (SAD). For the best matching, a Motion Vector (MV) is generated. Only the MV and the residue, i.e. the difference between the current PU and the reference PU, is transmitted in the bitstream.

Figure 2.9 - Details of the ME searching process.




Source: Porto, 2008

MC is performed in the decoder side to generate a reconstructed picture from the MV generated by ME. In order to do so, it accesses the reference pictures stored in the memory and fetches the PU pointed by the MV, thus reconstructing the picture. MC is also performed in the encoder side, to reconstruct a picture exactly equivalent to the one reconstructed in the decoder, to avoid mismatch between encoder and decoder. The current reconstructed picture can be used as a reference picture to encode other pictures.

### 2.5.5 Transforms and Quantization

The transforms module is responsible for transforming residue block to the frequency domain. In order to do so, it applies Direct Cosine Transformation (DCT) on the residue block. There are different sizes of possible residue blocks, called Transform Unit (TU) In HEVC: 4x4, 8x8, 16x16 and 32x32 (Sullivan, 2012).

After that application of the DCT, high frequency information is then discarded by the application of the quantization process. The quantization process is exactly the same as in H.264/AVC. It is parameterized by the Quantization Parameter (QP). QP has 52 levels (from 0 to 51). Each of these levels is mapped to a quantization step (Qstep), and only the first six ones are explicitly defined by the HEVC according to Table 2.1. From the seventh on, it is calculated by taking the double of the current Qstep index minus 6.

Table 2.1 – Different QPs values and their respective Qstep values.

| *QP* | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *...* | *10* | *11* | *12* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $Q_{step}$ | 0.625 | 0.6875 | 0.8125 | 0.875 | 1 | 1.125 | 1.25 | .. | 2 | 2.25 | 2.5 |

Source: Agostini, 2007

2.5.6 Entropy coding

The HEVC standard supports only one entropy coding algorithm: Context Adaptive Binary arithmetic Coding (CABAC). It selects the probability models of each of the syntax elements, based on the context of each one. The core algorithm is the same as in H.264/AVC with reduced contexts and modified coefficient scanning (Sullivan, 2012).

2.5.7 In-Loop Filter

After reconstruct the CU, the pictures are filtered by the In-Loop Filter, to reduce the blocking effects introduced by block partitioning and quantization. Two different filters are defined in HEVC: (i) the Deblocking Filter (DBF) and (ii) Sample Adaptive Offset (SAO). They are applied to reduce blocking artifacts that may appear after a picture is reconstructed, greatly compromising the visual quality of the video.

Unlike DBF, which is applied only on the samples in the edge of a block, SAO utilizes a predefined table in order to perform an adaptive offset. In doing so, SAO reduces the appearance of false borders, contributing to a general improvement of the visual video quality. The in-loop deblocking filter is the main focus of this worked. It will be explained in details in chapter 3, along with a proposed architecture in chapter 4.

## 2.6 Discussion on computational complexity

Through the use of a larger and more complex set of coding tools, when compared to its predecessor H.264/AVC, HEVC achieves higher coding efficiency at the cost of a significant increase in the computational complexity (Bossen, 2012). Therefore, real-time encoding and decoding for HEVC require hardware acceleration.

During the encoding process, 50%-70% of encoding time is spent in Rate-Distortion Optimized Mode Decision for Integer-/Fractional-pel Motion Estimation and intra-picture prediction using a software-based encoder (Diniz, 2013).

In the decoder side, motion compensation takes about half of the decoding time, followed by in-loop filters (DBF and SAO take about 20% of decoding time) and entropy decoding, which also takes around 20% of total decoding time (Bossen, 2012).

Therefore, there is a large set of tools which can be accelerated in order to improve both encoding and decoding processes. Despite Fractional-pel Motion Estimation acceleration (Diniz, 2013)(Afonso, 2012), other tools that consume significant portion in execution time are the in-loop filters (DBF and SAO), motion compensation (MC) and entropy decoding. In the next chapter, there is an entire section dedicated to discuss the importance of the deblocking filter in terms of computational complexity (section 3.3), based on results obtained from the profiling of the HM software reference code.

# 3 DEBLOCKING FILTER IN HEVC

This chapter describes the in-loop deblocking filter of the HEVC standard, which has the goal to reduce the occurrence of visible blocking artifacts at block boundaries that leads to a lower subjective quality of video. To achieve this goal, the deblocking filter performs the detection of blocking artifacts at block boundaries and attenuates them by applying the selected filter.

The block artifacts appear due to the block-based transform coding. As mentioned in the chapter 2, the HEVC standard is based on a hybrid coding scheme using block-based prediction and transform coding. In order to achieve lossy compression on video data, the input video signal is divided into rectangular blocks that are predicted from previously decoded data by either motion-compensated prediction or intra-prediction. The resulting prediction error, which is obtained by the difference between the original and the reconstructed frames, is further used as input to a transform operation (an integer approximation of the discrete cosine transform in HEVC). The transformed coefficients of the blocks are then quantized, resulting in data loss. Because the quantization process is applied in a block-by-block basis, neighboring blocks can quantize coefficients differently, leading to discontinuities at the block boundaries. Those discontinuities (called blocking artifacts) are more visible in flat areas of the image, where there is little detail to mask the loss of data generated by quantization process. Moreover, in a motion-compensated prediction process, predictions from adjacent blocks in a given frame does not necessarily come from adjacent blocks in the previously coded frames, which might lead to discontinuities at the block boundaries of the prediction signal.

Two approaches can be applied in order to reduce the appearance of blocking artifacts: (1) apply post-filtering to the video after it has been completely decoded, which is not standardized by HEVC, leaving to the designer some freedom to determine which algorithm to use, and (2) in-loop filtering, which is performed within the coding process and is part of the HEVC standard in order to avoid drifts between coding and decoding processes.

The deblocking filter in HEVC has been designed to improve the subjective quality while reducing the computational complexity, when compared to deblocking filter of the H.264 standard (Norkin, 2014). Even though, the in-loop deblocking filter is considered a computational hot spot, requiring dedicated hardware architectures, as we demonstrate in section 3.3. Furthermore, the HEVC the deblocking filter is more suitable for parallelization when compared its corresponding in H.264 standard, since it is designed in a way to prevent spatial dependencies across the picture. It facilitates easing the instantiation of multiple hardware accelerators to work simultaneously within the same frame.

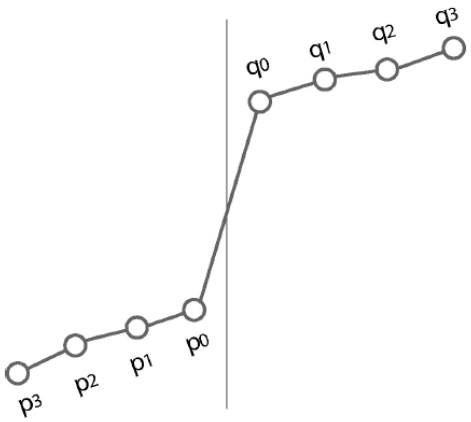
## 3.1 Filtering decisions

### 3.1.1 Block boundaries for Deblocking

The main difficulty while designing a deblocking filter is to decide whether the filtering process will be applied or not for a particular block boundary, as well as to decide the strength of the filtering to be applied. Two opposite cases can happen if the filtering decisions are not well designed. On one hand, excessive filtering may lead to an excessive smoothing of the picture details, implying in loss of data and reduction of the subjective quality. On the other hand, lack of filtering may lead to blocking artifacts that are easily identified by the human visual system, resulting in decrease of video subjective quality. This later problem is precisely the problem that the filter is designed to deal with.

The filtering decision process in HEVC is presented in the next sections. It uses as input the reconstructed samples on both sides of the block boundary to be filtered and some additional parameters to predict if a blocking artifact was created by the encoding process (must be filtered), or it is present in the original image (must not be filtered).

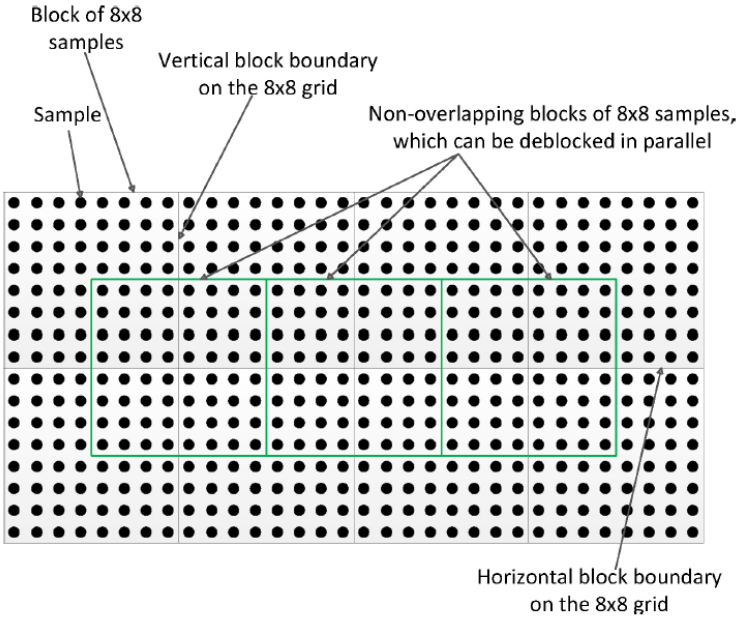
Figure 3.1 – 1-D example of block boundary with blocking artifact.




Source: Norkin (2012)

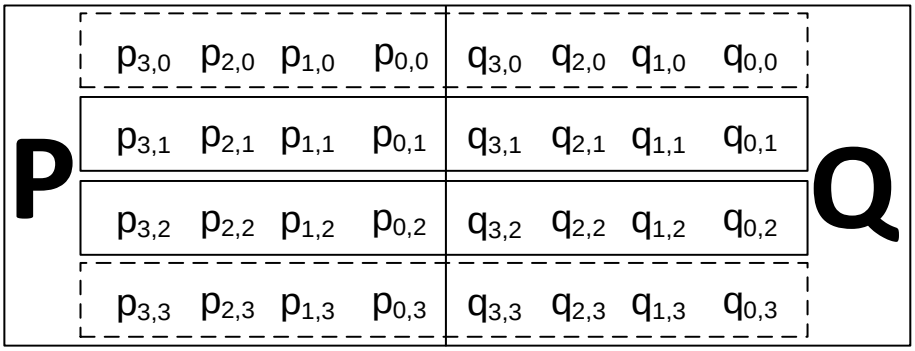
To better explain the block boundaries, we show in Figure 3.2 an example of a picture partitioned into blocks of 8x8 samples. Each block boundary of four-sample length (as shown in Figure 3.3) must be tested against all the conditions and, if necessary, deblocking filter must be applied to them. Only boundaries on the 8x8 grid that are either prediction unit or transform unit boundaries are subjected to deblocking. Figure 3.2 also describes which block boundaries can be filtered independently.

Figure 3.2 – Pictures samples and horizontal and vertical block boundaries on the 8x8 grid, and the non-overlapping blocks of the 8x8 samples, which can be filtered in parallel




Source: Norkin (2012)

Figure 3.3 – Four-sample length block boundary, formed by two adjacent blocks P and Q. Deblocking decisions are based on the samples marked with the dashed line.




Therefore, deblocking filter is applied to the samples on the boundary of the 8x8 block if the following conditions are satisfied: (1) the block unit is a prediction unit or a transform unit boundary; (2) the boundary strength is greater than zero; and (3) variation of the signal on both sides of a block boundary is below a specified threshold.

### 3.1.2 Boundary Strenght (Bs) and Edge level adaptativity

The Boundary strength (Bs), which is a parameter calculated for boundaries that are either prediction units or transform unit boundaries, can assume 3 values: 0, 1 or 2, as described in the table 3.1.

Table 3.1 – Definition of the Bs for the boundaries between two luma blocks.

| Conditions | Bs |
|---|---|
| At least one of the blocks is Intra | 2 |
| At least one of the blocks in non-zero coded residual coefficient and boundary is a transform boundary | 1 |
| Absolute differences between corresponding spatial motion vector components are greater than or equal to 1 in units of integer pixels | 1 |
| Motion-compensated prediction for the two blocks refers to different reference pictures or the numbers of motion vectors is different for two blocks | 1 |
| Otherwise | 0 |

Source: Norkin (2012)

For luma components, only block boundaries with Bs equal to one or two are filtered. This implies that usually no filtering is applied in flat areas of the image. It helps to avoid multiple subsequent filtering in areas of the image where samples are

copied from one part to another with a residual equals to zero, which could lead to an unnecessary over-smoothing of the area. The deblocking filter is first applied to all sets of two 4x4 neighboring blocks that fall into the conditions explained in the next sections and share a vertical boundary. After all vertical block boundaries have been filtered, all blocks belonging to the frame sharing a horizontal boundary are then tested and filtered.

For chroma components, only block boundaries with Bs equal to 2 are subject of filtering operations. Therefore, only blocks boundaries containing at least one intra block are filtered.

### 3.1.3 Local adaptivity and filtering decisions

When Bs is greater than zero, further conditions are checked in order to determine if the filtering operation is going to be applied to the block boundaries or not. Blocking artifacts are characterized by low spatial activity on both sides of the block boundary, whereas there is discontinuity at both sides of the block boundary, as shown in Figure 3.1. The equation 1 shows the condition used to determine this property. This condition is applied for each 4x4 block (see Figure 3.3) which conforms to the conditions mentioned in the section 3.1.1.

$$\left|p_{2,0} - 2p_{2,1} + p_{0,0}\right| + \left|p_{2,3} - 2p_{1,3} + p_{0,3}\right| + \left|q_{2,0} - 2q_{2,1} + q_{0,0}\right| + \left|q_{2,3} - 2q_{1,3} + q_{0,3}\right| > \beta \quad (1)$$

The variable $\beta$ in Equation 1 defines the threshold for the condition and depends on the quantization parameter QP that is used to adapt the quantization step for quantizing the prediction error coefficients.

Equation 1 is responsible for evaluating how much the signal on both sides of the block boundary deviates from a straight line. The decision of HEVC to test samples of only the first and the last lines of the 4x4 block is to reduce the computational complexity. Deblocking filter can be applied to the 4x4 block inside the 8 x 8 grid in two directions: horizontally (as explained before) and vertically. In order to test the vertical conditions, Figure 3.1 must be rotated 90° in the clockwise direction and rows and columns subscripts must be permutated.

For blocks with a corresponding Bs greater than zero and for which condition in Equation 1 holds, deblocking filter is applied. Further decisions must be considered in order to determine the filtering strength to be used to each 4x4 block according to the local signal characteristics. The HEVC standard defines two deblocking filter modes: normal and strength modes. Decisions to apply normal and strength filters are shown in section 3.1.2 and 3.1.3, respectively.

### 3.1.4 Decisions between normal and strength filter

The decision of which mode of the deblocking filter will be applied also depends on samples of the first and on the last row of the block boundary.

Conditions in Equations 2 and 3 assure that there is low spatial activity on both sides of the block boundary for the first and last lines across the 4x4 block boundary, respectively. This decision is similar to condition in Equation 1, but using a lower threshold value.

$$|p_{2,0} - 2p_{2,1} + p_{0,0}| + |q_{2,0} - 2q_{2,1} + q_{0,0}| < \frac{\beta}{8} \quad (2)$$

$$|p_{2,3} - 2p_{2,3} + p_{0,3}| + |q_{2,3} - 2q_{2,3} + q_{0,3}| < \frac{\beta}{8} \quad (3)$$

Conditions in Equations 4 and 5 check if the signal on both sides of the block boundary is flat for the first and the last lines across the 4x4 block boundary, respectively

$$|p_{3,0} - p_{0,0}| + |q_{0,0} - q_{3,0}| > \frac{\beta}{8} \quad (4)$$

$$|p_{3,3} - p_{0,3}| + |q_{0,3} - q_{3,3}| > \frac{\beta}{8} \quad (5)$$

Conditions in Equations 6 and 7 check that the difference in intensities of samples on both sides of the block boundary does not exceed the threshold, which is a multiple of the clipping value $t_c(QP)$ and depends on QP.

$$|p_{0,0} - q_{0,0}| < 2.5t_c \quad (6)$$

$$|p_{0,3} - q_{0,3}| < 2.5t_c \quad (7)$$

The variable $t_c$is used to determine the threshold of the conditions shown in Equations 6 and 7. It depends on QP as defined in Table 3.4. If conditions in Equations

2, 3, 4, 5, 6 and 7 hold, deblocking filter with strength mode is applied to the given P and Q 4x4 blocks. Otherwise, deblocking filter with normal mode is applied.

### 3.1.5 Deblocking decisions in normal filter mode

Normal filtering has two modes of filtering applications, differing in the number of pixels that are modified in the 4x4 block. Whether to apply first mode or second mode of the normal mode of the deblocking filter depends on Equations 8 and 9.

$$\left|p_{2,0} - 2p_{2,1} + p_{0,0}\right| + \left|p_{2,3} - 2p_{2,3} + p_{0,3}\right| < \frac{3}{16}\beta \quad (8)$$

$$\left|q_{2,0} - 2q_{2,1} + q_{0,0}\right| + \left|q_{2,3} - 2q_{2,3} + q_{0,3}\right| < \frac{3}{16}\beta \quad (9)$$

If condition in Equation 8 holds, pixels $p_{0,i}$ and $p_{1,i}$ for $0 \leq i \leq 3$ in the 4x4 block P are modified, i.e. the first two columns of samples that are closer to the block boundary in block P. Otherwise, only the column which are closer to the block boundary, represented by samples $p_{0,i}$ for $0 \leq i \leq 3$, has its samples modified. Similarly for block Q, if condition in Equation 9 is true, pixels $q_{0,i}$ and $q_{1,i}$ for $0 \leq i \leq 3$ are modified. Otherwise, only samples $q_{0,i}$ for $0 \leq i \leq 3$ are modified.

The values of the thresholds used in Equations 8 and 9 are lower than the values of the threshold used in condition Equation 1, but greater than the threshold values used in Equations 4 and 5, assuring that a longer (stronger) filtering operation will be performed in the block boundaries with lower spatial activity on the sides of the boundaries.

For normal filtering operation, filtering may be applied for each row of samples closer of the 4x4 block boundary, based on Equation 10.

$$\left|\delta_{0,i}\right| < 2.5t_c, 0 \leq i \leq 3 \quad (10)$$

The value of the variable $\delta_0$is defined in the next section. It can be considered as the value of the offset that has to be added to each original sample in order to perform the filtering operation, resulting on the filtered sample.

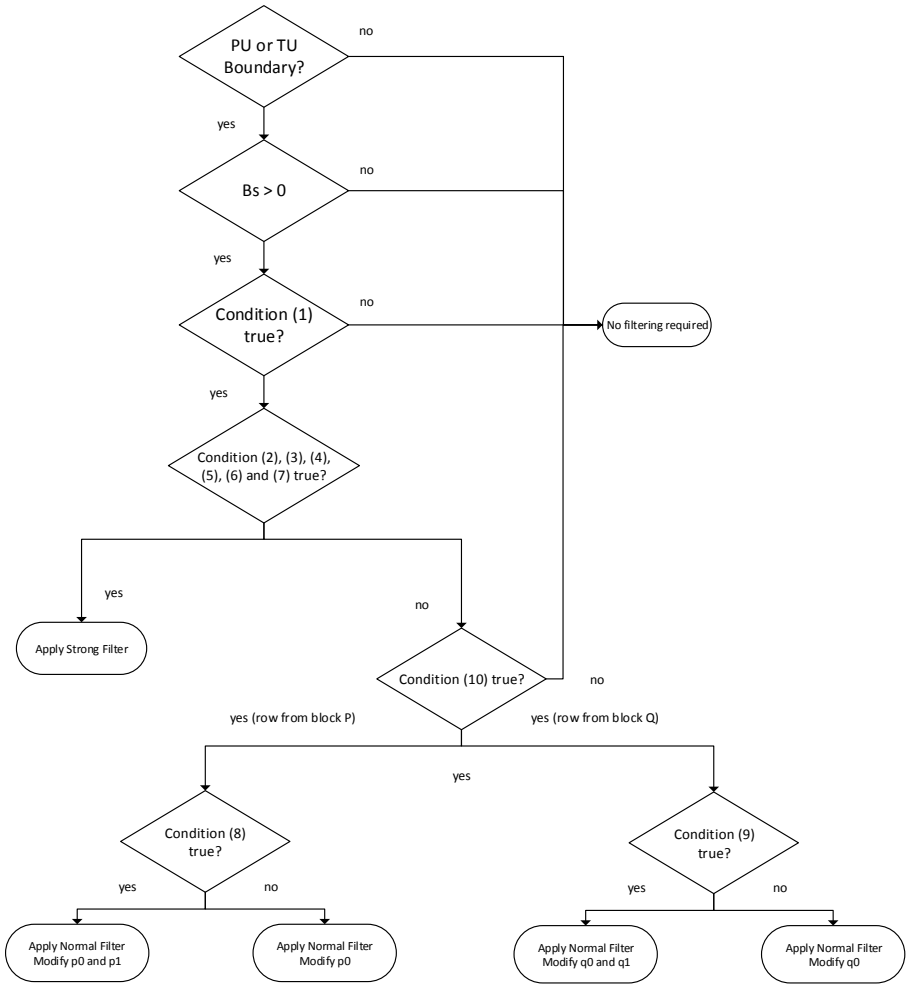
Therefore, when condition in Equation 10 holds for row $i = 2$, for example, normal filtering operation will be applied to row $i = 2$, whereas condition in Equation

10 does not hold for row $i = 3$ (both rows lie on the same 4x4 block), row $i = 3$ will not have its samples value altered by the normal filter.

## 3.2 Filtering operations

We have seen in the previous section the tests that each row from blocks P and Q must be tested against. Figure 3.4 summarizes all the tests. In this section, we will see how to calculate the offset values that will be added to the original sample values in order to generate the final filtered samples.

Figure 3.4 – Decision diagram showing the tests that need to be evaluated and the possible filtering modes.




### 3.2.1 Normal Filter

Whenever normal filtering operation has to be applied to a row of samples of a 4x4 block, an offset value must be calculated, based on the values of the samples on both sides of the row (P and Q blocks). Then, if the required conditions for normal filtering presented in the previous section hold, this offset is added to the original sample value resulting in the value of the filtered sample. The filtered sample values $p'_{0,i}$ and $q'_{0,i}$ are calculated for each row line $i$ across the block boundary, $0 \leq i \leq 3$, as shown in Equations 11 and 12.

$$p'_{0,i} = p_{0,i} + \Delta_{0,i} \quad (11)$$

$$q'_{0,i} = q_{0,i} + \Delta_{0,i} \quad (12)$$

The value $\Delta_{0,i}$ is obtained by performing a clipping operation on $\delta_{0,i}$, which is calculated as described by Equation 13.

$$\delta_{0,i} = 9(q_{0,i} - p_{0,i}) - 3(q_{1,i} - p_{1,i}) + 8 \gg 4 \quad (13)$$

The clipping operation is described in section 3.2.4. Additionally, if condition in Equation 8 holds, $p_{1,i}$ should also have its value modified by the filtering operation shown in Equation 14.

$$p'_{1,i} = p_{1,i} + \Delta_{p1,i} \quad (14)$$

Analogously, if condition in Equation 9 holds, $q_{1,i}$ should be modified as shown in Equation 15.

$$q'_{1,i} = q_{1,i} + \Delta_{q1,i} \quad (15)$$

Like before explained, $\Delta_{p1,i}$ and $\Delta_{q1,i}$ are also calculated by performing a clipping operation to $\delta_{p1,i}$ and $\delta_{q1,i}$ respectively. Those two values are calculated as shown in Equations 16 and 17.

$$\delta_{p1,i} = \Big(\big((p_{2,i} + p_{0,i} + 1) \gg 1\big) - p_{1,i} + \Delta_{0,i}\Big) \gg 1 \quad (16)$$

$$\delta_{q1,i} = \Big(\big((q_{2,i} + q_{0,i} + 1) \gg 1\big) - q_{1,i} - \Delta_{0,i}\Big) \gg 1 \quad (17)$$

### 3.2.2 Strong Filter

The strong filtering mode affects 3 pixels of both sides of the P and Q 4x4 blocks: $p_{0,i}$, $p_{1,i}$ and $p_{2,i}$ for block P and $q_{0,i}$, $q_{1,i}$ and $q_{2,i}$ for block Q. For each row of block P, the samples are modified as shown in Equations 18, 19 and 20.

$$p'_{0,i} = p_{0,i} + \Delta_{0s,i} \quad (18)$$

$$p'_{1,i} = p_{2,i} + \Delta_{2s,i} \quad (19)$$

$$p'_{2,i} = p_{2,i} + \Delta_{2s,i} \quad (20)$$

Analogously to block P, each line of block Q is modified as shown in Equations 21, 22, and 23.

$$q'_{0,i} = q_{0,i} + \Delta_{0s,i} \quad (21)$$

$$q'_{1,i} = q_{2,i} + \Delta_{2s,i} \quad (22)$$

$$q'_{2,i} = q_{2,i} + \Delta_{2s,i} \quad (23)$$

The values $\Delta_{0s,i}$, $\Delta_{1s,i}$, and $\Delta_{2s,i}$ are obtained by clipping the values of $\delta_{0s,i}$, $\delta_{1s,i}$ and $\delta_{2s,i}$ respectively. Those values can be calculated as described by the Equations 24, 25 and 26.

$$\delta_{0s,i} = \left(p_{2,i} + 2p_{1,i} - 6p_{0,i} + 2q_{0,i} + q_{1,i} + 4\right) \gg 3 \quad (24)$$

$$\delta_{1s,i} = \left(p_{2,i} - 3p_{1,i} + p_{0,i} + q_{0,i} + 2\right) \gg 2 \quad (25)$$

$$\delta_{2s,i} = \left(2p_{3,i} - 5p_{2,i} + p_{1,i} + p_{0,i} + q_{0,i} + 4\right) \gg 3 \quad (26)$$

### 3.2.3 Chroma Filter

As mentioned in section 3.1.1, chroma deblocking filter is only performed when Bs is equal to 2, and no further decisions need to be evaluated. Only the samples $p_{0,i}$ and $q_{0,i}$ closer to the block boundary are modified, as described by the Equations 27 and 28.

$$p'_{0,i} = p_{0,i} + \Delta_{c,i} \quad (27)$$

$$q'_{0,i} = q_{0,i} + \Delta_{c,i} \quad (28)$$

The offset value $\Delta_{c,i}$ is obtained by clipping the value of $\delta_{c,i}$, which is calculated as shown in Equation 29.

$$\delta_{c,i} = \left(\left(\left(p_{0,i} - q_{0,i}\right) \ll 2\right) + p_{1,i} - q_{1,i} + 4\right) \gg 3 \quad (29)$$

### 3.2.4 Clipping operation

To prevent excessive blurriness, the result of the offsets that are calculated based on the samples of a given row of a 4x4 block are subject to a QP-dependent clipping operation, before the addition to the original sample value in order to compose the filtered sample. The clipping operation is described by Equation 30.

$$\Delta = min(\max(-c, \delta), c) \quad (30)$$

The parameter $c$ of the clipping operation is adapted according to the type of the filter. The type of prediction used to reconstruct the block (inter or intra-prediction) also affects the clipping operation, as described in Table 3.2 and Table 3.3.

Table 3.2 – Definition of the value $c$ used as threshold for the clipping operation

| *Conditions* | $c$ |
|---|---|
| Normal filtering | $t_c(n)$ for $p_{0,i}/q_{0,i}$ and $0.5t_c(n)$ for $p_{1,i}/q_{1,i}$ |
| Strong filtering | $2t_c(n)$ |

Source: Norkin (2012)

Table 3.3 – Definition of the value $n$ used as input to determine the value of $t_c$

| *Conditions* | $n$ |
|---|---|
| Both blocks P and Q are inter-predicted | $t_c(QP)$ |
| One of the blocks P or Q are intra-predicted | $t_c(QP+2)$ |

Source: Norkin (2012)

The relation between the values of QP and $t_c$ and $\beta$ are defined in Table 3.4.

Table 3.4 – Derivation of threshold variables $t_c$ and $\beta$ for each QP

| *QP* | 0 | ... | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\beta$ | 0 | … | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 20 | 22 | 24 | 26 | 28 |
| $t_c$ | 0 | .. | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 |
| *QP* | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 |
| $\beta$ | 30 | 32 | 34 | 36 | 38 | 40 | 42 | 44 | 46 | 48 | 50 | 52 | 54 | 56 | 58 | 60 | 62 | 64 | - | - |
| $t_c$ | 3 | 4 | 4 | 4 | 5 | 5 | 6 | 6 | 7 | 8 | 9 | 40 | 11 | 13 | 14 | 16 | 18 | 20 | 22 | 24 |

Source: HEVC recommendation (ITU-T, 2013)

## 3.3 Profiling on HEVC reference software

In order to verify the contribution of deblocking filter in to the total execution time of HEVC encoder/decoder, we have profiled HEVC Test Model (HM) 10.0 encoder and decoder software using GNU gprof (Fenlason, 2000).

A total of 11 video sequences with different resolutions from the Common Test Conditions document (Bossen, 2012) have been analyzed. The sequences are shown in Table 3.5. We choose 5 video sequences from class B (1920x1080 pixels), 3 video sequences from class C (832x480 pixels) and 3 video sequences from class D (416x480 pixels). Each video sequence was encoded and then decoded using 4 different QP values (22, 27, 32 and 37). We have analyzed more video sequences of higher resolutions because they are used more often in advanced video applications.

Table 3.4 – Video sequences used for profiling

| *Class* | *Sequence* | *Resolution* |
|---|---|---|
| B | Kimono | 1920x1080 |
| | ParkScene | 1920x1080 |
| | Cactus | 1920x1080 |
| | BQTerrace | 1920x1080 |
| | BasketballDrive | 1920x1080 |
| C | PartyScene | 832x480 |
| | BasketballDrill | 832x480 |
| | BQMall | 832x480 |
| D | BlowingBubbles | 416x214 |
| | BQSquare | 416x214 |
| | BasketballPass | 416x214 |

### 3.3.1 Conclusions regarding the profiling of HEVC decoder

The results of the profiling of the HM decoder code using the class B, C and D video sequences are shown in Figures 3.4, 3.5 and 3.6 respectively.

Figure 3.4 – Results for the execution time of code related to deblocking filter operations of class B videos in the decoder

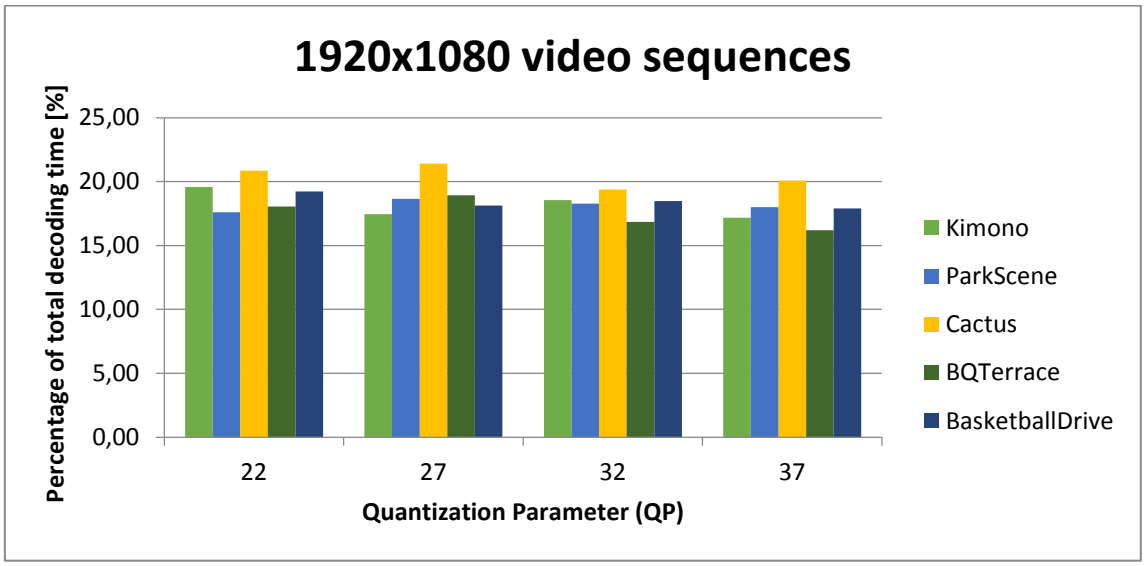



Figure 3.5 – Results for the execution time of code related to deblocking filter operations of class C videos in the decoder

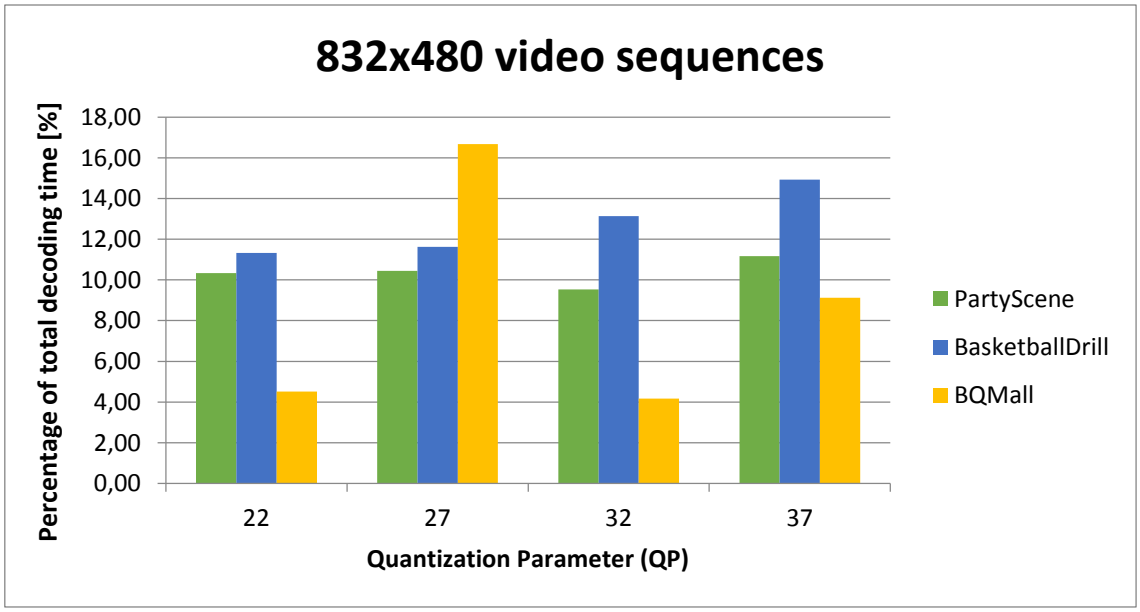
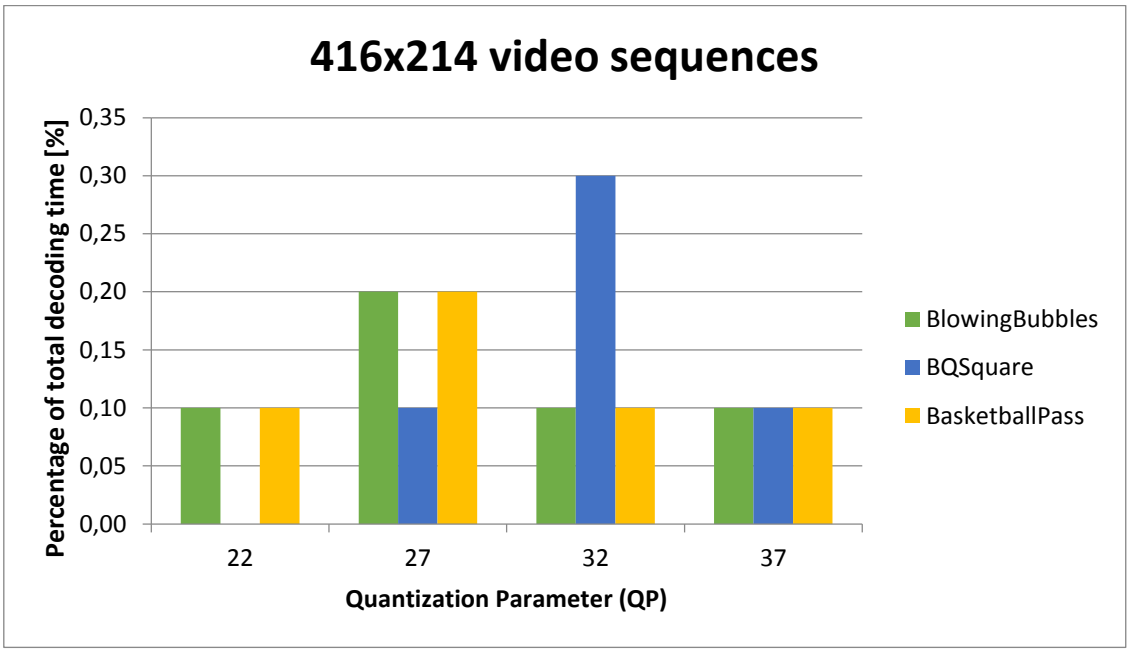


Figure 3.5 – Results for the execution time of code related to deblocking filter operations of class D videos in the decoder




In a general way, higher resolutions result in more deblocking filter operations, because it implies that more boundaries need to be tested and, if necessary, filtered. There is also a correlation between the QP and the number of filtering operations. Higher QPs often result in more filtering operations, because the loss of information during the quantization process is higher. Hence, it results in more probable signal discontinuities across block boundaries. This is the reason why the threshold values $t_c$ and $\beta$ depend on QP.

The software operations regarding the deblocking filter in the HM reference code can reach up to 20% of execution time while decoding a video sequence for high resolutions (1920x1080). It is expected that it corresponds to similar rates in video sequences with higher resolutions, which are trending do be adopted by new consumer devices. Therefore, in order to improve throughput and reduce power consumption, dedicated hardware architectures for the HEVC deblocking filter must be considered while implementing a decoder that is compliant to the HEVC standard.

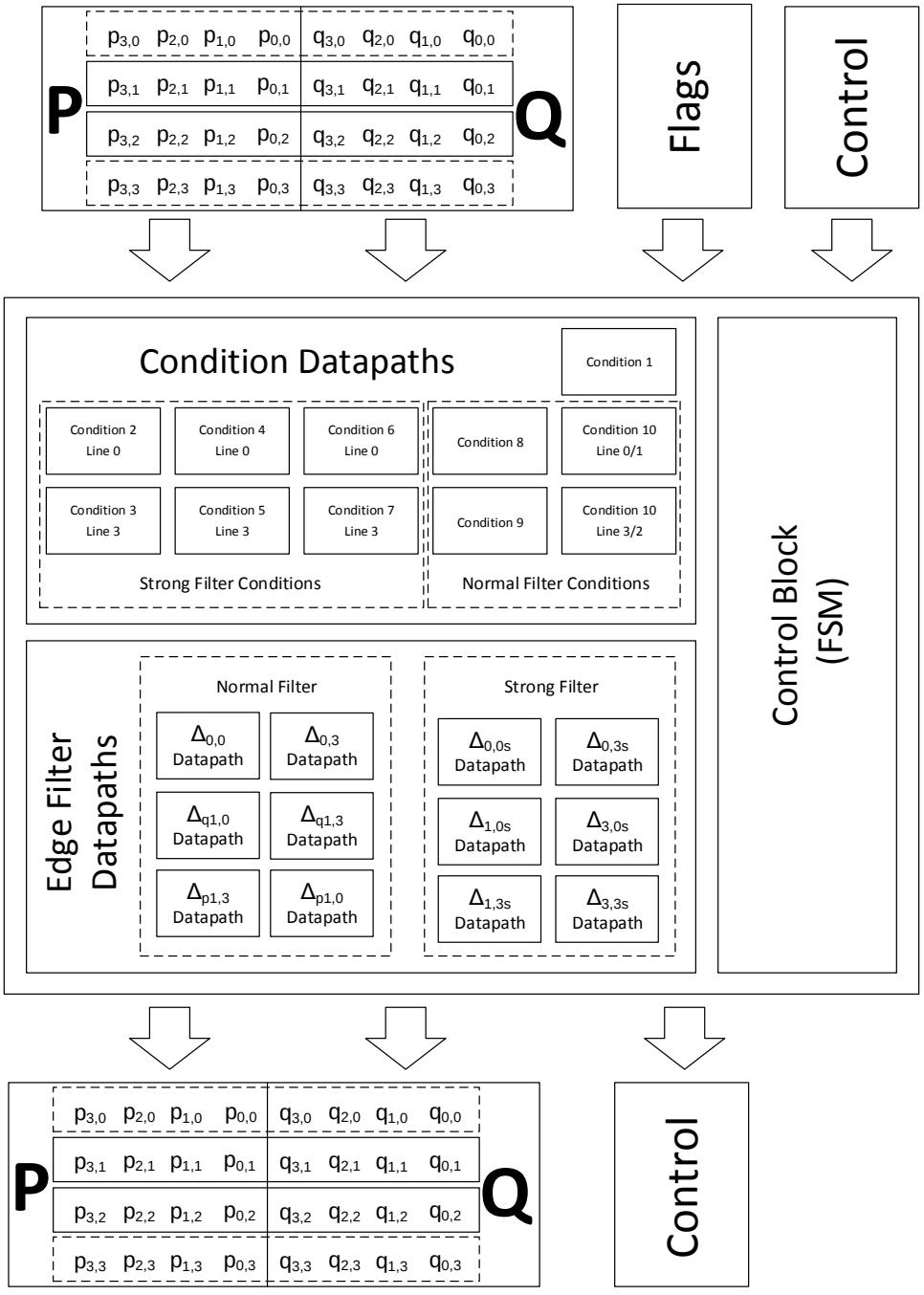
### 3.3.2 Conclusions regarding the profiling of HEVC encoder

The HM encoder reference software was also profiled using as inputs the video sequences from Table 3.4. The profiling results showed that deblocking filter operations can reach up to 4% of encoding time for some video sequences, which is a value far less significant if compared to the time spent in the decoding process. However, a HEVC compliant encoder also performs deblocking filter operations in the encoding loop, in order to remove blocking artifacts from reconstructed reference frames. Furthermore, it must be considered that the HEVC encoder has a set of more complex tools if compared to the decoder, while the entire encoding process demands far more computation time than the decoding process. Therefore, a dedicated hardware architecture for the deblocking filter can also be considered while designing a HEVC compliant encoder, in order to reduce the total time of the encoding process, achieving real-time requirements for encoding high-resolution video sequences.

# 4 DEBLOCKING FILTER HARDWARE ARCHITECHTURE

This chapter describes the proposed in-loop deblocking filter hardware architecture for the HEVC standard. Figure 4.1 illustrates the top-level architecture, along with its inputs and outputs.

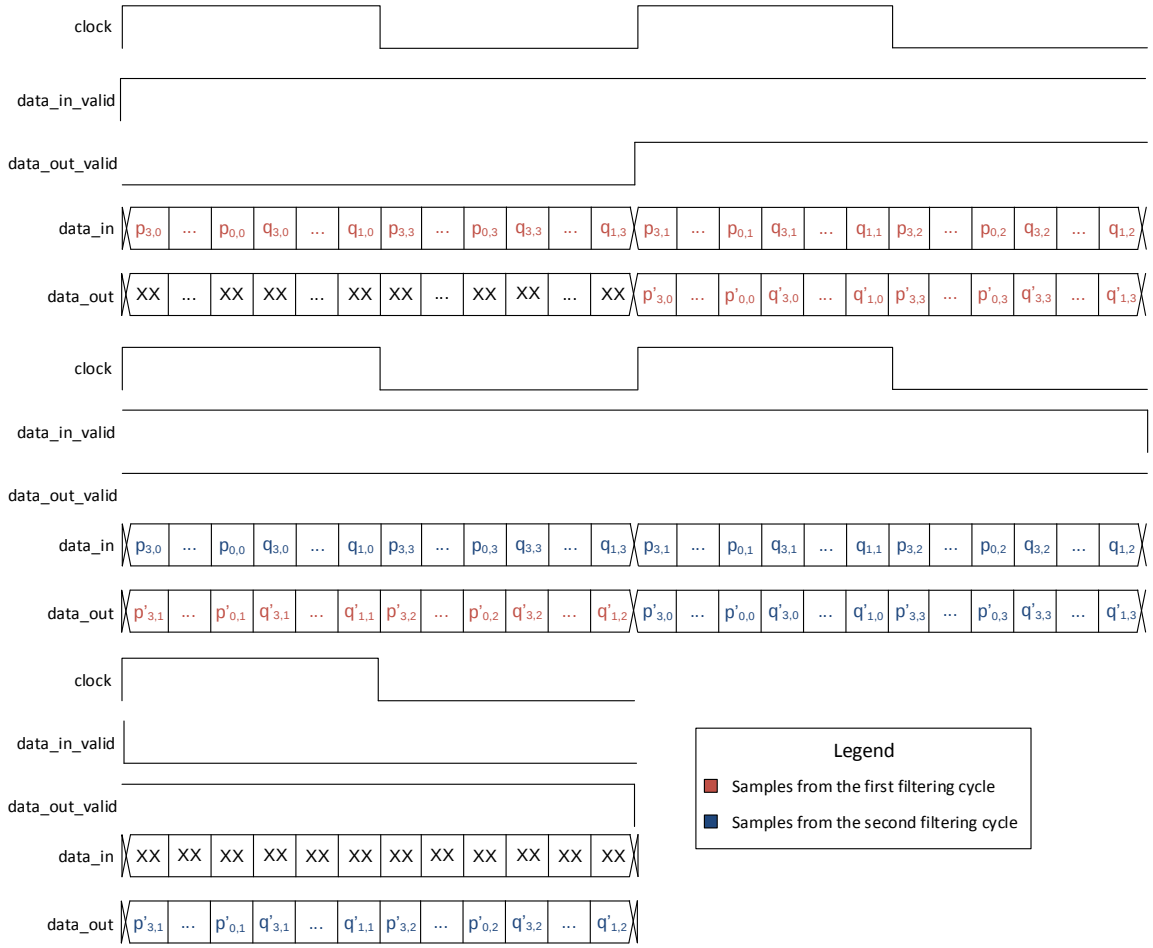
Figure 4.1 – Top-level diagram of the proposed HEVC Deblocking Filter hardware architecture



The proposed architecture receives as input 8-bit samples from 2 neighboring 4x4 blocks, each one belonging to a different adjacent 8x8 block on the grid (see Figure 3.2) that needs to be tested in order to decide: (1) if filtering is required and (2) the strength of the filtering to be applied if this is the case. If filtering is required, the filtered samples are provided in the output, along with some control signals to specify that valid output data is available.

The organization of the data and the size of the channel that will handle the I/O communication of our architecture are explained in Figure 4.2. Along with the samples, the architecture requires some flags that are needed for filtering decision (mainly threshold values used in the equations of the conditions described in chapter 3), as well as some control signals to establish a handshake between the master (e.g. a CPU that runs the HEVC encoder/decoder application) and our architecture (which plays the slave role).

Figure 4.2 – Wave-form diagram representing the complete filtering cycle (2 clock cycles) of two 4x4 sample blocks that need to be filtered

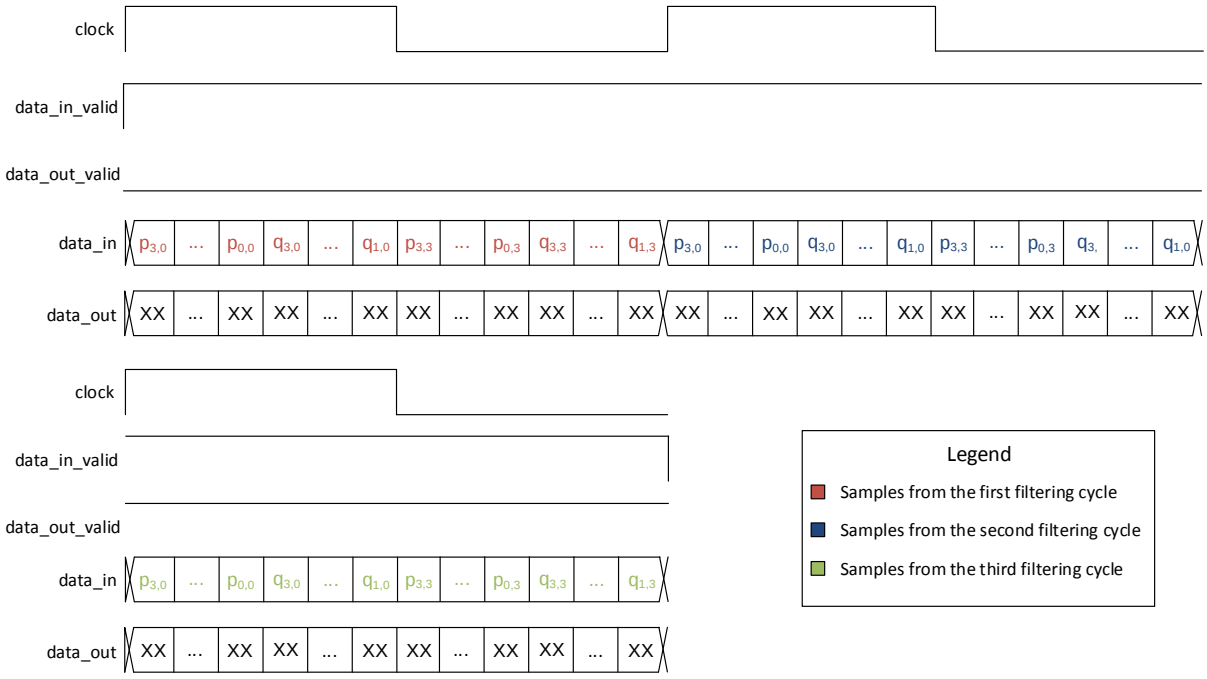



The input channel through which the input data is transmitted is 256-bit wide. Therefore, 32 samples can be transmitted within one clock cycle, with the most significant byte containing the sample belonging to the first row of block P which lies furthest from the block boundary. The second most significant byte contains the second sample from the first row which lies furthest from the P block boundary and so on, following a raster scan order. The least significant byte contains the sample from the last line of block Q, which lies furthest from the block boundary. The organization of the data inside the I/O channel is detailed in Figure 4.2.

Since each row of the each 4x4 block contains 4 samples, and the first and last row lines of both 4x4 blocks are required in order to perform the filtering testing, we can transmit both row lines of both 4x4 adjacent blocks at the first clock cycle and have

all the required data to test the filtering decision (see the dashed lines of Figure 3.3).

Figure 4.3 – Wave-form diagram representing the complete filtering cycle testing (1 clock cycle) of three 4x4 sample blocks that do not need to be filtered




After the first load cycle, the conditions datapaths will be able to decide whether filtering is required or not. If it is not the case, there is no necessity of transmitting the second and third row lines of the blocks P and Q, as they are only required for generating their respective filtered samples (see Figure 4.3). Therefore, at the next clock cycle, the architecture is ready to receive the first and last rows of further 4x4 blocks in order to compute the filtering testing conditions and the filtered samples.

## 4.1 Hardware datapaths

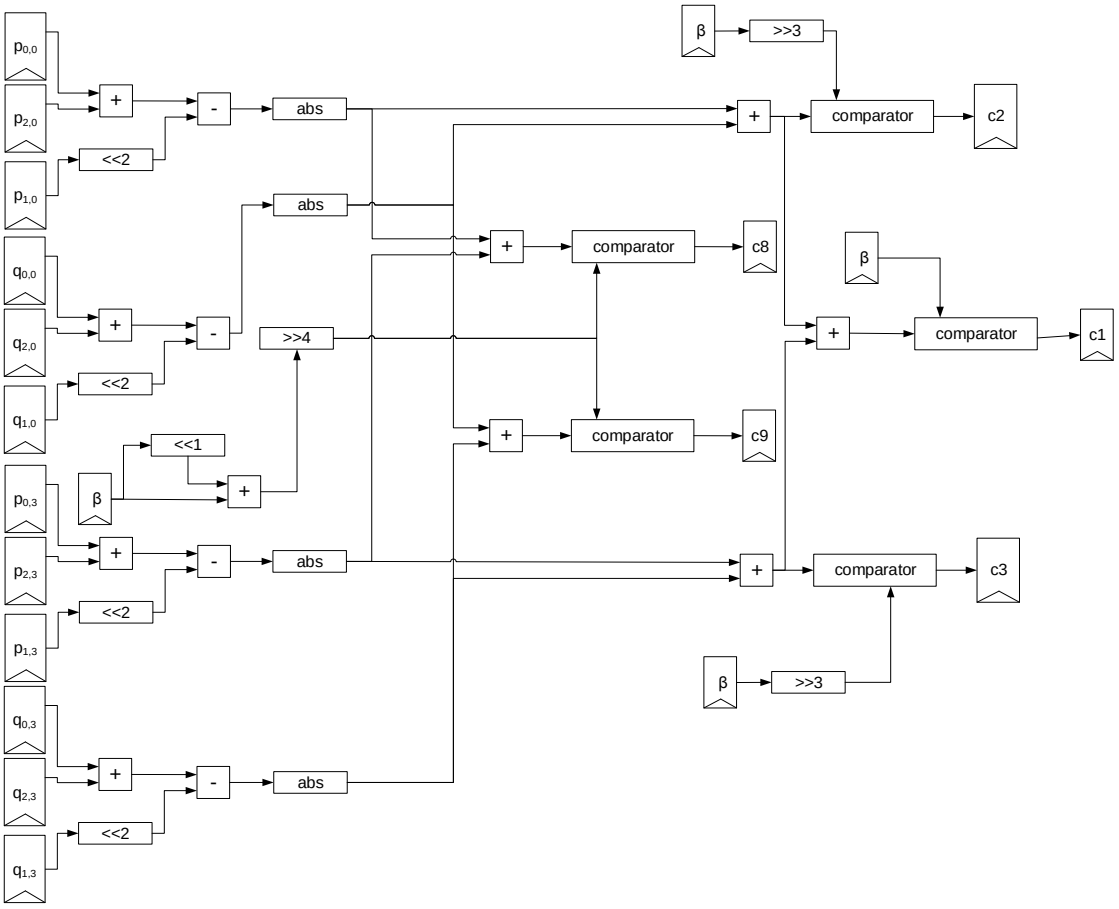
### 4.1.1 Decision datapaths

In order to determine the necessity of filtering of two given 4x4 blocks, all condition equations of the previous chapter must be computed. The architecture of the hardware datapaths to do this task are explained in this sub-section.

4.1.1.1 Datapath for conditions 1, 2, 3, 8 and 9

The datapath that implements conditions 1, 2, 3, 8 and 9 is shown in Figure 4.4. Since these five conditions have similar equations and same inputs (as described in chapter 3), we developed one single datapath that implements all of them. The condition equations require some multiplication by constants, which was implemented by shifting and adding the operand to its shifted value.

Figure 4.4 – Block diagram of the datapath responsible for calculation conditions 1, 2, 3, 8 and 9.

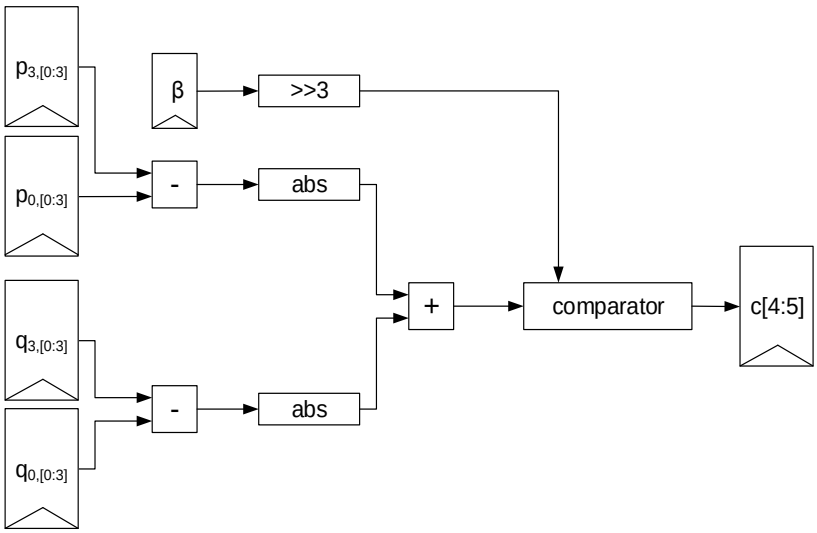



4.1.1.2 Datapath for conditions 4 and 5

This datapath implements conditions 4 and 5 and is detailed in Figure 4.5. There are two instances of this datapath in the architecture. One instance is responsible for the calculation of the first row of two adjacent 4x4 blocks, i. e. for condition 4. The other

instance is responsible for the calculation of the last row, resulting in condition 5. Both conditions are calculated in parallel, in the first clock cycle of a filtering cycle, since all the required samples in order to perform this task have already been transmitted to the architecture.

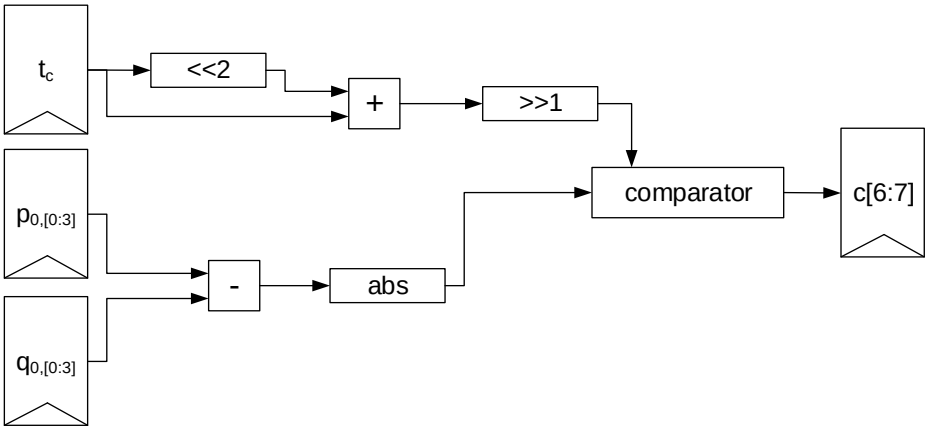
Figure 4.5 – Block diagram of the datapath responsible for calculation conditions 4 and 5.




4.1.1.3 Datapath for conditions 6 and 7

This datapath is responsible for implementing conditions 6 and 7. Like the previous datapath, this module is instantiated twice in the architecture. The first instance is responsible for the calculation of the first row of two adjacent 4x4 blocks, resulting in condition 6. The other instance is responsible for the calculation of the last row, resulting in condition 7. As the previous datapath, both conditions are computed in parallel by each instance of this datapath.

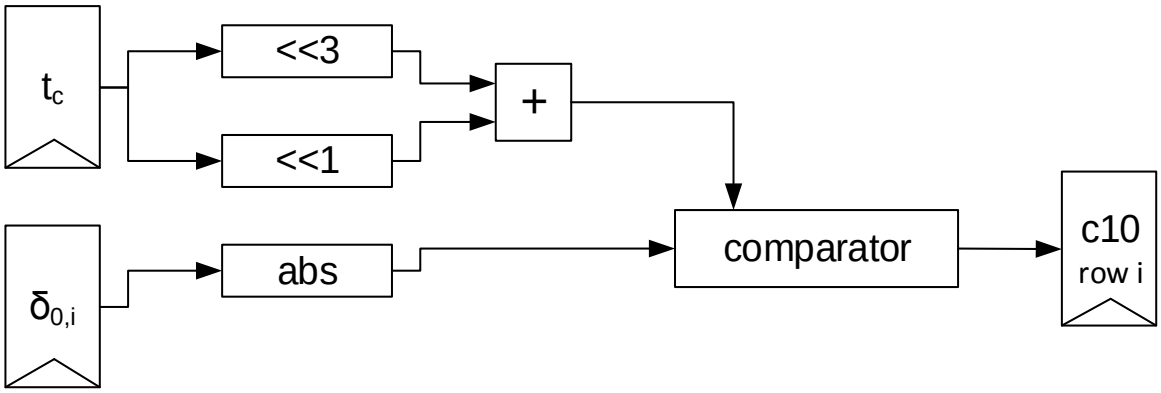
Figure 4.6 – Block diagram of the datapath responsible for calculation conditions 6 and 7.




4.1.1.4 Datapath for Condition 10

This datapath implements condition 10. Like the previous two datapaths, two instances of this datapath are present in the architecture. However, unlike all previous conditions, all four rows have to be tested against condition 10 if normal filtered is required, i.e. conditions 2, 3, 4, 5, 6 and 7 are false. Therefore, at the first clock cycle of a filtering cycle, one instance of the datapath is responsible for checking the first row of samples of blocks P and Q, while the other instance is responsible for the last row of samples. At the second clock cycle, however, the first instance checks condition 10 using as input the second row of samples and the other datapath the third one. This assures that all 4 rows of samples are tested against condition 10.

Figure 4.7 – Block diagram of the datapath responsible for calculation condition 10.



### 4.1.2 Edge filter datapaths

In order to calculate the filtered samples, an offset value must be generated according to the sample values belonging to both adjacent 4x4 blocks. After the offset value is calculated, it must be added to the original sample value in order to generate the final filtered sample.

#### 4.1.2.1 Normal Filter datapaths

There are two instances of each of the datapaths shown in Figure 4.8, 4.9 and 4.10 in the architecture. Each instance is responsible for calculation of one row of the 4x4 block at a given clock cycle. The second subscripts have been omitted in the following diagrams, for they represent the row-line, which can be 0 or 3, at the first clock cycle of a filtering cycle, or 1 or 2, at the second clock cycle of a filtering cycle (if filtering is required).

Figure 4.8 – Block diagram of the datapath responsible for calculation $\Delta_{0,i}$

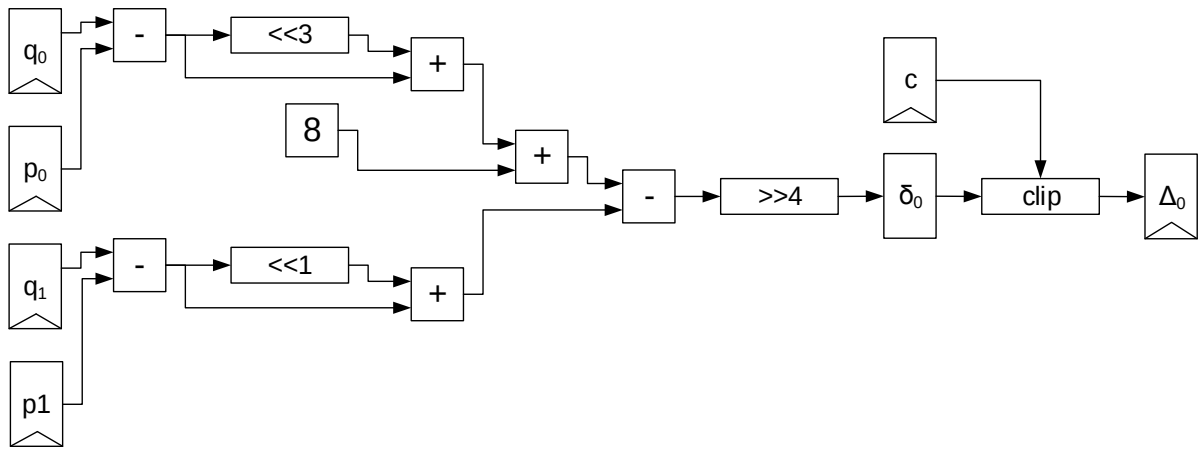


Figure 4.9 – Block diagram of the datapath responsible for calculation $\Delta_{p1,i}$

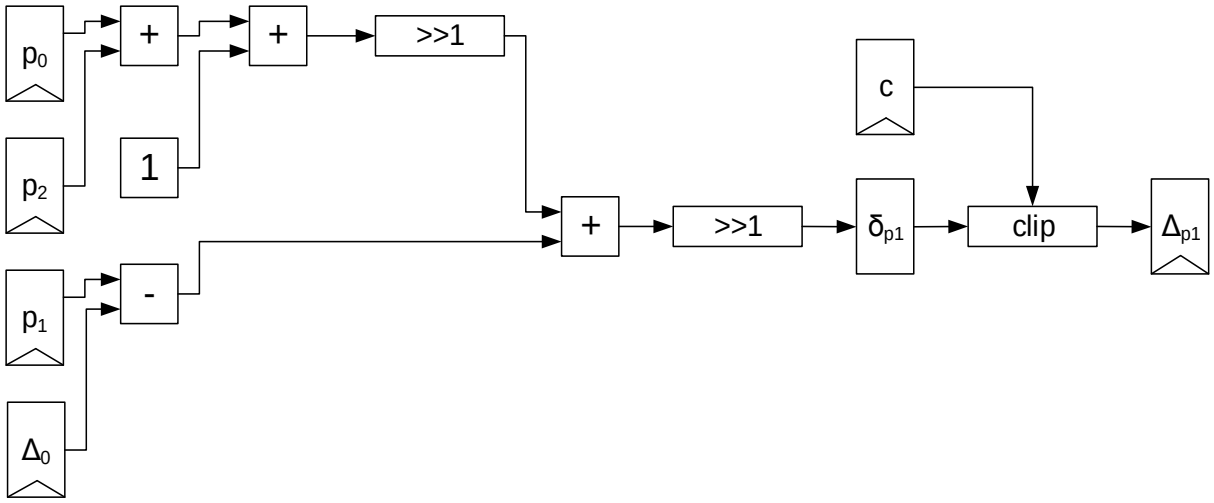
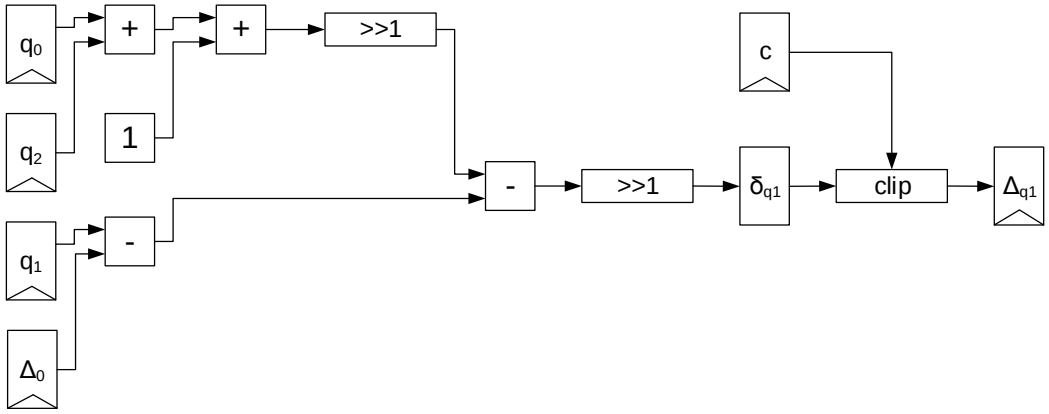



Figure 4.10 – Block diagram of the datapath responsible for calculation $\Delta_{q1,i}$




The datapaths from Figure 4.9 and Figure 4.10 are responsible for generating the data that will be added if the strongest mode of normal filter mode is required for a given row.

4.1.2.2 Strong Filter datapaths

The datapaths illustrated in Figures 4.11, 4.12 and 4.13 are responsible for generating the offset values when strong filter mode is required. Each datapath was instantiated twice in the architecture (each instantiation is responsible for one of the incoming row of the input blocks at each clock cycle). The second subscripts have been omitted in the following diagrams, for they represent the row-line, which can be 0 or 3,

at the first clock cycle of a filtering cycle, or 1 or 2, at the second clock cycle of a filtering cycle (if filtering is required).

Figure 4.11 – Block diagram of the datapath responsible for calculation $\Delta_{0s,i}$

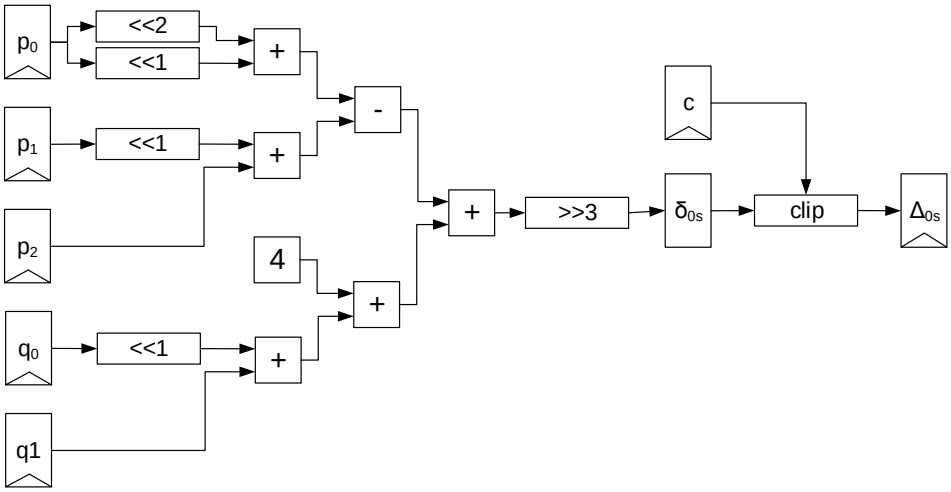



Figure 4.12 – Block diagram of the datapath responsible for calculation $\Delta_{1s,i}$

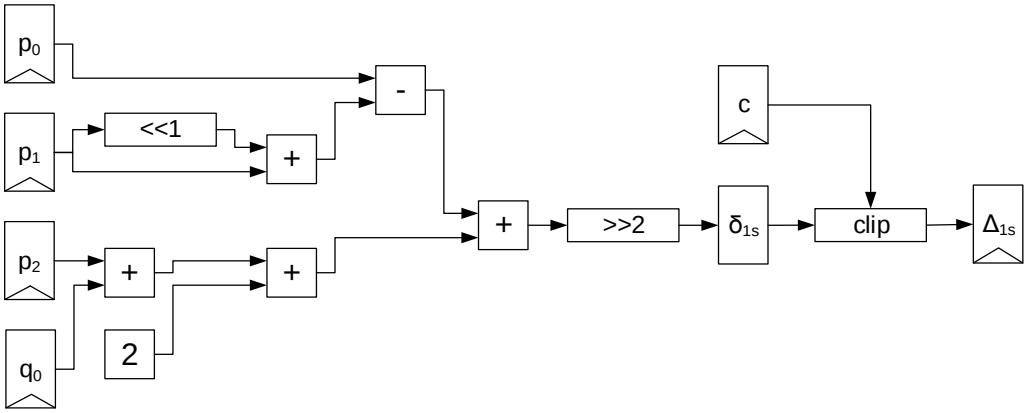
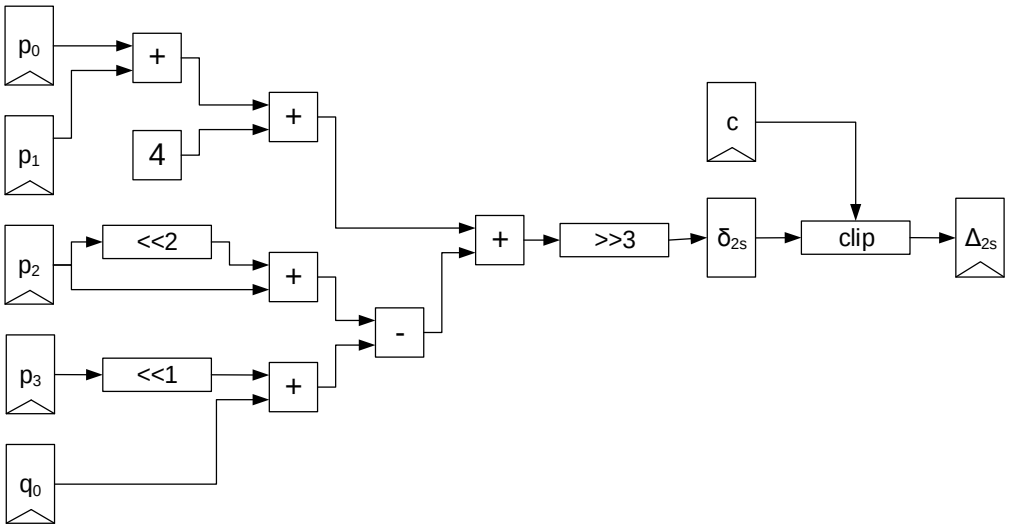


Figure 4.13 – Block diagram of the datapath responsible for calculation $\Delta_{2s,i}$




## 4.3 Clipping

To perform the clipping operation, the architecture includes a small memory block that receives as input a QP value and results as output the corresponding $\beta$ and $t_c$ values, as specified in Table 3.4. The clipping threshold values must conform to the ones presented in Table 3.2, as it may assume different values depending on the decision between the normal filter or the strong filter. Moreover, the clipping value also depends of the prediction type of blocks P and Q, as specified in Table 3.3.

## 4.4 Filtered sample generation

After the conditions have been evaluated and the necessity of filtering and its corresponding mode is known for a given row, we must add the values generated by the filter datapaths to their corresponding original values. In order to decide that, we have a simple multiplexer for each output sample, as follows.

Tables 4.1 and 4.2 defines the values of the filtered samples, which lie closer to the boundary between blocks P and Q and are always affected when filtering is required. In the tables below, ‘1’ stands for the logical concept of ‘true’, ‘0’ for ‘false’ and ‘x’ for ‘don’t care’.

Table 4.1 – Definition of the filtered $p'_{0,i}$ sample final value, based on the evaluated conditions

| $p'_{0,i}$ *value* | *Condition 1* | *Conditions 2, 3, 4, 5, 6 and 7* | *Condition 10* | *Description* |
|---|---|---|---|---|
| $p_{0,i}$ | 0 | x | x | Filter off |
| $p_{0,i} + \Delta_{0s,i}$ | 1 | 1 | x | Strong filter |
| $p_{0,i}$ | 1 | 0 | 0 | Normal filter off |
| $p_{0,i} + \Delta_{0,i}$ | 1 | 0 | 1 | Normal filter |

Table 4.2 – Definition of the filtered $q'_{0,0}$ sample final value, based on the evaluated conditions

| $q'_{0,i}$ *value* | *Condition 1* | *Conditions 2, 3, 4, 5, 6 and 7* | *Condition 10* | *Description* |
|---|---|---|---|---|
| $q_{0,i}$ | 0 | x | x | Filter off |
| $p_{0,i} + \Delta_{0s,i}$ | 1 | 1 | x | Strong filter |
| $q_{0,i}$ | 1 | 0 | 0 | Normal filter off |
| $q_{0,i} + \Delta_{0,i}$ | 1 | 0 | 1 | Normal filter |

Tables 4.3 and 4.4 define the values of the filtered samples, which lie at the second closest columns from the block boundary of blocks P and Q. These samples are modified whenever strong filter or normal filter in longer mode is required (condition 8 for block P and condition 9 for block Q is true).

Table 4.3 – Definition of the filtered $p'_{1,i}$ sample final value, based on the evaluated conditions

| $p'_{1,i}$ *value* | *Condition 1* | *Conditions 2, 3, 4, 5, 6 and 7* | *Condition 8* | *Condition 10* | *Description* |
|---|---|---|---|---|---|
| $p_{1,i}$ | 0 | x | x | x | Filter off |
| $p'_{1,i} + \Delta_{1s,i}$ | 1 | 1 | x | x | Strong filter |
| $p_{1,i}$ | 1 | 0 | 0 | x | Long normal filter off |
| $p'_{1,i} + \Delta_{p1,i}$ | 1 | 0 | 1 | 1 | Long normal filter on |
| $p_{1,i}$ | 1 | 0 | 1 | 0 | Normal filter off |

Table 4.4 – Definition of the filtered $q'_{1,i}$ sample final value, based on the evaluated conditions

| $q'_{1,i}$ *value* | *Condition 1* | *Conditions 2, 3, 4, 5, 6 and 7* | *Condition 9* | *Condition 10* | *Description* |
|---|---|---|---|---|---|
| $q_{1,i}$ | 0 | x | x | x | Filter off |
| $q'_{1,i} + \Delta_{1s,i}$ | 1 | 1 | x | x | Strong filter |

| | | | | | |
|---|---|---|---|---|---|
| $q_{1,i}$ | 1 | 0 | 0 | x | Long normal filter off |
| $q'_{1,i} + \Delta_{p1,i}$ | 1 | 0 | 1 | 1 | Long normal filter on |
| $q_{1,i}$ | 1 | 0 | 1 | 0 | Normal filter off |

Tables 4.5 and 4.6 define the values of the filtered samples, which lie at the third closest columns from the block boundary of blocks P and Q. These samples are modified only when strong filter mode is applied.

Table 4.5 – Definition of the filtered $p'_{2,i}$ sample final value, based on the evaluated conditions

| $p'_{2,i}$ *value* | *Condition 1* | *Conditions 2, 3, 4, 5, 6 and 7* | *Description* |
|---|---|---|---|
| $p_{2,i}$ | 0 | x | Filter off |
| $p_{2,i}$ | 1 | 0 | Strong filter off |
| $p'_{2,i} + \Delta_{p2,i}$ | 1 | 1 | Strong filter on |

Table 4.6 – Definition of the filtered $q'_{2,i}$ sample final value, based on the evaluated conditions

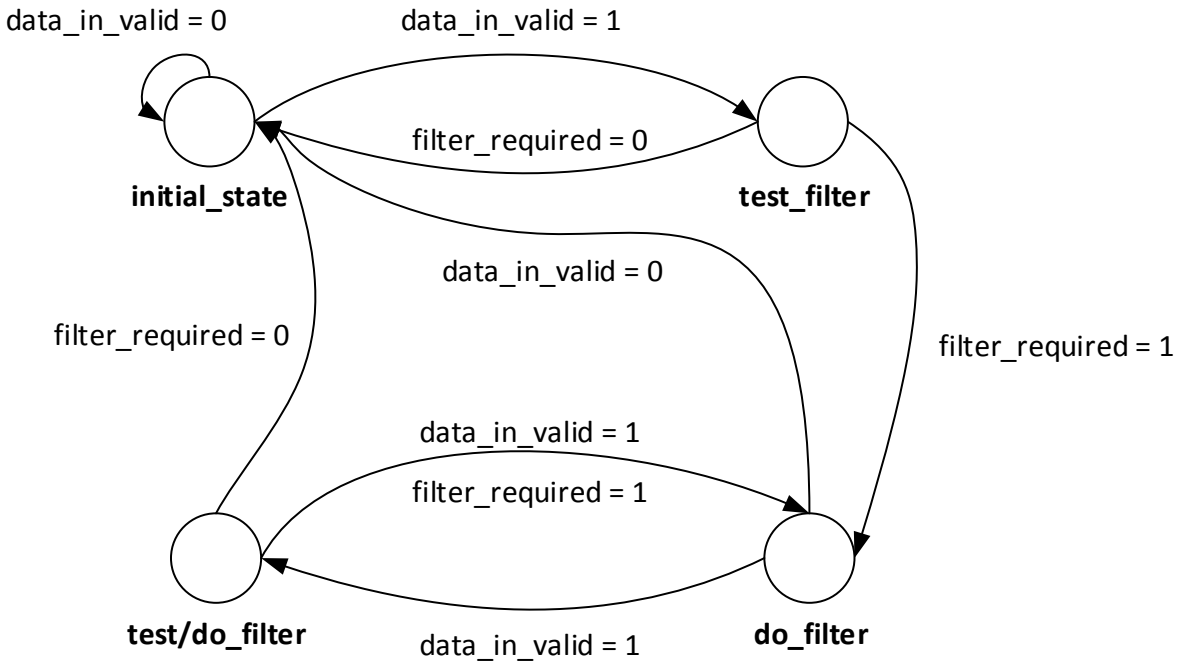
| $q'_{2,i}$ *value* | *Condition 1* | *Conditions 2, 3, 4, 5, 6 and 7* | *Description* |
|---|---|---|---|
| $q_{2,i}$ | 0 | x | Filter off |
| $q_{2,i}$ | 1 | 0 | Strong filter off |
| $q'_{2,i} + \Delta_{q2,i}$ | 1 | 1 | Strong filter on |

## 4.5 Control module

The control module of the architecture is relatively simple, since the deblocking filter is a dataflow application. Thus, complex datapaths were designed but controlled with a simple finite state machine (FSM).

The control module is responsible for handling the handshake protocol between master and slave, as well as selecting the correct filtered output samples based on the conditions described in the previous chapter.

Figure 4.14 – State diagram illustrating the states of the main FSM, as well as the conditions for state changes.




The main FSM has 4 states (see Figure 4.14), whose outputs are described in Table 4.7. As soon as the master starts a transmission, by signalizing with 'data_in_valid' signal, the deblocking filter architecture reads the input in the 'data_in' port. At the next clock cycle, if filtering of the given input is required, the filtered samples are available at the 'data_out' port and 'data_out_valid' signal changes to '1'. The deblocking filter architecture expects to receive the next 2 rows of samples, in order to filter them. At the next clock cycle, these samples will be available at the 'data_out' port. This process can be repeated, resulting in new blocks at every 2 clock cycles or no filtered block (whenever filtering is not necessary) at every clock cycle. The master can stop transmitting data at the end of the second clock cycle, by signalizing 'data_in_valid' with '0'.

Table 4.7 – Definition of the output control signal of the FSM for each state.

| *State* | *'data_out_valid' value* |
|---|---|
| initial_state | 0 |
| test_filter | 0 |
| do_filter | 1 |
| test/do_filter | 1 |

# 5 RESULTS AND COMPARISONS WITH RELATED WORK

## 5.1 Synthesis Results and Performance Estimation

The proposed hardware architecture for the HEVC Deblocking Filter was implemented using VHDL, synthesized and mapped to a Xilinx XC7K160TL-ffg676 Kintex-7 FPGA with speed grade 3 using Xilinx Vivado Tools Suite 2014.2. The timing constraint used is 100MHz. The implementation results are detailed in Table 5.1.

Table 5.1 – Implementation results for Xilinx Kintex-7 FPGA device

| *Resource* | *Utilization* | *Available* | *Utilization [%]* |
|---|---|---|---|
| Slice LUTs | 921 | 101400 | 0.91 |
| Slice Registers | 117 | 202800 | 0.06 |
| I/O | 274 | 442 | 61.99 |
| Clocking | 2 | 32 | 6.25 |

In order to achieve real time video encoding or decoding requirements, the detailed characteristics of the desired video sequence must be analyzed in order to determine the minimum necessary frequency that the architecture must operate. To estimate the minimum clock frequency, we must consider the following worst-case scenario: every boundary between all 8x8 blocks within a video frame will have to be filtered both horizontally and vertically.

Assuming a video sequence with 1920x1080 pixels resolution with 30 frames per second (1920x1080@30fps), there are 64,800 vertical block boundaries and 64,800 horizontal block boundaries. Assuming all of the boundaries must be filtered, the architecture has to filter 129,600 block boundaries. Each block boundary demands 2 clock cycles to be filtered, which will lead us to 259,200 clock cycles, plus 1 clock cycle until the pipeline is filled and 1 until it is emptied, resulting in 259,202 clock cycles in order to filter a single video frame. Since we have 30 frames per second, we need to filter an entire frame at every 33,33ms. Therefore, our target clock frequency for a video of the above mentioned characteristics is 7.776 MHz.

We can generalize the above formula, as described by equation 1, where $Freq$ stands for the minimum targeted frequency, $W$ for the video horizontal resolution

(width), $H$ for the video vertical resolution (height) and $fps$ for the frame rate of the video (measured in frames per second).

$$Freq = \left(\frac{W*H}{16} + 2\right) * fps \quad (1)$$

Assuming a 4096x2304@60fps video sequence, we need a minimum frequency of 35.39 MHz in order to comply with real-time requirements. It must be also considered that we can apply the deblocking filter in parallel to a given video frame, as there are some blocks that do not share any data dependency. Therefore, a higher throughput can be achieved at a cost of multiplying the final on-chip area by the number of desired instances. However, since our architecture is I/O bound (we have a 256-bit input channel and another 256-bit output channel), memory access would become a limitation factor.

### 5.2 Comparison with Related Work

Related work about hardware architectures for HEVC deblocking filter can be found in Ozcan (2013) and Shen (2013). Shen (2013) also considered a memory architecture in order to deal with the transmission of data to the deblocking filter and to transmit the filtered outputs to an external memory in an efficient way, trying to reduce bandwidth. They proposed several BRAM blocks to store specific parts of the quarter-LCU (32x32 block), achieving a smaller amount of required of I/O bandwidth. They adopted a four stage pipeline, which handles two 4x4 blocks (namely P and Q) at a time. Their architecture requires four clock cycles to test and filter all four lines from blocks Q and P, while we need only 2 clock cycles. However, their architecture also implements chroma-deblocking filter, while ours does not. Nevertheless, our technique to design luma datapaths can be easily applied to design chroma datapaths with a increase in FPGA resource usage (by instantiating the chroma datapaths in parallel).

The architecture proposed by Ozcan (2013) follows a similar approach, proposing a memory architecture to handle I/O of a 64x64 LCU, and a datapath capable of processing 4 samples in parallel. All conditions and offset filter values are calculated simultaneously by the datapath. In order to increase throughput, more than one datapath can be instantiated. All instances of the datapath read data from the same memory,

which stores the 64x64 LCU. Therefore, the necessary total time to filter a 64x64 LCU can be decreased.

Differently from the two above mentioned works, our proposed architecture did not take into consideration a memory architecture to store the values of the samples. We chose to have a straight 256-bit wide input and output channels and a simple handshake protocol to control data transfers between master and slave. Therefore, we can reduce complexity and ease integration of our architecture to a heterogeneous SoC, where the implementer would choose how to integrate our module to the existing memory architecture and processor.

Table 5.2 compares the results of our implementation with (Ozcan, et al., 2013). Compared to Ozcan (2013) our architecture achieves 13X performance with a reduction in 83% of the number of Slice LUTs used. We cannot stablish a valid comparison with (Shen, et al., 2013) because their work was only synthesized to ASIC and do not present synthesis results for FPGA

Table 5.2 – Comparisons with Related Work

| | **Ozcan (2013)** | **This work** |
|---|---|---|
| **FPGA device** | Xilinx Virtex 6 | Xilinx Kintex 7 |
| **Slice LUTs** | 5236 | 921 |
| **Slice Registers** | 1547 | 117 |
| **BRAM count** | 8 | 0 |
| **Frequency to process 1920x1080@30fps** | 108 MHz | 7.76MHz |

# 6 CONCLUSIONS

The HEVC standard aims to produce higher data compression in comparison to the previous H.264 standard (Fu, et al, 2012) being specially designed to deal with the new generation of high resolution videos (4Kx2K) (Sullivan, 2012). In order to increase bitstream compression rates, a set of tools that implement algorithms with high computational complexity are executed in order to encode and decode a video sequence. Therefore, not only the amount of data of video sequences has greatly increase with the popularization of beyond high-definition (1920x1080) videos, but also have the tools of the encoder become more complex, aiming to produce higher compression rates.

This work presents an evaluation of the HEVC standard, proving that among all tools, the in-loop deblocking filter has an important role in terms of execution time, being considered a computational hot spot for the HEVC decoder as presented in the profiling section.

In order to deal with that, we presented a hardware architecture that implements the deblocking filter, which intends to provide a higher throughput of filtered data when compared to a software implementation.

We achieved that goal, as our hardware implementation requires a smaller amount of clock cycles to filter a frame of a video sequence, when compared to the amount of clock cycles that a processor running a software based implementation of the deblocking filter would need. Our architecture achieves 13X performance gain over related work with 83% reduction in resource usage.

Our architecture also scales well with the HEVC parallel capabilities (tiles and WPP) because we can instantiate more than a filtering unit in order to process independent frames in parallel. In addition to tiles and WPP, the HEVC deblocking filter has also been designed to be executed in parallel, by not allowing overlapping of samples from different blocks within a video frame, like happened in the H.264 standard (Ozcan, et al., 2013).

However, there is a limit of parallel running instances of our architecture. Our deblocking filter hardware implementation requires large memory access bandwidth, for it needs two 256-bits ports for data transferring between memory and the hardware

datapaths. Therefore, memory access is our main constraint when aiming higher throughput by adding parallel deblocking filter units.

As future work, a prototype of our architecture can be developed, by integrating it to a processor. We can use our deblocking filter module to accelerate execution time of the HEVC deblocking filter, while other tools can still be executed by the processor with a software implementation.

## 6 REFERENCES


AFONSO, V. **Desenvolvimento de Arquiteturas para Estimação de Movimento Fracionária Segundo o Padrão HEVC**. Universidade Federal de Pelotas. Pelotas, p. 53. 2012.

AFONSO, et al. **Low Cost and High Throughput FME Interpolation for the HEVC Emerging Video Coding Standard.** Circuits and Systems (LASCAS), 2013 IEEE Fourth Latin American Symposium (February 2013).

AGOSTINI, L. **Desenvolvimento de Arquiteturas de Alto Desempenho Dedicadas à Compressão de Vídeo Segundo o Padrão H.264/AVC.** Universidade Federal do Rio Grande do Sul (UFRGS). Porto Alegre, p. 173. 2007.

BORDES, et al. **An Overview of the Emerging HEVC Standard.** In International Symposium on Signal, Image, Video and Communications, ISIVC],(July 2012).

BOSSEN, F. et al **HEVC Complexity and Implementation Analysis** IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, 2012, pp. 1685-1696

BULL, D. R. et al. **Introduction to the Issue on Emerging Technologies for Video Compression**. IEEE Journal of Selected Topics in Signal Processing, v. 5, n. 7, p. 1277-1281, Novembro 2011.

C. DINIZ, M. SHAFIQUE, S. BAMPI, J. HENKEL. **High-Throughput Interpolation Hardware Architecture with Coarse-Grained Reconfigurable Datapaths for HEVC.** in ICIP 2013.

G. J. SULLIVAN, J.-R. OHM, W.-J. HAN, T. WIEGAND, **Overview of the High Efficiency Video Coding (HEVC) Standard**, IEEE Trans. Circuits Syst. Video Technol., v. 22, n. 12, pp.1649-1668, Dec. 2012

GUO, Z. **An optimized MC interpolation architecture for HEVC.** Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference (March 2012)

**HEVC HM Reference Software (version 10.0).** https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-10.0/.

ITU-T Rec. H.262 and ISO/IEC 13818-2 (MPEG 2 Video), **Generic Coding of Moving Pictures and Associated Audio Information—Part 2: Video**, ITU-T and ISO/IEC JTC 1, Nov. 1994.

ITU. **ITU-T Recommendation H.264/AVC (05/03): advanced video coding for generic audiovisual services**. International Telecommunication Union. [S.l.]. 2003.

ITU. **ITU-T Recommendation H.265, High Efficiecy Video Coding**. Apr. 2013

http://www.live-production.tv/system/files/The_HEVC_Standard.pdf.

J. FENLASON, STALLMAN R. **GNU gprof.** The GNU Profiler, Free Software Foundation, Inc., 2000

KIM, et al. **High Efficiency Video Coding (HEVC) Test Model 10 (HM10) Encoder Description** 2013

NORKIN, et al. **HEVC Deblocking Filter.** Circuits and Systems for Video Technology, IEEE Transactions on (Volume:22 , Issue: 12 ) (Decenmebr, 2012)

OZCAN, et al. **A high Performance Deblocking Filter Hardware Architecture for High Efficiciency Video Coding.** Consumer Electronics, IEEE Transactions on (Volume:59 , Issue: 3 )(August 2013).

PORTO, M. S. **Arquiteturas de Alto Desempenho e Baixo Custo em Hardware para a Estimação de Movimento em Vídeo Digitais**. Dissertação (Mestrado em Ciência da Computação) - Universidade Federal do Rio Grande do Sul. Porto Alegre, RS. 2008.

SHEN, et al. **A high-throughput VLSI architecture for deblocking filter in HEVC.** Circuits and Systems (ISCAS), 2013 IEEE International Symposium