FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Verilog implementation of the VESA DSC compression algorithm

Carlos Alberto Pereira Ferreira

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor at FEUP: Prof. José Carlos Alves

Supervisor at Synopsys: Eng. Rui Raínho Almeida

July 27, 2016

Resumo

Ao longo do tempo, tem havido um aumento das resoluçoes suportadas pelos dispositivos multimédia. Para além disso, do ponto de vista da indústria, é vantajoso prolongar o uso das interfaces de vídeo existentes, o que garante a compatibilidade com dispositivos existentes e ao mesmo tempo assegurar o suporte de novos formatos de vídeo de maior resolução.

As técnicas de compressão permitem reduzir a informação redundante existente em sinais de vídeo. Os sinais de vídeo possuem informação que o sistema visual humano não consegue processar, logo, ao eliminar essa informação, os requisitos de largura de banda diminuem. Esta diminuição de largura de banda necessária, permite prolongar a longevidade das interfaces existentes como por exemplo, o HDMI.

O objectivo desta dissertação é implementar o algoritmo de compressão VESA DSC.
É necessário que a implementação seja capaz de processar o formato de vídeo 4K (no caso das tecnologias de 28nm e 40nm) e o formato de vídeo FullHD no caso da implementação em FPGA (Xilinx Virtex-7 XC7VX690TFFG1926-2).

Antes de descrever o funcionamento do codificador DSC, são apresentados conceitos básicos da teoria de compressão de vídeo, assim como, algum trabalho na área. A descrição do funcionamento do DSC é complementada com a exposição de detalhes de implementação que requerem alguma tomada de decisão ou que implicam um trade-off entre requisitos incompatíveis.

O algoritmo foi validado funcionalmente através da comparação com os resultados produzidos por um modelo de referência em C. Fazendo uma média entre os diversos tipos de "cobertura" de código, a pontuação obtida é de 83.58%. Trabalho futuro irá melhorar estes resultados.

A frequência máxima de relógio para a tecnologia de 28nm é de 200MHz enquanto que para a de 40nm é de 145 MHz. No caso da FPGA, a frequência máxima é de 47 MHz. Conclui-se que é necessário usar várias instâncias do codificador a funcionar em paralelo para poder processar os formatos de vídeo exigidos.

Abstract

Over the years, the demand for multimedia devices with higher resolutions has been increasing. Besides that, from the industry point of view, there is an interest in prolonging the use of existing video interfaces to guarantee the interoperability with existing devices while guaranteeing the support of new video formats with higher resolutions.

Compression techniques offer the opportunity of reducing the redundant information existing in video signals. Video signals possess information that the human visual system can not process, so, by eliminating it, the bandwidth requirements diminish. This diminish of bandwidth requirements, permits prolonging the longevity of existing interfaces, such as HDMI.

The dissertation objective is to implement the DSC encoder algorithm. It is required to be able to process 4K video signals in the 28nm and 40nm technologies and FullHD video signals in a FPGA (Xilinx Virtex-7 XC7VX690TFFG1926-2). Before discussing the operation of the DSC encoder, some basic concepts of video compression theory are discussed, and some work on the field is presented. The discussion of the basic principles of operation of the DSC encoder is complemented with a discussion of relevant implementation details that require some decision taking or imply trade-offs between conflicting requirements.

The algorithm was functionally validated by comparing it with the output of a reference C model. The tests performed permitted a score of 83.58%. This score value results from taking the average between the different types of coverage metrics. Future work will improve this result.

The maximum clock frequency achievable is 200 MHz for the 28nm technology and 145 MHz for the 40nm technology. Relative to the FPGA, the maximum achievable frequency is 47 MHz. It is concluded that it is necessary to use several encoder instances working in parallel to be able to process video signals with the desired resolutions.

Acknowledgements

I would like to express my gratitude to my supervisor at FEUP, Professor José Carlos Alves for always being available to guide me through my dissertation. His insightful comments and writing revisions were of invaluable importance to me.

I would like to also express my gratitude to my supervisor at Synopsys, Eng. Rui Raínho Almeida for all the invaluable help he provided me in the implementation phase of my dissertation.

I thank all my colleagues at Synopsys. In particular, I am grateful to Eng. Tiago Campos for all the support he provided me.

To all my friends, a thank you very much, for all the support and motivation.

Last but not the least I want to thank my family for supporting me throughout my life. Without them I would not be able to stand where I am.

Carlos Alberto Pereira Ferreira

“A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools.”

Douglas Noel Adams

Contents

List of Figures

List of Tables

Abbreviations

ASIC	Application Specific Integrated Circuit
AVC	Advanced Video Coding
BP	Block Prediction
bpc	bits per component
bpp	bits per pixel
CBR	Constant Bit Rate
CODEC	COderDECoder
DCT	Discrete Cosine Transform
DP	DisplayPort
DSC	Display Stream Compression
DUV	Device Under Verification
DVC	Distributed Video Coding
DWT	Discrete Walsh Transform
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FSM	Finit State Machine
HDMI	High Definition Multimedia Interface
HEVC	High Efficiency Video Coding
HVS	Human Visual System
ICH	Indexed Color History
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
ITU-T	Telecommunication Standardization Sector of the International Telecommunications Union
JPEG	Joint Photographic Experts Group
MAD	Mean Absolute Difference (MAD)
MB	Macro Block
MCP	Motion-compensated Prediction
ME	Motion Estimation
MIPI CSI	Mobile Industry Processor Interface Camera Serial Interface
MIPI DSI	Mobile Industry Processor Interface Display Serial Interface
MMAP	Modified Median-Adaptive Prediction
MPEG	Moving Picture Experts Group
MPP	Midpoint Prediction

MSE	Mean Square Error
MV	Motion Vector
PPS	Picture Parameter Set
QoE	Quality of Experience
QP	Quantization Parameter
RC	Rate Control
ROI	Region-of-Interest
SAD	Sum of Absolute Difference
SIFT	Scale Invariant Feature Transform
SR	Shift Register
SSP	Substream Processor
UHDTV	Ultra High Definition Television
VBR	Variable Bit Rate
VESA	Video Electronics Standards Association
VLC	Variable Length Code
VLSI	Very-large-scale integration

Chapter 1

Introduction

1.1 Context

Over the years, display manufacturers have been increasing the resolution of their products to attend to the more exigent user requirements and of course to differentiate from other products in the market [1]. For example, High Definition Multimedia Interface 2.0 (HDMI 2.0), that implements the CEA-861F standard supports resolutions of 4096 x 2160p @ 60Hz [2] with 16 bits per pixel component (48 bits per pixel) which requires high a bandwidth (530.8 Mpixel/s which is approximately 25.48 Gbit/s) from the transport layer. Higher resolutions will be widely available in the future such as 8K, which means more exigent requirements for the transport layer, that implies more expensive data links, that consume more power and are much harder to implement [1].

One possible solution is to increase the number of data links between the source device and the sink device [1]. For example, if we increase four times the number of data lanes, then we can increase four times the number of bits per second or reduce the clock speed four times. The problem with this approach is that not only we increase the complexity of the hardware, but also existing connectors and cables (the physical layer) would not be compatible, which means that a new physical layer would have to be implemented.

Another approach to deal with the increasing bandwidth requirements, is to compress the video stream. Instead of sending raw data through the data link, as it is currently done with HDMI, we can take advantage of the characteristics of the Human Visual System (HVS) and remove information from the video stream that people will not perceive. For example, we can compress high frequency content without that being noticeable, but we must preserve low frequency content such as uniform regions in a video frame. Compression allows us to reduce the bandwidth requirements by reducing the amount of information being send through the data links, so changes aren't necessary to the physical layer of nowadays interfaces such as HDMI, DP e MIPI DSI/CSI.

VESA Display Stream Compression (DSC) standard specifies algorithms for compression and decompression of video streams [3]. Those streams are between two devices [3] such as BluRay disc player and a display . VESA DSC is compatible with HDMI, DP and MIPI DSI/CSI interfaces and is designed for working in real-time [3].

1.2 Motivation

There are several video compression techniques available. Transform techniques map the input signal from the space-time domain representation to the frequency domain representation. The idea is to concentrate the energy of the input signal in the lower frequency components (the HVS is more sensible to low frequencies) and discard the high frequency coefficients [4]. Choosing a suitable transform function depends on the statistical properties of the signal. A popular transform function is the Discrete Cosine Transform (DCT) [4] and is used in many compression schemes such as H.264, MPEG-2, and JPEG.

Another type of transform is the wavelet transform, that maps the input signal to the time-scale domain [4], and has good performance with transient signals and is used, for example, in JPEG 2000.

Compression can also be lossless by reducing the statistical redundancy, which means representing the same information using fewer bits, using for example, Huffman codes, but the compression ratios are lower than lossy schemes. Lossless compression is based on information theory and is used in many compression schemes for audio and video, in conjunction with lossy techniques.

Movement estimation techniques can be used to remove redundancy between different frames of a video. If two consecutive frames are very similar, we only need to transmit the differences between them, which requires much less information than the whole frame.

As the reader can see, there are many compression schemes available that exploit the HVS and different types of redundancy existing in video signals, so at this point it is relevant to analyze why the necessity of another compression scheme, namely, the VESA DSC.

Many existing compression schemes are not visually lossless, which means the user can observe the degradation of quality. The current video compression methods are computationally expensive to implement because require storing many pixels in a buffer which increases the area of a integrated circuit, and some of them are proprietary [1]. If an algorithm is proprietary then the company who is making a new product may need to pay royalties for the algorithm, or instead use another algorithm, and the product will not be compatible with other products that implement the first algorithm. VESA DSC is a new video compression algorithm designed to be widely used in the industry (interoperable), visually lossless and low cost. The requirements that lead to the development of the VESA DSC, and that the existing compression schemes do not meet can be found on [1] and are :

- Compression should be visually lossless, which means that the user shouldn't perceive any degradation of video quality
- The algorithm should be able to work at low rates and support a constant data rate of compressed video
- Support handling sub-regions of a image (slices) that work as a single picture independent from other slices

- Support different colour spaces, bits per pixel and sub-sampling schemes
- Work in real-time and inexpensive (in terms of memory requirements) to implement in hardware

1.3 Objectives

The objective of this dissertation is to build a digital implementation in Verilog HDL of the VESA DSC v1.1 Standard. The implementation must be synthesizable for ASIC (TSMC28 and TSMC40LP) and FPGA (Xilinx Virtex-7 XC7VX690TFFG1926-2) technologies.

It is desired to be able to process process 4K video signals in the ASIC technologies and FullHD[1] video in the FPGA implementation. This roughly corresponds to a pixel frequency of 600 MHz for the ASIC technologies and 300 MHz for the FPGA technology. It will be verified if it is possible to achieve this performance with one encoder instance. This implies that the implementation must be able to achieve a clock frequency of at least 600 MHz in the 28nm technology (ASIC) and 300 MHz in the FPGA. If it is not possible to achieve such frequency values, other solutions to overcome this problem will be explored, namely, the parallelization of the algorithm. Using several encoders in parallel it is possible to lower the clock frequency requirements at the expense of an increase in the circuit area.

1.4 Document Outline

This dissertation is organized as follows:

Chapter 2 discusses some basic concepts related to video compression theory. Some work related to other compression algorithms is also presented.

Chapter 3 explains the basic principles of working of the DSC encoder.

Chapter 4 deepens the discussion in chapter 3. It will be presented to the reader some interesting implementation details, that involve some decision making. Operations that are simply a direct implementation of the Standard are not discussed.

Chapter 5 starts by a discussion of the test procedure followed by the test coverage results. Finally, the synthesis results are exposed and discussed.

The document ends with the Conclusion chapter.

[1]It is being considered a resolution of 1920 x 1080p @ 100 Hz.

Chapter 2

Video compression

This chapter reviews some concepts about video compression theory. The requirements and challenges imposed by the recent ultra-hight resolution video formats will also be discussed. This chapter ends with a discussion of some existing solutions in the field of video compression and a brief reference to machine learning approaches to video compression, that significantly differ from ”traditional” approaches.

2.1 Introduction

The amount of data needed to be stored and transmitted has been increasing over the time and nowadays it is almost impossible to do it without compression. Table 2.1 shows some requirements in terms of storage/transmission of typical applications, assuming that compression is not being used. In the case of the video stream, it is assumed that sub-sampling scheme is 4:4:4 and no blanking pixels are being used, which is a simplification that may not be valid in practice. Nevertheless it shows that compression schemes are essential.

Data compression algorithms, are used to reduce the number of bits required to represent data. There are two types of compression: *lossy* compression and *lossless* compression. In the case of lossy compression, the reconstructed data is different from the original data. In the case of lossless compression, the reconstructed data is the same as the original data. Typically lossy compression provides higher compression ratios (relationship between the original video data rate and the encoded video data rate) than lossless compression [5].

Table 2.1: Storage/Transmission requirements for uncompressed data.

Application	Bit rate	Debit
(Standard Audio CD) 2 channel, 16 bit @ 44100 Hz	1411.2 Kbits/s	620.2 MB/h
4k video (4096 x 2160p @ 60 Hz, 48 bit/pixel)	25.48 Gbit/s	11.47 TB/h

A typical encoder based from [6] is shown on figure 2.1. The decoder must take the inverse steps to recover the video sequence. The coder and decoder together are referred as CODEC [7]. Typically video compression schemes explore temporal and spacial redundancies existing in video signals to reduce the bit rate. The entropy encoder in figure 2.1 reduces the amount of redundancies existing in the representation of information by representing the same information using fewer bits. In the following sections these topics will be further discussed.

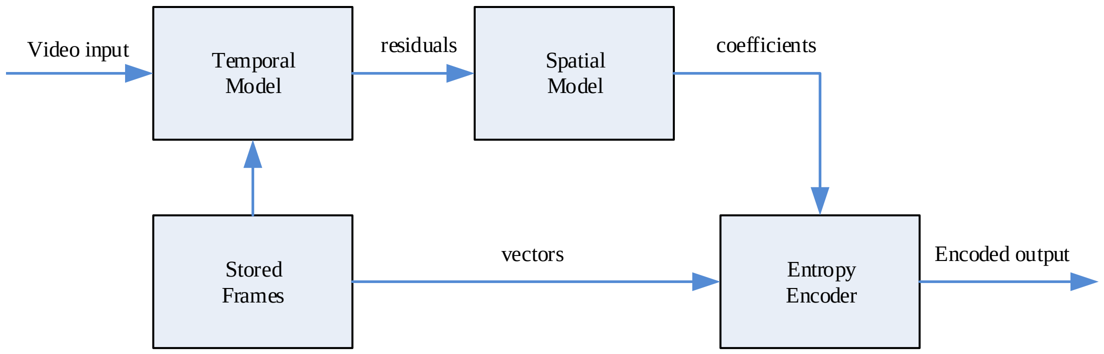


Figure 2.1: Typical video encoder block diagram.

2.2 Lossless compression

When lossless compression is used, no information is lost. The decompressed signal is equal to the original one. To understand how lossless compression works, the must know some concepts of information theory. Information theory is a very complex and broad subject so, only some very rudimentary information will be presented here. More information can be found on Information Theory literature such as [8].

Equation 2.1, extracted from [8] represents the entropy of a random variable X whose probability density function (pdf) is $p(x)$. Since the logarithm is in base two, the entropy is measured in bits.

$$H(X) = -\sum_{x} p(x)\log_2 p(x) \tag{2.1}$$

Analyzing equation 2.1 it resembles the equation of the expected value of a random variable. Indeed, entropy is the expected value of information because $-log_2 p(x) = log_2 \frac{1}{p(x)}$ represents the amount of information of a given symbol x produce by source X, so entropy gives the minimum average number of bits to represent each x [9].

It is shown in [8] that the average number of bits L to represent some random variable X is greater or equal than the entropy:

$$L \geq H(X) \tag{2.2}$$

To understand the role of the previous results in data compression consider an example extracted from [9]. If a given picture as an uniform histogram and has 256 gray-level values, then the entropy is 8 bits. This result was expected because if there are 256 possible values, then with 8 bits we can "encode" exactly 256 "things". One should notice that in this example it is impossible to use less bits to represent each gray-level. This result agrees with equation 2.2.

Now let's consider another example extracted from [8] where $p(x)$ is not uniform. The example consists of a horse race between 8 horses. Not all horses have the same probability of winning. The probability of each one winning is $\left(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}\right)$. The entropy is straightforward to compute and is 2 bits. Now consider the problem of sending the information which horse won the race. One can say that by using 3 bits, it is possible to transmit that information because $2^3 = 8$. But, if we use a variable number of bits, namely, using less bits to represent the horses which are more likely to win, then, on average less bits need to be transmitted. Using the result on equation 2.2 the average number of bits necessary is 2, which corresponds to the entropy.

Variable length coding (VLC) is used in many compression algorithms to reduce coding redundancy, which means using fewer bits to represent the same information without any information loss [6]. Huffman coding and arithmetic coding are two examples of variable length codes. Even lossy compression schemes typically use lossless techniques to achieve a further reduction in the bit rate, without diminishing the video quality. Lossless compression works well with synthetic images that have large areas of uniform color.

2.3 Lossy compression

As the name suggests, lossy compression implies loss of information. When developing a lossy compression scheme it is important to study the statistical properties of the signals that are going to be processed (in this dissertation it's video signals), to identify irrelevant information (redundant) that can be discarded without a major prejudice to the perceived quality. In video signals, spatial and temporal redundancies can be exploited for video compression [10].

Spatial redundancy is related to the similarity between pixels that are near each other within a frame. The more uniform a given region within a frame is, the more correlated the pixels are, and more opportunities for compression are available. In an extreme but unrealistic case, if in a given frame, all pixels are equal then only one pixel would be necessary to represent the frame, because the others pixels would be a replication of that same pixel.

Temporal redundancy is related to the similarity (correlation) between successive frames which tends to increase with the frame rate [10]. When the frame rate is high with respect to the movement of objects in the video scene, a substantial portion of two successive frames may be very similar. If instead of coding each frame individually, only the displacement is coded [7], then the amount of information to transmit/store diminishes. The information in the difference between adjacent frames may be further reduced by combining motion estimation with frame differentiating.

If we consider the characteristics of the Human Visual System (HVS), we have more opportunities to remove redundant data, namely, psychovisual redundant data [11]. Let's review some of the characteristics of the HVS and opportunities to eliminate unnecessary information:

- Contrast Sensitivity - If a signal has contrast below a threshold for a given frequency, the signal won't be detected by the HVS. The model used do describe this behavior is a low-pass filter. The HVS is less sensitive to errors in high frequency regions than in low frequency regions [12]. A high frequency region is described by rapid changes in color or brightness, while in low frequency regions, these changes are more smooth. Since our eyes behave like a low pass filter, high frequency content can be eliminated without diminishing the perceived quality.

- Masking - The presence of a signal reduces the visibility of another signal. In the spatial domain this means that a set of pixels with a small deviation from their correct values is much harder to detect in a image region rich in textures than in a smooth area. In the temporal domain, the masking effect is related to the difference in luminance between video frames [12].

- Visual attention - The HVS tends to focus on certain objects or regions of the video frame. For example, studies show that there is a natural tendency to look at the center of the scene and also to give attention to human faces. These regions that attract more attention are know as ROI (region-of-interest). The ROI should be encoded with higher quality, to diminish the compression impact in the perceived quality. There are several difficulties related to using the ROI because it requires using specialized hardware like a eye tracker or an expert must identify *a priori* these regions [13].

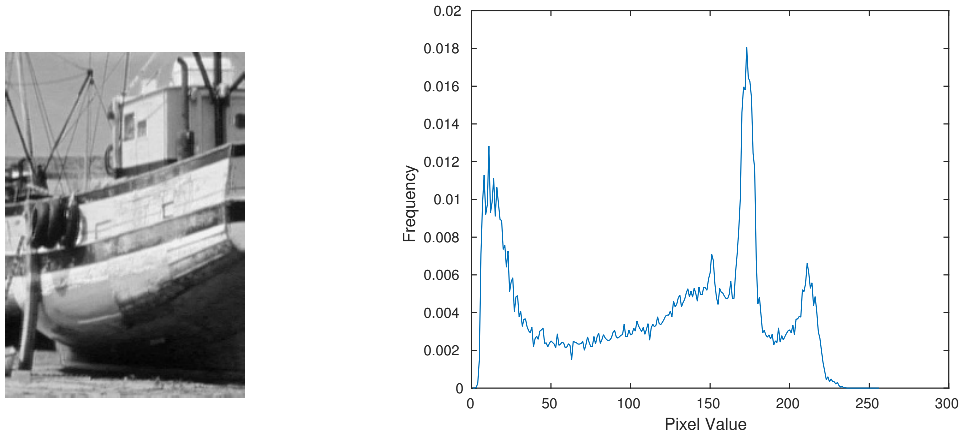
2.3.1 Spacial redundancy reduction

In this section differential coding also know as predictive coding will be described.

As described earlier, video signals usually present high spatial and temporal redundancies. Due to these redundancies, it is more efficient to encode the difference between the input signal and it's prediction[1] instead of coding the input signal directly [14], as long as the prediction is a good representation of the pixel being coded. These "differential" pixels have a high probability of assuming small amplitudes (near zero), so less bits are required to represent the original information. It is important to underline the fact that differential coding by itself doesn't compress data, it's the quantization process, to be described later, that in fact compresses data.

As an example of application, consider figure 2.2a and figure 2.2b that show the original image being coded and the respective histogram. As the reader can observe, the pixel values are well distributed in the range [0,255].

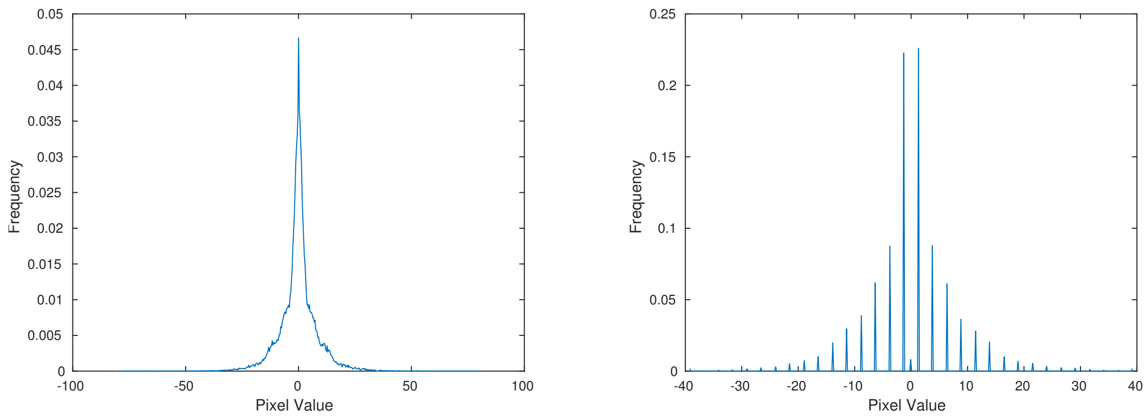
[1]The prediction is constructed using information of previous pixels only.



(a) Original boat image. (b) Boat image histogram.

Figure 2.2: Predictive coding example.

The difference image was determined, and the respective histogram is shown on figure 2.3a. As the reader can confirm, the pixel values are concentrated near zero, which implies a low variance. Figure 2.3b shows the histogram after applying a quantization operation on the difference image. The number of bits used to represent the information is 5, so, only 32 values of amplitude are available, which modifies the histogram.



(a) Difference image histogram. (b) Quantized difference image histogram.

Figure 2.3: Predictive coding example - cont.

2.3.2 Temporal redundancy reduction

As stated earlier, video signals usually possess high temporal redundancy. The frame rate of a video is typically high with respect to the image moving scenes, and by that reason, most of the contents of consecutive frames are very identical. Instead of coding the two consecutive frames as being independent [9], it is possible to take an approach similar to the one described in section 2.3.1 but this time applied to pixels in different frames.

Another approach that gives better compression ratios is to detect the regions that suffered a displacement between successive frames and calculate the vector that describes the movement, motion vector (MV). Thus a given region in a frame, can be predicted by displacement of a region, in another frame. This technique is known as motion-compensated prediction (MCP) [7]. Motion estimation (ME) is the process of finding suitable MVs, and the coding of the resulting difference signal, is know as MCP residual coding. The disadvantage of this approach is the added complexity requirements.

The basic steps are:

- Search for motion vectors - The idea is to find the best MVs according to a given criteria as shown later on this section.
- Prediction and differentiation - The present frame is predicted by the calculated MV in the previous step, and the previous frame. Prediction error is also calculated which is the present frame minus the predicted present frame [6].
- Encoding of the prediction error and motion vectors.

Motion compensation is performed by dividing the image into blocks know as *macroblocks* (MBs) of size N x N [9]. For example, if luminance images have size 16 x 16, then if a 4:2:0 chrominance scheme is used (horizontal sub-sampling), the MBs of the chrominance image will have a size of 8 x 8. If the 4:2:0 scheme is used (horizontal and vertical sub-sampling), then the MBs for chrominance images will have size 4 x 4. It is important to emphasize that the choice of the MBs size affects the performance of the algorithm. When smaller MBs are used, the estimation of movement is more precise, but that means an increase in the number of computations, number of MVs and information that is encoded and transmitted [6].

The task of finding the "best" MVs is an expensive one, so, usually the search is made in a small search window of known size. To decide the "best" match, several criteria exist like for example, Mean Absolute Difference (MAD), Sum of Absolute Difference (SAD) and Mean Square Error(MSE).

The MAD criteria is defined in equation 2.3, extracted from [9].

$$MAD(i,j) = \frac{1}{N^2}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1}|C(x+k,y+l) - R(x+i+k,y+j+l)| \tag{2.3}$$

A word about notation: p is an integer that will define the window size; k and l are indices for pixels in the MB; i and j represent the horizontal and vertical displacement; $C(x+k,y+l)$ are the pixels in the MB in the target frame; $R(x+i+k,y+j+l)$ are the pixels in the MB in the reference frame.

The objective is to find de MV (i,j) that minimizes a given criteria, in this case, the $MAD(i,j)$, for $i,j \in [-p,p]$.

There are several search algorithms. In table 2.2 some existing search algorithm and the required number of operations for second are shown.

As we can see on table 2.2 the Hierarchical Search is the search method that requires less operation per second. Sequential search is to heavy for real time encoding.

Table 2.2: ME algorithms comparison.

Search algorithm	Number of operations per second
Sequential Search	$(2p+1)^2 * N^2 * 3$
2D Logarithmic Search	$8 * (\lceil log_2 p \rceil + 1) + 1$
Hierarchical Search	$\left[\left(2 * \lceil \frac{p}{4} \rceil + 1\right)^2 \left(\frac{N}{4}\right)^2 + 9\left(\frac{N}{2}\right)^2 + 9N^2 \right] * 3$

2.3.3 Transform Coding

Transform coding is based on applying a mathematical transform to the image pixels so that the relevant image information is concentrated in a small number of transform values. Let $g(x,y)$ represent the gray level of the pixel at coordinates (x,y) and $T(u,v)$ represent a 2-D transformation where (u,v) are the pixel coordinates in the new domain [6].

The equations that represent the forward and inverse transformations were extracted from [6] and are respectively:

$$T(u,v) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} g(x,y) f(x,y,u,v) \tag{2.4}$$

and

$$g(x,y) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} T(u,v) i(x,y,u,v) \tag{2.5}$$

It can be shown that if the transformation kernel is symmetric, than the equation 2.5 can be represented as:

$$G = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} T(u,v) I_{u,v} \tag{2.6}$$

where $I_{u,v}$ is the basis image:

$$I_{u,v} = \begin{pmatrix} i(0,0,u,v) & i(0,1,u,v) & \cdots & i(0,N-1,u,v) \\ i(1,0,u,v) & i(1,1,u,v) & \cdots & i(1,N-1,u,v) \\ \vdots & \vdots & \ddots & \vdots \\ i(N-1,0,u,v) & i(N-1,1,u,v) & \cdots & i(N-1,N-1,u,v) \end{pmatrix} \tag{2.7}$$

Equation 2.6 means that the picture G (which is the collection of $g(x,y)$ for all possible values of x and y) can be represented as a weighted sum of the basis pictures $I_{u,v}$, where $T(u,v)$ represents the weights used in the summation. These weights measure the amount of correlation between the image G and the basis $I_{u,v}$, as described in [6, 15].

It is shown on [15] that for some choices of basis images, after a certain number of terms, the contribution of the weighing coefficients becomes negligible to the summation, so they can be discarded. The mean-square error (MSE) is also shown to be, the sum of the variances of the discarded weighing coefficients.

To achieve compression, it is important to choose a transformation (set of basis images) that concentrates the variance in a small number of weighing coefficients, which means, concentrating the energy of the signal in a small number of terms, and discard the terms that are irrelevant [15]. The weighing coefficients are then quantized and encoded.

There are some considerations that must be taken into account when quantizing the coefficients. The coefficients with higher variance, have more impact on the quality of the decompressed image, so, more bits should be used to represent those coefficients. The other ones can be represented with less bits, because they have less impact in the perceived quality of the decompressed image. The characteristics of the HVS can also be taken into account, namely, the frequency response of the eye, and so, coefficients whose frequencies correspond to the ones which our eyes are sensible, should be encoded with more bits [15]. It is important to emphasize that the transformation by itself doesn't compress data, it's the quantization process that indeed removes information.

There are several transformation functions in the literature, used for static image compression and video compression, like the discrete cosine transform (DCT), the discrete Walsh transform (DWT) and the discrete Hadamard transform. The DCT is used in several video coding standards such as H.264, MPEG 1, 2, 4 due to it's properties, like the decorrelation property [16] and energy compaction property.

Equations 2.8 and 2.9 were extracted from [6] and represent the 2-D DCT transformation kernel. $C(v)$ in equation 2.8 is the same as $C(u)$ in equation 2.9.

$$f(x,y,u,v) = i(x,y,u,v) = C(u)C(v)\cos\left(\frac{(2x+1)u\pi}{2N}\right)\cos\left(\frac{(2y+1)v\pi}{2N}\right) \tag{2.8}$$

$$C(u) = \begin{cases} \sqrt{\frac{1}{N}} & ,u = 0 \\ \sqrt{\frac{2}{N}} & ,u = 1,\dots,N-1 \end{cases} \tag{2.9}$$

Figure 2.4 shows the 8x8 DCT basis images.

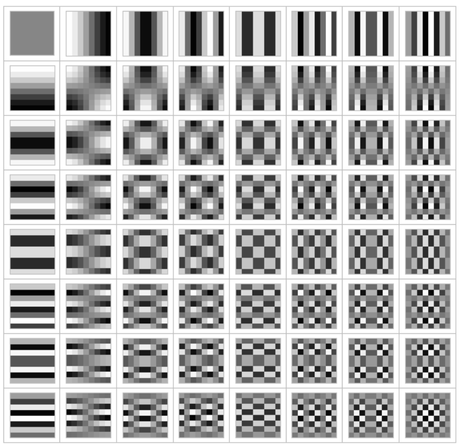


Figure 2.4: DCT 8x8 Basis Images.

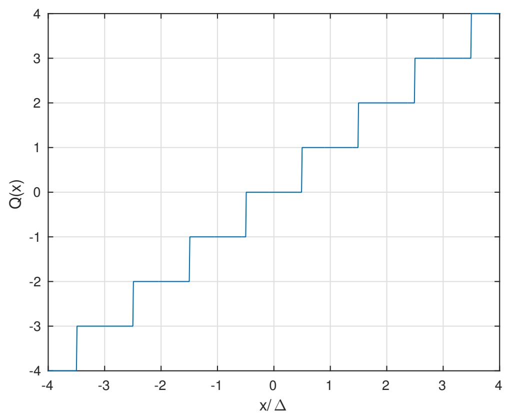
2.3.4 Quantization

A quantizer partitions the range of input values into a finite number of intervals. The endpoints of a interval are known as *decision boundaries*. The Δ symbol is used to represent the *step size* which is the length of the interval [9].

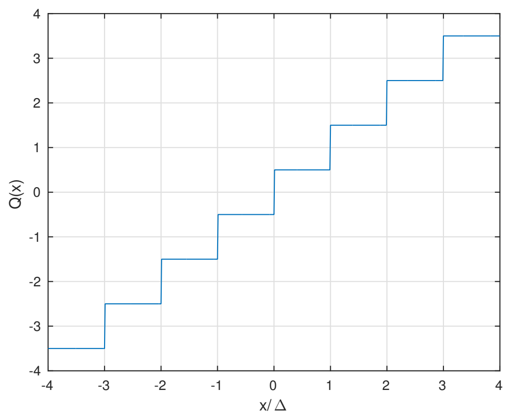
2.3.4.1 Uniform scalar quantization

In an uniform scalar quantizer, the step size is the same for all intervals. The output value of each interval is the midpoint of the interval [9].

We can distinguish two types of uniform scalar quantizers, namely, *midtread* and *midrise*. If the value zero is a possible output value, then the quantizer is of the midtread type, otherwise is midrise type. Figure 2.5a illustrates a input-output map of a midtread quantizer, while figure 2.5b shows the input-output map of a midrise quantizer.



(a) Midtread quantizer.



(b) Midrise quantizer.

Figure 2.5: Uniform quantizers.

As wee can see, the number of possible values at the quantizer output, is less than at the input side, which means that the quantization operation inserts distortion. When we increase the number of intervals, intuitively we expect that the distortion due to quantization will be lower, however, since each level is coded with a binary code, more bits will be necessary to represent the data. These concepts will be proved mathematically.

If the quantizer has M possible output values, then, the output binary rate is:

$$R = \lceil \log_2 M \rceil \tag{2.10}$$

Assuming that the range of the input signal belongs to the interval $[-X_{max}, X_{max}]$ and the values are uniformly distributed, then:

$$\Delta = \frac{X_{max} - (-X_{max})}{M} = \frac{2X_{max}}{M} \tag{2.11}$$

It is shown on [9] that, when the input values are uniformly distributed, the quantization error $e(x) = x - \frac{\Delta}{2}$ is also uniformly distributed on the interval $[-\frac{\Delta}{2}, \frac{\Delta}{2}]$. The mean value of $e(x)$ is $\bar{e}$ which is zero, and so, the MSE is equal to the variance which is given by:

$$\sigma_e^2 = \frac{1}{\Delta} \int_0^\Delta (e(x) - \bar{e})^2 \, \mathrm{d}x = \frac{\Delta^2}{12} \tag{2.12}$$

The signal variance is given by:

$$\sigma_x^2 = \frac{(2X_{max})^2}{12} \tag{2.13}$$

With the two previous equations we can compute the signal-to-quantization noise [9]:

$$SQNR = 10log_{10}\left(\frac{\sigma_x^2}{\sigma_e^2}\right) = 6.02n(dB) \tag{2.14}$$

The previous equation, permits us, for a desired SQNR value, find the necessary number of bits used to represent the output values of the quantizer, under the assumptions in which the equation was derived.

2.3.4.2 Nonuniform Scalar Quantization

Sometimes the assumptions that were used on uniform quantization don't apply. The distribution of input signal levels may be more dense near some decision levels. In that case a nonuniform quantizer should be used. The number of decision levels in those dense regions should be higher to have a finer quantization [9]. In the remaining regions, the number of decision levels can be lower (which will give a coarser quantization).

Lets consider the Lloyd-Max quantizer. Let $\beta = (\beta_0, \beta_1, \ldots, \beta_M)$ represent the set of the possible decision boundaries and $Y = (y_1, y_2, \ldots, y_M)$ represent the possible output values. Then it can be shown that the optimal values for each β_j and y_j (that give the lower distortion) are given respectively by:

$$y_j = \frac{\int_{\beta_{j-1}}^{\beta_j} x f_X(x)\,\mathrm{d}x}{\int_{\beta_{j-1}}^{\beta_j} f_X(x)\,\mathrm{d}x} \tag{2.15}$$

and

$$\beta_j = \frac{y_{j+1} + y_j}{2} \tag{2.16}$$

where $f_X(x)$ is the probability density function of the input signal's levels.

2.4 Requirements

When designing or using a compression scheme it is important to consider the application's requirements that obligate to use image compression and the impact of the compression performance in meeting those requirements. Some typical requirements [17] are shown next.

- Real time constraints
- Visual quality
- Power consumption
- Memory requirements
- Maximum bit rate of the compressed bit stream

The encoding and decoding operations take some time that depends on the complexity of the compression scheme. Some applications like video conferencing and streaming have real time constraints, namely the delay between successive video frames should be constant, otherwise it will be perceived by the user, lowering the *QoE* (Quality of Experience). Other applications such as a video game console connected to a TV, have higher requirements related to the delay inserted. Not only the delay must be constant, but also, it must be small, to ensure the timeliness of the player's response to a given game event for example.

Lossy compression deteriorates the quality of the original video sequence by introducing compression artifacts such as blocking, ringing [18] and temporal artifacts like flickering [19]. These coding artifacts can be more or less pronounced depending on the data rate at the output of the encoder. When the data rate is lowered, the compression ratio is higher, but the video quality lowers. So, for a given application there is a trade-off between the compression ratio and the visual quality, that imposes restrictions in selecting a compression scheme.

Memory requirements can also be a limitation when choosing a compression scheme. For example, let's consider that for a given algorithm it is necessary to store a complete frame and the video resolution is 4096 x 2160p. Even if the blanking pixels are not considered, the memory buffer must store $8,847,360$ pixels, which means, if each pixel is encoded with 36 bits (12 bits per pixel component) it's necessary to have approximately 38 MB which can be a problem if the algorithm is to be implemented in an integrated circuit.

Power consumption considerations must be taken account is some cases, specially for mobile devices. In that situations special attention must be given to the complexity of the encoder and decoder to ensure that power consumption requirements are met [20].

Other requirements can be related to error resilience capabilities, that is, how errors inserted by the communication channel affect the decompressed video quality, supported video formats and frame rates.

2.5 Video Coding standards

Over the years several video coding standards have been proposed. A standard specifies the data format at the output of the encoder and specifies the conforming decoder output results [21]. The objective of the standards is to guarantee interoperability between devices only. This dissertation is about VESA DSC which is a recent video coding standard. Although no relevant work has been published yet, Hardent has an implementation of the encoder and decoder already available in the market.

In the next sections some work related to popular video coding standards will be presented. Distributed video coding and machine learning approaches are alternative approaches to the "traditional" video compression standards, and are briefly discussed in section 2.6 and 2.7 respectively.

2.5.1 H.264/AVC

The H.264/AVC is one of the most used video coding standards and has applications in video telephony, storage, streaming and broadcast applications [22]. H.264/AVC uses the tools described in chapter 2, namely, intra-frame prediction, transform coding, quantization, entropy coding and inter-frame prediction. Some pertinent work on this standard will be described next.

The paper [23] proposes an H.264 intra-encoding architecture capable of handling 4 FullHD video streams at 25 fps in real-time. The solution addresses the problem of encoding simultaneously views in multicast video applications. To achieve an area efficient solution, the hardware is shared in a time-multiplexed way between the different views.

Motion estimation and compensation might not be adequate in some applications, specially when low power requirements are considered [24]. Designing a intra-frame architecture to reduce the spatial redundancy at high resolutions is a challenging task. It is difficult to use parallel and pipelined architectures because of data dependencies, and the amount of data involved. In [24] was proposed an intra-prediction implementation in 65nm lithography that supports resolutions of 7680 x 4320 @ 60 fps running at 273 MHz.

A perception-based video coding that detects human faces was proposed on [25]. Remember from the discussion on chapter 2, that the HVS as a tendency to focus on some regions (ROI) of the frame and human faces are generally a ROI. The algorithm proposed, uses an adaptive boosting (AdaBoost) classifier to detect the human faces. The macroblocks (MB) corresponding to the ROIs are then coded with more bits, while the other regions are more compressed, so it's possible to have lower bit rates, while maintaining the same subjective visual quality if this technique wasn't used.

2.5.2 HEVC/H.265

HEVC/H.265 operation is based on the same principles of the H.264/AVC standard. This is the most recent video coding standard of the ITU-T Video Coding Experts Group and the ISO/IEC Moving Picture Experts Group and can provided up to 50% bit rate reduction relatively to prior standards like the H.264/AVC for the same visual quality [26]. Some work relative to this standard will be discussed next.

The work presented in [27] proposes a 4K x 2K H.265/HEVC video codec chip implementation in a 28nm CMOS process. This an interesting implementation that not only implements the H.265 coder/decoder, but also, supports 13 additional video standard formats such as MPEG-2/4 and H.264. The implementation supports 4096 x 2160 (4K) @ 30 fps encoding/decoding. The power consumption is 126.73 mW when processing 4K @ 30 fps video. The die area is 1.49 x 1.45 mm^2. This implementation is adequate for smart-phone applications due to it's low power requirements. Further details about the implementation are found on the same article.

In [28] is proposed an intra encoder capable of handling resolutions up to 2160p @ 30 fps. This architecture uses 1086k gates and on-chip 52 kB of memory. The chip was fabricated using 90nm lithography.

As discussed on chapter 2, motion estimation and compensation are computational complex operations that normally dominate the overall complexity of a codec but that can have a very significant impact on the compression ratio. In work [29] was proposed a VLSI implementation of the HEVC motion compensation that supports Ultra High Definition Television (UHDTV) resolutions, namely, supports 7680 x 4320 @ 60 fps resolutions in real-time with 10 bit per pixel component. The maximum clock speed is 400 Mhz. The proposed solution was fabricated in a 40nm process. The total gate count is 419k gates. Further details about the implementation architecture can be found in the paper.

2.5.3 MJPEG

Motion JPEG (MJPEG) compresses each video frame using the JPEG standard as if no relationship exists between successive frames, so only intra-frame redundancy is explored which diminishes the efficiency of the codec. Since no motion estimation and compensation techniques are used in MJPEG, it requires lower requirements in terms of CPU performance and power, making it adequate for embedded systems [30].

The paper [30] proposes a low complexity motion detection algorithm known as Zipfian estimation, to be applied to the MJPEG. It is shown in the same paper that this technique doubles the compression ratio, and, has an approximate performance in terms of quality and bit rate to the H.264 codec when used in Intra mode, but with lower computational complexity.

2.6 Distributed Video Coding

As was discussed in chapter 2, sometimes the encoder and decoder complexity are not equal (asymmetric compression). In "traditional" video coding algorithms such as H.264, the encoder complexity is normally greater than the decoder complexity, basically, because the encoder is responsible for motion compensation which is a computationally complex operation [20].

Some applications require the complexity to be shifted from the encoder side to the decoder side, as in the example referred in [20], where the example given is a wireless camera that uploads data to a base station. Since the camera is the video source, the camera must be the responsible for compressing the video, but, due to hardware limitations in terms of memory and computational power, the encoding operation should be computationally simpler than the decoding operation.

Distributed Video Coding (DVC) is a video coding approach that accomplishes the above requirements. It is based on two Information Theory theorems, namely, Slepian and Wolf's theorem and Wyner-Ziv theorem.

Several architectures have been proposed in the literature [31], adequate for VLSI implementation.Further information about the above architectures can be found on the same article.

2.7 Machine Learning in Video Coding

In this section it will be discussed the Machine Learning approach to video compression. Machine Learning techniques can be used to analyze the video frames in a statistical point of view and create opportunities to compress data. Several approaches have been proposed in [13]:

- Inpainting related approaches - The idea is to not transmit all image blocks saving bits that way. Given the spatial and temporal redundancy of video signals, at the decoder side that missing blocks can be reconstructed based on the neighbors.

- Dictionary learning related approaches - This approach is based on the fact that most image patches (regions) are constituted by some basic textures patterns. By constructing a dictionary of texture patterns, it is possible to construct an image block by using a linear combination of the basic textures, present on the dictionary. On [32] was proposed an online learning based codec (ODL). The dictionary gets richer in basic textures as the codec runs, thus, the more video frames it processes, the more efficient it gets. It is also shown that the rate distortion performance at low bit rates for intra-frame video is superior to the H.264. It is shown on [13] that the dictionary approach only works with low bit rates and the coder is computationally exigent.

- SIFT related approaches - It's based on the premise that an image can be seen as collection of objects, and it is possible to identity each one of the objects and describe them using SIFT descriptors. Then by knowing the objects that constitute a given image it is possible to reconstruct it by using similar objects that exist on a given database. The problem is that the reconstructed image can be very different from the original one, and the SIFT descriptors may require prohibitive amounts of data to have practical application. Although in [13] is shown a alternative approach to overcome the above problems, the encoding and decoding times limit the application of SIFT related approaches in practice.

Chapter 3

Display Stream Compression

In this chapter it will be shown how the DSC algorithm works from a “high level” point of view without entering into low level implementation details. This chapter starts by a quick overview of the DSC compression procedure and then each block that constitutes the DSC encoder will be further analyzed and discussed. Much of the information on this chapter is based on the VESA DSC Standard[3]. Although the discussion is focused on the encoder, most of the blocks are common to the encoder and decoder.

3.1 Overview

The DSC algorithm receives pixels from a given video source in raster scan order, which is from the left to the right and from the top to the bottom of the frame as illustrated in figure 3.1.

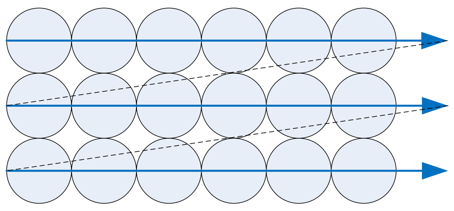


Figure 3.1: Raster scan order.

The DSC algorithm divides a video frame in equal slices. A slice is a rectangular portion of the frame, and its dimensions can be configured. In a given configuration the slice dimensions might be the same of the frame, while in another configuration the slice height (width) might be, for example, half of the frame height (width). Figure 3.2 shows a example where 4 slices per frame are used.

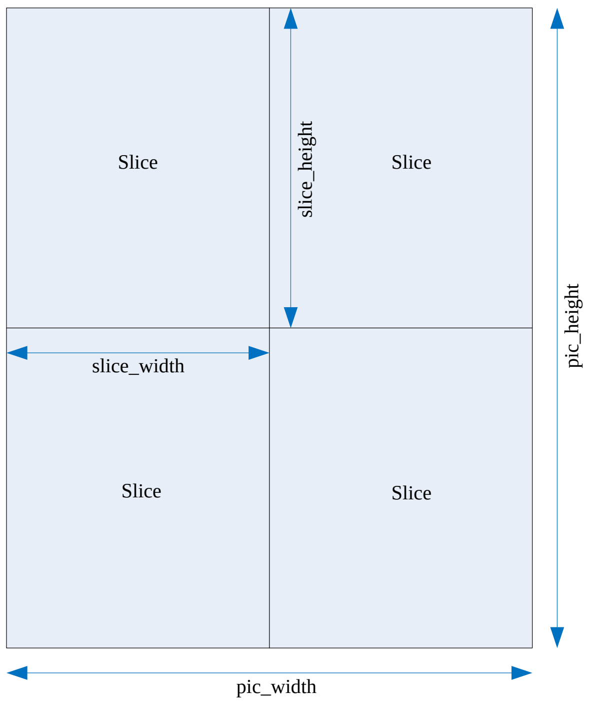


Figure 3.2: Frame partitioned in slices.The names `slice_width`, `slice_height`, `pic_width` and `pic_height` correspond to parameters defined in the Standard.

A slice can be encoded and decoded independently of another slice which can reduce the propagation of errors (and their impact in the perception of visual quality) and allow the parallelization of the algorithm, by having multiples encoder instances processing several slices in parallel. Relative to slice height, to reduce the impact of artifacts at the slice boundaries, extra bits are allocated to the first line of each slice. The problem is, the number of bits available to the other lines must be reduced so the total number of bits that encode a slice is equal to the desired bit rate times the number of pixels that constitute the slice, so, dividing a frame in several tiny slices prejudices the overall compression and perceived quality.

Relative to the slice width, using slices whose width is less than the frame width, offers the opportunity to have several encoder instances, each one responsible for updating the corresponding slice, which provides a way of processing video streams that require a clock frequency superior to the one the encoder can work. Consider for example a encoder capable of achieving a clock frequency of 100 MHz and a video format that requires a pixel frequency of 200 MHz. Using only one encoder, it is impossible to encode the video stream, but if a configuration that uses 2 slices per line is used with 2 encoder instances, then, each one of the encoders must only process half the pixels and it is now possible to successfully encode the video stream. More on this topic will be discussed in the chapter 5.

Besides providing a video stream to be encoded, first it is necessary to configure the encoder for proper operation. This task is performed by providing the encoder (and decoder) with what is known as the Picture Parameter Set (PPS). The PPS is a collection of parameters such as the slice width (`slice_width`) and the frame width (`pic_width`). For further details the reader is referred to the DSC Standard [3].

Figure 3.3 shows the DSC encoder architecture.

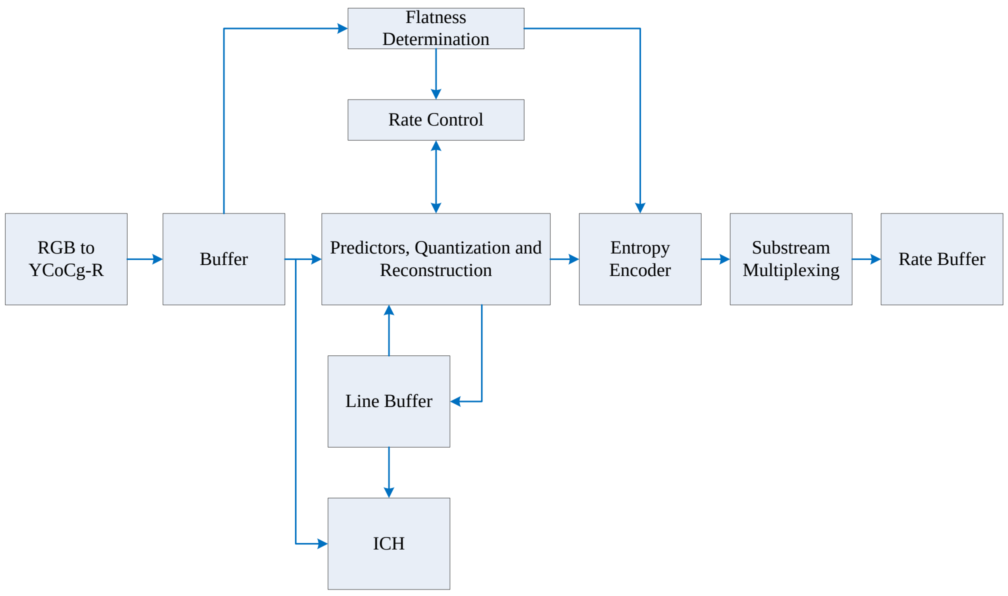


Figure 3.3: DSC Encoder Architecture.

The DSC reduces the redundancy of video signals and thus compresses the video stream, by using 2 techniques: Prediction coding (P-coding) and Indexed Color History (ICH) coding. The prediction techniques follow the theory described in chapter 2. There are 3 types of prediction: Modified Median-Adaptive Prediction (MMAP), Block Prediction (BP) and Midpoint Prediction (MPP). Figure 3.4 illustrates this discussion.

Each sample of a pixel is predicted by each one the predictors mentioned above, and the residual (difference between the original sample and the prediction result) is then quantized. For each group of 3 adjacent pixels in a line it is necessary to choose which prediction type to use. First, a decision is taken between using BP or MMAP. After that, it is necessary to choose between BP/MMAP and MPP. The criteria for these decisions will be described later on this chapter. The operations so far described are performed by the Prediction, Inverse Quantization and Reconstruction blocks.

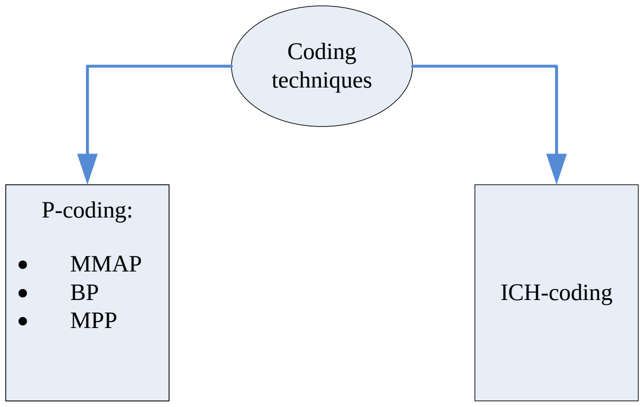
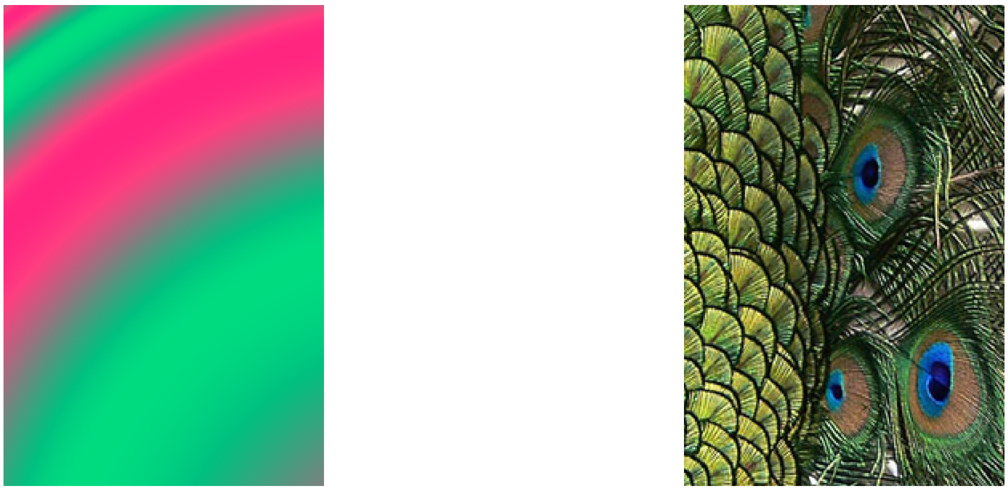


Figure 3.4: The coding techniques of the DSC algorithm.

The other technique used by DSC to reduce redundant information is the ICH coding. The basic idea consists of storing the most recent 32 reconstructed pixels in a memory, and then, a search is done over the 32 entries to find the best ones that represent each pixel within the group. Since the encoder and decoder maintain the same ICH state, it is only necessary to send the 5 bit index for each pixel, achieving that way, compression. There are 3 operations associated to ICH, namely, the update/storage of the reconstructed pixels in the shift register, the search for best fit, and the decision between prediction and ICH coding (P-mode vs ICH-mode). The ICH block is responsible for these operations, although in the implementation the selection between P-mode and ICH-mode, is performed by a dedicated block as described in chapter 4.

The Entropy Encoder is responsible for removing redundancies in the data representation and generates variable length codes (VLC). The entropy encoder organizes samples into units. One group is constituted by 3 pixels. A unit contains information of a component's group (so it contains 3 samples of a group, for example, 3 samples of luminance component (Y) plus a prefix. The prefix as indicated in the DSC 1.1 Standard, is a unary code (thermometer code) that indicates the non-negative difference between the size of the largest residual in the unit and the predicted size if P-coding is used, otherwise is a special code that indicates that ICH-coding is being use for the current group. The Entropy Encoder is also responsible for constructing the syntax elements, which correspond to the smallest portion of the bitstream, with a meaning in the context of the DSC operation.

One of the objectives of the DSC is to maximize subjective video quality (Visually lossless). As described on chapter 2, it is known that the Human Visual System (HVS) is more sensitive to low frequency content (in the sense of uniform regions) than high frequency content, so, in "flat" regions, the quantization parameter should diminish, otherwise, quality degradation may be perceived in that regions. The flatness determination block analyzes the "future" incoming pixels to detect the presence of flat pixels. That information is used in the Rate Control block to adjust the quantization parameter and is also sent to the decoder. Figure 3.5 shows two distinct images in terms of spatial frequency contents. Figure 3.5a shows an image that has several “flat” regions, while figure 3.5b shows an image rich in high frequency content due to the presence of many small details and sharp transitions between colors. The figure 3.5b besides generating less flatness adjustments than figure 3.5a, provides more opportunities to reduce redundant information.



(a) Image with significant “flat” regions (low spatial frequency contents).

(b) Image with high spatial frequency content.

Figure 3.5: Two images with distinct spatial frequency contents.

The Substream Multiplexer packs the information (syntax elements) generated by the Entropy Encoder, in words of fixed size, known as mux words. This process is performed for each component type. Finally, the Substream Multiplexer multiplexes each of the component streams[1] generated by the Entropy Encoder, to produce the bitstream that is sent to the decoder. Figure shows an example of the contents of the bitstream between the encoder and decoder. The process that determines the order in which the mux words are inserted is discussed in section 3.8.

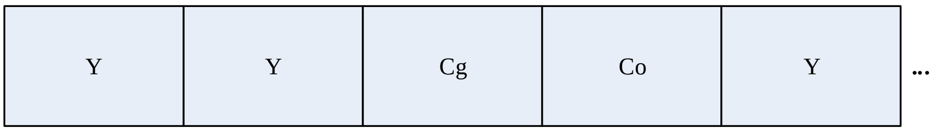


Figure 3.6: Possible construction of the bitstream between the encoder and decoder.

Since the number of bits to encode each unit varies from unit to unit, the data is first stored in the Rate Buffer to damp the bitrate variations. The data is removed from the Rate Buffer at a constant bit rate (CBR). Although variable bit rate (VBR) is described in the DSC Standard, it is not implemented at this time due to the HDMI transport layer limitations for VBR usage. To ensure that there is always data in the Rate Buffer to sustain the desired bitrate (i.e to avoid underflow) and to ensure that the Rate Buffer does not overflow, the DSC has a Rate Control block. The control is

[1]Each component stream is constituted by a series of syntax elements of that component.

made on the quantization parameter[2] (Qp). The idea is to maximize subjective quality so lower Qp values are privileged, but, when lower Qp values are used, the Rate Buffer will tend to overflow, so, eventually the Qp value is increased to diminish the buffer fullness. One can understand that frames that are difficult to encode (with many "flat" pixels) will have a tendency to make the buffer approach the overflow condition, while in oppose, easy frames to compress, will have a tendency to make the buffer approach the underflow condition. The Rate Control block decides the Qp to be used for each group. As discussed earlier, flatness information permits the Qp to drop quickly, so, the Rate Control block takes it into account before deciding the final Qp value to use in the next group.

In the following subsections it will be discussed how each one of the sub-blocks works. The approach chosen is to follow the video stream from the moment it enters the encoder and discuss each one of the operations performed on the video stream, until it is outputted. Before proceeding it is worthwhile to first discuss some parameters/operations that are used in the rest of this text:

- `bits_per_pixel` - PPS parameter that defines the number of bits used to represent each encoded pixel. The gain of compression is determined by this parameter, as shown on section 3.9.
- `masterQp` - Master quantization parameter. This parameter was introduced earlier with the generic name "quantization parameter". This parameter is not used directly in most of the blocks. First it is first mapped into two other quantization parameters that are different for luminance and chrominance components (`qlevelY` and `qlevelC` respectively).
- `mapQptoQlevel` - The operation of mapping the `masterQp` value into the quantization parameter `qlevelY` and `qlevelC`.

3.2 RGB to YCoCg-R

The DSC algorithm supports an input RGB or YCbCr video sequence with a component bit depth of 8,10 or 12 bits per component although the encoding and decoding process cannot be performed in the RGB color space. In that case, the first step consists of a color space conversion from RGB to YCoCg that is signaled by setting the `convert_rgb` PPS parameter to 1. If the input video color space is YCbCr then the `convert_rgb` should be cleared to 0 and the first RGB to YCoCg conversion block is bypassed.

3.3 Buffer

The Buffer block on figure 3.3 stores temporarily pixels from the RGB to YCoCg-R block. The reason for this is that the Flatness Determination block for processing a given pixel, requires information from "future" pixels. This can be accomplished by delaying the encoding process and store the color space converted pixels.

[2]The name used in the DSC Standard is masterQp and will be introduced shortly.

3.4 Predictors, Quantization and Reconstruction

3.4.1 Modified Median-Adaptive Prediction (MMAP)

The MMAP method generates a prediction using information from previously coded pixels to the left and above, of the current group being predicted. Figure 3.7 illustrates this concept and defines names for the pixels involved, which will be useful to understand the equations.

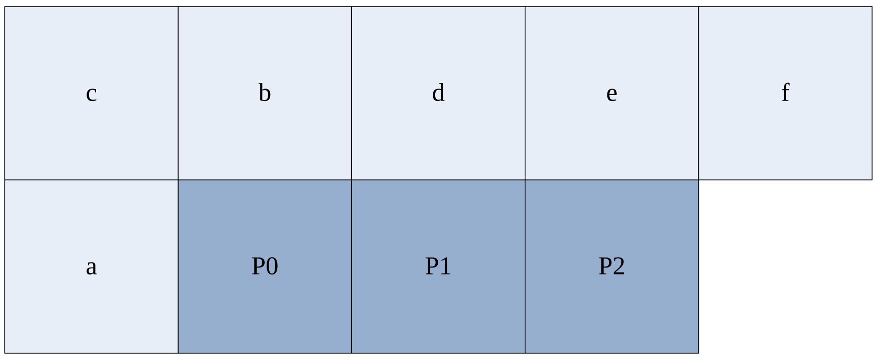


Figure 3.7: Pixels require to predict the current group. The letters “c”,“b”,“d”,“e” and “f” represent reconstructed pixels from the previous line. The letter “a” represents the rightmost reconstructed pixel of the previous group. “P0”,“P1” and “P2” symbolize the pixels of the current group that are being predicted.

The first step consists of applying a low-pass filter with coefficients [0.25 0.5 0.25] to each one of the pixels from the previous line (c,b,d and e). As an example, equations 3.1 and 3.2 show how the filter should be applied to pixel b and d respectively.

$$filt_b = \frac{c+2b+d+2}{4} \tag{3.1}$$

$$filt_d = \frac{b+2d+e+2}{4} \tag{3.2}$$

Once the pixels are filtered, they are blended with the original pixels as follows (example for pixel b):

$$diff_b = CLAMP\left(filt_b - b, -2^{qlevel-1}, 2^{qlevel-1}\right) \tag{3.3}$$

$$blend_b = b + diff_b \tag{3.4}$$

Where the clamp operation consists of saturating the first argument $(filt_b - b)$ between the second an the third argument, so, the equation saturates the difference to the binary range of

$qlevel$. For example, consider that $qlevel = 6$, then, if the first argument value is greater than 32, the clamp result is 32. The same reasoning applies to negative values.

Equations 3.5, 3.6 and 3.7 give the prediction value for each pixel of the group for lines that are the not the first line of each slice.

$$P0 = CLAMP\left(a + blend_b - blend_c, MIN(a, blend_b), MAX(a, blend_b)\right) \tag{3.5}$$

$$\begin{aligned} P1 =& CLAMP(a + blend_d - blend_c + R0, \\ & MIN(a, blend_b, blend_d), MAX(a, blend_b, blend_d)) \end{aligned} \tag{3.6}$$

$$\begin{aligned} P2 =& CLAMP(a + blend_e - blend_c + R0 + \\ & + R1, MIN(a, blend_b, blend_d, blend_e), MAX(a, blend_b, blend_d, blend_e)) \end{aligned} \tag{3.7}$$

The variables $R0$ and $R1$ represent the inverse quantized residuals that result from encoding the first and second pixel, respectively, of the current group.

Since slices can be independently encoded and decoded, when the first line of a slice is being processed, the pixels from the previous line are not available. In this case, equations 3.8, 3.9 and 3.10 should be used instead.

$$P0 = a \tag{3.8}$$

$$P1 = CLAMP\left(a + R0, 0, 2^{cpntBitDepth} - 1\right) \tag{3.9}$$

$$P2 = CLAMP\left(a + R0 + R1, 0, 2^{cpntBitDepth} - 1\right) \tag{3.10}$$

where $cpntBitDepth$ represents the bit depth of the component being processed.

3.4.2 Block Prediction (BP)

The BP predictor is simply a reconstructed pixel that is to the left of the current pixel being predicted in the same slice line. The pixel to use is determined by the BP vector that is the same for all pixels of a given group. The BP vetor is constituted by a single value in the range $[-3, -10]$, that indicates the displacement that should be applied to each pixel of the group. Figure 3.8 clarifies this concepts.

Figure 3.8 shows the reconstructed pixels of the current slice line and the current group being predicted. The negative numbers represent the offset value relative to the 3rd pixel of the current group. The figure also shows the prediction results for two possible BP vectors. For example, if BP vector is -5, then the prediction result for the first pixel of the current group is the pixel represented by -7, which corresponds to the 5th previous pixel of the current slice line. The same reasoning applies to the remaining pixels of the group.

In section 3.4.4 it will be described the algorithm to determine the BP vector value. Before leaving this section, it is important to mention that the BP predictor cannot be used when the horizontal position within the slice line is less than 10 because in that case, some vectors would address pixels outside of the current slice line.

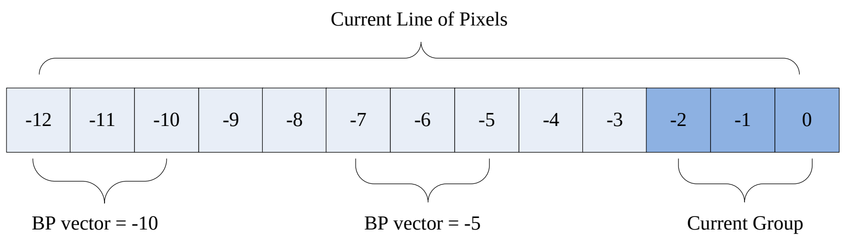


Figure 3.8: Block Prediction application example.

3.4.3 Midpoint Prediction (MPP)

The third prediction method available is the midpoint prediction. The prediction value obtained with this method is a value near half of the range of the component being predicted, hence the name midpoint.

The following equation determines the prediction value:

$$P = \frac{2^{cpntBitDepth}}{2} + prevRecon \ \% \ 2^{qlevel} \tag{3.11}$$

where $cpntBitDepth$ represents the bit depth of the component being predicted and $prevRecon$ is the rightmost reconstructed pixel from the previous group, which was not necessarily obtained by the MPP method. The $qlevel$ parameter represents the current quantization level being used for the component being processed. Notice that for the first group of each slice, $prevRecon$ is not available, so, it should be set to 0.

3.4.4 BP vs MMAP decision

For each group it is necessary to select a predictor to use. First a decision is taken between using BP or MMAP, then in section 3.4.5 it will be described how the selection between the result of BP/MMAP decision and MPP is performed.

The BP/MMAP decision is made on the previous line[3], which is useful from the implementation point of view, because it provides sufficient time to perform the required operations which are time consuming and would be difficult to be done in a group time. The are two possible outcomes

[3]When the slice line x is being processed, the results of the decision process apply to line $x+1$.

of this decision process. If BP is selected, then the algorithm provides the BP vector that should be used when processing the group on the next line of the slice, otherwise MMAP is selected.

The decision algorithm involves determining the Sum of Absolute differences (SAD) for 3 groups of pixels. Figure 3.9 shows the groups involved in the decision process. To determine the SAD for each group, it is necessary to compare each pixel of the group, with each one of the 10 pixels to the left of the current pixel being processed. This is exemplified in figure 3.9 for the first pixel of the current group (group C). The pixels involved in this operation are shown with orange color. After applying some mathematical operations, a decision is generated. If BP is selected, then the selected vector is the one with the lowest SAD. Further details are found in the Standard.

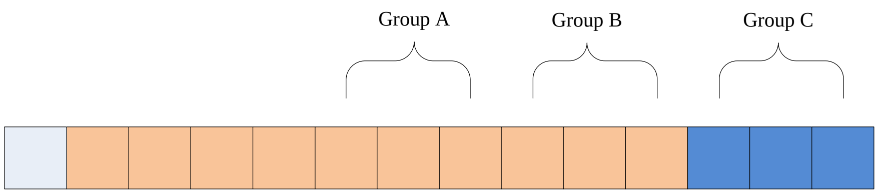


Figure 3.9: The letters A,B and C represent the groups of pixels used to perform the decision between MMAP/BP for group C.

From this brief discussion, the reader can confirm that the decision process is indeed computationally heavy. Before leaving this section, it is important from a theoretical point of view to emphasize that BP is not selected for "flat" regions of a frame. This behavior is expected, considering the fact that the BP predicts a pixel by choosing a previous pixel as the prediction value and the Human Visual System (HVS) easily detects distinct pixels in an uniform region.

3.4.5 BP/MMAP vs MPP decision

Once a BP/MMAP decision is performed, it is necessary to decide whether use BP/MMAP or MPP for each component type. The algorithm is as follows:

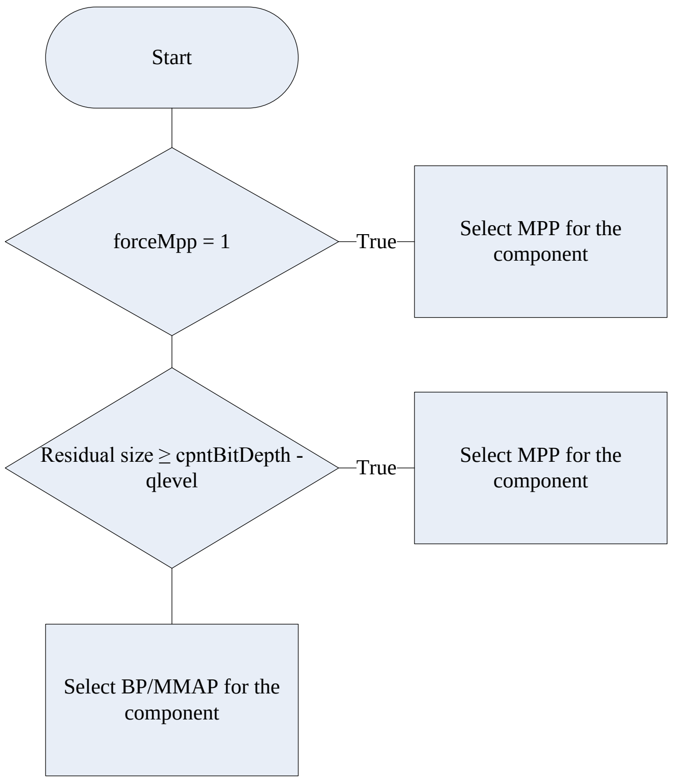

Start
forceMpp = 1
True
Select MPP for the component
Residual size ≥ cpntBitDepth - qlevel
True
Select MPP for the component
Select BP/MMAP for the component

Figure 3.10: BP/MMAP vs MPP decision.

The `forceMpp` signal shown on figure 3.10 is generated by the Rate Control block and enforces a minimum data rate to avoid an underflow of the Rate Buffer. This happens because the resulting MPP residuals' size is always considered to be $cpntBitDepth - qlevel$ which is for a given component, the maximum residual size possible for the quantization level currently in use.

3.4.6 Quantization

As described is chapter 2, the compression is effectively obtained by the quantization operation. The quantizer used by the DSC algorithm is of the uniform type and 2^{qlevel} output values are possible. First a rounding value is summed to the residual being quantized to reduce the power of the quantization error:

$$ROUND = \begin{cases} 2^{qlevel-1} - 1 & , qlevel > 0 \\ 0 & , qlevel = 0 \end{cases} \tag{3.12}$$

$$E' = \begin{cases} ROUND - E & , E < 0 \\ E + ROUND & , E \geq 0 \end{cases} \tag{3.13}$$

the quantization is then performed with truncation by a divisor that is a power of 2:

$$QE = \begin{cases} -E' >> qlevel & ,E < 0 \\ E' >> qlevel & ,E \geq 0 \end{cases} \tag{3.14}$$

where “>>” is the right shift operator.

3.4.7 Inverse Quantization and Reconstruction

The Inverse Quantization and Reconstruction block is responsible, as the name suggests, for reconstructing the original pixels. The inverse quantization step may seem a bit odd, because the quantization operation is irreversible. Inverse quantization consists of recovering the magnitude of the quantized residual as follows:

$$Q^{-1} = QE \times 2^{qlevel} \tag{3.15}$$

where QE represents the quantized residual. The reconstructed pixel is obtained by equation 3.16:

$$reconsPixel = CLAMP(predPixel + Q^{-1}\ ,\ 0\ ,\ maxval) \tag{3.16}$$

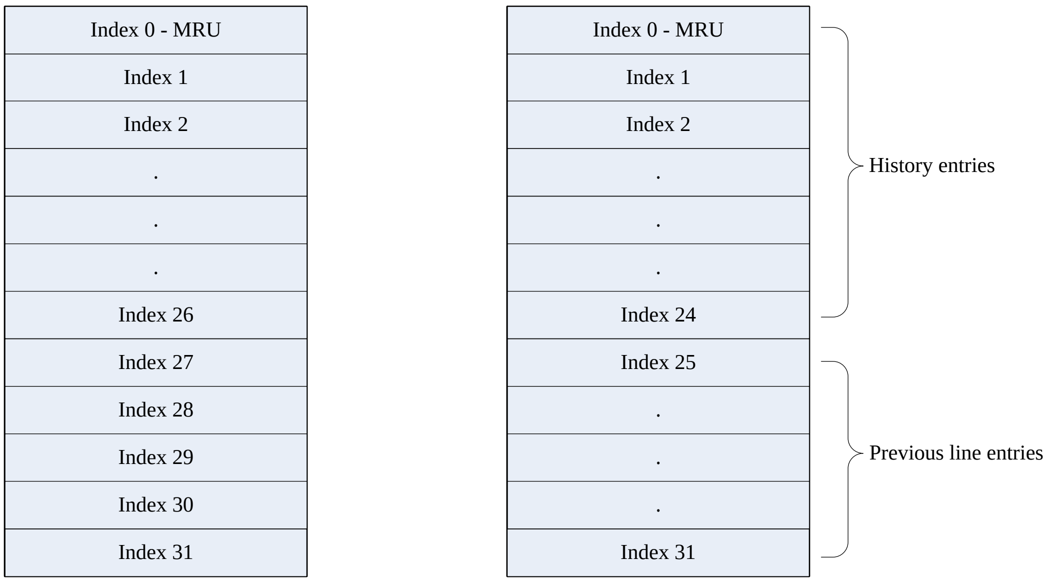
where $maxval$ is the maximum value for each component of the pixel and $predPixel$ is the predicted pixel, that originated the quantized residual that is now being inverse quantized. The clamp operation ensures that the reconstructed pixel value is always non-negative and that each component value is less than or equal, to the maximum value possible for the bit depth being used.

3.5 Indexed Color History (ICH)

Section 3.4 described the prediction techniques that generate a prediction value that is used to compute a prediction residual which is then quantized, and sent to the entropy encoder (unless ICH is selected for the current group). In this section the operation of the Indexed Color History, which is an alternative technique to achieve video compression, will be described.

The ICH is a memory with 32 entries that store reconstructed pixels that were previously generated by prediction techniques. The principle of operation is based on the premise that for each group being encoded, there might exist “good enough” pixels, according to a given criteria describe later in this section, to represent the pixels being encoded. This is true because as described in chapter 2, video signals exhibit spacial redundancy. Since the encoder and decoder maintain the same ICH state (because reconstructed pixels are used) then only the index of the selected pixels must be sent, which means that is possible to code a group with 3×5-bit indexes plus 1 bit prefix (explained on section 3.7), achieving that way the minimum bit rate possible, for the DSC, of 5.33 bits/pixel.

Figure 3.11 shows the structure of the ICH memory. Entries in the range from 0 to 24 (inclusive) are history entries (i.e, entries that store previously reconstructed pixels from the current slice line) and the remaining 7 entries correspond to reconstructed pixels from the previous line. The exception to this organization, is when the first line of each slice is being processed, because, the previous line of reconstructed pixels is not available. In that case, all 32 entries correspond to history entries. The most recently used value is stored on entry 0.



(a) First line of a slice. (b) Remaining lines of a slice.

Figure 3.11: ICH memory structure.

As groups are coded, the contents of the ICH memory must be updated. The update process is only performed on group times, which means, that all pixels of a group "see" the same ICH state. It is necessary to distinguish the following two situations: the update of history entries, and the update of entries reserved to storage of pixels from the previous line.

Starting by the history update, two situations must be considered: when the group is ICH coded and when it is P-mode coded. If the group is P-mode coded, then three pixels are stored into the ICH memory, as the most-recently used entries (more precisely, the rightmost pixel of the group is stored in index 0). The three least-recently used pixels are lost when the other entries shift down three positions. This process is illustrated on figure 3.12.

When the group is ICH coded, the update process consists of reorganizing the existing values. The pixel corresponding to the rightmost indice is moved to location at address 0, the second one to address 1 and the third one to address 2. The exception to this rule is when the three indices are not unique. In those situations, only the last occurrence of the replicated indice should be considered. As an example, consider the situation where the three indices used are (4,4,6), then, the location at addresses 0 and 1 should store the contents of indice 6 and 4 respectively. To fill the gap created by moving the contents of the addressed entries, some entries might have to shift down. For a given entry if the index is less than the three indices, it should shift down (towards

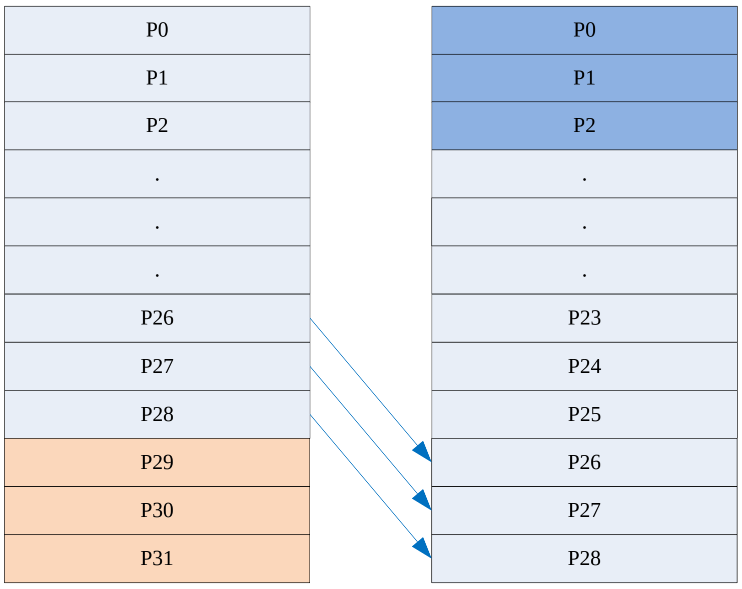


Figure 3.12: ICH history entries update (P-mode group).

the least-recently used entry) 3 positions, if it is less than two of the indices, then it should shift down 2 positions and so on. Figure 3.13 illustrates this concept.

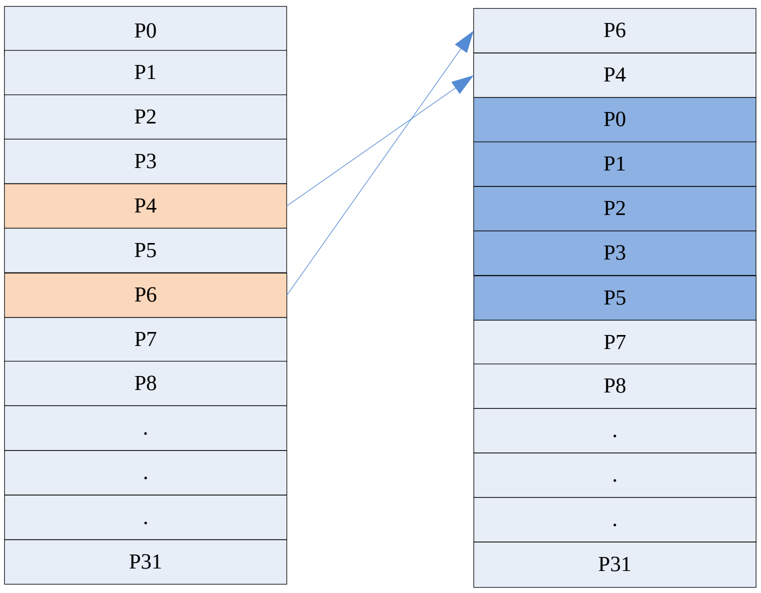


Figure 3.13: ICH history entries update (ICH-group).

As discussed earlier, the last seven entries of the ICH memory store pixels from the previous line. The pixels that are stored are the two adjacent pixels to left of the current group being processed (in the previous line), the three pixels above, and the two adjacent pixels to the right. This mode of operation can be thought as a sliding window of dimension three, that slides 3 pixels as each group is coded. Figure 3.14 illustrates this concept for several groups of a slice line. Each group is represented with a different color. The matching window is shown with the same color. The figure also shows how special cases like the first and the last group are handled, which consists of "clamping" the sliding window to the extremes of the slice line, as shown by light blue and the gray dashed lines respectively.

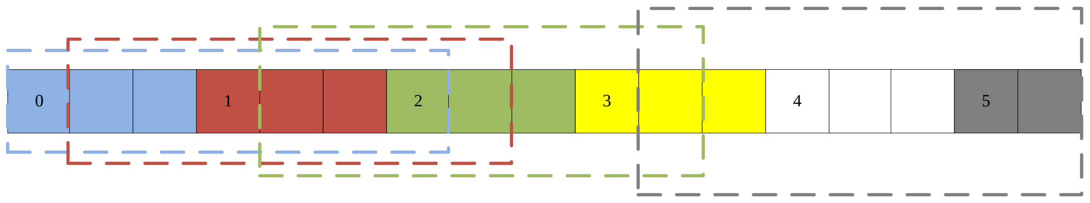


Figure 3.14: ICH last line pixels update.

The problem of determining the best candidate index for each pixel of a given group will now be discussed. The best candidate index for a given pixel, is the one that results in the smallest weighted sum of absolute differences (SAD) value:

$$weightedSAD = 2 \times ABS(orig_Y - ich_Y) + ABS(orig_{Co} - ich_{Co}) + ABS(orig_{Cg} - ich_{Cg}) \quad (3.17)$$

where $orig_Y$, $orig_{Co}$, $orig_{Cg}$ represent the Y, Co and Cg components respectively of the original(color space converted) pixel, and ich_Y, ich_{Co}, ich_{Cg} represent the same components but from the pixel stored in the ICH memory. As the reader can see, the luma (Y) component has more importance when determining a good candidate for the pixel being processed. This is because the human visual system has more acuity for the luminance component, than for chrominance components, as described in chapter 2. The search is performed over the 32 ICH entries. If two indices have the same weighted SAD value, then the smallest indice is selected.

For each group being coded it is necessary to select between predictive mode (P-mode) and indexed color history mode (ICH-mode). The algorithm is as follows. First it is necessary to determine if there is at least one entry in the ICH memory for each pixel of the group, whose coding error for each sample, does not exceed a threshold. Equation 3.18 defines how the coding error is determined:

$$codingError = ABS(origPixel - ichPixel) \quad (3.18)$$

Equation 3.19 together with 3.20 define how the threshold is determined:

$$modQp = MIN(masterQp + 2\ ,\ 15 + 2 \times (bitsPerComponent - 8)) \quad (3.19)$$

$$threshold = 2^{MapQpToQlevel(modQp)-1} \tag{3.20}$$

where $MapQpToQlevel$ maps the master quantization parameter ($masterQp$) into the $qlevel$ parameter for luminance and chrominance components, so, the threshold value is different for each type of component.

After determining if the previous test was successful, it is necessary to determine the maximum component-wise errors for each type of coding (ICH-mode and P-mode) and component. Equations 3.21 and 3.22 exemplify this computation for the luma component. The other components follow the same reasoning.

$$maxYErrIch = MaxOverPixelsInGroup(ABS(origY - ichY) >> shift) \tag{3.21}$$

$$maxYErrPMode = MaxOverPixelsInGroup(ABS(origY - reconsY) >> shift) \tag{3.22}$$

The value of $shift$ is equal to $bitsPerComponent - 8$. The variable $origY$ represents the luminance sample of the original pixel. The variables $ichY$ and $reconsY$ represent the reconstructed pixels generated by ICH-coding and P-coding respectively.

Before making the final decision, the number of bits required to code the group in ICH-mode and P-mode must be determined. Section 3.7 explains how it is performed. The final decision is ICH mode if:

$$\begin{aligned} &(logErrIchMode \leq logErrPMode)\ \&\& \\ &(bitsIchMode + 4 \times logErrIchMode < bitsPMode + 4 \times logErrPMode) \end{aligned} \tag{3.23}$$

where $logErrIchMode$ can be determined by:

$$\begin{aligned} logErrIchMode =& 2 \times ceil_{log2}(maxYErrIchMode) + ceil_{log2}(maxCoErrIchMode) + \\ & + ceil_{log2}(maxCgErrIchMode) \end{aligned} \tag{3.24}$$

The same reasoning applies to P-mode ($logErrPMode$). The function $ceil_{log2}$ is defined as:

$$ceil_{log2}(x) = ceil(log_2(x+1)) \tag{3.25}$$

which basically determines the minimum number of bits necessary to represent x.

3.6 Line Buffer

The line buffer block stores the previous line pixels that are used by the MMAP and ICH blocks. The DSC offers the option of storing the pixels with a smaller bit depth to minimize memory requirements. In that case, before storing the pixels, it is necessary to quantize them, following a process similar to the one described in section 3.4.6. When reading a pixel from the line buffer, it is necessary to recover the order of magnitude of each sample, by shifting left a number of bits equal to the one that was used to shift right in the quantization process.

3.7 Entropy Encoder

The entropy encoder is the last block that effectively compresses data. The remaining blocks only manipulate the bitstream that is sent to the decoder and generate control signals that affect the operation of the blocks so far described.

The Entropy Encoder reduces the statistical redundancy existing in the representation of video signals by using variable length codes (VLC) to represent the residuals (or ICH indices). Before discussing how this block works let's review the concept of unit. The Entropy Encoder organizes the samples of a group into units. A unit consists of three samples of a given component of a group, plus a prefix, so, for each group there are three units. The Y unit, the Co unit and the Cg unit. Figure 3.15 illustrates this concept. If a P-coded group is being coded, then, the three residuals of the unit are represented with the same number of bits, in 2's complement format. On the contrary, for ICH-coded groups, the residual portion of the unit is replaced by the selected ICH indexes.

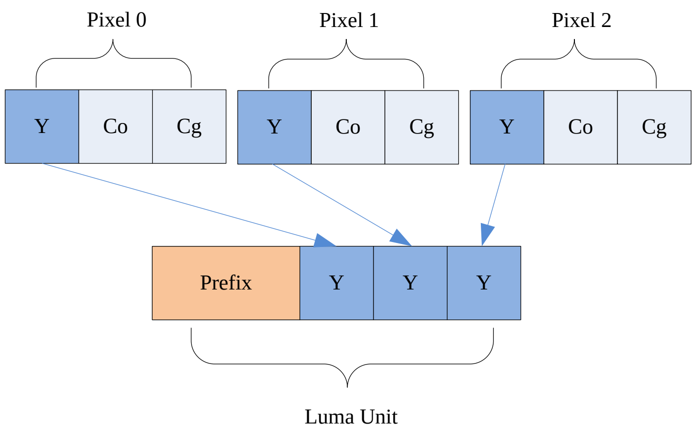


Figure 3.15: Entropy Encoder Unit for P-coded groups.

The determination of the prefix value is a rather complex process that is not relevant for the discussion in this chapter. The reader is referred to the Standard for further details. Besides determining the prefix and suffix of each unit, the Entropy Encoder is also responsible for signaling flatness information in the luma (Y) stream. As discussed in the introduction of this chapter, the flatness determination block analyzes the next pixels to detect if they are flat. If indeed they are flat, then the quantization parameter should drop, to not prejudice the perceived visual quality. This information is also inserted in the luma stream, along with, to which group the flatness information applies, otherwise the decoder cannot correctly modify the quantization parameter when decoding the group.

3.8 Substream Multiplexing

Section 3.7 described how each syntax element that constitutes the bitstream is created. In this section it will be described the process of joining together the streams of each component (Y,Co and Cg streams). Figure 3.16 shows the structure of the Substream multiplexer block. As the reader can see, there are three FIFOs, one for each component type, that hold the syntax elements generated by the Entropy Encoder block. There is also a decoder model, which is an idealization of a decoder and a multiplexer.

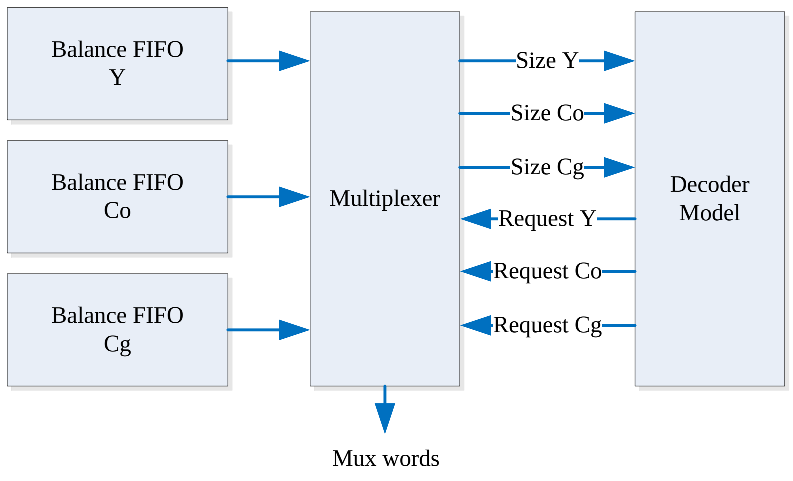


Figure 3.16: Substream multiplexer block diagram.

The substream multiplexer works with words of bits, each one 48 or 64 bits wide. The size of each mux word depends in the `bits_per_component` PPS parameter. For a configuration of 8 bits per component the mux word size is 48 bits, otherwise it is 64. Before activating the decoder model, each balance FIFO accumulates the information of certain number of groups to avoid underflowing the balance FIFOs. It is stated on [3] that this number is $muxWordSize + maxSeSize - 1$, where $maxSeSize$ is the maximum possible size for a syntax element, which is $4 \times bitsPerComponent + 4$.

After the FIFOs are loaded with the required amount of data, the decoder model is started. Before proceeding it is important to briefly discuss the structure of the decoder model. Figure 3.17 shows the structure of the decoder model. As expected, it is the counterpart of the encoder substream multiplexer block. From a practical point of view, the substream processor (SSP) can be considered as a counter. For each clock cycle the demultiplexer separates the incoming bits into each substream that are then stored in the SSP. Each group time, one syntax element of each type is processed, and the equivalent number of bits is subtracted from the SSP fullness.

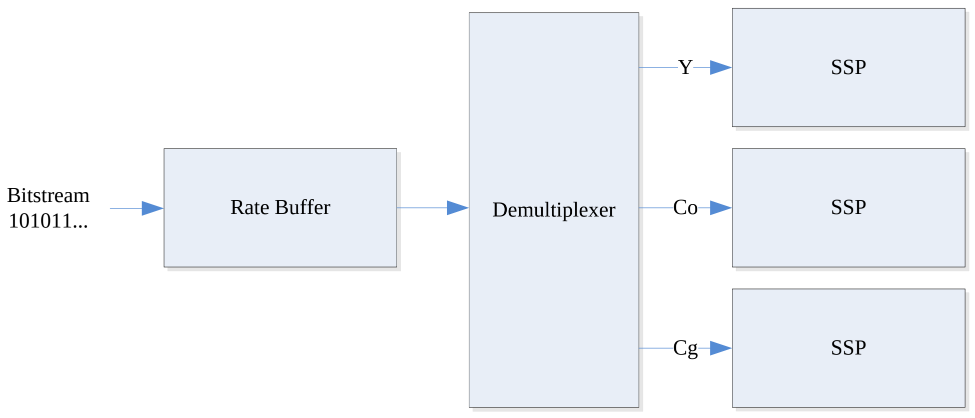


Figure 3.17: Decoder model.

Returning to the operation of the substream multiplexer and Decoder Model, each group time, the SSP fulness of each component type, diminishes by the size of the syntax element of the group being processed. If there is the risk of a given SSP underflowing when processing the next group, then a request signal is sent to the multiplexer, a mux word is removed from the corresponding Balance FIFO and sent to the Rate Buffer. Next, the SSP fullness increases by the size of a mux word. The SSP might underflow when the fullness level is less than $4 \times bitsPerComponent + 4$, which is the maximum size of a syntax element, for a given bits per component value. If multiple request signals exist, then the order in which the mux words should be outputted is:

- Y mux word
- Co mux word
- Cg mux word

3.9 Rate Buffer

The Rate Buffer block holds the mux words generated by the substream multiplexer block to allow streaming the encoded data with a constant bit rate. To avoid underflowing the Rate Buffer, there is a period of time at the beginning of a slice, where bits enter but do not leave the buffer. This period of time is known as the initial transmission delay and is configurable through the `init_xmit_delay` PPS parameter. The DSC Standard [3] defines several possible values for this parameter, which takes into account empirical results. The number of bits that leave the Rate Buffer at each pixel time, is configured through the `bits_per_pixel` PPS parameter which represents the number of bits that code each pixel at the encoder output. Possible values are in the range from 6 bits per pixel (bpp) to 36 bpp in steps of $1/16$ bpp, where 36 bits per pixel is only achievable if at the input of the encoder the pixels are coded with 36 bits (12 bits per component). The compression ratio can be expressed as:

$$CompressionRatio = \frac{bitsPerPixel}{3 \times componentBitDepth} \tag{3.26}$$

3.10 Flatness Determination

A set of four consecutive groups of 3 adjacent pixels in a line is known as a supergroup. Before encoding the first group of each supergroup, a search is performed over the entire supergroup to determine if any group is flat. There are two types of flatness: “somewhat flat” and “very flat”. To decide the flatness type (or not flat at all) two tests are applied to each group of the supergroup.

The following pseudo-code resumes the first flatness test:

```
max_sample = -MAXINT
min_sample = MAXINT

for each pixel component C
  for each pixel i involved in the test
    sample = original_pixel(C,i)

    max_sample = MAX(max_sample, sample)
    min_sample = MIN(min_sample, sample)
  }
  if( (max_sample - min_sample) > MAX(flatness_thresh, 2^flatQlevel(C)) )
    //somewhat flat test fails
  if( (max_sample - min_sample) > flatness_thresh )
    //Very flat test fails
}
if(very flat test passes)
  //Final decision is very flat
else if(somewhat flat test passes)
  //Final decision is somewhat flat
else
  //Perform Flatness Test 2
```

The flatness threshold is given by $2 << (bitsPerComponent - 8)$. The $flatQlevel$ value is different for each component type and can be determined as:

$$flatQlevel = MapQpToQlevel(MAX(0, masterQp - 4)) \tag{3.27}$$

where *MapQpToQlevel* maps the modified master quantization parameter into quantization parameters, usable for the luminance and chrominance components. More about this mapping will be described in section 3.11.

If the first test fails, then a second test is performed. The algorithm is essentially the same, the only difference is that the search is performed over six pixels instead of four. Figure 3.18 shows the pixels involved in both flatness tests for each group.

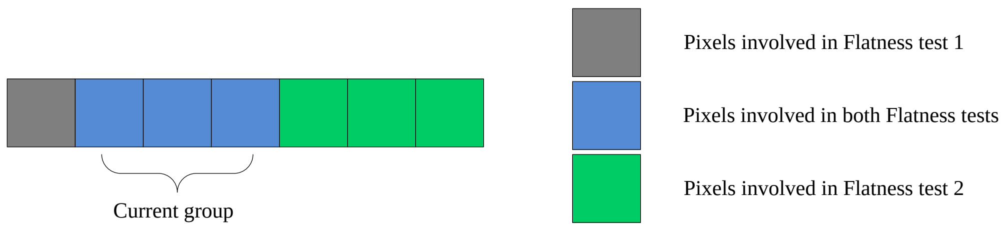

Current group
Pixels involved in Flatness test 1
Pixels involved in both Flatness tests
Pixels involved in Flatness test 2

Figure 3.18: Pixes used in the flatness tests.

If a group passes the flatness test, then the flatness type and the group offset within the supergroup (i.e. which group of the supergroup) is signaled in the luma syntax element as described in section 3.7. A QP adjustment is also performed to the group that is signaled as flat. Further details can be found on [3].

3.11 Rate Control

In this section it will be briefly discussed the operation of the Rate Control block. The main purpose of the Rate Control block is to generate the master quantization parameter (*masterQp*) to be used for each group taking into account that the subjective picture quality should be maximized and the rate buffer must not underflow or overflow. It is important to emphasize that the masterQP value used for a given group is not signaled in the bitstream, so, the decoder must mimick the behavior of the encoder RC algorithm. The sub-blocks that constitute the Rate Control top block are shown on figure 3.19.

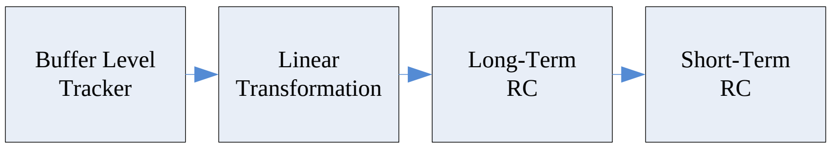


Figure 3.19: Rate Control sub-blocks.

The RC algorithm operates around an idealized rate buffer, that assumes an idealized encoder that can process a pixel per pixel time without delay. The implication of this assumption is that the buffer fullness of this idealized rate buffer, does not correspond to the fullness level of a real rate buffer because the delay introduced by the Substream Multiplexer block is ignored. The advantage of this idealized rate buffer is that simplifies the RC algorithm, while guaranteeing that the system works in real-time.

The Buffer Level Tracker block for each group time, accumulates the number of bits that code the group (information that is generated by the Entropy Encoder block) and subtracts the number of bits that leave the rate buffer which is a configurable PPS parameter (`bits_per_pixel`). The resulting value is called $bufferFullness$.

The Linear Transform block applies a linear transformation described by equation 3.28. The objectives of this transformation can be found on [3] and are replicated here:

- Maintain the perceived visual quality constant during the initial transmission delay period
- Ensure that the first line of each slice is coded using more bits (to reduce the effect of blocking artifacts in the edges of the slice)
- Ensure that the slice bit budget constraint is met (the number of bits that code a slice, must be equal to the number of pixels in the slice times the number of bits that are used to represent each encoded pixel (`bits_per_pixel`) in constant bit rate mode.

$$rcFullness = rcFormScale \times \frac{bufferFullness + rcFormOffset}{8} \tag{3.28}$$

The scale factor and the offset value result from a complex superposition of conditions that can be found on [3].

The next step consists of performing the long-term RC algorithm. The long-term RC block compares the $rcFullness$ value, obtained from the linear transformation block and compares it to a set of thresholds to determine in which range it fits. For each range it is defined three parameters: `range_min_qp`, `range_max_qp` and `range_bpg_offset` which are respectively: the minimum and maximum QP values and the target bits per group offset.

The short-term RC receives information from the long-term RC and entropy encoder to determine the short-term quantization parameter (`stQp`). Further details can be found on [3]. The `stQp` might be further modified if there is flatness information for the group. The resulting value is the `masterQp` value.

Chapter 4

DSC Implementation

In the previous chapter it was discussed how the DSC algorithm works from a high-level point of view. This permitted the reader to gain some knowledge on the operation of the DSC algorithm so it can now understand the implementation details to be presented in this chapter. It will be shown how the main operations and algorithms discussed on [3] are mapped to hardware modules. For each module it will be discussed some relevant details of implementation. Timing relationships will also be examined. This will give the reader further insight about how the operations of the different blocks are orchestrated together.

4.1 Introduction

Figure 4.1 shows the architecture of the implementation of the DSC Encoder. As the reader can confirm, it closely follows the one shown on figure 3.3 with some differences. Besides some rewired connections between blocks, the decision between Predictive coding and ICH coding is performed by a dedicated block.

4.1.1 High-level Timing Diagram

Figure 4.2 shows a timing diagram that indicates when each one of the blocks should operate and when the results are ready. The Rate Buffer and the delay inserted by the Buffer block are ignored without prejudice to the present discussion. As the reader can observe, the two coding techniques work in “parallel” (i.e, they are independent of each other) and at the end of the third clock cycle the results are ready. The decision between which coding technique to use for the current group is performed within the same clock cycle, so the fourth pixel has the required information to be processed.

The Entropy Encoder works on the fourth clock cycle. In the 5th clock cycle, the Rate Control operations are started and require 3 clock cycles to be completed (excluding the Short-Term RC). A distinction was made between the operation of the Short-Term RC and the remaining Rate Control blocks to emphasize the idea that the Short-Term RC block uses the results of the Long-Term RC for the previous group. A new quantization parameter is available in the 7th clock cycle.

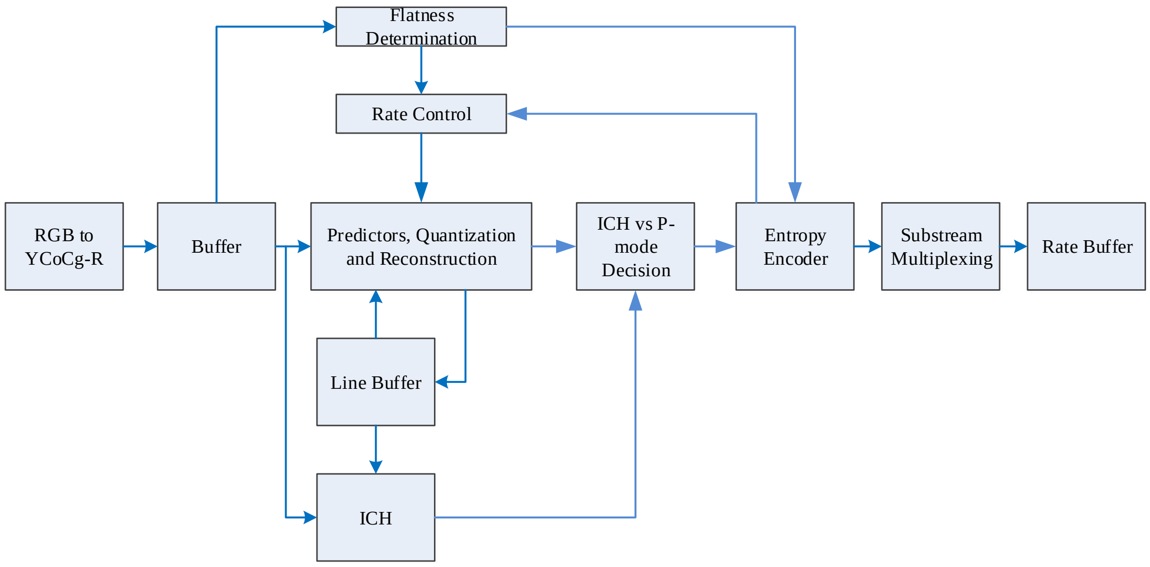


Figure 4.1: Architecture of the DSC Encoder block implementation.

The Flatness Determination block performs its operations in the last pixel of every four groups. The outputs are then used by the Entropy Encoder and the Rate Control blocks.

The Substream Multiplexer receives in the 5th clock cycle the syntax elements generated by the Entropy Encoder block. After a certain delay period, (to fill the Balance FIFOs with sufficient information to avoid an underflow), each group time, 0 to 3 mux words are sent to the Rate Buffer. The Rate Buffer after the initial transmit delay period, outputs the bitstream.

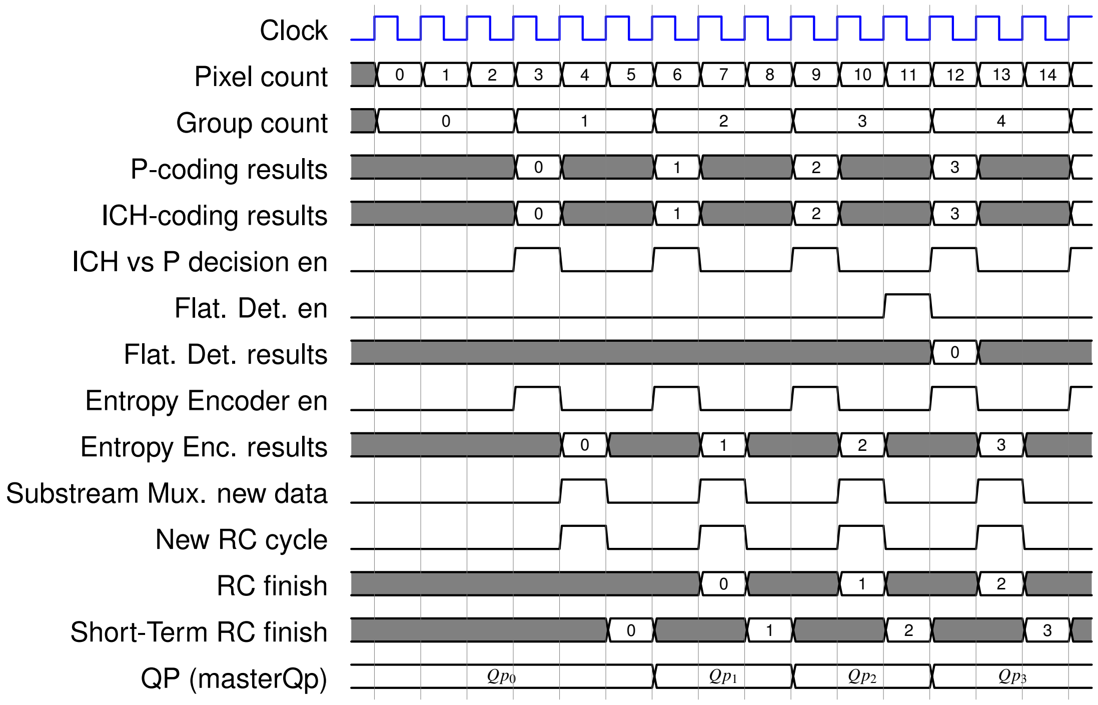


Figure 4.2: DSC Encoder timing diagram.

In the following sections, the topics briefly discussed above, and other details of implementation will be further discussed.

4.2 RGB to YCoCg-R

As discussed on chapter 3, this block performs the color space conversion to YCoCg if the input video signal is in the RGB color space. This block is very simple in terms of implementation. It is a direct implementation of the equations described on [3]:

$$cscCo = R - B \tag{4.1}$$

$$t = B + cscCo >> 1 \tag{4.2}$$

$$cscCg = G - t \tag{4.3}$$

$$Y = t + (cscCg >> 1) \tag{4.4}$$

This block is implemented as a combinational block with outputs registered, thus introducing a latency of one clock cycle.

4.3 Buffer

The Buffer block shown on figure 3.3 is not implemented as a memory block but as a shift-register. Instead, it is implemented as a shift-register with 19 entries, each one 38 bits wide. Each entry is 38 bit wide because because if the input video uses 12 bit per component, and a color space conversion from RGB to YCoCg is performed, then, by equation 4.1 and 4.3, the reader can observe that the chrominance components require an extra bit of dynamic range, compared to the luminance component. This implies that each pixel requires in the worst case scenario $12+13+13=38$ bits to be represented.

The reason for storing 19 pixels, is to have available, the required pixels to perform the flatness determination tests. Further details can be found in section 4.12.

4.4 Predictors, Quantization and Reconstruction

Figure 4.3 shows the architecture of the Predictive coding block. As the reader can observe, for each predictor instance, there is a dedicated quantizer block and a block that reconstructs the pixel ("Inverse Quantization and Reconstruction" block). This is required because for each group, the decision of which predictor to use, for each component, requires information from the quantized residuals, generated by each prediction method. Taking into account the number of clock cycles available (three), and the amount of information to process, it is easily concluded that a hardware sharing approach is not possible.

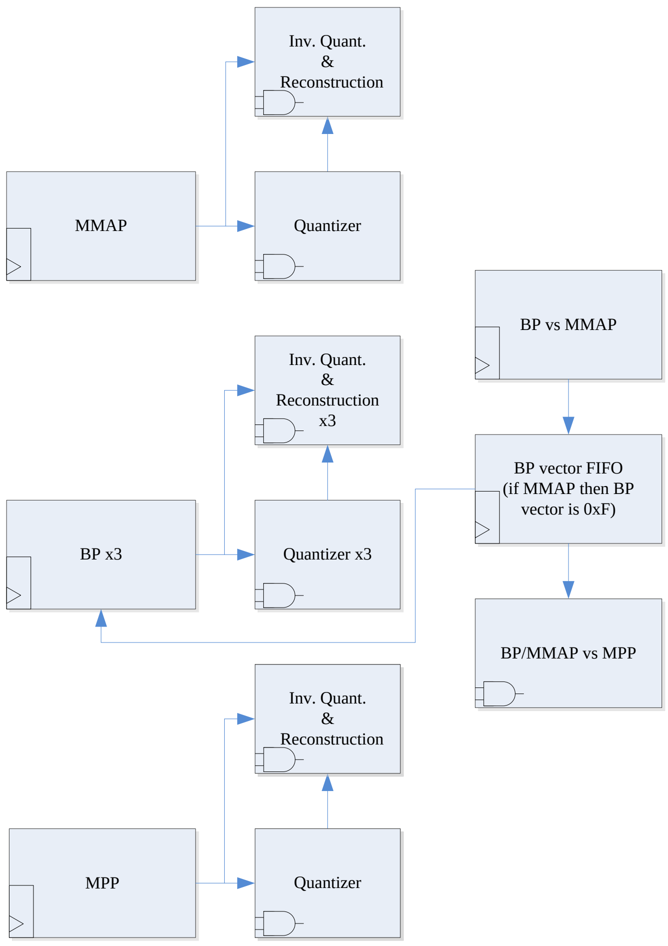
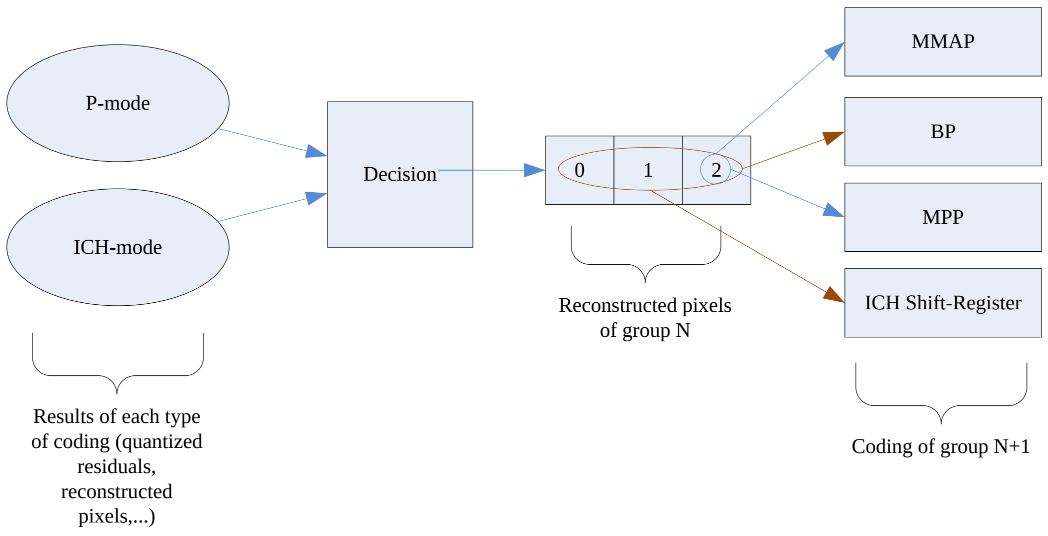


Figure 4.3: Predictive coding top block.

At each clock cycle, the MMAP and MPP blocks generate a prediction value, the corresponding prediction error is computed and quantized and then this value is registed. Besides the quantized residuals, the respective size and information used for taking decisions about which predictor to use, are also computed and registed. The exception to this rule is when the third pixel of the group is being processed. In that case, the values are not registed (only a comparison is made with the results of previous pixels) because the decision process (both the decision between the predictors, and the decision between ICH-coding and P-coding) would be postponed to the 4th clock cycle. To process the 4th pixel (first pixel of the next group), information of the previous group is required, namely, the reconstructed pixels of the group. Figure 4.4 illustrates this concept.



P-mode
ICH-mode
Results of each type of coding (quantized residuals, reconstructed pixels,...)
Decision
0
1
2
Reconstructed pixels of group N
MMAP
BP
MPP
ICH Shift-Register
Coding of group N+1

Figure 4.4: Data dependencies between groups. The figure shows the required pixels from the previous group, for each prediction method.

The BP method is governed by the same principles. The difference is that the three pixels are processed in parallel. The reason for this implementation is that three pixels should be processed using the same state (i.e, the same line of reconstructed pixels). This operation of storing the current line of reconstructed pixels, consumes 1 clock cycle, so, processing one pixel in each clock cycle is not an option. This implementation has also other benefits related to the critical path optimization. This topic is discussed in chapter 5.

In the following subsections, some relevant implementation details relative to each sub-block will be discussed.

4.4.1 Modified Median-Adaptive Prediction (MMAP)

In section 3.4.1 was discussed the operation of the MMAP block and some equations that dictate how the predicted values are obtained were shown. The equations that are relevant to the present discussion are repeated here for convenience:

$$P0 = CLAMP\left(a + blend_b - blend_c, MIN(a, blend_b), MAX(a, blend_b)\right) \tag{4.5}$$

$$\begin{aligned} P1 =& CLAMP(a + blend_d - blend_c + R0, \\ & MIN(a, blend_b, blend_d), MAX(a, blend_b, blend_d)) \end{aligned} \tag{4.6}$$

$$\begin{aligned} P2 =& CLAMP(a + blend_e - blend_c + R0+ \\ & + R1, MIN(a, blend_b, blend_d, blend_e), MAX(a, blend_b, blend_d, blend_e)) \end{aligned} \tag{4.7}$$

As the reader can observe, from pixel to pixel there are some results that can be reused. For example, the minimum operation involved in the determination of $P1$ can reuse the results of the minimum operation of $P0$. In the case of $P2$, the gains are even more relevant, because it reduces the problem of finding the minimum between four values, to the problem of finding the minimum between two values.

The previous discussion can also be extend to other values involved in the determination of the prediction values.

4.4.2 Block Prediction (BP)

There are not additional implementation details relevant to be discussed.

4.4.3 Midpoint Prediction (MPP)

There are not additional implementation details relevant to be discussed.

4.4.4 BP vs MMAP decision

The implementation of this block follows a suggestion given in the DSC Standard [3]. The given suggestion consists on reusing previous SAD results to reduce the amount of search operations. As discussed in section 3.4.4, each block of three pixels is compared from reference pixels with vectors in the range from -1 and -3 to -10. When the search is performed for the next group, the pixels in the positions (0,-1,.2) will be located in the positions (-3,-4,-5). The results of comparing the pixels in the positions (-3,-4,-5) with the relevant vectors, are the same as the ones obtained when the search was performed for the previous group. With this in consideration, the problem of determining the SADs for 3 blocks of pixels, is reduced to finding the SAD for only one block and reuse previous results. These operations constitute a pipeline with a latency of 10 clock cycles.

If a valid vector is determined, then the value is stored in a FIFO (“BP/MMAP FIFO” block), otherwise the value stored is “0xF”, which is a invalid vector value that indicates that MMAP should be selected. Notice that the computed decisions are only used in the next slice line.

4.4.5 BP/MMAP vs MPP decision

There are not additional implementation details relevant to be discussed.

4.4.6 Quantization

Besides quantizing the residuals, this block also computes the necessary number of bits to represent the residuals, taking into account that the residuals are represented in 2’s complement. This value is needed by the BP/MMAP vs MPP decision block and the Entropy Encoder block. There is a signal that indicates if MPP residuals are being processed. This is required because the MPP residuals size is always $cpntBitDepth - qlevel$, and besides that, if the residuals size is greater than this value, then they are further quantized.

4.4.7 Inverse Quantization and Reconstruction

There are not additional implementation details relevant to be discussed.

4.5 Indexed Color History (ICH)

Recalling the discussion on section 3.5, it was described the operation of the ICH coding technique. This section provides details about the implementation of the update operation, and the search for the best ICH entry for each pixel of the group. Contrarily to what is suggested in the section that describes the ICH operation on [3], the decision between ICH coding and Predictive coding, which are two independent coding techniques, is performed by a dedicated block that is instantiated in the encoder top block. The reason for this, is that the decision process requires information from both types of coding, and, if a dedicated block was not used, then the ICH block would require "direct communication" from the Predictive block, and would not be independent.

Figure 4.5 shows the structure of the ICH block. This block is constituted by 7 blocks: one memory block that behaves like a shift-register, three "search for best fit" blocks and the remaining ones, verify if the original pixels are within the quantization error (the test described on section 3.5 that verifies if the coding error does not exceed a threshold).

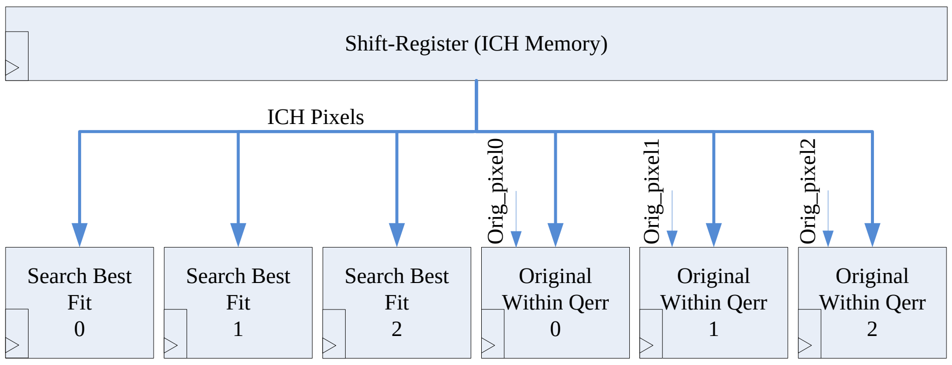


Figure 4.5: ICH block diagram.

New pixels arrive at the input of the encoder each clock cycle, so, the ICH block must process a group in 3 clock cycles (group time). As the reader recalls, the contents of the ICH memory are updated each group, which leaves 2 clock cycles to perform the remaining operations. For this reason it was necessary to parallelize the encoding process. Three instances of the "search for best fit" block and three instances of the "original within quantization error" block are used, and each one process a pixel, in one clock cycle. In the last clock cycle, the maximum component-wise errors are computed and registed. The same also applies to the results of the "original within quantization error" block. Those informations are used by the ICH/P decision block. The selected entries and corresponding pixel values are also registed and propagated to the outputs of the ICH block. Figure 4.6 illustrates the previous discussion.

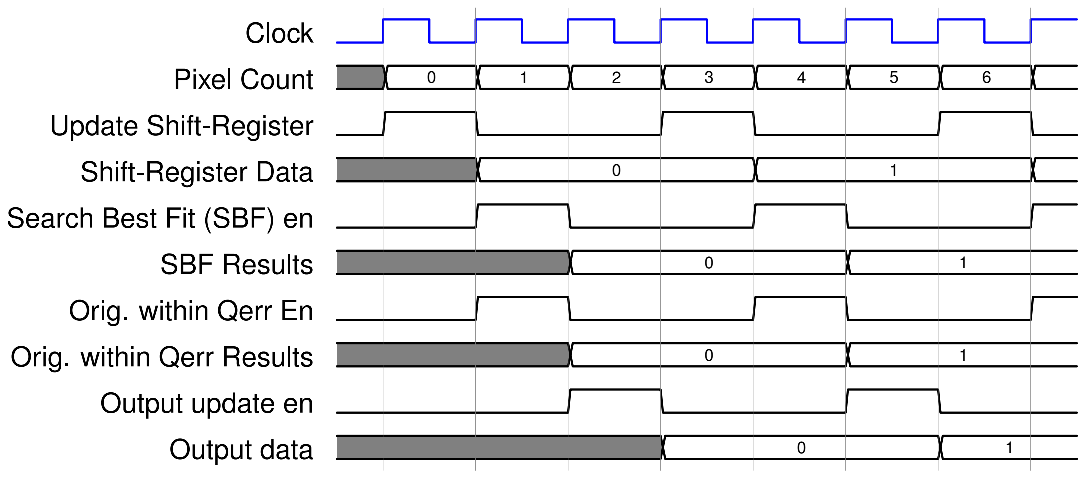


Figure 4.6: ICH timing diagram.

The remainder of this section describes the operation of each sub-block starting by the shift-register block. The shift-register block consists of a 32×39 bit memory. Each entry is 39 bits wide because in the worst case each pixel needs 38 bits to be represented and an extra valid bit is necessary to mark the entry as valid or invalid. Figure 4.7 illustrates the organization of the memory.

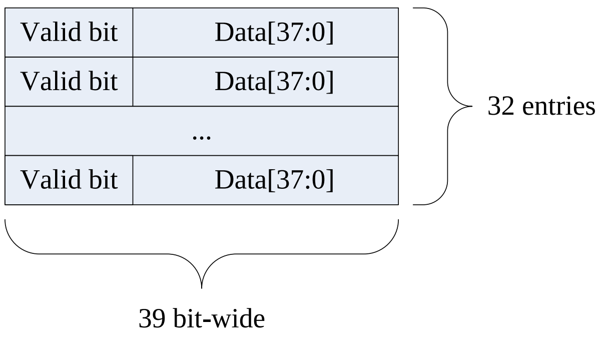


Figure 4.7: ICH memory organization.

The update process takes one clock cycle to be performed. Some combinational logic is required to control how the memory is updated and to handle some special cases, namely, when the previous group was ICH coded and some selected indices were not unique. In that case some indices are ignored by the update operation. It is also important to handle the transition from the first line to the second line of a slice because the first group must have available the first 7 pixels from the previous line, which implies loading the 7 pixels in one clock cycle. The problem of "clamping" the sliding window for the last groups, as described in section 3.5, was solved by keeping track of the position of the rightmost pixel of the sliding window, and disabling the update process when that pixel is to the right of the slice boundary.

The search for best fit block computes the 32 possible weighted SAD values in parallel. To find the minimum SAD value, a "divide-and-conquer" approach was implemented. The minimum is always computed between two values. In the first step, 16 values result, then, the minimum search is performed again and at the end of the second iteration, only 8 values remain. This process is repeated until the minimum value is found. This block has a total of 31 blocks that find the minimum between two values. The search for best fit block, takes one clock cycle to perform the described operations, so, each block that finds the minimum between two values, is implemented using combination logic only. Figure 4.8 illustrates how the search is performed.

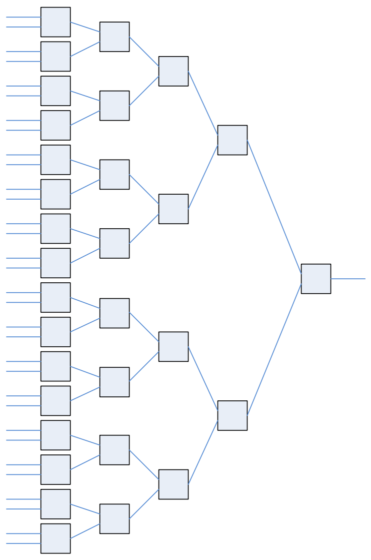


Figure 4.8: "Divide-and-conquer" search algorithm.

The "original within quantization error" block performs the operations described on section 3.5 for the 32 ICH pixels in parallel. At the end of the clock cycle, the results for each ICH pixel are ORed together and registed. If there is at least one entry that satisfies the criteria, then the output is "1", otherwise is "0".

4.6 ICH-mode vs P-mode decision

Once the results of the Prediction block and ICH block are available, and before performing the entropy encoding operation, it is necessary to decide between which coding technique to use.

The most relevant aspects of implementation, that are worth being mentioned, are related to the implementation of the $ceil_{log2}$ function. This function was discussed on 3.5 but now will be discussed from another point of view. As shown on [3], this function can be implemented as:

```
ceil_log2(x){
  temp = x
  ceil_log2 = 0
  while(temp != 0){
    temp = temp >> 1 //or equivalently: temp = temp/2
    ceil_log2 = ceil_log2 + 1
  }
  // result of ceil_log2(x) ready
}
```

Analyzing the above algorithm, the reader can conclude that the $ceil_{log2}$ function computes the number of bits that are required to represent the argument x. A direct implementation of the above algorithm is not feasible in hardware because of the while loop, nevertheless, the algorithm suggests a feasible solution: analyze the argument x in the direction from the most significant bit towards the least significant bit and find the first "1" bit. The result of the function will be equal to the weight of the digit plus 1. Consider the example shown on figure 4.9. The resulting value for this example is 6. As reader can confirm, the result is the same as the one produced by the code above.

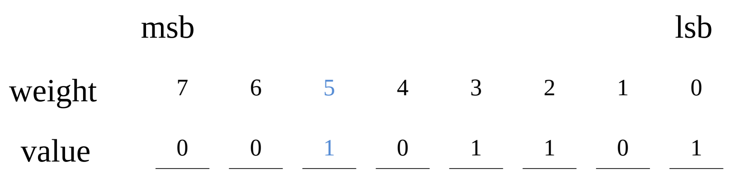


Figure 4.9: $ceil_{log2}$ computation example.

In terms of hardware this maps to a priority encoder and an adder block that sums the constant "1".

The remaining operations involved in the final ICH decision, are implemented as shown on [3] and section 3.5. The outputs are not registed (purely combination block) because of the reasons given in section 4.1.

4.7 Entropy Encoder

As discussed on section 3.7, the Entropy Encoder is responsible for building each syntax element. Each syntax element is constituted by a unit, whose size might vary from unit to unit. Furthermore, luma (Y) syntax elements might also include flatness information, namely, a flatness flag that indicates if there is a flat group in the next four groups (supergroup), a flatness type flag that indicates the type of flatness and a field that indicates which group the flatness adjustments apply to. The important point to emphasize here, is that the size of the syntax elements is not always the same. From an implementation point of view, this is problematic. Since there are variable length fields in each syntax element, a simple concatenation of the buses that represent each field, is not possible. It will now be discussed how this problem was handled.

It can be shown that the maximum syntax element size for each type of component, is 52 bits. With this in mind, the solution consisted in representing each field of a syntax element with 52 bits, and then, use bit masks in conjunction with shift operations to construct the "final" syntax element. A brief discussion of bit masks will now be made.

Consider for example the mask "00111" and the results of applying the mask to the value "$x = 11010$", using the AND and the OR operations as show on figure 4.10. As the reader can observe, the AND operation selected in x, the bits that are set in the mask used. The OR operation forced the bit positions in x, corresponding to "1" bits in the mask, to be set.

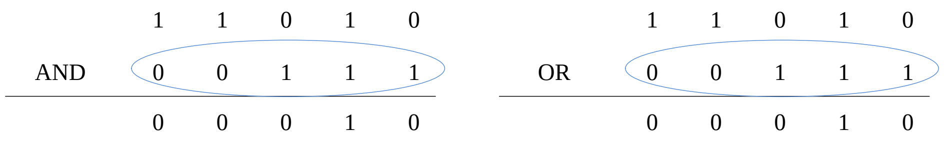


Figure 4.10: Bitmask operations.

With the previous discussion in mind, the syntax element construction will be discussed. Without loss of generality, it will be discussed the case when P-coded groups are being processed. First, the residuals are sign extended to fill the 52 bit bus. Next a mask with a number of "1"s equal to the largest residual size is applied to each residual:

$$maskedRes = residual \ \& \ mask \tag{4.8}$$

this ensures that each residuals is represented with the same size and ensures correct sign extension. Next, the three residuals are joined together:

$$suffix = (maskedRes0 << 2 \times maxResSize) \mid (maskedRes1 << maxResSize) \mid maskedRes2 \tag{4.9}$$

which constitutes the suffix of a unit. Observe that the residual corresponding to the last pixel of a group, occupies the least significant bits of the suffix. The next step consists of adding the prefix. First a mask is applied to the prefix to select the desired bits (which varies between 1 and 13), then the complete unit is obtained as:

$$unit = (maskedPrefix << suffixSize) \mid suffix \tag{4.10}$$

where $suffixSize$ is the size of the suffix. This completes the construction of an unit. In the case of the Co and Cg components, an unit corresponds to a syntax element. In the case of the

Y component, flatness info might need to be added, following a similar procedure to the one described above.

The remaining operations, namely, the size prediction for each unit, and the computation of the size of the group are straightforward to implement and follow the discussion on section 3.7. The results of this block are available in 1 clock cycle after the enable signal.

4.8 Substream Multiplexing

The substream multiplexer, multiplexes the three bitstreams (one of each component), that are generated by the entropy encoder. A decoder model is used to decide the order in which mux words are inserted in the multiplexed bitstream.

The architecture of the implemented block is shown on figure 4.11. As the reader can see, it closely follows the one shown on figure 3.16. The difference is in the FIFO that stores the size of each syntax element, which will be discussed next.

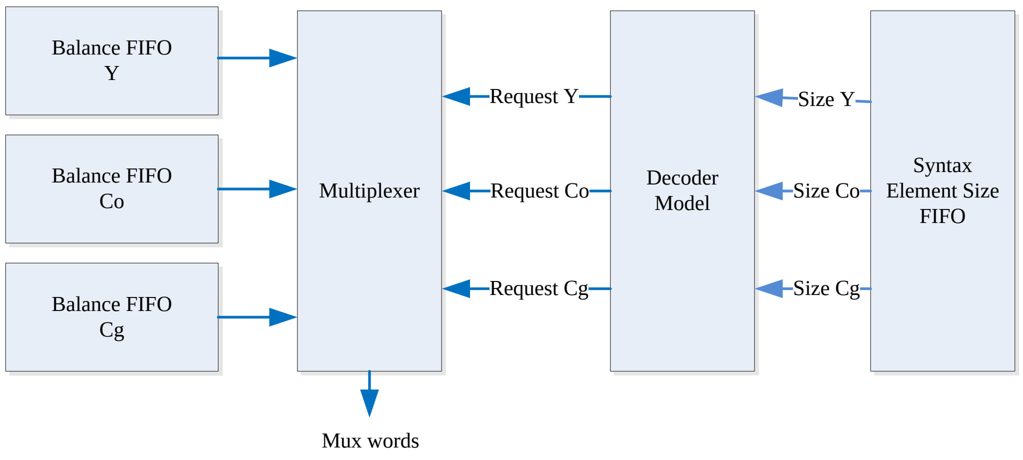


Figure 4.11: Substream multiplexer top block.

As discussed on section 3.8, before enabling the decoder model, it is necessary to accumulate a certain number of bits to ensure that the FIFOs do not underflow in any circumstance. After that it is necessary to know the size of each syntax element that is being processed by the decoder model, so, this implies that the size of each syntax element (i.e the size of the data written in the FIFO, each group time) must also be stored. Hence the need to have a FIFO storing the size of each syntax element. The number of entries of this FIFO should be equal to the number of entries of the balance FIFO, which is $muxWordSize + maxSeSize - 1$. Since the maximum mux word size is 64 bits and the maximum syntax element size is 52 bits, then, the number of entries is 115. To represent a syntax element size, 6 bits are needed, so, the memory requirements for the syntax element FIFO are $115 \times (6 \times 3) \approx 259$ bytes. Notice that only a single FIFO is needed to store the syntax element size of the three components.

Relatively to the balance FIFOs, each one has 115 words of *muxWordSize* (64 bit in the worst case scenario), which corresponds to a maximum size of 920 bytes per FIFO. There are some important details of implementation, worth being mentioned. As the reader recalls, the size of each syntax element can vary from syntax element to syntax element. This presents some difficulties of implementation. If each time a new syntax element arrives, a memory word is used, a scenario like the one shown on figure 4.12 might occur. The figure represents a fragmented FIFO, which in practice means that less than the desired 115 words of capacity are available. This is not acceptable because it is not possible to ensure that under all circumstances, the FIFOs do not overflow.

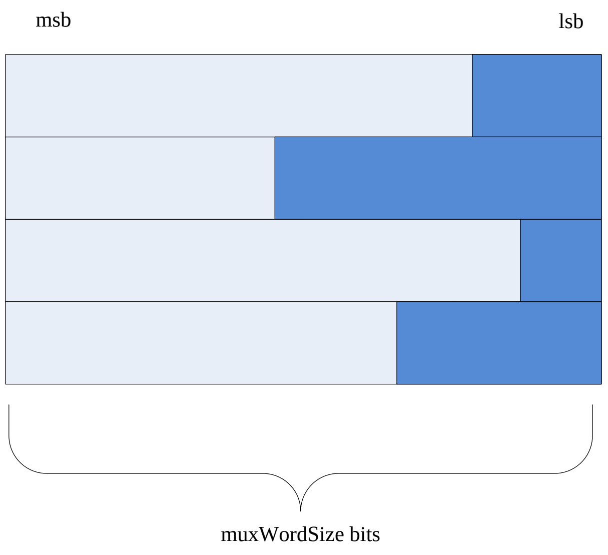


Figure 4.12: Balance FIFO fragmentation. The dark blue regions of each memory line corresponds to valid bits.

The problem was solved by using an auxiliary buffer with the size of a mux word. If the auxiliary buffer has sufficient space to hold the arriving syntax element, then the data is stored in the auxiliary buffer. When the buffer is full, it is moved to the “main” memory, avoid that way, the fragmentation problem. If the auxiliary buffer does not have enough space to hold the incoming data, then the contents of the auxiliary buffer plus the most significant bits of the incoming data that still fit in the buffer, are moved to the main memory. The remaining bits are stored in the auxiliary buffer. This situation is illustrated on figure 4.13.

The technique used to add data to the auxiliary buffer consists of using bit masks and shift operations, which is very similar to the technique used on section 4.7 to construct the syntax elements. When the auxiliary buffer has enough space to hold the incoming data, the auxiliary

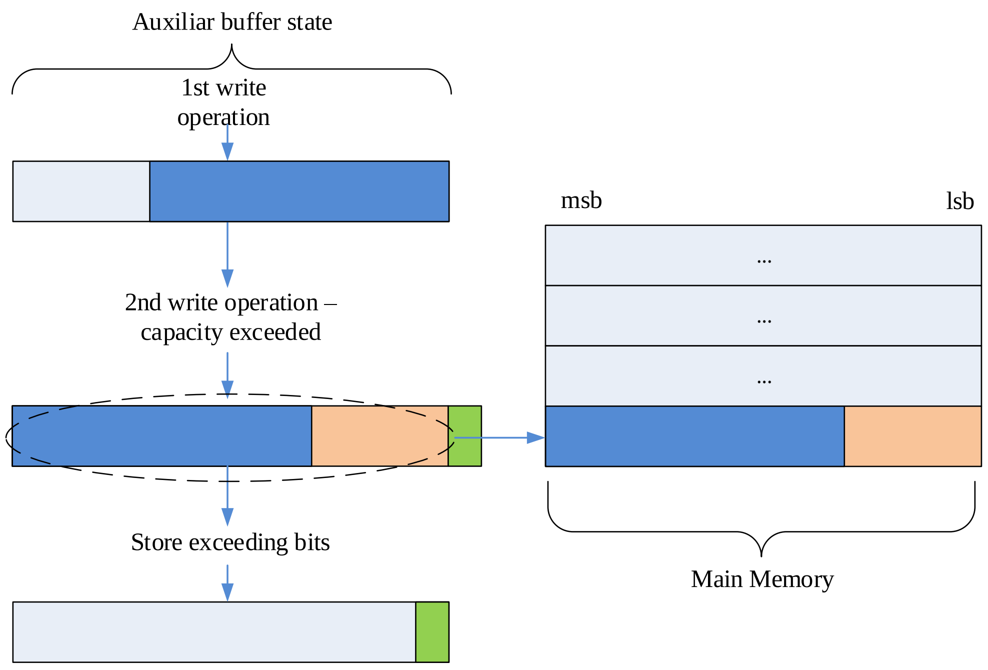


Figure 4.13: Balance FIFO operation.

buffer is updated as follows:

$$auxBuffer = (auxBuffer << numBitsToWrite) \mid (syntaxElem \ \& \ mask) \tag{4.11}$$

where *mask* selects the bits that will be stored. If the auxiliary buffer does not have enough space to hold the incoming data, then the buffer will store the bits that were not stored in the main memory:

$$auxBuffer = syntaxElem \ \& \ maskRemainingBits \tag{4.12}$$

where *maskRemainingBits* selects the least significant bits of the syntax element, that were not stored in the main memory. Observe that new data is always written in the least significant bits.

This approach of using an auxiliary buffer unfortunately brings another problem. When the last syntax element of a slice is written, the auxiliary buffer might be partially filled. Since a read operation reads a complete mux word from the main memory, then it is not possible to read the partial mux word. To correct this problem, a flush signal was added to the FIFO. This signal overrides the value of the number of bits to write, to a value that guarantees that the data in the auxiliary buffer is moved to the main memory.

Relative to the Decoder Model, there are not any caveats. The Decoder Model has three counters that represent the fullness level of each substream processor (SSP). Each time the fullness level is less than the maximum syntax element size possible, for the configuration in use (see section 3.8), a request signal is asserted. The fullness level is then updated as:

$$sspFullness = sspFullness + muxWordSize - SizeSyntaxElementProcessed \tag{4.13}$$

where $SizeSyntaxElementProcessed$ represents the size of the syntax element that is being processed.

It will now be discussed how the blocks work together and their timing relationships. During the delay period at the beginning of a slice, each group time, a syntax element of each component is stored in the balance FIFOs. This operation takes one clock cycle. In the next two clock cycles, no operation is performed.

After the balance FIFOs have enough data ($muxWordSize + maxSeSize - 1$) then in the first clock cycle each syntax element and respective size are stored in the corresponding FIFOs. A read operation of the syntax element size FIFO is performed. Notice that the value read, is only used for the next group, because the decoder model works on the same clock cycle. If a request signal is asserted for a given component, then in the second clock cycle, a read operation of the corresponding Balance FIFO is performed. In the third clock cycle, the mux words to be outputted, are available. Figure 4.14 illustrates the previous discussion. The first group has the request signal for each component asserted, so, there is a mux word of each component to be outputted. The second group has only the request signal for Co and Cg asserted, so, only two mux words are outputted.

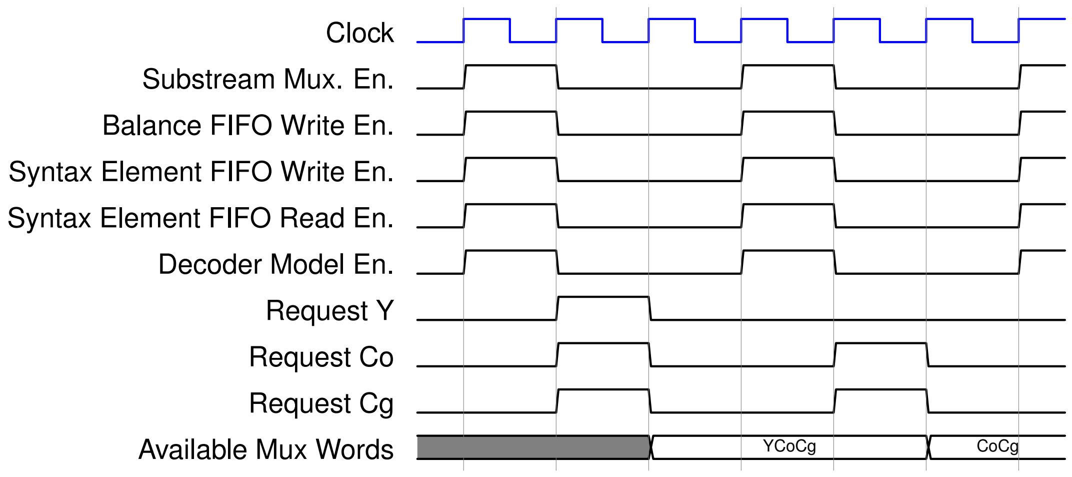


Figure 4.14: Substream multiplexer block timing example.

Before leaving this section it is important to mention that a FSM is used to control the operation of the Substream Multiplexer block. The FSM is shown on figure 4.15. The state after reset is “Initial Delay”. After the balance FIFOs have accumulated enough data (i.e, the Balance FIFOs are primed) the Decoder Model is activated and the block operates as described earlier. After the last syntax element of a slice is stored in the corresponding Balance FIFO, the FSM makes a transition to the “Finish streams” state. In this state data from the next slice is being stored but it is not removed (from the Balance FIFOs). Data from the current slice continues being removed.

After the last syntax element of the current slice is processed, if the Balance FIFOs are already primed with data from the next slice, then a transition to state "Send mux words" is performed, otherwise the next state is "Initial delay".

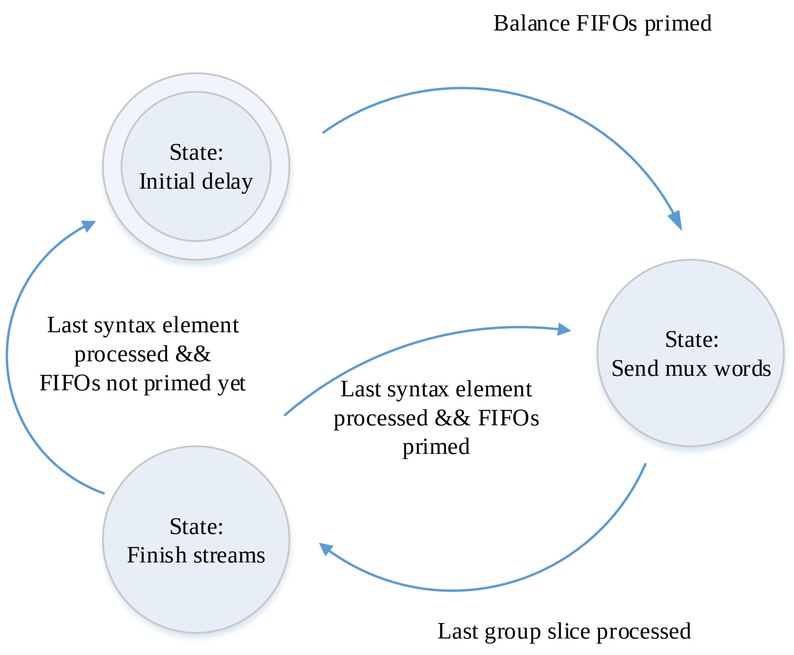

Balance FIFOs primed
State: Initial delay
Last syntax element processed && FIFOs not primed yet
Last syntax element processed && FIFOs primed
State: Send mux words
State: Finish streams
Last group slice processed

Figure 4.15: Substream Multiplexer FSM.

4.9 Line Buffer

The line buffer, as described on section 3.6, stores the previous line reconstructed pixels. In this implementation, it was assumed that the maximum width of a frame is 4096 pixels, so a 4096 x 38 bit (19 KB) memory is required. The memory is external to the encoder module, for two reasons. First, it is expensive in terms of area, to implement it using flip-flops. Second, it gives more flexibility. For example, FPGA's normally have external ram memory modules, but those modules are only used if it is possible to map the required ram to the one existing in the FPGA. By considering an external memory, the user has the possibility of correctly instantiate the memory, so it uses the existing ram module, instead of being implemented in flip-flops, which would quickly deplete the available resources.

Each group time, a new reconstructed group of pixels is available to be stored. Since only one pixel per clock is written in the ram, an auxiliary buffer (shift-register) is used two temporarily store the other two pixels. Auxiliary buffers are also needed for the read operation. As discussed in section 3.6, the blocks that need access to the pixels stored in the line buffer are the MMAP and ICH blocks. The MMAP block, to perform the filtering operation, it requires access to three pixels from the previous line. Since only one pixel is read per clock cycle, then a temporary buffer is required to store the pixels that were already read. The ICH block needs access to 7 pixels from the previous line, so, it also requires auxiliary buffers for the same reason.

4.10 Rate Buffer

The Rate Buffer block is responsible to damp the bitrate variations and to control the bitrate at the output of the encoder. Each group time, the Rate Buffer receives up to 3 mux words from the Substream Multiplexer. Each clock cycle a number of bits configured by the `bits_per_pixel` PPS parameter, is outputted.

From an implementation point of view, this brings some problems. The first one is related to how should the memory be organized. As the reader recalls, a mux word size can be 48 or 64 bits depending on the configuration in use. If each memory entry is 64 bit-wide, then, space is wasted then 48 bit mux words are processed, which must be compensated by increasing the number of memory lines. Another option is to choose the memory width to be the minimum common multiple between 64 and 48, which is 192. The disadvantage of this approach is that it adds complexity to the write operation. Not only is necessary to manage a pointer that points to the current line in the memory, but also, a pointer to the position within the line. The same reasoning applies to the read operation. Figure 4.16 and figure 4.17 illustrate the two alternatives discussed. The first one was implemented.

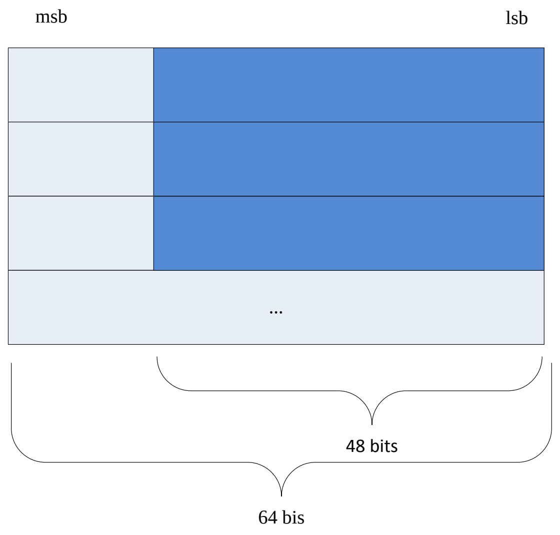


Figure 4.16: One mux word per memory address. In this example if the size of each mux word is 48 bit, then, 16 bit are wasted per memory address.

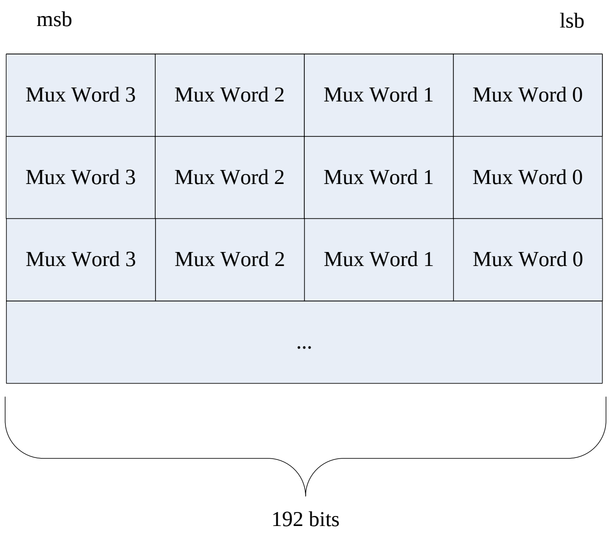


Figure 4.17: Several mux word per memory address. This figure shows the case when the size of each mux word is 48 bits. If the mux word size for the configuration currently in use is 64 bit, then, the number of mux words per address would be 3. This approach adds extra complexity to the final circuit because not only is necessary to identify how many mux words should be stored per address, but also, the boundaries of each mux word (where the mux word starts and ends) changes with the configuration.

The second caveat is related to the read operation. The number of bits that are read each clock cycle is smaller than the mux word size, which means that several read operations will be performed in the same mux word, and it is necessary to select the correct bits to read. This was solved by using bit-masks in a similar way as was used in section 4.7 to construct the syntax elements, but this time, for reading bits.

Relative to the read operation there is a last caveat. When the mux word size is not a multiple of `bits_per_pixel`, for example, when the mux word size is 48 bits and and `bits_per_pixel` is 15 bits/pixel, then a situation similar to the one depicted in figure 4.18 might occur.

In this situation it is necessary to read some bits of mux word x, and some bits from mux word $x+1$. Since only one mux word can be read per clock cycle, it is necessary to have available 2 complete mux words at all times. This was accomplished by using two auxiliary buffers and some combinational logic as shown on figure 4.19. When there is a need to read bits from the other buffer, bit-masks select the bits from each buffer, and then they are joined together and sent to the output. The multiplexer then reads selects the other buffer as the source buffer. The depleted buffer will be loaded with a new mux word from the main memory and the process repeats.

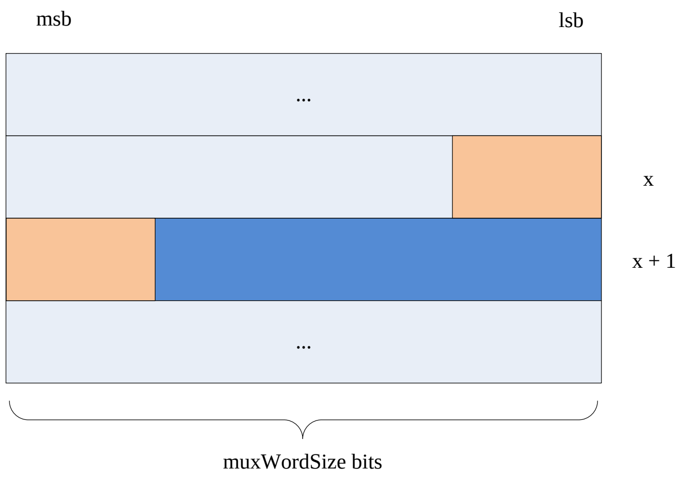


Figure 4.18: Read from two mux words at the same time. The dark blue rectangles represent available bits. The orange rectangles, represent the bits that will be read.

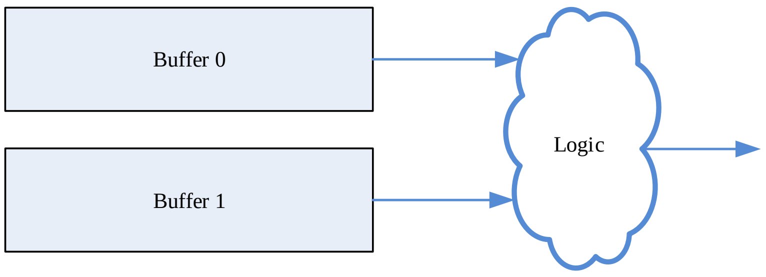


Figure 4.19: Auxiliary buffers.

Before leaving this section, it is important to mention that a FSM is used to control the operation of the Rate Buffer, namely, control the delay periods at the beginning of each slice ,that are required before data can be removed. This is analogous to the discussion on section 4.8. The slice bit budget corresponds to the total number of bits that code a slice. Figure 4.20 shows the FSM of the Rate Buffer block.

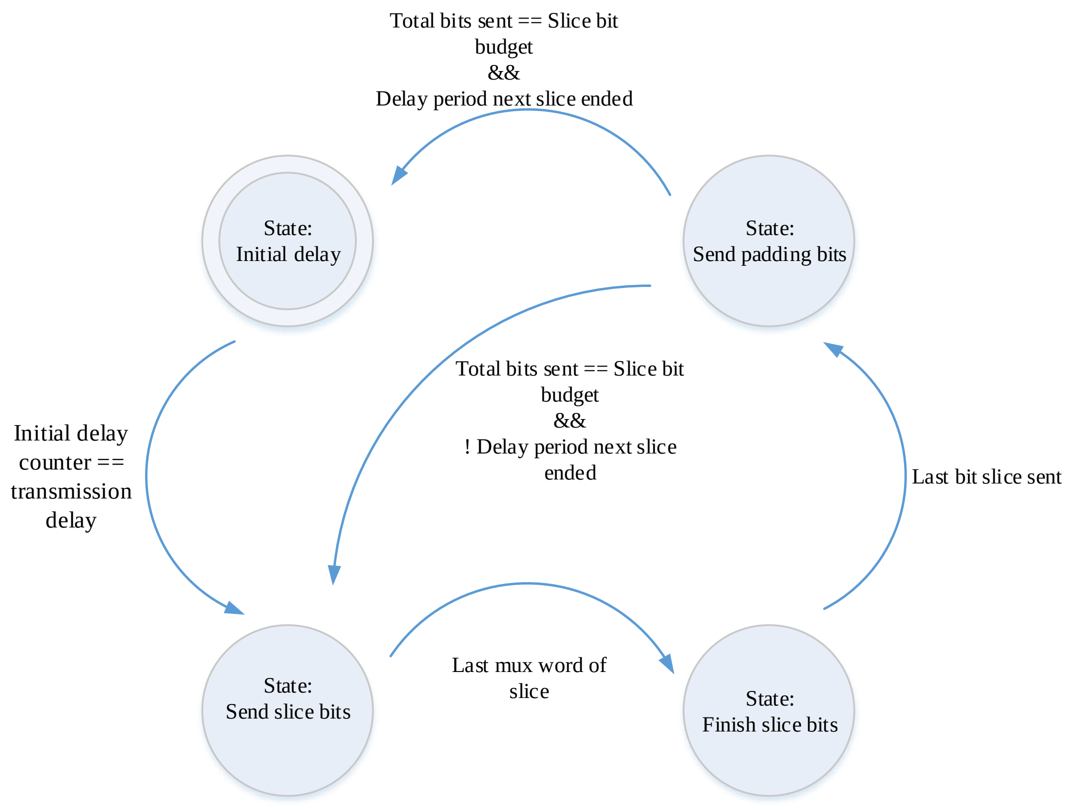

Total bits sent == Slice bit budget && Delay period next slice ended
State: Initial delay
State: Send padding bits
Total bits sent == Slice bit budget && ! Delay period next slice ended
Initial delay counter == transmission delay
Last bit slice sent
State: Send slice bits
Last mux word of slice
State: Finish slice bits

Figure 4.20: Rate Buffer FSM.

4.11 Rate Control

The Rate Control block architecture is the same as the one shown on figure 4.21. For convenience reasons, it is repeated in figure 4.21. The algorithms performed by each block are very complex from a theoretical point of view, specially the determination of the scale and offset values used in the Linear Transform, and the flowchart of the Short-Term RC block that decides the short term quantization parameter. In terms of implementation, each algorithm closely follows the guidelines on [3] and the reference C model. For that reason, the discussion will be focused on the timing relationships between the operation of each block, information that the C model lacks.

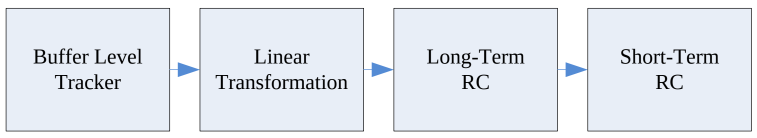


Figure 4.21: Rate Control block diagram.

The Rate Control top block operates on a group time basis (each 3 clock cycles), after the Entropy Encoder finishes coding a group. This block receives from the Entropy Encoder information that is necessary to perform the Rate Control operations, namely, the number of bits required to code a group.

The buffer level tracker determines the hypothetical rate buffer fullness level. This value is propagated to the output in the same clock cycle (combinational output). This block also generates a signal that forces the Midpoint Prediction to be selected for the next group. Since this signal will only be used in the last clock cycle of the next group, this signal is registed in a flip-flop.

After the buffer fullness is determined, the Linear Transform block determines the scale and offset value to be used in the computation of $rcModelFullness$ as described in section 3.6. This is performed in the first clock cycle too. In the second clock cycle the $rcModelFullness$ is computed.

In the third clock cycle, the Long-Term RC operation is performed. The scheduling so far described is not "rigid" in the sense that there is some flexibility in choosing which operations are performed in each clock cycle. The only restriction is that at the end of the third clock cycle, the results of the Long-Term RC must be available. The implemented schedule was chosen in such a way that guarantees that a complete clock cycle is available to perform the multiplication. Since this multiplication involves a multiplier that is not necessarily power of 2, then this operation must be implemented using a multiplier circuit, which is a slow circuit.

The Short-Term RC should use the results of the Long-Term RC of the previous group. This is accomplished by performing the Short-Term RC in the first clock cycle, where the results of the Long-Term RC for the current group, have not yet been computed. In the second clock cycle, the quantization parameter is modified to include flatness adjustments.

4.12 Flatness Determination

Section 3.10 described the operation of the Flatness Determination block. It was discussed the definition of supergroup, the flatness tests and the pixels involved in each test. As the reader recalls, each supergroup is constituted by four groups, and two flatness tests are performed for each group.

At this point it is important to discuss some timing related caveats. As stated in the Standard, when the remainder of the group number by four is 3 (i.e, every four groups) a flatness flag in the luma syntax element indicates if the supergroup that starts at the second group to the right has flatness informations. Figure 4.22 illustrates the discussion above. The figure also shows the pixel in which the tests are performed (shown with dark blue color)and the pixels involved.

Each group is tested using the $masterQp$ value currently in use for the current group, so, there is a window of 3 pixels in which is possible to perform the tests[1]. In this implementation, the tests are performed in the last pixel of the group, because it minimizes the amount of pixels that need to be stored. Notice that these are "future" pixels that have not been processed yet. The Buffer block

[1]The masterQp value changes on group times.

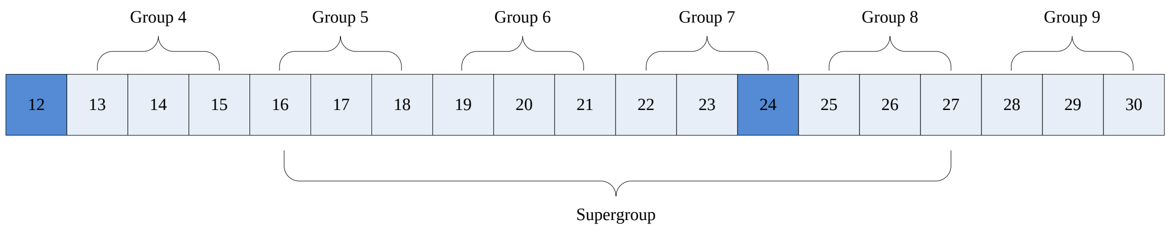


Figure 4.22: Flatness Determination timing. The figure shows the groups of pixels involved in the flatness determination for the supergroup that starts with group 5. Notice that this operation is performed when processing the pixel represented with the number 12.

on figure 4.1 is responsible for storing the required pixels, so, the minimum buffer capacity is the equivalent to 18 pixels, which corresponds to $18 \times 38 = 684$ bits (85 bytes).

Now it will be discussed how the above description is mapped to hardware modules. Figure 4.23 represents the architecture of the Flatness Determination block. As the reader can observe, there are four instances of the “Flatness Checker” block. Each instance is responsible for performing both tests for each group. Figure 4.22 also shows the pixels that each Flatness Checker block receives as inputs. The ignore signal permits ignoring the test results. This is necessary to deal with the case where some groups belong to the next slice line, because the search does not expand multiple slice lines [3].

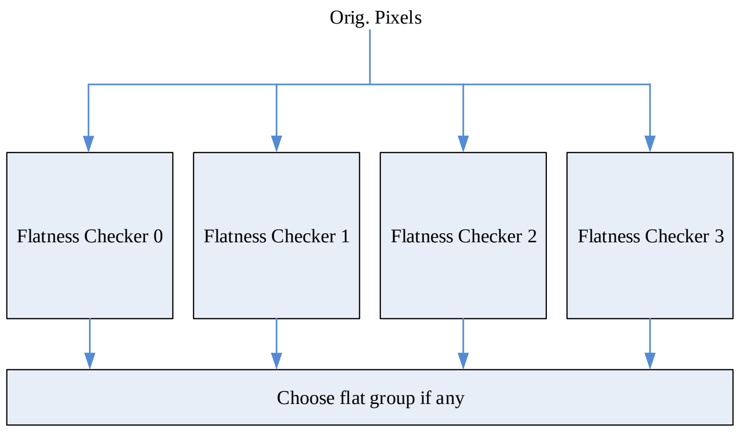


Figure 4.23: Flatness Determination block.

The outputs of each Flatness Checker block are not registed. The top block (i.e, the Flatness Determination block) might receive multiple flatness flags because more than one group of the supergroup might be flat. The top block is responsible to select the flatness information of the lowest index Flatness Checker, and to register it in flip-flops. The delay between the instant the Flatness Determination block is enabled, and the results are available, is one clock cycle.

Chapter 5

Results

In this chapter it will be described how the device under verification (DUV) was verified. It will be discussed the applied tests and which corner cases should be taken into account. Furthermore, coverage results will be presented. Later on this chapter, the synthesis results will be presented and discussed.

5.1 Test Results

A project of this dimension requires a verification procedure that does not rely on visual validation. The VESA DSC v1.1 Standard is accompanied of an implementation of the algorithm in the C language. This program outputs the generated bitstream to a file, according to a file format specified in the Standard [3]. Each file (with extension .DSC) contains the bitstream corresponding to the encoding of one frame. Figure 5.1 specifies the file format.

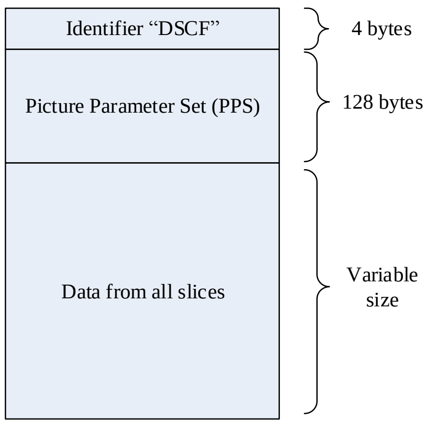


Figure 5.1: .DSC file format.

Figure 5.2 illustrates the verification procedure. The procedure consists of feeding the C model with the frame to be compressed, and the necessary configurations, like for example, the dimensions of each slice. At the end of this procedure a .DSC file with the reference bitstream will be available. The same procedure is replicated by the testbench. The tesbench reads the frame to be compressed from a file, feeds the device under verification (DUV) with the frame data and the required configurations. The testbench also records the bitstream generated by the DUV to a .DSC file. The DUV passes the test if both the .DSC files are equal bit by bit. Figure 5.2 illustrates the previous discussion.

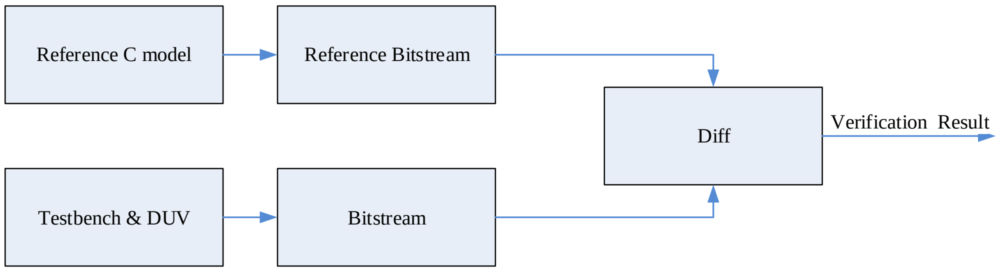


Figure 5.2: Verification procedure.

The DSC encoder was verified using frames with the dimensions of 1920x1080. The configuration that was used for verification consists of two horizontal[1] slices per frame, RGB input pixels with 8,10 and 12 bits per component and encoded pixels represented with 12 bits (i.e `bits_per_pixel` is configured to be 12). The use of two slices permitted the verification of the initialization conditions that apply each time a new slice is being processed. Changing the bits per component parameter, besides permitting the verification of conditions that depend on this parameter it also permits the verification of corner cases related to the manipulation of mux words by the Substream Multiplexer and Rate Buffer blocks. For example, relative to the Rate Buffer block, when the mux word size is 64[2] bits and `bits_per_pixel` is 12, then the operation of reading bits from both the auxiliary buffers at the same time, as described on section 4.10, is tested because the mux word size is not evenly divisible by `bits_per_pixel`. The contents of the slice being coded indirectly determine most of the decision outcomes, such as which type of coding should be selected for each group and the flatness adjustments.

Table 5.1 summarizes the coverage results. These results do not take into account any coverage exclusions such as protection code that during normal operation it is not stimulated. The previous discussion applies particularly to the FSM coverage. It is important to emphasize that there are not significant discrepancies between the coverage results of each block that constitutes the DSC encoder. Nevertheless, future work will improve this results to ensure a score of 100 percent.

Table 5.1: Coverage results.

Coverage	%
Line	90.14
Condition	90.70
Toggle	86.83
FSM	66.67
Score	83.58

[1]The slice height is half the frame height

[2]The mux word size is 64 bits when the bits_per_component is 10 or 12 bits.

5.2 Synthesis Results

In this section the synthesis results will be presented. Next, it will be discussed how the results can be improved.

Table 5.2 and table 5.3 summarize the results of the synthesis for each target technology (ASIC and FPGA respectively). The most relevant result for this dissertation is the maximum clock frequency supported. As expected, the ASIC technologies offer best performance (speed) relative to FPGA. Relative to the ASIC technologies, the TSMC28 technology offers best performance relative to the TSMC40LP technology because the transistors are smaller. Smaller transistors have smaller parasitic capacitors, which permits faster signal transitions. As the reader can observe from table 5.3, 88% of the resources that were used correspond to combinational elements. Due to the fact of the DSC algorithm being demanding in terms of timing, many complex operations must be performed in one clock cycle which explains this result. Next, it will be explained why it is not feasible to achieve the desired clock frequency values (600 MHz for ASIC and 300 MHz for FPGA). A solution that permits processing video signals that require this clock frequency through parallelization will also be discussed.

Table 5.2: Synthesis results for the ASIC technologies.

	Target technology	
	TSMC28	TSMC40LP
Max clock freq. (MHz)	200	125
Area (Kgates)	1054	1093
Total power consumption estimated (mW)	14.18	26.11

Table 5.3: FPGA (Xilinx Virtex-7 XC7VX690TFFG1926-2) results.

Max clock frequency (MHz)	47
Sequential elements (%)	12
Combinational elements (%)	88

As the reader recalls from chapter 3 and 4, at each group time, it is necessary to select the type of predictor that should be used for the current group. After that, it is necessary to decide which type of coding results should be sent to the Entropy Encoder (P-coding or ICH-coding results). This process of selection involves computing the maximum component-wise errors over the pixels in the group, computing the corresponding $ceil_{log2}$value, perform some arithmetic operations given by equation 3.24, which requires an estimation of the number of bits it will be required to code the group and finally the decision can be taken. The problem is that the results of this decision process are necessary to encode the 4th pixel (corresponding to a new group) and information from the 3rd pixel is necessary, so, this leaves a period of one clock cycle to perform all of these computations as illustrated on figure 5.3. Of course some results can be progressively computed. For example, as each pixel is processed, it is possible to compute the corresponding component-wise error, compare it to the maximum value found so far, and update the value. The problems is, this only reduces the burden of finding the maximum of three values to finding the maximum between two values at the end of a group. Indeed this is the main limitation factor (the transition between groups) to the maximum achievable clock frequency.

The limitation described above, indeed corresponds to the critical path of the encoder. More precisely, the timing report of the synthesis, indicates that the path that starts at the MMAP output and goes through the mentioned blocks/operations is the slowest one. Next, two suggestions of how to improve the performance will be discussed.

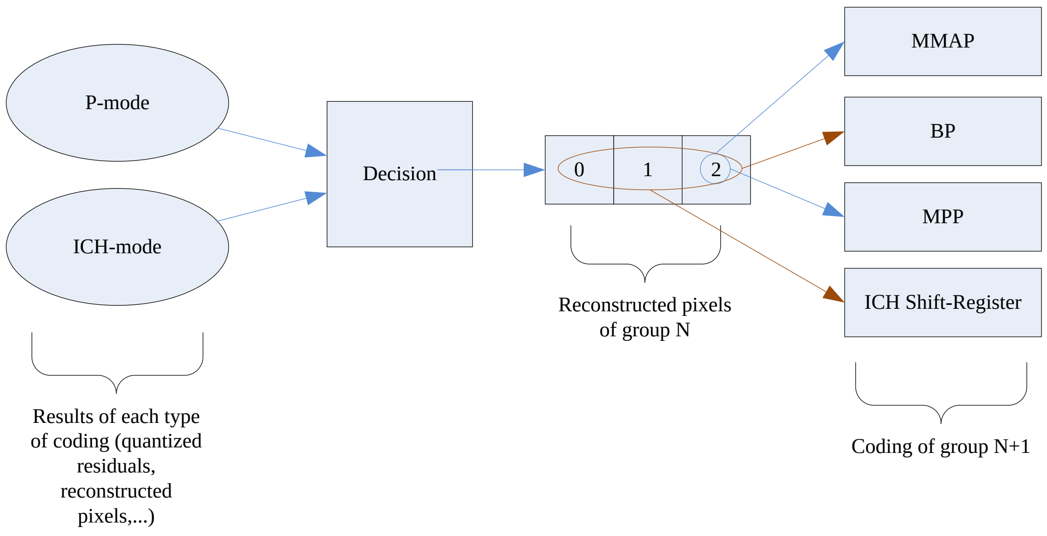

P-mode
ICH-mode
Decision
0
1
2
MMAP
BP
MPP
ICH Shift-Register
Results of each type of coding (quantized residuals, reconstructed pixels,...)
Reconstructed pixels of group N
Coding of group N+1

Figure 5.3: Data dependencies between groups.

The first approach consists of using multi-cycle paths. Consider the scheduling of operations shown on table 5.4 and the respective circuit on figure 5.4

Table 5.4: Schedule of operations.

Operation	Require time to complete
A	2 ns
B	1.4 ns
C	1.6 ns

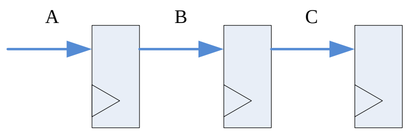


Figure 5.4: Circuit representation of the operations being scheduled.

In this example, to avoid setup time violations, the minimum period of the clock signal must be 2 ns. Some operations can be completed in less than 2 ns, but, since one operation is performed per clock cycle, then, each operation will require the whole 2 ns period to be completed. This represents "wasted time" that could be used to process the next operation. The results of operation C are only required at the end of the 3rd clock cycle. If a multi-cycle path constraint is given to the synthesis tool, then it is possible to schedule the 3 operations in the same clock cycle. In that case, the minimum period of the clock signal is $\frac{2+1.4+1.6}{3} \approx 1.7$ns, which represents a small improvement. Additionally it is necessary to guarantee that the flip-flops that store the results of operation C are only updated at the end of the 3rd clock cycle to avoid metastability related problems.

Following a similar approach, it is possible to modify the existing blocks to schedule the operations of the P-coding block and the ICH vs P-mode decision block in the same clock cycle. After that, a multi-cycle path condition must be given to the synthesis tool to signal that the results should be available at the end of the 3rd clock cycle. With this technique it might be possible to obtain a few MHz extra. Before leaving this discussion it is important to emphasize that this approach not always improves the circuit speed. For example, returning to the example in table 5.4, if all operations require 2 ns, then, no improvement in clock frequency would be obtained.

The second approach consists of using multiple encoder instances to process a given frame. As discussed in chapter 3, it is possible to configure the encoder for multiple slices per line usage. In this case, each instance would process one slice. Extra hardware would also be required to perform the clock domain crossing and the slice multiplexing function[3]. Figure 5.5 illustrates this concept, for the case where 2 encoder instances are used. The final bitstream would be a repeating pattern constituted by a line of slice 0 and then a line from slice 1.

This technique although does not increase the maximum achievable clock frequency value, permits increasing the throughput. For example, with three encoders implemented in the 28nm technology (TSMC28), each one working at 200 MHz, it is possible to process video signals that require a frequency of 600 MHz. The penalty is a significant increase in the required area, in this example, roughly three times.

[3]The slice multiplexer can also be seen as a CDC structure

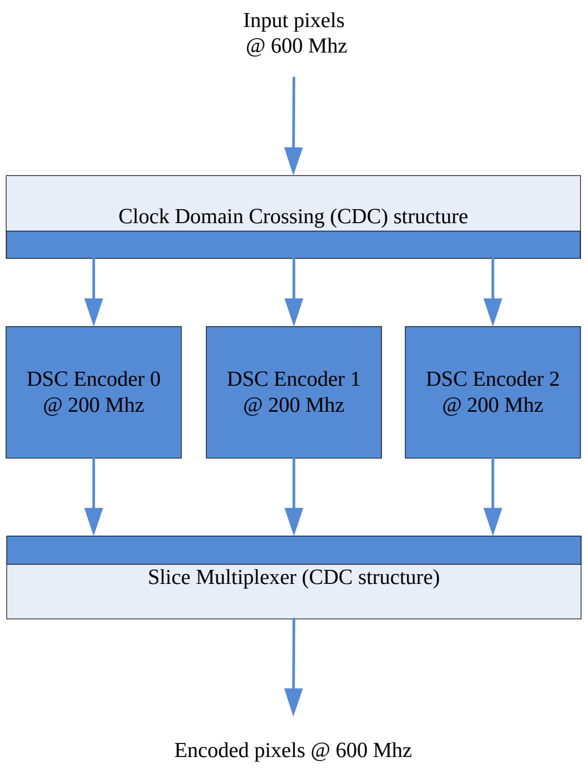


Figure 5.5: Multiple encoder instances. The dark blue color represents the domain of the 200MHz clock.

Chapter 6

Conclusion

The desire to produce displays that support higher resolutions over the time, as led to a significant increase of bandwidth requirements from the physical interfaces such as HDMI. Since video signals exhibit information that is not processed by the Human Visual System, it is possible to explore the existing redundancies to reduce the required bandwidth by using compression techniques.

This dissertation results from the implementation of the VESA DSC encoder. The encoder was functionally validated by comparing the generated output bitstream, with the reference bitstream generated by the C model. Relative to code coverage, the results were: 90.14% of line coverage, 90.70% of condition coverage, 86.83% of toggle coverage and 66.67% of FSM coverage. As future work, the coverage results at the time this dissertation was written, will be improved.

Relative to the performance of the encoder, the maximum clock frequency achieved in the ASIC technologies were 200 MHz and 125MHz for TSMC28 and TSMC40LP technologies respectively. Relative to the FPGA, the maximum clock frequency achieved was 47 MHz. To process 4K video (that requires a clock frequency value of roughly 600 MHz) 3 encoder instances must be used for the TSMC28 technology at the cost of a substantial increase (roughly three times) in the required circuit area.

As mentioned in the introduction chapter, there is an implementation of the VESA DSC v1.1 IP core. Relative to the clock frequency, it is stated that the encoder can achieve a clock frequency of 65 Mhz for a Virtex 7 XC7VX330T FPGA. In this implementation the obtained clock frequency was $47MHz$. Although the FPGAs used are different, the results are in the same order of magnitude.

From this point it would be interesting as future work implementing the decoder block. After a hardware demonstrator can be built. With a hardware demonstrator that includes a encoder and decoder, it is possible to measure the subjective visual quality of the displayed images. With such a demonstrator, it is even possible to introduce errors and observe the impact they have on the displayed images. This of course requires the necessary adaptations to take into account the transport layer(such as HDMI) requirements.

References

[1] Walls, Frederick and MacInnis, Sandy, “VESA Display Stream Compression.” http://www.vesa.org/wp-content/uploads/2014/04/VESA_DSC-ETP200.pdf.

[2] “ANSI/CEA Standard, A DTV Profile for Uncompressed High Speed Digital Interfaces, ANSI/CEA-861-F,” August 2013.

[3] *VESA Display Stream Compression (DSC) Standard Version 1.1*, August 2014.

[4] Andrade, Maria de Teresa , “Slides de Princípios de Compressão.” Unidade Curricular de Processamento e Codificação de Informação Multimédia 2014/2015.

[5] K. Sayood, *Introduction to data compression*. 2012.

[6] Y. Q. Shi and H. Sun, *Image and Video Compression for Multimedia Engineering: Fundamentals, Algorithms, and Standards*. Boca Raton, FL, USA: CRC Press, Inc., 2nd ed., 2008.

[7] G. J. Sullivan and T. Wiegand, “Video compression-from concepts to the H.264/AVC standard,” *Proceedings of the IEEE*, vol. 93, no. 1, pp. 18–31, 2005.

[8] T. M. Cover and J. A. Thomas, *Elements of Information Theory 2nd Edition*. Wiley Series in Telecommunications and Signal Processing, Wiley-Interscience, 2 ed., jul 2006.

[9] Z.-N. Li and M. S. Drew, *Fundamentals of Multimedia*. 2004.

[10] I. Richardson, *The H. 264 advanced video compression standard*. 2011.

[11] C. Cramer, E. Gelenbe, and P. Gelenbe, *Image and video compression*, vol. 17. 1998.

[12] J. S. Lee and T. Ebrahimi, “Perceptual video compression: A survey,” *IEEE Journal on Selected Topics in Signal Processing*, vol. 6, no. 6, pp. 684–697, 2012.

[13] M. Xu, Y. Liang, and Z. Wang, “State-of-the-art Video Coding Approaches : A Survey,”

[14] K. Thyagarajan, *Still Image and Video Compression with MATLAB*. Wiley, 2011.

[15] P. A. Wintz, “Transform picture coding,” *Proceedings of the IEEE*, vol. 60, pp. 809–820, July 1972.

[16] K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*. 2014.

[17] E. Bash, “Digital Video Concepts, Methods, and Metrics_ Quality, Compression, Performance, and Power Trade-off Analysis-Apress (2014),” *PhD Proposal*, vol. 1, 2015.

[18] J. P. Vink, G. De Haan, and H. Schmeitz, “Local estimation of video compression artifacts,” *Digest of Technical Papers - IEEE International Conference on Consumer Electronics*, no. 2, pp. 247–248, 2011.

[19] J. Pandel, “Measuring of flickering artifacts in predictive coded video sequences,” *WIAMIS 2008 - Proceedings of the 9th International Workshop on Image Analysis for Multimedia Interactive Services*, pp. 231–234, 2008.

[20] S. Rup and R. Dash, “Recent advances in distributed video coding,” *2009 2nd IEEE International Conference on Computer Science and Information Technology*, no. 1, pp. 130–135, 2009.

[21] J. R. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, “Comparison of the coding efficiency of video coding standards-including high efficiency video coding (HEVC),” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1669–1684, 2012.

[22] T. Wiegand, “Overview of the H. 264/AVC video coding standard,” *... and Systems for Video ...*, vol. 13, no. 7, pp. 560 –576, 2003.

[23] M. Khan, M. Shafique, L. Bauer, and J. Henkel, “Multicast fullhd h.264 intra video encoder architecture,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 34, pp. 2049–2053, Dec 2015.

[24] G. He, D. Zhou, W. Fei, Z. Chen, J. Zhou, and S. Goto, “High-performance h.264/avc intra-prediction architecture for ultra high definition video applications,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 22, pp. 76–89, Jan 2014.

[25] L.-J. Kau and M.-X. Lee, “Perception-based video coding with human faces detection and enhancement in h.264/avc systems,” in *Circuits and Systems (MWSCAS), 2015 IEEE 58th International Midwest Symposium on*, pp. 1–4, Aug 2015.

[26] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, “Overview of the high efficiency video coding (HEVC) standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[27] C.-C. Ju, T.-M. Liu, K.-B. Lee, Y.-C. Chang, H.-L. Chou, C.-M. Wang, T.-H. Wu, H.-M. Lin, Y.-H. Huang, C.-Y. Cheng, T.-A. Lin, C.-C. Chen, Y.-K. Lin, M.-H. Chiu, W.-C. Li, S.-J. Wang, Y.-C. Lai, P. Chao, C.-D. Chien, M.-J. Hu, P.-H. Wang, Y.-C. Huang, S.-H. Chuang, L.-F. Chen, H.-Y. Lin, M.-L. Wu, and C.-H. Chen, “A 0.5 nj/pixel 4 k h.265/hevc codec lsi for multi-format smartphone applications,” *Solid-State Circuits, IEEE Journal of*, vol. 51, pp. 56–67, Jan 2016.

[28] G. Pastuszak and A. Abramowski, “Algorithm and architecture design of the h.265/hevc intra encoder,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 26, pp. 210–222, Jan 2016.

[29] S. Wang, D. Zhou, J. Zhou, T. Yoshimura, and S. Goto, “Vlsi implementation of hevc motion compensation with distance biased direct cache mapping for 8k uhdtv applications,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[30] N.-S. Nguyen, D.-H. Bui, and X.-T. Tran, “Reducing temporal redundancy in mjpeg using zipfian estimation techniques,” in *Circuits and Systems (APCCAS), 2014 IEEE Asia Pacific Conference on*, pp. 65–68, Nov 2014.

[31] T. C. W. Lei and F. S. Tseng, “Study for Distributed Video Coding Architectures,” *2014 International Symposium on Computer, Consumer and Control*, pp. 380–383, 2014.

[32] Y. Sun, M. Xu, X. Tao, and J. Lu, “Online Dictionary Learning based Intra-Frame Video Coding via Sparse Representation,” pp. 16–20.