INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY

GRADUATE PROGRAM IN ELECTRONICS, COMPUTER SCIENCE, INFORMATICS AND COMMUNICATIONS

THESIS

MASTER OF SCIENCE IN ELECTRONIC SYSTEMS

WYNER-ZIV CODEC IMPLEMENTATION FOR DISTRIBUTED VIDEO APPLICATIONS

BY

RAUL VILLARREAL SANCHEZ

TECNOLÓGICO DE MONTERREY

MONTERREY, N. L., DECEMBER 2011

INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY

GRADUATE PROGRAM IN ELECTRONICS, COMPUTER SCIENCE, INFORMATICS AND COMMUNICATIONS

THESIS

MASTER OF SCIENCE IN ELECTRONIC SYSTEMS

WYNER-ZIV CODEC IMPLEMENTATION FOR DISTRIBUTED VIDEO APPLICATIONS

BY

RAUL VILLARREAL SANCHEZ

TECNOLÓGICO DE MONTERREY

MONTERREY, N. L., DECEMBER 2011

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS MONTERREY

GRADUATE PROGRAM IN ELECTRONICS, COMPUTER SCIENCE, INFORMATICS AND COMMUNICATIONS

THESIS

MASTER OF SCIENCE IN ELECTRONIC SYSTEMS

Wyner-Ziv Codec Implementation for Distributed Video Applications

by

Raúl Villarreal Sánchez

TECNOLÓGICO
DE MONTERREY®

Monterrey, N.L., December 2011

©Raúl Villarreal Sánchez, 2011.

Wyner-Ziv Codec Implementation for Distributed Video Applications

by

Raúl Villarreal Sánchez

T h e s i s

Submitted to the Graduate Program in Electronics, Computer Science, Informatics and Communications

at the

Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Monterrey

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electronic Systems

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

Monterrey, N.L., December 2011

Dedicatoria

A mi papá Raúl Villarreal Peña que siempre me ha dado las mejores herramientas para la vida y nunca ha dejado que algo me falte.

A mi mamá Martha Alicia Sánchez de Villarreal que siempre esta empujándome a alcanzar mis metas y que nunca me ha dejado rendirme a mitad de camino.

A mi hermana Martha Alicia Villarreal Sánchez que me inspira a seguir adelante y me brinda alegría cuando más lo necesito.

A los tres que siempre me han apoyado con mis decisiones, brindado su amor incondicional y que nunca me han faltado, les dedico esta tesis que es fruto de mi esfuerzo y su apoyo.

Agradecimientos

Quiero agradecer de una forma muy especial a mis tíos Juán Pablo Robles y Elsa Aracely Sánchez por haberme brindado su hospitalidad durante la realización de este trabajo.

Agradezco a mi asesor, Dr. Ramón Martín Rodríguez Dagnino por su tiempo y sus consejos que gracias a ellos logré este objetivo.

Agradezco a mis sinodales Dr. César Vargas Rosales y MSc. Luís Ricardo Salgado Garza, cuyas contribuciones me ayudaron a completar este trabajo con éxito.

Gracias a mis amigos de Querétaro, por todos los grandes momentos que hemos pasado y por brindarme los mejores años de mi vida.

Gracias a mis amigos de Celaya que me reciben y me dan calor de hogar cada que voy de visita y me hacen sentir como si nunca me hubiera ido.

Gracias a mis amigos de Monterrey, por todas las experiencias que me dejaron vivir junto a ellos durante estos dos años, por dejarme conocerlos y ser parte de su vida.

Por último, gracias a mi familia, abuelas, tíos, primos, que durante estos años siempre los llevé conmigo y sus palabras de aliento siempre me impulsaron a seguir con este proyecto.

Abstract

Video codecs are thought to be used in systems where codification is done once and decodification is done several number of times, this implementation requires the encoder to be 5 to 10 times more complex than the decoder [Aaron d] because of the interframe codification.

A new paradigm for distributed video applications have been arising in the last few years based on the work of Aaron Wyner and Jacob Ziv. Frames are encoded in an intraframe coding scheme but decoded in an interframe scheme using an information generated at the decoder known as side information.

Wyner-Ziv codec modules are designed using VHDL and synthesized for the Spartan-3A DSP Video Starter Kit, the modules were designed in pipeline so real-time execution can be performed. The hardware implementation has a maximum frequency of 64.496 MHz which is enough to process video with 1080i resolution.

Lossy and lossless compression methods were employed to get the maximum compression rate so the codec can be implemented in a low bandwidth communication system.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Standards for video coding such as H.263 are already well documented and have good performance, unfortunately that schemes are not achievable by some devices without the desirable processing power such as mobile devices, and also they are only efficient for some kind of applications such as broadcasting.

Video coding standards such as H.263 and MPEG-2 exploit statistical information from the source signal [Girod 05], hence, any statistical estimation or main codification algorithm is being held at the encoder. Codecs under this standards have complex encoders but simpler decoders.

Distributed Video Coding is a paradigm which states that efficient compression can also be achieved by exploiting source statistics at the decoder only [Girod 05].

David Slepian and Jack Wolf in [Slepian 73] proved that, for two independent but correlated signals X and Y, if they are encoded separately and jointly decoded information can be reconstructed as if they were jointly encoded. Their algorithm also included how to select the minimum number of bits of representation per character, R_x and R_y, for the information to be correctly reconstructed by proposing an admissible error region bounded by the conditional probabilities of $R_x > H(X|Y)$ and $R_y > H(Y|X)$.

Aaron Wyner and Jacob Ziv then proposed a random variable known as side information Y in order to increment the rate distortion performance [Wyner 76], this side information can be known by both the encoder and the decoder or only by the decoder, by Wyner and Ziv work it can known that both are equally efficient. In practice, side information frames Y are also called key frames and are usually special encoded frames or a noisy version of X, also this information is important for any distributed video coding implementation. [Peixoto 08]

Figure 1.1 shows the basic structure of a Wyner-Ziv codec [Girod 05], the architecture consists in a quantization module, a Slepian-Wolf encoder, a Slepian-Wolf decoder and a reconstruction module, it can be seen from the diagram that the decoder have access to the side information Y which is the Wyner-Ziv codec main characteristic.

This block diagram is not the final system diagram, it is just a general representation of how the codec is constructed, it is possible to add or modify the modules as they are needed within the constrains of the codec.

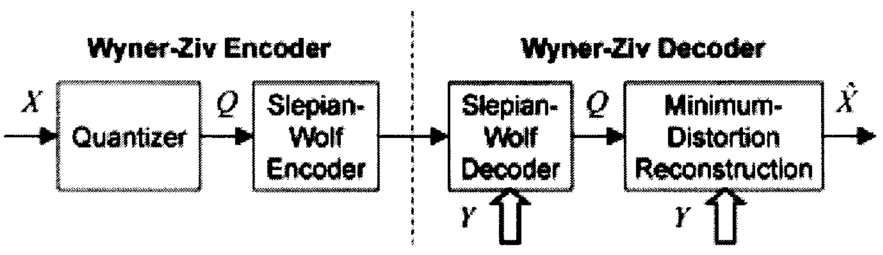


Figure 1.1: Basic Wyner-Ziv Codec Structure. [Girod 05]

Some implementations under Wyner-Ziv scheme had been done already, each one with different implementations and algorithms. Some examples of this implementations are the PRISM algorithm by Puri and Ramchandran [Puri 02], pixel domain transformation used by Bernad Girod, et al. [Girod 05].

Eduardo Peixoto, et al. proposed a method where the movement between two consecutive key frames F_2k-1 and F_2k+1 can be modeled to obtain the frame F_2k [Peixoto 08]. João Ascenso, et al. proposed an improved side information construction based in motion compensated frame interpolation algorithms at the decoder [Ascenso 06].

These methods were implemented during the last years, the first one was done in 2008 and the second one in 2006, by these examples a new codec can be implemented by looking for opportunity areas.

1.1 Video Basics

Before going further, some basic concepts about video have to be discussed as they serve as general knowledge and will help to understand some modules functionality.

Video works over the principle that motion is generated by showing a sequence of slightly different images at a rate high enough for the human eye to be able to interpret motion. This effect is possible because the human visual system and its property to retain an image even if it was removed from the sight [Haskell 97].

A video system can process this individual images, from now on referred as frames, generated by a video source like a camera. Frames can be processed independent from each other or they can be processed as a group of frames that depend one from each other, these processing methods are known as intraframe coding and interframe coding respectively.

For intraframe coding, each frame is processed individually, no temporal redundancies are taken into account, only statistical redundancies are used to encode data. This codec scheme sends all the information of each frame in the video sequence. Strictly speaking, this would be an optimum codification method as the decoder does not have to estimate or predict any of the frames in the sequence, in reality this is not an efficient method as a lot of information is sent from the encoder to the decoder and the decoder is not always fast enough to reconstruct the information so these codecs are usually slow.

It was said before that video can be interpreted as a sequence of slightly different frames, but even if it is true that no frame captured depends from past frames as it is possible to capture totally different images at the same rate they are being captured, the reality is that frames in a video sequence are correlated because, for a sequence of frames, the frame $N + 1$ is most likely to be similar to the frame N and so, the frame sequence will have temporal redundancies that can be used to reduce the amount of information to be sent from the encoder to the decoder. These are the basis for interframe coding, it takes advantage from statistical and temporal redundancies in a video sequence. Of course less information is needed to be sent but more hardware is required to interpret this temporal redundancies and so, there are devices that are not able to encode video in this manner.

The discussion so far have covered the two coding schemes used to process video, but in order to understand what steps have to be taken to correctly encode and decode the information it is important to know where this information comes from and how it comes.

Imaging is the representation of a physical scene, there are different types of imaging, e.g., video, X-rays, just to list few [Haskell 97]. The one important for this project is the video which comes from different sources, the most recent are the solid-state sensors, the image in the lens is scanned and converted into an electrical signal that can be read and manipulated [Haskell 97].

There are two methods for scanning the image in the camera lens. The first one is called progressive and it consists in scanning the image from left to right and from the top to the bottom of the image, i.e., the first line of the image is scanned from left to right and converts the intensity perceived in each point to an electrical signal, when it finishes the first line it moves to the next line and continues the scanning until it reaches the end of the image [Haskell 97].

The second method is known as interlaced scan, it separates the image in two fields that are sampled at different times and then lines from both fields are interleaved, this way consecutive lines will be from different fields. As there are specific solutions for every problem, how the image is scanned will depend on the application target, for example, for computer displays the progressive scanning is more efficient while for TVs interlaced scanning will be more efficient [Haskell 97].

This thesis is focused in progressive video and this is important because there is a module in the codec that will scan 8×8 quantized blocks, and the scanning pattern is different for progressive and interlaced video applications.

It is worthily to know that each electrical value read by the imaging scan that is known as a *pixel* is most commonly known in video terminology as a *pel* and it is the minimum area reproduced by an imaging device. This was explained because the next characteristic about video that will be covered is the Image Aspect Ratio (IAR), which determines the ratio between the width and the height of the image to be displayed.

Before displaying video over a display it is necessary to know if the display supports the IAR of the video or if the video has to be adjusted to be compatible with the IAR of the display where the video is intended to be showed, the most common IARs used are the 4:3 and 16:9 which are IARs for standard and widescreen displays respectively [Haskell 97].

Pels are shapeless so their size can be adjusted depending the IAR used by the display, their size are calculated by calculating the Pel Aspect Ratio (PAR) with Eq. 1.1.

$$PAR = IAR * \frac{height}{width} \tag{1.1}$$

Now that some aspects about representation and size of the frames have been explained it is also important to know how video takes care of color. To understand the next discussion two concepts are introduced, hue, which is the color produced by visible light, and saturation, which is the degree of purity of the color [Haskell 97].

The image scanning will give three values for three different colors, one value for red, one for blue and one for green, the combination of different amounts of these three colors generates a new color, and so, all the visible colors can be represented by combining these colors. The main problem is that video systems do not use this color space and it has to be transformed in order to be processed, when video has been processed it is again converted to RGB space and displayed.

The most common video system is the composite video system which consists in three different systems that will be further discussed, the NTSC system, used in North and South America, the Caribbean, and some parts of Asia, the PAL and SECAM systems, used in the rest of the world [Haskell 97].

NTSC, PAL and SECAM video systems are called composite systems because their luminance and chrominance values are multiplexed over the same carrier and send over the same transmission media [Haskell 97].

The color space transformation is easy as the spaces employed by the composite video systems can be generated from the RGB color space, in fact, the values used to convert from RGB to any other color space will be R', G' and B' which are the same RGB values but gamma corrected, which is an energy and luminance balance correction as they are not linear when read.

NTSC system uses the YIQ color space which values are calculated with Eq. 1.2, it uses 4.2MHz of bandwidth for luminance, 1.3MHz for I and 0.6MHz of bandwidth for Q [Haskell 97].

$$\begin{aligned} Y &= 0.299R' + 0.587G' + 0.114B' \\ I &= 0.596R' - 0.274G' - 0.322B' \\ Q &= 0.211R' - 0.523G' + 0.311B' \end{aligned} \tag{1.2}$$

PAL system uses the YUV color space which values can be calculated with Eq. 1.3 and it assigns from 5MHz to 5.5MHz of bandwidth for luminance while it uses 1.3MHz of bandwidth for chrominance values [Haskell 97].

$$Y = 0.299R' + 0.587G' + 0.114B'$$

$$\begin{aligned} U &= -0.147R' - 0.389G' + 0.436B' \\ V &= 0.615R' - 0.515G' - 0.1B' \end{aligned} \tag{1.3}$$

SECAM system uses the color space $YDrDb$ which values for luminance and chrominance are found with Eq. 1.4, this system uses 6MHz of bandwidth to the luminance while it uses at least 1MHz of bandwidth for chrominance [Haskell 97].

$$\begin{aligned} Y &= 0.299R' + 0.587G' + 0.114B' \\ Dr &= -1.333R' + 1.116G' - 0.217B' \\ Db &= -0.450R' - 0.883G' + 1.333B' \end{aligned} \tag{1.4}$$

This knowledge is useful because, when designing a video system it is necessary to know where it is going to be implemented as every system has its own parameters, bandwidth usage and transmission media, then, this characteristics have to be known for the system to be compatible.

So far analog component systems have been reviewed but it is for the interest of this thesis that digital component system is reviewed as the codec is implemented over a digital system.

Digital component system uses the color space known as $YCrCb$ that derives from the YUV color space as its values are found with an offset of a scaled version of Eq. 1.3, so the values for luminance and chrominance are found with Eq. 1.5, each value is represented with 8 bits.

$$\begin{aligned} Y &= 0.257R' + 0.504G' + 0.098B' + 16 \\ Cr &= 0.439R' - 0.368G' - 0.071B' + 128 \\ Cb &= -0.148R' - 0.291G' + 0.239B' + 128 \end{aligned} \tag{1.5}$$

When working with digital video another concept is introduced, the video format, which is the distribution and number of samples for the luminance and chrominance values. Each format represents a relation between luminance and chrominance samples and if they are vertical or both vertical and horizontal distributed.

4:2:0 format implies that there are going be half the chrominance samples (Cr and Cb) than the luminance samples (Y) in both directions, horizontal and vertical, this format is used by MPEG-1 and MPEG-2 and it can be used for interlaced and non-interlaced configurations [Haskell 97].

4:2:2 format is almost the same as 4:2:0 format in the number of samples for luminance and chrominance values but with the difference that they are only distributed in the vertical axis of the image. This format is also used by MPEG-1 and MPEG-2 architectures and it can also be implemented as interlaced and non-interlaced, another difference is that even when both configurations can be used, interlaced video is the most common for this video format as it takes advantage on the characteristic that this format eliminates the color degradation associated with the interlaced 4:2:0 chrominance [Haskell 97].

4:4:4 format uses the same number of samples for luminance and chrominance values, also they are distributed both horizontally and vertically, this format is most commonly used in computer displays, hence, is more associated to progressive video system and is only supported by MPEG-2 [Haskell 97].

Discussion before was important because, just like IAR, it is important to know what kind of application will be implemented to know the functionality of the video system and how it has to be adapted so it can work.

1.2 MPEG-2

So far concepts about how video is obtained and what kind of formats it supports had been discussed, the information generated by the video source has to be processed, as it was mentioned before, there are two ways to code video, intraframe coding and interframe coding, the first one takes each frame captured, process it as an individual image and sends it through a transmission media, this is suitable for devices with low processing capabilities as they will use only the incoming data to encode video.

The second way, interframe, not only uses the incoming data, but it also uses past frames to encode the present frame, this coding style sends less information as it takes advantage from temporal redundancies but it requires more hardware to encode the data.

Both styles take advantage from statistical redundancies as both sends the data with variable length codes that are assigned depending of the probability of the event to be encoded. Only the interframe codec will take advantage from temporal redundancies.

This is where codec standards are involved, they encode and decode the incoming data following a number of rules that determine what happens when a luminance or chrominance block is being encoded, how the data will be processed for an intraframe or interframe codec and all that decisions to be made in order to comply with a certain standard.

One of the most commonly video codec used is the MPEG-2 which is able to work in both intraframe and interframe coding schemes. The standard not only covers video but also audio and additional information such as video manipulation by the user. The standard sends all this information through the same channel by data multiplexing [Haskell 97].

As MPEG-2 is a standard that takes into account video, audio and additional data, this discussion will only review the second part of this standard which talks about video. It is important to note that the official name of the MPEG-2 standard is ISO-113818 and is divided in 10 parts, each one states the rules and requirements for a video system to comply with this standard, as it was said before in the same paragraph, the second part is the one that states the rules for video coding.

MPEG-2 does not standardize encoding techniques, it does have some coding techniques but they are not standardized, MPEG-2 defines the rules for the bitstream and decoding semantics [Haskell 97], hence, MPEG-2 encoders are generic and they are built however the designer wants the encoder to be, the only restrictions is that the output bitstream has to be constrained inside a set of rules declared by the standard.

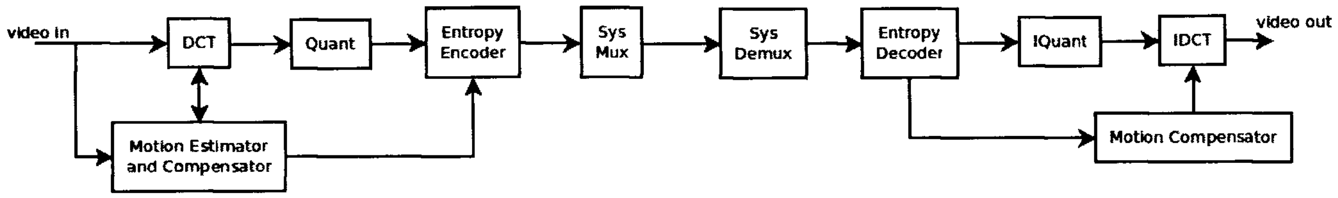


Figure 1.2: MPEG-2 Generic Block Diagram

Fig. 1.2 shows a generic MPEG-2 codec architecture for interframe coding, it takes advantage from both spatial and temporal redundancies, the DCT module will take advantage from the spatial redundancy and the motion estimator will take advantage from the temporal redundancies.

The motion estimator/compensator is the module that makes the codec an interframe codec, it categorize frames in three types, I, B and P frames. An I-frame is an intra frame, this picture will be encoded in an intraframe fashion, i.e., all information about the picture is sent, not caring if there are still images inside the picture. A P-frame is a predicted frame also known as delta-frame, this frame will only send differences between the actual frame and the past frames and so, few information is sent over the channel. At last, the B-frame is a bidirectional frame which takes information from past frames and next frames to determine the contents of the picture.

This information is important because of how MPEG-2 standard asks for information. The standard divides the video sequence en groups of pictures (GOP), the GOP will assign each picture as an I, P or B frame. Then, the encoder will have information enough to know how the frame will be encoded and how it has to send the bitstream.

Now that it is known how many pictures there are going to be in the GOP and how it will encode each of those pictures, the codec has to know how the image is separated. First, the image is cut into slices, each slice contains a group of macroblocks, that will be discussed later. This slices are for synchronization so transmission errors can be prevented [Haskell 97].

Macroblocks are 16×16 motion compensation units [Haskell 97]. When the codec is found to be encoding a P-frame or a B-frame, some macroblocks can be skipped and so, they are not sent to the decoder. They are groups of blocks, which are the smallest data and the ones that are processed by the codec.

Blocks are 8×8 data blocks and they are the ones that are processed by each module in the video codec, some codecs use 4×4 blocks but this thesis uses 8×8 data blocks.

Of course the codec architecture presented in Figure 1.2 is just a simple representation of how a MPEG-2 codec looks like but the designer can modify the modules he thinks are more convenient or efficient but always within the constrains dictated by the MPEG-2 standard.

The reason for discussing the MPEG-2 standard is because the encoder for this thesis is closely related to the MPEG-2 encoder. As the encoder will be designed as an intraframe encoder, the pictures can be thought to be I-frames in the MPEG-2 codec.

There are differences like the fact that the Wyner-Ziv encoder does not have a motion estimator module.

Some of the basic modules that are found in a MPEG-2 encoder are the DCT module, the Quantization module and the Entropy encoder, the Wyner-Ziv codec will use the quantization table for MPEG-2 for I-frames and the Huffman tree used in such standard for the entropy encoder.

1.3 Slepian-Wolf

The work presented in [Slepian 73] about noiseless coding of correlated sources is to propose a codec structure where, for two independent but correlated signals X and Y that are encoded independently but conditionally decoded, the decoded data will fall in an admissible region. This was achieved by founding a set of admissible rate points, R_x and R_y, to code the data such that, for every $\epsilon > 0$ there exists for some $n = n(\epsilon)$ encoders $C_x(n, s_2, M_x)$, $C_y(n, s_1, M_y)$, and decoders $D_x(n, s_4, M_x, M_y)$, $D_y(n, s_3, M_x, M_y)$ with $M_x = \lfloor \exp(nR_x) \rfloor$, $M_y = \lfloor \exp(nR_y) \rfloor$, such that, $Pr[\{X^* \neq X\} \cup \{Y^* \neq Y\}] < \epsilon$ [Slepian 73].

Where n is the number of realizations, s_1, s_2, s_3, s_4 are the states of the switches in Figure 1.3, for this thesis it can be seen that the states of that switches are 0011 which is the reason that encoder C_x only depends of M_x and C_y depends of M_y and both decoders depend of M_x and M_y which are the probability functions of both sources X and Y.

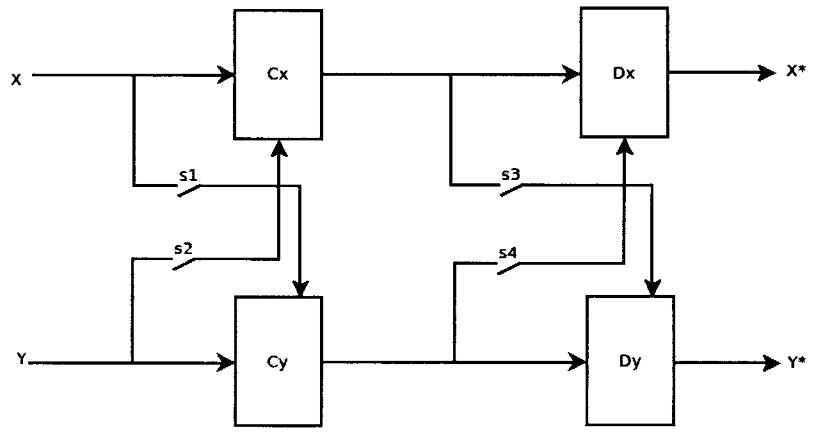


Figure 1.3: Configuration Options for the Codec

Probability functions M_x and M_y are in function of the entropy of the random variables, this is because for n realizations it is expected $np_x(1)$ realizations for event 1, $np_x(2)$ realizations for event 2, and so, where $p_x(i)$ is the probability for event i to occur.

It is well known that the total probability of a given sequence is the product of the individual probabilities for each realization in the sequence [Symes 04], then, for a long sequence for a random variable X it can be found that.

$$\begin{aligned} p_T &= p_x(1)^{np_x(1)}\, p_x(2)^{np_x(2)} \cdots p_x(A)^{np_x(A)} \\ &= \exp\left[np_x(1)\ln(p_x(1))\right] \cdots \exp\left[np_x(A)\ln(p_x(A))\right] \\ &= \exp\left[-nH(X)\right] \end{aligned}$$

These equations can be found in [Slepian 73] they are presented here in order to understand Slepian and Wolf work and justify the thesis problem.

Shannon Theorem says that the most optimal representation for a given data sequence, i.e., the minimum number of bits per symbol for the data to be interpreted without errors, is equal to the entropy of the sequence. Hence, if the entropy of a given random variable X is 3, then the optimal representation for X is 3 *bits/symbol*, therefore, $R(X) = H(X)$ in *bits/symbol*.

According to Slepian and Wolf work, the objective is to find an admissible region of values for R that, for a given random variable X, $R(X) \geq H(X)$.

Of course the last explanation only takes into account one source X, but Slepian and Wolf talk about two correlated sources X and Y coded independently but decoded conditionally, so the admissible region will be interpreted as a plane and an admissible rate point will be represented as a point in that plane. Slepian and Wolf found that, for a codec with switches values 0011 (Fig. 1.3) the admissible rate region R is presented in Fig. 1.4.

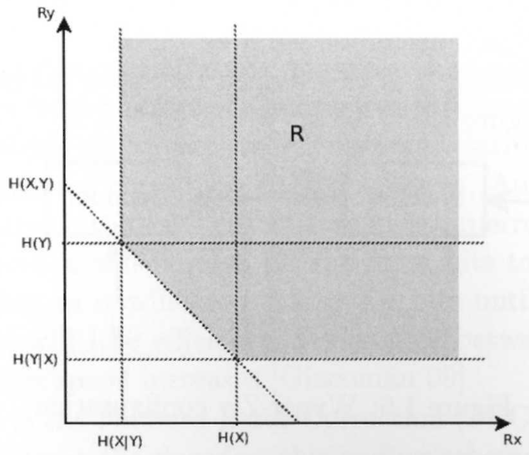


Figure 1.4: Admissible Region for 0011 codec

It is demonstrated that various correlated sources can be coded independently but decoded conditionally by selecting an admissible rate point (R_x, R_y) to represent each of the sources and so, a video signal can be represented as a group of frames that can be encoded in an intra-frame fashion (independently coded) and decoded in an inter-frame fashion (conditionally decoded).

1.4 Wyner-Ziv

Slepian and Wolf had demonstrated that two statistically dependent source signals can be encoded being unaware from each other and still getting to reconstruct the data by jointly decoding them. They demonstrated it by finiding two admissible rate points for each signal inside an admissible rate region defined in Figure 1.4. Wyner and Ziv contributed by proposing a side information sequence to increment the rate distortion performance. In contrast with Slepian-Wolf codec, which is a lossless codec [Girod 05], Wyner-Ziv codec is implemented on a lossy model.

As Wyner-Ziv introduced a lossy model in order to reduce the transmission rate, they introduced the side information sequence so the average distortion of the signal would be less than a d (error) value.

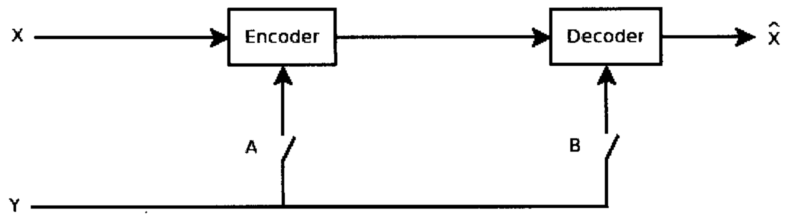


Figure 1.5: Wyner-Ziv codec configuration options

They consider a codec structure like in Figure 1.5 where switches A and B determine if the side information is known at the encoder, decoder, both or none of them, for the sake of simplicity and the fact that the video codec is based upon the characteristic that the side information is only known at the decoder, only the results for the case where only switch B is closed are presented here.

Knowing that only switch B is closed then the block diagram from Figure 1.5 is reduced to the one in Figure 1.6.

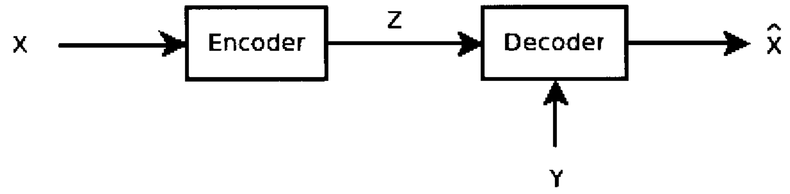


Figure 1.6: Wyner-Ziv configuration

If Figure 1.6 is analyzed it can be seen that the encoder will only use the source information X to generate Z but the decoder will need Z and the side information Y to generate the reconstructed data $\hat{X}$, hence, $\hat{X} = F_D(Y, F_E(X))$ [Wyner 76], where F_E and F_D are the functions for the encoder and decoder respectively.

When $d = 0$ it is easy to note that it is referring to the case where no loss of information is conveyed, so the next statement is true according to Slepian and Wolf.

$$R^*(0) = R_{X|Y}(0) = H(X|Y) \tag{1.6}$$

Where $R^*(0)$ in Eq. 1.6 is the rate for the configuration in Figure 1.6. And so, they demonstrate that, for $d > 0$.

$$R^*(0) \geq \lim_{d \to 0} R^*(d) \tag{1.7}$$

and that,

$$E\left[D\left(X, \hat{X}\right)\right] \leq d \tag{1.8}$$

where $D\left(X, \hat{X}\right)$ is the distortion function.

Hence, when $d \to 0$ the transmission rate will be lower than when $d = 0$, therefore, adding side information at the decoder will reduce the amount of data to be sent. Side information is normally a noisy version of the encoded data but different methods can be employed to generate this information such that, for better side information a better reconstruction will be obtained.

1.5 Turbocodec

Compression is important because the amount of data that have to be sent is reduced. Sending the compressed data through a transmission media without any kind of protection would yield errors in the decoding process. As data is compressed a single bit error would affect more than one coefficient. Then it is important to ensure that data will be received with the minimum BER (Bit Error Rate).

Even if digital transmission is less likely to be affected by noise in the channel as the decoder does not have to estimate the value of the incoming data, it just have to select the best value in a fixed number of discrete values [Giacomán 09], error correction coding have to be employed to correct errors generated during transmission.

There are two main error correction coding schemes, Automatic Repeat Request (ARQ), which is a technique that will detect transmission errors at the decoder, it will feedback the encoder and it will request for the error bits to be retransmitted. This coding scheme is reliable as it will keep asking for bits until the correct sequence is received. This method will lose efficiency as distance between encoder and decoder increase and transmission speed increases [Giacomán 09].

The second scheme is the Forward Error Correction (FEC), this scheme will try to correct most of the errors at the decoder, this coding scheme will not use a feedback from the decoder to the encoder. This method is clearly less efficient than the ARQ scheme as it will correct most of the errors while ARQ corrects all the transmission errors, but as distance and transmission speed increases ARQ will lose performance and so FEC will be preferred.

It is worth to know that this coding schemes are known as channel coding schemes. This error correction techniques are popular among transmission systems because their implementation uses a little power and few additional hardware is required [Giacomán 09].

From last discussion can be concluded that FEC schemes will be preferred over ARQ schemes as they are limited and does not suits real-time requirements [Giacomán 09]. Among the most common channel coding methodologies there can be found block codification, convolutional codification, data interleaving and concatenated codification [Giacomán 09].

Concatenated codification is popular as it has a really low error probability. An example of a classical concatenated codec with FEC scheme is presented in Figure 1.7.

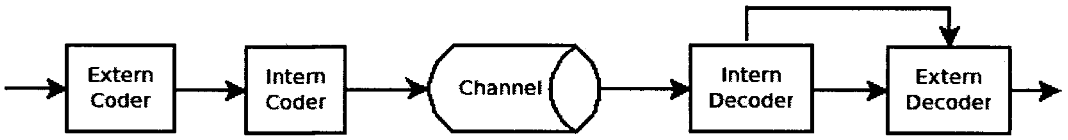


Figure 1.7: Classic concatenation FEC codec [Giacomán 09]

A common codec like the one in Figure 1.7 will consist in an external codec and an internal codec, the external one will normally be a Reed-Solomon coder/decoder, the internal coder will normally be implemented as a Trellis coder and the internal decoder is implemented with a Viterbi decoder [Giacomán 09].

Now that concepts about channel codification and its different implementations and schemes have been discussed, it has to be noted that the objective for channel coding is to achieve the shannon capacity. Shannon capacity is a channel capacity measure. Shannon have said that if the transmission rate is less than the channel capacity, it is possible to approach an error free communication considering a channel with additive white gaussian noise [Giacomán 09].

Turbocodes are channel codification techniques that are able to achieve the shannon capacity within 1 dB, they are normally implemented with an interleaved coder and an iterative decoder and they implement a FEC scheme for error correction.

This characteristics make the turbocodec an eligible module for the transmission module in the codec. They are efficient, fast and almost error free.

The codec implemented here has to be integrated with the turbocodec generated by Myriam Alanis Espinosa for her master thesis, "Implementation of 3GPP-LTE Turbo Codes", for better understanding of the turbocodec more information can be found in either [Giacomán 09], or [Espinosa 10].

1.6 Methodology

The codec block diagram is based in a block diagram proposed by Anne Aaron and Bernard Girod on [Aaron 04].

Implementation will be written in VHDL, and modules will be designed in a pipeline fashion in order to maintain the real-time operation of the codec, the codec must achieve a good compression rate such as a low bandwidth.

Some blocks are briefly explained to understand their functionality in the codec.

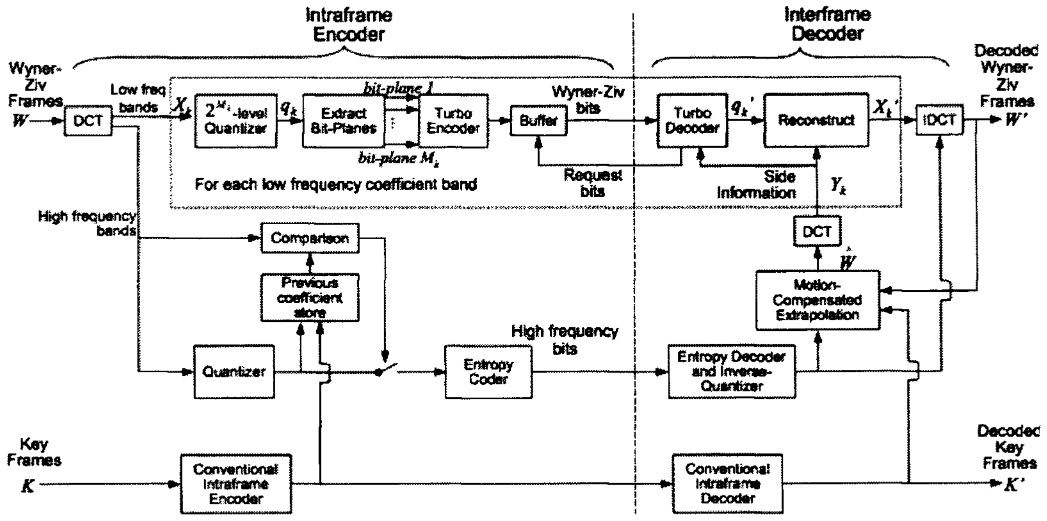


Figure 1.8: Aaron and Girod Block Diagram [Aaron 04]

DCT

Chapter 2 explains the Discrete Cosine Transform which is the first step for lossy compression, it decorrelates the 8×8 input matrices and stores most of the information in the first coefficients of the transformed matrix. The matrix generated by this module will have 64 values, an 8×8 output matrix, the first coefficient is known as DC coefficient while the rest are known as AC coefficients.

This knowledge is useful because each coefficient will follow a different data path depending if it is a DC or AC coefficient.

In Figure 1.8 this module is found at the beginning of the system, it is implemented with a recursive algorithm which is found to be optimal because the real-time constrain is met. As images have two dimensions the transform has to be a two dimensional transformation but the separability property of the DCT will let us use the recursive algorithm for the 1D DCT to obtain the 2D DCT of the 8×8 block.

2D IDCT module is also obtained with a recursive algorithm for the 1D IDCT of the decoded data.

Quantization

The matrix generated by the DCT module will group most of the coefficients in the upper left corner of the matrix, also, the first coefficient represents the average energy on that block. Also, the most important values are stored in the first coefficients and the contribution of the last coefficients will not drastically affect the final result.

Quantization will convert to zero those coefficients that will not be needed and this module is explained in Chapter 3.

There are two quantizer modules in Figure 1.8, the low frequencies band quantizer is marked as a 2^M-level quantizer while the high frequencies band is just marked as a quantizer.

In this thesis it is used the same quantizer structure for both quantizer block modules in Figure 1.8. The quantizer consists in a simple division and rounding algorithm using the quantization matrix for intrablocks in the MPEG-2 standard. In this project the quantizer is implemented in one whole module but it has to be separated between the low frequencies band coefficients and the high frequencies band coefficients.

The diagram in Figure 1.8 shows a block labeled as "Reconstruct" this block is inferred to have an inverse quantizer and an entropy decoder, which implementation will be discussed later in this chapter. The inverse quantizer is implemented taking advantage of the multipliers in the FPGA so the implementation will have a quantization coefficients memory and a multiplier.

Zig-zag reordering

For a lossless compression method to be more effective, the runs of zeros have to be extended, one of DCT characteristics is that low spatial frequencies are concentrated in the upper left corner of the matrix while the high spatial frequencies coefficients will be concentrated in the lower right corner of the matrix, hence, zig-zag scanning of the matrix will order the data from the lowest frequency coefficient to the highest frequency coefficient and so, runs of zeros will be extended.

In the block diagram in Figure 1.8 this scanning process is performed after the DCT and is the process that will separate the low frequencies band from the high frequencies band.

For this project, the zig-zag reordering is performed after the quantization process but it can be easily implemented after the DCT module, in order to accomplish this, the quantizer has to be modified so the data coming from the DCT process are quantized with the right quantization coefficients.

This module is implemented with a Moore FSM (Finite State Machine) to calculate the next memory address to be read, this process will save a memory reading as most implementations involve a look-up table that stores the order in which the quantized matrix is read, and also, the amount of hardware needed will be less as no look-up table will be used.

Zig-zag reordering will be also explained in Chapter 3.

Entropy Encoder

The entropy encoder is a lossless compression method, it is explained in Chapter 5, it takes the quantized coefficients and generates pairs of data known as (Run, Level) pairs, for each pair, the fist value represents the run-length of zeros before a non-zero value which will be the second value in the pair.

(Run, Level) pairs are entropy encoded with a tree algorithm developed by Huffman. This algorithm will generate variable length codes according with the probability of success of each (Run, Leve) pair.

Figure 1.8 shows one entropy encoder in the high frequencies band data path but one more entropy encoder is added to in the low frequencies band data path before the turbocodec module as most of the non-zero coefficients will be processed in the low frequencies band data path.

From DCT properties it is known that high frequency coefficient will be mostly zeros, hence, the entropy encoder in the high frequency data path will be only a RLE (Run Length Encoder) while the entropy encoder that will be added in the low frequency band is a RLE and Huffman encoder combination.

The Huffman encoder is implemented with a look-up table algorithm to find the coded word and a shift register to align the variable length codes into a single bitstream.

Entropy Decoder

The entropy decoder efficiency is the most important in the codec as it is the slowest module because it takes variable length codes as input and gives fixed length (Run, Level) pairs as output. The decoder, also known as VLD (Variable Length Decoder), has to align the variable length codes and match the bitstream with all possible (Run, Level) pairs. Hence, an efficient algorithm has to be found to reduce the number of matches also as the symbol memory size.

The entropy decoder is found in Figure 1.8 in the high frequencies band band data path and a decoder module will be added in the reconstruct block module. Like in the entropy encoder, the VLD in the high frequencies band data path will be a run length decoder but the VLD in the low frequencies band data path will be a Huffman decoder followed by a run length decoder.

The Huffman decoder is implemented with a MLBP (Maximum Likely Bit Pattern) algorithm that will be discussed in Chapter 5. This algorithm will decode a codeword per clock cycle so real-time constrain is maintained.

Chapter 2

DCT - Discrete Cosine Transform

The Discrete Cosine Transform is an orthogonal transformation and is found to be compared to the Karhunen-Loève Transform which is known to be optimal [Ahmed 74], for this reason the DCT is widely used for digital signal processing, data compression and filtering [Huang 09].

It is used in international standards such as JPEG, MPEG, H.261, H.263 [Huang 09], the DCT is the first step in data compression as it stores most of the information in the first coefficients.

First introduced by Ahmed et al. [Ahmed 74], the DCT has been object of interest for many researches in the area of digital signal processing because of its properties that maximize the compression rate for a given set of data. Because of this, research around fast algorithm implementations had been made in the last years and the motivation behind this research is that video can be represented as a sequence of frames and the DCT is the module that makes most of the computation during the compression stage, then fast algorithms have to be implemented in order to keep the real time implementation of the video codec.

For this codec a recursive algorithm for both, the DCT and the IDCT, were implemented but in order to be able to understand these algorithms the DCT itself has to be discussed.

2.1 One Dimensional DCT

Let us consider the Fourier transform definition.

$$X(\omega) = \sqrt{\frac{1}{2\pi}} \int_{-\infty}^{\infty} x(t)\, \mathrm{e}^{-j\omega t} dt \tag{2.1}$$

When $x(t)$ is only defined for $t \geq 0$ then the function $y(t)$ can be defined as. [Rao 90].

$$\begin{aligned} y(t) &= x(t)\,, \; t \geq 0 \\ &= x(-t)\,, \; t \leq 0 \end{aligned}$$

Then, the Fourier transform of $y(t)$ can be found using Eq. 2.1.

$$\begin{aligned} Y(\omega) &= \sqrt{\frac{1}{2\pi}}\left(\int_0^{\infty} x(t)\,\mathrm{e}^{-j\omega t}dt + \int_{-\infty}^{0} x(-t)\,\mathrm{e}^{-j\omega t}dt\right) \\ &= \sqrt{\frac{1}{2\pi}}\int_0^{\infty} x(t)\,[\mathrm{e}^{-j\omega t}+\mathrm{e}^{j\omega t}]dt \\ &= \sqrt{\frac{2}{\pi}}\int_0^{\infty} x(t)\cos(\omega t)\,dt \end{aligned} \tag{2.2}$$

Eq. 2.2 is known as the Fourier Cosine Transform (FCT) which was obtained by mirroring the function $x(t)$ over the y axis, this generates an even function as this condition is found to be true for $y(t)$.

$$y(t) = y(-t)$$

Then it is obvious that Eq. 2.2 only applies for even functions. There are also some alterations for the discrete version of the FCT but they will be defined later in this chapter.

As the title of this chapter says, it is for the interest of this project that discrete version of the FCT is defined. If we analyze Eq. 2.2 it is found that the kernel of the transformation will be $\cos(\omega t)$.

Let us consider the case where the kernel is defined for $\omega_m = 2\pi m\delta f$ and $t_n = n\delta t$, then it is easy to find that the kernel will be expressed as next.

$$K(\omega_m, t_n) = \cos(2\pi m\delta f n\delta t) \tag{2.3}$$

Then, if $\delta f\delta t = 1/2N$, the kernel will depend on the discrete variables m and n which are samples in frequency and time respectively, also N represents the number of samples taken. By knowing this, the kernel will be declared as in Eq. 2.4 [Rao 90].

$$K(m,n) = \cos\left(\frac{mn}{N}\right) \tag{2.4}$$

The matrix generated by Eq. 2.4 will be known as a $N \times N$ transformation matrix. If this transformation matrix is multiplied with a vector, $\mathbf{x} = \{x_0, x_1, \cdots, x_N\}^T$ the resulting vector will be a transformed vector, $\mathbf{X} = \{X_0, X_1, X_2, \cdots, X_N\}$. Then a discrete cosine transform can be defined as.

$$X(m) = \sum_{n=0}^{N} x(n)\cos\left(\frac{mn}{N}\right) \tag{2.5}$$

Eq. 2.5 was first introduced by Kitajima in 1980 with the name of Symmetric Cosine Transform [Rao 90] and it is, in fact, a DCT definition, but in 1984 Wang classified four types of DCT that are defined in the next discussion.

Let $x(n)$ be a data sequence of N elements and $X(k)$ be the transformed sequence of the original data.

There exist 8 different versions of the DCT each used in different kind of applications. Four of them are presented here with their respective inverse transformations [Rao 90].

DCT-I

Forward

$$X(m) = \sqrt{\frac{2}{N}} k_m \sum_{n=0}^{N-1} k_n x(n) \cos\left(\frac{mn\pi}{N}\right); \qquad m,n = 0,...,N-1 \qquad (2.6)$$

Inverse

$$x(n) = \sqrt{\frac{2}{N}} k_n \sum_{m=0}^{N-1} k_m X(m) \cos\left(\frac{mn\pi}{N}\right); \qquad m,n = 0,...,N-1 \qquad (2.7)$$

DCT-II

Forward

$$X(m) = \sqrt{\frac{2}{N}} k_m \sum_{n=0}^{N-1} x(n) \cos\left(\frac{(2n+1)m\pi}{2N}\right); \qquad m,n = 0,...,N-1 \qquad (2.8)$$

Inverse

$$x(n) = \sqrt{\frac{2}{N}} \sum_{m=0}^{N-1} k_m X(m) \cos\left(\frac{(2n+1)m\pi}{2N}\right); \qquad m,n = 0,...,N-1 \qquad (2.9)$$

DCT-III

Forward

$$X(m) = \sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} k_n x(n) \cos\left(\frac{(2m+1)n\pi}{2N}\right); \qquad m,n = 0,...,N-1 \qquad (2.10)$$

Inverse

$$x(n) = \sqrt{\frac{2}{N}} k_n \sum_{m=0}^{N-1} X(m) \cos\left(\frac{(2m+1)n\pi}{2N}\right); \qquad m,n = 0,...,N-1 \qquad (2.11)$$

DCT-IV

Forward

$$X\left(m\right)=\sqrt{\frac{2}{N}}\sum_{n=0}^{N-1}x\left(n\right)\cos\left(\frac{\left(2n+1\right)\left(2m+1\right)\pi}{2N}\right);\qquad m,n=0,...,N-1 \quad (2.12)$$

$$x\left(n\right)=\sqrt{\frac{2}{N}}\sum_{m=0}^{N-1}X\left(m\right)\cos\left(\frac{\left(2n+1\right)\left(2m+1\right)\pi}{2N}\right);\qquad m,n=0,...,N-1 \quad (2.13)$$

DCT-II and DCT-III are the most commonly used in digital signal processing areas and are known as DCT and IDCT respectively [Rao 90].

2.2 Two Dimensional DCT

Images can be interpreted as a set of values arranged in a matrix where each value represents a pixel, these set of values have to be transformed with a two dimensional version of the DCT.

It was mentioned that the DCT is an orthogonal transform, among the characteristics of orthogonal transformations, the most useful when working with multidimensional transformations is the one which says that for multidimensional sets of data, the DCT will be obtained by finding the transform of the first dimension, then, for the semi-transformed set of data, the transform for the second dimension is obtained, and so on until all dimensions in the set of data are transformed.

For two dimensional sets of data, the DCT would be obtained by finding the 1D DCT for each row in the matrix and then finding the 1D DCT of each column of the semi-transformed matrix [Rao 90].

Let $x\left(m,n\right)$ be an $N\times N$ matrix and $X\left(u,v\right)$ be its two dimensional DCT. Then 2D DCT is defined as.

$$X\left(u,v\right)=\frac{2c\left(u\right)c\left(v\right)}{N}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}x\left(m,n\right)\cos\left(\frac{\left(2m+1\right)u\pi}{2N}\right)\cos\left(\frac{\left(2n+1\right)v\pi}{2N}\right) \quad (2.14)$$

It is also necessary to know the inverse function so 2D IDCT is defined as.

$$x\left(m,n\right)=\frac{2}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}c\left(u\right)c\left(v\right)X\left(u,v\right)\cos\left(\frac{\left(2m+1\right)u\pi}{2N}\right)\cos\left(\frac{\left(2n+1\right)v\pi}{2N}\right) \quad (2.15)$$

Where.

$$\begin{aligned} c\left(k\right) &= \frac{1}{\sqrt{2}}\text{ if }k=0,\\ &= 1\text{ otherwise}\end{aligned}$$

In Eq. 2.14 is observed that, for an $N \times N$ matrix, one dimensional DCT of N points is calculated for each row and column. For the sake of simplicity it will be assumed that $M = N$, M is associated to the number of rows while N is associated with the number of columns, then, the two dimensional DCT definition will yield M N-point one dimensional DCT along the rows and N M-point one dimensional DCT along the columns [Rao 90].

Each 1D DCT will yield N multiplications and $N-1$ additions. For an $N \times N$ matrix the number of multiplications will be $2N^2$ and the number of additions will be $2\left(N-1\right)^2$, for an 8×8 matrix there are necessary 128 multiplications and 98 additions.

Knowing this, it is necessary to find a fast algorithm to calculate the 2D DCT for 8×8 blocks. Next discussion will talk about a recursive algorithm for the two dimensional DCT and the last one will talk about a recursive algorithm for the two dimensional IDCT.

2.2.1 Recursive Algorithm for 2D - DCT

A recursive algorithm is proposed by [Zhijin 96] and a VHDL implementation for one dimensional DCT is presented by [Mendoza 06]. It was said that a two dimensional DCT can be obtained by applying a one dimensional DCT to the rows and then to the columns of the semitransformed matrix.

This algorithm let the DCT be obtained with half of the multiplications and with no data shifts [Zhijin 96] which represents a significant reduction in computation, also the hardware implementation is in a pipeline so when all the stages in the pipeline are full a new coefficient will be obtained per clock cycle.

Eq. 2.8 is recalled but for the sake of simplicity $\sqrt{\frac{2}{N}}$ and k_m are omitted and then included at the end of the algorithm, then this new expression is obtained.

$$X\left(k\right) = \sum_{n=0}^{N-1} x\left(n\right) \cos\left(\frac{\left(2n+1\right)k\pi}{2N}\right) \tag{2.16}$$

This same equation can be expressed as a matrix equation where the cosine function is represented as a transformation matrix with $N \times N$ elements and $x\left(n\right)$ is a column vector with the data to be transformed.

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \vdots \\ X_{N-1} \end{bmatrix} = \begin{bmatrix} C_{2N}^0 & C_{2N}^0 & \cdots & C_{2N}^0 \\ C_{2N}^1 & C_{2N}^3 & \cdots & C_{2N}^{2(N-1)+1} \\ C_{2N}^2 & C_{2N}^4 & \cdots & C_{2N}^{(2(N-1)+1)2} \\ \vdots & \vdots & \ddots & \vdots \\ C_{2N}^{N-1} & C_{2N}^{2(N-1)} & \cdots & C_{2N}^{(2(N-1)+1)(N-1)} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{bmatrix} \tag{2.17}$$

Where $C_{2N}^a = \cos\left(\frac{a\pi}{2N}\right)$

Let us assume that $N = 8$, DCT coefficients will be obtained using Eq. 2.16.

$$X_0 = C_{16}^0 \left(x_0 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7\right)$$

$$X_1 = C_{16}^1(x_0 - x_7) + C_{16}^3(x_1 - x_6) + C_{16}^5(x_2 - x_5) + C_{16}^7(x_3 - x_4)$$
$$X_2 = C_{16}^2(x_0 + x_7) + C_{16}^6(x_1 + x_6) - C_{16}^6(x_2 + x_5) - C_{16}^2(x_3 + x_4)$$
$$X_3 = C_{16}^3(x_0 - x_7) - C_{16}^7(x_1 - x_6) - C_{16}^1(x_2 - x_5) - C_{16}^5(x_3 - x_4)$$
$$X_4 = C_{16}^4(x_0 + x_7) - C_{16}^4(x_1 + x_6) - C_{16}^4(x_2 + x_5) + C_{16}^4(x_3 + x_4)$$
$$X_5 = C_{16}^5(x_0 - x_7) - C_{16}^1(x_1 - x_6) + C_{16}^7(x_2 - x_5) + C_{16}^3(x_3 - x_4)$$
$$X_6 = C_{16}^6(x_0 + x_7) - C_{16}^2(x_1 + x_6) + C_{16}^2(x_2 + x_5) - C_{16}^6(x_3 + x_4)$$
$$X_7 = C_{16}^7(x_0 - x_7) - C_{16}^5(x_1 - x_6) + C_{16}^3(x_2 - x_5) - C_{16}^1(x_3 - x_4)$$

Hence, Eq. 2.16 can be rewritten to the next expression.

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} \left[x(n) + (-1)^k x(N-1-n)\right] cos\left(\frac{(2n+1)k\pi}{2N}\right) \tag{2.18}$$

The next two expressions can be derived from Eq. 2.18.

$$g(n) = x(n) + x(N-1-n) \tag{2.19}$$
$$h(n) = x(n) - x(N-1-n) \tag{2.20}$$

It can be seen from last analysis that even coefficients are related with Eq. 2.19 while odd coefficients are related with Eq. 2.20, therefore the next equations can be obtained from Eq. 2.16.

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} g(n)\cos\left(\frac{(2n+1)2k\pi}{2N}\right) \tag{2.21}$$

$$X(2k+1) = \sum_{n=0}^{\frac{N}{2}-1} h(n)\cos\left(\frac{(2n+1)(2k+1)\pi}{2N}\right) \tag{2.22}$$

Eq. 2.21 is an $N/2$-point DCT because.

$$C_{2N}^{(2n+1)2k} = C_N^{(2n+1)k} = C_{2\frac{N}{2}}^{(2n+1)k}$$

Therefore,

$$G(k) = \sum_{n=0}^{\frac{N}{2}-1} g(n)\cos\left(\frac{(2n+1)k\pi}{N}\right) \tag{2.23}$$

The objective is to find another DCT related with Eq. 2.22 and to achieve this objective a known trigonometric identity is used.

$$\cos(A \pm B) = \cos(A)\cos(B) \mp \sin(A)\sin(B) \tag{2.24}$$

Therefore, the next expression can be obtained from Eq. 2.22 and Eq. 2.24.

$$X(2k+1) + X(2k-1) = 2\sum_{n=0}^{\frac{N}{2}-1} h(n)\cos\left(\frac{(2n+1)k\pi}{N}\right)\cos\left(\frac{(2n+1)\pi}{2N}\right) \tag{2.25}$$

A new expression is obtained from Eq. 2.25.

$$b(n) = 2h(n)\cos\left(\frac{(2n+1)\pi}{2N}\right) \tag{2.26}$$

DCT of Eq. 2.26 can be obtained with the next expression.

$$B(k) = \sum_{n=0}^{\frac{N}{2}-1} b(n)\cos\left(\frac{(2n+1)k\pi}{N}\right) \tag{2.27}$$

It is clear that Eq. 2.27 is the same as Eq. 2.25, hence.

$$X(2k+1) = B(k) - X(2k-1) \tag{2.28}$$

Let us consider the case when $k = 0$.

$$\begin{aligned} B(0) &= \sum_{n=0}^{\frac{N}{2}-1} b(n)\cos(0) \\ &= 2\sum_{n=0}^{\frac{N}{2}-1} h(n)\cos\left(\frac{(2n+1)\pi}{2N}\right) \\ &= 2X(1) \end{aligned}$$

Then, for the case where $k = 0$, $X(1) = B(0)/2$.

Eq. 2.23 and Eq. 2.28 show how an N-point DCT can be split into two $N/2$-point DCTs with this recursive algorithm.

Hardware Implementation

The algorithm is implemented in VHDL and it was designed in a pipeline architecture. The block diagram used is presented in Figure 2.1. Only half of the architecture is shown as the other half is exactly the same as the one in the figure.

The pipeline consists in 10 stages, each one activated each clock cycle, except for the second one which is activated each eight cycles.

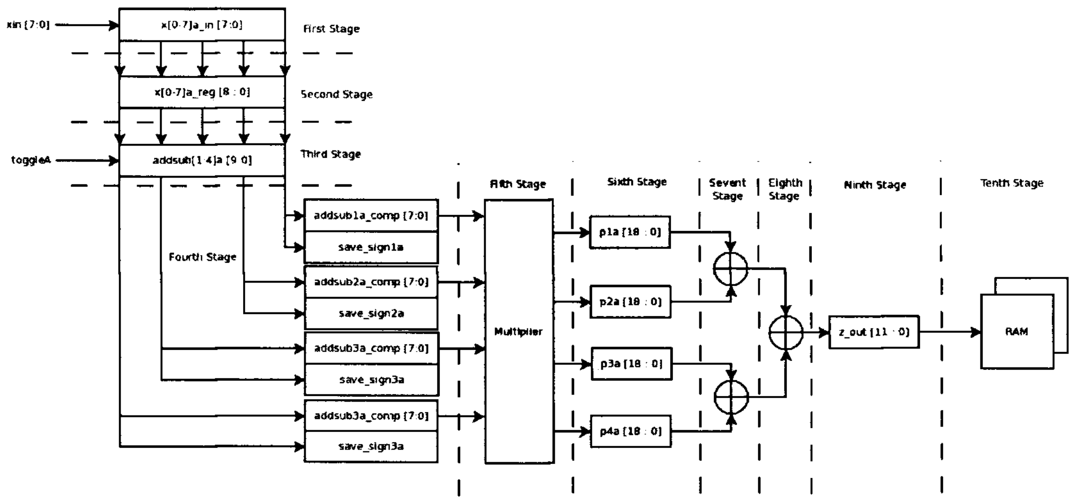


Figure 2.1: Block Diagram for DCT Implementation

A brief description of each stage is presented.

First Stage. A shift register is implemented that will take one input value per clock cycle

Second Stage. This stage is activated each eight cycles, when the eighth cycle is reached, it takes the values in the shift register and store them for another eight cycles.

Third Stage. Data is added or subtracted in order to calculate the expressions in Eq. 2.19 and Eq. 2.20.

Fourth Stage. A sign adjustment according the sign is done in this stage, this is because the module is designed with unsigned values.

Fifth Stage. This stage takes the values $g(n)$ and $h(n)$ and multiplies them with the corresponding DCT coefficients.

Sixth Stage. This is another sign adjustment, it takes the multiplier output and decides if its sign has to be changed.

Seventh and Eighth Stages. This is where the final coefficient value will be obtained, it will add the multiplied values from the fifth stage.

Ninth Stage. A register will be added to avoid glitches.

Tenth Stage. This stage stores the value in a RAM. There are two memories so one can be written while the other is being read.

As it was said before, this block diagram will calculate the semi-transformed matrix by transforming the rows of the data matrix. The second part of the DCT has the same architecture but it will transform the columns of the semi-transformed matrix.

It has to be noted that the pipeline will not give valid values until the pipeline is full, hence, a delay will be added and the first valid value will be obtained after 96 clock cycles, after that number of clock cycles the DCT will give a new valid value each clock cycle and a valid 8×8 block each 64 clock cycles.

The DCT operation is one of the most complex modules in the system because the number of multiplication and additions that have to be performed in order to obtain the transformed data.

It is important to know the mathematical complexity of the module because the amount of operations performed have direct impact in the amount of time required to obtain a valid output, even if the module is implemented with a pipeline and a new coefficient is obtained each clock cycle, the more optimum the algorithm the less operations required and the less number of pipeline stages.

Also, as this module is one of the most complex in the system, it will be used as a reference to determine how complex is the system itself.

Table 2.1 and Table 2.2 compare the mathematical complexity of the algorithm proposed in this work with other algorithms already implemented. Table 2.1 shows the comparative table between the different 1D DCT algorithms for input vectors of 8 values as the recursive algorithm presented propose an algorithm for the 1D DCT and uses the separability property of the DCT to obtain the 2D DCT. Table 2.2 will compare the amount of operations performed between the algorithm used here and other algorithms using input matrices of 8×8 values.

Algorithm	Number of Multiplications	Number of Additions
Recursive DCT	32	56
Lee et al. [Lee 94]	12	29
Cvetkovié et al. [Cvetkovié 92]	12	29
Chen et al. [Chen 77]	16	26

Table 2.1: Comparative table between 1D DCT algorithms

Algorithm	Number of Multiplications	Number of Additions
Recursive DCT	512	896
Row-Column	192	464
Chan et al. [Chan 91]	144	464
Cho et al. [Cho 91]	94	466

Table 2.2: Comparative table between 2D DCT algorithms

From Table 2.1 and Table 2.2 it can be seen that the algorithm proposed is the most mathematical complex algorithm but the pipeline will compute various DCT coefficients each clock cycle so real-time constrain is maintained. This module also represents a bottleneck because it is the first module in the system, it has to be fast enough to compute the transformed coefficients while it keeps getting data from the preprocessed video.

2.2.2 Recursive Algorithm for 2D - IDCT

A recursive algorithm is proposed by [Lee 84], the algorithm is similar to the recursive DCT one so the hardware implementation is almost the same but with few more stages. The algorithm is explained below.

Eq. 2.9 is taken as base equation but $\sqrt{\frac{2}{N}}$ is omitted and then included at the end of the procedure, also, next equality will be assumed.

$$k_m X(m) = \hat{X}(k)$$

Then, the analysis begins with the next expression which is the IDCT definition.

$$x(n) = \sum_{k=0}^{N-1} \hat{X}(k) \cos\left(\frac{(2n+1)k\pi}{2N}\right) \tag{2.29}$$

From Eq. 2.19 and Eq. 2.20 it can be inferred that.

$$x(n) = \frac{g(n) + h(n)}{2} \tag{2.30}$$

$$x(N-1-n) = \frac{g(n) - h(n)}{2} \tag{2.31}$$

From [Zhijin 96] it can be found that $g(n)$ is related with the even coefficients while $h(n)$ is related with the odd coefficients so the next two expressions can be easily found.

$$g(n) = \sum_{k=0}^{\frac{N}{2}-1} \hat{X}(k) \cos\left(\frac{(2n+1)2k\pi}{2N}\right) \tag{2.32}$$

$$h(n) = \sum_{k=0}^{\frac{N}{2}-1} \hat{X}(2k+1) \cos\left(\frac{(2n+1)(2k+1)\pi}{2N}\right) \tag{2.33}$$

Eq. 2.32 is clearly an IDCT of $N/2$ points so an expression for Eq. 2.31 that looks like Eq. 2.34 has to be found to represent another IDCT of $N/2$ points.

$$h'(n) = \sum_{k=0}^{\frac{N}{2}-1} H(k) \cos\left(\frac{(2n+1)k\pi}{N}\right) \tag{2.34}$$

The next trigonometric identity is used to found the expected expression.

$$2\cos(A)\cos(AB) = \cos(A+AB) + \cos(A-AB) \tag{2.35}$$

The next expression is proposed using Eq. 2.33.

$$\begin{aligned} 2\cos\left(\frac{(2n+1)\pi}{2N}\right) h(n) &= 2\sum_{k=0}^{\frac{N}{2}-1} \hat{X}(2k+1) \times \\ &\left[\cos\left(\frac{(2n+1)\pi}{2N}\right)\cos\left(\frac{(2n+1)(2k+1)\pi}{2N}\right)\right] \end{aligned} \tag{2.36}$$

Eq. 2.38 is derived from Eq. 2.35.

$$2\cos\left(\frac{(2n+1)\pi}{2N}\right)h(n) = \sum_{k=0}^{\frac{N}{2}-1}\hat{X}(2k+1)\cos\left(\frac{(2n+1)k\pi}{N}\right) + \sum_{k=0}^{\frac{N}{2}-1}\hat{X}(2k+1)\cos\left(\frac{(2n+1)(k+1)\pi}{N}\right) \tag{2.37}$$

First half of the right part of Eq. 2.38 is an IDCT of $N/2$ points of the odd DCT coefficients, the second half can be rewritten as.

$$\sum_{k=0}^{\frac{N}{2}-1}\hat{X}(2k+1)\cos\left(\frac{(2n+1)(k+1)\pi}{N}\right) = \sum_{l=1}^{\frac{N}{2}-1}\hat{X}(2l-1)\cos\left(\frac{(2n+1)l\pi}{N}\right) \tag{2.38}$$

If $\hat{X}(-1) = 0$ then,

$$\sum_{l=1}^{\frac{N}{2}-1}\hat{X}(2l-1)\cos\left(\frac{(2n+1)l\pi}{N}\right) = \sum_{k=0}^{\frac{N}{2}-1}\hat{X}(2k-1)\cos\left(\frac{(2n+1)k\pi}{N}\right) \tag{2.39}$$

This last expression can be substituted in Eq. 2.38 to get the final expression.

$$h(n) = \sum_{k=0}^{\frac{N}{2}-1}H(k)\cos\left(\frac{(2n+1)k\pi}{N}\right) \tag{2.40}$$

Where,

$$H(k) = \hat{X}(2k+1) + \hat{X}(2k-1)$$

Substituting Eq. 2.40 and Eq. 2.32 in Eq. 2.30 and Eq. 2.31 the N-point IDCT can be obtained with two $N/2$-point IDCTs.

Hardware Implementation

Hardware implementation is similar to the DCT implementation one, the only difference is that a second multiplier stage is added as Eq. 2.30 and Eq. 2.31 require $h(n)$ to be divided by another cosine fuction. This constant could be added to the kernel coefficient but that would mean that 8 IDCT coefficients could be obtained in 4 clock cycles, this is clearly more efficient than obtaining one coefficient per cycle but the second stage of the pipeline is activated each 8 cycles so the pipeline should have to wait for that cycles to finish.

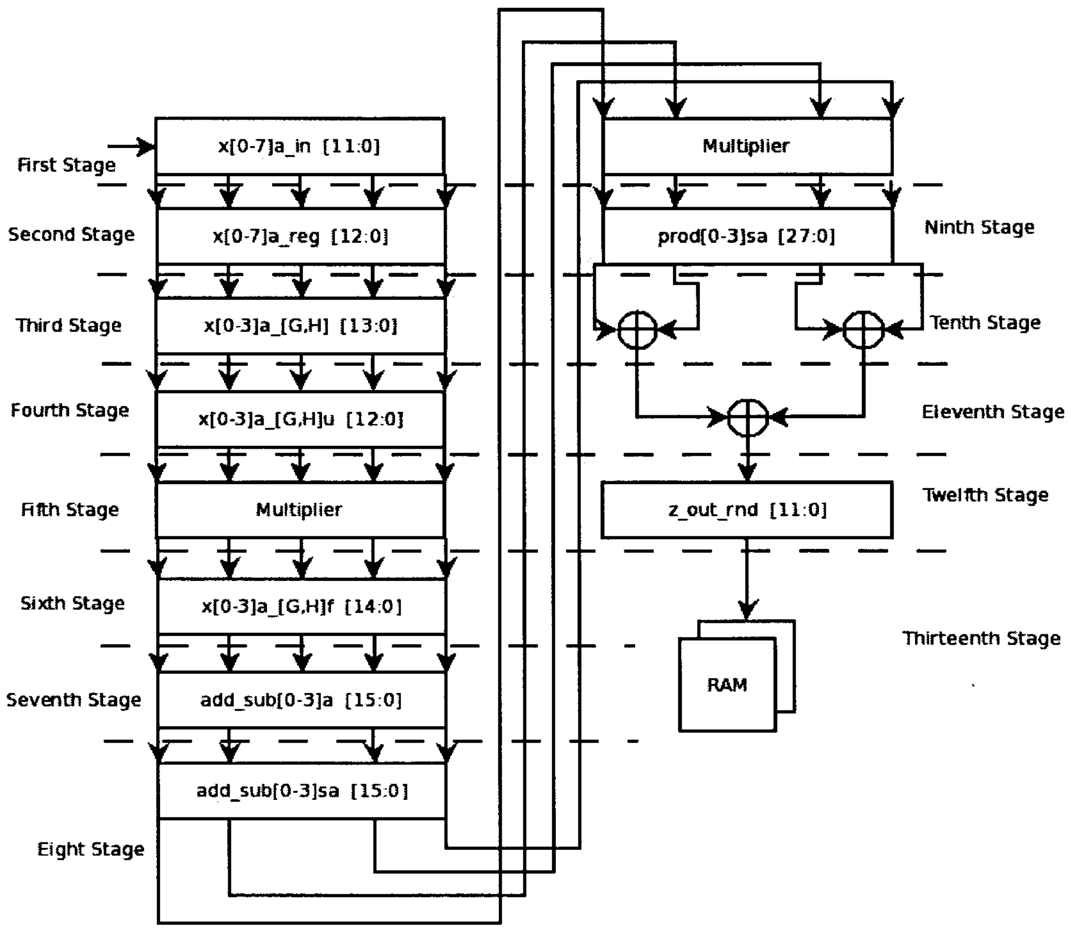


Figure 2.2: Block Diagram for IDCT Implementation

The block diagram and an explanation of each stage is presented in Figure 2.2. As in DCT hardware implementation section, only half of the block diagram is presented because the second half is exactly the same but taking the column values of the semi-transformed matrix in the RAM memory.

First Stage. This is a shift register which inputs are the DCT coefficients.

Second Stage. This stage is activated each 8 cycles and stores the coefficients in the shift register so it can keep taking new values while the rest of the pipeline is transforming the actual values.

Third Stage. $G(k)$ and $H(k)$ are split.

Fourth Stage. This is a sign adjustment for $H(k)$ to obtain $\hat{H}(k)$.

Fifth Stage. $H(k)$ will be multiplied by $1/C_{2N}^{2k+1}$ to obtain $\hat{H}(k)$

Sixth Stage. Another sign adjustment for the $\hat{H}(k)$ values and a store register for the $G(k)$ values.

Seventh Stage. This stage adds or subtracts values from $G(k)$ and $\hat{H}(k)$ that will be used to obtain the results of Eq. 2.30 and Eq. 2.31.

Eighth Stage. This is a sign adjustment for the multipliers, it worths to remember that this architecture is designed with unsigned data so a sign adjustment is needed for multiplications.

Ninth Stage. Data is multiplied with the cosine coefficients.

Tenth Stage. This is the last sign adjustment before resulting values are added to form the final value.

Eleventh and Twelfth Stages. Adder for the resulting values to form the final semi-transformed value.

Thirteenth Stage. This stage stores the semi-transformed matrix in a RAM memory so they can be used in the second half of the IDCT.

The complete transformation will be calculated with the same fashion as the DCT, first the row values will be calculated to form a semi-transformed matrix, and then the column values will be used to find the final transformed matrix.

This implementation has a pipeline structure so there will be also a number of not valid values before a valid value can be obtained, for this section each part of the implementation consume 20 clock cycles and the RAM memory section uses 64 clock cycles, then, for this module to give a valid value it is needed 104 clock cycles, after that number of cycles a new valid value will be obtained per cycle.

The IDCT is the most complex module in the system, it requires more operations to be completed than the DCT so this module will be another reference for the amount of complexity of the whole system, hence, any optimization regarding the complexity of the system has to start with this module.

IDCT	Number of Multiplications	Number of Additions
1D IDCT	96	56
2D IDCT	1536	896

Table 2.3: Mathematical complexity of the IDCT algorithm

Table 2.3 shows the number of multiplications and additions to complete an 8×8 block reconstruction. Results about the complexity of the IDCT algorithm is not commonly documented as it is directly related to the DCT so any optimization performed on the DCT algorithm will optimize the IDCT. Hence, Table 2.3 only shows the complexity of the algorithm presented in this thesis.

Chapter 3

Quantization and Zig-Zag Reordering

It was said before that the Discrete Cosine Transformation decorrelates the 8×8 data block and concentrates most of the energy of the original block in the upper left coefficients of the transformed matrix [Haskell 97], then most coefficients are not going to be needed to be sent as they do not really contribute while doing the inverse transformation process, and so, they can be considered with value of zero.

Because of this a quantization process is necessary to reduce the number of coefficients that are coded.

The next step is to scan the quantized matrix in a way that coefficients are arranged from the lowest spatial frequency value to the highest and this is done with a zig-zag scan that will also increase the runs of zeros.

3.1 Quantization

Quantization is done the same fashion as JPEG and MPEG-2 for Intra-frames standards, DC coefficient (the coefficient in the upper left corner) are difference quantized with the last DC coefficient [Haskell 97] and the rest of the coefficients are quantized with their actual value.

The quantization process is normally done by taking the DCT coefficient value and dividing it by a constant value defined by a quantization matrix and rounding it to the nearest integer value [Haskell 97]. This will led to the next expression.

$$q[i][j] = round\left(\frac{D[i][j]}{Q[i][j]}\right) \tag{3.1}$$

Where $D[i][j]$ is the DCT coefficient value located in the i'th row and the j'th column and $Q[i][j]$ is the quantization value in the quantization matrix in the same position.

There exist different quantization matrixes for different applications, the one used for this codec is the MPEG-2 quantization matrix for Intrablocks that is defined in Table 3.1.

8	16	19	22	26	27	29	34
16	16	22	24	27	29	34	37
19	22	26	27	29	34	34	38
22	22	26	27	29	34	37	40
22	26	27	29	32	35	40	48
26	27	29	32	35	40	48	58
26	27	29	34	38	46	56	69
27	29	35	38	46	56	69	83

Table 3.1: MPEG-2 Quantization Table

The problem with Eq. 3.1 is that there exists no division module in the FPGA where the module is intended to be implemented, then a division module has to be designed and implemented to obtain a valid quantized value.

3.1.1 Division Algorithm

Computational division is a complex operation that requires multiple clock cycles to obtain a valid quotient, the algorithm that will be used is known as "Restoration Division" and it calculates bit per bit using the algorithm in Figure 3.1(b) that is an algorithm similar to the one proposed by [Stallings 02].

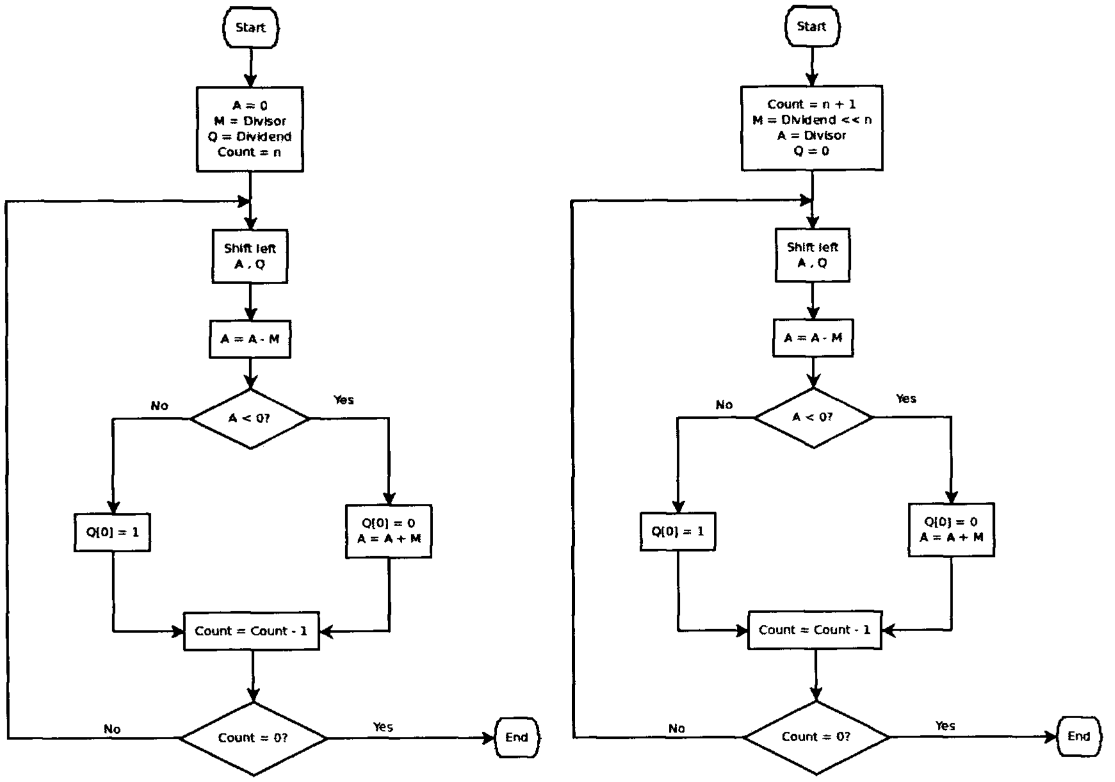
The algorithm proposed by [Stallings 02] does the division using the divisor and reminder, each step it shifts the reminder and quotient, then it compares the reminder with the divisor and if the divisor is greater than the reminder then the less significant bit of the quotient will be set as 0, if the reminder is greater than the divisor then the quotient will be set as 1; the algorithm loops n times, where n is the number of bits wanted to represent the quotient, when that number of loops are completed the quotient and the reminder are obtained. The algorithm presented in [Stallings 02] is shown in Figure 3.1(a).

The algorithm used in this project was almost the same but the dividend is shifted n spaces to the left at the beginning of the algorithm and each cycle the divisor is shifted one time to the left and then compared with the dividend, if the divided is greater than the divisor the less significant bit of the quotient is set to 0 and if the divisor is greater then it is set to 1 at the end the final value will be rounded depending the last bit which is the first decimal value, if it is 1 then the result will be rounded upwards and if it is 0 then it will rounded downwards.

This kind of algorithms are known as slow division algorithms and they come from the expression in Eq. 3.2

$$D = Q * V + R \tag{3.2}$$

Where D is the dividend, Q the quotient, V the divisor and R the reminder [Stallings 02].



(a) [Stallings 02] algorithm (b) Algorithm used in this codec

Figure 3.1: Divisor algorithms

Hardware Implementation

There are two ways to implement the divider algorithm, one of them is by processing each coefficient individually using one shift register and one comparator and multiple cycles to obtain the quantized value, the other way is to implement a pipeline of dividers so multiple values can be calculated in a clock cycle. This last implementation needs more hardware but it satisfies the real time requirement for the codec.

Then the pipeline implementation is done using 9 dividers where each divider is implemented using the implementation in Figure 3.2.

The divider has 4 inputs, N which is the dividend, D which is the divisor, Q is the quotient and *sign* the original sign of the numerator.

Following the algorithm in Figure 3.1(b), the dividend is shifted one time to the left, the divisor is shifted 8 times since it is the number of bits to be obtained at the end, the quotient is shifted one time to the left so last bit can be updated according to Figure 3.1(b) algoritm and the sign is passed as it is for future usage. Then the dividend and divisor will be subtracted to know which one is greater, if the dividend is grater then the last bit of the quotient will have the value of 1 and if the divisor is greater then it will take the value of 0. The dividend will have its original value shifted

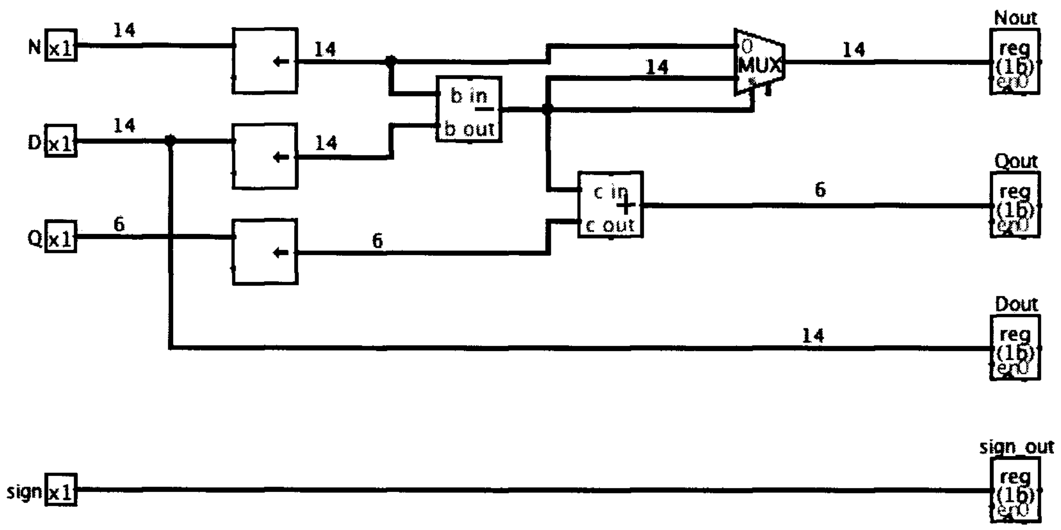


Figure 3.2: Divider Architecture

one time to the left if the divisor was grater and the subtracted value otherwise.

When the nine stages are completed the final quotient is adjusted depending the original sign of the dividend and 9 bits, 8 integer bits and 1 decimal bit, are generated, the last step into quantization is to round this value to the nearest integer.

The quantization process is the least complex module in the system, it only implements two adder/subtracter modules in each diver, as the quantizer is in a pipeline architecture there is going to be 18 adder/subtracter modules (9 dividers in the architecture) which will be also the number of operations required to obtain the quantized value.

3.2 Zig-Zag Reordering

Once DCT values are quantized most of the coefficients will have zero values and so they do not have to be sent, instead of that, a run-length of zeros value can be sent, in fact, this is how quantized values are sent, for each non-zero value, a run-length of zeros, that precedes the non-zero value, is sent. Zig-zag reordering increases the runs of zeros as this pattern makes the coefficients decrease in size [Lakhani 03].

The reordering pattern is presented in Table 3.2

Grosse et al. [Grosse 97] had demonstrated that there is a way to implement the zig-zag reordering with a state machine with 4 inputs representing the limits of the matrix. The method used for this section will be a Moore FSM but with different parameters to control the next state.

It can be seen from Table 3.2 that there are only four states, the one where only index i is incremented such as the transition from 0 to 1, the one where index i is decremented but index j is incremented such as the transition from 1 to 2, the state where only index j is incremented such like the step from 2 to 3 and the last one where

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Table 3.2: Zigzag reordering [CCITT 93]

index i is incremented and index j is decremented like the transition from 3 to 4. Also it has to be noted that increments and decrements have the value of one each time.

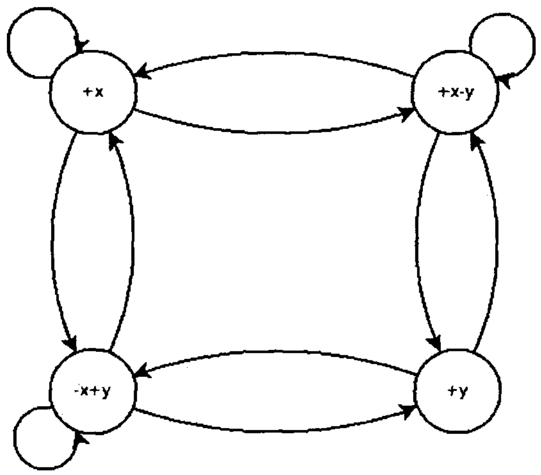


Figure 3.3: Finite State Machine for Zig-zag reordering

The FSM in Figure 3.3 represents the finite state machine for the zigzag reordering, the next state logic is controlled with a counter which determines the number of coefficients reordered. When the reordering is complete the state machine will continue with the next 8 × 8 quantized block.

actual	i = 0		j = 0		i = 7		j = 7		other
state	j even	j odd	i even	i odd	j even	j odd	i even	i odd	
+x	+x	-x+y	+x			+x-y	-x+y	+x	
-x+y				+y	+x				-x+y
+y			+x-y			-x+y			
+x-y	+x						+y		+x-y

Table 3.3: State table for zig-zag reordering FSM

Table 3.3 represents the logic used for the state machine, some spaces were left in blank as they are unreachable conditions. A counter is set to start in 0 and end in 63, for every count it is verified if any of the conditions are true, the information generated by verifying for true conditions will determine the next state. For an 8×8 block the counter values in which a condition is met are the ones in the contour of the matrix in Table 3.2.

The FSM implementation will produce a more complex implementation than the common look-up table implementations. This complexity increment is worth because there are other modules far more complex than the zig-zag reordering and it is faster than the look-up table implementations.

The complexity of this module resides in the output logic of the FSM. Each state in the FSM increments or decrements an index that is used as half of the address for the quantized matrix scanning. Most of the time the FSM does two operations, one addition and one subtraction, when the scanning process is in a coefficient in the boundaries of the matrix only one operation is performed, hence, the number of additions/subtractors needed to scan a whole quantized matrix is 114.

Chapter 4

Entropy Encoder and Decoder

So far, lossy compression methods were applied in order to be able to implement a lossless compression method. Discrete Cosine Transform were used to decorrelate the data and concentrate most of the energy in the first coefficients, Quantization were used to eliminate the coefficients that will not contribute in the reconstruction of the data and zig-zag scanning were made to reorganize the coefficients from the lowest spatial frequency value to the highest. The result is an array of coefficients with large chains of zeros in between.

When there are long chains of a certain value in a sequence it is easier to send how many times this value is repeated than sending the the same value several number of times, this compression method is known as Run Length Encoding (RLE). Quantized matrices consist mostly of zeros so RLE can be applied before coding the data.

The entropy encoder will consist in an RLE and a Huffman encoder. The module will receive quantized values and the RLE will verify if the value is zero or non-zero, the output of this module will be a pair of values, one indicating the number of zeros before the second value which is a non-zero value. This output will feed the Huffman encoder that will assign variable length codes according the probability of occurrence of each pair of data.

The information generated by the Huffman encoder will be sent to the decoder using a turbocodec. At the decoder there will be a huffman decoder that will receive the variable length codes and convert them into pairs of runs of zeros and non-zero level values. The pairs of data are translated to quantized coefficients and ordered in zig-zag so the same quantized matrix can be obtained at the decoder.

4.1 Entropy

Entropy is known to be the measure of the uncertainty of a random variable [Cover 06]. It is defined as the average amount of information conveyed by a each symbol generated by a DMS (Discrete Memoryless Source) [Symes 04] Let us consider the discrete random variable X with probability mass function $p(x) = Pr[X = x], x \in X$, the entropy of the discrete random variable X is defined as.

$$H(x) = -\sum_{x \in x} p(x) \log_2 (p(x)) \tag{4.1}$$

Let us consider two cases, the one where, for a known alphabet, only one symbol is send, and so, this symbol will have a probability of 1 while the rest of the symbols will have a probability of zero. The second case is when every symbol have the same probability to occur, hence, the discrete random variable will have an uniform pdf.

The first one is directly obtained as.

$$\begin{aligned} H(x) &= -\sum_{n=0}^{N-1} p(x_n) \log_2 (p(x_n)) \\ &= -\log(1) - 0\log(0) - 0\log(0) - \cdots - 0\log(0) \\ &= 0 \end{aligned}$$

The second case is obtained as next.

$$\begin{aligned} H(x) &= -\sum_{n=0}^{N-1} p(x_n) \log_2 (p(x_n)) \\ &= -\sum_{n=0}^{N-1} \frac{1}{N} \log_2 \left(\frac{1}{N}\right) \\ &= -\frac{1}{N} \log_2 \left(\frac{1}{N}\right) - \frac{1}{N} \log_2 \left(\frac{1}{N}\right) - \frac{1}{N} \log_2 \left(\frac{1}{N}\right) - \cdots - \frac{1}{N} \log_2 \left(\frac{1}{N}\right) \\ &= -\frac{N}{N} \log_2 \left(\frac{1}{N}\right) \\ &= -\log_2 \left(\frac{1}{N}\right) \end{aligned}$$

Last two demonstrations show that for probabilities 0 and 1, where resulting values are already known, entropy is equal to zero, then, the source is completely deterministic and so, the resulting value is already known, but for an uniform distribution, where all results have same probability to occur, the entropy of the information will met its maximum value and so it is harder to predict the expected value.

Figure 4.1 shows an example for a coin where one face changes its probability of occurrence from 0 to 1, it can be seen that for a fair coin, where heads and tails have a $p(x) = 0.5$, entropy is equal to 1, which according to Shannon, it is required 1 bit to represent each event and when the coin is completely unfair the result is already known and then no bits are required to represent each event.

Shannon had said that, the information conveyed in an event $I(E)$, measured in bits, in terms of the probability of that event $p(E)$ is defined as [Symes 04].

$$I(E) = \log_2 \left(\frac{1}{p(E)}\right) \tag{4.2}$$

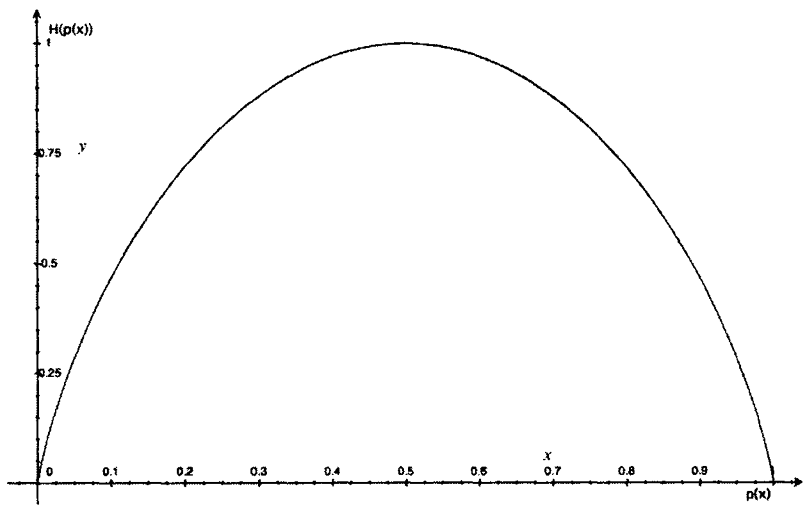


Figure 4.1: Entropy variation for tossing a coin

Then, substituting Eq. 4.2 in Eq. 4.1 we know that.

$$H(x) = \sum_{x \in X} p(x)\, I(x) \tag{4.3}$$

Events in a known alphabet of a DMS are not necessary to have the same probability and so, using Eq. 4.2, it can be seen that for an event with higher probability less bits are necessary to represent that event. Also, Eq. 4.3 represents the average amount of information conveyed in a DMS.

Eq. 4.2 and Eq. 4.3 lead to the "Shannon's noiseless source encoding theorem" which says that, in order to code a source in the most efficient manner possible, and that the code is uniquely decodable, the average number of bits per symbol used must be at least equal to the entropy of the source [Symes 04], which basically says that, for a certain source, different codes can be assigned for each element in its alphabet.

This method is known as variable length coding (VLC) and there are two main implementations, the arithmetic VLC and Huffman Coding, this thesis implements the VLC with Huffman codes.

4.2 Run Length Coding

It is already known that quantized matrix will consist mostly of zeros, also it is known that incoming data to the entropy encoder will have runs of zeros. If it is known at the decoder how many zeros there are before each non-zero value, it is not necessary to encode the each zero founded, it would be easier to only send a value which represents the number of zeros.

This lossless compression method is known as RLC (Run Length Coding) which is a coding method used when long runs of a certain symbol are being send [Symes 04].

This module will precede the Huffman Encoder, it will receive one quantized value at a time and it will check if it is a zero value or a non-zero value, if it is a zero value it will increment an internal counter and if it receives a non-zero value the counter is reseted and a (Run, Level) pair is sent to the Huffman encoder.

This module will also let the Huffman encoder know if it is a DC or AC coefficient, if the pair is valid and if the block is out of non-zero values (End of Block).

Hardware Implementation

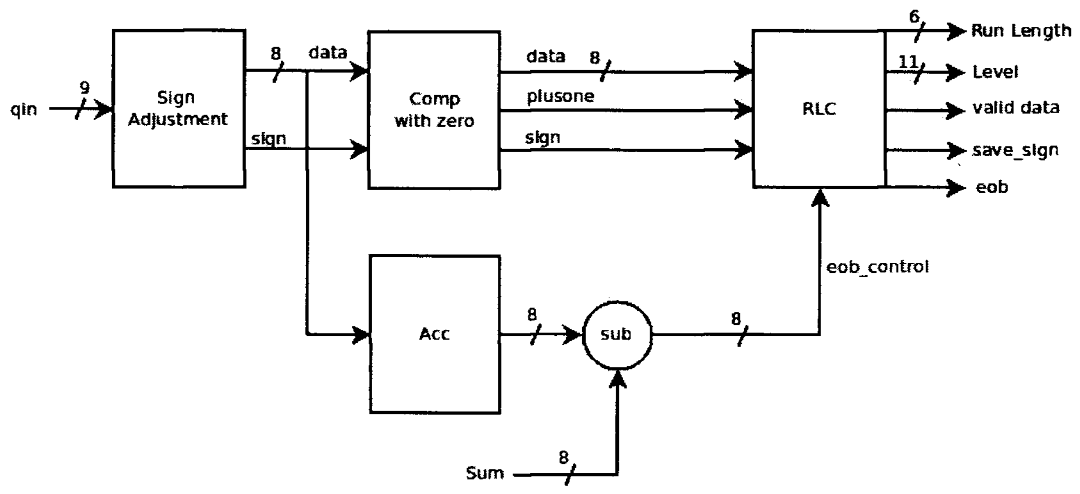


Figure 4.2: Block Diagram for Run Length Encoder

The block diagram presented in Figure 4.2 has 4 main modules. The first one is a sign adjustment which takes the quantized valued, if it is positive, the value is send as it is and the sign signal is 0, if it is negative, it is changed to positive and sign signal is set to 1. The second one is a comparer to zero, it checks if input value is zero, if it is zero the signal plusone is set to 1 so an internal counter in the RLC module is incremented, if the input value is not zero then the plusone signal value is 0 so the counter in the RLC module is reseted. The third one is the Acc which is an accumulator used to know if there are non-zero coefficients left in the matrix. The fourth module is the RLC which gives different values to its outputs to control the huffman encoder.

The number of operations done by this module will vary depending on the number of non-zero coefficients and the lengths of the codes. The module has one adder in the RLC block module, which is used to count zeros, and one subtractor, which is used to know when the block is out of non-zero values. For a codeword to be decoded it will take a minimum of zero additions for (Run, Pairs) where Run is zero, and a maximum of 31 additions for the $(31, 1)$ pair.

4.3 Huffman Coding

It was said before that different codes can be assigned for each symbol in a known alphabet, also the data have been compressed to a few values per quantized matrix, now an optimum code has to be found to encode this information.

To achieve this objective a minimum redundancy code has to be found, this is a code that, for a message consisting of a finite number of numbers and a finite number of coding digits, the message will yield the lowest possible average length [Huffman 52].

For a message with a finite alphabet which elements are organized from the most probable event to the least probable event with probability $P(x)$ and code length $L(x)$ for each symbol in the alphabet, if next statement is true.

$$P(0) \geq P(1) \geq P(2) \geq \cdots \geq P(N-1) \tag{4.4}$$

Then the next statement is necessary to be true

$$L(0) \leq L(1) \leq L(2) \leq \cdots \leq L(N-1) \tag{4.5}$$

In order to found an optimum code Eq. 4.4 and Eq. 4.5 have to be true.

There is also a list of requirements mentioned in [Huffman 52] that are also needed to construct an optimum code, they are listed here.

a. No two messages will consist of identical arrangements of coding digits.

b. The message codes will be constructed in such a way that no additional indication is necessary to specify where a message code begins and ends once the starting point of a sequence of messages is known.

c. $L(0) \leq L(1) \leq \cdots \leq L(N-2) = L(N-1)$

d. At least two and not more than D of the messages with code length $L(N)$ have codes which are alike except for their final digits.

e. Each possible sequence of $L(N)-1$ digits must be used either as a message code or must have one of its prefixes used as a message code.

Huffman had proposed a tree structure where all the elements in the source alphabet are ordered from the most probable event to the least probable event, the least two events are combined so their probability are the sum of their probabilities, if the elements have to be rearranged they are reordered from the most probable event to the least probable event, this procedure is followed until only two events are left in the list.

When there are only two elements in the list each one is assigned with a bit value, either the most probable element will have the value of 1 or the least probable element will have the value of 1. The two elements are unfolded one level each and elements in this level will obtain a new bit value, the value assigned have to follow the same logic as the previous level assignment, i.e., if the most probable element obtained the value of one, in the present level the most probable element must have the value of one. This procedure is repeated until all the events have been unfolded.

When the last procedure is completed each element will be associated with a string of bits each one with different length. The strings generated for each event will be the code associated with that event.

Event	p(E)	Code
a	0.5	0
b	0.25	10
c	0.125	110
d	0.125	111

Table 4.1: Huffman Example

An example is presented based in Table 4.1. The alphabet consist in four elements $E(x) = \{a, b, c, d\}$, second column is the probability associated with each event in the alphabet, it can be seen that the sum of all probabilities will have the value of 1. The third column is the huffman code associated with each element. Huffman codes are obtained with the huffman tree in Figure 4.3

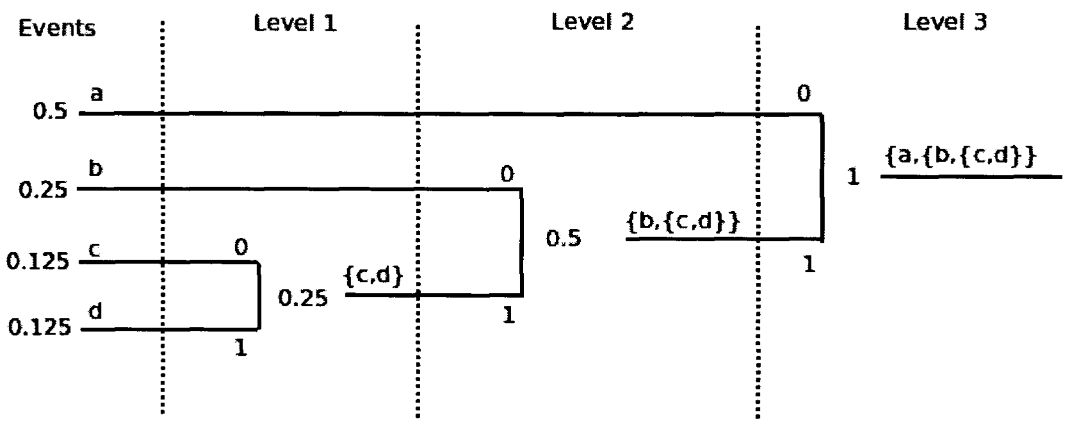


Figure 4.3: Huffman tree for Table 4.1

This method is known as a tree structure method and is categorized as an instantaneous code because when a codeword is identified no more information than the code itself is necessary to be decoded.

This module combined with the RLC constitute the final stage in the lossless compression for the video data. The idea is that the RLC will eliminate runs of zeros and huffman encoder will decrease the average code length and obtain a minimum redundancy code.

Hardware Implementation

Figure 4.4 shows the block diagram for the huffman encoder, it will receive two input values, first one is the DC coefficient value and the second one is the (Run, Level) pair value, both of them are encoded differently and the multiplexer will choose which one is sent, the multiplexer is controlled with a counter that counts from 0 to 63.

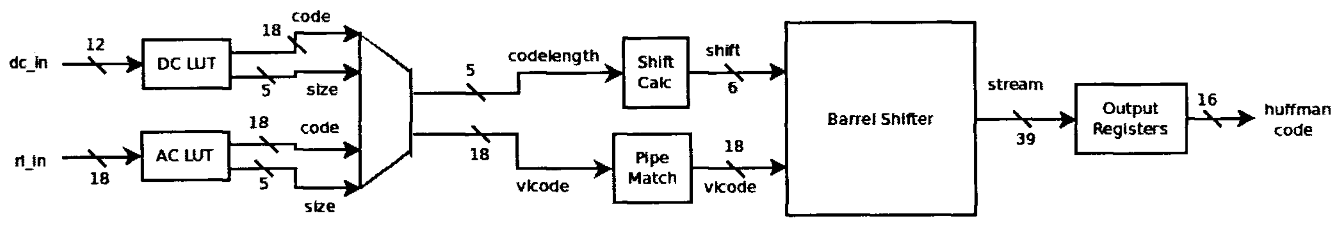


Figure 4.4: Huffman Encoder block diagram

Once the value to be transmitted is selected, a shift register, which is controlled by the number of bits to be send for the corresponding symbol, shifts the VLC value and puts its output in a register that will be send when full.

For this modules the tables used to code each (Run, Level) pair were Table 7.1 for the AC coefficients and Table 7.2 for the DC coefficients.

4.4 Huffman Decoder

The algorithm proposed for the Huffman encoder uses fixed-length symbols and assign variable length codes to them with a look-up table. Decoding is harder because the decoder receives a continuous bitstream and in order to found the (Run, Level) pairs sequence the decoder must know where a new code begins.

The trivial solution would be by following the tree from the root to the (Run, Level) pair it represents and repeat the same procedure for each incoming code. This solution is not efficient because different (Run, Level) pair would be decoded at different clock cycles which is not suitable for real-time applications [Yanmei 08].

Algorithms for variable length decoders (VLD) are divided into two categories, bit-serial (tree-based methods) and bit-parallel (look-up table-based methods) [Yanmei 08]. Discussion before have already said that bit-serial methods are not suitable for real-time applications, then, a bit-parallel method has to be implemented.

Bit-parallel methods, similarly to the huffman encoder, use a look-up table to found which (Run, Level) pair is represented by which codeword. There are some issues, the most significant one is the nature of the codewords which lengths are not fixed, then it is hard to implement a pipeline structure as the module has to decide the codeword length to extract the code and align the following bitstream for the next codeword [Hsieh 96].

The second issue is the memory size, VLC codebook is constructed with only 114 words, but VLD is mandatory to match all codewords for the incoming bitstream, this requires a memory size of 65536 without pre-processing the input bitstream [Hsieh 96].

These methods are also called fast decoders because they decode a codeword in a clock cycle by matching all possible codes [Hsieh 96], different from bit-serial methods which takes from 3 to 17 clock cycles to decode a codeword. Memory efficiency is important because the amount of data it has to be stored, ROM based memories lack of programmability while CAM (Content-addressable memory) based memories require more hardware.

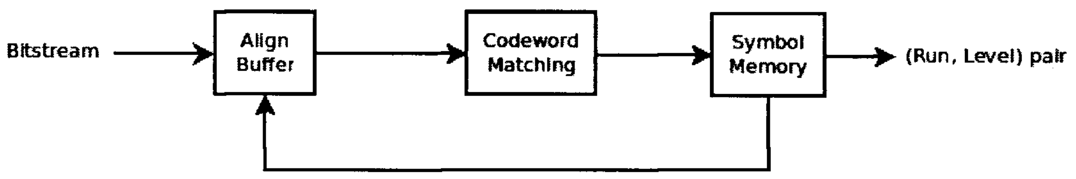


Figure 4.5: VLD Block Diagram

Figure 4.5 shows a general VLD architecture. It can be seen that the incoming bitstream is stored in an alignment buffer, then it is matched with the codewords in the codebook and if a match is found the system will search in the look-up table for the (Run, Level) pair.

Algorithms for pre-processing of input bitstream consist in clustering the codewords in partitions that can be generated either by recognizing a bit pattern or with a fixed number of bits, the partition then is used as first reference for the memory searching [Hsieh 96].

Pre-processing the input bitstream will reduce the codebook memory size as the codewords combinations are reduced, the amount of memory size that is reduced depends of how efficient the pre-processing is.

Memory selection is also important as each kind of memory has its own benefits, then, each memory is selected depending the application. The most common VLDs are PLA, ROM, CAM and RAM memory based [Yanmei 08].

Cheng-Teh Hisieh and Seung P. Kim have proposed two decoding methods in [Hsieh 96]. First one consists in a decoding based in a Maximum Likely Bit Pattern Matching concurrent algorithm (MLBP), this algorithm will assign the codewords in the codebook into groups of same length and same precedence sequence of bits. The second method algorithm is a Concurrent Decoding Algorithm, this algorithm will try to decode more than one codeword so the alignment buffer is reduced in size.

Yanmei Qu et al. proposed an optimized look-up table for MPEG-2 and a block-based buffer architecture between the VLD and the IDCT to store reconstructed transform coefficients [Yanmei 08].

4.4.1 Maximum Likely Bit Pattern (MLBP)

MLBP algorithm will consist in creating groups of codewords with the same length and then divide them in groups with a common bit pattern. The MLBP groups are activated by matching bit patterns and the information obtained is used to decode the symbol the codeword represents.

Each group in the MLBP algorithm will consist in a MLBP sequence, which is the common sequence among the codewords that belong to the group, a maximum of three bits, normally the last three bits in the codeword, known as reminder that will indicate which codeword is being decoded, and a priority check that will mask the priority bits when more than one group is activated by a MLBP sequence.

The algorithm will try to match bit patterns with the MLBP groups, if there are any matches the group or groups that satisfy the match are activated and used as first reference to access the symbol memory. More than one group can be activated, this happens when a group is a subset from other group, when this happen the priority bits will determine which group is used to decode the codeword.

The MLBP is extracted from the bit pattern and the remaining bits are used as the second reference in the symbol memory. The group number combined with the reminder are used as the address for the symbol memory [Hsieh 96].

Table 4.2 is just a section of Table 1 in [Hsieh 96] which shows the intrablock Huffman table grouped in MLBP. The table presented here only contains the first 10 groups which are the codes with lengths from 2 to 7 bits. Complete table is shown in Appendix B.

4.4.2 Concurrent Decoding Algorithm

This method is a combination between a bit-serial algorithm and a bit-parallel algorithm, it is based upon the assumption that for a 16-bit input bitstream and that the smallest code is 2-bit long, the worst case will consist in a 16-bit bitstream with 8 2-bit codewords, then, 8 cycles are consumed to decode all the codewords and then a big buffer has to be implemented because the amount of data that will arrive in that 8 clock cycles.

The VLC is constructed so the most common events are coded with the shortest codes, then the last case is not as unusual as one would like it to be which implies that the buffer has to be big enough. The algorithm proposed by Chen-Teh Hsieh and Seung P. Kim tries to identify this cases and decode the symbols in a concurrent manner so less clock cycles are required and a smaller buffer can be implemented.

The algorithm will consist in constructing a codeword-length tree that represents the number of codewords that can be concurrently decoded from a given n-bit input bitstream. When a bitstream is received, all the possible bit patterns are matched with all the codeword lengths at each level, then, if it is detected that two or more codewords can be decoded concurrently they are decoded at the same time. The number of codewords that will be able to be concurrently decoded will depend in the number of levels in the tree.

From last discussion it is obvious that the performance will increase with more levels but it has to be considered that adding levels implies more complex hardware. Most of the implementations under this algorithm only use two levels since the probability is getting smaller for successfully decoding more number of codewords at the same time

length	codeword	MLBP	group no.	remainder	priority
2	10	10	1	x	
3	0 10	x10	2	0	
	1 10			1	
4	011 0	011	3	0	
	011 1			1	
5	001 01	001	4	01	pc-1
	001 10			10	
	001 11			11	
	1110 0	1110	5	0	
	1110 1			1	
6	0001 00	0001	6	00	
	0001 01			01	
	0001 10			10	
	0001 11			11	
	0000 01	0000 01	7	x	
7	00001 00	0000 1	8	00	
	00001 01			01	
	00001 10			10	
	00001 11			11	
	11110 00	1111 0	9	00	
	11110 01			01	
	11110 10			10	
	11110 11			11	
	1111 100	1111 100	10	x	pc -2

Table 4.2: MLBP table for the first 10 groups [Hsieh 96]

[Hsieh 96].

An example from [Hsieh 96] with a two level codeword-length tree and a 4-bit input bitstream and codelengths from 2 - 4 bits is presented for a better understanding of the algorithm.

The possible combinations for a 4-bit input bitstream are, two 2-bit cod words, one 3-bit codeword and one 4-bit codeword, the codeword-length tree for this example is provided in Figure 4.6. It can be seen from the figure that when the decoder receives either one 4-bit codeword or one 3-bit codeword, they are decoded with a conventional bit-parallel algorithm but when two 2-bit codewords are received then they can be decoded at the same time.

4.4.3 Cost-effective VLD

Yanmei Qu et al. in [Yanmei 08] have used a scheme similar to the MLBP algorithm in [Hsieh 96], they have contributed by optimizing the look-up table for MPEG-2 and by adding the inverse quantization and inverse zig-zag reordering processes in order to

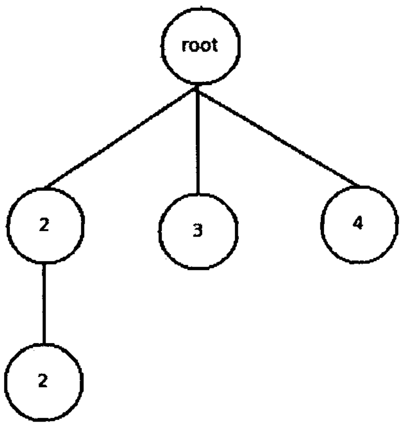


Figure 4.6: Example Codeword-length Tree

save one memory read/write operation which not only saves hardware cost but also memory read/write operations are slow in comparison to other module operations.

Other improvements were made like the addition of block-based buffers between the VLD and IDCT, also they have changed the RAM-based memory system for a ROM-based memory system in order to reduce the implementation area and save hardware cost. Block-based buffers substitute the Macroblock-based buffers to reduce the memory size and to save decoding cycles when skip mode occurs [Yanmei 08].

Algorithms based in MLBP are known as group-based VLDs because the codewords are grouped in equal bit patterns which makes easier to found the symbol each codeword represents and it reduces the memory size as each symbol is decoded by looking for bit patterns. Most VLD use this algorithm because its efficiency and high speed [Yanmei 08].

Hardware Implementation

The approach for the hardware implementation is the MLBP with an 16-bit input bitstreams which are the way they are generated at the huffman encoder. As it was said before, a bit pattern that matches a MLBP group is searched, when a group is found a small memory with the (Run, Level) pairs associated with that group is activated.

It is known that bitstream will arrive as in Figure 4.7 where s is the sign of the codeword.

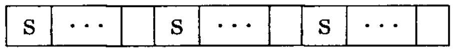


Figure 4.7: Codewords organization in the bitstream

Input bitstream is received as in Figure 4.7, the bitstream is matched with all the MLBP prefix groups so a match is obtained, matching will be done with an array of AND-gates.

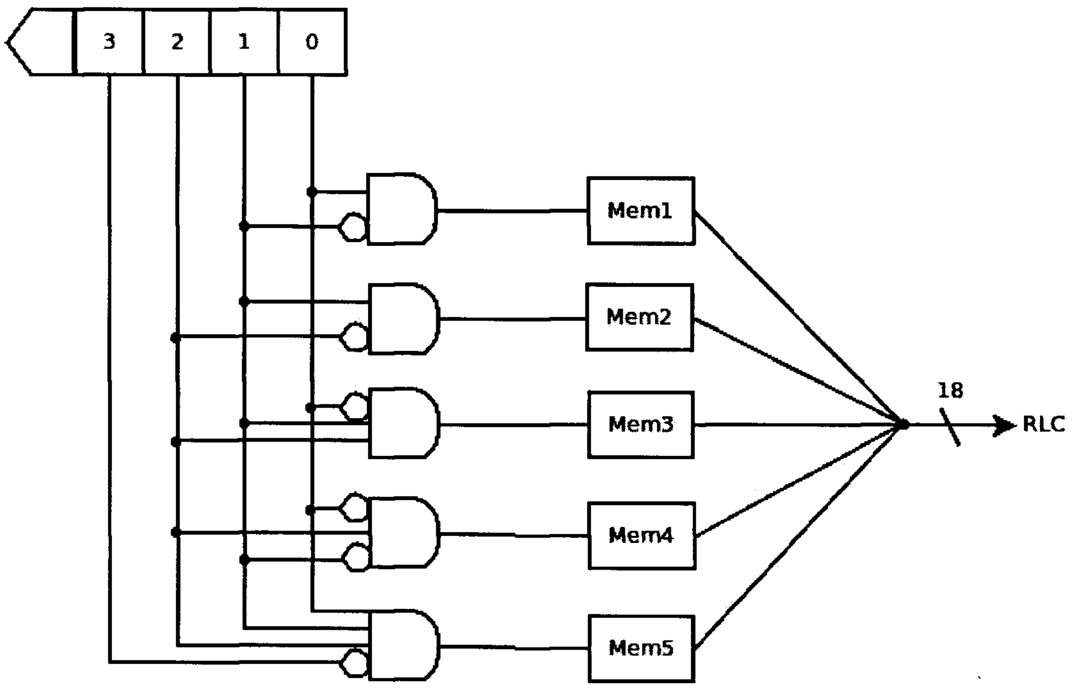


Figure 4.8: Group Matching for the first 5 MLBP Groups

Some bit patterns are subsets of another bit pattern, i.e., when the bigger bit pattern is activated the group which depends on the subset sequence, too, are activated. For this groups a priority is assigned, the bigger the sequence, the higher the priority. This is because the huffman coding rule which says that no codeword can be used as prefix for another codeword.

When a group is activated the reminder bits are used as the address for the small memory of (Run, Level) pairs. With the group memory activated and the reminder as memory address a (Run, Level) pair will be decoded and the last step is to remove the used codeword and shift the input bitstream n positions to the right, where n is the length of the decoded codeword.

Figure 4.8 shows the block diagram for the first 5 MLBP groups, the diagram only shows how the matches are evaluated but the memories will have the remainder of each group as input address and some of them will have a priority flag input which is used when two groups are activated by the same prefix.

Priority enable is implemented as in Figure 4.9, *higher_priority* and *lower_priority* are the enable signals for each of the MLBP groups and *enable_mem*0 and *enable_mem*1 are the enable signals for the memories in Figure 4.8.

A different data path is used for DC coefficients. The algorithm is slightly different from the AC coefficients data path as DC coefficients use a different representation method. Either way the architecture will be the same as in Figure 4.5 and will follow the same implementation strategy as Figure 4.8, the connections between the bit positions in the bitstream and the matching logic will be different, basically, the matching logic will look for the first zero in the bitstream to determine the length of the code for the

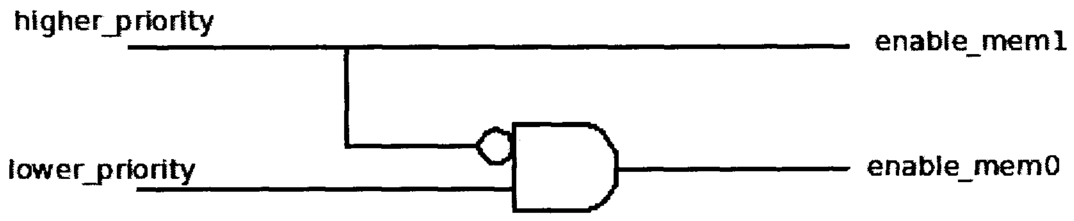


Figure 4.9: Priority Enables Circuit Implementation

DC coefficient.

When a VLC size is detected, the next n bits, where n is the number of bits in the VLC size, will be the direct representation of the DC coefficient, only a sign adjustment and a sign extension are needed to obtain the real coefficient. According to Table 7.2, a code that starts with 0 indicates a negative value and a code that starts with 1 indicates a positive value.

Mathematically speaking, the variable length decoder is simple, the only operation performed is a multiplication which shifts the bitstream buffer n positions, where n is the length of the codeword decoded. The operations performed in this module are mostly logical but the number of matches, the memory access to the LUTs and the logic used to determine the number of bits to be shifted will slow the module.

It has to be noted that it is hard to pipeline the module as most of the logic is required to be done in the same clock cycle. Hence, optimizations of this module have to be done in the operation speed more than in the complexity of the algorithm.

Chapter 5

Results

The results presented now are the hardware used by each module being synthesized on the FPGA Spartan 3A DSP (XC3SD3400A-4FG676) and the maximum frequency for each one, also a simulation of the matrix in Table 5.1.

155	155	155	158	158	156	158	159
155	155	155	158	158	156	158	159
155	155	155	158	158	156	158	159
155	155	155	158	158	156	158	159
155	155	155	158	158	156	158	159
151	151	151	154	157	156	156	156
155	155	155	156	157	158	156	153
149	149	149	153	155	154	153	154

Table 5.1: Reference matrix for simulations

DCT

Maximum frequency for DCT will be 79.371 MHz

Logic Utilization	Used	Available	Utilization
Number of Slices	1794	23872	7%
Number of Slice FF	2171	47744	4%
Number of 4 Input LUTs	1837	47744	3%
Number of bound IOBs	23	469	4%
Number of GCLKs	1	24	4%
Number of DSP48s	8	126	6%

Table 5.2: DCT Device Utilization

The simulations results with the module implemented as described in the hardware implementation for the DCT in Chapter 2.

Table 5.3 shows the simulation results for the DCT, it can be seen that almost all the information is stored in the upper left corner while the lower right corner consists almost of zero values. This table will serve as input table for the Quantization simulation.

224	-10	-4	0	4	-1	-2	0
10	1	2	-3	2	1	-1	0
-5	0	-1	1	0	0	1	0
2	0	0	1	-1	0	0	0
-1	-1	1	-2	1	0	0	0
3	3	-1	1	-1	0	0	0
-5	-3	0	0	1	-1	0	0
4	2	0	0	0	1	0	0

Table 5.3: DCT Simulation Results

IDCT

Maximum frequency for IDCT will be 62.496 MHz

Logic Utilization	Used	Available	Utilization
Number of Slices	2619	23872	10%
Number of Slice FF	3164	47744	6%
Number of 4 Input LUTs	2913	47744	6%
Number of bound IOBs	24	469	5%
Number of GCLKs	1	24	4%
Number of DSP48s	24	126	19%

Table 5.4: IDCT Device Utilization

IDCT is the last module in the simulation, it will take Table 5.9 as input and will generate Table 5.5.

155	151	140	111	151	118	103	95
156	152	140	110	151	118	102	95
157	153	141	110	152	118	101	93
160	155	141	108	153	117	99	91
156	152	140	110	151	118	102	94
159	154	141	109	153	117	100	91
161	155	141	108	153	117	99	90
161	155	142	158	154	117	99	90

Table 5.5: IDCT Simulation Results

This results are compared with the initial reference table, Table 5.1. The results basically show that there exist losses of information during the block processing but at the end the data is consistent. It is important to remember that as the human visual system acts as a very efficient filter, it is not necessary to obtain the original values exactly as they were obtained from the camera or any video source.

Quantization and zig-zag reordering

Maximum frequency for quantization and zig-zag reordering will be 139.762 MHz

Logic Utilization	Used	Available	Utilization
Number of Slices	1304	23872	5%
Number of Slice FF	1498	47744	3%
Number of 4 Input LUTs	1284	47744	2%
Number of bound IOBs	25	469	5%
Number of GCLKs	1	24	4%
Number of DSP48s	0	126	0%

Table 5.6: Quantization and Zig-Zag reordering Device Utilization

As said before, this module will take the data from the DCT and will perform the quantization using the Table 3.1 as Quantization table, for the simulation the module will take as sample matrix the Table 5.3.

Results are shown in Table 5.7 and they will be the input for the entropy encoder and for the inverse quantization. As the entropy decoder is left for further work the resulting matrix will be used as input matrix for inverse quantization.

28	-1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Table 5.7: Quantization Simulation Results

Inverse zig-zag reordering and inverse quantization

Maximum frequency for inverse zig-zag reordering and inverse quantization will be 67.866 MHz

Logic Utilization	Used	Available	Utilization
Number of Slices	1113	23872	4%
Number of Slice FF	1275	47744	2%
Number of 4 Input LUTs	1017	47744	2%
Number of bound IOBs	25	469	5%
Number of GCLKs	1	24	4%
Number of DSP48s	1	126	0%

Table 5.8: Inverse Zig-Zag reordering and Inverse Quantization Device Utilization

Table 5.9 shows the simulation results for the inverse quantization process, the input matrix used was the same as the one in the Table 5.7, this table is the reconstruction of the Table 5.3, it can be seen that there are losses of information as both tables are not equal.

224	-16	0	0	0	0	0	0
16	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Table 5.9: Inverse quantization simulation results

Entropy Encoder

Maximum frequency for Entropy Encoder will be 67.376 MHz

Logic Utilization	Used	Available	Utilization
Number of Slices	322	23872	1%
Number of Slice FF	367	47744	0%
Number of 4 Input LUTs	469	47744	0%
Number of bound IOBs	37	469	7%
Number of GCLKs	1	24	4%
Number of DSP48s	3	126	2%

Table 5.10: Entropy Encoder Device Utilization

This is the last compression module, it will take the values in Table 5.7 and will assign a variable length code for each coefficient. The simulation results are shown in the next array of bits.

11110111001011000110

The meaning of this string is described in Table 5.11

VLC Code	(Run,Level)	AC/DC
1111011100	28	DC
101	(0,-1)	AC
100	(0,1)	AC
0110	EOB	

Table 5.11: Huffman Representation for Simulation Results

Entropy Decoder

Maximum frequency for Entropy Decoder will be 52.497MHz.

Logic Utilization	Used	Available	Utilization
Number of Slices	559	23872	2%
Number of Slice FF	110	47744	0%
Number of 4 Input LUTs	1073	47744	2%
Number of bound IOBs	38	469	8%
Number of GCLKs	1	24	4%
Number of DSP48s	5	126	3%

Table 5.12: Entropy Encoder Device Utilization

The algorithm proposed in the variable length decoder section will generate the (Run, Level) pairs in Table 5.11 using as input the bitstream generated by the Huffman encoder module. It has to be noted that the module is the slowest in the system, this is because the module is poorly pipelined, in fact, it is almost a single cycle architecture, hence, a more efficient algorithm has to be implemented if higher resolutions are required.

Chapter 6

Conclusions and further work

An encoder was developed using lossy and lossless techniques and the results showed that the compression ratio is fair enough so a low bandwidth can be achieved, also the slowest module, which is the IDCT, operates with a maximum frequency of 62.496 MHz so this frequency determines the operation frequency of the codec system. In an 1080i (1920 x 1080) with frame rate of 30 Hz video system, the number of pixels per second will be $30 * 1920 * 1080 = 62,208,000\ pixels/sec = 62.208$ MHz, therefore, the codec as it is by now can process this amount of data.

Even when pipeline implementation uses more hardware than a single cycle architecture the hardware utilization was low, the module that used most resources was the IDCT that used 19% of the DSPs and 10% of slices in the FPGA which is normal as it is the module with more pipeline stages.

As Figure 1.8 shows, the Motion Compensator module will generate the additional information for the decoder so is important to found an efficient implementation because as it was said before, the better the side information the better performance the Wyner-Ziv codec will have.

Also, more efficient VLD architectures have to be researched as this module is one of the slower modules in the codec, a bad implementation for this module would require a bigger (Run, Level) pair memory and a bigger alignment buffer before the symbol memory, also as a bigger clock cycle period would be required so a better implementation of this module will yield a better performance of the system.

The most critical elements in the system are the DCT, IDCT and the Huffman decoder. DCT and IDCT are the most computationally complex modules in the system as the DCT uses 8 multipliers and 14 adders while the IDCT uses 24 multipliers and 14 adders, they are both implemented with recursive algorithms which are known to be optimal and their maximum operating frequency are 79.371 MHz and 62.496 MHz respectively. Hence, if the codec is needed to be reduced in complexity, then, this two modules have to be optimized.

The Huffman decoder is the slowest module in the system, it has most of its logic implemented in one clock cycle as it is hard to pipeline the module because of the feedback which determine how many bits were used to decode the last (Run, Level) pair, its maximum operating frequency is about 52.497 MHz, hence, to increment the speed of the system this module has to be optimized. It has to be noted that the Huffman decoder as it is right now, if it is added to the whole system, the performance described at in the first paragraph will not be achieved so a more efficient algorithm is needed.

Chapter 7

Appendix A

This appendix will list the Huffman Codewords corresponding for each pair of runs of zeros and non-zero levels.

7.1 Huffman Table

VLC Codeword	Run	Level	VLC Codeword	Run	Level
0110	End of Block		8*0 0101 11 s	0	24
10 s	0	1	8*0 0101 10 s	0	25
110 s	0	2	8*0 0101 01 s	0	26
0111 s	0	3	8*0 0101 00 s	0	27
1110 0 s	0	4	8*0 0100 11 s	0	28
1110 1 s	0	5	8*0 0100 10 s	0	29
0001 01 s	0	6	8*0 0100 01 s	0	30
0001 00 s	0	7	8*0 0100 00 s	0	31
1111 011 s	0	8	8*0 0011 000 s	0	32
1111 100 s	0	9	8*0 0010 111 s	0	33
0010 0011 s	0	10	8*0 0010 110 s	0	34
0010 0010 s	0	11	8*0 0010 101 s	0	35
1111 1010 s	0	12	8*0 0010 100 s	0	36
1111 1011 s	0	13	8*0 0010 011 s	0	37
1111 1110 s	0	14	8*0 0010 010 s	0	38
1111 1111 s	0	15	8*0 0010 001 s	0	39
8*0 0111 11 s	0	16	8*0 0010 000 s	0	40
8*0 0111 10 s	0	17	010 s	1	1
8*0 0111 01 s	0	18	0011 0 s	1	2
8*0 0111 00 s	0	19	1111 001 s	1	3
8*0 0110 11 s	0	20	0010 0111 s	1	4
8*0 0110 10 s	0	21	0010 0000 s	1	5
8*0 0110 01 s	0	22	8*0 1011 0 s	1	6
8*0 0110 00 s	0	23	8*0 1010 1 s	1	7

VLC Codeword	Run	Level	VLC Codeword	Run	Level
8*0 0011 111 s	1	8	1111 000 s	9	1
8*0 0011 110 s	1	9	8*0 1000 1 s	9	2
8*0 0011 101 s	1	10	1111 010 s	10	1
8*0 0011 100 s	1	11	8*0 1000 0 s	10	2
8*0 0011 011 s	1	12	0010 0001 s	11	1
8*0 0011 010 s	1	13	8*0 0001 1010 s	11	2
8*0 0011 001 s	1	14	0010 0101 s	12	1
8*0 0001 0011 s	1	15	8*0 0001 1001 s	12	2
8*0 0001 0010 s	1	16	0010 0100 s	13	1
8*0 0001 0001 s	1	17	8*0 0001 1000 s	13	2
8*0 0001 0000 s	1	18	0000 0010 1 s	14	1
0010 1 s	2	1	8*0 0001 0111 s	14	2
0000 111 s	2	2	0000 0011 1 s	15	1
1111 1100 s	2	3	8*0 0001 0110 s	15	2
0000 0011 00s	2	4	0000 0011 01 s	16	1
8*0 1010 0 s	2	5	8*0 0001 0101 s	16	2
0011 1 s	3	1	0000 0001 1111 s	17	1
0010 0110 s	3	2	0000 0001 1010 s	18	1
0000 0001 1100 s	3	3	0000 0001 1001 s	19	1
8*0 1001 1 s	3	4	0000 0001 0111 s	20	1
0001 10 s	4	1	0000 0001 0110 s	21	1
1111 1101 s	4	2	8*0 1111 1 s	22	1
0000 0001 0010 s	4	3	8*0 1111 0 s	23	1
0001 11 s	5	1	8*0 1110 1 s	24	1
0000 0010 0 s	5	2	8*0 1110 0 s	25	1
8*0 1001 0 s	5	3	8*0 1101 1 s	26	1
0000 110 s	6	1	8*0 0001 1111 s	27	1
0000 0001 1110 s	6	2	8*0 0001 1110 s	28	1
8*0 0001 011 s	6	3	8*0 0001 1101 s	29	1
0000 100 s	7	1	8*0 0001 1100 s	30	1
0000 0001 0101 s	7	2	8*0 0001 1011 s	31	1
0000 101 s	8	1	0000 01	Escape	
0000 0001 0001 s	8	2			

Table 7.1: Huffman Code for MPEG-2 Intra blocks

Table. 7.1 is used to encode intra blocks for the MPEG-2 standard, *s* means the sign of the encoded data and the nomenclature $8 * 0$ means a run of 8 zeros.

Range	Size	VLC Size	VLI
-2047 to - 1024	11	9*1	9*0 00 to 0 9*1 1
-1023 to -512	10	8*1 0	9*0 0 to 0 9*1
-511 to - 256	9	7*1 0	9*0 to 0 8*1
-255 to -128	8	6*1 0	8*0 to 0 7*1
-127 to -64	7	5*1 0	7*0 to 0 6*1
-63 to -32	6	4*1 0	6*0 to 0 5*1
-31 to -16	5	1110	5*0 to 0 4*1
-15 to -8	4	110	4*0 to 0111
-7 to -4	3	101	000 to 011
-3 to -2	2	01	00 to 01
-1	1	00	0
0	0	100	
1	1	00	1
2 to 3	2	01	10 to 11
4 to 7	3	101	100 to 111
8 to 15	4	110	1000 to 4*1
16 to 31	5	1110	1 4*0 to 5*1
32 to 63	6	4*1 0	1 5*0 to 6*1
64 to 127	7	5*1 0	1 6*0 to 7*1
128 to 255	8	6*1 0	1 7*0 to 8*1
256 to 511	9	7*1 0	1 8*0 to 9*1
512 to 1023	10	8*1 0	1 9*0 to 9*1 1
1024 to 2048	11	9*1	1 9*0 0 to 9*1 11

Table 7.2: Variable Length Codes for MPEG-2 DC coefficients

Table 7.2 is used to assign variable length codes to the DC coefficients to be encoded.

Chapter 8

Appendix B

The table presented here represents the MLBP groups and which codewords are related to them. Codewords are the same as in Table 7.1 which is the huffman table for intrablocks in the MPEG-2 video standard.

8.1 MLBP Group Table

length	codeword	MLBP	group no.	remainder	priority
2	10	10	1	x	
3	0 10	x10	2	0	
	1 10			1	
4	011 0	011	3	0	
	011 1			1	
5	001 01	001	4	01	pc-1
	001 10			10	
	001 11			11	
	1110 0	1110	5	0	
	1110 1			1	
6	0001 00	0001	6	00	
	0001 01			01	
	0001 10			10	
	0001 11			11	
	0000 01	0000 01	7	x	
7	00001 00	0000 1	8	00	
	00001 01			01	
	00001 10			10	
	00001 11			11	
	11110 00	1111 0	9	00	
	11110 01			01	
	11110 10			10	

length	codeword	MLBP	group no.	remainder	priority
7	11110 11	1111 0	9	11	
	1111 100	1111 100	10	x	pc -2
8	00100 000	0010 0	11	000	pc-1
	00100 001			001	
	00100 010			010	
	00100 011			011	
	00100 100			100	
	00100 101			101	
	00100 110			110	
	00100 111			111	
	11111 010	1111 1	12	010	pc-2
	11111 011			011	
	11111 100			100	
	11111 101			101	
	11111 110			110	
	11111 111			111	
9	0000001 00	0000 001	13	00	pc-3
	0000001 01			01	
	0000001 11			11	
10	000000110 0	0000 0011 0	14	0	pc-3
	000000110 1			1	
12	000000010 001	0000 0001 0	15	001	
	000000010 010			010	
	000000010 101			101	
	000000010 110			110	
	000000010 111			111	
	000000011 001	0000 0001 1	16	001	
	000000011 010			010	
	000000011 100			100	
	000000011 110			110	
	000000011 111			111	
13	8*0 10 000	8*0 10	17	000	
	8*0 10 001			001	
	8*0 10 010			010	
	8*0 10 011			011	
	8*0 10 100			100	
	8*0 10 101			101	
	8*0 10 110			110	

length	codeword	MLBP	group no.	remainder	priority
13	8*0 11 011	8*0 11	18	011	
	8*0 11 100			100	
	8*0 11 101			101	
	8*0 11 110			110	
	8*0 11 111			111	
14	8*0 010 000	8*0 010	19	000	
	8*0 010 001			001	
	8*0 010 010			010	
	8*0 010 011			011	
	8*0 010 100			100	
	8*0 010 101			101	
	8*0 010 110			110	
	8*0 010 111			111	
	8*0 011 000	8*0 011	20	000	
	8*0 011 001			001	
	8*0 011 010			010	
	8*0 011 011			011	
	8*0 011 100			100	
	8*0 011 101			101	
	8*0 011 110			110	
	8*0 011 111			111	
15	8*0 0010 000	8*0 0010	21	000	
	8*0 0010 001			001	
	8*0 0010 010			010	
	8*0 0010 011			011	
	8*0 0010 100			100	
	8*0 0010 101			101	
	8*0 0010 110			110	
	8*0 0010 111			111	
	8*0 0011 000	8*0 0011	22	000	
	8*0 0011 001			001	
	8*0 0011 010			010	
	8*0 0011 011			011	
	8*0 0011 100			100	
	8*0 0011 101			101	
	8*0 0011 110			110	
	8*0 0011 111			111	

length	codeword	MLBP	group no.	remainder	priority
16	8*0 0001 0000	8*0 0001 0	23	000	
	8*0 0001 0001			001	
	8*0 0001 0010			010	
	8*0 0001 0011			011	
	8*0 0001 0100			100	
	8*0 0001 0101			101	
	8*0 0001 0110			110	
	8*0 0001 0111			111	
	8*0 0001 1000	8*0 0001 1	24	000	
	8*0 0001 1001			001	
	8*0 0001 1010			010	
	8*0 0001 1011			011	
	8*0 0001 1100			100	
	8*0 0001 1101			101	
	8*0 0001 1110			110	
	8*0 0001 1111			111	

Table 8.1: MLBP Group Table

Table 8.1 is used as look-up table to decode VLC codewords. $8 * 0$ is the same as saying that there are a string of eight zero values. Each group will be interpreted as a small memory in the hardware implementation, the group number will be used to identify from which memory the symbol is going to be decoded and the reminder will be used as memory address.

Bibliography

[Aaron 04] Anne Aaron and Bernard Girod. *Wyner-Ziv Video Coding With Low Encoder Complexity.* Proceedings of International Picture Coding Symposium, 2004.

[Aaron d] Anne Aaron, Rui Zhang and Bernard Girod. *Wyner-Ziv Coding of Motion Video.* Rapport technique, Stanford University, Standord, CA 94305, n.d.

[Ahmed 74] N. Ahmed, T. Natarajan and K. R. Rao. *Discrete Cosine Transform.* IEEE Transactions on Computers, January 1974.

[Ascenso 06] João Ascenso, Catarina Brites and Fernando Pereira. Improving frame interpoation with spatial motion smoothing for pixel domain distributed video coding. 2006.

[CCITT 93] CCITT. *Information Technology - Digital Compression and Coding of Continuous-Tone Still Images - Requirements and Guidelines.* Rapport technique, International Telecommunication Union, 1993.

[Chan 91] S. C. Chan and K. L. Ho. *A New Two-Dimensional Fast Cosine Transform Algorithm.* IEEE TRansactions on Signal Processing, Vol. 39, No. 2, February 1991.

[Chen 77] Wen-Hsiung Chen, C. Harrison Smith and S. C. Fralick. *A Fast Computational Algorithm for the Discrete Cosine Transform.* IEEE TRansactions on Communications, Vol. COM-25, No. 9, September 1977.

[Cho 91] Nam Ik Cho and Shang Uk Lee. *Fast Algorithm and Implementation of 2-D Discrete Cosine Transform.* IEEE Transactions on Circuits and Systems, Vol. 38, No. 3, March 1991.

[Cover 06] Thomas M. Cover and Joy A. Thomas. Elements of information theory. Wiley, second edition, 2006.

[Cvetkovié 92] Zoran Cvetkovié and Miodrag V. Popovié. *New Fast Recursive Algorithms for the Computation of Discrete Cosine and Sine Transforms.* IEEE Transactions on Signal Processing, Vol. 40, August 1992.

[Espinosa 10] Myriam Alanis Espinosa. Implementation of 3gpp-lte turbo codes. Master's thesis, Tecnológico de Monterrey, Diciembre 2010.

[Giacomán 09] Eduardo Giacomán. Análisis del desempeño de los intercaladores basados en polinomios cuadráticos de permutación en canales awgn y rayleigh. Master's thesis, Tecnológico de Monterrey, Agosto 2009.

[Girod 05] Bernard Girod, Anne Margot Aaron, Shantanu Rane and David Rebollo-Monedero. *Distributed Video Coding.* Proceedings of the IEEE, Vol. 93, No. 1, pp. 71–83, January 2005.

[Grosse 97] H.J. Grosse, M. R. Varley, T.J. Terrel and Y.K. Chan. *Hardware Implementation of Versatile Zigzag-reordering Algorigthm for Adaptive JPEG-like Image Compression Schemes.* Conference Publication, No. 443, July 1997.

[Haskell 97] Barry G. Haskell, Atul Puri and Arun N. Netravali. Digital video: An introduction to mpeg-2. International Thomson Publishing, 1997.

[Hsieh 96] Cheng Hsieh and Seung P. Kim. *A Concurrent Memory-Efficient VLC Decoder for MPEG Applications.* IEEE Transactions on Consumer Electronics, Vol. 42, No. 3, 1996.

[Huang 09] Cheng Huang and Youlian Zhu. *Fast Algorithm for Arbitrary Length Discrete Cosine Transform.* IEEE Computer Society, 2009.

[Huffman 52] David A. Huffman. *A Method for the Construction of Minimum-Redundancy Codes.* Proceedings of the I.R.E, Vol. 40, 1952.

[Lakhani 03] Gopal Lakhani. *Modified JPEG Huffman Coding.* IEEE Transactions on Image Processing, Vol. 12, No. 2, 2003.

[Lee 84] Byeong Gi Lee. *FCT - A Fast Cosine Transform.* 1984.

[Lee 94] PeiZong Lee and Fang-Yu Huang. *Restructured Recursive DCT and DST Algorithms.* IEEE Transactions on Signal Processing, Vol. 42, No. 7, July 1994.

[Mendoza 06] Ulises Mendoza and René Romero. *VHDL Core for the computation of the One-Dimensional Discrete Cosine Transform.* 2006.

[Peixoto 08] Eduardo Peixoto, Ricardo de Queiros and Debargha Mukherjee. *On Side Information Generation for Wyner-Ziv Video Coding.* XXVI Simpósio Brasileiro de Telecomunicações, September 2008.

[Puri 02] Rohit Puri and Kannan Ramchandran. *PRISM: A New Robust Video Coding Architecture Based on Distrubution Compression Principles.* Allerton Conf. Communication, Control, and Computing, Allerton, IL, 2002.

[Rao 90] K. R. Rao and P. Yip. Discrete cosine transform: Algorithms, advantages, applications. Academic Press Limited, 1990.

[Slepian 73] David Slepian and Jack K. Wolf. *Noiseless Coding of Correlated Information Sources.* IEEE Transactions on Information Theory, Vol. IT-19, No. 4, pp. 471–480, July 1973.

[Stallings 02] William Stallings. Computer organization and architecture: Designing for performance. Prentice Hall, 6 edition, 2002.

[Symes 04] Peter Symes. Digital video compression. McGraw-Hill, 2004.

[Wyner 76] Aaron Wyner and Jacob Ziv. *The Rate-Distortion Function for Source Coding with Side Information at the Decoder.* IEEE Transactions on Information Theory, Vol. IT-22, No. 1, pp. 1 – 10, January 1976.

[Yanmei 08] Qu Yanmei, Shunliang Mei and Yun He. *A Cost-effective VLD Architecture for MPEG-2 and AVS.* Journal of Signal Processing Systems, Vol. 52, 2008.

[Zhijin 96] Zhao Zhijin and Quian Huisheng. *Recrusive Algorithms for Discrete Cosine Transform.* Proceedings of ICSP, 1996.

Tecnológico de Monterrey, Campus Monterrey

30002007497332

http://biblioteca.mty.itesm.mx