H.264 CODEC blocks implementation on FPGA

Master thesis performed in Division of Electronic System

by

Umair Aslam

LiTH-ISY-EX--14/4815--SE

Linköping, Sweden 2014
H.264 CODEC blocks implementation on FPGA

Master thesis in Division of Electronic System
at Linköping Institute of Technology
by

Umair Aslam

LiTH-ISY-EX--14/4815--SE

Supervisor: Kent Palmkvist
Examiner: Kent Palmkvist
Linköping, Sweden
November 27, 2014
Titel
Title: H.264 CODEC blocks implementation on FPGA

Författare
Author: UMAIR ASLAM

Sammanfattning
Abstract

H.264/AVC (Advance Video Coding) standard developed by ITU-T Video Coding Experts Group (VCEG) and ISO/IEC JTC1 Moving Picture Experts Group (MPEG), is one of the most powerful and commonly used format for video compression. It is mostly used in internet streaming sources i.e. from media servers to end users.

This Master thesis aims at designing a CODEC targeting the Baseline profile on FPGA. Uncompressed raw data pixels are fed into the encoder in units of macroblocks. At the decoder side the compressed bit stream is taken and the original frame is restored. Emphasis is put on the implementation of CODEC at RTL level and investigate the effect of certain parameters such as Quantisation Parameter (QP) on overall compression of the frame rather than investigating multiple solutions of a specified block of CODEC.

Nyckelord
Keyword: CODEC, RGB, Quantisation, DCT, luma, video coding, RTL, MPEG, QP, compression ratio
Abstract

H.264/AVC (Advance Video Coding) standard developed by ITU-T Video Coding Experts Group (VCEG) and ISO/IEC JTC1 Moving Picture Experts Group (MPEG), is one of the most powerful and commonly used format for video compression. It is mostly used in internet streaming sources i.e. from media servers to end users.

This Master thesis aims at designing a CODEC targeting the Baseline profile on FPGA. Uncompressed raw data is fed into the encoder in units of macroblocks of 16×16 pixels. At the decoder side the compressed bit stream is taken and the original frame is restored. Emphasis is put on the implementation of CODEC at RTL level and investigate the effect of certain parameters such as Quantisation Parameter (QP) on overall compression of the frame rather than investigating multiple solutions of a specified block of CODEC.
Preface

This Master thesis presents the design and implementation of H.264 CODEC in VHDL and synthesized on Altera DE2-115 FPGA board. Thesis is performed at Electronic System Division of ISY at Linköping University, Sweden.

This report is focused on background and implementation details of CODEC. A conclusive comparison with some of the other implementations is presented at end. Alternative solutions and sidetracks are not discussed.

Acknowledgment

There are many people that have helped me during my thesis, to whom I would like to express my sincere gratitude. First a special acknowledgment goes to my examiner and supervisor Kent Palmkvist at the division of Electronic Systems, Linköping University.

I would also like to thank my friends Ahmed, Bilal, Awais for their support and encouragement. This thesis report was written on LibreOffice Writer. All figures have been drawn using LibreOffice Draw and Kolourpaint. Graphs have been plotted using Matlab.
Abbreviations:

AVC  Advance Video Coding
bps  bits per second
CABAC  Context-Adaptive Binary Arithmetic Coding
CAVLC  Context-Adaptive Variable Length Coding
DCT  Discrete Cosine Transform
DFT  Discrete Fourier Transform
EDA  Electronic Design Automation
fps  frames per second
HDTV  High Definition Television
IPTV  Internet Protocol Television
ISO  International Organization for Standardization
ITU  International Telecommunication Union
MB  Macroblock
MC  Motion Compensation
ME  Motion Estimation
MF  Multiplication Factor
mif  memory initialization file
MPEG  Motion Picture Expert Group
NAL  Network Abstraction Layer
QP  Quantisation Parameter
RGB  RED Green Blue
RTL  Register Transfer Level
# Table of Contents

1. Introduction ................................................................................................................................. 1
   1.1 Problem Specification ............................................................................................................. 1
   1.2 Objective ................................................................................................................................. 1
   1.3 Limitations ............................................................................................................................... 1
   1.4 Thesis Report ........................................................................................................................... 1

2. Background ................................................................................................................................... 3
   2.1 Introduction ............................................................................................................................... 3
   2.2 Sampling .................................................................................................................................. 3
      2.2.1 Spatial sampling ................................................................................................................ 5
      2.2.2 Temporal sampling .......................................................................................................... 5
   2.3 Frames ..................................................................................................................................... 5
   2.4 Color Space ............................................................................................................................... 6
      2.4.1 RGB .................................................................................................................................. 6
      2.4.2 YCbCr .............................................................................................................................. 6
   2.5 YCbCr sampling format ............................................................................................................ 7

3. H.264 Standard ................................................................................................................................. 11
   3.1 Overview of video CODEC .......................................................................................................... 11
   3.2 H.264/AVC ................................................................................................................................ 13
   3.3 Slices ........................................................................................................................................ 14
   3.4 Profiles ..................................................................................................................................... 15
   3.5 Transform ................................................................................................................................ 15
      3.5.1 DCT .................................................................................................................................. 16
      3.5.2 Hadamard Transform ....................................................................................................... 16
   3.6 Inverse Transform .................................................................................................................... 17
      3.6.1 Inverse DCT ...................................................................................................................... 17
      3.6.2 Inverse Hadamard Transform ......................................................................................... 18
   3.7 Quantisation ............................................................................................................................... 19
      3.7.1 DC Quantisation ............................................................................................................... 20
   3.8 Inverse Quantisation ................................................................................................................ 21
      3.8.1 Inverse DC Quantisation ................................................................................................. 21
   3.9 Prediction .................................................................................................................................. 22
      3.9.1 Intra Prediction ................................................................................................................. 22
   3.10 Reordering ............................................................................................................................... 23
   3.11 Addition / Subtraction .............................................................................................................. 24

4. HARDWARE IMPLEMENTATION ................................................................................................. 25
   4.1 Tools & Technology .................................................................................................................. 25
      4.1.1 Software ........................................................................................................................... 25
Illustration Index

Figure 2.1: Spatial redundancy in image........................................................................................................4
Figure 2.2: Spatial and temporal sampling [1]..................................................................................................4
Figure 2.3: Video structure [7]..................................................................................................................5
Figure 2.4: Different sampling patterns [1]....................................................................................................8
Figure 2.5: YV12 arrangement of data in memory [2]....................................................................................9
Figure 2.6: IMC4 arrangement of data in memory [2]...................................................................................10
Figure 3.1: Video communication system....................................................................................................11
Figure 3.2: Generic video CODEC [1][7].......................................................................................................12
Figure 3.3: H.264 encoder [1]....................................................................................................................13
Figure 3.4: H.264 decoder [1].....................................................................................................................14
Figure 3.5: Slice arrangement in frame [5].....................................................................................................15
Figure 3.6: Zigzag scan for 4x4 matrix [1]....................................................................................................23
Figure 4.1: H.264 CODEC hardware implementation block diagram............................................................27
Figure 4.2: Source code for Data-type declaration.........................................................................................28
Figure 4.3: DCT transform block architecture..............................................................................................30
Figure 4.4: Source code for DCT memory indexing.......................................................................................31
Figure 4.5: Source code for MF values in quantisation block.........................................................................32
Figure 4.6: Quantisation block architecture..................................................................................................32
Figure 4.7: State diagram of CODEC...........................................................................................................34
Figure 4.8: State machine & signals................................................................................................................35
Figure 4.9: Source code for V selection in inverse-quantisation block..............................................................37
Figure 4.10: Inverse-quantisation block architecture....................................................................................37
Figure 4.11: Inverse-transform block architecture.........................................................................................38
Figure 5.1: Matlab plot for individual 4x4 blocks & combined 16x16 macroblock..........................................42
Figure 5.2: Matlab plot for the Forward path..................................................................................................43
Figure 5.3: Matlab plot for DC block in the Forward & Reverse path..............................................................43
Figure 5.4: Matlab plot for the Reverse path..................................................................................................44
Figure 5.5: Matlab plot for effect of QP on compression .............................................................................45
Figure 5.6: Frame-1 of video file, with grids enabled.......................................................................................55
# Index of Tables

Table 1: Quantisation step size........................................................................................................... 19
Table 2: PF value according to matrix index [6].................................................................................... 20
Table 3: Multiplication factor (MF) [1].................................................................................................. 21
Table 4: Scaling factor (V) [1]............................................................................................................. 22
Table 5: Intra prediction modes........................................................................................................... 22
Table 6: Reordering of coefficients..................................................................................................... 23
Table 7: Pin assignment for FPGA...................................................................................................... 26
Table 8: Resource utilization comparison in different FPGAs............................................................... 48
Table 9: Properties of video file.......................................................................................................... 55
Table 10: YUV coefficients of selected block at different frames......................................................... 58
1. INTRODUCTION

Increase in video quality standards over the past few years demand new techniques and algorithms to manipulate high data bandwidth. H.264 is a relatively new video compression standard, which delivers better compression ratio compared to its counterpart standards along with other useful features. This thesis aims at designing a Baseline profile-3 CODEC with resolution of $720 \times 480$, in HDL and synthesize it onto an FPGA board.

1.1 Problem Specification

Capture raw uncompressed video data and pass it through CODEC. The bit-stream of the compressed data comes at output of CODEC, which can be further put into a separate decoder, although decoder architecture is already present in the CODEC in the Reverse path. Design of the CODEC is done at RTL level and simulated in ModelSim to verify correct functionality. Finally the design is synthesized on FPGA board.

1.2 Objective

Main objective of the thesis is to design H.264 encoder and decoder using minimal amount of hardware. Run the design at different Quantisation Parameters, QP and study affect of the compression process.

1.3 Limitations

Thesis deals with the implementation of a CODEC, aiming at the Baseline profile rather than targeting one specific block of the CODEC and investigate it. For prediction purposes, only Intra $16 \times 16$ prediction mode is used. Entropy coding and filter implementation are out of scope for this thesis implementation.

1.4 Thesis Report

Thesis report comprises of 5 chapters. Outline of each chapter is given below:

Chapter 1: **Introduction**, Briefly describes an overall scenario, problem definition and limitations of this project work.

Chapter 2: **Background**, This chapter covers essential background material for
digital video aspects including color space, sampling and bits per pixel.

Chapter 3: **H.264 Standard**, This chapter starts with overall structure of a generic video CODEC and then refined the concepts to H.264 standard. Major components of the CODEC such as transformation, quantisation, prediction are discussed in detail.

Chapter 4: **Hardware Implementation**, In this chapter both the encoder and decoder design aspects are taken. Hardware implementation of each block in the CODEC is discussed.

Chapter 5: **Results and Discussion**, Summarizes the result taken from changing different parameters and their affect on overall compression process. Comparison with some of the other implementations is discussed.
2. BACKGROUND

2.1 Introduction

With the widespread of technological advancements, especially in field of electronics and communication, devices like HDTV, DVD and IPTV are exponentially increasing across the globe. New and advance technologies are evolving for high data transmission. To do so, compression is required to transport big data. Especially video requires efficient compression algorithms.

Compression can be achieved by removing redundancy [1]. Digital video compression algorithms (CODEC) works as the backbone of most video handling devices. The CODEC can either be implemented in hardware, mostly in the form of hardware accelerator, or in software. This chapter will cover the basis of essential background material necessary required to understand any modern video compression algorithm.

2.2 Sampling

Getting digital video from source and transfer it to desired destination while compressing it is the main job of the encoder [1]. At the destination, the encoded data is decoded and original frame is once again retrieved. This whole process contains several steps, which will be discussed in the next chapter. The main goal of this whole exercise is to reduce bandwidth to a manageable size, while maintaining acceptable video quality.

In the video encoder, compression is achieved by removing redundancy in temporal, spatial and/or frequency domains. By removing redundancy, information can be lost [1]. So video algorithms that have higher compression ratio, there is more tendency of data-loss (distortion) when frames are reconstructed at decoder. Natural video scenes are highly correlated. There are big blocks of homogeneous area in frame. An efficient encoder exploits this feature to achieve compression. Figure 2.1 shows certain areas of a frame, where nearby pixels are highly correlated. When coding these areas, it is possible to represent these areas by big macroblocks that require small motion vectors. As adjacent pixels here are very close, so their difference is approximate to zero. In these homogeneous areas, spatial redundancy is high.
Chapter 2.

Transforms especially the Discrete Cosine Transform (DCT), is very effective in homogeneous parts of frame.

![Image of a scene with a homogenous region highlighted]

**Figure 2.1:** Spatial redundancy in image

When a scene is captured in camera, it is in the form of a frame. A continuous sampling of frames over a period of time produces a video. Sampling is repeated at different intervals e.g (1/25, 1/30 seconds interval) [1].

![Diagram of spatial and temporal sampling]

**Figure 2.2:** Spatial and temporal sampling [1]
Figure 2.2 shows a typical sequence of a video file, where spatial redundancy is found within the frame and temporal redundancy in continuous flow of the frames.

**2.2.1 Spatial sampling**

In spatial sampling, a single frame is divided into multiple rectangular blocks. Each block has its own color and brightness characteristics. The number of rectangular blocks in frame determine the overall quality of the frame.

**2.2.2 Temporal sampling**

In temporal sampling, a rectangular frame is captured over a period of time. The higher the frame rate, better is the video quality and vice versa. Similarly more the frame rate, higher is the data bandwidth. Frame rate lower than 10 are sometimes used in low bit-rate video communication [1].

**2.3 Frames**

Multiple frames when played over a time period makes a video. So a single frame is just a snapshot of a picture at a specific time in a video file. Each frame is subdivided into rectangular lines called a grid. A frame has certain characteristics including width, height, bits per pixel etc. Number of grid lines determine the height of the frame while the length of the grid tells about the width of the frame. Each grid comprises of a group of data. The smallest unit of this data is called a pixel.

![Diagram of video structure](http://example.com/image.png)

**Figure 2.3:** Video structure [7]
Although a pixel is the smallest unit in video encoding, in most of the video coding standards macroblock is considered as the basic unit. A macroblock can range from 16×16 down to 8×8 and further 4×4 combination. Figure 2.3 shows hierarchical order of a video file.

2.4 Color Space

Digital video is subdivided into two categories, i.e., monochrome and color video. A monochrome image needs no additional information besides the brightness or luminance for each pixel. A color image requires more than one component to represent a single pixel. Mostly three components are required to represent a single pixel. Two popular categories of color spaces are RGB and YCbCr.

2.4.1 RGB

In RGB color space, a single pixel is represented by three values. As the name suggests they are the three different colors red, green and blue. These colors have different weight to represent a single pixel. Any other color can be derived by changing the proportion of these three colors.

2.4.2 YCbCr

YCbCr is another way of representing color images, where Y is the luma component. Cb represents blue and Cr represents red component. YCbCr is also termed as YUV format, where Y represents luma component. Chroma components blue and red are represented by U and V respectively. The luma component (Y) can also be derived from RGB color space by using Equation (1)

\[ Y = (K_r \times R) + (K_g \times G) + (K_b \times B) \]  

Where K is the weighing factor and is represented by the following Equation (2). ITU-R recommendation defines \( K_b = 0.114 \) and \( K_r = 0.299 \).

\[ K = (K_b + K_r + K_g) = 1 \]  

YCbCr is a more efficient way to represent the color space as Cr and Cb component can be represented by lower resolution as compared to luma (Y), as human eye is more sensitive to brightness than color [1]. In this way both color components can be
represented by less number of bits. This characteristic of the YCbCr color space

gives it more freedom in sampling format that will be discussed in next section. Data

in RGB color space can be converted to YCbCr and vice versa. As \( K_g \) can be
calculated by using the Equation (2), so it does not need to be stored or transmit. So

Equation (1) is modified as shown in Equation (3).

\[
Y = K_r(R) + (1 - K_b - K_r)G + K_bB
\]  

(3)

The chroma components can be calculated by using the Equation (4) and Equation

(5).

\[
C_b = \frac{0.5}{(1 - K_b)}(B - Y)
\]  

(4)

\[
C_r = \frac{0.5}{(1 - K_r)}(R - Y)
\]  

(5)

Equation (3),(4) and (5) are used to convert to YCbCr color space from RGB color

space.

Usually RGB image is converted to YCbCr format after capturing, in order to reduce

storage space and/or transmission requirements.[1] The resulting image in YCbCr is

converted back to RGB color space before displaying.

2.5 YCbCr sampling format

Most common sampling formats used in YCbCr are 4:4:4, 4:2:2 and 4:2:0. Although

all of these three patterns have same components, luma (Y), red (Cr) and blue (Cb),

their sampling frequency differs. 4:4:4 means for each luma sample there is

corresponding number of Cr and Cb components also. So all have the same

sampling frequency. 4:4:4 sampling format is very similar to RGB color space, as it

uses same amount of data to represent an image. Second pattern is 4:2:2. Here the

chrominance components have same vertical resolution when compared with luma,

but horizontal sampling is half as compared to luma. The last format is 4:2:0 which is
used in the thesis. In this format both the chroma red and chroma blue has half the horizontal as well as vertical resolution compared to luma. So for every four luma samples, there is one chroma red and chroma blue sample each.

![Different sampling patterns](image)

*Figure 2.4: Different sampling patterns [1]*
Usually each pixel value is represented in 8-bits. A group of four pixels in 4:4:4 sampling, from Figure 2.4 requires 96 bits to represent. As $12 \times 8 \text{bits}$ equals 96, and each of the pixels requires $96/4 = 24$ bits per pixel. Similarly 4:2:0 sampling requires 12 bits to represent a single pixel [1].

When data is taken from memory, it is arranged in a specific format. The number of bytes from one row of pixels in memory to the next row of pixels in memory is called stride [2]. For example in YV12, luma samples are arranged in a continuous array of strides, followed by red and then blue samples as shown in Figure 2.5. The stride length of chroma samples is half, as compared to luma.

![Figure 2.5: YV12 arrangement of data in memory [2]](image)

Similarly in IMC4 format luma samples appear first. They are followed by blue and red components. Each full-stride line in chroma area starts with the blue samples, followed by red samples, that begins at next half stride boundary as shown
in Figure 2.6. IMC2 format is identical to IMC4, except the red and blue components swap their position. [2]

**Figure 2.6:** IMC4 arrangement of data in memory [2]
3. **H.264 STANDARD**

A natural video scene is a continuous stream of frames, sampled over a time period. When representing in digital domain, each frame has a length and width. This is also known as dimensions of the frame. Whole frame is represented by a group of pixels commonly called Macroblock (MB). Macroblock usually range from $16 \times 16$ pixels, down to $8 \times 8$ or pixels. These macroblocks are passed through encoder to compress data. Different techniques both in spatial as well as temporal domain are used for this purpose. In this chapter general aspects of a video CODEC, its major components and their role will be discussed. Then discussion will be further targeted to H.264 video coding standard.

![Diagram of Video Communication System](image)

**Figure 3.1**: Video communication system

### 3.1 Overview of Video CODEC

An encoder performs video compression in order to reduce the amount of data provided by a source signal. The compressed signal is passed to a decoder which uncompressed it in order to reconstruct it back at the destination. There are certain rules and standards which both encoder and decoder are obliged to follow in order to perform their duty effectively. These rules are set by company or a group of experts which design the CODEC. The generic form of a video protocol is shown in Figure 3.1. Main goal of CODEC is to reduce data bandwidth as well ensuring high quality. These goals of compression while retaining high quality are usually conflicting [1], as higher compression ratio leads to lower quality of the video signal and vice versa.
A general video encoder consists of the following components, as can be seen in Figure 3.2

- Transform
- Quantisation
- Reordering
- Entropy coding
- Prediction

After the compression, the bit-stream at the output of the encoder can either be transmitted over a network or stored in memory. At the decoder side, decompression takes place. The video frame is reconstructed from the compressed bit stream by using the following components.

- Entropy decoding
- Ordering
- Inverse quantisation
- Inverse transform
- Constructing frame from prediction-motion-vectors.

**Figure 3.2:** Generic video CODEC [1][7]
3.2 H.264/AVC

As communication standards are maturing with time, so are the applications using them. Video streaming is one such application. Evolution of wireless networks from GSM, GPRS to 3G and then 4G standards have increased throughput of networks. So more efficient multimedia streaming is possible with the help of efficient communication standards and advance video compression algorithms.

Currently there are many image and video coding standards such as JPEG, MPEG-2, MPEG-4. In 2003, H.264/AVC (also known as MPEG part 10) was developed jointly by ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). It has gained higher compression ratio as compared to its predecessor. Compared with older video standards, bit-rate savings of 40% or more are reported [3]. However, the improvement in performance also cause increase in computational complexity, so more complex hardware and software is required to do the job.

Each 4×4 block of luma samples and associated chroma samples are fed into the encoder. After transformation and quantisation, they are reordered and finally entropy coded, as shown in Figure 3.3. The reverse path is used for inverse quantisation and

Figure 3.3: H.264 encoder [1]
inverse transform. In this way an approximate of actual image is formed in encoder, which is used in prediction. For Inter prediction, previous reference frame(s), formed through coded samples in reverse path are used. For Intra mode, prediction-vectors are calculated using the current frame samples, which have earlier been coded. The prediction is subtracted from the input samples as shown in Figure 3.3.

The decoder receives compressed bit stream and entropy decodes the data [1]. After inverse quantisation and inverse transform, samples are added with the prediction vectors to form frame. Block diagram of the decoder is shown in Figure 3.4.

![Figure 3.4: H.264 decoder [1]](image)

### 3.3 Slices

A picture can be split into smaller units called slices as shown in Figure 3.5. There can be one or several slices in a picture [4]. These slices are composed of macroblocks. Combining the macroblocks in slices helps in coding different modes. These slices are defined with coding modes e.g I slice, P slice, B slice etc. For example in I slice, all macroblocks are intracoded [5].
Chapter 3.

H.264 Standard

3.4 Profiles

The profile defines a specific set of functions, defined for a specific set of applications. The three profiles supported by H.264 are Baseline, Main and Extended. The Baseline profile is the simplest, offering support for inter and intra coding (I,P slices) as well as entropy coding with context-adaptive variable length codes (CAVLC). The Main profile includes interlacing, support for B-slices and entropy coding using context-based arithematic coding (CABAC). The Extended profile further supports for SP and SI slices and improved error resilience. [1].

3.5 Transform

The first stage involves transforming data from one domain to another. This process is called Transformation. There are various transforms proposed for image and video compression, but most popular are Discrete Cosine transform (DCT) and Discrete wavelet transform (DWT). In H.264 there are three different types of transforms [6].

1. DCT based transform for each 4×4 block.
2. Hadamard transform for 4×4 block. (Intra 16×16 DC values)
3. Hadamard transform for 2×2 block. (Cr,Cb DC values)

Figure 3.5: Slice arrangement in frame [5]
3.5.1 DCT

Discrete Cosine Transform operates on $X$, a block of $N \times N$ samples and creates a block $Z$ of same dimension. Following is the procedure for DCT based transform.

$$Z = AXA^T$$  \hspace{1cm} (6)

where

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}$$

So the above Equation (6) becomes

$$Z = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix}$$

Similarly Inverse Discrete Cosine Transform (IDCT) can be defined by Equation (7).

$$X = A^TZA$$  \hspace{1cm} (7)

3.5.2 Hadamard Transform

Hadamard transform is used to code DC blocks in Intra prediction. DC blocks are gathered after the DCT transformation prior to the Hadamard transformation. Given below is the Hadamard transform for $4 \times 4$ luma DC coefficients, where $X$ represents block of $4 \times 4$ DC coefficients.

$$Z = (BXB^T)/2$$  \hspace{1cm} (8)

where $B = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$

So Equation (8) becomes
\[ Z = \left( \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) \cdot \left( \begin{bmatrix} X \\ 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) / 2. \]

DC coefficients of each 4×4 chroma components are gathered in a 2×2 matrix, which is then transformed using the Hadamard transform.

Unlike luma, where DC transform only takes place if predicted in the Intra 16×16 mode, chroma values always have a DC transform.

\[ Z = C \cdot X \cdot C^T \quad (9) \]

where

\[ C = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \]

So Equation (9) becomes

\[ Z = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot [X] \cdot \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \]

where \( X \) is DC coefficients of chroma.

### 3.6 Inverse Transform

Like Transform, Inverse-transform also splits into Inverse DCT and Inverse Hadamard transform. Both are explained below.

#### 3.6.1 Inverse DCT

Inverse Discrete Cosine Transform operates on \( Z \), a block of \( N \times N \) samples and creates a block \( X \) of same dimension. Following is the procedure for inverse-transform.

\[ X = A^T \cdot Z \cdot A \quad (10) \]
where

\[
A = \begin{bmatrix}
1 & 1 & 1 & 1/2 \\
1/2 & -1 & -1 & 1/2 \\
1 & -1/2 & -1 & 1 \\
1 & -1 & 1 & -1/2 \\
\end{bmatrix}
\]

So above Equation (10) becomes

\[
X = \begin{bmatrix}
1 & 1 & 1 & 1/2 \\
1/2 & -1 & -1 & 1/2 \\
1 & -1/2 & -1 & 1 \\
1 & -1 & 1 & -1/2 \\
\end{bmatrix}
\begin{bmatrix}
1 & 1/2 & -1/2 & 1/2 \\
1 & -1 & -1 & 1 \\
1/2 & -1 & 1 & -1/2 \\
\end{bmatrix}
\]

### 3.6.2 Inverse Hadamard Transform

Inverse Hadamard transform is used to decode DC blocks if Intra prediction mode is used [6]. DC blocks are gathered after DCT transformation prior to Hadamard transformation. Given below is the inverse Hadamard transform for $4 \times 4$ luma DC coefficients, where $X$ represents block of $4 \times 4$ DC coefficients.

\[
Z = BXB^T
\]

where

\[
B = \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & -1 & -1 & 1 \\
1 & -1 & 1 & -1 \\
\end{bmatrix}
\]

So Equation (11) becomes

\[
Z = \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & -1 & -1 & 1 \\
1 & -1 & 1 & -1 \\
\end{bmatrix}
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & -1 & -1 \\
1 & -1 & -1 & 1 \\
1 & -1 & 1 & -1 \\
\end{bmatrix}
\]

For chroma samples, Equation (9) is used to compute Inverse Hadamard Transform.
3.7 Quantisation

Quantisation is a mathematical operation used in compression algorithms. The main aim of the quantiser is to reduce the range of coefficients, mapping them to specific ranges. This step also reduces precision. In video CODECs, quantisation takes place in two steps. A forward quantiser used in the encoder and an inverse quantiser in the decoder [1].

The quantiser in H.264 is controlled by the Quantisation Parameter (QP). It is the step size between two successive values. If the step size is large, the range of quantised value is small giving a higher compression and vice versa. The output of the forward quantiser is an array of coefficients mostly converging to zero.

Given below is the mathematical form of quantisation.

\[ A_{ij} = \text{round} \left( \frac{B_{ij}}{Q_{\text{step}}} \right) \]  

(12)

Where \( B_{ij} \) is data after transformation.

There are 52 QP values, each having its corresponding Qstep value as shown in Table 1 [6].

<table>
<thead>
<tr>
<th>QP</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
<th>51</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qstep</td>
<td>0.63</td>
<td>0.59</td>
<td>0.81</td>
<td>0.88</td>
<td>1</td>
<td>1.13</td>
<td>1.25</td>
<td>1.38</td>
<td>1.625</td>
<td>...</td>
<td>224</td>
</tr>
</tbody>
</table>

Table 1: Quantisation step size

To avoid division, Equation (12) is modified as

\[ A_{ij} = \text{round} \left( B_{ij} \cdot \frac{P_F}{Q_{\text{step}}} \right) \]

where PF varies according to coefficient position in matrix. Its value can be obtained, from Table 2.
Table 2: PF value according to matrix index [6]

<table>
<thead>
<tr>
<th>PF</th>
<th>Position (i,j)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>(0,0), (0,2), (2,0), (2,2)</td>
</tr>
<tr>
<td>0.4</td>
<td>(1,1), (1,3), (3,1), (3,3)</td>
</tr>
<tr>
<td>0.32</td>
<td>others</td>
</tr>
</tbody>
</table>

as

$$\frac{PF}{Q_{\text{step}}} = \frac{MF}{2^{\text{qbits}}}$$

and

$$\text{qbits} = 15 + \text{floor} \left( \frac{Q\text{P}}{16} \right)$$  \hspace{1cm} (13)$$

So

$$A_{ij} = \text{round} \left( B_{ij} \times MF + f \right) \ll \text{qbits}$$  \hspace{1cm} (14)$$

where f is $2^{\text{qbits}/3}$ for Intra prediction.

and

f is $2^{\text{qbits}/6}$ for Inter prediction.

### 3.7.1 DC Quantisation

For DC values, the process of quantisation changes slightly. For luma and chroma, DC coefficients are quantised using Equation (15).

$$A_{ij} = \text{round} \left( B_{ij} \times MF_{\text{zero}} + 2f \right) \ll \left( \text{qbits} + 1 \right)$$  \hspace{1cm} (15)$$

where $MF_{\text{zero}}$ is the multiplication factor at matrix index (0,0). So value of MF depends only on QP and not on the position in the matrix.
H.264 Standard

Chapter 3.

Table 3: Multiplication factor (MF) [1]

<table>
<thead>
<tr>
<th>QP</th>
<th>Position (0,0), (0,2), (2,0), (2,2)</th>
<th>Position (1,1), (1,3), (3,1), (3,3)</th>
<th>Position others</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>13107</td>
<td>5243</td>
<td>8066</td>
</tr>
<tr>
<td>1</td>
<td>11916</td>
<td>4660</td>
<td>7490</td>
</tr>
<tr>
<td>2</td>
<td>10082</td>
<td>4194</td>
<td>6554</td>
</tr>
<tr>
<td>3</td>
<td>9362</td>
<td>3647</td>
<td>5825</td>
</tr>
<tr>
<td>4</td>
<td>8192</td>
<td>3355</td>
<td>5243</td>
</tr>
<tr>
<td>5</td>
<td>7282</td>
<td>2893</td>
<td>4559</td>
</tr>
</tbody>
</table>

3.8 Inverse Quantisation

Inverse quantisation takes place according to the following Equation.

\[ Z_{ij} = \text{round} \left( X_{ij} \times V_{(0,0)} \times 2^{\text{floor}(Qp/6)} \right) \]

where \( V = Q\text{step} \times PF \times 64 \)

Value of \( V \) for QP range from 0 to 5 are shown in table Table 4.

3.8.1 Inverse DC Quantisation

For Luma DC 4×4 matrix, inverse quantisation takes place according to the following Equation.

\[ Z_{ij} = \text{round} \left( X_{ij} \times V_{(0,0)} \times 2^{\text{floor}(Qp/6)-2} \right) \quad \{\text{for } Qp > 12\} \]

For Chroma DC 2×2 matrix, inverse quantisation takes place according to the following Equation.

\[ Z_{ij} = \text{round} \left( X_{ij} \times V_{(0,0)} \times 2^{\text{floor}(Qp/6)-1} \right) \quad \{\text{for } Qp \geq 6\} \]
3.9 Prediction

All macroblocks in H.264 are predicted either using Inter prediction or Intra prediction. In Inter mode, prediction is made by motion-compensation of one or more frames stored earlier. In Intra mode, prediction is formed on samples that have previously been coded[1]. In either case, this prediction is subtracted from current macroblock which is then transformed, quantised and sent to decoder, along with the prediction vectors. The decoder make an identical prediction based on motion vectors.

3.9.1 Intra Prediction

Intra prediction is further divided into Intra $4 \times 4$ and Intra $16 \times 16$. Intra $4 \times 4$ mode is suitable for areas with significant detail while Intra $16 \times 16$ mode is more suitable for smooth areas of picture [4]. This thesis deals with only Intra $16 \times 16$ prediction. If the Intra $16 \times 16$ mode is used, the prediction matrix is formed using the current coefficients which have been encoded and then decoded in current frame [7]. Four modes are available.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 : vertical</td>
<td>Upper samples of previous macroblock are used</td>
</tr>
<tr>
<td>1 : horizontal</td>
<td>left samples of previous macroblock are used</td>
</tr>
<tr>
<td>2 : DC</td>
<td>Mean of vertical &amp; horizontal samples of previous macroblock are used</td>
</tr>
<tr>
<td>3 : plane</td>
<td>Function for vertical &amp; horizontal samples of previous macroblock are used</td>
</tr>
</tbody>
</table>

Table 5: Intra prediction modes
3.10 Reordering

The output of the quantisation block is mapped in a certain order, to group together nonzero coefficients. This enables efficient representation of quantised coefficients. The output is an array of coefficients comprising of a DC value at start followed by few integers and than long chain of zeroes. Given below is the zigzag scan path to order $4 \times 4$ quantised matrix [1].

![Zigzag scan for 4x4 matrix](image.png)

Consider a matrix as shown below

$$
\begin{bmatrix}
-2 & 4 & 0 & -1 \\
3 & 0 & 0 & 0 \\
-3 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
$$

Coefficient of the above matrix will be arranged as shown below.

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reordered Values</td>
<td>-2</td>
<td>4</td>
<td>3</td>
<td>-3</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6: Reordering of coefficients
3.11 Addition / Subtraction

Subtraction is performed prior to the transform. The prediction matrix is subtracted from the input matrix. Similarly, addition is performed after the inverse transform where prediction matrix is added to the inverse transform matrix.
4. HARDWARE IMPLEMENTATION

Implementation includes both designing the modules in HDL and verify their functionality in software and then synthesize it onto an FPGA. Design and verification is done using EDA tools. Various factors like timing, power and area can be calculated before actual hardware is implemented. Although optimization can be performed for speed, area or power, thesis work only focus on area.

4.1 Tools & Technology

There are number of simulators available to design and simulate behavior of HDL code. These simulators provide very close timing behavior, compared to actual hardware. Similarly there are various FPGAs from different companies. Many of the FPGA manufacturing companies also provide some tools as part of vendor's design suite as well as evaluation board.

4.1.1 Software

The thesis is carried out using VHDL. All modules are first designed and simulated independently to confirm their functionality. After that they are combined and simulated again to verify their behavior. ModelSim version 10.2b is used for simulation while Quartus II version 10.1 is used for synthesis.

Some of the basic building blocks used in the thesis are imported from the Altera MegaWizard plugin, found in the Quartus II tool [8]. These blocks are

- RAM 1-PORT
- ROM 1-PORT
- LPM_ADD_SUB
- LPM_MULT

As ModelSim is a third party tool, a couple of Altera libraries are imported into the ModelSim. These libraries are altera_mf and lpm. After design verification in ModelSim, it is synthesized using the Quartus. Major pin assignments are as follow:


### Table 7: Pin assignment for FPGA

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Direction</th>
<th>DE2-115 pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clk</td>
<td>In</td>
<td>PIN_Y2</td>
</tr>
<tr>
<td>resett</td>
<td>In</td>
<td>PIN_M23</td>
</tr>
<tr>
<td>ROM_STARTER</td>
<td>Out</td>
<td>PIN_G21</td>
</tr>
<tr>
<td>RAM_STARTER</td>
<td>Out</td>
<td>PIN_F17</td>
</tr>
<tr>
<td>QP_VECRT (5 DOWNTO 0)</td>
<td>In</td>
<td>PIN_AC26, PIN_AB27, PIN_AD27, PIN_AC27, PIN_AC28, PIN_AB28</td>
</tr>
</tbody>
</table>

#### 4.1.2 Hardware

For synthesis purposes, the Altera DE2-115 board is used. This board contains a CYCLONE IV EP4CE115 FPGA. Major features of this board which are used in the thesis are:

- Built-in USB Blaster for FPGA configuration
- 128 MB SDRAM, 2MB SRAM, 8MB Flash
- 18 toggle switches
- 18 red LEDs, 9 green LEDs
- Four debounced pushbutton switches
- 50 MHz oscillator

The project is clocked using a 50 MHz oscillator. After simulation and synthesis, the programmer window in Quartus is used to put the design file (.sof) into DE2-115 board. The In-System Memory Content Editor is used to analyze contents of ROM and RAM.
Figure 4.1: H.264 CODEC hardware implementation block diagram
4.2 Forward path

The H.264 CODEC can be divided into two paths, a Forward path and a Reverse path. Input pixels stored in the memory source are transformed and quantised. After quantisation the values goes to the reordering module and then to entropy coding as well as enter the Reverse path. In the Reverse path coefficients are inverse quantised and inverse transformed to form the prediction block. The red path in Figure 4.1 represents the Forward path.

4.2.1 Reading input

There are several ways to store raw data in memory as discussed in the section 2.5. Initially data was examined to choose the correct model, as the luma and chroma sample position differs with model. An uncompressed video file was chosen as input source. Data types used in the CODEC implementation are illustrated in Figure 4.2.

```
Package My_Datatype IS
  TYPE blockk IS (LUMA_NORMAL, LUMA_DC, CHROMA_NORMAL, CHROMA_DC);
  TYPE mat_4b4 IS ARRAY(0 to 15) OF STD_LOGIC_VECTOR(8 downto 0);
  TYPE mat_4b4_b IS ARRAY(0 to 15) OF STD_LOGIC_VECTOR(11 downto 0);
  TYPE array31elm IS ARRAY(0 to 31) OF STD_LOGIC_VECTOR(8 downto 0);
  TYPE array31b IS ARRAY(0 to 31) OF STD_LOGIC_VECTOR(11 downto 0);
  TYPE mat_4b IS ARRAY(0 to 3) OF STD_LOGIC_VECTOR(8 downto 0);
END My_Datatype;
```

**Figure 4.2:** Source code for Data-type declaration

Samples were stored in YV12 format. Data partitioning of pixels is illustrated in Figure 2.5. After examining the raw data, pixels were taken and stored in a ROM. Pixels were put in ROM by means of a memory initialization file (.mif). The ROM has 1536 memory location, with each location being 9 bit wide. A predefined ROM from Altera Mega-functions was chosen. This ROM can be initialized either by Intel-hex-file (.hex) or memory-initialization file (.mif). This memory-initialization file is attached to the ROM-unit by specifying the address of the file in the ROM-unit attributes [8].
Chapter 4. HARDWARE IMPLEMENTATION

The ROM has two input ports \textit{Clk} and \textit{Address}. There is one output port \textit{Dataout}, which is 9 bits wide. A ROM-controller specifies address for the ROM. This controller has a counter. First are the 256 luma samples sent. Then the controller halts for 96 cycles to allow data to be processed. When the DC-luma is calculated, the counter is again enabled to allow 64 more coefficients to be read from the ROM. Then is the counter halted again the ROM address, until the chroma red values are fully transformed. Same procedure is then applied again to chroma blue. When the chroma blue values are finally fetched, then whole process is repeated again, until the last index of ROM memory location.

As the ROM is a read only memory, to set a new pixel value, the data in the memory initialization file requires an update before start of the encoding process. For simulation purposes, pixels can be loaded from text file. This file is initially filled with pixels in it. The other way is to set \textit{.mif} file for the ROM. The ROM based approach is preferred, as it can be used for both simulation as well as synthesis purposes. Also it presents a more realistic model of the hardware system, where an address is generated to fetch data at every clock cycle. Taking input from a \textit{.txt} file do not require any address generation mechanism. Each value taken from the ROM is passed to the subtraction unit, which subtracts the corresponding prediction sample from the current input. Whole process is tightly synchronized, to pass the current index value for both input and prediction sample generators. Subtraction unit is a combinational logic circuit. Data at the input of the subtraction module appears at the output in same clock cycle.

4.2.2 Transformation

The data at the output of subtraction block appears as the input to the transformation block. There are three types of transformation that takes place in the H.264 depending on pixel type. These are discussed in detail in the section 3.5. First the DCT transformation is performed on every pixel regardless of its type. The DCT module first collects the coefficients in a temporary memory. When 16 coefficients are stored in memory equivalent to a $4 \times 4$ matrix, the first matrix multiplication takes place. In designing this part focus was put on to minimize use of multiplier/divider circuit. Only addition and subtraction are used to multiply the input matrix with the first DCT matrix coefficients. Four coefficient are taken from the input memory at a time and passed to a hierarchical adder as shown in Figure 4.3.
In this way the first row of the DCT-1 matrix is multiplied by the column of the input matrix and generate a partial product of one coefficient entry for the $4 \times 4$ partial matrix stored in a partial_memory. This continues 15 times to compute the DCT transform matrix. Both the input memory as well as the partial_memory operate in same way. They have 32 word depth. At any time, only one half which is 16, memory locations are used to calculate the DCT. Two signals $i$ and $p$ as illustrated in Figure 4.4, are added with the memory index to make the effective address.

As Intra $16 \times 16$ prediction mode is used, so each $4 \times 4$ DCT matrix generates one DC component, which is present at the $(0,0)$ location of $4 \times 4$ transformed matrix. In this way a $16 \times 16$ luma coefficient matrix generates a $4 \times 4$ matrix of the DC-luma coefficients.
The Hadamard transformation for DC coefficients take place in same way as normal DCT. The only difference is the matrix coefficients. So the same architecture used in DCT calculation, is also used here except changing control signals for the hierarchical adder unit. Finally the result of the Hadamard transformation are scalar divided by two. This can be achieved by a simple right shift.

### 4.2.3 Quantisation

Transform pixel coefficients are then fed into the quantisation module. H.264 assumes scalar quantisation [1]. So each coefficient is quantised according to its position in the macroblock. After quantisation, the strength of coefficient is greatly reduced. So it is the core part of any compression algorithm. Quantisation depends upon several factors, as described in the Equation (14), The most important is QP. For H.264 it has 52 values, according to which different parameters change. Table 1 show how Qstep changes according to the QP.

```vhdl
PROCESS (indexx,Clok)
BEGIN
  IF (Clok'EVENT AND Clok = '1') THEN
    IF (indexx = 15) THEN
      IF (i = 0) THEN i<= 16; p <= 0;
      ELSIF (i = 16) THEN i<= 0; p <= 16;
      END IF;
    END IF;
  END IF;
END PROCESS;
```

**Figure 4.4:** Source code for DCT memory indexing
CASE pixxel_Type IS
    WHEN LUMA_DC | CHROMA_DC =>
        MF_zero <= "010000000000000" ; -- 8192
        MF_one  <= "010000000000000" ; -- 8192
        MF_two  <= "010000000000000" ; -- 8192
        see_xl  <= '1';
    WHEN LUMA_NORMAL | CHROMA_NORMAL =>
        MF_zero <= "010000000000000" ; -- 8192
        MF_one  <= "000110100011011" ; -- 3355
        MF_two  <= "001010001111011" ; -- 5243
        see_xl  <= '0';
    WHEN OTHERS => NULL;
END CASE;

Figure 4.5: Source code for MF values in quantisation block

Figure 4.6: Quantisation block architecture
Default value of QP is set to 10 and implementation supports three QP values which are 10, 22 and 34. Corresponding other parameters are selected using the CASE statement, as illustrated in Figure 4.5.

Quantisation is a combinational module. Value placed at the input of the quantisation block appears at the output in same clock cycle. Whole procedure follows Equation (14). Each coming input coefficient is multiplied by the MF signal. The MF signal is 15 bits wide. Multiplication is carried out using a customized multiplier taken from the Altera standard LPM [8]. The output length of multiplier is 27 bits. This output is added with F and the result is shifted right according to Equation (14). The DC components from Intra $16 \times 16$ mode as well as from chroma are quantised according to Equation (15). The only difference is the MF value, which is always selected for (0,0) position regardless of coefficient position in the matrix. Also the shifting variable, qbits changes to qbits+1. The quantisation in the H.264 is a lossy process. Some information is lost during the process and this process is irreversible. The original signal cannot be retained, if inverse quantisation is applied to output of quantised coefficients.

4.2.4 State Machine

The state machine is the heart of whole project. It generates various signals which in turn control other modules in the CODEC. The state machine is implemented using a counter. After every sixteen cycles, the signal see_dc goes to 1 for a clock cycle. This signal serves as input to another counter which increments its signal, state_machine_counter. This increment takes place every time see_dc goes to 1. Following states are used.

- normal_luma_state
- dc_luma_state
- red_normal_chroma_state
- red_dc_chroma_state
- blue_normal_chroma_state
- blue_dc_chroma_state
Chapter 4. HARDWARE IMPLEMENTATION

Two signals `present_state`, `next_state` of type `state` are used, while the state machine initializes to `normal_luma_state`.

The function of the state machine is simple. At different counter values, the state changes which in turn changes other control signals of the CODEC. The flow of state machine is illustrated in Figure 4.7. After taking a 16×16 macroblock of luma samples, the state machine stops the ROM from further taking input coefficients in order to calculate DC luma. Same behavior is observed for chroma red and chroma blue samples. Only difference is, instead of taking 16×16=256 luma samples, for chroma it is 64 samples each as explained in section 2.5. Loading of quantised coefficients in the RAM is also controlled by the state machine.

The last signal controlled by state machine is `pixxell_type` of enumerated data type `blockk` as illustrated in Figure 4.2. This signal tells other modules in the design about the type of current coefficient. Timing behavior of complete cycle of the state machine is shown in Figure 4.8.

Figure 4.7: State diagram of CODEC
4.3 Reverse path

At the output of the quantisation block, there are two paths. One goes to the reordering module. The second path is known as reverse path where coded data is again decoded to insure data integrity with the decoder. In the Reverse path, coded coefficients are inverse quantised and inverse transformed. A prediction vector is added to inverse transformed coefficients to store the prediction block as illustrated in Figure 4.10. The green path constitutes the Reverse path.

4.3.1 Inverse DC transform and Inverse quantisation

Coefficients at output of the quantisation module constitutes input of Inverse DC Transform. Inverse Hadamard transform is applied to coefficient samples according to Equation (11) for luma coefficients and Equation (9) for chroma coefficients.

Inside the Inverse DC transform module, data at input is stored in a memory, inp_temp which can store 32 words of 9-bit each. Data from the inp_temp is fed to the hierarchical adder unit. Output of the hierarchical adder is stored in a second memory called partiall_memory. The partiall_memory have same storage capacity as inp_temp. A second hierarchical adder unit add/subtracts coefficients from the partiall_memory. The result is placed at output of the Inv_DC_transform block. The DC transform takes place before quantisation in the forward path, but order is not reversed as might be expected in the reverse path of CODEC [1]. As illustrated in Figure 4.1. Inverse_DC_transform block is placed before the Inverse quantisation block.

The Inverse-quantisation module is very similar to the quantisation module. Data placed at the input is inverse quantised and available at the output in the same clock cycle. Similarly as in the quantisation module, input coefficients are multiplied by MF, here MF changes to V. This V comprises of V_zero, V_one, and V_two depending upon coefficient position in the 4×4 matrix as shown in Figure 4.9. For Luma-Intra 16×16 and Chroma-DC, V is V_zero independent of the position of the coefficients in the matrix.

Inverse-quantisation is performed according to section 4.3.1, where input is multiplied by V. The result is then multiplied by the Qpby6 signal and put at the output of the inv_quant module as shown in Figure 4.10.
ARCHITECTURE QP_SELECT_arch of QP_SELECT IS
  SIGNAL my_qp : INTEGER RANGE 0 to 51 := 10;
BEGIN
  my_qp <= conv_integer (QP_VECTR);
  QP    <= my_qp;

  WITH my_qp SELECT
    V_ZERO  <= "00000100000"   WHEN 10 ,   -- 16*2=32
        "00010000000"   WHEN 22 ,   -- 16*8=128
        "01000000000"   WHEN 34 ,   -- 16*32=512
        "00000100000" WHEN OTHERS;   -- DEFAULT
            --CASE QP=10 (16*2=32)

    V_ONE   <= "00000110010"   WHEN 10 ,   -- 25*2=50
        "00011001000"   WHEN 22 ,   -- 25*8=200
        "01100100000"   WHEN 34 ,   -- 25*32=800
        "00000110010" WHEN OTHERS;   -- DEFAULT
            --CASE QP=10 (25*2=50)

    V_TWO   <= "00000101000"   WHEN 10 ,   -- 20*2=40
        "00010100000"   WHEN 22 ,   -- 20*8=160
        "01010000000"   WHEN 34 ,   -- 20*32=512
        "00000101000" WHEN OTHERS;   -- DEFAULT
            --CASE QP=10 (16*2=32)
  END QP_SELECT_arch;

Figure 4.9: Source code for V selection in inverse-quantisation block

Figure 4.10: Inverse-quantisation block architecture
4.3.2 Inverse transform

Coefficients at output of the inverse-quantisation module are first stored in a memory in order to put back the DC components at their respective index positions. When a complete macroblock is formed, it is sent to the inverse-transform module. The process is very similar to the transform module in working. The only difference is in the DCT matrix, as can be observed from Equation 10. Each coefficient is divided by 64 at the output of the inverse-transform module by using a right-shift operation. Figure 4.11 shows inverse-transform module which is similar to transform, except left-shift is replaced by right-shift operation.

4.4 RAM

The output of the quantisation module is stored in a custom designed memory module called “RAM”. It is a M9K memory block imported from the Altera MegaWizard, which features in the Cyclone IV devices [8]. This memory structure can be configured according to the user specifications. For this thesis, its data storing limits are set the same as that of input module ROM i.e., each location contains 9 bits, with total 1536 locations. So 13824 bit memory is used as RAM. Also it has the following characteristics.
Chapter 4. HARDWARE IMPLEMENTATION

• 1 data port for read, 1 data port for write.

• 1 address port.

• write-enable (wren) signal.

The RAM is controlled by the state machine. Output of the quantisation module is fed into the RAM, when set_ram is asserted high by the state machine.

Hardware resources used by the FPGA are shown in Appendix 1. Fmax is found to be 24.77 MHz. Memory-map generated by ModelSim version 10.2b is shown in Appendix 2. A random video file is run and one of its 8×8 block is analyzed at location (7,12), where first digit represents row and second digit represents column of the frame. While video is running, same frame is analyzed at different frame numbers in Appendix 3. Corresponding coefficients of this 8×8 block are shown in Table 10.
5. RESULTS AND DISCUSSION

As previously noted in section 4.1.2, the FPGA is clocked at 50MHz. The CODEC takes input pixels from the ROM. This module can store 1536 pixel coefficients. These pixels pass through the CODEC and the result at the output is stored in the RAM. Simulation results at the output of each module are compared with the expected results, stored in text files and the difference is calculated in the test-bench.

Results found at output of the transformation are 100% accurate, as simple add/subtract operations are involved. However, for the DC transform, there are two values out of the total 1536 values, whose difference is non-zero. A careful study of the signals involved in the DC transform reveals that during second multiplication cycle according to Equation (8), there is overflow occurring. This causes the output at the second multiplication stage to get corrupt. This can be prevented by increasing the width of signals from 12 bits to 14 or 15 bits and subsequently increasing the width of the other sub-blocks in hierarchy. Overall performance is satisfactory with 99.86% accuracy when compared to all 1536 samples and 96.875% when compared with the four DC blocks.

A random 16×16 block is taken as shown in Figure 5.1. Each 4×4 block is also shown separately. The behavior of the transformation and quantisation in the Forward path is illustrated in Figure 5.2. Y-axis of quantisation is scaled down ten times, as compared to the transformation. The result of the transformation and quantisation can be compared with the input in Figure 5.1. Similarly DC coefficients are Hadamard transformed and quantised in the Forward path, while its inverse takes place in the Reverse path as shown in Figure 5.3. In the Reverse path, inverse-quantisation and inverse-transform takes place as shown in Figure 5.4.

There is a significant change in the behavior of data at the output of the quantisation, due to the change in QP. As the QP is increased, data coefficients converge towards zero, as illustrated in Figure 5.5. Three different QP values i.e. 10, 22 and 34 are selected to observe the behavior of quantisation. This behavior is exactly according to the Equation (12), as an increase in the QP cause an increase in the qbits according to Equation (13). As a result, more number of bits are shifted towards right, resulting in decrease in the integer strength.
Figure 5.1: Matlab plot for individual 4x4 blocks & combined 16x16 macroblock
Figure 5.2: Matlab plot for the Forward path

Figure 5.3: Matlab plot for DC block in the Forward & Reverse path
Figure 5.4: Matlab plot for the Reverse path
Figure 5.5: Matlab plot for effect of QP on compression
5.1 Comparison

Implementation of H.264 is quite complicated due to its large number of profiles and levels. As functionality, performance and cost are variable, so different applications have different demand of profiles and levels. Although a lot of work has been done in video compression and particularly the H.264, finding a complete generic implementation of H.264 is very rare in academics. Mostly part of its architecture or some specific modules are designed, considering optimization for one or more of factors like speed, area, power, efficiency etc., However in industry there are many implementations of H.264 targeting different profiles and levels. They are FPGA based, as well as stand-alone ASIC IP cores.

[10] Propose a hardware implementation of H.264 encoder, with a majority of blocks implemented in this thesis also. These blocks are transformation (AC & DC), quantisation, inverse-quantisation, inverse-transformation (AC & DC), Hadamard forward and reverse transformation, Intra 16×16 with extra intra 4×4 prediction. Implementation is done on Altera Stratix II, EP2560F1020C3 FPGA at 100 MHz. For the intra 16×16 mode, number of cycles to compute one MB is 573, while the resource used are present in Table 8.

[11] Implements a hardware architecture of H.264/AVC for Intra 16×16 prediction. Major modules present in this architecture are integer transformation, Hadmard transformation, quantisation (AC & DC), inverse-quantisation (AC & DC) and inverse integer transform. Hardware is implemented using VHDL on Stratix II FPGA, clocked at 160 MHz. Results for comparison are present in Table 8.

[12] Presents implementation and verification of H.264/AVC encoder for HDTV applications, aiming Baseline profile with level 3.2. Design is implemented on Xilinx Vertex-6 board operating at 200 MHz frequency. Various blocks like motion estimation, fraction motion estimation, variable length coding, de-blocking filter, NAL coding, which are not part of this thesis are also implemented. Resource utilization is present in Table 8 for comparison with this thesis.

[13] Presents a scheme for two-dimensional DCT module used in H264. Here two same 1D-DCT modules are used for calculating 2D-DCT. Proposed architecture here can perform a DCT of 4×4 block in twelve cycles, while this thesis performs the
same job in sixteen cycles, with thirty-two cycle one time initial delay is present to initially fill the memory in DCT unit. Implementation is quite similar to this thesis work, where first 1-D DCT is carried out using a partial memory.

[14] Architecture for transformation is more efficient as it uses pipe-lining. It can perform $4 \times 4$ DCT in 12 cycles. In [14] 587 logic elements are used to implement DCT. In this thesis, the resource utilization for only the transform module is 1202 logic elements.

For commercial use, a large number of firms provide the H.264 FPGA based IP cores. These FPGA prototypes are very efficient and offer complete solutions from Baseline profile to High profile, according to the customer demands. Main advantage of these IP cores is that their architecture are based on FPGA, so they are flexible and offer great customization.

[15] Provides H.264/AVC Baseline HD encoder. This core can encode at full HD (1080p) or higher rates. Core can be configured to operate on Intra-only mode. Implementation results are shown in Table 8.

[16] Provides H.264 encoder. Three profiles i.e. Baseline profile, Main profile and High profile are supported. Cores can be configured for encoding of video up to level 5.2.

[17] Offers solution for H.264 by providing both encoder and decoder which support 4:2:0 / 4:0:0 / 4:2:2 / 4:4:4 color space. For encoder 300K gates are required, while for decoder 200K gates are required. Maximum performance is $3840 \times 2160$, for both FPGA & ASIC.

[18] This is a third party H.264 encoder provided by the Xilinx. It support profile level 3.1 with resolution up to $4096 \times 4096$. Implementation summary regarding resource usage is illustrated in Table 8.

[19] Offers H.264 core in two variations. The H.264E-I Intra profile, which is smaller and have less compression ratio, as compared to H.264E-P, which is larger but compression ratio is high. These cores can operate on frames, having resolution from $1280 \times 720$ to $3840 \times 2160$. Resource usage according to different FPGAs is illustrated in Table 8.
<table>
<thead>
<tr>
<th>Reference Implementations</th>
<th>Logic Elements (LE)</th>
<th>Memory</th>
<th>Maximum Frequency (MHz)</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thesis</td>
<td>6236</td>
<td>46761</td>
<td>50</td>
<td>Altera Cyclone IV</td>
</tr>
<tr>
<td>Design implementation on FPGA of H.264/AVC intra decision frame : [10]</td>
<td>28511</td>
<td>32 KB</td>
<td>100</td>
<td>Altera Stratix II</td>
</tr>
<tr>
<td>FPGA design for H.264/AVC encoder : [12]</td>
<td>37178</td>
<td>150</td>
<td>130</td>
<td>Altera Stratix III</td>
</tr>
<tr>
<td>A Pipelining Hardware Implementation of H.264 Based on FPGA : [14]</td>
<td>587</td>
<td>-</td>
<td>-</td>
<td>Altera Cyclone</td>
</tr>
<tr>
<td>Jointwave WDE960: [17]</td>
<td>300K</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Jointwave WDE960: [17]</td>
<td>240K</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>A2e Technologies, H.264 Encoder : [18]</td>
<td>9804</td>
<td>-</td>
<td>142</td>
<td>Virtex-6 LXT</td>
</tr>
<tr>
<td>VISENGI, H.264 Encoder IP core : [19]</td>
<td>68169</td>
<td>38222</td>
<td>-</td>
<td>Altera Cyclone IV</td>
</tr>
<tr>
<td>VISENGI, H.264 Encoder IP core : [19]</td>
<td>31313</td>
<td>41269</td>
<td>-</td>
<td>Altera CycloneV</td>
</tr>
</tbody>
</table>

**Table 8:** Resource utilization comparison in different FPGAs
5.2 Conclusion

The aim of this thesis was to implement H.264/AVC CODEC and understand operations of each module e.g. transform coding, quantisation, DC transform, prediction etc.

To better understand the design, the CODEC is split into separate modules. Individual modules are designed and tested separately. Instead of conventional matrix multiplication for each $4 \times 4$ block, `CASE` statements are used with a counter range from zero to fifteen. The counter values act same as the matrix index range from $(0,0)$ to $(3,3)$ for a $4 \times 4$ coefficient block. LPM_ADD_SUB IP core is used in each `CASE` statement to add four matrix entries, corresponding to a complete row/column at a time. Quantisation is tested for different QP values, in order to better estimate the compression. Each $4 \times 4$ block is rearranged, by putting back the respective DC components before the inverse transform in the Reverse path. Finally the Intra $16 \times 16$ is used as the prediction mode.

According to the experimental results, compression has a direct relationship with QP. Coefficients before quantisation remain same, as the transformation is standard for all the coefficients and there is no direct variable involved in the operations. As H.264 standard has multiple profiles and levels [3], it is hard to estimate the best architecture. The comparison presented in Table 8 shows how hardware resources vary with implementation on different FPGAs. Therefore it is difficult to claim the best H.264 CODEC in terms of performance, area and power.

5.3 Future Work

Compression is a modern technique and has a lot of room for improvements. In this thesis, only the Intra $16 \times 16$ prediction method is used. So Inter-prediction is another option available to further investigate affects of the prediction on overall compression ratio. QP value range from 0 to 51 according to Table 1. Only three QP values are tested in this thesis work. All values can be tested and compared to select the most optimized QP value. For future work, it is also proposed to implement the Entropy coding to get actual realization of bit-stream of 0's and 1's.
Bibliography


APPENDIX 1: HARDWARE RESOURCES USED IN FPGA

Quartus II Version: 10.1 Build 197 01/19/2011 SP 1 SJ Full Version

Revision Name: codecc

Top-level Entity Name: codecc

Family: Cyclone IV E

Device: EP4CE115F29C7

Timing Models: Final

Total logic elements: 6,236 / 114,480 (5%)
  - Total combinational functions: 6,118 / 114,480 (5%)
  - Dedicated logic registers: 691 / 114,480 (<1%)

Total registers: 691

Total pins: 12 / 529 (2%)

Total virtual pins: 0

Total memory bits: 46,761 / 3,981,312 (1%)

Embedded Multiplier 9-bit elements: 4 / 532 (<1%)

Total PLLs: 0 / 4 (0%)

Fmax calculated using TimeQuest Timing Analyzer is 24.77 MHz.
### APPENDIX 2: MEMORY MAP GENERATED BY MODELSIM 10.2B

<table>
<thead>
<tr>
<th>Instance</th>
<th>Range</th>
<th>Depth</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/codeccc/rom_unit/altsyncram_component</code></td>
<td>[0:2047]</td>
<td>2048</td>
<td>9</td>
</tr>
<tr>
<td><code>/codeccc/rom_unit/altsyncram_component</code></td>
<td>[1:0]</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td><code>/codeccc/Transform_unit/tak_inp</code></td>
<td>[0:31]</td>
<td>32</td>
<td>9</td>
</tr>
<tr>
<td><code>/codeccc/Transform_unit/partial</code></td>
<td>[0:31]</td>
<td>32</td>
<td>12</td>
</tr>
<tr>
<td><code>/codeccc/DC_transform_unit/partial</code></td>
<td>[0:15]</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td><code>/codeccc/DC_transform_unit/DC_inp</code></td>
<td>[0:5]</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td><code>/codeccc/DC_transform_unit/tak_inp</code></td>
<td>[0:18]</td>
<td>19</td>
<td>12</td>
</tr>
<tr>
<td><code>/codeccc/Ramm_unit/altsyncram_component</code></td>
<td>[2047:0]</td>
<td>2048</td>
<td>9</td>
</tr>
<tr>
<td><code>/codeccc/Ramm_unit/altsyncram_component</code></td>
<td>[1:0]</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td><code>/codeccc/Inv_DC_transform_unit/tak_inp</code></td>
<td>[0:31]</td>
<td>32</td>
<td>9</td>
</tr>
<tr>
<td><code>/codeccc/Inv_DC_transform_unit/partial</code></td>
<td>[0:31]</td>
<td>32</td>
<td>12</td>
</tr>
<tr>
<td><code>/codeccc/Inv_DC_transform_unit/DC_inp</code></td>
<td>[0:15]</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td><code>/codeccc/Register_store_unit/ReadWrite</code></td>
<td>[0:383]</td>
<td>384</td>
<td>16</td>
</tr>
<tr>
<td><code>/codeccc/Inverse_Transform_unit/tak_inp</code></td>
<td>[0:31]</td>
<td>32</td>
<td>16</td>
</tr>
<tr>
<td><code>/codeccc/Inverse_Transform_unit/partial</code></td>
<td>[0:31]</td>
<td>32</td>
<td>16</td>
</tr>
<tr>
<td><code>/codeccc/Intra_Prediction/INOUTPUT/INTRA</code></td>
<td>[0:1440]</td>
<td>1441</td>
<td>9</td>
</tr>
</tbody>
</table>
APPENDIX 3 : FRAME ANALYSIS

A video file is analyzed using the tool “YUVTools (version 3.0.892)”. Properties of the video file are illustrated in Table 9. Frame-1 is shown in Figure 5.6, with grids represent a $8 \times 8$ block in frame. A random block at location, column: 12 & row: 07 is selected and analyzed at different frames of video. Table 10 shows the YUV coefficients of that block at different frame numbers.

![Frame-1 of video file, with grids enabled](image)

**Table 9**: Properties of video file

<table>
<thead>
<tr>
<th>Sampling Format</th>
<th>Component order</th>
<th>Interlaced / Progressive</th>
<th>Packed / planar</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>YUV_420</td>
<td>YUV</td>
<td>progressive</td>
<td>planar</td>
<td>$352 \times 288$</td>
</tr>
<tr>
<td>Frame Number</td>
<td>YYY</td>
<td>UUU</td>
<td>VVV</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>160 156 157 156 156 154 156 159</td>
<td>139 138 138 138</td>
<td>119 119 120 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>170 164 167 167 164 164 166 165</td>
<td>139 139 138 138</td>
<td>119 118 119 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>167 167 167 168 166 167 166 164</td>
<td>127 133 137 138</td>
<td>129 122 119 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>135 135 136 152 159 158 157 159</td>
<td>117 122 133 139</td>
<td>137 131 125 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>155 160 159 152 156 162 161 162</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>234 239 219 173 159 167 165 163</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>255 226 190 190 178 170 166 163</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>200 170 170 179 148 140 154 155</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>166 157 155 156 159 157 159 157</td>
<td>138 139 138 137</td>
<td>121 118 120 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>171 166 164 165 165 167 167 164</td>
<td>138 140 138 138</td>
<td>121 118 119 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>158 168 168 169 166 165 166 166</td>
<td>127 133 138 138</td>
<td>130 123 119 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>132 135 137 151 158 156 159 159</td>
<td>119 120 132 139</td>
<td>137 133 125 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>158 161 158 151 156 161 162 162</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>221 239 220 175 160 167 166 165</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>242 230 191 190 178 171 170 167</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>199 175 171 180 148 140 156 158</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>166 161 154 156 156 155 158 156</td>
<td>135 139 138 138</td>
<td>123 119 120 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>163 170 163 164 164 162 162 163</td>
<td>135 139 137 138</td>
<td>122 118 119 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>146 164 169 169 164 163 164 165</td>
<td>127 133 137 137</td>
<td>129 123 119 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>132 133 140 153 157 156 160 160</td>
<td>121 121 132 139</td>
<td>136 132 124 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>165 166 160 151 157 162 162 164</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>201 242 221 172 161 168 165 165</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>208 221 196 188 176 170 169 165</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>194 171 173 178 145 139 158 156</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>150 166 156 157 155 156 159 157</td>
<td>131 138 138 138</td>
<td>125 119 119 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>142 172 166 166 165 166 167 165</td>
<td>131 139 138 138</td>
<td>126 118 118 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>136 157 168 168 166 166 167 167</td>
<td>126 133 138 138</td>
<td>131 123 119 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>137 131 138 152 157 156 161 158</td>
<td>122 121 132 140</td>
<td>134 132 124 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>174 170 165 153 156 160 162 162</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>186 228 228 174 161 166 165 166</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>191 199 196 187 176 171 170 166</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>185 174 174 176 147 142 157 153</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>138 164 161 155 157 158 158 155</td>
<td>129 138 139 138</td>
<td>127 119 119 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>125 160 169 164 165 164 165 165</td>
<td>129 138 139 138</td>
<td>128 119 118 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>132 146 165 168 163 161 165 167</td>
<td>127 133 138 138</td>
<td>131 123 118 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>141 132 139 154 156 155 160 160</td>
<td>123 122 132 140</td>
<td>133 132 125 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>179 170 164 154 156 162 164 163</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>186 204 221 177 159 164 165 164</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>193 180 192 193 176 169 170 165</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>167 183 181 180 149 143 157 155</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>143 149 164 156 156 157 158 157</td>
<td>127 137 139 138</td>
<td>128 121 118 120</td>
<td></td>
</tr>
<tr>
<td></td>
<td>130 140 170 166 165 164 165 164</td>
<td>127 137 138 138</td>
<td>130 121 118 120</td>
<td></td>
</tr>
<tr>
<td></td>
<td>133 136 157 168 164 162 163 166</td>
<td>126 133 137 138</td>
<td>131 124 118 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>144 137 138 154 156 157 158 159</td>
<td>124 123 131 139</td>
<td>131 132 125 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>185 177 163 157 157 162 164 162</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>194 192 200 177 159 164 165 163</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>193 186 193 194 176 170 169 164</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>151 181 189 179 148 144 157 154</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame Number</td>
<td>YYY</td>
<td>UUU</td>
<td>VVV</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>177 189 165 138 162 162 157 158</td>
<td>126 125 134 139</td>
<td>128 129 123 117</td>
<td></td>
</tr>
<tr>
<td></td>
<td>162 171 151 124 157 170 161 163</td>
<td>125 125 135 139</td>
<td>131 131 124 118</td>
<td></td>
</tr>
<tr>
<td></td>
<td>135 129 132 131 142 162 164 165</td>
<td>127 126 133 139</td>
<td>130 130 123 118</td>
<td></td>
</tr>
<tr>
<td></td>
<td>152 146 143 141 139 152 160 157</td>
<td>127 124 132 139</td>
<td>129 131 125 119</td>
<td></td>
</tr>
<tr>
<td></td>
<td>188 185 183 163 167 161 162 162</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>193 194 196 197 179 165 164 165</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>185 185 190 197 178 164 170 166</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>199 199 151 171 180 145 155 155</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>176 178 177 182 183 146 148 164</td>
<td>128 127 125 132</td>
<td>128 128 129 125</td>
<td></td>
</tr>
<tr>
<td></td>
<td>171 167 163 163 164 130 136 167</td>
<td>125 125 124 131</td>
<td>131 131 131 126</td>
<td></td>
</tr>
<tr>
<td></td>
<td>142 139 138 131 129 133 134 154</td>
<td>128 128 126 131</td>
<td>130 130 130 124</td>
<td></td>
</tr>
<tr>
<td></td>
<td>151 151 152 151 144 146 140 146</td>
<td>127 127 125 131</td>
<td>130 129 131 125</td>
<td></td>
</tr>
<tr>
<td></td>
<td>188 188 187 187 187 184 177 163</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>192 194 194 193 197 191 171</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>178 178 183 185 187 196 193 167</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>186 185 190 206 177 150 189 175</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>176 177 177 178 176 177 188 170</td>
<td>128 127 128 126</td>
<td>128 129 128 129</td>
<td></td>
</tr>
<tr>
<td></td>
<td>175 173 172 169 163 158 165 151</td>
<td>125 126 126 124</td>
<td>132 131 131 132</td>
<td></td>
</tr>
<tr>
<td></td>
<td>143 141 141 137 134 128 129</td>
<td>129 128 128 126</td>
<td>130 130 130 131</td>
<td></td>
</tr>
<tr>
<td></td>
<td>157 152 151 153 154 156 149 146</td>
<td>131 127 128 125</td>
<td>130 130 129 130</td>
<td></td>
</tr>
<tr>
<td></td>
<td>205 201 190 188 188 188 186 183</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>169 195 198 191 192 191 194 195</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>087 150 184 176 180 183 186 190</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>132 174 192 184 185 198 202 155</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>176 173 175 178 176 177 178 177</td>
<td>128 128 127 128</td>
<td>128 129 129 128</td>
<td></td>
</tr>
<tr>
<td></td>
<td>185 178 176 174 171 169 166 161</td>
<td>125 125 126 126</td>
<td>132 131 131 131</td>
<td></td>
</tr>
<tr>
<td></td>
<td>147 142 142 142 141 141 139 136</td>
<td>129 129 128 128</td>
<td>130 130 130 130</td>
<td></td>
</tr>
<tr>
<td></td>
<td>141 139 152 156 150 153 154 155</td>
<td>133 133 128 128</td>
<td>131 129 130 129</td>
<td></td>
</tr>
<tr>
<td></td>
<td>198 195 202 205 196 189 189 187</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>185 180 168 180 201 191 191 191</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>093 091 082 109 173 178 177 181</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>122 124 126 150 189 188 187 190</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>175 171 174 176 173 176 176 176</td>
<td>128 129 128 127</td>
<td>129 128 129 129</td>
<td></td>
</tr>
<tr>
<td></td>
<td>186 181 182 180 174 172 170 168</td>
<td>125 126 126 125</td>
<td>131 131 131 131</td>
<td></td>
</tr>
<tr>
<td></td>
<td>146 145 146 142 141 141 141 141</td>
<td>128 129 130 128</td>
<td>130 130 129 130</td>
<td></td>
</tr>
<tr>
<td></td>
<td>138 140 142 141 148 156 156 154</td>
<td>129 131 133 130</td>
<td>135 133 129 129</td>
<td></td>
</tr>
<tr>
<td></td>
<td>201 201 199 197 197 205 201 191</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>187 187 183 180 170 164 189 197</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>091 087 095 090 084 081 141 182</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>110 112 126 129 130 135 172 193</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>172 171 175 174 173 173 173 174</td>
<td>129 128 128 128</td>
<td>129 128 128 128</td>
<td></td>
</tr>
<tr>
<td></td>
<td>192 188 187 182 180 180 177 173</td>
<td>125 125 125 126</td>
<td>131 131 131 131</td>
<td></td>
</tr>
<tr>
<td></td>
<td>158 153 150 145 146 142 141</td>
<td>127 128 129 130</td>
<td>130 131 130 129</td>
<td></td>
</tr>
<tr>
<td></td>
<td>133 133 141 142 143 145 143 155</td>
<td>128 129 131 133</td>
<td>129 134 134 129</td>
<td></td>
</tr>
<tr>
<td></td>
<td>182 181 194 202 200 198 195 200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>200 190 182 184 184 179 176 164</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>183 168 113 084 092 091 087 079</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>183 174 131 111 120 128 130 132</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame Number</td>
<td>YYY</td>
<td>UUU</td>
<td>VVV</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>191</td>
<td>151 152 150 148 147 147 160 121 076 157 157 159 157 155 170 125 060 165 160 162 158 156 168 131 073 135 127 133 145 149 157 132 108 145 151 149 143 149 159 134 116 228 235 210 165 151 167 139 108 255 219 186 181 165 168 145 111 190 155 160 168 141 150 133 081</td>
<td>138 138 137 131 139 138 137 131 126 133 137 132 116 122 132 133</td>
<td>119 120 118 121 117 119 118 121 129 123 119 121 137 132 124 120</td>
<td></td>
</tr>
</tbody>
</table>

**Table 10:** YUV coefficients of selected block at different frames